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

CONTACT
Heather Mahalik

heather@smarterforensics.com
Twitter: @heathermahalik

11200 Rockville Pike, Suite 200

North Bethesda, MD 20852
301.654.SANS(7267)

DFIR RESOURCES
digital-forensics.sans.org
Twitter: @sansforensics

SANS EMAIL
GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

Author: Heather Mahalik hmahalik@gmail.com or heather@smarterforensics.com Twitter: @heathermahalik

Evidence" and Destr uctio

n

FOR585-4-H01-01

This page intentionally left blank.

This page intentionally left blank.

This page intentionally left blank.

- Why important?
  - Created regularly for most
  - May contain legacy information
  - May contain Apple Watch data
  - When you cannot get into the device but have access to a host computer or backup, this may be your best bet!
  - This will NOT get you into a locked iOS device, unless you crack the backup password and it's the same as the device passcode
  - Backups may exist on many computers!

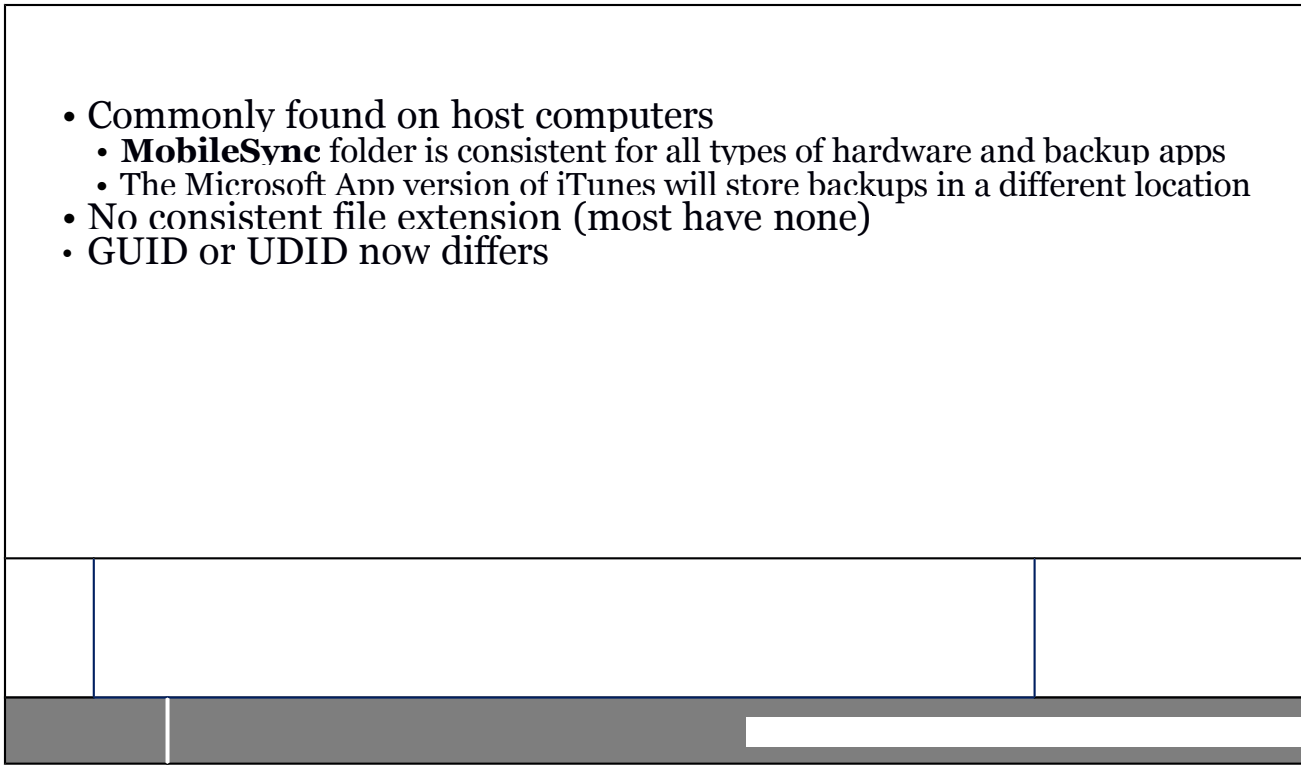
Smartphone backup files are an important part of any investigation. Users can wipe and delete data from their smartphones, but that doesn't delete or omit data that resides on a backup file on external media, the host computer, or in iCloud.

Backup files may contain data that the user believes no longer exists. When a user deletes data from his/her device, that information may exist temporarily in the backup file and may be recovered during a forensic examination. Depending on the smartphone, backup method, and storage space, multiple backup files may reside for a single device. Thus, as the data on the devices continue to change, each backup may contain unique data. Some iOS devices overwrite old backup files during the backup process, while other versions will create a new backup or snapshot with a date. Additionally, it is common for backups to exist both in iCloud and on the user's host computers. For example, I have backups in iCloud, on my Mac and on my PC. Chances are good that none of these backups are the same and some will contain data that will be missing from another.

Creating a backup file of a smartphone may be the only acquisition method, depending on the device.[1] This concept is discussed in the upcoming slides. Make sure the backup created for forensic examination is always set to encrypt the backup with a passcode that you remember and document in your case notes.

Reference:

[1] https://for585.com/backup (Describes the method for creating backup files of multiple smartphones. This may be useful to your investigation should backup file creation be the only method for pulling data from a device).



Both Windows and Macs can be used to back up iOS devices. Every computer, external storage, backup drive, etc., should be examined for legacy backup files. As previously stated, backup files may contain information that is no longer stored on the device that could be unique to the investigation. You may find multiple backups for the same device (depending on the iTunes version) or simply a backup that is updated when the user connects their device to sync.

On Windows computers, the following locations may store iOS backup files:
Windows Vista and later: C:\Users\<UserName>\AppData\Roaming\Apple Computer\MobileSync\Backup or C:\Users\<UserName>\Apple\MobileSync\Backup

The location of the backup depends on if the user backed up the device using natively installed iTunes or the Windows app version of the iTunes. The second location is where the Windows (Microsoft) app version of iTunes is stored, while the first is the original location of iOS backups.

Windows XP: C:\Documents and Settings\<User Name>\Application Data\Apple Computer\MobileSync\Backup

On a Mac, the following location stores iOS backup files. (NOTE: If you don't see the Library folder listed, you must access the user folder by following the path Go > Go to Folder > Library, and access is provided.)

Mac OS: MacHD/Users/<UserName>/Library/Application Support/MobileSync/Backup

Searching for iOS backup files is not simple because the filenames do not have a consistent extension. It is best to search for the folders that commonly contain backup files. In the past for older versions of iTunes, a general search could be conducted for *.md*, which should find the common file extensions (.mddata, .mdinfo, and so on). Since file extensions are no longer applied to iTunes backups, it's best to search for info.plist or manifest.plist, as we know these files exist in all iTunes backups. If the user has not moved the backup files, you could simply search for **MobileSync** and find the backup directory on both PC and Mac.

The backup files used to be stored in a folder named with a 40-digital alphanumeric GUID.[1] Since September 2018-present with the release of Apple Watch Series 4, iPhone XR, iPhone XS, and iPhone XS Max, the UDID format changed. All devices released after these use the new format: 8 characters of ChipID padded with zeros on the left, without leading zeros. e.g., 8020 for iPhone XS/XR Seriex and 8030 for iPhone 11 Series. A hyphen, followed by 16 characters of ECID in hexadecimal padded with zeros on left. The general format will be:[0padding]CHIP-[0padding]ECID[2].

As new hardware releases from Apple be ready for backup naming conventions and formats to change.

References:

- https://for585.com/itunesbackups
- https://for585.com/udid

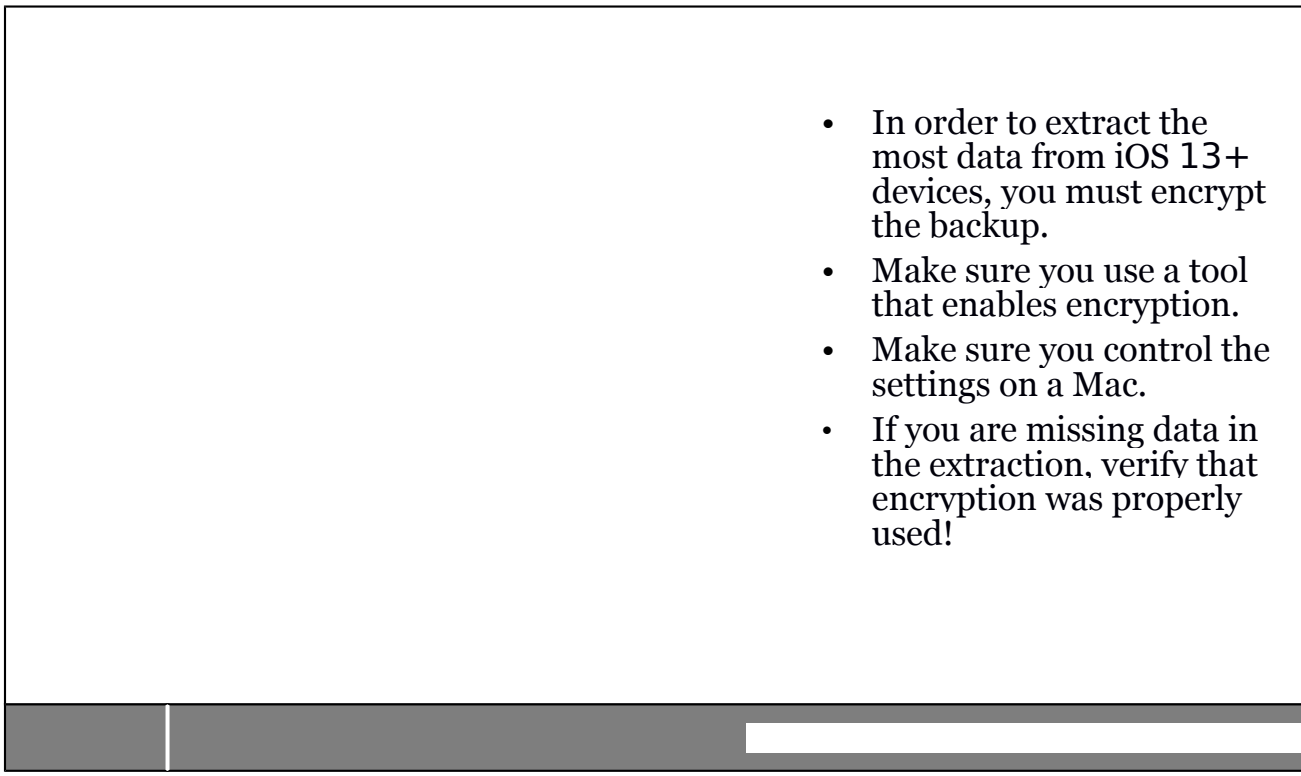
- Does it matter where the backup came from?
- Will your tools support both?
- What is required for access?

Whether a backup is stored in iCloud or on the local machine may determine your amount of access. For example, if you are allowed to examine data at rest, you should be able to examine the iTunes or Finder backups. However, if you don't have access to the user's cloud information, whether it be due to lack of password and account information or legal reasons, you are going to be restricted from what you can examine. The backup files themselves should not matter, depending on the tools that you have. Most tools should support ingesting and parsing of both iTunes and iCloud backups. Open-source tools seem to struggle with encryption, and this may be something you stumble upon. If you want to see this in action, attempt to load Lab 1.2 into iBackupBot in the VM. You will get an error about the backup being encrypted. The issue is not the cracking the iTunes passcode, but instead leveraging it to decrypt the backup. These methods will be covered in the upcoming slides.

The most important thing to realize is that all backups likely contain unique data. I back up to my Mac and PC. This is not a frequent occurrence and to be honest I am not sure of the frequency. However, I back up to iCloud every single night. See the difference?

The options for iCloud are presented to the user when they log into iCloud or via the iOS device interface. Features like "Find My" are also available and should be considered later in this section in the spyware portion of this course. Most are aware that iCloud enables the user to

place the iOS device into Lost Mode; or Erase the iOS device.



As we covered in section 3, iOS 13, iOS 14, and iOS 15 require all extractions other than Full File System to be encrypted extraction. This includes iTunes backups on a PC and backups via the Finder on Mac. If you parse your extraction and find that you are missing data, chances are that you didn't encrypt the extraction.
Remember, in order to extract native Calls, Safari searches, Apple Maps, Health, Wallet and more, you must encrypt the backup or logical/advanced logical extraction.

When creating a backup with a Mac, use caution when encrypting the backup. If you enable “Remember this password in my keychain” you will not be able to completely parse the data as the tools will not be prompted for the encryption passcode. Why? Because the keychain keeps the passcode and the manifest.plist isn’t tracking the encryption status. This causes major issues, which is why you should NOT check that box.

- Databases and plists

- Info.plist
- Manifest.plist
- Status.plist
- Manifest.mbdb (up to iOS 9)
- Manifest.db (iOS 10+)

The iOS backup file should provide full access to the file system (backup) of the Data Partition. The volume of the file system reflects the name of the device. For example, if the iPad was assigned the name “Heather's iPad”, the file system would be listed as "Heather's iPad" with the files and folders listed as directories within the volume.

The file system contains the database files that hold a vast amount of information relating to the data stored within the backup file. Database files for maps, location information, third-party applications, and standard communications are often not fully parsed by forensic tools, such as AXIOM, Inspector, Physical Analyzer, and Oxygen. Remember, the database files can be viewed in an SQLite Browser tool but must be examined in the Hex view to carve deleted data.

The other component of the backup includes plist files that are created with a backup. The Info.plist contains device information (equipment identifiers, applications installed, backup information, and so on). The Manifest.plist lists if the backup is encrypted. This comes into use and is required should the backup file need to be accessed forensically if it is locked. The Manifest.mbdb contains a listing of data stored in the backup. Even if the backup is encrypted, this data can be parsed for more information. Scripts for parsing the Manifest.mbdb can be found at https://github.com/halpomeranz/mbdbls. With iOS 10+, the manifest.mbdb was replaced with the manifest.db. You may find that this file can be encrypted!

For more information on how SQLite databases are formatted, refer to https://for585.com/sqlite.

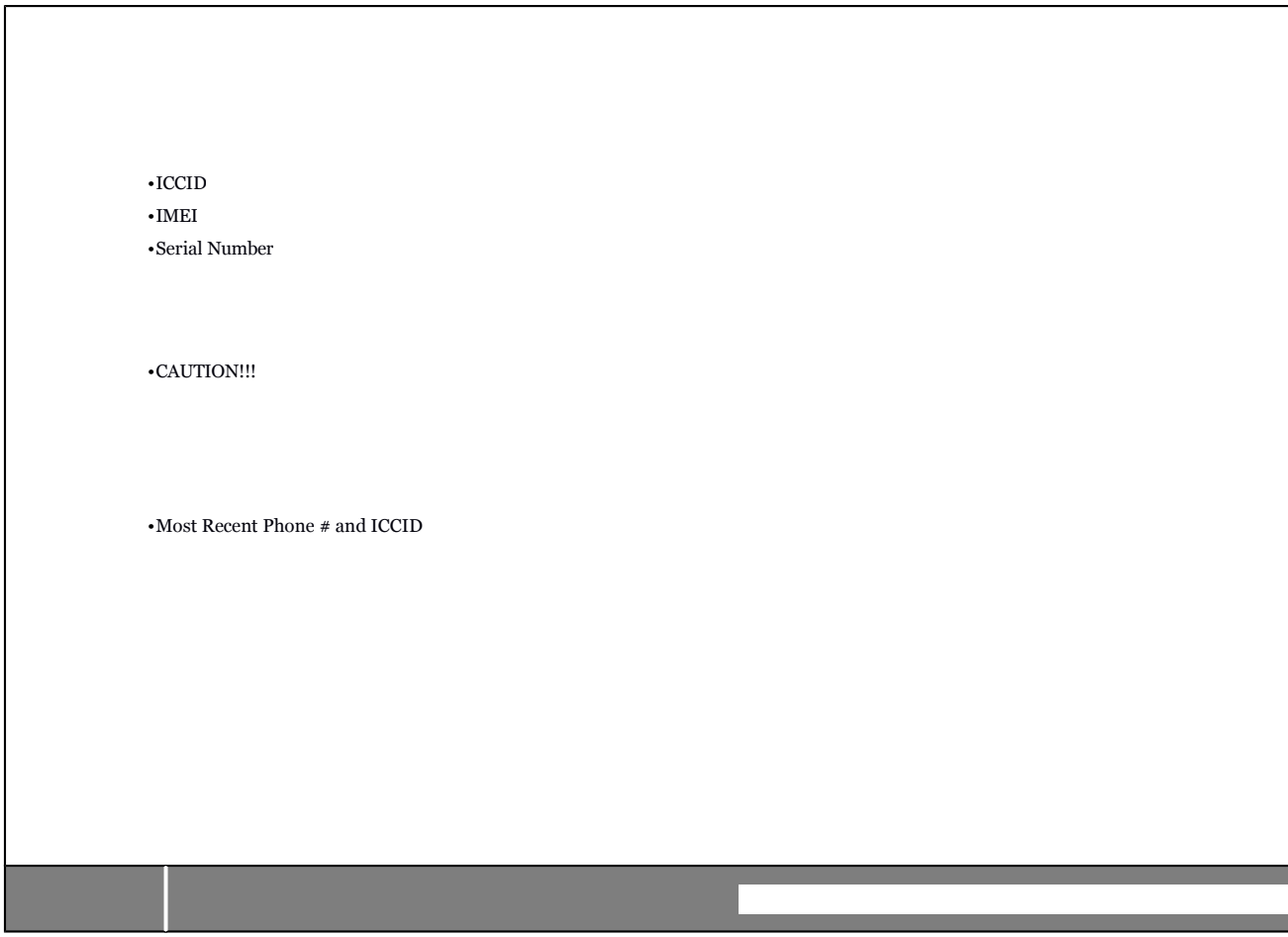
- With iTunes
- With the Microsoft store iTunes app
- With a commercial tool
- With a free backup tool

Caution – This timestamp should not be trusted!

Before we start examining the primary plists contained in iOS backups, we have to consider how the backup was created. For this portion of the course, we are focusing on user-created iOS backups. This means the user launched iTunes and backed up their data to a PC or Mac. However, the type of iTunes used matters here. By type, I mean was iTunes natively installed from the Apple store or a legitimate installer or was

the user pushed to the Microsoft store to install iTunes as an app? Most people wouldn't even know, but it matters. Where the data is stored will be different and some timestamps become untrustworthy. If you aren't sure what you have installed, create a backup, and follow the path.

As we previously discussed, the Microsoft app of iTunes will store the data in C:\Users\<User>\Apple\MobileSync\Backup and Apple iTunes (native install) will store the data as discussed previously in the ~\AppData\Roaming\Apple Computer\MobileSync\Backup directory. The reason this is so important here is the date modified of the backup that was created by the Microsoft app. I installed the native iTunes app and the date of the backup shown in the bottom graphic had a new modified time because iTunes must interact with that directory. DO NOT trust these dates as they have zero correlation as to when the backup was created. You need more concrete evidence, and we are going to investigate that now.



The Info.plist file can be viewed with a plist editor, Physical Analyzer, AXIOM, Inspector, XRY, or Oxygen Forensics. The Info.plist file is found in the backup directory and should always be examined to verify the data being parsed by the tool. The Info.plist contains the equipment identifiers, last backup date, phone number (if relevant), device name, device type, sync settings, build version, Backup folder name, applications, and so on.[1]

Use caution here, as the last backup date in the Info.plist may be the date the tool (such as Physical Analyzer) acquired the device! In this example, that is exactly what happened. The real backup date is contained in the status.plist for this device. If you were provided a backup file to examine and didn't know anything about the device, the items highlighted in the slide will assist you in your initial analysis.

Two examples of Info.plist files are provided below in both the xml and list views.

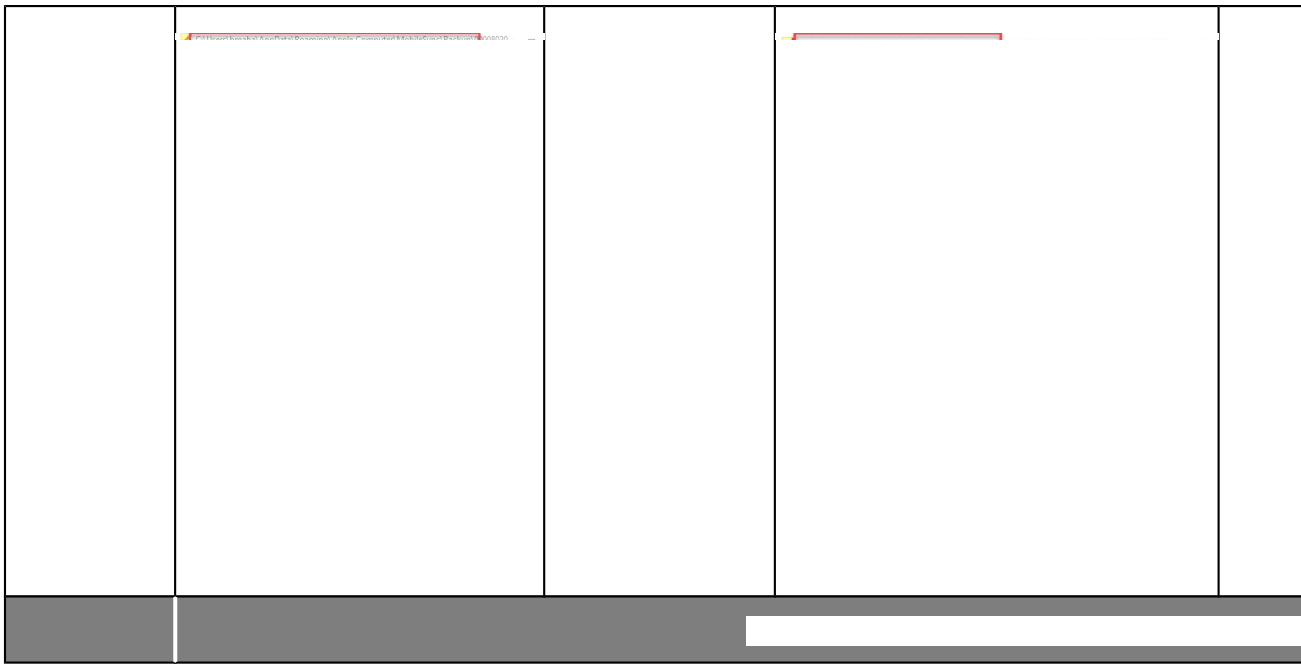
Reference:

[1] https://for585.com/itunes (iTunes Backup description on the iPhone Wiki).

- Only tracks successful backups

The status.plist is one of the files that tells the truth about the backup completion time if Apple iTunes was used. If the Microsoft app was used, the info.plist will store the most accurate time of the backup and the status.plist may not be updated. When the status.plist is available, the file will include the completion time of the last successful backup. If the backup fails, this file will not be updated. This is the first plist that is created and written to during an iOS backup extraction.

The manitfest.plist tracks encryption settings and applications installed on the device. The datetime stamp included in this file tracks the last time encryption settings changed. If the backup were encrypted and the user decided to create a new unencrypted backup, that would be the date reflected here. This would also be the case if the user decided to encrypt a backup that was previously unencrypted. Forensic tools may also change this file if the tool provides the option to encrypt the backup during the forensic extraction.



Here we are looking at the different installations of iTunes and how it controls the timestamps in the three plist files previously discussed. On the left, we are looking at the full install of iTunes on a Mac or a PC. Here, this is a PC example. The path is the key in knowing how the backup was created.
C:\Users\<UserName>\AppData\Roaming\Apple Computer\MobileSync\Backup is the storage location for Apple iTunes and its backups. The timestamps included here are:

- Manifest.plist – the last time the encryption status for the backup of that device changed
- Info.plist – the date the backup was last completed – NOTE – will be timestamped by forensic tools! Also, the date is always a few seconds after the backup was really completed.
- Status.plist – the true time the last successful backup completed.

For the right side, we are looking at the iTunes version of the Microsoft App. The path is C:\Users\<UserName>\Apple\MobileSync\Backup. Here, the only trusted timestamp is the info.plist which is a few seconds after the backup completed. The other timestamps represent the last time the full install version of iTunes was used on a different system. These should not be relied upon. For more information on this, refer to the blog posts by Heather Mahalik on smarterforensics.com and https://www.cellebrite.com/en/blog/.

Apple iTunes.

Microsoft App iTunes

iOS 10 introduced a new format for the Manifest as well as encrypting the database if the user selected to encrypt the backup. In the top screenshot, we simply opened the Manifest.db and found that the data was not encrypted. The data within this file is the same as it was with older iOS versions contained in backups. The second screenshot is an encrypted backup, which contains an encrypted Manifest.db. This file can be decrypted with the proper password or with your forensic tools. Physical Analyzer automatically decrypted this database when the password for the backup was entered during the ingestion phase. For more information on what has changed in iOS 10 – iOS 15 backup files and the support provided by your commercial tools, refer to Heather Mahalik's blog at https://smarterforensics.com/blog/.

Pre- iOS 10 backup Manifest.mbdb structure is shown below.

iOS backup files created prior to iOS 10 are shown in the screenshot on the left. These backups are comprised of files named for their name, path, and domain, as well as .plist files and the Manifest.mbdb.[1] The Manifest.mbdb contains a list of files included in the backup. This file is not encrypted when the backup file is encrypted by the user. Thus, you can always examine this file for forensic artifacts relevant to your case. It may be a simple file listing, but it could point out an artifact you were looking for. In the example on the right, Hal Pomeranz's script was used to parse the Manifest.mbdb. Let's say this backup was encrypted and we could not access any data because the password couldn't be cracked. When we parse this backup, we learn a few things about the device. We can see that apps

are used on the device, such as Facebook, PalTalk, Firefox, and more. More importantly, if we kept reading the Manifest.mbdb results, we would be able to see that Cydia is set to push to the device. This device is jailbroken! Use the UDID of the backup folder along with the Info.plist file to see if you can locate the device and obtain a file system or full physical image of the device, regardless of the passcode.

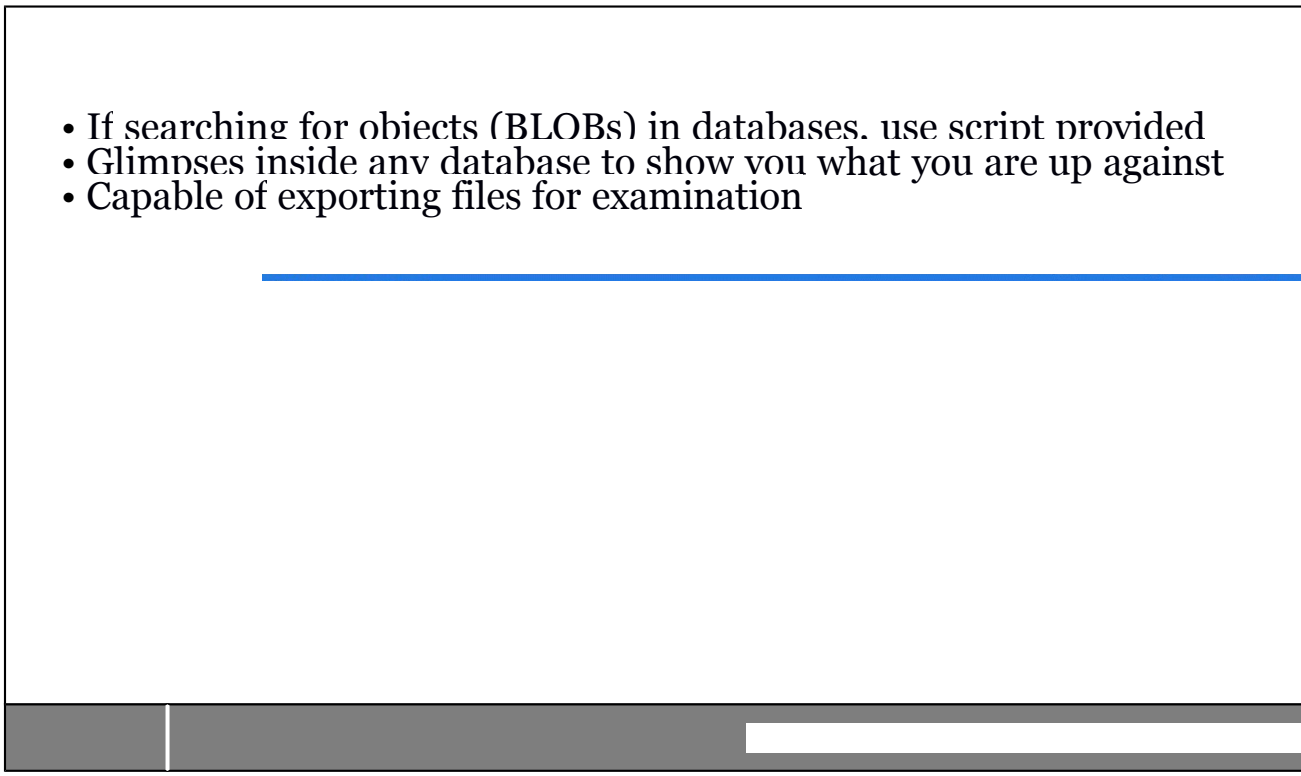
Reference:

- https://for585.com/iback (iTunes Backup description on the iPhone Wiki).

When iOS versions update, many things may change. Be it a new file being leveraged, or data being formatted in a different way, it complicates mobile forensics. It's up to us to identify these changes, adapt our methods, and alert the vendors of the tools if they aren't already parsing it. Think about this – if you found it, so did someone else. However, they may not know it's parsing incorrectly. Please report these issues. Even if it's to our Google group to make others aware.

Since data changes are a reality, you will see this in Lab 4.2 later in this section. You will be presented with iOS 15 data and a query that worked for iOS 13. The point – forcing you to modify what is available and make

it work!



We love when a student produces a solution to fill the gaps commercial tools present. Jon Baumann did just that—he built a script (SQLite Miner Master) that parses databases that contain BLOBs and identifies the content. This script can be found at https://github.com/threeplanetssoftware/sqlite_miner. It is available in your VM, but make sure you check for updates. This script may prove to be helpful in Lab 4.1.

- And a lot of people keep older backups

- Apple Maps – differs drastically across versions
- Dynamic-Text.dat

This page intentionally left blank.

Lab 4.1

Advanced iOS Backup File Analysis

This page intentionally left blank.

This page intentionally left blank.

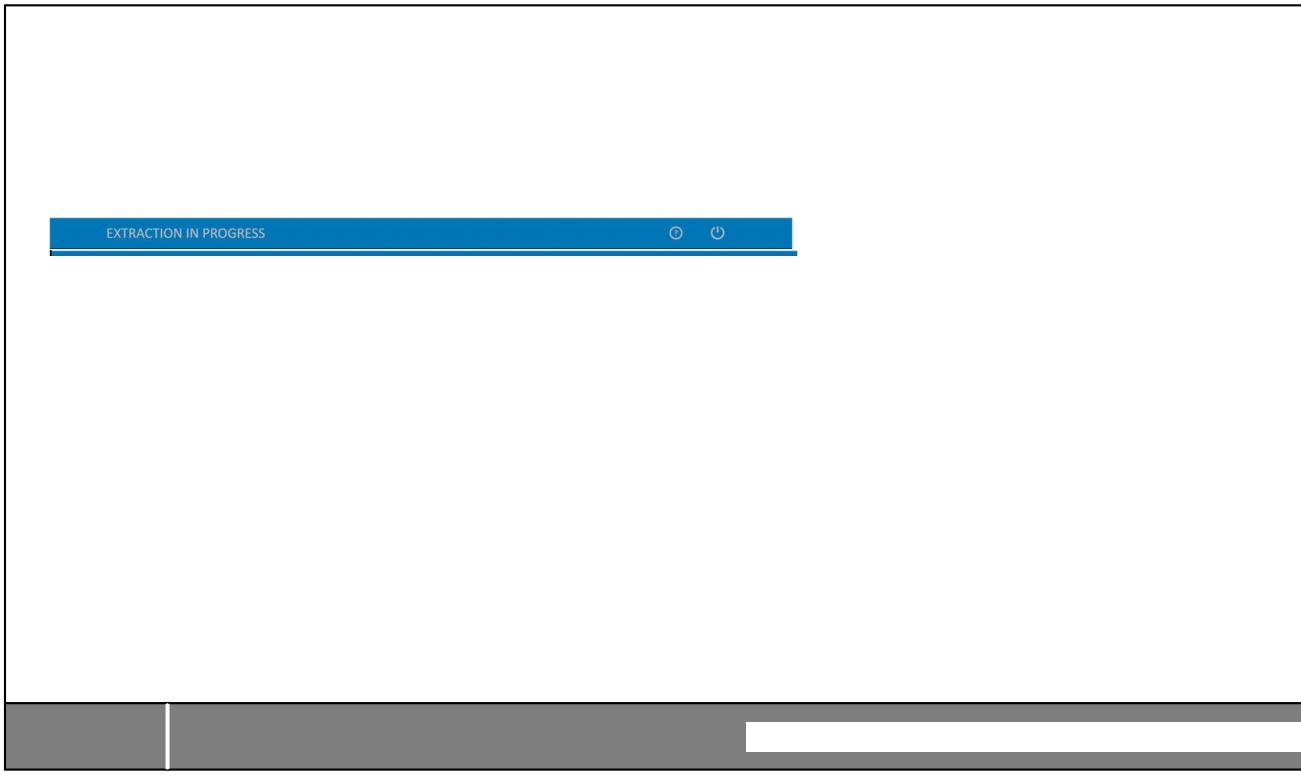
When an iOS backup file is created, the state of the keychain depends on the user/examiner. The keychain for the iOS device is designed to protect the key files. When a backup is initiated, the keychain can be captured. The “level of security” of the keychain then falls on the user/examiner and their elected settings. If the user elects to “encrypt” the backup file using iTunes, they must set a password to access the encrypted backup file. This password is then backed up with the backup file and is available for password cracking, as shown in previous slides. This method, even though the user believes they are encrypting their data, is less secure because the keychain file containing the password is backed up and accessible.

For iOS 13+ devices, if the backup is not encrypted, the backup will NOT contain the keychain, health, native calls, safari history, and possibly Apple Maps. Backup encryption is so important! We need it to gain a full picture of the user activity.

The other option for security occurs when the user decides to create a backup without utilizing the encryption offered by Apple. When this occurs, Apple intervenes and protects the user by not allowing the backup file to capture the keychain file. This means that the backup itself is parsable by forensic tools without prompting for a password; however, the backup does not contain valid user information, as mentioned above.

In summary, the backup file is more secure if the user can set a password

during the “encrypt backup file” selection that cannot be cracked by a forensic tool. However, most users select simple passcodes that are easily cracked. If a simple passcode is used, the method of not encrypting the backup file at all provides the most secure solution. iTunes now warns users of this vulnerability in their sensitive data.



When you are in the process of acquiring an iOS device, your tool may give you a clue as to what you will be dealing with when parsing the data. Here we can see that UFED is prompting us to “Encrypt backup” for Beth’s iPhone. Your tool may only prompt you **if** iTunes has not already encrypted the backup. If you do not see this prompt in your tool (assuming you are using a tool like UFED), you will see a screen that states that iTunes backup encryption is already set on the device. Get ready to crack that password if this is the case.

To use Elcomsoft Phone Breaker (EPB), select "Choose backup."
Navigate to the backup file you want to crack. The option includes:

- **Apple**: iOS device backup (backup file created with iTunes or a tool utilizing iTunes).

Once your selection is made, navigate to the backup file.

To manually determine if an iOS backup file is encrypted, simply go to the Manifest.plist and read the status flag:

0 = Not Encrypted
1 = Encrypted

If the Boolean values are not listed, the words “true” and “false” will be leveraged in the Manifest.plist.

This screenshot shows Elcomsoft Phone Password Breaker recognizing that the file is not encrypted. This normally occurs for non-encrypted images or when the backup is corrupt. If your tool prompts you to enter a password and you get this error when attempting to crack the manifest.plist, manually verify for encryption by opening the plist and looking to see if encrypted is set to true.

For encrypted backup files, the user can run either Dictionary or Brute-Force attacks. In this example, we attempt to crack an encrypted iOS backup file.

The Brute-Force attack settings are shown in this slide. The examiner can determine the password length and the character sets and add dictionary or

custom files. Dictionary and custom files found on other devices associated with the investigation may prove to be useful. We recommend starting with 4 as a numeric PIN and then trying 6 as a numeric PIN. This is the fastest to crack.

For unsuccessful attempts at password cracking, the examiner can elect to recover other passwords, which allows her to select another cracking method. The screenshot with the passcode shows what the results look like when encryption is successfully cracked. Notice that the first two letters are shown, and the rest are not provided in the demo mode. A dictionary file was used to crack this passcode.

iOS 11 - 15 “Reset All Settings” can be used to create a new encrypted backup for which you choose the password.

- Could be used if no other options are available
- The device must be unlocked
- Changes will be made to the dates of underlying default databases
- Hash values of many databases remain the same
- User data will be preserved

With iOS 11, Apple backslid somewhat with the security of previously encrypted backups. They introduced the ability to remove the previously encrypted backup restrictions from the device itself through resetting the settings on the device. This feature also works in iOS 12 – iOS 15. If you can’t restore an encrypted backup because you don’t have the password, and new backups or forensic extractions you created of the same phone also result in encrypted backups for which you don’t have the password; iOS 11- iOS 13 now offers us the possibility of making a new encrypted backup of the device and resetting the password to one we know. [123]

Here’s what to do to reset all settings:

- On your iOS device, go to Settings > General > Reset.
- Tap Reset All Settings and enter your iOS passcode, if prompted.
- Follow the steps to reset your settings. NOTE – you will need the device passcode on newer iOS versions.

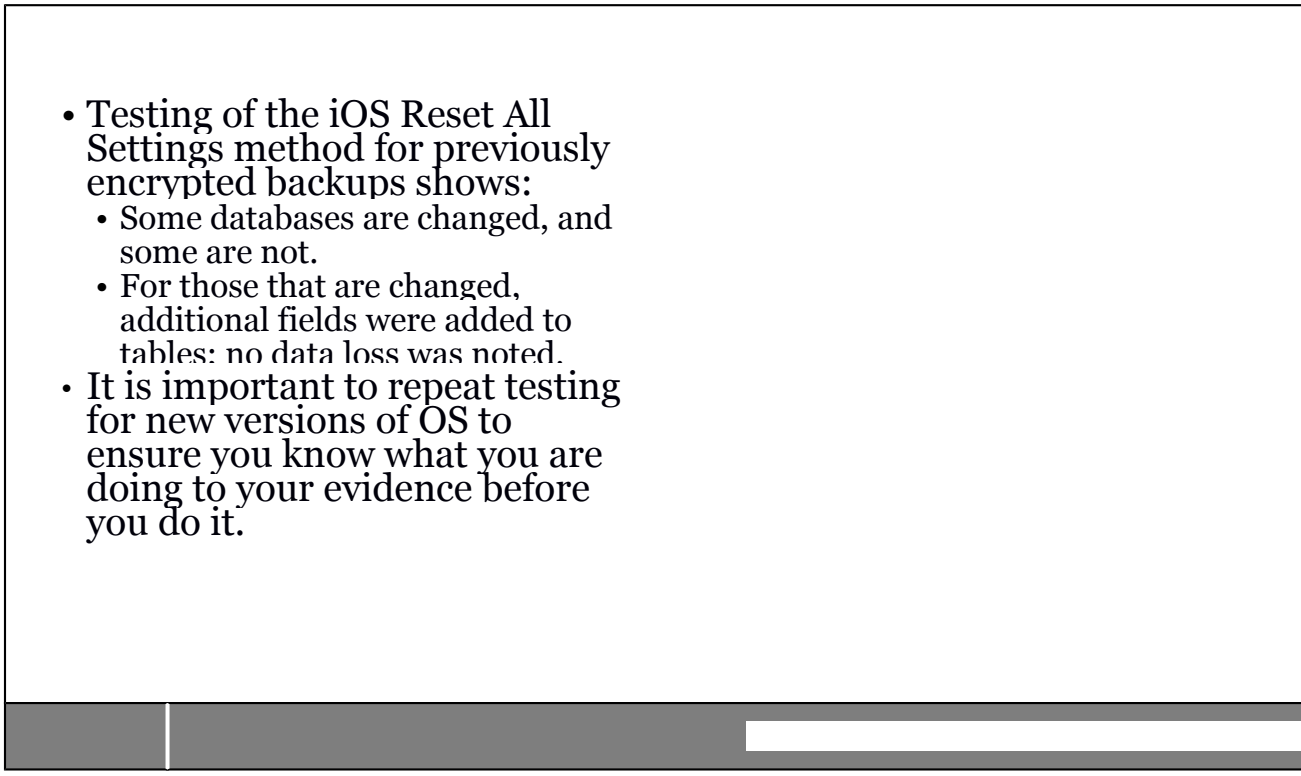
This method is NOT getting beyond a phone lock, simply an iTunes encryption passcode.

Resetting your settings won’t affect your user data or other passwords, but it will reset settings like display brightness, home screen layout, wallet,

keychain, and wallpaper. When we connect a reset phone to our forensic tools, we will find that the backup encryption password has been removed.

References:

- https://for585.com/c1lqm (Gillware Digital Forensics blog: "Forensic Case Files – A New Solution for Previously Encrypted iOS Backups")
- https://for585.com/3gend (ElcomSoft blog: "iOS11 Makes Logical Acquisition Trivial")
- https://for585.com/gqcn7 (Apple's help page for previously encrypted backups)



It is important to repeat testing for new versions of OS to ensure you know what you are doing to your evidence before you do it. Testing of the hash values of database files (shown in part above) before and after the Reset All Settings process shows that many of the hash values remain the same, indicating that the data within them is not changed during the reset process. The file modified/created/and last accessed dates of the database file may still be changed, though.

Not all of the databases are unaffected, though, and this is important to keep in mind. It is extremely important to test this process using test phones with the particular version of iOS from your case if you decide that using it is appropriate in your case. In the instances where hash values didn't match, additional fields were found to be added to tables within the databases and no data loss was noted.

If you are dealing with an iOS device as evidence from a civil or criminal case with a previously encrypted backup, obviously you will need to consider the ramifications of making changes to the original device and weigh that into the equation. Be sure you have the legal authority to make changes to the device and remember that careful documentation of the process is essential.

Upon successful reset, the encrypted backup password will be removed, allowing you to create an encrypted backup or extraction that is encrypted with your password of choice.

This page intentionally left blank.

There are so many options for extracting cloud data of iOS devices. We are going to discuss several options in the upcoming slides. For iOS 11.2+, icloud backups using 2FA require a new type of authentication token, similar to the one that is needed to get the iCloud Keychain and synced messages. The problem is that generating such a token requires some device-specific data to be sent to Apple servers, and it seems that this kind of data can be generated on iOS devices only. Elcomsoft has worked hard on this issue to reverse engineer the kernel code (and related protocols) and can successfully extract updated iOS versions of cloud data. The screenshots below show a successful iCloud extraction from iOS 15.

Elcomsoft Phone Breaker (EPB) can be used to download backup files stored on iCloud as well as download Synced data collected by Apple. A full, licensed version of EPB is required to access these backup files. The next few slides walk you through how to access data stored in iCloud. CAUTION: Make sure you have legal authority or consent to use methods such as this for accessing iCloud data. Another tool worth looking into is inflatabledonkey, which can be used to extract iOS backup files from iCloud.[2]

References:

[1] https://for585.com/donkey (GitHub for inflatabledonkey)

The user's Apple ID and password/token are required to access the backup files stored in iCloud. Another option is to leverage tokens for extraction. Without proper credentials or tokens, the backups cannot be retrieved. The third option is Device, which seems to only work with older iPhones and iPads.

WARNING: Two-factor authentication (2FA) will alert a user, and you will have to enter the verification code if the user has enabled these settings. This is not just true for iCloud, but all cloud artifacts. In this slide we see Elcomsoft Phone Breaker. All tools must support 2FA if they want to be current and pull cloud artifacts.

All backup files stored in iCloud associated to the entered Apple ID are displayed. It is wise to capture all instances shown to ensure that nothing is overlooked.

The examiner has the option to select what is included in the backup file. Downloading data from iCloud is a slow process. The download may fail if the forensic computer enters sleep mode or if there is an Internet connection issue. The download is more efficient if less data is selected for download. However, it is wise to capture everything to ensure that all data is available for analysis. The downloading begins once Done is selected.

When downloading the backups, we recommend downloading in chunks due to failures when attempting to download a large or multiple large backups. Make sure you get "Errors – 0" when complete. If you get any errors at all, it's best to download the entire dataset, in chunks, a second time.

Elcomsoft has the capability of collecting data that is synced by Apple and data that is backed up by the user. In some situations, this data may be similar. However, look at the differences in what is collected. On the left, we can see some items that have an orange (or lighter shaded) checkbox. The lighter shading means the phone passcode must be entered to extract this information. That is the only thing protecting this data. When possible, collect both the Synced data from iCloud and the iCloud backup files.

Above, we see the options, as previously discussed, when the examiner elects to download the iCloud backup.

As discussed in the previous slide, some data may only exist in Synced data from iCloud if the user elects to not have the data backed up. Here we see Apple Maps, which has notoriously been a problem for extracting and parsing. This data exists in most cloud containers, but always in Synced data from iCloud. On the right, we see Safari tabs. Crazy enough, Safari tabs are not parsed by some commercial tools. Here we can see the data parsed nicely in Elcomsoft Phone Viewer.

Safari tabs are shown above.

Explore Keychain

There are additional features built into the cloud extraction tools that may assist in your investigation. Keep in mind, you need authority to access these features. The example above is the **Explore Keychain** feature from Elcomsoft Phone Breaker. Here, we are loading a backup of interest. If the backup is encrypted (which is required for the keychain to be present in the backup), the backup password must be entered. Once entered, the keychain is decoded and displayed, as shown on the right.

This page intentionally left blank.

Lab 4.2

iOS 15 Database Analysis

This page intentionally left blank.

This page intentionally left blank.

- More devices =

more mobile malware

- The upward trend in detected mobile malware threats has continued steadily as smartphone use has increased
- Examiners should be prepared to deal with mobile malware in their cases
  - Data compromise and exfiltration
  - Malware as a defense
  - Stalking Investigations

With the proliferation of mobile devices, mobile malware has become an increasing risk. There is an ongoing trend of users moving away from the traditional PC market of notebooks and desk-based computers toward mobile devices, a trend that Gartner has described as reflecting a long-term change in user behavior since at least 2013.[1] As the use of traditional desktop and notebook computers declines and the use of mobile devices increases, mobile devices increasingly become the logical target of malware.

Kaspersky has been providing a very detailed report of the evolution of mobile malware over the past several years, and the latest statistical analysis represented that malware was increasingly targeting users' personal data (stalkerware), and Trojans notably began popping up in legitimate App stores. One such example of stalkerware is the commercially available application, FinSpy, which has incorporated capabilities to intercept messages from applications like Threema and Signal by leveraging the DirtyCow exploit (Android, CVE-2016-5195).[2]

References:

- https://www.gartner.com/reviews/market/mobile-threat-defense-solutions(Gartner threat report)
- https://for585.com/kasperskyreport

- Android
  - Most affected platform
    - Are almost 50% more likely to be affected than iOS
    - Android OS has greater market share than others
    - Consider how devices receive updates
    - Consider application lifecycle and longevity compared to iOS
- Other operating systems are not immune
  - iOS attacks are often sophisticated and may be carried out by nation states
    - 2017: Mobile Safari JavaScript Pop-Up Scareware

According to Juniper, until around 2010, the most targeted platforms for mobile malware were Nokia's Symbian operating system and Oracle's Java platform. Since that time, a shift occurred toward malware that targeted Google's Android operating system, a trend that followed the mobile market. Currently, the majority of detected malware threats targeted Android devices.[1]

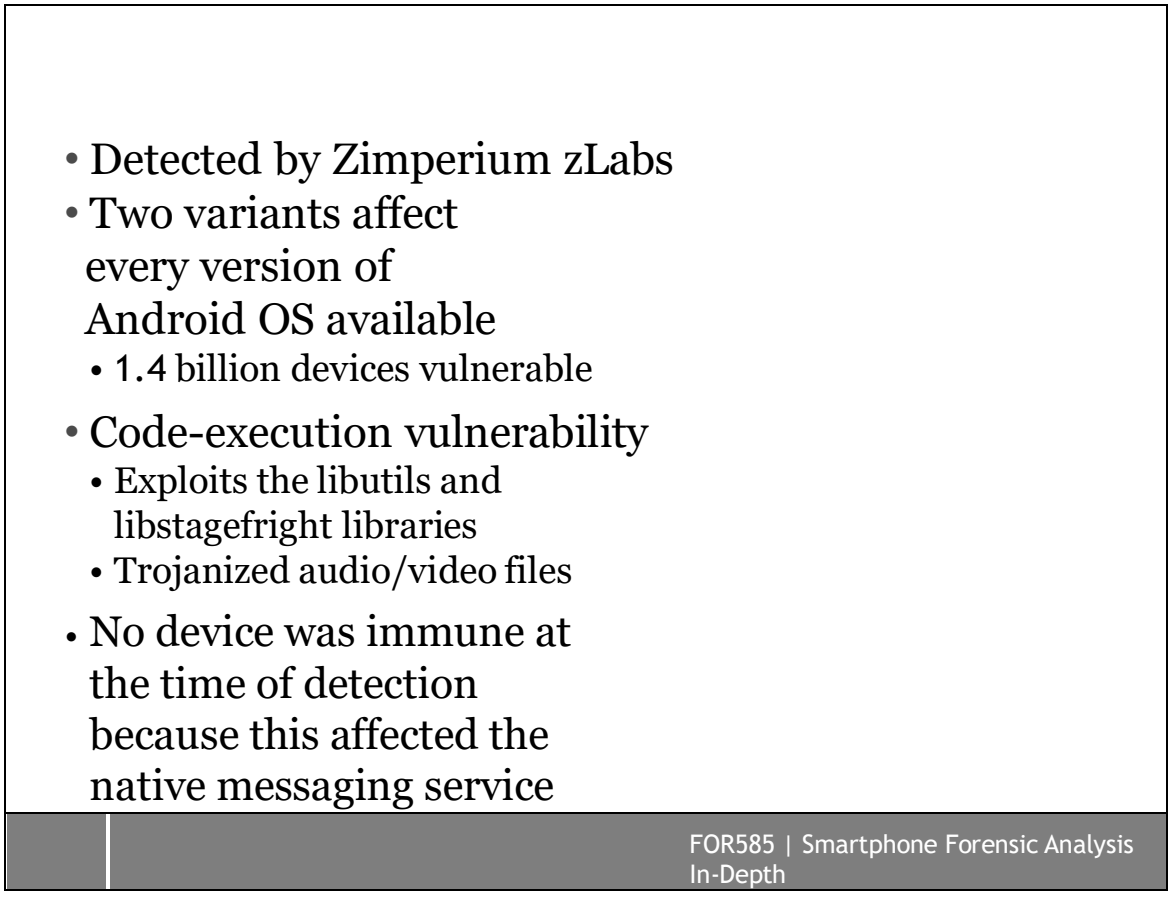
Exploits for iOS devices have been demonstrated, but fewer are often seen in the wild prior to security patches being released. For example, in 2014, WireLurker proved to be an active and ongoing threat to iOS devices.
WireLurker loads itself onto iPhones and iPads when the devices are connected via USB to a Mac computer onto which an infected OS X app has already been downloaded.[2] Threats to iOS devices have been identified every year, including XCode Ghost, AceDeceiver, Pegasus Spyware, and exploits to Java on Mobile Safari. Some of these threats have been distributed through the iOS App Store before being discovered and mitigated and rely on misuse of developer and/or enterprise certificates and also, take advantage of unpatched firmware.

The Android platform still presents a much easier target for malware developers based on many of the criteria that were discussed during the Android module. Unlike iOS, it is more common for Android devices to become stagnant and no longer receive software updates (security patches) due to Manufacturer and Carrier limitations, hardware support, and other factors.

While the platforms listed below are no longer actively supported, it is important to note that NO mobile operating system is immune to attacks, as many earlier malware variants attacked the Windows Mobile, JAVAME, BlackBerry, Symbian, and Nokia firmware as well.

References:

- https://for585.com/u2sn- (Juniper report)
- https://for585.com/hfj59 (Digital Trends article on WireLurker)



In April 2015, the Android OS Stagefright vulnerability was detected by Joshua Drake of Zimperium zLabs. Two variants of the malware exist, dubbed Stagefright and Stagefright 2.0. Together they are said to affect any device running a version of Android from 1.0 to the present. This means that nearly 1.4 billion Android devices are vulnerable. The exploit gives an attacker Remote Code Execution (RCE) ability, wherein once he has gained access, he can execute commands and kick off processes without any permissions from the user.

The original malicious code required an attacker to send an MMS message to an unsuspecting user containing a malicious .mp3 or .mp4 file, thus knowing his phone number to successfully deliver the exploit. A new variant, Stagefright 2.0, infects phones if the user simply visits a website that is hosting a malicious multimedia file.[1]

How It Works

The Stagefright vulnerability exploits the libStagefright library, which allows the Android OS to pre-process video files prior to a user opening the file. This library is utilized by

both default Android applications, Messaging and Hangouts, which work to preview the message content as soon as it is delivered to the device. This speeds up processing time should the user want to open and subsequently download the message content. Because the OS has already previewed the file prior to a user opening it, the device can be infected, even if the message is deleted prior to opening.[1]

Android devices rely on patches being pushed from the cellular carrier or handset manufacturer, which makes mitigating the risk of vulnerable devices a more daunting task than it is for Apple.

Protecting Your Device
The vulnerability has been patched in all new Android OS versions, but prior to the patch being released, disabling the Auto-Retrieve feature in the default messaging applications was used to provide a temporary mitigation against the Stagefright vulnerability. Depending on the installed version of Android and the device manufacturer, your messaging application may be named something like "Messages," "Messenger," or another similar name.[2]

References:

- https://for585.com/4agcf (Android Central article on Stagefright)
- https://for585.com/0cm5y (Digital Trends article on Stagefright)

- FluBot malware masquerading as "Voicemail"
- Delivered to users through SMS smishing attacks
- Requires file download to listen to voicemail
- Leverages Android Accessibility to disable Play Protect
- Establishes a C2 to exfil contact data
- Attempts to steal

FluBot
FluBot infiltrates devices using the native SMS service and smishing attacks designed to trick a user into installing an application which will facilitate the ability to "listen to a voicemail" message. The Android Accessibility service is leveraged to disable Play Protect, a Google service which prevents certain applications from being installed on Android devices.

Attackers use overlay attacks, html pages which mimic a users' legitimate banking or payment apps, to trick users into supplying sensitive details such as username and passwords. This sensitive data is then exfilled to the newly established C2. While the exact attacker is unknown, it has been loosely attributed to a Russian attack group based on analysis of the C2.[1]

Additional Examples:

RANA Family
The Malware family known as RANA, which is attributed to Threat Actor, APT-39, has continuously evolved their malware to exploit devices of Android users. The attack begins with infecting the devices using Optimizer.apk, when upon setting up C2 communications sends AES-encrypted sensitive system and device data back to the server.
Capabilities include the ability to record audio and take pictures as well as forcing the device to answer incoming calls received from certain pre-defined telephone numbers.[2]

They have, again, expanded their capabilities by focusing on messaging data from popular communication applications like Instagram app, Skype, Telegram, Viber, and WhatsApp, by exploiting Android's Accessibility Service to retrieve sensitive information to exfil from the devices. [3]

RedDrop
RedDrop malware was discovered in March 2018 by security research firm Wandera's machine intelligence engine "MI:RIAM." It is a zero-day threat that was unknown within the mobile security community prior to its discovery. RedDrop is included within at least 53 applications and is distributed by a network of 4,000+ domains registered to the same underground group.

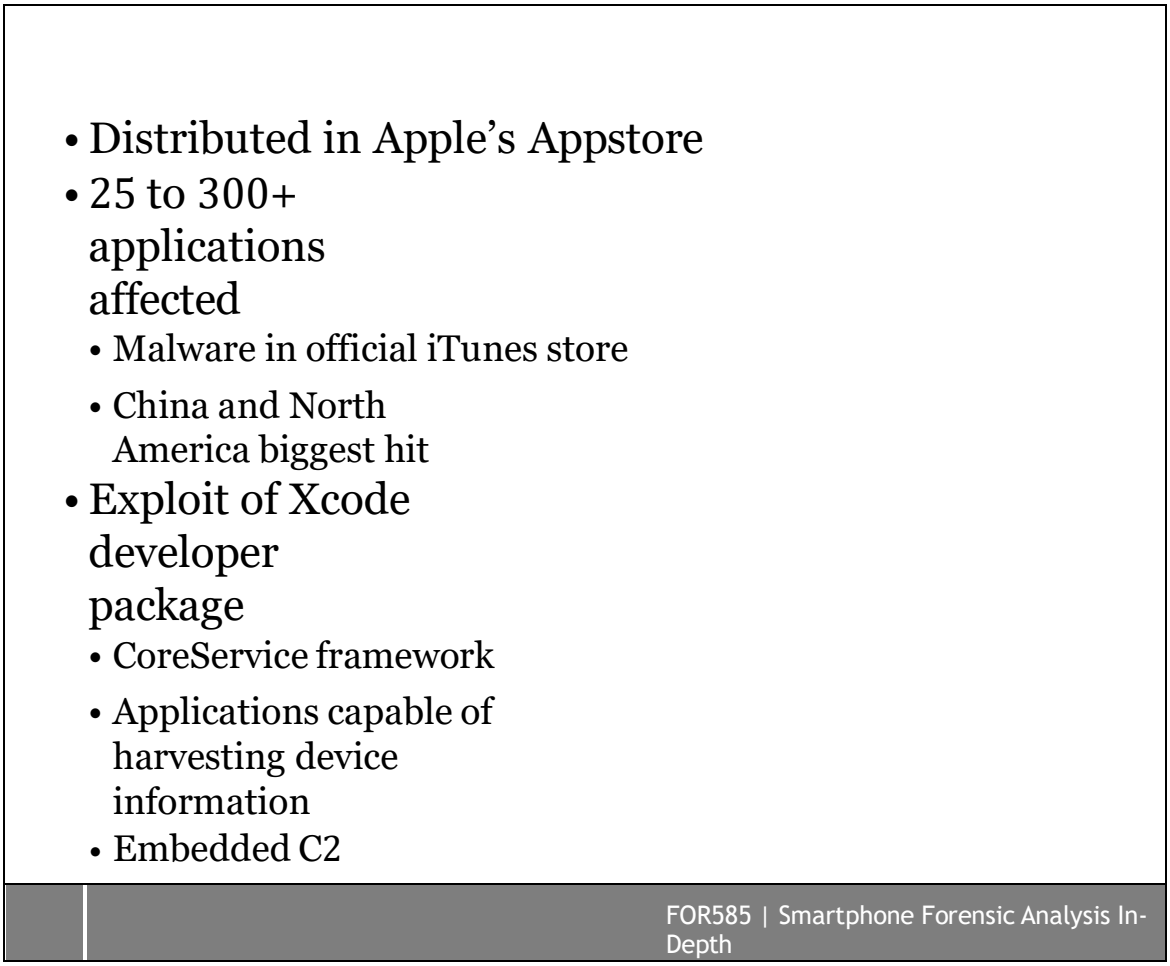
When the app is opened, seven additional APK files are silently downloaded, unlocking new malicious functionality. Each user interaction with affected applications triggers the sending of an SMS to a premium service, which is then deleted before it can be detected.

The additional APKs include spyware-like components. They are capable of harvesting sensitive user data and can passively record the device's audio and

exfiltrate photos, contacts, user files, and other data. RedDrop then uploads the data straight into remote file storage systems, such as Dropbox, for extortion purposes.[4]

References:

- https://for585.com/flubot
- https://for585.com/ranamalware
- https://for585.com/ranamalware2
- https://for585.com/nul-o (Wandera's write-up on RedDrop malware for Android)



In September 2015, the iOS malware XcodeGhost was discovered to have infiltrated Apple's tightly guarded iTunes Appstore. Although Chinese researchers report that over 300 applications could be affected, Apple recently published its list of the top 25 applications that were shown to be developed with the malicious code. Included in the top 25 list is the popular messaging application WeChat.[1]

The malware was injected into hijacked versions of Xcode, Apple's coding framework, which runs on Mac OS and is used for developing software for iOS and OS X. These tampered versions of Xcode were distributed via third-party sites for download. The

malware has hit China the hardest because developers there often experience exorbitantly longer download times when trying to access Apple's legitimate Xcode software from its U.S.- based servers.[2] Those who developed code with the hacked versions of Xcode unknowingly submitted malicious software versions to Apple, where it was published in the iTunes Store before the exploit was caught.

What Can It Do?
In the hijacked versions of Xcode, the CoreService development framework was replaced, and it then garnered elevated permissions for applications created with the software. The infected applications mined basic information, such as application name, version number, system version, language, country name, developer, app installation time, device name, and device type.

This collected data is then transmitted to a command and control (C2) server where it can be used for any number of purposes. Initial reporting suggested that this C2 could also issue commands to the infected device, which would be capable of opening a web browser or mimicking a login screen of a previously installed application to invoke users to enter usernames and passwords to be harvested and sent to the C2 for nefarious purposes. Apple has reported that there have been no examples of this scenario to date.[2]

References:

- https://for585.com/6v-qg (MacRumors.com list of XcodeGhost compromised apps)
- https://for585.com/c2ytd (Trend Micro article on XcodeGhost)

- Discovered by Ian Beer for Google's Project Zero and subsequently patched prior to being exploited in "the Wild"
- Kernel vulnerability in Apple Wireless Direct Link (AWDL) responsible for features like AirDrop/Sidecar
- Specially crafted Wi-Fi packets crash device/allow for malware

install
- Attacker could have access to sensitive device data and assets: keychain/microphone/camera
- Requires proximity to device

In 2020, Google Project Zero bug bounty hunter, Ian Beer, discovered a kernel vulnerability in Apple Wireless Direct Link (AWDL), which, if exploited, could give an attacker real-time device access and monitoring capabilities as well as siphon off sensitive data like images, audio files, video and passwords, and encryption keys from Apple's keychain. AWDL is the functionality behind Apple features like AirDrop and Sidecar.[1]

The vulnerability allowed for specially crafted Wi-Fi packets delivered to a nearby device to crash the phone and install malware. The exploit itself did require relatively close proximity to the target. The vulnerability was reported and then patched in May of 2020, and at that time, there had been no known examples of it being detected/abused "in the wild". [1]

Reference:

[1] https://for585.com/ioskernelvuln

- Bootrom exploit
- Vulnerability is hardcoded onto the chip; therefore, no patch
- Affects A5 through A11 chips
- (iPhone models: 4S – X)

- Allows for unsigned code to run on iOS devices
- Tethered exploit
- Requires physical access to the device
- Not persistent
- Will not survive a reboot

FOR585 | Smartphone Forensic Analysis In-Depth

In late September of 2019, a bootrom vulnerability affecting devices that have chips spanning A5 through A11 was detected and reported by a researcher who uses the Twitter handle, axi0mX.[1] This affects hardware models of the phone from the iPhone 4S through iPhone X. iPhone XS and iPhone 11 models have upgraded to the A12 chip and are not vulnerable to checkm8. The find was likely prompted by Apple patching a use-after-free vulnerability in the iBoot USB code, which was communicated during the iOS 12 beta release in the summer of 2018.[2]

This is a tethered exploit, which means that physical access to the device is required, but if an attacker gains access, they can then evoke unsigned code to run on a device to include malware and other spyware like applications. In addition to requiring physical access to the device, this particular exploit is not persistent, meaning that it will not survive a device reboot.

Shortly after the detection of the checkm8 vulnerability in November of 2019, the Checkra1n jailbreak was released. The Checkra1n jailbreak installs Cydia and Checkra1n applications to the Home screen and also, can be detected by the presence of the Checkra1n logo upon booting.

Additional Examples:

AceDeceiver - 2016 - iOS
In March 2016, Palo Alto discovered a new family of malware for iOS able to successfully infect non-jailbroken devices, which they named "AceDeceiver."[3] AceDeceiver is different from previously identified iOS malware because it attacks Apple's Digital Rights Management (DRM) system rather than using exploited enterprise certificates like previously discovered iOS malware did.

AceDeceiver installs itself without any enterprise certificate by exploiting design flaws in Apple's DRM mechanism. It has been removed from the App Store but can still spread due to the unique attack vector. It uses FairPlay, part of Apple's DRM system, to install malicious apps on iOS devices regardless of whether they are

jailbroken using a "FairPlay Man-in-the-Middle (MITM)" attack. This type of attack has been used since at least 2013 to share pirated iOS applications. This method could be adopted to spread new variants of iOS-based malware in the future.

While iOS-based malware only makes up a very small percentage of mobile malware, examiners need to be aware of the potential that it might exist on a device involved in their investigation. Mobile sandboxes can be useful in simple analysis of iOS-based malware. An example analysis report related to YiSpecter malware can be found at the link below.[4]

References:

- https://for585.com/iosbootromvulnerability
- https://for585.com/iosibootpatch
- https://for585.com/q0n7g (Palo Alto's write-up of AceDeceiver)
- https://for585.com/aqtkp (YiSpecter IPA analysis example from Palo Alto)

- **FORCEDENTRY** - targets Apple's image rendering library on all current versions at the time (Apple iOS, MacOS and WatchOS devices) **CVE-2021-30860**
- "Zero Click" malware (no user interaction) exploiting the native **iMessage** exploit on phones running the most up- to-date iOS version (14.6)
- Infects the device with **Pegasus Spyware** – originally designed by **Israeli NSO group** for Android, iOS and other operating systems and was initially detected after a failed installation attempt in 2016
- Detected by CitizenLabs based on iMessage attachments with multiple files with a .gif extension that caused the device to crash which were consistent with Adobe files

While “zero click” malware can affect a number of platforms, it may be most recently associated with the early fall of 2021 FORCEDENTRY attack, which leveraged Apple’s image rendering libraries associated with their iMessage platform. The term, zero click, implies that the exploit can be carried out on the device without any interaction from the user, similar to the StageFright attack we saw associated with the Android messaging platform in 2015.

This particular attack vector was used to infect vulnerable Apple devices with the Pegasus spyware, a platform agnostic spy-like utility originally detected in 2016, thanks to a fast-acting high-profile government official who was targeted.

CitizenLabs, who investigated the latest attack vector, noted the presence of multiple Adobe files, with the .gif file extension which caused system-related crashes.

Apple immediately responded with a firmware update to patch the affected operating system versions, and many vendors responded with solutions for detecting vulnerable and breached devices.[3,4]

References:

- https://for585.com/forcededntry
- https://www.sans.org/webcasts/what-you-need-to-know-about-cve-2021-30860-aka-forcedentry/
- https://github.com/mvt-project/mvt
- https://for585.com/imazingpegasus

- POISON CARP using non-zero days to exploit iOS and Android devices
- One Click exploits targeting Senior Tibetan group members
  - Users receive carefully

		crafted phishing/smishing messages • Android – 8 browser exploits/ 1 spyware kit • iOS – iOS exploit chain/ iOS spyware

The actor, POISON CARP, has been tied to multiple attacks on the mobile devices of senior member Tibetan group leaders. The attack begins with carefully crafted messages via the WhatsApp messaging platform coming from actors posing as journalists, NGO workers, tourists, and other legitimate business contacts.[1] The message often entices the user to click a link, which, in many cases, was shortened using bit.ly.

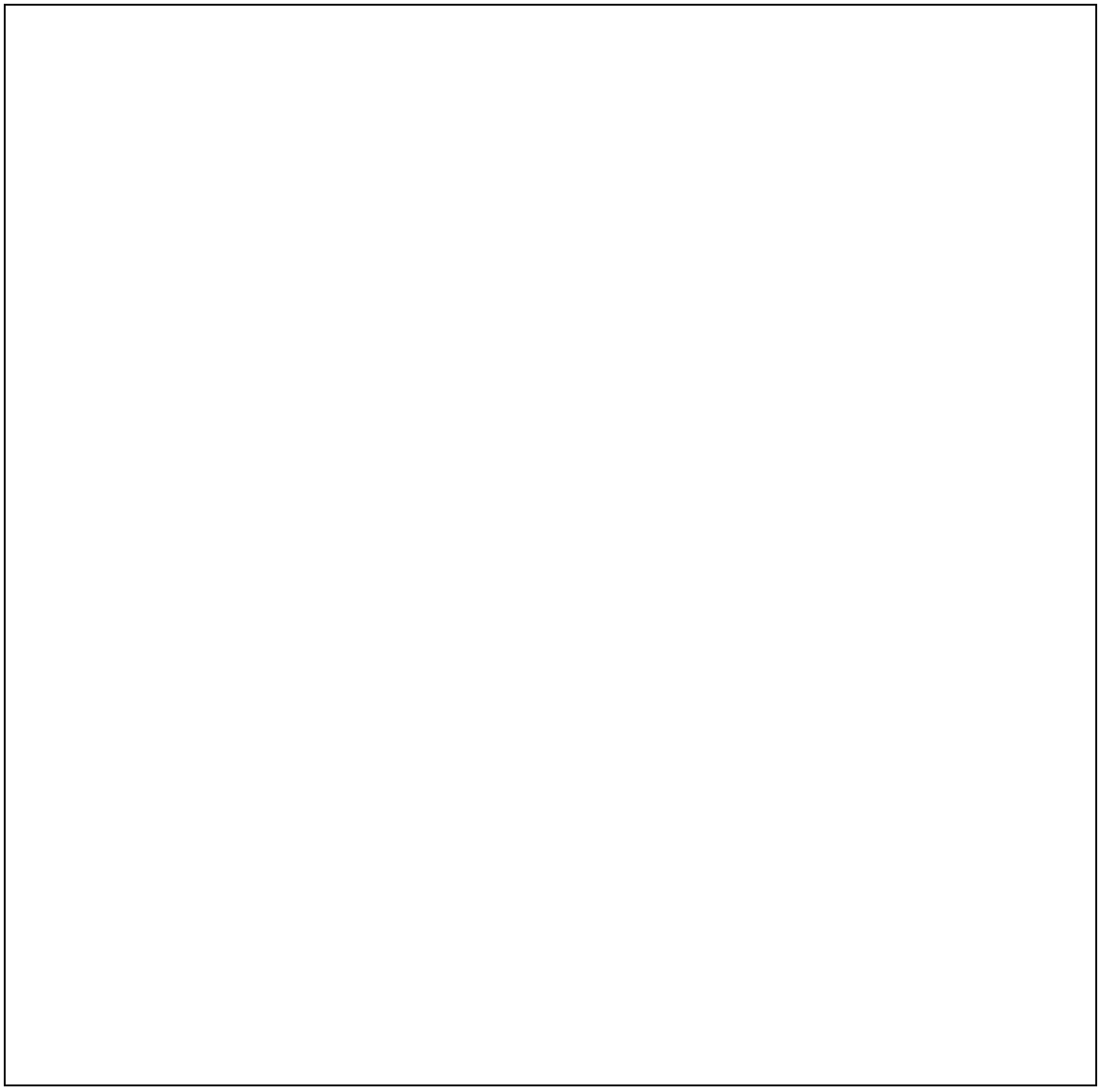
For iOS users, the shortened link would redirect users to www.msap[.]services (this link is intentionally disabled) where an iOS exploit chain resided targeting versions 11.0 through 11.4. Should the requesting iPhone's User-Agent string reveal that they were running one of the vulnerable firmware versions, the URL returned a valid html page with two iframes. One iframe displayed a benign decoy webpage, whereas a second, invisible iframe would take users to another exploit page on a different website.[1] The iOS exploit was a WebKit JavaScriptCore exploit which allowed privilege escalation and enabled a spyware-like payload to run on the device. The application establishes contact with the C2 where it then begins targeting user information like make, model, phone number, IMEI, IMSI, ICCID, network method, and storage capacity. The device could then be remotely queried for a list of applications from which the actor would like to exfil data. Analysis concluded that many applications were already hardcoded into the implant application for retrieval. Some of those include but are not limited to Viber, Voxer, Telegram, Twitter, WhatsApp, Facebook, WeChat, Yahoo Mail, Gmail, Outlook, QQMail, and Skype.

The process for Android users is similar, starting with an enticing WhatsApp message which users are encouraged to click by various social engineering schemes. Targeted

users have their UA string examined in order to determine how they accessed the site—whether they accessed it using a vulnerable browser version. For example, if a user accessed the server through Facebook using a webkit, and the User-Agent determined that the Chrome version was vulnerable, the exploit delivers and runs shellcode on the device that in-turn downloads a "Loader" (or ARMv7 ELF binary file), which it stores in (/data/data/com.facebook.katana/[NameOfBinary]) to maintain secrecy and persistence on the device. When the user invokes the legitimate Facebook application, it then reads the .so file into memory and kicks off the spyware installation and utilization. The malicious applications can then exfil sms, contacts, call logs, and invoke the device microphone and camera for exfil purposes.[1]

Reference:

- https://for585.com/oneclickexploits



A Kaspersky mobile threat report lists the following malware types in order of prevalence:

- RiskTool
- Trojan-Banker
- Adware
- Trojan-Ransom
- Trojan-Dropper
- Trojan-Spy
- Trojan
- Backdoor
- Trojan-SMS
- Trojan-Downloader

As with traditional computing platforms, there are mobile malware versions of backdoors, trojans, and worms. **Backdoors** are programs that provide unauthorized remote access to the mobile device. **Trojans** are programs that purport to be one thing but additionally perform other functions, such as stealing data, interfering with user control of the device, or using device resources without the user's knowledge. Trojans are often subdivided into the type of action they perform: trojan dropper, trojan spy, trojan downloader, and so on. **Worms** are programs that replicate themselves, creating similar or exact copies to connected devices or removable storage media.

RiskTool and Adware currently top the list when it comes to mobile threat detections. RiskTool capabilities include file concealment, hiding the windows of a running application and the ability to terminate applications/processes, and while the actual threat Adware seems low, due to the nature of their hooks into running applications, there is the potential for them to be misused by attackers. [1]

The bulk of malware is still attributed to various types of Trojans, with subcategories for their particular behavior: Trojan-Dropper, Trojan-SMS, Trojan-Spy, etc. [1]

SMS trojans send SMS text messages to premium text messaging services without the user's knowledge or consent, racking up large bills for the user. Fake install apps are malicious programs that mimic the activities of legitimate apps but require users to pay attackers via premium SMS charges. Trojan spy apps capture and transfer user data to attackers without the user's knowledge.[2]

References:

- https://for585.com/kasperskyreport
- https://for585.com/rp6l9 (Security Intelligence Article on mobile malware)

- Follow the money: Ransomware rises as solutions for mobile banking increase
- Leverage BIND_DEVICE_ADMIN, SYSTEM_ALERT_WINDOW, or most recently, built-in **Services** (like notifications), to keep ransom "note" in the foreground
- Most often data on filesystem is unaffected

A currently emerging threat is mobile **ransomware**. Infecting a system with malware that encrypts files and folders on the system and then requires the user to pay in order to regain use of files, or which accuses the user of a crime, such as possession of child pornography, and then requires payment of a "fine" to resolve the issue are examples of ransomware schemes. Common in the computing world, there is now an increase in the incidence of these schemes on mobile devices.

After a device is infected with ransomware, the malicious app prevents the use of the device and displays a pop- up window that notifies the user that his device has been encrypted or that he has committed an illegal offense and must pay a fine to regain use of his device or decrypt his data. Underneath, the data may or may not actually be encrypted, so it's worth attempting an extraction and examination on infected devices prior to considering ransom payment.

The Trojan-Ransom.AndroidOS.Small malware and its modification, Trojan-Ransom.AndroidOS.Small.o, were the most active in Russia and Kazakhstan. A new, similar mobile malware, AndroidOS/MalLocker.B, was detected by Microsoft Defender for Endpoint, and while the malware is new, many of the tactics are the same.[1] Users are often infected while visiting questionable sites, those hosting gaming, gambling, and pornography, and can be passed around via social engineering to users looking for free or cracked applications at no cost.

What is interesting about this application, and many of those that attempt to extort users by claiming to lock, encrypt, or delete files until a ransom is paid, is that most often, the file system data is left untouched by the offending malware.

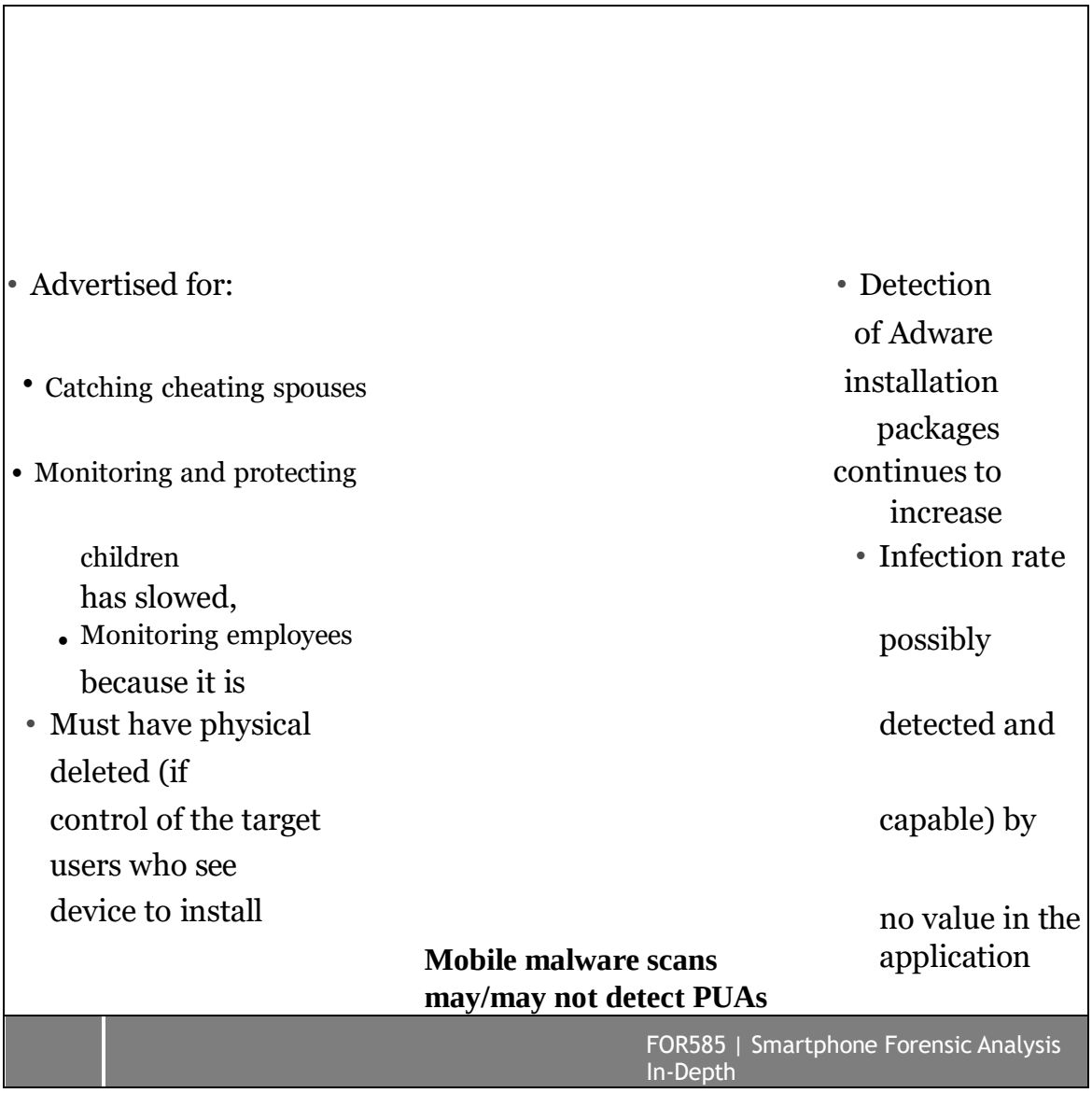
The way the attack is carried out, however, leaves the users to believe that they have no other way to access their data other than paying the ransom, because the screen appears to be locked down and the user cannot navigate away.

Historically, ransomware leveraged special permissions like BIND_DEVICE_ADMIN, that when granted to an app can wipe, lock or reset the passcode to a device or SYSTEM_ALERT_WINDOW, which with permission, can be used to overlay their notification message, in this case, the ransomware request, over all other screens with no way for the user to exit out of the offending screen.

After system protections were put into place to specifically thwart this type of attack method, new methods were introduced that included leveraging Android service components like call (and other notification types) and the "onUserLeaveHint()" callback method to create an automatic pop-up of the ransomware screen, which is pushed to the foreground of the user's device.

Reference:

- https://for585.com/androidransomware



Potentially unwanted applications such as **adware**, which displays ad content and may monitor user activity to provide targeted ads; **trackware**, which gathers data that can identify the device user to a third party; and **spyware**, which collects data about the user's activity and content from the device to report back to a third party are also potential threats in the mobile landscape. Although these programs can have legitimate uses, they can also be annoying to the user or potentially be used for criminal purposes.

One of the more onerous mobile threats is mobile spyware: software that allows the attacker to monitor the activities of the target phone and its user. Mobile spyware is generally advertised as a legitimate tool for catching cheating spouses, girlfriends, or boyfriends; monitoring and protecting children from predators and cyberbullies; and monitoring employees for productivity purposes and to prevent and detect data theft.

Mobile spyware programs can be purchased fairly inexpensively, and most work by using an installed application on the phone itself in conjunction with a web-based account that is used for reporting and monitoring activities on the target phone.

Different spyware packages have different monitoring capabilities, but most can track the location of the phone, intercept and report SMS and MMS messages, and report incoming and outgoing calls (sometimes allowing for recording or near real-time monitoring of calls). Some spyware can also be used to control the target phone to turn on the microphone to monitor the surroundings of the target phone or even to activate the camera to take video or still images remotely. Obviously, the opportunities to misuse mobile spyware to steal data, violate privacy, and illegally intercept and monitor phone calls are nearly endless. The presence of these software packages can be difficult to detect, but spyware should be suspected if a person reports that someone has inside knowledge of the content of private messages, emails, call content, or their physical location.

The following slides overview some of the mobile spyware programs on the market, but there are scores of different varieties of mobile spyware apps, and those apps may change over time; therefore, time and effort may be required on the examiner's part to find artifacts of mobile spyware on a device.

- **Verizon**: Family Locator Plan application's
- **US Cellular**: Family Protector Plan
- **AT&T Family Tracker**
- **Sprint**: Family Locator

- What is the intended purpose?
- What kind of artifacts can be generated?
- Can this serve as another source of evidence if other instances of user data are deleted?

Consider that any of the carrier-provided family monitoring tools can be effectively used as spyware applications when utilized beyond their intended purposes. These services are activated by an authorized individual on the mobile account and are designed to work with the phones they're being used to monitor. Like many other spyware applications, carrier monitoring tools generally consist of an application installed on the phone in conjunction with an internet-based portal login. In some cases, the phone may come from the carrier with the monitoring software installed, but it is not activated until the service is activated by the carrier. With the proper legal process served to the provider, information can be obtained about whether family monitoring plans are actively being used.

All U.S. mobile carriers have parental controls that are almost identical. Each offers the capability to:

- Block picture messaging
- Block unknown phone numbers
- Limit what time your kids can text or call
- Filter web browsing
- Use GPS tracking to monitor physical location

Some of these features can also be used similarly to spyware.

The debate over what counts as monitoring a loved one versus spying on them is one that is ongoing. For a discussion on the pros and cons of parents monitoring cell phone use, see https://for585.com/3ozrk.[1]

As previously stated, there are a large number and variety of apps (many of them free) on the Play Store and third-party app stores that are designed to be used as spyware applications for monitoring phone usage, location, and communications.

It is not possible to describe all the artifacts that could be present from these applications, and they may not show as icons on the phone on which they are installed. It is necessary to carefully examine the file system of a device suspected to be compromised by spyware applications to locate any apps that might be monitoring phone activities in a clandestine manner. Additional .ipa and .apk files are available for practice within your course media files.

It's been more than a decade since the first detection of malware threats to mobile devices, and the threat has changed significantly in the intervening years. As mobile technology gets more sophisticated and users depend on their mobile devices for functionality, the threats have become more technically sophisticated. We can expect to see additional changes as technology moves forward and changes; therefore, learning the skills it takes to address mobile malware threats is important.[2]

References:

- https://for585.com/3ozrk
- https://for585.com/n4slp (Sophos mobile malware timeline)

- Official app stores (low threat)
- Third-party app store repositories
  - Side-loading
- Malicious websites for direct "drive-by" download

installation
  - Pornography sites
- Direct victim targeting through advertisements, email, SMS, and MMS
  - "Smishing"
- QR codes
- NFC chips
- "Zero Click"
  - Requires NO user interaction

Malware infection can occur through numerous infection vectors, including through official app stores, when harmful apps are passed into the marketplace as legitimate programs, and third-party app-store repositories, which might not have stringent controls to vet new apps in their systems.[1] This method is particularly a risk for Androids with outdated operating systems and jailbroken iPhones.

Malware infections can also result from malicious websites that are designed to infect devices via direct "drive- by" download installation of malware or direct victim targeting through email, SMS, and MMS, where delivery of the malware can be achieved through a malicious attachment or URL via SMS- or MMS-based phishing attacks referred to as "smishing." Other emerging methods of malware delivery to mobile devices include directing the user to a malicious website via a QR code and close-quarters infection using NFC chips.

Reference:

[1] https://for585.com/malwareinfections

- Poor battery life
- Dropped calls
- Unusually large phone bills
- Data plan spikes

- Performance problems
- Unexpected device behaviors

Indications that a smartphone may have a malware infection or have spyware installed include poor battery life, dropped calls and call disruption, unusually large phone bills, data plan spikes, performance problems, and other unexpected behaviors, such as disappearing icons, mysterious icons that weren't installed by the user, unexpected restarts, etc.[1]

Reference:

- https://for585.com/nwoez (readwrite.com blog post on mobile malware signs).

Many different methods of mobile malware detection are currently in use or under development, including:

- **Signature-based malware detection**: A pattern-matching approach taken by many commercial antivirus solutions. The scanner scans for a sequence of bytes

within a program code to identify and report a malicious code or uses hash values of known bad files.

- **Specification-based malware detection**: Makes use of a set of rules for what is considered "normal" activity to determine the maliciousness of programs that violate the predefined rule set.

- **Behavioral-based detection**: Not only performs surface scanning, but also identifies the malware's action. This approach generates a database of malicious behaviors by studying several families of malware on a target operating system and distinguishes a malicious program from normal application behaviors. Behavioral-based detection systems are capable of detecting metamorphic malware that keeps reproducing.

- **Data mining detection**: Detects patterns in large amounts of data, such as bytecode, and uses these patterns to detect future instances in similar data.

- **Cloud-based malware detection**: Such as Google Bouncer, which automatically scans applications on the Google Play Store for malware. As soon as an application is uploaded, Bouncer checks it and compares it to other known malware, trojans, and spyware. Every application is run in a simulated environment to see if it will behave maliciously on an actual device. Google Play can remotely uninstall applications in the event that an installed app is later found to be malicious.

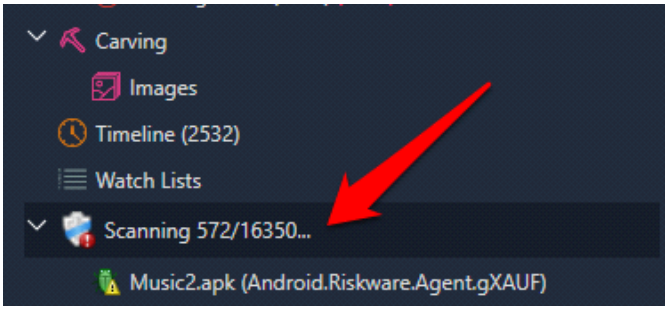
Physical Analyzer can be used to scan for mobile malware using Bitdefender's malware signatures. By clicking the blue shield in the Tools dropdown, you can open the malware detection wizard inside Cellebrite Physical Analyzer.

The first time you run the malware scanner in Physical Analyzer, you need to download the signature database from the website. Physical Analyzer prompts you through the steps to install the malware signature database.

The database can be installed locally or via the internet. Remember, your malware scanner is only as good as your latest signatures. **Make sure to update regularly.**

Once you install the malware database, you are prompted to choose the partition or file you want to scan. Select by checking the box and click the Scan button.

Once you click the Scan button, scanning begins in the background. You can watch the scan progress in the left pane of Physical Analyzer, and you can continue to work in Physical Analyzer as the scan progresses. Once complete, results appear in the tree pane on the left.



Check for suspicious application files

Research any error on me for suspicious files abou

Check installed or applications for suspicious files

Check Downloads folder(s) for visits to suspicious sites

Check browser history for suspicious email

Check for activity links from SMS, MMS, and time of infection

Examine phone r around the t tificatio t suspected hat migh c give you t lues infection

When examining data from a device that is suspected to be infected with either mobile malware or spyware, it is important to remember that the forensic tools you use to examine the device may not automatically detect malware on the device. They also may not detect spyware as an application of concern because spyware apps are

considered to have legitimate purposes, and so they are often not considered to be malware.

Therefore, during examination for malware and spyware, you may have to dig deeper and use clues from the device to help you determine if it is infected, what malware is causing the problem, and what that malware is doing on the device.

As mentioned previously, malware and spyware are introduced to a device in a limited number of ways (via an app store, malicious websites, email, SMS, MMS, and so on), and have to be installed in order to run on the phone. Therefore, if you are able to narrow down a date range for suspicious symptoms or can identify specific errors the phone is reporting, these may be important clues in finding malware and spyware on the device. By examining the data areas on the device that are associated with infection mechanisms, you can attempt to identify suspicious texts, email messages, or application downloads and installations.

If you know approximately when the phone started to act suspiciously, this information can be used to narrow the scope of your search to those files that came into the device on or around the time of suspected infection. Error messages and odd notifications displayed on the screen of the phone might give you further clues to research potential malware infections.

- The **App** directory from the **USERDATA** partition contains the package for all INSTALLED Android applications
- Displayed name depends on the tool you use, as the file system is normalized
- Most .apk files are legitimate applications
- Individual suspicious .apk files can be exported for further examination

The **App** folder contains downloaded .apk files for individual applications that are installed on the phone. The name of the path displayed depends on the tool you're conducting your examination in, as the file system is normalized. There may be hundreds of .apk files within the Data/App and/or Root/App folder. You may find additional .apk files located in the Downloads folder and in other locations on the device or associated MicroSD card.

The above images show data from the exact same phone image displayed in Oxygen Forensics and in Cellebrite in order to illustrate the differences in how the path name is displayed as the tools normalize the data. This is an important example of why it's so important to document what tools were used during examination. When viewing

individual files in Cellebrite, hovering the mouse over the file shows the file dates and times, as well as other information, such as file size. Once you locate a suspicious .apk file, it can be exported from the image for further examination and evaluation.

- Android accounts for the vast majority of malware, so we'll focus there
- Phone errors may give important clues about infection with malware or spyware
- Check for the associated .apk file in **USERDATA/App**
- Export the suspicious application for further analysis

In the example presented on this slide, the phone started giving an error "com.process.system isn't responding. Do you want to close it?" This error was considered to be suspicious because the phone recently started to respond slowly, and the battery was draining quickly.

A check of Data/App revealed an .apk file named com.process.system–1.apk with file dates around the time the phone started to act strangely. By exporting the suspicious .apk file from the image, it can be run through online malware sandbox sites for static and dynamic malware analysis, as shown later in this section.

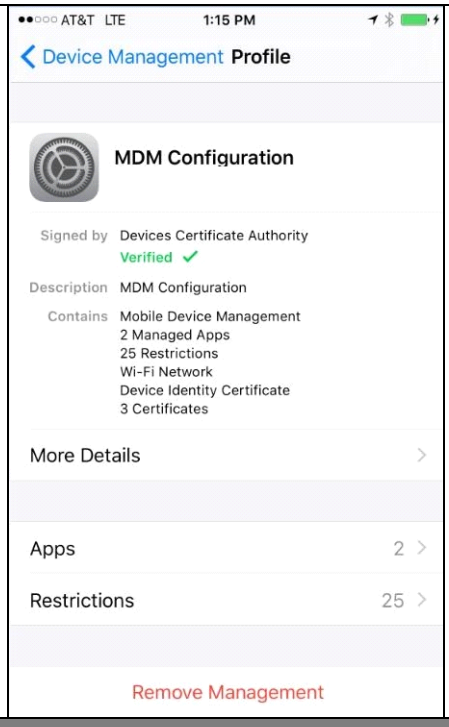
This error and com.process.system turn out to be associated with TRacer, a mobile spyware tool sold by KillerMobile.

Settings > General > Device Management

Check out details, Apps, Restrictions, etc.

Suspicious ones should be uninstalled

- Weather Underground, had unsuspecting users installing provisioning profiles after an application upgrade, which created new email accounts on devices which were receiving SPAM.



Pay attention to iOS applications that require that you enter the Settings menu to complete application setup. Remember, this is not a normal way to Download and Install iOS applications. This process could end up creating unwanted profiles on an iOS device for the propagation of spam or possibly malware.

Provisioning profiles are a legitimate developer tool used to provide access for testing. In fact, if you are doing development work, it is not uncommon to have profiles installed for things like Beta release versions of iOS software. By their nature, they have unique characteristics that give them certain permissions on devices that have granted authorization. This could mean the creation of root-level certificates and new accounts.

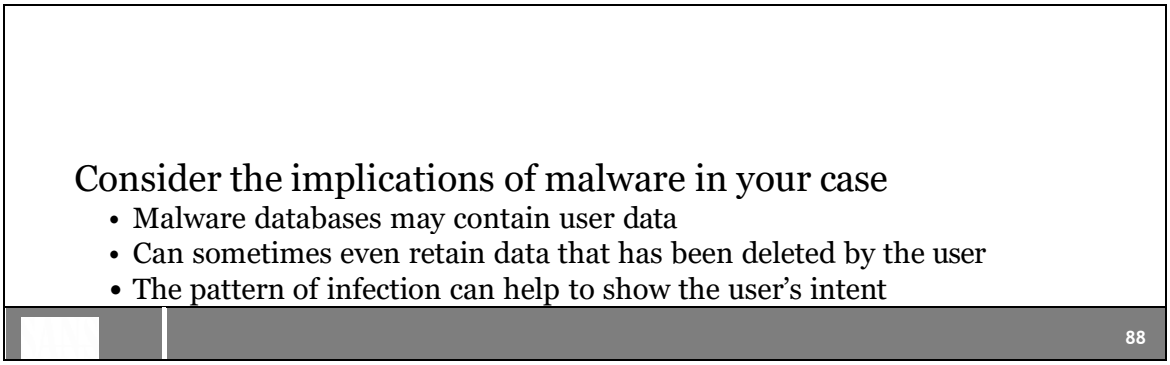
Examine the device for any provisioning profiles that the user did not setup, or that may have been setup inadvertently for possible insights into the strange behavior. On the device, Profiles can be found by going to Settings > General > Device Management. It is possible, because of having a corporate or company-owned

device, that you have a legitimate profile installed; if you don't fit this category, investigate any of the suspicious profiles. The absence of the Device Management option means that no profiles are installed on the device. Suspicious profiles can be removed from this location as well.

One well-known application, Weather Underground, had unsuspecting users installing provisioning profiles after an application upgrade, which created new email accounts on devices for the purpose of receiving SPAM.

References:

- https://for585.com/iosprovisioningprofiles
- https://for585.com/iosprofilemisuse



The presence of mobile malware may be automatically assumed to be problematic to an investigation because it brings up the potential for a malware-based defense. The same methods used to rule out malware as the culprit in a traditional computer forensic investigation can be used to rule it out in a mobile forensic examination as well. Consider whether the activity being investigated happened before or after the malware infection. Also research the specific malware variant and consider whether it is capable of the suspicious actions under investigation. This investigative methodology will help to rule the malware in or out as the source of suspicious activities.

The presence of malware may actually be useful to an investigation. Databases associated with mobile malware may contain user data that has been otherwise deleted by the user and can be a unique source of location data or other information important to an investigation.

Malware can also provide clues about the user's intent. In the above screenshot from an actual criminal case, Cellebrite's malware scanner has identified 13 positive malware hits using Bitdefender definitions. Most of these hits are executable files for video players. Windows-based executable files won't work on an Android- based phone. The second hit for the "flv_player_installer.apk" file would execute and run on an Android device. Always remember to update your malware definitions in whatever malware scanning tool you use. New variants are detected every day and are added to antivirus

software regularly. In this case, the first malware scan detected just one positive hit. A scan several months later resulted in additional positive hits.

When comparing the browsing history on the phone against the dates and times of the detected .exe and .apk files, it becomes apparent that the user was downloading multiple video players in an attempt to play videos with child exploitation content. In this particular case, the defendant pled guilty to child pornography charges based partially on evidence provided by mobile malware files.

Lab 4.3A

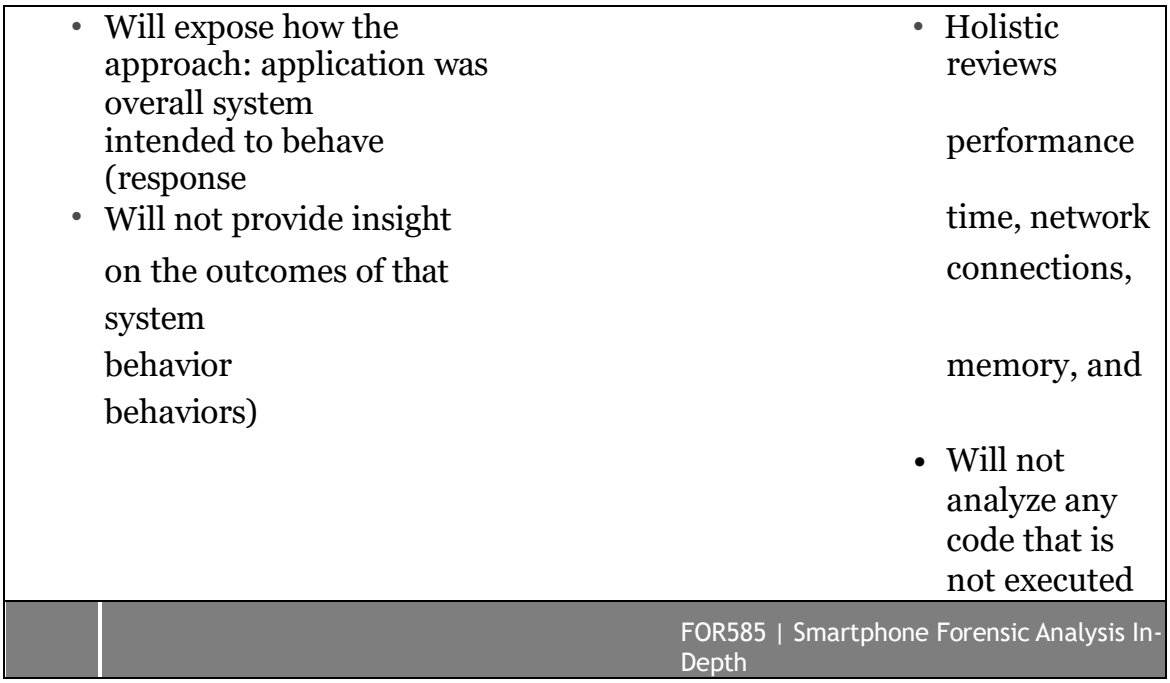
Hunting for Android Malware

FOR585 | Smartphone Forensic Analysis In-Depth

This page intentionally left blank.

- Code review performed takes place in a in **non-runtime** simulated live environment
- Analysis **live** or

environment



Static analysis is the process of inspecting an application's runtime behaviors for flaws and vulnerabilities, backdoors, and malicious code. The analysis is often performed with standalone tools in a non-runtime environment. This means that we look statically at how the written code was intended to behave, but we cannot see the outcomes of that behavior.[1]

To get a more holistic view of how the application actually behaves in the real world, dynamic analysis is usually performed. **Dynamic analysis** takes overall system performance into account and monitors behavior, response time, network connections, and system memory.[2]

Each method has its pros and cons. Static analysis may be more thorough and cost-effective, but it may be difficult to uncover minor coding flaws, and dynamic analysis is only effective in testing the part of the code that is being executed by the program. If it's not executed during a runtime simulation, then there will be no results related to that particular piece of code.[3]

References:

- https://for585.com/staticdynamic
- Ibid
- Ibid

- Applications are distributed as .ipa files
- Access to iOS applications is limited and will require Full Filesystem access (jailbreak)
- Unpack .ipa file to reveal the iOS .app source code
- Static examination proves to be more difficult than Android analysis
- Reversing tools are limited

iOS application files are distributed to the device as .ipa files. They have a standardized structure that is recognized by iTunes and the App Store. Performing static analysis on iOS devices is more difficult because iOS provides limited access to their .ipa files, and typically the files that you want to access can only be obtained through a jailbreak. Should you be fortunate enough to get your hands on .ipa files outside of iTunes for analysis, you will also need a Mac to access the application contents. Free tools, such as iFunBox, used to provide some functionality for viewing and transferring iOS applications from a connected device.[1]

The structure of an .ipa file is packaged similar to the .zip file format, and .ipa files can be unzipped to further examine their contents.[2] Inside the .ipa file is a package named "payload" that contains all application data. The file can be unpacked to review the contents. An unzipped copy of Pokémon GO is shown here in the slide above.

- To unzip an .ipa file, simply right-click on it and choose "unzip."
- Next, right-click on the "payload" package and choose "Show Package Contents" to view application contents inside.

References:

- https://for585.com/viewipa
- Ibid

Locate iOS applications on JB devices: /private/var/containers/Bundle/Application/<GUID>/<MyApp>.app

- **On LEFT:** Examining payload directory, right clicking on package name and "Show Package contents"
- **On RIGHT:** Locate the app GUID of interest and expand the contents of the .app file
- Both will provide you with the same files for static analysis

Locate the application on a jailbroken iOS device in /private/var/containers/Bundle/Application/<GUID>/<MyApp>.app

Notice, when the MyApp.app folder is expanded, the contents matches the payload data seen on the left.

Every iOS bundle will contain some required and recommended files. The most important of these, for our purposes, will be the ones that contain the application source code.

Once inside the MyApp.app folder, locate the binary source code by searching for a file of the same name, minus the .app extension.

EX: Bumble.app > Bumble

Reference:

https://for585.com/iosappbundle

iOS applications are developed in Obj-C and Swift. Application reversing proves to be more difficult than Android static analysis, and, therefore, dynamic analysis, combining both network and behavioral inspection, often provides the best insight. Some of the tools that can be leveraged for static analysis are strings, otool, and class-dump.

The application binaries stored on the file system, at rest, are encrypted, so in order to access their contents in any meaningful way, they must be accessed on a live, jailbroken system and the contents dumped out in a decrypted state that has been afforded by the device operating system, which is accomplished by your commercial forensic tools. Free tools for accomplishing this task include: frida-ios-decrypt, Clutch, and dumpdecrypted.dylib.[1]

With a decrypted binary, **otool** will identify libraries of possible interest, whose functionality can be further investigated using open-source search methodologies, **class-dump** can provide useful header information for classes, protocols, and categories, and **strings** can return a number of useful results like HTML source code, application directory structure, and files of interest.

If you are interested in reversing iOS or Android applications in-depth, consider taking SEC575, which covers all these topics and more.

Reference:

- https://fadeevab.com/decrypt-ios-applications-3-methods/

Android application files have an .apk file extension. APK files are archive files that generally contain the following files and directories:[1]

- **META-INF directory:**
  - **MANIFEST.MF:** the Manifest file, which contains metadata about the comprehensive package.
  - **CERT.RSA:** The certificate of the application.
  - **CERT.SF:** The list of resources and SHA-1 digest of the corresponding lines from the MANIFEST.MF file.
  - **lib:** A directory containing the compiled code that is specific to a software layer of a processor.
  - **res:** A directory containing resources not compiled into resources.arsc (see below).
  - **assets:** A directory containing applications assets.
  - **AndroidManifest.xml:** An additional Android manifest file that includes the name, version, access rights, and referenced library files for the

application. This file may be in Android binary XML, which can be converted into readable plaintext by tools such as AXMLPrinter2, android-apktool, and Androguard.

- **classes.dex:** The programming classes compiled in the .dex file format interpreted by the Dalvik Virtual Machine.
- **resources.arsc:** A file containing precompiled resources, such as binary XML, for example.

Every Android application runs in its own process, with its own instance of the Dalvik Virtual Machine. Dalvik has been written so that a device can run multiple VMs efficiently.

Android programs are compiled into .**dex** (Dalvik Executable) **files**, which are zipped into a single .**apk file** on the device. The .**dex files** can be created manually by the developer or automatically by Android when translating the compiled applications written in Java. These files are used to hold a set of class definitions and their associated data, and .dex files are interpreted by the Dalvik Virtual Machine.

.dex file structure includes the following elements:[3]

- Header: Contains basic file information such as size and elements
- String_ids: Contains a list of identifiers of all strings used
- Type_ids: Includes a list of identifiers of all the types (classes, boards, primitives)
- Proto_ids: Contains a list of prototypes (structures) for file references
- Fields: Includes a list of field identifiers
- Methods: Contains a list of identifiers of all the methods included in the file
- *Classes:* Consists of eight parts: class id, access_flags, super class type_id, interface list address, source file name string_id, class data address, address of the data initializing the static fields, address of the related annotations to the class
- Data: A section of data

Example Scenario for unpacking/decompiling a .apk package:
The *classes.dex* file within an .apk file contains programming classes for the application and review of this file can show us what the application is programmed to do.

Step 1: Unpacking
The compiled contents of an .apk file can be viewed by simply renaming the file to .zip and opening it. This process is known as “unpacking” the .apk file. In the example shown in the slide, the .apk file associated with the application “Pokemon Go APK” is named **pokemongo.apk**.

By renaming it to **pokemongo.apk.zip** and then opening the zip file, you can now see and browse through the various files and folders within the .apk file. This may give you some important information about the application’s activities, both legitimate and illegitimate. In the example, an .apk file named Zombie Highway has been unpacked.

Step 2: Decompiling
The next part of the examination process is decompiling the classes.dex file for the application within the .apk file.

To do this, locate the **classes.dex** file from within the unpacked .apk file.

Next, **COPY** the **classes.dex** file into the directory associated with the program **dex2jar**. Then open a command prompt and navigate within the command prompt window to the "dex2jar" folder on your desktop.

Inside the dex2jar-2.0 folder exists the batch file which you will run against the classes.dex file that was just placed in this directory.
From the command prompt type:
d2j-dex2jar.bat classes.dex.

Step 3: Results of Decompiling
Running this batch file creates Java .jar file named **classes_dex2jar.jar** that contains all of the Java classes that were contained within the classes.dex file.

Dex2jar is included in the course VM, on the desktop in the fences area, and is available as a free download at
https://github.com/pxb1988/dex2jar.

Step 4: Analysis of the Application
You can now use a free Java Decompiler tool called **jd-gui** to view the data you just unpacked and decompiled from the .apk file.

Double-click the **jd-gui** executable file and click Run if requested. Next, select **File** and **Open File**, then navigate to the **classes_dex2jar.jar** created in the previous step and open this file to view the underlying code.

The Java Decompiler allows you to view, navigate, and search the data that was packed inside the .apk file to see what the application was programmed to do. You may find IP addresses, email addresses, information about permissions, or other information that will help determine whether the application is malicious. It includes color- coded source code and will highlight the various classes you select as you click on them.

Color-coding for classes in JD-GUI is as follows:[4]

File-related classes (in **red**): for access, reading, and writing local files.
Java reflection classes (in **green**): for creating new classes and instances and invoking methods dynamically.

jd-gui is included in the course VM, on the desktop, in the fences area. It's available as a free download at
https://code.google.com/archive/p/innlab/downloads.

References:

- https://for585.com/nvaf6 (.apk file)

- https://for585.com/k14j0 (Dalvik Executable Format)
- https://for585.com/q6opw (.dex file anatomy description)
- https://for585.com/j8qyx (On Cyber Blog: GM Bot Android Malware Teardown)

- Introduced in Oreo (Android 8)
  - .vdex
  - .odex
  - .art
- Located in **USERDATA/app/<app_name>**

- Decompile and Extract .dex bytecode from .vdex using vdexExtractor tool

https://github.com/anestisb/vdexExtractor

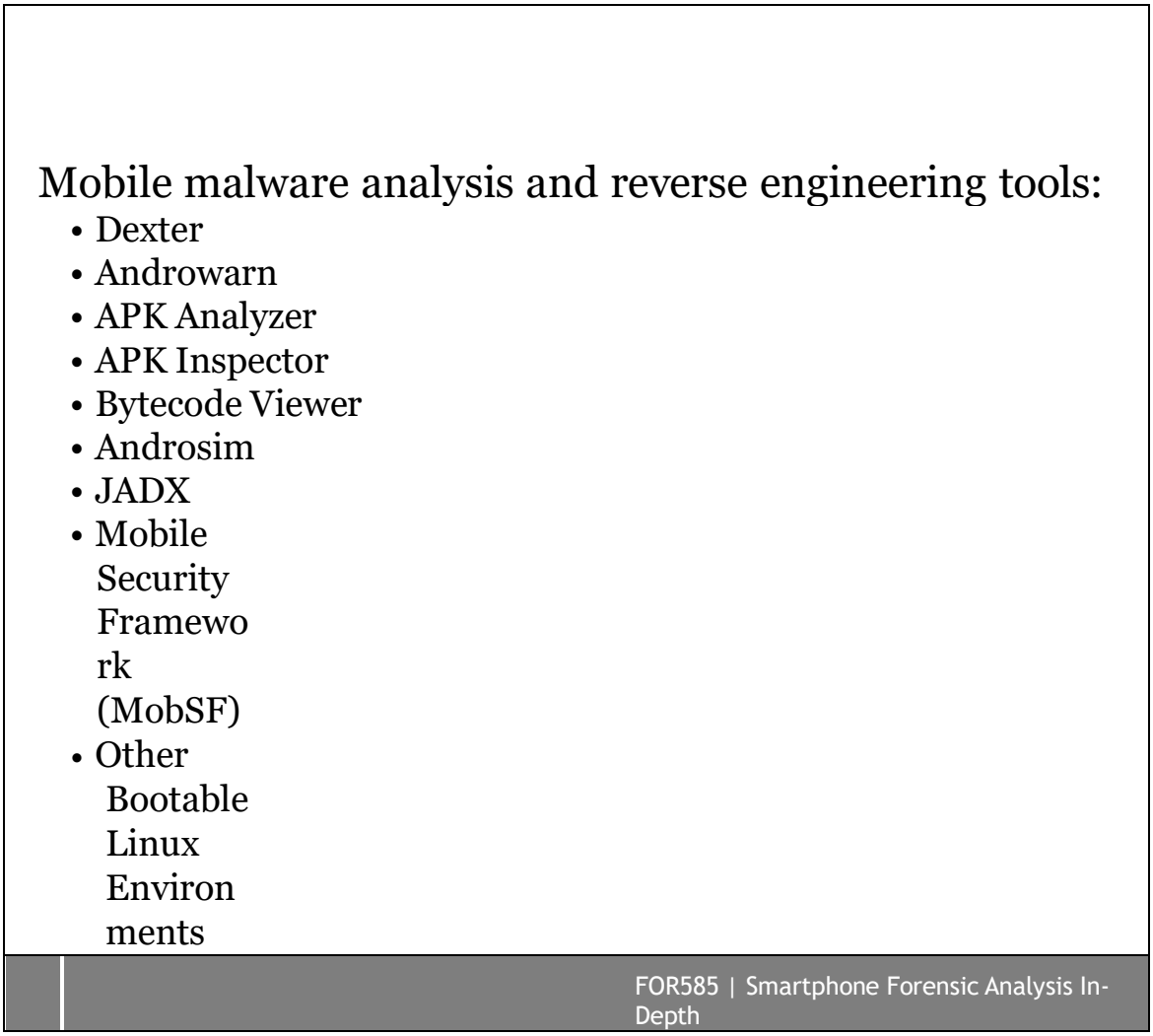
Introduced in Oreo (Android 8, API-26), you may notice new files when attempting to locate the Android application package. While the path to the data remains consistent: **USERDATA/app**, you will find several new files within the specific application folder.

- **.vdex:** Uncompressed DEX code and metadata files for verification.
- **.odex:** Ahead of Time (AOT) compiled code, used for reducing start time and improved application performance.
- **.art** (optional): Android Runtime (ART) for some strings and classes in the APK. ART replaced Dalvik as a way to translate Android bytecode instructions.[1]

For the purposes of static analysis, we need to target the .vdex file. In order to review the source code, it must first be converted using a tool like the open-source vdexExtractor which is available at https://github.com/anestisb/vdexExtractor and is a "Command line tool to decompile and extract Android Dex bytecode from Vdex files that are generated along with Oat files when optimizing bytecode from dex2oat ART runtime compiler."[2]

References:

- https://for585.com/vdexandroid
- https://github.com/anestisb/vdexExtractor



A number of tools can be helpful in reverse engineering and analyzing mobile malware, including:[1]

- **Adhrit:** Android APK reversing and analysis suite (https://github.com/abhi-r3v0/Adhrit)
- **Dexter**: Web service for uploading Android applications that are then statically analyzed. Dexter provides a quick overview of the metadata within the application and the included packages.
- **Androwarn:** "Yet another static code analyzer for malicious Android applications" (https://github.com/maaaaz/androwarn)
- **APK Analyzer:** View the absolute and relative size of files that compromise the APK and perform comparisons, view DEX, Android resource, and AndroidManifest.xml

- **APK Inspector**: Collection of many tools within one user interface. APK Inspector comes with Jad, a Java decompiler.
- **Bytecode Viewer:** a lightweight user-friendly Java Bytecode Viewer
- **Androsim:** Compares two Android .apk files (diff)
- **JADX:** More robust tool similar to jd-gui. Handles obfuscated code and handles the decompiling for you.
- **Mobile Security Framework (MobSF):** Automated pen testing, malware analysis, and security assessment framework (for Android/iOS/Windows) for static and dynamic analysis. (https://github.com/MobSF/Mobile-Security- Framework-MobSF)
- **FREE bootable Linux environments:** mobile device forensics and analysis tool. The virtual machine includes emulators that allow you to simulate network traffic and allow the examiner to perform both static and dynamic application analysis

EXAMPLES:

- **Santoku Linux:** https://santoku-linux.com/
- **Mobexler:** https://mobexler.com/
- **Tsurugi Linux:** https://tsurugi-linux.org/

The software we provide for use in this course can be found at the following links: Android Developer Toolkit: SDK Tools: https://developer.android.com/studio#download Java Development Kit: https://www.oracle.com/java/technologies/javase-downloads.html

Reference:

- https://for585.com/wr-c6 (Uceka blog entry on Android malware analysis).

- JADX is a free JAVA decompiler
- Handles Unpacking and Decompiling .apk files
- Can also loa

d
dec
om
pile
d

- **Requires fewer steps**
- More robust than jd-gui
- AndroidManif
  est.xml is
  included for
  analysis

JADX will feel very similar to jd-gui, and it will eliminate the steps of unpacking the .apk and converting the
.dex file to a .jar before loading into the tool, which allows the examiner to view the Java source code. The tool is available for Windows, Linux, and macOS.

Navigating through the decompiled code, you will likely see one or more **packages** that consist of many different **classes** that provide the blueprint for the **objects** that make up the Android application. Classes (and the objects derived from them) will consist of both a state and a behavior. A state is stored in a field or variable, and behaviors are exposed through methods or functions.[1]

Classes

- State: (stored as a **field/variable**)
- Behavior: (exposed in **method/function**)

Classes can be public, private, and/or protected, and these are denoted by different colors within the tool interface.

- **Green Circle—Public:** can be used outside of this class by any other class/function
- **Red Square—Private:** can only be used inside this class
- **Yellow Diamond—Protected:** class members and functions can only be used inside the class or by classes derived from this class

Reference:

[1] https://for585.com/javadefinition

- Another open-source programming language for Android
- Can still be analyzed with your typical analysis/reverse engineering tools
- Structure will include a “kotlin” directory
- Differences in obfuscated and unobfuscated versions of the code

Kotlin is another programming language for Android, which is essentially just a streamlined version of Java. It is also a free, open-source tool. Compiling an application written in Kotlin will still combine the same artifacts into the .apk application package (classes.dex, resource files, and the AndroidManifest.xml among others).

When examining unobfuscated code, slight differences in syntax language exist that can be seen when analyzing the classes.dex files. The major difference between Kotlin-developed applications and Java is that applications developed in Kotlin will have a slightly larger size, which can be directly attributed to the existence of an extra "kotlin" directory and a larger classes.dex file. Other main items, like the AndroidManifest.xml, resources.arsc and res, will have the exact same size as their Java-based counterparts.[1] Compiling obfuscated code will result in identically sized classes.dex files when performing static code analysis. This is likely due to how the obfuscation is handled within Android Studio prior to compiling.

Reference:

[1] https://for585.com/kotlin

- Applications may be obfuscated

  to disguise their true purpose
- **Methods and Variable** names are usually nonsensical
- Error **message/strings** can also be **encrypted** to prevent easily reversing code
- ProGuard is a free tool

- DexGuard and PreEmptive are some examples of paid obfuscators which provide additional resources

Obfuscation Examples: PreEmptive

It is possible that the application you are statically analyzing has been obfuscated. The purpose of obfuscation is to make the process of reverse engineering an application to gain insight into its processes and methods difficult or near impossible, and it is a practice that is becoming more common, especially with applications that contain sensitive data or those that want to steal yours!

There are both free and paid tools that offer obfuscation. ProGuard is an open-source tool that can be used to obfuscate code, and the examples shown above are from a commercial tool, called PreEmptive, which provides a layered approach to obfuscating Java, Kotlin, and Android applications.

While obfuscation makes nonsense of most of the code base, like the names of the variables, classes, and methods, it may be possible to run a strings search on the code to locate data of value. As mentioned in the blog by Chris Smith, sometimes Android applications developed in Kotlin may have an accompanying metadata file that is not obfuscated when using ProGuard, which can still provide insights into the application's sensitive data.[2]

While de-obfuscation is not impossible, it is a time-consuming process that will involve learning about and piecing together parts of the code to determine its functionality, which is outside the scope of this course.

References:

- https://for585.com/obfuscationtoolspreemptive
- https://for585.com/kotlinobfuscation

- Start by reviewing the following items
  - **Activities** (UI)
  - **Services** (running in the

background)
- **Broadcast Receivers**
  - (Intents)
- **Content Providers**
- **Requested Permissions**
  - Are the application permissions consistent with the application's purpose?

Figuring out where to start when you are performing static analysis can be daunting. The items listed below will help to enumerate what you may be able to infer about the application and its capabilities.

Requested Permissions: These are the permissions that an application is granted by the user (1) during installation or
(2) upon execution/runtime. These will be included in the AndroidManifest.xml file.[1]
Activities: Main application components that the user interacts with directly. Applications are often comprised of one or more activities that the user can navigate to and from and interact with directly through the User Interface (UI).[2] **Services:** The components are

integrated with the application for handling tasks that often happen in the background. The user isn't usually directly interacting with the services. Some examples of a service-related activity include a file download that happens in the background, music playing during a game application, or calculations that are kicked off and running in the background.[3]

Broadcast Receivers: This facilitates the exchange of information for applications. Information can be delivered between app components or between entirely different applications using broadcast receivers. Applications can be programmed to "listen" for events, and these events are then delivered to the broadcast receivers by using **Intents**.[4]

Content Providers: How information is stored within an application and shared between multiple applications. An example of a Content Provider is the contact database often used by many native and third-party applications. The code would call the ContentResolver that passes on the request to the proper Content Provider.[5]

References:

- https://for585.com/permissions
- https://for585.com/activities
- https://for585.com/services
- https://for585.com/broadcastreceivers
- https://for585.com/contentprovider

An application's required permissions will be declared and contained in the AndroidManifest.xml file, and up until Android version 6, users accepted/or denied an application based on their overall blanketed application permission approach. Many applications often declared access to permissions which overextend the scope of the application itself.

In Android 6 (Marshmallow) a new permission management model established different categories of permissions, including those deemed dangerous and normal. An application can be installed without an overarching consent to agree to all of the requested permissions, but a user is asked to grant permission to the application when launched if "dangerous permission" is requested. By going into the Settings menu, changes to granted and denied permissions can be altered. Only applications written to take advantage of API 23 or later will be afforded this level of user scrutiny. Think of "dangerous" permissions as those that grant access to user- created data generated and maintained on the device.[1]

Android researcher, Josh Hickman along with Magnet's, Chris Vance did a nice job of specifically highlighting those "dangerous permissions and their implications in a blog post referenced below.[2]

References:

- https://for585.com/androidpermissionlevels
- https://thebinaryhick.blog/2021/01/26/androids-dangerous-permissions/

- Export suspected malware .apk files to a well- marked folder, such as “Suspected Malware”
- Submit suspicious files to your favorite sandbox site(s) for analysis
- Review results to determine what actions the .apk file was involved in

NOWSECURE

Suspicious .apk (or .ipa for iOS) application files can be examined by sandbox services online for easy automated analysis. Files should be exported from the smartphone dump to a well-marked folder. Once exported, these files can be submitted to one or more online sandbox sites for static and/or dynamic analysis. File hashes can also be submitted, and results are returned if an identical application has been analyzed previously. Numerous free sites exist for inspecting .apk files to determine what their functions are, such as:

VirusTotal:

https://

www.virustotal.com/gui/

JoeSandbox

https://

www.joesandbox.com

These websites provide detailed information about submitted files that help you to determine if an .apk file is malicious and what that file is designed to do. A list of available automated application testing resources is also maintained by FOR610 Reverse-Engineering Malware course developer, Lenny Zeltser.

https://zeltser.com/automated-malware-analysis/

In addition to free tools, there are a number of paid resources that will provide a very detailed report of mobile applications available for iOS and Android devices. **NowSecure**, a mobile device security research firm, has a very comprehensive tool for identifying security risks in both Android and iOS applications. The company sponsors and maintains the open-source software (OSS) toolkit, FRIDA, developed by

NowSecure Security Researcher **Ole André Vadla Ravnås**, which is used by developers, reverse engineers, and security researchers worldwide.

- Continuous monitoring option
- Static, Dynamic, and Behavioral testing
- Identify changes with each new app version
- Risk scores based on CVSS-scored findings, summarized by impact, and broken down by type
- Description on how vulnerability could be used by attackers
- Directly link vulnerability to path in source code
- Identifies weaknesses in OSS and third-party libraries and detects common security risks like XSS and data leakage

Paid tools will often provide more in-depth reports, like these snippets provided by NowSecure Platform, their Intel/Security Automation tool. NowSecure utilizes a combination of Static, Dynamic, and Behavioral testing to generate security reports on mobile applications. Of particular interest to forensic examiners, the tool will identify the use of vulnerable third-party libraries or possible application leakages through the use of elevated permissions, which can provide analysts with another area to investigate in their search for sensitive user data on the device.

For FOR585 students who are interested in supplementary capabilities offered by this tool, refer to your course Dropbox link or utilize the site below to sign-up for a trial license:

https://info.nowsecure.com/free-mobile-app-security-report.html

Lab 4.3B

Android .apk Static Malware Analysis

This page intentionally left blank.

This page intentionally left blank.

It is not uncommon for users to go through and "clean up" or delete data that they do not want to leave behind on smartphones. This can include selectively deleting messages, clearing device cache, or browsing/chat history, deleting media files or databases, installing third-party cleanup utilities, removing entire applications directories, or even invoking a device wipe.

A completely clean device may look suspicious, but there are many legitimate reasons that may cause a user to wipe a device. Based on just the data left on the device, we may be able to get a good idea of when the wipe occurred.

- .obliterated file exists (may not always present)
- *containermanagerd.log*
- com.apple.purplebuddy.plist
- *logd.0.log*

The main methods for wiping an iOS device occur through the device or the Find My iPhone application. To wipe the iOS device, simply log into the device and go into the General > Settings and select Erase All Contents and Settings. The device commences a wipe, theoretically rendering all data unrecoverable. Keep in mind that this is device-dependent and recovery method-dependent because some traces on iOS devices that have been wiped have been recovered.

Wiping a device may delete all traces to evidence relating to your investigation. The best methods to determine if a wipe occurred include examining the timestamps on the device and examining the file system for the .obliterated file.[1]

On most devices, you may see the presence of the .obliterated file, which is created when an iOS device is wiped.[1] The .obliterated file is located in the /private/var/root folder. The presence of the .obliterated file shows that the iOS device was wiped at some point; however, the exact date cannot be determined based on the examination of this file alone, but the modify date is a good indication. The file should be zero bytes in size and may not always exist. Again, the different iOS devices and version affect the creation of this file. If you find an .obliterated file, try to correlate the datetime of that file to the activated date. To obtain this file, a Full File System extraction is required. The com.apple.purplebuddy.plist from /root/private/var/mobile/Library/Preferences/ has an entry of interest and is captured via a backup. There are additional files that will also help, but again, you will need a Full File system to access the /private/var/root/Library/Logs/MobileContainerManager/conterinermanagerd.log.X and logd.0.log found in private/var/db/diagnostics/.[2] For the conaintermanagerd.log, you may find several files that are appended with numbers (0,1,2, etc.) The oldest file will have the largest number, which means the one that is holding the most recent wipe information is the one that starts with 0. The file logd.0.log tracks timezone changes. When a user sets up an iPhone, the select a region/timezone and this file are updated.

References:

- https://for585.com/wipe
- https://for585.com/wipeartifacts

Below is an example of the file system view containing the .obliterated file. The modify time of this file is a good indication of when it was wiped.

- SMS.db
- callhistory.storedata
- **AddressBook.sqlitedb**
- Voicemail.db
- Notes.sqlitedb

The timestamps for the files listed in this slide commonly maintain the original HFS+ creation timestamp. Thus, when the device is wiped and/or restored from a backup, the dates/times of these files are updated to reflect the *approximate* reboot time after a wipe or restore.[1] Testing in current iOS versions shows that while it can give a general time frame of when the wiping activity occurred, dates are not exact. NOTE: The modify time has NOTHING to do with the wipe/restore.

Third-party applications may retain the original date of installation; however, depending on the device and wipe method, the timestamps may be updated. Common practice is for the third-party applications to be copied back onto the device after the wipe, maintaining the original date of installation.

If the timestamps of the third-party application files precede the timestamps for the rest of the files (especially those highlighted above), and the device appears to have been altered (contains a limited amount of data), it is likely that a wipe occurred.

Reference:

[1] https://for585.com/wipe

As discussed in section 3, ArtEx is an open-source tool that is great for triage and forensic validation. In this example, we used ArtEx to validate our predictions on when an iPhone X running iOS 14.2 was wiped. As the slide shows, we select **Device Wipe** and are presented with a datetime for the wiping activity. Additionally, we can click on the file under **Source** and examine the native data. On the highlighted line, you will see the magic words, "this is an erase install." A very important aspect of this file that was detected by Heather Mahalik and Ian Whiffin, as stated in their blog, is that this file stores all data in Pacific time.[1] It must be converted manually to the local time for the user. This log shows the time of 13:26:30 but the device was wiped at 16:26:30 since the user lives in Eastern time.

Reference:

[1] https://for585.com/wipeartifacts

While in ArtEx, we dive a bit deeper, and we look for the timezone change from logd.0.log. When iOS devices are first booted, or booted after a wipe, the timezone is automatically Pacific time. Why? Because that is where Apple is. When the user opts to set the timezone, this file is updated. Hence, it's a fantastic file to see when the device was first booted. If the user lives in Pacific time, rely on the other artifacts mentioned.

- reboot,factory_reset,(unix epoch timestamp
- First in, first out (data will be missing if more than 3 reboots

occurred.)

- Can be accessed from non-rooted device using ADB commands

  (Android Triage scripts)

- Check the modified time for **factory_reset** & **last_boot_time_utc** files

- Samsung only, multiple wipe entries recorded

- Samsung only, wipe timestamp and reason (Ex: recovery, userrequested, adb) recorded

- Device MFR (OEM)
- OEM Implementation
- Android version
- Access to filesystem

  (root vs. non-root)

The research into detecting iOS wipe times conducted by Ian Whiffin, Jared Barnhart, and Heather Mahalik sparked a companion project from Android researcher, Josh Hickman. Detecting evidence of a wipe or the booting into an Android OS after a factory reset can prove to be slightly more difficult due to the existence of so many variants of the open-source operating system. However, Josh pulled together a large collection of Android artifacts which contain evidence of wiping activity. Many of these files will require a full filesystem acquisition or physical access to the rooted Android device.

Josh's research (conducted on a Pixel and a Samsung variant) can be found on his blog, https://thebinaryhick.blog/2021/08/19/wipeout-detecting-android-factory-resets/, but some of the highlights have been included here.

The persistent_properties file file at /data/property/ is available on both manufacturers tested (Samsung and Pixel). It should contain a **reboot,factory_reset,** (followed by a unix epoch timestamp). This, however, is a first in, first out artifact; if rebooted more than three times, the evidence of wipe (factory_reset) timestamp is eliminated. If you only see multiple reboot,userrequested (along with timestamp) indicators, it is possible that your suspect device has been rebooted more than three times, thus flushing the artifact associated with the factory reset. This artifact can also be accessed from a non-rooted Android device by running ADB queries.
Selecting "Collect Basic Information" using Mattia Epifani's Android Triage script will output results of this information to a text file for review.[2]

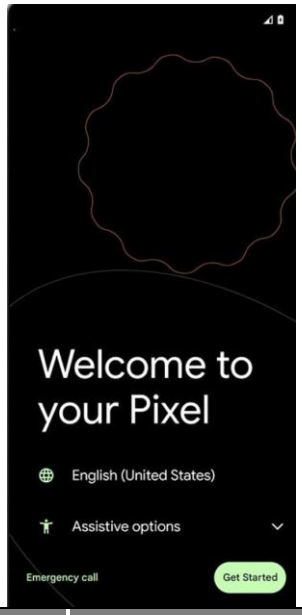
The directory, /data/misc/bootstat/, should exist on multiple manufactured Android devices (to include the Samsung and Pixel) and is a relatively new artifact available in Android version 11+. The directory contains the files **factory_reset** as well as **last_boot_time_utc**. The modified timestamps for those files are reflective of when the activity occurred, even though the files themselves are empty.

The next two artifacts are relevant to Samsung devices only
/efs/recovery/history will log wipe events for every wipe that occurred on a device after the "–-wipe_data" identifier along with a corresponding timestamp. This file appears to track multiple wipes and may vary slightly depending on whether the wipe was conducted via the phone UI or by entering the system recovery.

And finally, **/data/system/users/service/data/eRR.p**, which is also unique to Samsung, tracks device wipes and subsequent reasons (ex: recovery, userrequested, adb) The timestamps are stored in UTC based on the device time bias.

References:

- https://thebinaryhick.blog/2021/08/19/wipeout-detecting-android-factory-resets/
- https://github.com/RealityNet/android_triage



Per Josh Hickman's blog post, "Wipeout-Detecting Android Factory Resets", the file referenced below, specific to Samsung Android devices, will record the last approximate wipe time for a particular device. The artifact that is being tracked in this

particular case, is the user response to the Samsung EULA, which will appear following the Samsung welcome screen.

/data/data/com.sec.android.app.setupwizardlegalprovider/databases/swlpdb.db

It is also relevant to pay close attention to messages generated by the carrier or the manufacturer. Prepaid SIM cards may generate SMS or email messages when money is applied to the account, and some manufacturers send welcome messages via email or SMS.

Reference:

- https://thebinaryhick.blog/2021/08/19/wipeout-detecting-android-factory-resets/

- Timestamp associated with xml tag indicates wipe time
- com.google.android.setupwizard XML tag (Pixel)
- com.sec.android.SecSetupWizard XML tag (Samsung)

- /data/data/com.google.android.apps.wellbeing/databases/app_usage (Pixel)
- /data/data/com.samsung.android.forest/databases/dwbCommon.db (Samsung)
- **DEVICE_STARTUP** indicator is followed shortly by **ACTIVITY** associated with com.google.android.pixel.setupwizard (Pixel) or com.google.android.setupwizard (Samsung)

- **date_added** colum references system files that get created upon start up
- /data/data/com.android.providers.media/databases (Pixel)
- /data/user/%USER%/com.google.android.providers.media.module/database (Pixel)
- /data/user/%USER%/com.android.providers.media.module/databases (Samsung)

Understanding how files are created on a system after a wipe will also provide a close correlation with the exact time of a wipe. For example, many of the files listed above will contain valuable information.

The appops.xml file located at /data/system/ will include xml tags along with a unix epoch timestamp closely tied to wiping activities.
The XML tags below are specific to the device manufacturer: com.google.android.setupwizard - XML tag

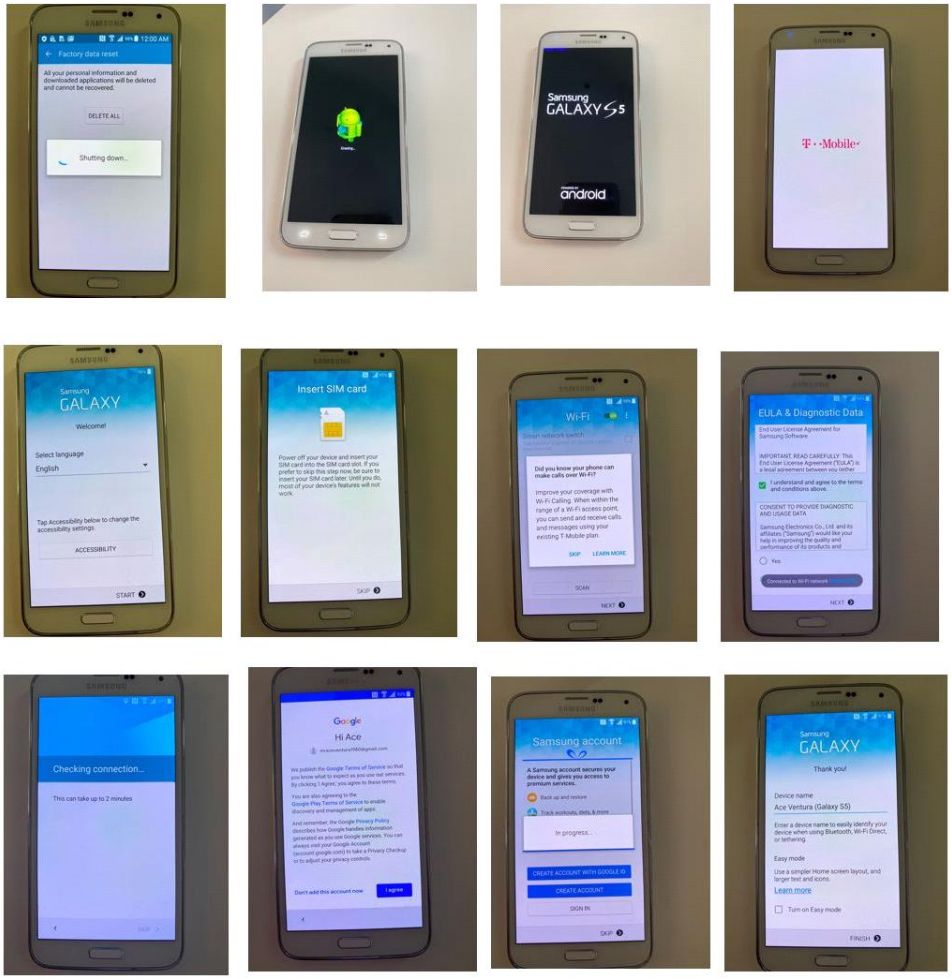
(Pixel) com.sec.android.SecSetupWizard - XML tag
(Samsung)

Digital Wellbeing is an artifact that presents on modern Android devices (Android 10 +). The database, which logs events along with their associated Android package, contains DEVICE_SHUTDOWN and DEVICE_STARTUP events tied to the "android" package. Shortly thereafter, activity related to the "setupwizard" package can be found on freshly wiped and restored devices. This artifact varies slightly per manufacturer.

Path to database for Digital Wellbeing:
/data/data/com.google.android.apps.wellbeing/databases/app_usage (Pixel)
/data/data/com.samsung.android.forest/databases/dwbCommon.db (Samsung)

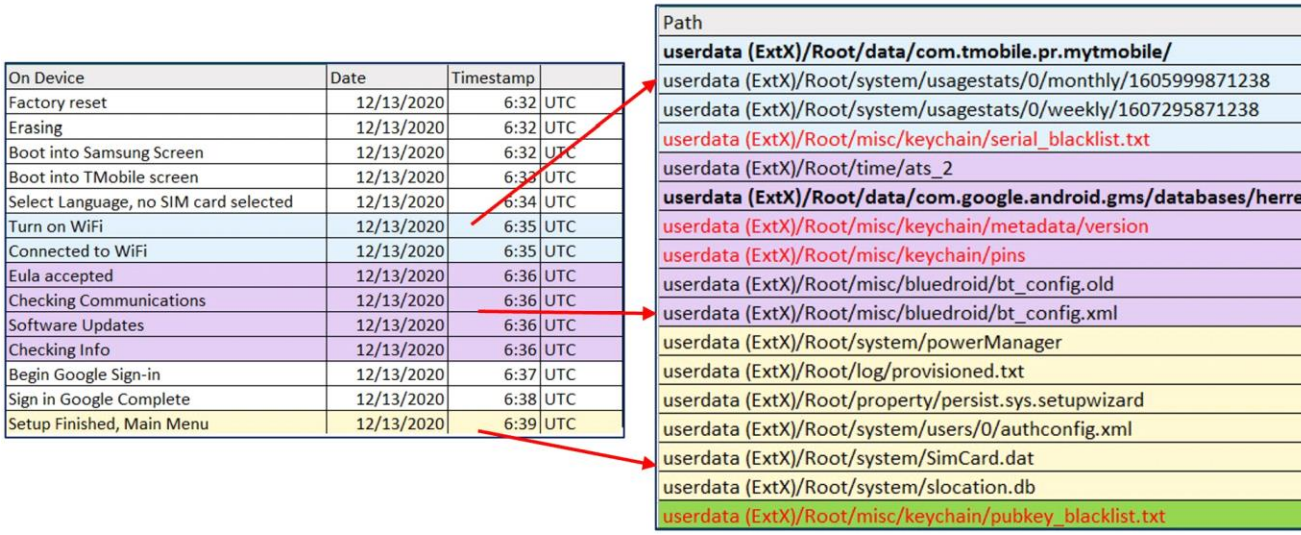
Android setupwizard
packages:
com.google.android.pixel
.setupwizard (Pixel)
com.google.android.setu
pwizard (Samsung)
com.sec.android.app.Sec
SetupWizard (Samsung)

Sample timeline of events



You may recognize Android's internal.db, the database that tracks internal, on-device files, by its similarly sounding name and external storage tracking equivalent, external.db. Using this database, only available on FFS acquisitions, you will notice many of the system files that created after a device boots into the operating system for the first time. Again, the artifacts will differ slight per manufacturer but the date_added column timestamp can be closely correlated to a factory reset of an Android device.

Path to internal.db:
/data/data/com.android.providers.media/databases (Pixel)
/data/user/%USER%/com.google.android.providers.media.module/database (Pixel)
/data/user/%USER%/com.android.providers.media.module/databases (Samsung)



On Device	Date	Timestamp	
Factory reset	12/13/2020	6:32	UTC
Erasing	12/13/2020	6:32	UTC
Boot into Samsung Screen	12/13/2020	6:32	UTC
Boot into TMobile screen	12/13/2020	6:33	UTC
Select Language, no SIM card selected	12/13/2020	6:34	UTC
Turn on WiFi	12/13/2020	6:35	UTC
Connected to WiFi	12/13/2020	6:35	UTC
Eula accepted	12/13/2020	6:36	UTC
Checking Communications	12/13/2020	6:36	UTC
Software Updates	12/13/2020	6:36	UTC
Checking Info	12/13/2020	6:36	UTC
Begin Google Sign-in	12/13/2020	6:37	UTC
Sign in Google Complete	12/13/2020	6:38	UTC
Setup Finished, Main Menu	12/13/2020	6:39	UTC

In fact, there are many actions carried out following a wipe and reboot process will can be closely tracked to their corresponding artifacts. To illustrate the process of wiping/factory resetting a device, several make/model/firmware versions were utilized and reviewed to find the best collective evidence that would suggest a wipe.

The artifacts that proved to be the most consistent across devices were those located at the paths listed below:

- userdata/misc/keychain/serial_blacklist.txt
- userdata/misc/keychain/metadata/version
- userdata/misc/keychain/pins
- userdata/misc/keychain/pubkey_blacklist.txt

The creation dates of most files will be reflective of steps in the reboot process occurring after the factory reset/device wipe. In some cases, the creation date is not accurate (i.e., 1/1/1970, and, therefore, the modification date should be used).

The images in this slide reflect the on-screen prompts that occur during reboot after reset and the corresponding files that have timestamps consistent with when the artifact was created.

It's also important to note that depending on the user's selections, different artifacts could be created. For example, in this case, the user connected to a network as part of the initial setup process, which generated the artifact seen in the herrevad.db.

Lastly, don't overlook manufacturer or service provider artifacts, like in this case, many of the service provider artifacts (T-Mobile) correlate with the time of reboot after factory reset.

Reference:

[1] https://thebinaryhick.blog/2021/08/19/wipeout-detecting-android-factory-resets/

Consider your user:

- Did they log back into the device with a Gmail account?
- Many Google artifacts will be recreated with dates that precede the wipe.

Unfortunately, Android devices are quite varied, which leads to the complexity and the number of files that could pinpoint the time of wipe. But let's assume, similar to our iOS example, that our user was savvy enough to try to recreate artifacts on the system to disguise evidence of a wipe. Using third-party activity and correlating those artifacts with system activity, is still possible.

In the example above, the file created in the usagestats directory at **data/system/usagestats/0/weekly/############** has a created date that closely corresponds with the time of wipe/restore; however, when our user logged back into the device using a Gmail account, many Google artifacts from dates prior to the wipe were recreated on the device.

What is nice about the usagestats directory is that applications that have been deleted will not have their content removed from the usagestats activity. This is a great place to start investigating not only the presence, but also the usage of certain applications.

Android Usagestats used to be an artifact that wasn't parsed by commercial forensic tools, but thanks to the work that is being done in the research community, we have seen that it is now an artifact that gets much deserved focus and is now part of "Analyzed Evidence" or "Artifacts". This area of research was tackled early on by former student and forensic examiner/researcher, Alexis Brignoni in his Android-Usagestats-XML- Protobuf parser. This stand-alone tool has also been incorporated into his Android Log Events parser, ALEAPP.

GitHub - abrignoni/ALEAPP: Android Logs Events And Protobuf Parser.

The activity was stored in .xml format up through Android Pie (9.x), and in later firmware, (10+), it is now stored as a Protobuf. [1]

One thing you may notice when looking at the Usagestats files in both Oxygen (left) and Cellebrite (right) is that the timestamps don't appear to be in a recognized format. In his script, Alexis accounts for these timestamps.
The Android-Usagestats-XML-Protobuf parser is available in GitHub: GitHub - abrignoni/Android-Usagestats- XML-Protobuf: Android Usagestats XML + Protobuf Parser, and there is a post specific to his research on Android Usagestats, as well as many other topics, maintained on his personal blog: https://abrignoni.blogspot.com.

Reference:

https://github.com/abrignoni/Android-Usagestats-XML-Protobuf

(1580601600000 + 30987200) / 1000 = 1,580,632,587.2

As you can see, the results of the Android-Usagestats-XML-Protobuf parser accurately accounted for the strange looking timestamps, so what exactly is going on behind the scenes?

If you look through the different files in each of the respective sub-directories for yearly, monthly, weekly, and daily activity, you likely noticed that some timestamps don't seem to fit into any of our previously addressed timestamp conversions and some timestamps are negative.

Each file in the usagestats sub-directories has a name that is reflective of the file's creation date in Unix Epoch (ms). Application usage that occurs after the file's creation date will have a positive (but unrecognizable time last time active format). This is because the timestamps are offsets from the creation date of the file. To correctly calculate the last time active, you must add the time offset associated with a particular application to the time recorded in the filename. From there, you can apply the normal Unix Epoch (ms) timestamp conversion: **(time/1000,'unixepoch','localtime')**.

Negative timestamps will occur because the application activity precedes the file's creation date. In these cases, the full, accurate timestamp is recorded in the file, but it is logged as negative. In these cases, taking the absolute value of the timestamp before applying the Unix Epoch (ms) timestamp conversion results in the correct timestamp.

The script provides a database file as well as an html report file for distribution.

To run the script, copy out the entire usagestats folder (or just your files of choice) to a folder on your computer. Place the usagestats_conv.py file in the same directory as your usagestats folder (example below).

Run the command (example below).

Html output

Generate keyword lists from files that have been parsed. Make sure the keyword search term is unique to avoid sorting through too many false positives.

When attempting a keyword search of the entire device, the keyword search should be initiated from the device physical image (in Cellebrite Physical Analyzer) if present. When using Oxygen Forensic Detective, the “search within file content” selection should be chosen. Both Autopsy and Axiom allow for keyword lists to be added prior to and after image parsing is complete and also support individual keyword searching capability.

Search for keywords that conform to certain patterns by conducting a grep search across the entire device or a file or database of interest. In the example on the right, the entire physical image of the device is being searched for a Regular Expression that meets the following criteria:

any number, letter, or special character contained in the first set of brackets **PLUS** the **@** symbol, followed by any number, letter, or special character contained in the second set of brackets, followed by a period **[.]** and ending in 3 letters (uppercase or lowercase).
[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[A-Za-z]{3}

In the example above, physical searches for a unique search term were conducted over two different mobile device image files (*.bin). In the example on the left, the unique term appeared multiple times throughout the physical device but only existed two times within logical files on the file system, as evidenced by the "Source" category within Cellebrite Physical Analyzer's interface. Like we learned in Section 1, this is an example of wear leveling, where the operating system efficiently moved that entire database to various physical pages on the NAND flash to prevent wearing out the chip.

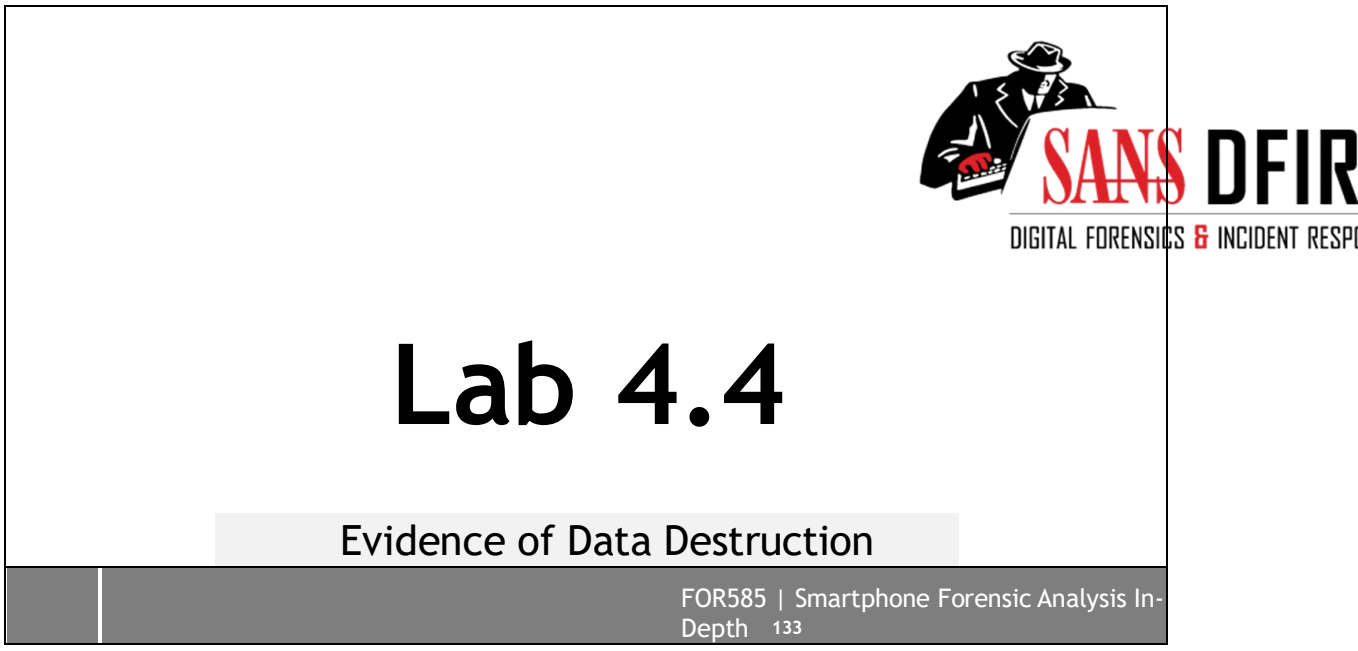
In the example on the right, another keyword search was conducted for a unique term over the entire physical memory. Each of the search hits were tied to a logical file on the file system, which in this case was a database file or the temporary file (WAL) for that database. If this same search was conducted on just the database in question, **mailstore.kimmiecheers1998@gmail.com.db** HYPERLINK "mailto:mailstore.kimmiecheers1998@gmail.com.db", it would appear that there is no "source" for the search hits. This is NOT an example of wear leveling due to the fact that the search was conducted on a single database file and not the physical memory.

Utilize forensic utilities to mount the files system and present files of interest. In the event that none of the files in your file system are parsed, you may have to manually carve for those items that you expect to find on the device. This is where you should become familiar with the types of files and their respective file headers on each of these devices.

Tools like Cellebrite Physical Analyzer and Magnet's Axiom will do a very good job of carving a mobile device acquisition for image files; however, depending on device support, you may have to separately carve for additional items of interest. Depending on the tool you are using, you may want to initiate additional carves for HEIC/HEIF files and KTX files from iOS devices.

Because the bulk of user data on smartphones is stored in SQLite files, you should be very familiar with the 16- byte SQLite format 3 database header of **53 51 4c 69 74 65 20 66 6f 72 6d 61 74 20 33 00**.

Not all devices utilize SQLite format 3. Some older operating systems utilize proprietary or non-SQLite databases, so keep this in mind for carving. Carving for known byte signatures across the device image will help to isolate files of possible interest. It is important that you know what type of files are most commonly used on the operating system you are analyzing in order to make sure you have recovered all possible data.



This page intentionally left blank.