POLITECNICO DI TORINO

Master's Degree course in Computer Engineering

Master's Degree Thesis

Forensic Analysis of Mobile Spyware

Investigating Security, Vulnerabilities, and Detection Challenges in Android and iOS Platforms

Supervisors
prof. Andrea Atzeni
prof. Paolo Dal Checco

Candidate
Andrea Caruso

October 2024

Per Giulia, per tutte.

Summary

In law enforcement investigations, the role of digital forensics is extremely important. Because of the technological advancement which we experimented over the last two decades, smartphones have become a fundamental part of everyday life. On one hand, this phenomenon has brought some advantages to our society but, on the other, smartphones (just like any other form of technology) could be used to perform malicious actions. Unfortunately, in law enforcement investigations are not rare the cases where smartphones have been used to spy on someone, e.g., consorts spying on each other when they fear cheating, company administrators spying on employees to make sure they do not misbehave, or even cases where attackers install spyware on mobile phones of company employees to exfiltrate sensible data and then sell them to third parties. In all these cases, the role of digital forensics is to identify, acquire, process, and analyse electronically stored data which could be used as a legally valid proof in a court of law. Data must be acquired and processed in a way which guarantees their integrity, otherwise the proof would be invalid. However, applications which are installed on mobile devices to conduct spying may be able to alter or even destroy such data. The goal of this thesis is to study spyware targeting mobile devices and to understand how they interact with the target system.

Both Android and iOS operating systems have been considered, however iOS research has been more difficult to carry out because of the lack of up-to-date resources both online and in the literature. Research papers focusing on Android spyware features only consider OS versions up to Android 12 but they are far more detailed than iOS spyware-related papers. Android spyware are described in depth by explaining which Android APIs are used for compromising the target system, without exploiting any vulnerability. These details are not given when talking about iOS spyware and, in most cases, research focuses on spyware that exploit system vulnerabilities by just mentioning the exploited CVEs. Articles regarding iPhone jailbreak, required to install applications from third party stores (e.g., spy apps), do not consider iOS versions higher than 14.8.1 because jailbreak is not possible on newer iOS versions.

This thesis starts by providing some background on Android and iOS platform architectures. Android and iOS software stacks are described along with mobile applications file formats. Understanding how mobile applications are structured helps comprehending how they interact with the system, which is the ultimate goal of the thesis, since spyware will be discussed. However, iOS Package Apps (IPAs) description is far less detailed than the one for Android Packages (APKs). APK and IPA files are both compressed archives but, while APKs compression is specifically designed for a minimum memory footprint, IPA files are regular ZIP archives with a different extension. APK files embed an XML file called “manifest”, which contains all the information regarding how the app is launched, which permissions it requests, and which services it provides. On the other hand, IPA files may contain a provisioning profile as a property list file (`.plist` extension) which lists the deployment permissions of the applications. Unless the iOS app is published to the App Store, it must be equipped with a provisioning profile obtained from the official Apple’s developer console (that, in a sense, proves that Apple “authorised” the application to request such permissions).

Then, a detailed description of both Android and iOS platforms security is provided. Application sandboxing, storage encryption, backups security, mobile device management, updates and security patches are topics which are discussed in depth. iOS security relies a lot on hardware security and multiple key hierarchies are used to enforce data integrity and confidentiality. As Android security is more software-oriented, iOS can be considered more secure than Android. When an Android application is installed, a new unique UID (Linux user ID) is created for it, all of its code runs with the same permissions of that user and the app is assigned its private area of the file system. iOS implements runtime security by combining sandboxing and address space layout randomization (ASLR). Upon installation, an app is assigned to a unique and random home directory where to store its files. Third-party apps must use iOS services to access data of other apps, as well as system files. Most iOS system files and resources run as the "mobile" user (which is unprivileged) just like all third-party apps, and the entire OS partition is mounted as read-only. Android employs file-based encryption, which allows to encrypt different files with different keys, and metadata encryption, which encrypts everything else with a single key present at boot time. iOS systems adopt an encryption technique which is called "data protection". Data Protection uses a hardware-enforced key hierarchy and assigns a class to each stored file. The file can only be decrypted if the corresponding class keys are unlocked. The Apple File System even allows different portions of the same file to be encrypted with different keys. System backups could be used to collect forensically relevant data during investigations, but they're usually stored on cloud (Google cloud or iCloud) and properly encrypted. MDM profiles allow to remotely manage mobile devices and it's interesting to understand what actions could be triggered by installing an MDM profile on Android and iOS systems as, in some cases, they could lead to data alteration and destruction. Finally, updates and security patches are important to discuss as different companies deliver them with different cadence, and they're not always available for older devices. However, devices which are not kept up-to-date incur into serious security risks because of the increasing number of vulnerabilities which are discovered over time.

Google Play Store (for Android) and App Store (for iOS) are described as well, with a major focus on their security measures. Both stores adopt a vetting system to review new applications which are submitted by developers. Anti malware solutions like Google Play Protect are used to analyse submitted apps and machine learning models are employed to classify applications as safe or malicious. In some cases, human operators may review applications as well. If an app passes all security checks, it can finally be published on the store. It has been proven, in the literature, that it's possible to publish malicious applications on the Google Play Store by using a combination of techniques like code obfuscation, encryption, incremental malicious updates, repackaged apps, and others. However, there are no corresponding research papers that demonstrate such a thing is possible on the App Store too.

After having understood what security measures are adopted by Android and iOS, attacks targeting both operating systems have been described. The major categories of attacks, i.e., hardware-based, kernel-based, application-based, and others have been explained in depth for Android and some possible countermeasures (beside vulnerabilities patching) have been presented. Attacks on iOS have been explained by diving into some of the most famous and dangerous spyware which have ever been created. The operation triangulation attack, as well as Predator and Pegasus spyware attack chains have been analysed, emphasizing how they can compromise target systems with a series of 0-day (and also 0-click, when talking about Pegasus) exploits. Moreover, iOS jailbreak is described and its forensics implications are presented.

The remaining chapters of the thesis focus on mobile spyware. Principles and features of a spyware have been presented, in general. Discussion on Android spyware includes details regarding what permissions spyware request and how they actively use them to spy on the target system. The bind accessibility service permission is by far the most dangerous permission which an Android application could be granted. The accessibility service was originally designed to help users affected by some form of disability and it exposes APIs which allow to have full control of the device's screen. Some malware developers use the accessibility service to simulate fake touches on the screen and self-grant all other permissions which are requested by an app (e.g., access to camera, microphone, location, etc...). If an Android app is also given device admin privileges, it could potentially prevent the user from performing the uninstallation procedure. Unfortunately, it has been impossible to find resources giving technical details, like those for Android spyware, for iOS spyware too. After this overview on mobile spyware, some spy apps which are available either on official stores or on third party stores have been described and tested on Android 14 and iOS 17.6.1, which are the latest available versions of Android and iOS. Clearly, since no jailbreak is available for iOS 17, iOS spy apps could only be retrieved from the App Store.

Tested applications are publicized as parental control or anti-theft apps and they're totally legal. However, such applications offer features which make them behave similarly to real spyware. The most interesting study cases have been the third party Android apps named "Cerberus Antitheft" and "MobileTracker free", which both offer features like full control of the device's peripherals, SMS remote commands, SD card data wipe, and even the possibility to remotely trigger a device factory reset. These applications have also been proven to be stealthy and persistent, up to some extent. Cerberus was able to keep working as intended after a device reboot, without the need to unlock the device by inserting the password. MobileTracker free implemented a protection mechanism to prevent uninstallation, even though it's not perfect. On the other hand, iOS applications have exhibited far less features, like geolocation tracking and web browser search history logging, but nothing more. Installation procedures for Android and iOS spy apps greatly differ. The only common factor is that physical access to the target device is required. Third party Android spy apps require to enable the "allow installation from unknown sources" setting and also require to disable the Play Protect security measure, which would otherwise detect them and remove them from the device. Android spy apps which are downloaded from the Play Store implement far less features than their third party variants and do not need any security measure to be disabled, as the won't be detected as malicious by Play Protect. iOS spy apps require the installation of an MDM profile on the target device, which will allow to send remote commands from an MDM server to the app. Moreover, uninstallation can only be blocked after connecting the iPhone to a computer where the app's anti-tampering software has been installed and executed. Information like SMS messages and call register logging can been retrieved with ease by Android apps but require to perform a device backup over Wi-Fi on iOS.

Since tested apps always require physical access to the target mobile device, a chapter of the thesis was dedicated to the design of an attack which could lead to successful installation and configuration of an Android malicious app. The attack requires the app to be published on the Play Store and to avoid malware detection, as well as the end victim to be tricked into granting the bind accessibility service permission to the app. All other permissions will be self-granted.

Finally, some digital forensics tools have been presented and spy apps have been tested against them. The SpyGuard network analysis tool is the one which has performed worst, detecting only 2 apps out of 6. On the other hand, the Mobile Verification Toolkit (MVT) was able to detect 4 apps out of 6 by performing a device backup and by analysing it with the help of indicators of compromise (IOC). MVT was also able to extract system logs from the target Android device, which helped in detecting malicious apps even when the tool did not flag them as dangerous. Further testing has been performed with MVT after having all the spy apps uninstalled. This phase of the research truly highlighted how difficult it is to detect spyware on iOS, due to the lack of information in the decrypted backup files and the impossibility to perform a full system dump without a jailbroken device. On the other hand, detection on Android is still possible, up to some extent, thanks to a full system dump.

Acknowledgements

I would like to thank professor Atzeni and professor Dal Checco for their precious guidance during the whole thesis work. Their contagious passion for what they teach, their kindness and their care for students are all remarkable qualities which are not easily found in teachers nowadays. If I had to write the thesis over again I would still choose you as supervisors another thousand times. I really had fun writing this thesis and I am happy I've learnt a lot of new things which I think will be useful to me in the future, during my career. Thanks a lot for this experience!

I thank my family for all the support they've been giving me, during university but also life in general, both moral and economic. You taught me how to face problems and it's thanks to you if I am the person that I am today. Thank you so much, mum and dad! I love you!

Last but not least, I thank all the friends which I've shared good and bad moments with. Thanks to all the friends I've met in Turin, who made university the best experience I could ever ask for. I never really liked going to school, not because I didn't like studying but because I always lacked human relationships. During both middle school and high school I've never met anyone which I considered as a true friend. Thank you so much for all the evenings we spent together, either going out for a walk, or having dinner, or playing cards and board games. Thank you for all the conversations we had. We could literally talk about anything for hours and I would never get bored. Thanks to all the friends I met in campus, especially to my ex room mate Francesco P. who is genuinely one of the best people I've ever known. Thanks to Emanuela, Donato, Francesco N., Edoardo, Stefano, Silvano, and Simone, who I've spent most time with, either at university or outside it. You guys really made my days here in Turin, thank you so much. Thanks to Alessia and Alessandro who I've shared most of the dinners at McDonald's with. Eating junk food with you guys, is just different. You are both incredibly nice people and I look forward to hang out with you more often and to have funny conversations like the ones we've had so far. I thank Elisa, for being one of the sweetest people I've ever met, one who would do anything to help a friend. Thanks to Mattia, who is one of the first people I met at Politecnico. Despite us not hanging out that often, I consider you as a good friend and I hope we won't lose each other in the future. Same thought goes to Grazia, which I've had the pleasure to share the "suffering" caused by the IoT Python project with. Thanks to Connor, you are definitely the coolest guy ever. Thank you for all the sparkling conversations we had on the roof of the campus. I look forward to the day we will meet again amico. Thanks to Alice, I really wish we could share more memories of Turin. I miss the old days where we all used to spend evenings together at campus. Thanks to all the friends from Palermo too, in particular Dario and Gabriele, who I have known since I was a child. Thanks to them and to Alessandro C. for all the times we played League of Legends and had fun together (jungle diff). Thanks to Alessandro V., forever our captain during explorations at Monte Pellegrino (and Monte Gallo too...). Thanks to Alberto, Andrea R., Costanza, Federica, Carlotta, Vito, Domenico, Tore, and many others. Thanks to every single one of you.

With love, Andrea.

Contents

Chapter 1

Introduction

Digital forensics is a branch of forensic science that focuses on identifying, acquiring, processing, analysing, and reporting on data stored electronically. Electronic evidence is a component of almost all criminal activities and digital forensics support is crucial for law enforcement investigations. One of the most critical aspects of an investigation is the integrity of proofs. The goal of this research is to study spyware that targets mobile devices, in order to understand how it can infiltrate target systems, modify them, and consequently alter or even destroy forensically-relevant data. We will focus on spyware that targets both Android and iOS operating systems.

Android is an open source Linux-based OS developed by Google while iOS is a proprietary OS developed by Apple. As of 2024, according to [1], Android is the most prevalent mobile platform among different smartphone platforms with a market share of $\sim 72\%$, immediately followed by iOS with a market share of $\sim 28\%$. As both platforms continue to evolve, new threats arise and new vulnerabilities are constantly being discovered and reported, making it difficult for researchers to fix them as quickly as they are discovered. A large number of Android app markets contain vulnerable and malicious apps, thereby compromising millions of mobile devices. Despite it being more secure than Android, iOS is not exempt from risks as it has already been successfully attacked multiple times over the years. Due to these factors, assessing Android and iOS security is an important task which will be addressed in the first chapters of this document.

First, in Chapter 2 we will discuss both Android and iOS platform architectures. Due to the lack of iOS technical documentation regarding its architecture, iOS discussion will not be as detailed. On the other hand, when discussing Android's architecture we will have a major focus on processes creation, IPC, and Android packages (APKs). This will be useful to understand in depth how Android spyware is structured and how it interacts with the system.

In Chapters 3 and 4 we will describe the security measures adopted, respectively, by Android and iOS. iOS security mainly relies on hardware while Android security is more software-oriented. This should already give a hint on why iOS is more secure than Android. We will discuss topics like application sandboxing and filesystem permissions as well as backups and cloud security, for both platforms. We will also go over the Mobile Device Management (MDM) solutions adopted by Android and iOS, and how they may affect the integrity of forensic proofs.

In Chapters 5 and 6 we will discuss the importance of Google Play and App stores, as they are the official application stores for, respectively, Android and iOS devices. The vetting process employed by both stores, before allowing new apps to be published, will be discussed. Despite this system being very rigorous, sometimes attackers are still able to trick it for publishing and distributing malicious applications.

Then, in Chapters 7 and 8, we will discuss the major categories of attacks which can be launched against Android and we will describe some of the most powerful spyware which compromise iOS devices by using 0-day exploits. We will list some of the most critical vulnerabilities which have been discovered over the years, both in Android and iOS, and how they made the systems insecure, despite the adoption of proper security measures. Although these vulnerabilities have already been patched, they still represent an interesting study case, especially because not all users comprehend the importance of security updates and also because older devices may not even support such updates. In Chapter 13, the design of a possible attack which could lead to spyware installation and configuration on a target system, will also be presented and discussed.

Finally, from Chapter 9 to 14, we will describe more in depth Android spyware, focusing on how they interact with the system on a lower level. We will then provide some examples of spy apps which are available online (either for free or after buying the license). The testing phase of some of these apps on Android 14 and iOS 17.6.1 will be described as well, trying to highlight on a higher level what features they expose and which app is more similar to a real spyware compared to the others. The same apps will be tested against some digital forensics tools, publicly available online, which employ heuristic methods and public Indicators Of Compromise (IOC) to detect traces of spyware on mobile devices, either with network flows analysis or with backups analysis.

Chapter 2

Background

2.1 Mobile spyware

A **spyware** is a malware which aims at spying on users by collecting their personal and sensitive information, as well as data regarding their devices and how they use them in everyday life. Attackers may use a spyware to collect and sell user data or to steal user credentials and spoof their identities. Spyware leaves businesses vulnerable to data breaches and data misuse, often affects device and network performance, and slows down user activity.

Spyware targeting mobile devices steals data such as:

- call logs, including phone numbers, date, time, and duration of each call;
- contacts list;
- photos, videos and audio recordings stored on the device;
- Short Message Service (SMS) messages;

More in general, a spyware may also be able to:

- steal user credentials for banking accounts;
- log browser history, including URLs, and date-time of when a website has been visited;
- monitor e-mail exchanges and/or social media chats;
- log user keystrokes, which may lead to sensitive data leaks (e.g., user's credentials);
- take pictures and record the surroundings by using the device's microphone and camera;
- track the device's location by using Global Positioning System (GPS) trackers;
- take control of the device through commands sent via SMS messages, data transfers, and remote servers often referred to as **command and control** (**C&C**);

2.2 Android software stack

The Android software stack is made by [2]:

Linux Kernel: It's responsible for low-level services like processes and memory management, threading, and drivers.

Hardware Abstraction Layer (HAL): Allows the Android software to be hardware-agnostic by exposing standard interfaces to the Java API Framework for interacting with hardware devices (e.g., sensors, camera, bluetooth, etc...). Library modules for hardware components are loaded by the system as soon as access to such components is requested with an API call.

Android runtime (ART): It runs virtual machines on low-memory devices. Starting from Android 5.0, each running app has its own ART instance that executes the app's Dalvik bytecode, specifically designed to minimize memory consumption on Android.

Native C/C++ libraries: Many system components and services (e.g., ART and HAL), are built over these libraries. Some of their features are exposed to apps with Java framework APIs.

Java API Framework: Java APIs expose Android features to apps. All apps reuse the same components and services to interact with the system. An app's UI is managed with a view system; non-code resources (e.g., layout files, graphics, etc...) are accessed with resource managers; notifications and alerts are displayed by using the notification manager; the app's lifecycle is managed by the activity manager; data from other apps is accessed by using content providers.

Application Layer: includes both default system's apps and third-party apps.

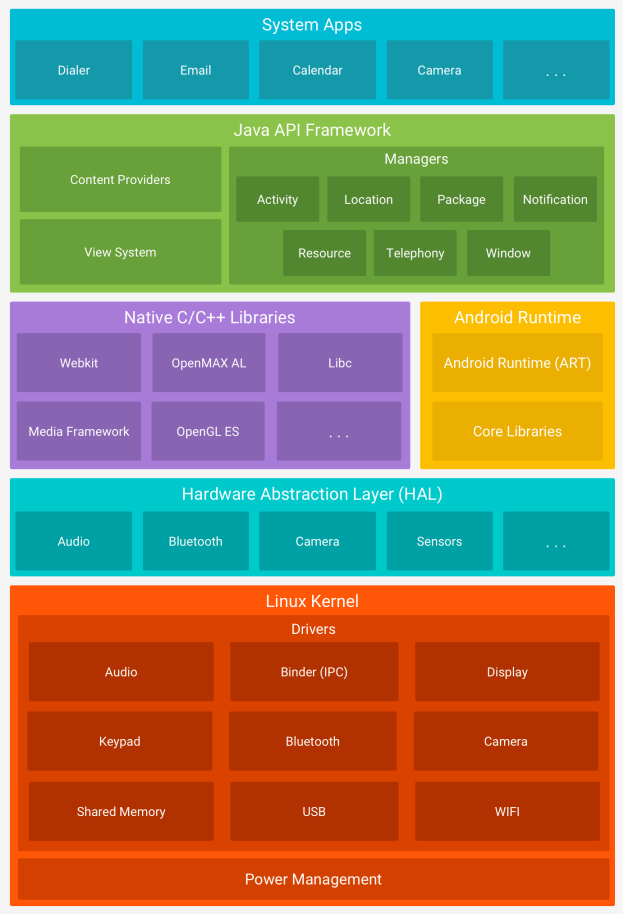


Figure 2.1: Android software stack

2.3 Android Linux Kernel

Android is built over the Linux kernel, with only a few extensions to the kernel itself. However, the user space implementation is quite different from the one of a traditional Linux distribution [3].

2.3.1 Processes

As in a traditional Linux system, Android's first user-space process to run is `init` [3]. The `init` process starts different daemon processes, focused on low-level features (e.g., managing the file system, hardware access, etc...). One of them is the `zygote` process, which is the root of an additional layer of high-level processes. These processes are responsible for executing all parts of the system implemented in Java by running Dalvik's Java language environment.

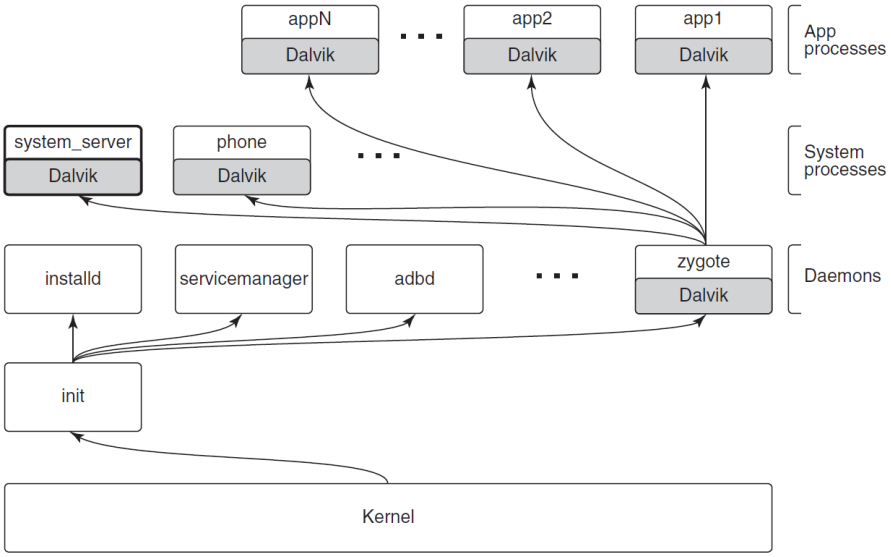


Figure 2.2: Android process hierarchy

Android's `init` process does not run a shell in the traditional way, since a typical Android device does not have a local console for shell access. Instead, the **Android Debug Bridge** daemon process `adbd` listens for remote connections that request shell access, and forks shell processes for them as needed. Android Debug Bridge (ADB) is a tool developed by Android for performing debug and other tasks by connecting an Android device to a computer, either via USB or via Wi-Fi. Once the connection has been established, it is possible to run shell commands from the computer [4]. Some of these commands can be used for installing applications on the device, as well as managing their permissions and uninstall them. Commands don't affect system apps.

The first process `zygote` always starts is called `system_server`, which contains all of the core OS services (e.g., power manager, package manager, window manager, and activity manager). Other processes will be created and stopped from `zygote` as needed. Some of these are "persistent" processes that are part of the basic OS, such as the telephony stack in the phone process, which must always be running. Apps interact with the OS through calls to libraries provided by it, which together compose the **Android framework**. Some of these libraries can perform their work within that process, but many will need to perform inter-process communication (IPC) with other processes, often services in the `system_server` process.

Figure 2.3 shows the typical design for Android framework APIs that interact with system services. In this example, the application process interacts with the **package manager** by using its exposed APIs through the `PackageManager` class. This class connects to the corresponding service in the `system_server` process. This is possible because, at boot time, the `system_server` publishes each available service under a well-defined name in the **service manager** daemon process, which is started by `init`. The `PackageManager` in the application process retrieves a connection from the service manager to its system service by using that same name. Once the `PackageManager` class has connected with its system service, it can call its APIs. The `PackageManagerService` maintains state information that is needed by multiple apps. Most application calls to `PackageManager` are implemented as inter-process communication by using Android's **Binder IPC** mechanism.

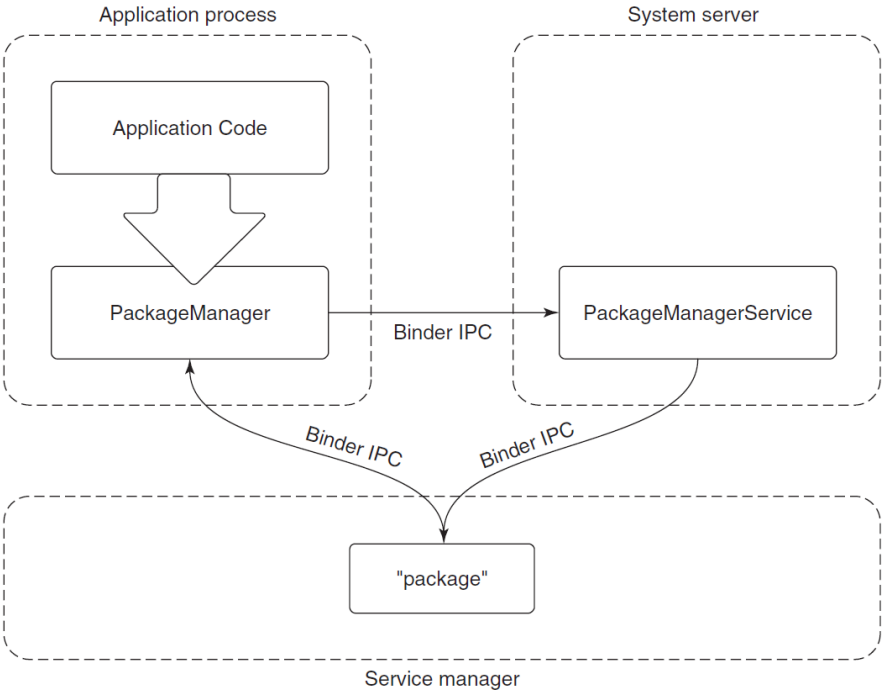


Figure 2.3: Publishing and interacting with system services

2.3.2 Android Linux extensions

For the most part, Android relies on standard Linux features provided by the Kernel. However, there are also several significant extensions to Linux that Android relies on [3].

Wake Locks

Used for managing how the system goes to sleep [3]. While a mobile device's screen is off, the device still needs to be able to receive phone calls, receive and process data for incoming chat messages, and many other things. While the screen is on, the system always holds a wake lock that prevents the device from going to sleep, so it will stay running, as we expect. When the screen is off, however, the system itself does not generally hold a wake lock, so it will stay out of sleep only as long as something else is holding one. When no more wake locks are held, the system goes to sleep, and it can come out of sleep only due to a hardware interrupt.

Out-Of-Memory Killer

Since Android doesn't have a swap space, it's much more common for OOM failures to occur [3]. In both Linux and Android, a "badness" level is assigned to each process based on the amount of RAM used by it. In standard Linux, only when an OOM failure occurs, the "baddest" process gets killed. Android introduces its own OOM Killer, which kills the baddest process whenever RAM is getting "low". Low RAM is identified by a parameter indicating how much available RAM (free and cached) is acceptable. When the system goes below that limit, the out-of-memory killer runs to release RAM from elsewhere. This is to ensure the system never gets into bad paging states, which can negatively impact the user experience when foreground apps are competing for RAM, since their execution becomes much slower due to continual paging in and out.

2.3.3 Dalvik

Dalvik implements the Java language environment on Android that is responsible for running applications and system code [3]. Almost everything in the `system_service` process (e.g., package manager, window manager, activity manager) is implemented with Java language code executed by Dalvik. Java code in an Android application is provided in Dalvik's bytecode format, based around a register machine rather than Java's traditional stack-based bytecode.

Dalvik's bytecode format allows for faster interpretation, while still supporting **JIT** (**Just-in-Time**) compilation. Dalvik bytecode is also more space efficient, both on disk and in RAM, through the use of string pooling and other techniques. Process isolation is enforced by running each application in its own Linux process with its own Dalvik environment.

`zygote` is responsible for initializing Dalvik, so that it gets ready to run system or application code written in Java. All new Dalvik-based processes are forked from `zygote`. Creating a new process from `zygote` involves a Linux `fork`, but there is no `exec` call. The new process is a replica of the original `zygote` process, with all of its pre-initialized state already set up and ready to go. After the `fork`, the new process has its own separate Dalvik environment, though it is sharing all of the pre-loaded and initialized data with `zygote` through **copy-on-write** pages. In order for the new process to be ready, three more steps are required: the process must be assigned to a UID; Dalvik initialization must be completed by starting all necessary threads; the code to be executed must be loaded.

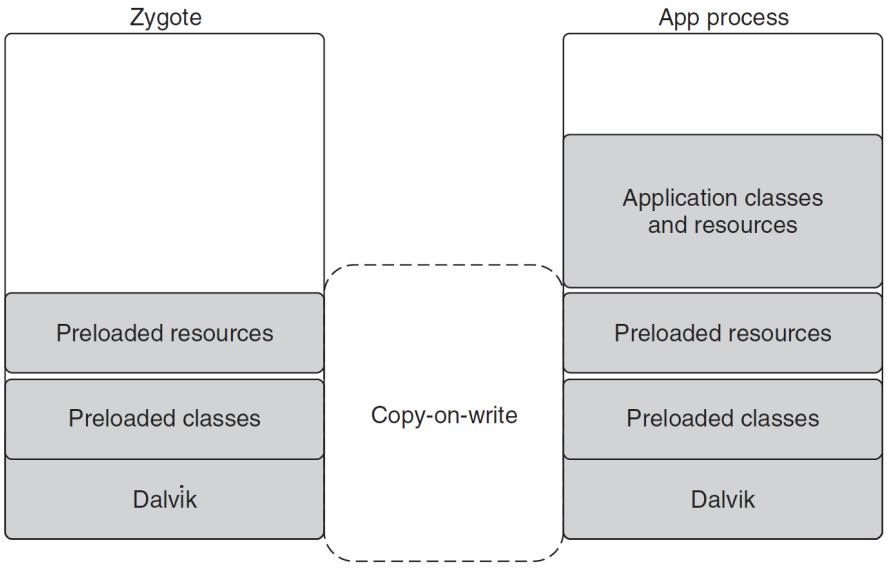


Figure 2.4: Creating a new Dalvik process from `zygote`

2.4 Android Binder IPC

Android's system design enforces process isolation, between applications as well as between different parts of the system itself. Hence, inter-process communication is essential for coordinating different processes. Android's Binder IPC mechanism is a general purpose IPC facility that most of the Android systems is built on top of. The Binder architecture is divided into three layers [3]:

- The lowest layer is made by the **kernel module**, that implements the actual cross-process interaction and exposes it through the kernel's `ioctl` function;
- Then, the **object-oriented user-space API**, allows applications to create and interact with IPC endpoints through the `IBinder` and `Binder` classes;
- The higher layer is made by an **interface-based programming model**, where apps declare their IPC interfaces and can ignore all the low-level implementation details of IPC;

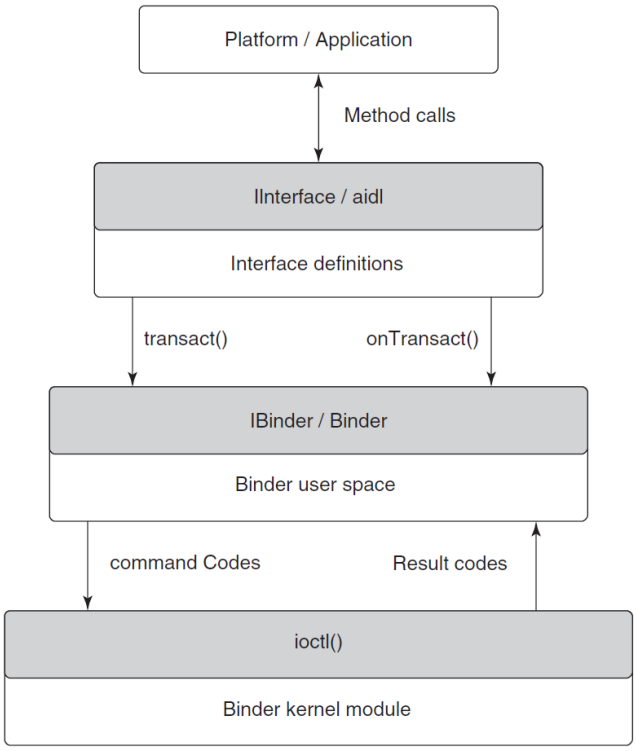


Figure 2.5: Binder IPC architecture

2.4.1 Kernel module

Binder includes a special kernel module that implements its own IPC mechanism, based on **Remote Procedure Calls** (**RPCs**) [3]. In other words, the sending process submits a complete IPC operation to the kernel, which is then executed in the receiving process. The sender may block while the receiver executes, allowing a result to be returned back from the call. Binder IPC is message based. A message in Binder is referred to as a **transaction**, and at a higher level can be viewed as a function call across processes. Each transaction which the user space submits to the kernel is a complete operation, as it identifies: the target of the operation, called **object**; the identity of the sender; the complete data being delivered.

The kernel determines the appropriate process to receive that transaction, delivering it to a waiting thread in that process. For each process that can receive transactions, the kernel creates a shared memory area with it. When it's handling a transaction, it first determines the process that will be receiving it and then copies the data directly into that shared address space. Each process has a thread pool created by the user space to handle incoming transactions. The kernel will dispatch each incoming transaction to a thread currently waiting for work in that process's thread pool. Calls into the kernel from a sending process do not need to come from the thread pool, as any thread in the process is free to initiate a transaction. Transactions given to the kernel identify a target object. However, the kernel must determine the receiving process. To do so, it keeps track of the available objects in each process and maps them to other processes. References to objects in remote processes are identified by an integer **handle**, which is much like a Linux file descriptor.

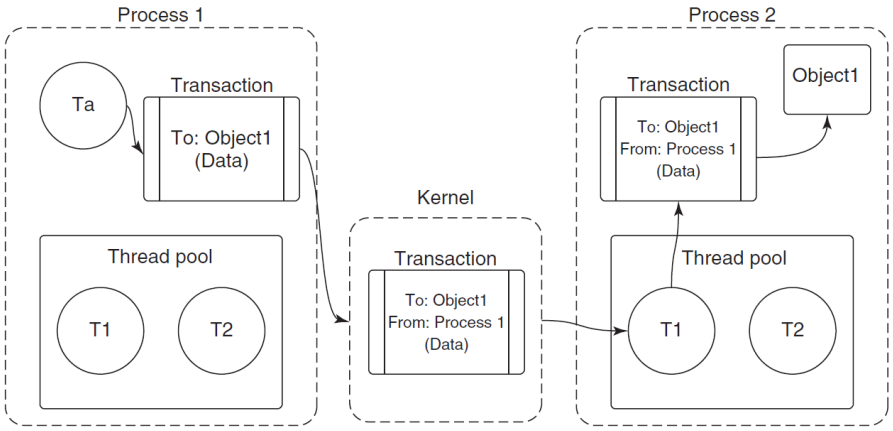


Figure 2.6: Basic Binder IPC transaction

2.4.2 User-space API

Most user-space code doesn't directly interact with the Binder kernel module. Instead, there's a user-space object-oriented library that provides a simpler API in the form of three classes [3]:

- **IBinder**: It's an abstract interface for a `Binder` object. Its key method is `transact`, which submits a transaction to the object. The implementation receiving the transaction may be an object either in the local process or in another process.
- **Binder**: It's a concrete `Binder` object. Its key method is `onTransact`, which receives a transaction that was sent to it. The main responsibility of a `Binder` subclass is to look at the transaction data it receives here and perform the appropriate operation.
- **Parcel**: It's a container for reading and writing data that is in a `Binder` transaction. It has methods for reading and writing typed data but, most importantly, it can read and write references to any `IBinder` object.

2.4.3 Interfaces and AIDL

Instead of dealing with `Binder` objects and `Parcel` data, this layer is made by interfaces and methods, and it makes use of the **Android Interface Definition Language** (**AIDL**) command-line tool [3]. This tool is an interface compiler, taking an abstract description of an interface and generating from it the source code necessary to define that interface and implement the appropriate **marshalling** and **unmarshalling** code needed to make remote calls with it.

```
package com.example

interface IExample {
    void print(String msg);
}
```

Listing 2.1: AIDL interface example

An interface description like the one presented above is compiled by AIDL to generate three Java-language classes:

- `IExample`: Supplies the Java-language interface definition.
- `IExample.Stub`: Performs unmarshalling by turning incoming `onTransact` calls into the appropriate method call of `IExample`.
- `IExample.Proxy`: Performs marshalling by implementing each method of the `IExample` class to transform the call into the appropriate `Parcel` contents and send it off through a `transact` call on an `IBinder` it is communicating with.

The bulk of Android's IPC is written using this mechanism. Most services in Android are defined through AIDL and implemented as shown here:

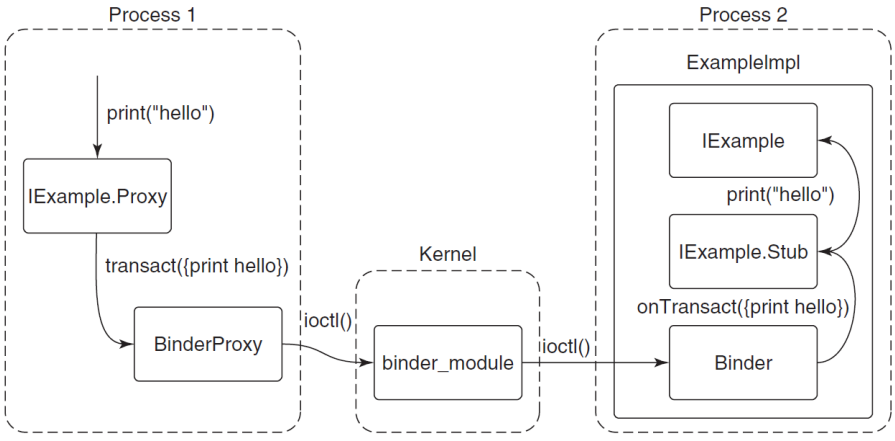


Figure 2.7: Full path of an AIDL-based Binder IPC

2.5 Android applications

2.5.1 Android packages (APK)

In Android, an application is not an executable file with a main entry point but it is a container of everything that makes up that app: its code, graphical resources, declarations about what it is to the system, and other data [3]. An Android application by convention is a file with the `.apk` extension, which stands for **Android Package**. This file is actually a normal zip archive, containing everything about the application. The important contents of an apk are:

- A **manifest** describing what the application is, what it does, and how to run it;
- Resources needed by the application, including strings it displays to the user, XML data for layouts and other descriptions, graphical bitmaps, etc...;
- The code itself, which may be Dalvik bytecode as well as native library code;
- Signing information, securely identifying the author.

Application entry points

Android applications don't have a main entry point which is executed when the user launches them. Instead, they publish under the manifest's <`application`> tag a variety of entry points describing all the things the app can do [3]. Applications can publish pieces of themselves that provide specific functionality, which other applications can make use of either directly or indirectly. An application's entry points are expressed as four distinct component types, defining the core types of behavior that applications can provide:

- **activity**;
- **receiver**;
- **service**;
- **content provider**;

Each of the different four component types an application can contain has different semantics and uses within the system. In all cases, the `android:name` attribute supplies the Java class name of the application code implementing that component, which will be instantiated by the system when needed. Listing 2.2 shows a possible manifest for a simple e-mail application.

2.5.2 Package manager and activity manager

The **package manager** is the part of Android that keeps track of all application packages [3]. It parses every application's manifest, collecting and indexing the information it finds in them. With that information, it then provides facilities for clients to query it about the currently installed applications and retrieve relevant information about them. It is also responsible for installing applications (creating storage space for the application and ensuring the integrity of the apk) as well as everything needed to uninstall (cleaning up everything associated with a previously installed app). Above the package manager sits another important system service, the **activity manager**. While the package manager is responsible for maintaining static information about all installed applications, the activity manager determines when, where, and how those applications should run. Despite its name, it is actually responsible for running all four types of application components and implementing the appropriate behavior for each of them.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="com.example.email">
<application>
    <activity android:name="com.example.email.MailMainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name="com.example.email.ComposeActivity">
        <intent-filter>
            <action android:name="android.intent.action.SEND" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="*/*" />
        </intent-filter>
    </activity>
    <service android:name="com.example.email.SyncService"></service>
    <receiver android:name="com.example.email.SyncControlReceiver">
        <intent-filter>
           <action android:name="android.intent.action.DEVICE_STORAGE_LOW"/>
        </intent-filter>
        <intent-filter>
           <action android:name="android.intent.action.DEVICE_STORAGE_OKAY"/>
        </intent-filter>
    </receiver>
    <provider android:name="com.example.email.EmailProvider"
        android:authorities="com.example.email.provider.email">
    </provider>
</application>
</manifest>
```

Listing 2.2: E-mail app manifest example

2.5.3 Activities

An **activity** is a part of the application that interacts directly with the user through a user interface [3]. When the user launches an application on their device, this is actually an activity inside the application that has been designated as such a main entry point. The application implements code in its activity that is responsible for interacting with the user. If an activity is declared as the main entry point for the application, it will be shown to users as an application they can launch from the main application launcher and it will be started as soon as the user launches the app. When an activity is launched for the first time, the activity manager creates an `ActivityRecord` instance in its process to keep track of the activity. One or more of these activities are organized into containers called **tasks**, which roughly correspond to what the user experiences as an application.

When an activity is currently running, its state is **resumed** If the user switches away from the currently running application A (not exiting it) and launches a different application B:

- a new process is created and it is associated to B;
- A's status becomes **stopped**;
- the `ActivityRecord` associated to A holds A's **saved state**;

If an application starts to require a lot of RAM, the system can simply get rid of its process but the activity manager can still keep track of the corresponding `ActivityRecord` with all the associated info.

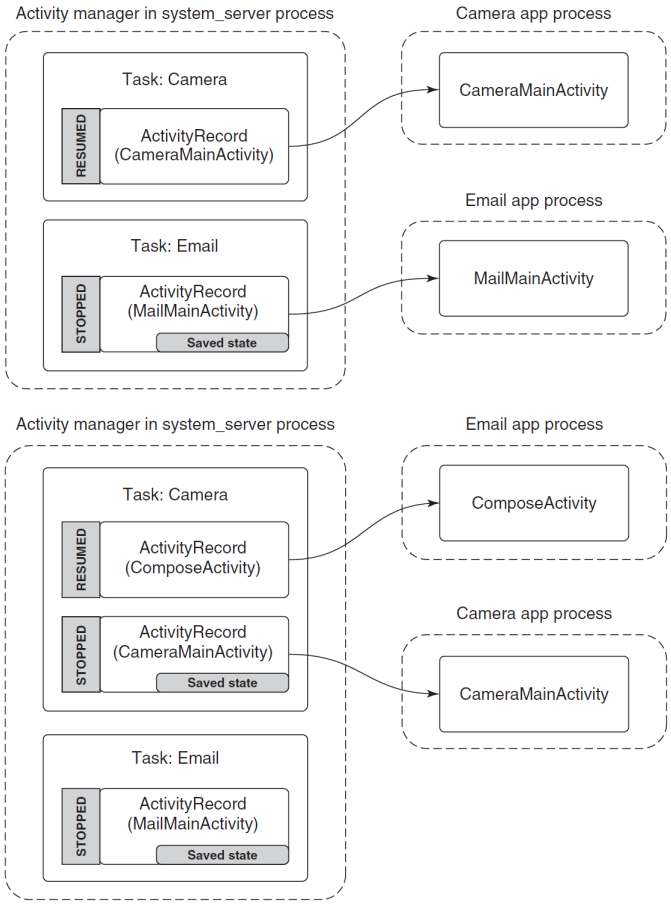


Figure 2.8: Starting the camera application after email and sharing a picture through email application

2.5.4 Services

A **service** has two distinct identities [3]:

- It can be a self-contained long-running background operation, like maintaining an active network connection while the user is in other applications, downloading or uploading data in the background, etc...;
- It can serve as a connection point for other applications or the system to perform rich interaction with the application. This can be used by applications to provide secure APIs for other applications, such as to perform image or audio processing, etc...;

Service as a long-running background operation

When the service is **started**, the activity manager creates a `ServiceRecord` to keep track of it. When the service it's done, it gets **stopped**. It is possible for the application's process to go away while the service is started, such as if the process crashes, but the activity manager will continue to maintain its `ServiceRecord` and can decide to restart the service, if desired [3].

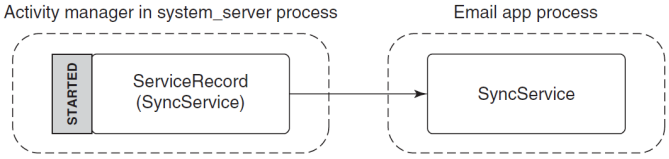


Figure 2.9: Starting an application service

Service as a connection point for other applications

The service must expose an interface (usually defined with AIDL) for an API to be called by other applications [3]. Another process must **bind** to the application service, in order to get access to its interface. This creates a connection between the two applications. To do so:

1. The client application tells the activity manager that it would like to bind to the service;
2. If the service is not already created, the activity manager creates it in the service application's process;
3. The service returns the `IBinder` for its interface back to the activity manager, which now holds that `IBinder` in its `ServiceRecord`;
4. Now that the activity manager has the service `IBinder`, it can be sent back to the original client application;
5. The client application now having the service's `IBinder` may proceed to make any direct calls it would like on its interface;

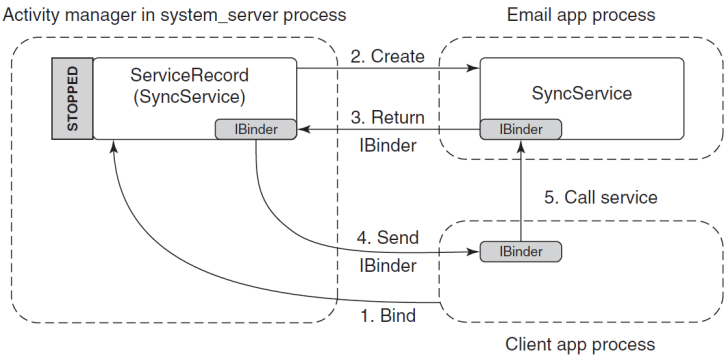


Figure 2.10: Binding to an application service

2.5.5 Receivers

A **receiver** is the recipient of (typically external) events that happen, generally in the background and outside of normal user interaction [3]. Receivers conceptually are the same as an application explicitly registering for a callback when something interesting happens but do not require that the application be running in order to receive the event. When an event occurs (e.g., device's storage or battery becomes low, etc...) the system will send a **broadcast** with the event code, to be delivered to all receivers interested in it. The broadcast is processed by the activity manager in order to deliver it to interested receivers.

The activity manager asks the package manager for a list of all receivers interested in the event, which is placed in a `BroadcastRecord` representing that broadcast. The activity manager will then proceed to step through each entry in the list, having each associated application's process create and execute the appropriate receiver class. Receivers only run as one-shot operations. When an event happens, the system finds any receivers interested in it, delivers that event to them, and once they have consumed the event they are done. A particular receiver is only a transient entity for the duration of a single broadcast. Each time a new broadcast is sent to a receiver component, a new instance of that receiver's class is created.

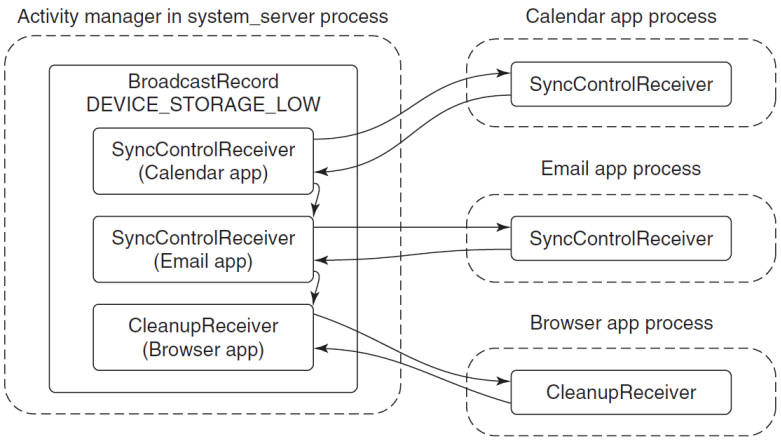


Figure 2.11: Sending a broadcast to application receivers

2.5.6 Content providers

The **content provider** is the primary mechanism that applications use to exchange data with each other [3]. All interactions with a content provider are through URIs using a `content:` scheme. The authority of the URI is used to find the correct content-provider implementation to interact with. In the email app example presented above, the content provider specifies that its authority is `com.example.email.provider.email`. Thus URIs operating on this content provider would start with `content://com.example.email.provider.email/`. The suffix to the URI is interpreted by the provider itself to determine which data within it is being accessed.

To interact with a content provider, applications always go through a system API called `ContentResolver`, where most methods have an initial URI argument indicating the data to operate on. The `ContentResolver.query` method performs a database query on a given URI and returns a `Cursor` for retrieving the structured results. Using a content provider is similar to binding to a service:

1. The application calls `ContentResolver.query` to initiate the operation;
2. The URI's authority is handed to the activity manager for it to find (via the package manager) the appropriate content provider;
3. If the content provider is not already running, it is created;
4. The content provider returns to the activity manager its `IBinder` implementing the system's `IContentProvider` interface;
5. The content provider's `IBinder` is returned to the `ContentResolver`;
6. The content resolver can now complete the initial query operation by calling the appropriate method on the AIDL interface, returning the `Cursor` result;

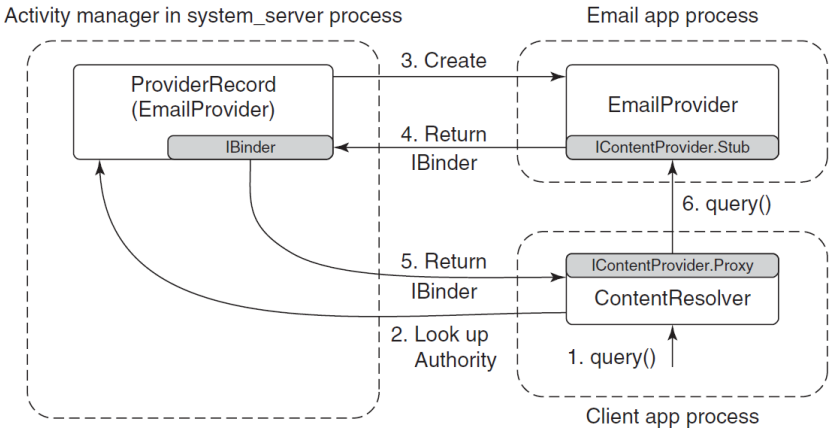


Figure 2.12: Interacting with a content provider

2.5.7 Intents

An **intent** is the mechanism Android uses to discover and identify activities, receivers, and services. There are two major types of intents [3]:

- An **explicit intent** is one that directly identifies a single specific application component by using a pair of strings *package name* + *class name*. From the package name, the package manager can return everything needed about the application, such as where to find its code. From the class name, we know which part of that code to execute;
- An **implicit intent** is one that describes characteristics of the desired component, but not the component itself. The process of finding the component matching an implicit intent is called **intent resolution**, which can have three possible outcomes:
  - No match is found;
  - A single unique match is found;
  - There are multiple activities that can handle the intent (only one can be launched);

Intent structure

The intent structure contains an abstract description of an action to be performed. Each intent contains a pair of [3]:

- **Action**: It's the general action to be performed;
- **Data**: It's the data to operate on, expressed as a URI;

Here are some examples of action/data pairs:

- `ACTION_VIEW content://contacts/people/1`
  - Display information about the person with ID equal to 1.
- `ACTION_EDIT content://contacts/people/1`
  - Edit information about the person with ID equal to 1.
- `ACTION_VIEW content://contacts/people`
  - Display the list of all people.

There are also some **secondary attributes** that can be included with an intent:

- **Category**: Gives additional information about the action to execute.
- **Type**: Specifies the MIME type of the intent data. If this attribute gets set, the data type is not inferred from the data itself but it is explicitly set (may differ from the actual type).
- **Component**: Specifies the name of a component class to be used for the intent. If this attribute is set, all of the others become optional.
- **Extras**: This attribute can be used to provide additional information to the component.

Here are some examples of other operations you can specify as intents using these additional parameters:

- `ACTION_MAIN` with category `CATEGORY_HOME`
  - Launch the home screen.
- `ACTION_GET_CONTENT` with MIME type `vnd.android.cursor.item/phone`
  - Display the list of people's phone numbers. The user can browse the list and return one specific number to the parent activity.

2.6 iOS software stack

The basic architecture for iOS is divided into four layers [5]:

- The lower layers:
  - core OS;
  - core services layer;

  manage basic services in iOS;
- The upper layers:
  - media layer;
  - cocoa touch layer;

  handle the user interface and advanced graphics;

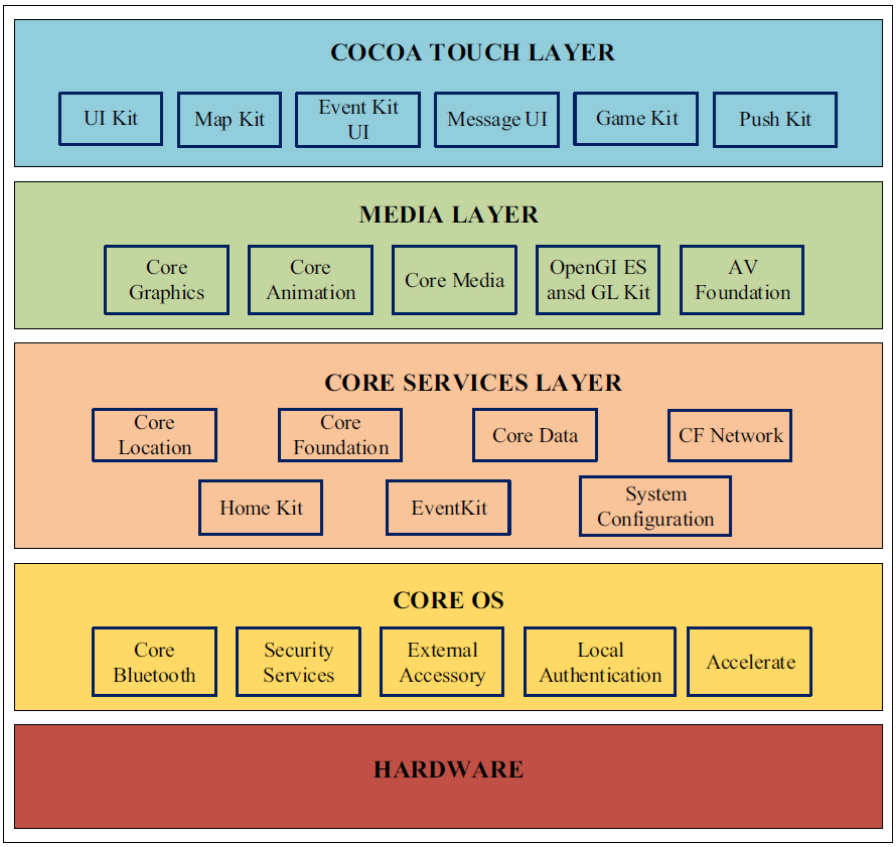


Figure 2.13: iOS architecture layers

2.6.1 Cocoa Touch Layer

The Cocoa Touch layer contains frameworks for developing iOS apps [5]. This layer defines the basic application infrastructure and support for key technologies such as multitasking, touch-based input, push notifications, and many high-level system services. Here some of the key technologies available in the Cocoa Touch layer [6]:

- **AutoLayout**: Introduced in iOS 6, with some improvements with respect to the previous model, it allows to define rules for how to lay out the elements in the user interface.
- **Storyboards**: Introduced in iOS 5, they replace `.nib` files as the recommended way to design an app's GUI. Unlike `.nib` files, storyboards allow to design the whole user interface in one place, making it possible to simultaneously see all its views and view controllers and how they interact with each other. Storyboards can also define segues, which are transitions from one view controller to another. Applications can define these transitions visually in Xcode or initiate them programmatically. These transitions allow to capture the flow of the user interface in addition to the content.
- **Document Support**: Introduced in iOS 5, the UIKit framework introduced the UIDocument class for managing data associated with user documents. This class is useful for implementing document-based applications, especially the ones that store documents in iCloud. Not only the UIDocument class provides a container for all the document-related data, but it also provides built-in support for asynchronous reading and writing of data, safe automatic saving of data, support for iCloud conflicts detection, and other features.
- **Multitasking**: Applications built using iOS SDK 4.0 are not terminated when pressing the device's "Home" button but they keep running in the background. UIKit defines multitasking support defined which allows an app to transition to and from the background state. When entering the background, most applications get suspended by the system to save battery. Hence, they remain in memory but they're not executed.
- **UI State Preservation**: Introduced in iOS 6, this feature allows an app to restore its UI to the state it was in last time it got used. This is possible because, when entering the background, an app saves the state of its views and view controllers. The UIKit implements features for state preservation.
- **Standard System View Controllers**: Many of the frameworks in the Cocoa Touch layer contain view controllers for presenting standard system interfaces. A view controller could be used for performing one of the following tasks from the corresponding framework:
  - Display or edit contact information (Address Book UI framework);
  - Create or edit calendar events (Event Kit UI framework);
  - Compose an email or SMS message (Message UI framework);
  - Open or preview the contents of a file (`UIDocumentInteractionController` class in the UIKit framework);
  - Take a picture or select a photo from the user's gallery, or shoot a video clip by using the `UIImagePickerController` class of the UIKit framework;

Other technologies supported by cocoa touch layer are: Printing, Apple Push Notification Service, Local Notifications, Gesture Recognizers, Peer-to-Peer Services, External Display Support.

2.6.2 Core Services Layer

The Core Services layer contains system services which are used by all applications and many system components, even if the user does not use them directly [5, 7]:

- **iCloud Storage**: Introduced in iOS 5, it lets an app store user documents and data on cloud, so that they can be accessed by all of the user's authorised devices. Applications can use the iCloud storage in two ways:
  - **iCloud document storage**: Stores documents and data in the user's iCloud account;
  - **iCloud key-value data storage**: Used to share small amounts of data among instances of an application;
- **Automatic Reference Counting** (**ARC**): Introduced in iOS 5, it is a compiler-level feature that makes it easier to manage Objective-C objects. Instead of having to remember when to retain or freeing an object, ARC evaluates its lifetime requirements and it will automatically add the proper method calls at compile time.
- **SQLite**: The SQLite library allows to embed a lightweight SQL database into an app.
- **XML Support**: It's implemented by the `NSXMLParser` class of the Foundation framework and it allows to extract elements from an XML file. Additional support is provided by the `libXML2` library.
- **Block Objects**: Introduced in iOS 4.0, are C anonymous functions. Data supplied to block objects as input is usually called "closure" or "lambda" in other programming languages. In iOS, blocks are commonly used:
  - To facilitate processing all the items in a collection;
  - To perform asynchronous tasks, when combined with dispatch queues;
  - To implement completion handlers for one-time operations;
  - To replace delegates, delegate methods, and callbacks;

The core services layer supports also: Data Protection (discussed in Chapter 4), File sharing, In-App purchases and other features.

2.6.3 Media Layer

The Media layer contains all the technologies for creating the user's multimedia experience (i.e., audio, video, etc...). Here some of the key technologies available in the media layer [5, 8]:

- **Graphics Technologies**: Generally, a straightforward way to create an app consists in using pre-rendered images in combination with the UIKit framework. However, if high-quality graphics are needed, these technologies will be useful.
- **Audio Technologies**: They implement all the features to record and play audio, as well as triggering the vibration on some devices.
- **Video Technologies**: They implement the features for recording and playing videos, which could also be embedded into an app.
- **AirPlay**: It's support is implemented by the AV Foundation framework and the Core Audio family of frameworks, and it allows an app to stream audio to an Apple TV and to third-party AirPlay speakers and receivers.

2.6.4 Core OS

The Core OS layer contains the low-level features that most other technologies are built upon [5]. Even if the user does not use these technologies directly, they are most likely being used by system frameworks, as they're useful for dealing with security and communicating with an external hardware accessory [9].

2.7 Apple File System

Apple File System (**APFS**) [10–12] is optimized for Flash and SSD storage and it makes extensive use of encryption techniques for protecting data. APFS uses copy-on-write operations to ensure data reliability and I/O coalescing allows not to lose performance. In iOS, storage is divided into at least two APFS volumes:

- System volume;
- Data volume;

When a single APFS container has multiple volumes, its unallocated space is shared among them. Hence, each volume uses only part of the overall container.

2.8 iOS Package App (IPA)

Application files in iOS are compressed archives with the `.ipa` extension [13–15] (which stands for iOS Package App). Despite their extension, these are just regular ZIP files and can be extracted by modifying the suffix to `.zip`. The content of an IPA file is the following:

- `Info.plist` **file** [16]: Describes the application to the iOS operating system by using a list of properties such as, e.g., which files are the app icon, the app display name, version, unique ID, main executable filename.
- **Main executable file**: All the Objective-C and Swift code used to develop the app is encoded into this file. The application file contains within it entitlement information, which is Apple's name for permissions.
- **External executable libraries**: Other libraries that the main executable uses.
- **Frameworks**: This is a folder that contains frameworks used by the application. Each framework resides under its own folder called `Frameworks/.framework` and contains its native code and resources. In addition, Swift applications have the Swift runtime libraries directly under the `Frameworks` folder.
- **Plugins**: The application executes application extensions.
- **Resources**: Documents, images, icons, video and audio files.
- `www` **folder**: If the application is a web app developed with Cordova or React Native for example, the web data will be stored into this folder. It may contain web pages, resources, Javascript and CSS files, and so on. Files might be encrypted;
- **Nib** [17] or **storyboard files**: These files describe the application's UI and how it interacts with its logic.
- **Signature information**: iOS verifies that all native code elements have not been modified since they were "signed" by the developer/distributor. This includes the executable, frameworks, libraries, PlugIns (app extensions), and WatchKit.

- **Provisioning information** (most commonly `embedded.mobileprovision`): This file determines the **deployment permissions** of the application. Unless the application is published to the App Store (or acquires an enterprise code signature), it must be equipped with a provisioning profile. The provisioning profile is also a `.plist` file, and should be obtained by the developer from Apple's developer console. The provisioning information contains inside it a copy of the application's entitlements, so an application's permissions must all be "authorized" by Apple.

2.9 Extensions support in iOS

iOS allows apps to provide features to other apps by using signed executable binaries packaged within them, called **extensions** [10].

2.9.1 Extension points

An **extension point** is a system area that supports extensions, provides APIs and enforces a set of policies [10]. The system detects and automatically launches extension processes when needed. Extensions usage can be restricted to specific system apps by using **entitlements** (Section 4.5.2).

2.9.2 How extensions communicate

Extensions run in their own address space and communicate with apps by using IPC mechanisms which are mediated by the system framework [10]. Extensions are isolated from their wrapping apps, as well as from each other and from the other apps that use them (i.e., they're sandboxed). However, they share the same access rights to privacy controls as the container app. So if a permissions is granted to an app, it will also be automatically granted to all the extensions embedded in (and not activated by) that app.

Chapter 3

Android system and Kernel security

3.1 Security overview

Android inherits some of its security features from the Linux kernel, including [18]:

- A permissions model based on UIDs;
- Extensible mechanism for secure IPC;
- Process isolation;
- The possibility to remove parts of the kernel which have been tampered with;

User resources are protected from one another:

- Users can't read each other's files;
- Users can't exhaust each other's memory and CPU resources;
- Users can't exhaust each other's devices (e.g., GPS, Bluetooth, and telephony);

3.1.1 Android security mechanisms

The main security mechanisms adopted by Android are [18]:

- **Application Sandbox**: It exploits the permissions system to isolate apps from each other and protect the overall system from malicious apps.
- **System Partition and Safe Mode**: The system partition is mounted as read-only and, when the device is booted into Safe Mode, third-party apps can't be automatically launched by the system but require user interaction to start running.
- **Filesystem Permissions**: Each application inherits the permissions assigned to its own UID and its files can't be read nor written by another app, unless the developer decided to explicitly share them with other applications.
- **Security-Enhanced Linux**: It's a security mechanism that enforces access control policies and Mandatory Access Control (MAC) on processes.

- **Verified boot**: Supported on Android 7.0 and later, it guarantees the integrity and authenticity of the system software thanks to a chain of trust that goes from a hardware root of trust (RoT) up to the system partition. Each stage of the boot verifies the next one, before executing it.
- **Cryptography**: Cryptographic APIs are provided to Android apps and offer implementations of standard primitives like AES, RSA, DSA, and SHA. These APIs are also used for implementing network security protocols like SSL and HTTPS.
- **Lockscreen Credential Protection**: Access to the device is allowed only after submitting the correct user credential (i.e., PIN, password, or pattern). The credential is also used to protect the cryptographic key for decrypting "credential-encrypted data".
- **Storage Encryption**: All devices with Android 10 or later us a combination of file-based encryption and metadata encryption:
  - **File-based encryption** encrypts file data and names on the user-data partition by using a different encryption key for each storage directory, which can either be credential-encrypted or device-encrypted.
  - **Metadata encryption** encrypts all data, on the user-data partition, that hasn't been encrypted with file-based encryption. The encryption key is not derived from the user's lockscreen credentials but it's still protected by Verified Boot.

3.1.2 Rooting of devices

Rooting an Android device means to find a way to acquire root privileges. As discussed in Chapters 10 and 11, spyware need root privileges to be able to fully exploit the target system. However, rooting an Android device is not an easy task:

- Only the Linux kernel and a few core services run with root privileges.
- SELinux still enforces MAC on user processes with root privileges.
- Verified boot prevents permanent modification of the OS software, even by root processes.

However, as stated by Android Developer [18]:

- On many Android devices, the bootloader can be unlocked and an alternative OS which may provide root privileges to the user could be installed.
- On some Android devices, an attacker with physical access to the device can use an USB cable to install a new OS that provides root privileges to the user.

Android Developer also suggests some approaches to protect data at rest on a rooted device:

- Application data encrypted with a criptographic key stored on-device is not protected from users with root privileges on a rooted device. Hence, encryption with a key stored off-device (e.g., on a server) should be used to add extra protection. However, this approach can only protect data as long as the key is not supplied to the device (which, sooner or later, will likely happen).
- A securer approach consists in using hardware solutions to ensure that user data cannot be accessed without knowing the user's lockscreen credential.

3.2 Application Sandbox

Android uses Linux user-based protection to isolate and protect app resources [19]. When an application is installed, a new unique **UID** (Linux user ID) is created for it, and all of its code runs as that "user". Linux user IDs thus create a sandbox for each application, with their own isolated area of the file system. Android's kernel-level Application Sandbox enforces security between the apps and the system thanks to the "identity" which is assigned to each process with user and group IDs. Hence, an application is not allowed to perform specific actions if its UID doesn't hold the corresponding permissions. This security measure affects both native code and OS apps, as the sandbox is implemented at kernel-level.

3.2.1 Sandbox protections

On a properly configured device, an app can break out of the sandbox only if the Linux kernel gets compromised. As discussed in Chapter 7, most of Android vulnerabilities are in the kernel. So, additional security measures have been implemented by developers over the years, as such vulnerabilities were being patched, to provide defense-in-depth. Thanks to this, the original UID-based **Discretionary Access Control** (**DAC**) sandbox got significantly improved.

These are some of the protections implemented in previous Android versions [19]:

- In Android 5.0, despite SELinux implementing MAC between the apps and the system, third-party apps were executed within the same context and they were not isolated from each other by SELinux itself but by the UID DAC instead.
- In Android 6.0, the SELinux sandbox was strengthened to isolate apps across different user profiles on the same device. Moreover, for apps with `targetSdkVersion >= 24`, default DAC permissions on an app's home dir changed:
  - from `751 (owner: rwx, group: rx, public: x)`
  - to `700 (owner: rwx)`
- In Android 8.0, the `seccomp-bpf` filter was implemented to limit the number of system calls which apps were allowed to use.
- In Android 9, the SELinux sandbox was extended to provide inter-app isolation for all non-privileged apps with `targetSdkVersion >= 28`.
- In Android 10, the applications' view over the filesystem got reduced by denying access to paths like `/sdcard/DCIM`. Apps still had full access to their own directories (i.e., package-specific paths).

The `targetSdkVersion` is an integer attribute, declared in the app manifest within the "uses-sdk" tag, designating the API level that the app targets. If not set, the default value equals that given to `minSdkVersion`, which is the minimum Android API level required for the app to run.

```
<uses-sdk android:minSdkVersion="integer"
android:targetSdkVersion="integer"
android:maxSdkVersion="integer" />
```

Listing 3.1: Structure of the <uses-sdk> tag

3.2.2 Filesystem UIDs

Binder IPC includes the UID of the caller process in every transaction delivered across processes, so that recipients can easily retrieve that information. Lower-level parts of the Android system have fixed UIDs below `10000`, but most apps are dynamically assigned one by the package manager at install time, from the `10000-19999` range of application UIDs. This means at most 10.000 applications can be installed on the system. The range starting at `100000` is used to implement a multi-user model in Android: an application that is granted UID `10002` as its identity would be identified as `110002` when running as a second user [3].

UID	Purpose
`0`	Root
`1000`	Core system (`system_server` process)
`1001`	Telephony services
`1013`	Low-level media processes
`2000`	Command line shell access
`10000-19999`	Dynamically assigned application UIDs
`100000`	Start of secondary users

Table 3.1: Common UID assignments in Android

When an application is first assigned a UID, a new storage directory is created for it, with the files there owned by its UID. The application gets free access to its private files there, but cannot access the files of other applications nor vice versa. Content providers are one of the few mechanisms that can transfer data between applications. Even the system itself, running as UID `1000`, can't modify app files. This is why the `installd` daemon exists: it runs with special privileges to access and create files and directories for other applications, and it provides an API to the package manager for it to create and manage the data directories of apps.

3.2.3 Filesystem permissions

An app lists the permissions it needs in its manifest and, prior to installing it, the user is informed of what it will be allowed to do based on them [3]. Figure 3.1 shows how an e-mail application could make use of permissions to access pictures in the camera app. In this case, the camera app has associated the `READ_PICTURES` permission with its pictures. The e-mail app declares in its manifest that it requires this permission and can then access a URI owned by the camera, such as `content://pics/1`. Upon receiving the request for this URI, the camera app's content provider asks the package manager whether the caller holds the necessary permission. If it does, the call succeeds and requested data is returned to the application.

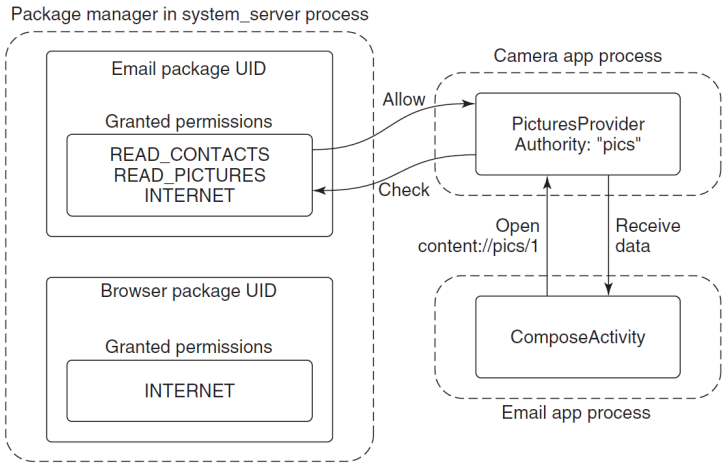


Figure 3.1: Requesting and using a permission

Permissions are not tied to content providers but are associated with UIDs instead. Given this, a permission check can be performed by retrieving the UID associated with the incoming IPC and asking the package manager whether that UID has been granted the corresponding permission. When an application does not hold a permission needed for an operation it is performing, a `SecurityException` is thrown back at it by the content provider.

3.2.4 Secure data access through content provider

Android also supports an implicit secure data access mechanism through intents and content providers. Figure 3.2 illustrates how this situation works for our picture emailing example [3]:

1. The camera application has created an intent asking to share the image `content://pics/1`;
2. In addition to starting the email compose application as we had seen before, this also adds an entry to a list of "granted URIs", noting that the new `ComposeActivity` now has access to this URI;
3. Now, when `ComposeActivity` looks to open and read the data from the URI it has been given, the camera application's `PicturesProvider` that owns the data behind the URI can ask the activity manager if the calling email application has access to the data, which it does, so the picture is returned;

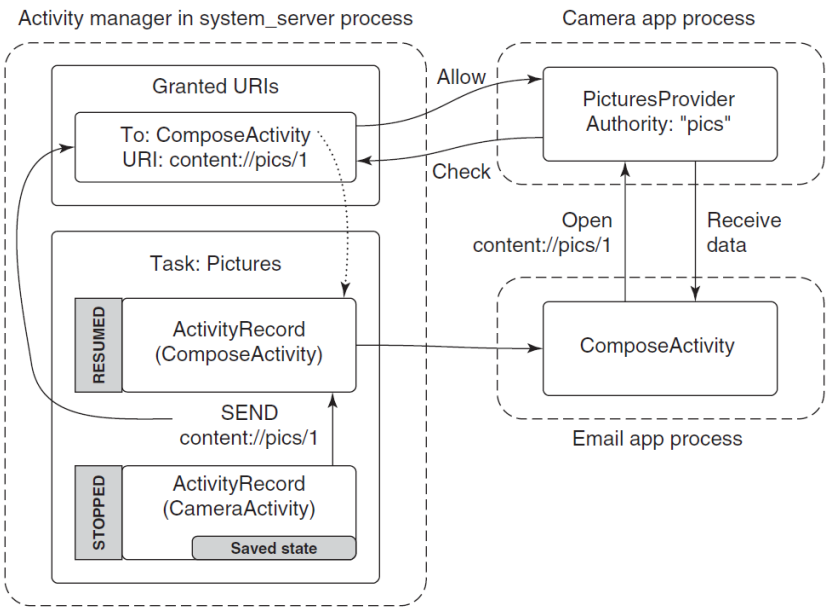


Figure 3.2: Sharing a picture using a content provider

This fine-grained URI access control can also operate the other way. There is another intent action, `android.intent.action.GET_CONTENT`, which an application can use to ask the user to pick some data and return to it. This would be used in the e-mail application, e.g., to operate the other way around: the user, while in the e-mail application, can ask to add an attachment.

This will launch an activity in the camera application to select one. The following figure illustrates this new flow. It's almost identical to the one of the previous figure, the only difference being in the way the activities of the two applications are composed, with the email application starting the appropriate picture-selection activity in the camera application. Once an image is selected, its URI is returned back to the email application, and at this point our URI grant is recorded by the activity manager.

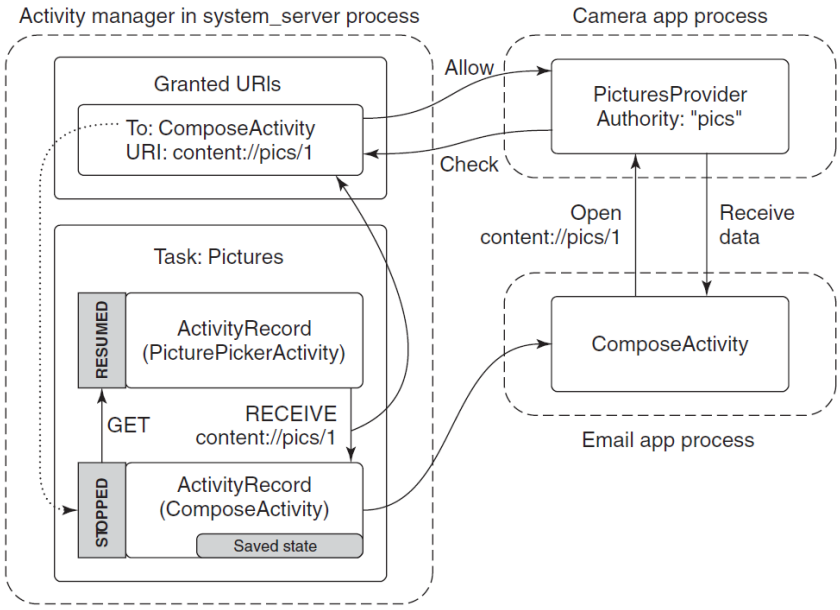


Figure 3.3: Adding a picture attachment using a content provider

3.3 SELinux

Security-Enhanced Linux (SELinux) is a flexible **Mandatory Access Control** (**MAC**) system built into the Linux Kernel that provides greater access granularity over the existing Linux DAC. SELinux applies MAC on all processes, even those with root privileges, and follows the **default denial principle**, meaning anything which isn't explicitly allowed gets denied.

SELinux can be configured to operate in two modes [20]:

- **Permissive mode**: permission denials are logged but not enforced;
- **Enforcing mode**: permissions denials are both logged **and** enforced;

Additionally, there's also a per-domain permissive mode which allows to make specific domains permissive while using enforcing mode for the rest of the system. A **domain** identifies one or more processes in the security policy with a label. Processes which are labeled with the same domain are treated the same way by the policy.

Here's how SELinux evolved through the years [20]:

- Starting from Android 5.0, enforcement mode was extended to all domains (more than 60) while before it was only used for a limited set of domains (i.e., `installd`, `netd`, `vold`, `zygote`). Specifically:
  - No processes other than `init` should run in the `init` domain;
  - Any generic denial (for a `block_device`, `socket_device`, `default_service`) indicates that device needs a special domain.
- In Android 6.0, the permissiveness of the security policy got tightened. This introduced better isolation between users, IOCTL syscall filtering (the `ioctl` system call is used for device-specific IO operations), limited access to the `/proc` directory, and it enhanced the overall system security.
- In Android 7.0, SELinux configuration was updated to further reduce the attack surface by locking down the application sandbox.
- In Android 8.0, SELinux was updated to work with Treble [21], which separates the Android system framework from lower-level vendor code. By updating the SELinux configuration, device manufacturers and SOC vendors could update their parts of the policy, build and then update their images (e.g., `vendor.img`, `boot.img`, etc...) without depending from the Android platform.

3.4 Storage encryption

3.4.1 File-based encryption (FBE)

File-based encryption (**FBE**) is required in Android 10 and later, but it is already supported starting from Android 7.0 [22]. FBE encrypts file data by using a different key for each file. Encryption keys are independent from each other and they can be unlocked separately as needed. FBE enables encrypted devices to directly boot to the lock screen, without needing the user to insert their credentials for the phone to start operating. While **full-disk encryption** (**FDE**) did not allow a locked device to start any of its basic services (e.g., telephony stack, accessibility services, etc...) until the lock screen credentials were inserted, devices encrypted with FBE and not yet unlocked can still, e.g., receive phone calls, start alarms, and more. Applications can be made aware of encryption, thanks to the FBE APIs, and hence they can operate within a restricted context until the user inserts the lock screen credentials. Each user of an FBE-enabled device can use two separate storages for their applications:

- **Credential Encrypted** (**CE**) storage, available after unlocking the device;
- **Device Encrypted** (**DE**) storage, available also during direct boot;

FBE is enabled by adding the option

```
fileencryption=contents_encryption_mode[:filenames_encryption_mode[:flags]]
```

to the `fs_mgr_flags` column of the `fstab` line for the `userdata` partition, which contains user-installed apps and data, including customization data.

The default criptographic algorithm for encrypting **file contents** is `aes-256-xts` [23], but it can be changed to `adiantum` [24] by setting the `contents_encryption_mode` parameter. The default algorithm for encrypting **file names** changes from `aes-256-cts`, if file contents have been encrypted with `aes-256-xts`, to `adiantum`, if `adiantum` itself has been used for contents encryption. However, `aes-256-heh` and `aes-256-hctr2` are also supported and can be enabled by setting the `filenames_encryption_mode` parameter. Starting from Android 14, `aes-hctr2` is preferred for devices with cryptography accelerators.

3.4.2 Metadata encryption

Starting from Android 9, support for **metadata encryption** has been added [25]. Information like file size, permissions (i.e., read, write, execute), creation/modification times, and directory layouts, collectively called **filesystem metadata**, are not encrypted by FBE. So, metadata encryption is used for encrypting whatever has not been encrypted by FBE. The encryption key is only present at boot time and it's protected by the Keymaster HAL [26], which is also protected by verified boot. The default encryption algorithm to be used on the internal storage metadata is `aes-256-xts`, but devices with no AES accelerator `adiantum` could be used instead. If FBE is enabled, metadata encryption is automatically enabled on any adoptable storage. Starting from Android 11, metadata encryption is mandatory for the internal storage.

3.4.3 Adoptable storage encryption

Starting from Android 6.0, an external storage (e.g., SD card) can be **adopted** (i.e., formatted and encrypted) so that it can only work with one specific device [27]. An adopted storage can be used for securely storing user and app data. However, FBE support for adoptable storage has only been added in Android 9. Random encryption keys are generated for each adopted device and stored on the internal storage. If FBE is used on the internal storage, it will be used for the adoptable storage as well, together with metadata encryption. Otherwise, full-disk encryption will be used instead. Note that encryption keys must not be extractable from the Android device, or the adopted storage could be mounted elsewhere.

3.5 Android Backups

3.5.1 Mobile device backup

Mobile device backup can be automatically performed with **Google One**. An extra copy of photos, videos, contacts, call history, device settings (e.g., Wi-Fi passwords, wallpapers, etc...), applications data and other elements will be stored on the cloud. When using a new device, backup data can be restored. Backup copies are sent to Google servers and encrypted with the user's Google account password. Some data may be encrypted by using also one among the PIN, sequence, password used for unlocking the device's screen. Backup data (except backup copies stored on Google Photo) will be deleted either if the user requests to do so or if the mobile device is not used for 57 days.

3.5.2 Application data backup

Applications commonly store data for use, whether locally on the device, within external storage, or remotely in cloud storage. When stored data relates to sensitive information, such as a user's personal data or authentication keys and passwords, additional security measures can be applied to prevent the leaking of this sensitive data if the backup is accessed by someone other than the intended user. One of the most common methods of retrieving backup data consists in using the **ADB backup command** [4], which requires **Developer options** [28] to be enabled on the device. As enabling Developer options requires the use of the user's PIN, any potential attacker would need to first compromise a user's PIN in order to retrieve any backup data.

Here are some of the major **risks** associated to application data backups:

- An attacker with access to an application's backup file can retrieve any unencrypted data that the application has backed up and exploit it in future attacks;
- Any attacker with physical access to an unlocked device or access to a compromised Google Account can perform a backup of an application and its data;
- It is important to note that, as of Android 12, application data is not included within the backup data, as long as the `android:debuggable` attribute in the Android Manifest is set to `false`;
- Not all phone manufactures adhere to an application's backup policy. As a result, data that is not intended to be backed up may be backed up anyway through a unique OEM-backup solution. In such cases, additional measures should be implemented to protect user data, such as encryption and removing or not storing unnecessary data that may be considered sensitive;

3.5.3 Auto backup for apps

Auto Backup is a feature introduced in Android 6.0 to automatically back up user data from apps to the user's Google Drive. On Android 9 and later, the backup is end-to-end encrypted by using the device's lock screen credentials. The backup includes, by default, files saved to the app's internal storage, shared preferences files, and others. Data is stored in a private folder on Google Drive, with a maximum size of 25 MB for each app. When a backup is made, any older backup for the same app is deleted and only the new (most recent) backup is retained. Backup data can't be read by the user nor by any app on the user's device [29].

Backups are managed as follows:

- If the user owns more devices, then a backup dataset exists for each device;
- If a device is reset to factory settings but it's then set up with the same account, the backup is stored in a new dataset;
- Datasets are automatically deleted after a period of inactivity;

3.6 Biometric authentication security

Implementations of biometric authentication must meet the requirements presented in the **Android Compatibility Definition Document** (**CDD**) [30].

Compatibility is checked by evaluating an implementation on the following basis [31]:

- **Architectural security**: "The resilience of a biometric pipeline against kernel or platform compromise. A pipeline is considered secure if kernel and platform compromises do not confer the ability to either read raw biometric data or inject synthetic data into the pipeline to influence the authentication decision".
- **Biometric security performance**: "Measured by the Spoof Acceptance Rate (SAR), False Acceptance Rate (FAR), and, when applicable, Imposter Acceptance Rate (IAR) of the biometric. SAR is a metric introduced in Android 9 to measure how resilient a biometric is against a physical presentation attack".

The following metrics are used for evaluation [32]:

- **Spoof Acceptance Rate (SAR)**: "Defines the metric of the chance that a biometric model accepts a previously recorded, known good sample. For example, with voice unlock this would measure the chances of unlocking a user's phone using a recorded sample of them saying: "Ok, Google" (spoof attack). Also known as Impostor Attack Presentation Match Rate (IAPMR)".
- **Imposter Acceptance Rate (IAR)**: "Defines the metric of the chance that a biometric model accepts input that is meant to mimic a known good sample. For example, in the Smart Lock trusted voice (voice unlock) mechanism, this would measure how often someone trying to mimic a user's voice (using similar tone and accent) can unlock their device (imposter attack)".
- **False Acceptance Rate (FAR)**: "Defines the metrics of how often a model mistakenly accepts a randomly chosen incorrect input. This measure alone does not provide sufficient information to evaluate how well the model stands up to targeted attacks".
- **False Rejection Rate (FRR)**: "Defines the metrics of how often a model mistakenly rejects a correct input. This measure alone does not provide sufficient information to evaluate how well the model stands up to targeted attacks".

3.6.1 Presentation Attack Instrument (PAI)

It's the physical mean used by an attacker to perform a presentation attack, i.e., an attack which tries to spoof the victim's physical traits [31].

- **Face** PAI species:
  - 2D PAI: printed photos, photos/videos on a monitor or a phone display;
  - 3D PAI: 3D printed masks;
- **Iris** PAI species:
  - printed face pictures clearly showing the iris, photos/videos displayed on a monitor that clearly shows the iris, or even prosthetic eyes;
- **Fingerprint** PAI species:
  - printed fingerprints or molded replica;

Spoofing is possible also by further manipulating the PAI or even the environment, i.e., the **presentation format** [31]:

- **Face** PAI presentation format:
  - Folding printed pictures to (slightly) mimic depth by creating curves at the cheeks;
  - Changing light conditions;
  - Changing the phone orientation or dirtying its lens;
- **Iris** PAI presentation format: placing a contact lens over the display/printed photo;
- **Fingerprint** PAI presentation format: editing a high resolution fingerprint image before creating its 3D replica;

3.6.2 Biometric authentication solution evaluation process

The evaluation protocol is made up of the following phases [33]:

- **Enrollment phase**: "All existing biometric profiles are removed from the device and a new profile, with the target face/iris/fingerprint that will be used for calibration and testing, is enrolled".
- **Calibration phase**: "Determine the optimal presentation attack for a given authentication solution, i.e., the optimal values for parameters that maximize the chances of spoofing the authentication solution".
  - For face, iris and fingerprint, it consists in finding the optimal **calibrated position**, e.g., the optimal angle for positioning the sensor when performing authentication attempts.
- **Test phase**: "Perform multiple attacks and evaluate the number of successes".

The SAR is then computed as

$$SAR = \frac{\sum_{i=1}^{E} S_i}{U \cdot E} \cdot 100\%$$

where

- E is the number of enrollments;
- U is the number of unlock attempts per enrollment;
- S_i is the number of successful unlocks for enrollment i;

3.6.3 Biometric classes

Biometric classes are defined by using the aforementioned metrics and each implementation is evaluated based on its architectural security and on the results of the spoofability tests [33]:

- **Class 3** (formerly **Strong**):
  - SAR of all PAI species: $0 - 7\%$;
  - Secure pipeline;
- **Class 2** (formerly **Weak**):
  - SAR of all PAI species: $7 - 20\%$;
  - Secure pipeline;
- **Class 1** (formerly **Convenience**):
  - SAR of all PAI species: $20 - 30\%$;
  - Insecure/Secure pipeline;

For all classes it must hold: $FAR \leq 1/(5 \cdot 10^4)$, $FRR \leq 10\%$

3.6.4 Fingerprint authN insights

Fingerprint matching

The high-level flow of fingerprint matching is [34]:

1. The user places a finger on the sensor;
2. The vendor-specific library determines if there's a match in the set of currently enrolled templates;
3. Matching results are passed to the `FingerprintService`;

Architecture

The Fingerprint HAL interacts with the following components [34]:

- `BiometricManager` interacts directly with an app in an app process. Each app has an instance of `IBiometricsFingerprint.hal`;
- `FingerprintService` operates in the system process, which handles communication with fingerprint HAL;
- **Fingerprint HAL** is a C/C++ implementation of the IBiometricsFingerprint HIDL interface. This contains the vendor-specific library that communicates with the device-specific hardware;
- Keystore API and Keymaster components provide hardware-backed cryptography for secure key storage in a secure environment, such as the **Trusted Execution Environment** [32].

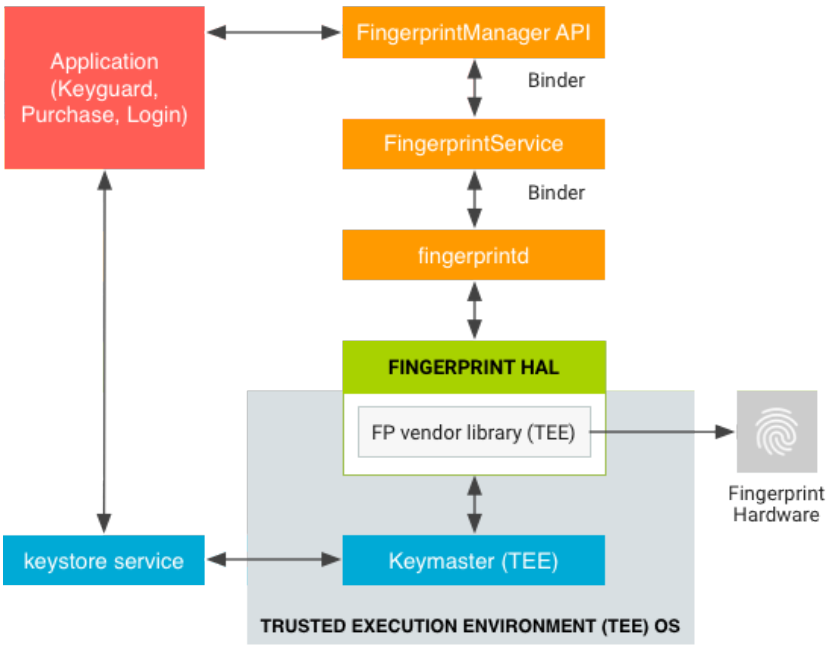


Figure 3.4: High-level data flow for fingerprint authentication

A vendor-specific HAL implementation must use the communication protocol required by a TEE. Raw images and processed fingerprint features must not be passed in untrusted memory. All such biometric data needs to be stored in the secure hardware such as the TEE. Rooting must not be able to compromise biometric data.

`FingerprintService` and `fingerprintd` make calls through the Fingerprint HAL to the vendor-specific library to enroll fingerprints and perform other operations.

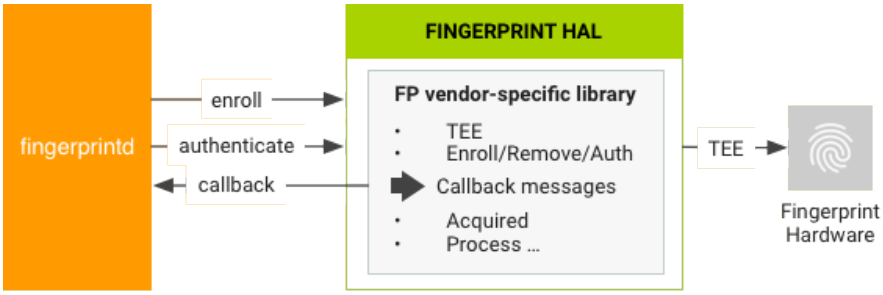


Figure 3.5: Interaction of the fingerprint daemon with the fingerprint vendor-specific library

Fingerprint templates

A **fingerprint template** is the biometric data obtained by a fingerprint sensor [34]. The process of obtaining a fingerprint template is called **enrollment**. During the enrollment, data is transformed into zeroes and ones, creating a mathematical file, which is further stored in the biometric database. In short, a fingerprint template is the digital representation of a fingerprint.

Fingerprint authN implementation guidelines

Here the Android implementation guidelines for fingerprint authentication [34]:

- "Raw fingerprint data or derivatives (for example, templates) must never be accessible from outside the sensor driver or TEE. If the hardware supports a TEE, hardware access must be limited to the TEE and protected by an SELinux policy. The Serial Peripheral Interface (SPI) channel must be accessible only to the TEE and there must be an explicit SELinux policy on all device files".
- "Fingerprint acquisition, enrollment, and recognition must occur inside the TEE".
- "Only the encrypted form of the fingerprint data can be stored on the file system, even if the file system itself is encrypted".
- "Fingerprint templates must be signed with a private, device-specific key. For Advanced Encryption Standard (AES), at a minimum a template must be signed with the absolute file-system path, group, and finger ID such that template files are inoperable on another device or for anyone other than the user that enrolled them on the same device. For example, copying fingerprint data from a different user on the same device or from another device must not work".

3.7 Updates and security patches

In Android, there are two types of updates. There are major upgrades to a new version of Android, usually once a year, and there are more frequent security patches, usually on a monthly, bi-monthly, or quarterly basis [35].

- **Upgrades**: "They represent a major shift in the functionality and sometimes even the overall look of the device's software".
- **Security patches**: "They enhance the security of the device. Sometimes, they also add new features, although this depends on the device's manufacturer".

3.7.1 Receiving Android updates

Updates are delivered through over-the-air (OTA) update packages [36] which could come either from the original equipment manufacturer (OEM), who produced the device, or from the carrier, who provides service to the device [37]. For example, updates to Google Pixel devices come from the Google Pixel team, who tests each update with a carrier technical acceptance (TA) procedure, before releasing it. Google Pixel devices usually receive patches and upgrades before devices which are not manufactured by Google, whose updates are edited by the OEM and adapted to specific products (which likely causes delays). Generally, how frequently updates are delivered to devices depends on how seriously the OEM takes them [35].

3.7.2 User concerns

There are way too many factors that may influence the frequency of security updates releases [38]:

> "A straightforward security update rollout process becomes cumbersome with every new device/model released by OEMs. Samsung has a billion active users with 1400 unique models (e.g., SM-970U) of 402 devices (e.g., Galaxy S22) associated with 97 countries and 109 carriers. When we consider all unique model and country-carrier combinations, this requires the creation, customization, and testing of approximately 20K variations of each security update periodically (e.g., monthly). Other OEMs (e.g., Xiaomi), though on a smaller scale, also operate in many regions with large sets of end users. This workload is causing irregularities in the security update support of end-user devices, such as delays in the security update process or the failure to deliver the security updates at all for some models".

3.7.3 Risks of using unpatched Android devices

Regularly installing security patches helps minimizing the risks of using an unpatched Android device. As a matter of fact, on average, the number of CVEs in unpatched devices goes from ≈ 80 in the first three months, to ≈ 380 after two years and reaches ≈ 600 after five years. Usually, CVEs increase immediately after the support for a specific product is not provided anymore. The number of critical CVEs usually stops to grow after two years, while the number of medium and high CVEs keeps growing until three years of use of the unsupported device are reached, after which the growth slows down. On the other hand, low-severity CVEs remain the lowest throughout the entire period [38].

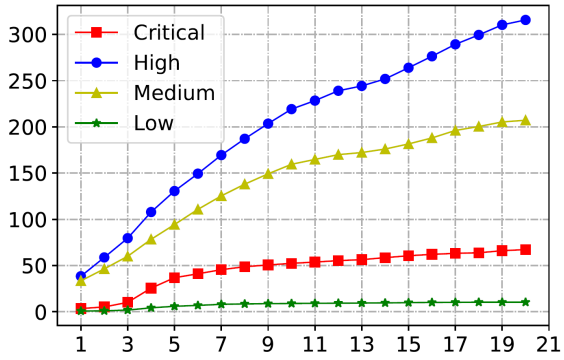
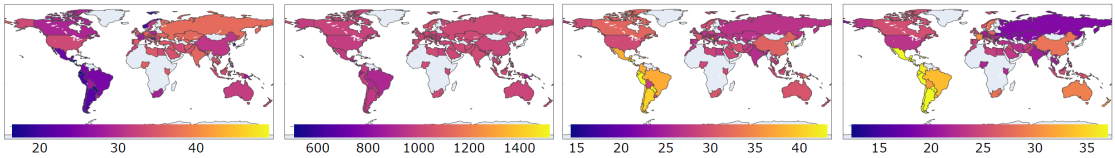


Figure 3.6: Accumulated CVEs, per device, every quarter (3 months) for ≈ 5 years

3.7.4 Samsung security updates distribution over countries

The research made by [38] also gathered some statistics regarding Samsung security updates distribution over the world. These data help understanding that geographical distribution of security patches is not uniform and hence some users may not be able to update their devices as frequently as others. Here we cite their results:

> "Three of the top five countries receiving the most security updates are in Europe and the other two are in Central and Southern Asia. These top five countries receive monthly support for around three years with an update frequency of 26-28 days and a release delay of 18-21 days. On the other hand, all of the bottom five countries are in Latin America and the Caribbean. While devices in these countries receive monthly support, they received only 15-17 security updates despite having almost the same duration (≈ 3 years) of support as the top countries. Similarly, these countries have a release delay of around 50 days, while we observed that the release delay of top five countries is around 20 days".



(a) Update Count (b) Support Duration (days) (c) Update Frequency (days) (d) Release Delay (days)

Figure 3.7: Geographical distribution of security patches for Samsung devices with a monthly support

3.8 Enterprise identity, security and management

3.8.1 Device and profile management

Android offers enterprise solutions that allow company administrators to monitor their employees' devices. An Android enterprise solution is made by the following components [32, 39]:

- **Enterprise Mobility Management (EMM) console**: A web application which IT admins use to manage the organization, the devices, and even the apps.
- **Android Device Policy**: It's an application, provided by Android, that enforces management policies to devices.
- **Managed Google Play**: It's an alternative version of the Google Play store, specifically designed for companies. It can be included in the EMM console for enabling app search, publishing and management.

Android Enterprise's tools and services support any or all of the following work profiles [32, 40]:

- **Employee-owned devices (BYOD)**: With a "bring your own device" (BYOD) solution, employees can use their own devices for working. In order not to compromise security, the work profile and the corresponding environment is kept separate from the private ones. The company has full control of the work profile (i.e., apps, data, settings, etc...) but has no visibility nor access to the employee's personal profile.
- **Mixed-used company-owned devices**: It's similar to the personally-owned device work profile but this time the device is owned by the company, which can enforce security policies and restrictions also to the personal profile (e.g., block installation of specific apps). Moreover, the work profile can't be removed unless the device's ownership is given up to the employee.
- **Work-only company-owned devices**: This work profile, designed for company-owned fully managed devices, is for work purposes only. The company can enforce far more policies, including device-level policies.
- **Dedicated devices**: Dedicated devices are a type of fully managed devices which can only run one app or set of apps, as they are special-purpose devices. Users are prevented from enabling other applications or performing illegal actions.

3.8.2 Work challenge

Work profiles can be protected with a separate passcode which has to be inserted prior accessing the work apps. Biometric authentication is also supported, and the work challenge can be reset in case of need. Using a work challenge enhances security and provides better separation between the work profile and the personal profile on a device. Admins can customize different features across personal and work profiles [32]:

- Whether work contacts can be viewed in personal apps or not.
- Whether text copied from one profile can be pasted in the other.
- Whether data and files in one profile can be shared to apps in the other profile. In general, personal data can be opened in work apps but not vice versa.

3.8.3 Enterprise application management

With Managed Google Play, admins can distribute public and private applications directly to their employees' devices, whether they're personally owned (BYOD) or company-owned [32].

The following security policies can be enforced for apps:

- **Allowlist**: Lists the apps which can be installed and blocks installation of any unknown app which is not listed;
- **Blocklist**: Blocks installation of all the listed apps but does not prevent the user from installing (potentially malicious) apps which are not in the list;

Installation can be performed in different ways:

- Users can install permitted apps from the company's Managed Google Play store;
- The EMM can push apps to fully managed devices by using the **Device Policy Controller** (**DPC**) and without any user interaction;

Admins can also push updates and security patches through the Managed Google Play.

What is a DCP?

The EMM develops a DPC app that can be used by customers in conjunction with the EMM console and server [41]. The customer deploys the DPC to the user devices that they manage. The DPC acts as the bridge between the EMM console (and server) and the device. An admin uses the EMM console to perform a range of tasks, including configuring device settings and apps. The DPC creates and manages the work profile on the device on which it is installed and can also provision a managed Google Play Account for use on the device itself.

Android Enterprise is no longer accepting new registrations for custom DPCs using the Google Play EMM API. All new EMM solutions should now use Android Management API, which comes with its own DPC provided by Google.

Private apps

Private apps are applications which get published on Managed Google Play and can be installed only by the users within the enterprise that developed them.

These apps can be delivered to users in two ways [32]:

- **Google-hosted** (recommended): The APK is stored on Google data centers, it's scanned for malware, and can be downloaded through a secure connection (i.e., SSL). This option includes a feature called "Google Play app signing", where the company's private key for signing the app gets securely stored and managed by Google, on behalf of the company.
- **Externally-hosted**: The APK is stored on the company's servers and it is only downloadable via intranet or VPN. Requests from Managed Google Play to download APKs from an external server embed a JSON Web Token (JWT) which must be validated by the company, to authenticate the requests.

Either way, app metadata (i.e., title, description, screenshots, and graphics) are always stored by Google Play. Moreover, private cannot be made public due to Google Play policies for malware.

Managed configurations

Managed configurations offer a finer grained control of managed apps, as they allow admins to remotely edit the settings for an app, including its permissions. This can be done by using either the Android Management API or the Google Play EMM API, which also allow to set in-app credentials. Managed configuration options can be updated in Managed Google Play and they can also be edited by the app developer [32].

3.8.4 Google Cloud Identity and Access Management (IAM)

Enterprise-grade access control

Identity and Access Management (IAM) lets administrators authorize who can take action on specific resources, giving full control and visibility to manage Google Cloud resources centrally. For enterprises with complex organizational structures, IAM provides a unified view into security policy across the entire organization, with built-in auditing to ease compliance processes.

How IAM works

In IAM, permission to access a resource isn't granted directly to the end user [42]. Instead, permissions are grouped into roles, and roles are granted to authenticated **principals** (i.e., members). An **allow policy**, also known as an IAM policy, defines and enforces what roles are granted to which principals. Each allow policy is attached to a resource. When an authenticated principal attempts to access a resource, IAM checks the resource's allow policy to determine whether the action is permitted. It's also possible to use **deny policies** to prevent principals from using specific IAM permissions.

A principal can be a Google Account (for end users), a service account (for applications and compute workloads), a Google group, or a Google Workspace account or Cloud Identity domain that can access a resource. Each principal has its own identifier, which is typically an email address. Google Cloud supports federation with most identity providers [43] and with on-premises directories such as Active Directory. Most identity providers enable single sign-on (SSO) for users and groups.

Chapter 4

iOS system and Kernel security

4.1 Hardware security and biometrics

4.1.1 Hardware security overview

Apple devices' security mainly relies on hardware [10]. Components that enforce security are:

- A boot ROM as hardware root of trust (HW RoT);
- Dedicated engines to efficiently execute cryptography algorithms (e.g., AES engine);
- A **Secure Enclave** (from iPhone 5s), with its own boot ROM and crypto engines, which:
  - generates and stores encryption keys for data at rest;
  - securely stores and processes data for biometric authN (i.e., Touch ID, Face ID);

  The Secure Enclave communicates with the external AES engine over a secure channel, so that the keying material used for encryption/decryption is not disclosed to the CPU, nor to the OS. However, since it does not have its own storage, information are securely stored on separate storage which is different from the one used by the CPU and the OS

Thanks to this design, **Data Protection** and **FileVault** technologies can protect files without leaking encryption keys. Thanks to hardware security, iOS devices implement secure boot so that only trusted OS software is loaded during the boot procedure.

Figure 4.1 shows a more detailed diagram of the Secure Enclave architecture design.

4.1.2 Face ID

The Face ID biometric authentication system uses **TrueDepth** camera, i.e., a system that maps the geometry of a user's face [10]. Neural networks are used for determining the attention level of the user, as well as matching percentage, and mitigate physical and digital spoofing. The attention level of the user serves to express the user's will of unlocking the device (e.g., by opening the eyes and watching the devices screen). After user's attention is detected, the TrueDepth camera creates a sequence of depth maps and 2D images of the user's face by using infrared light. All maps and images are digitally signed and sent to the Secure Enclave. To counter spoofing, the sequence gets randomized according to a device-specific random pattern. A portion of the **Secure Neural Engine**, protected within the Secure Enclave, transforms collected data into a mathematical representation to be compared against the stored one.

4.1.3 Touch ID

The Touch ID biometric authentication system reads fingerprint data from any angle and keeps expanding the user's fingerprint map over time [10]. During an authentication attempt, the **advanced imaging array** scans the finger and sends the scan to the Secure Enclave. The fingerprint is analysed by using **subdermal ridge flow angle mapping**, a process that discards some smaller details of the user's finger which would be needed to reconstruct the fingerprint. During enrollment, the Secure Enclave encrypts and stores the fingerprint as a template which never leaves the device (i.e., it's not even backed up). The Secure Enclave communicates with the Touch ID sensor over a serial peripheral interface bus by encrypting and authenticating exchanged data. The encryption key is a session key that gets shared among the enclave and the sensor by using AES key wrapping, with both sides providing a random key that establishes the session key and uses AES-CCM.

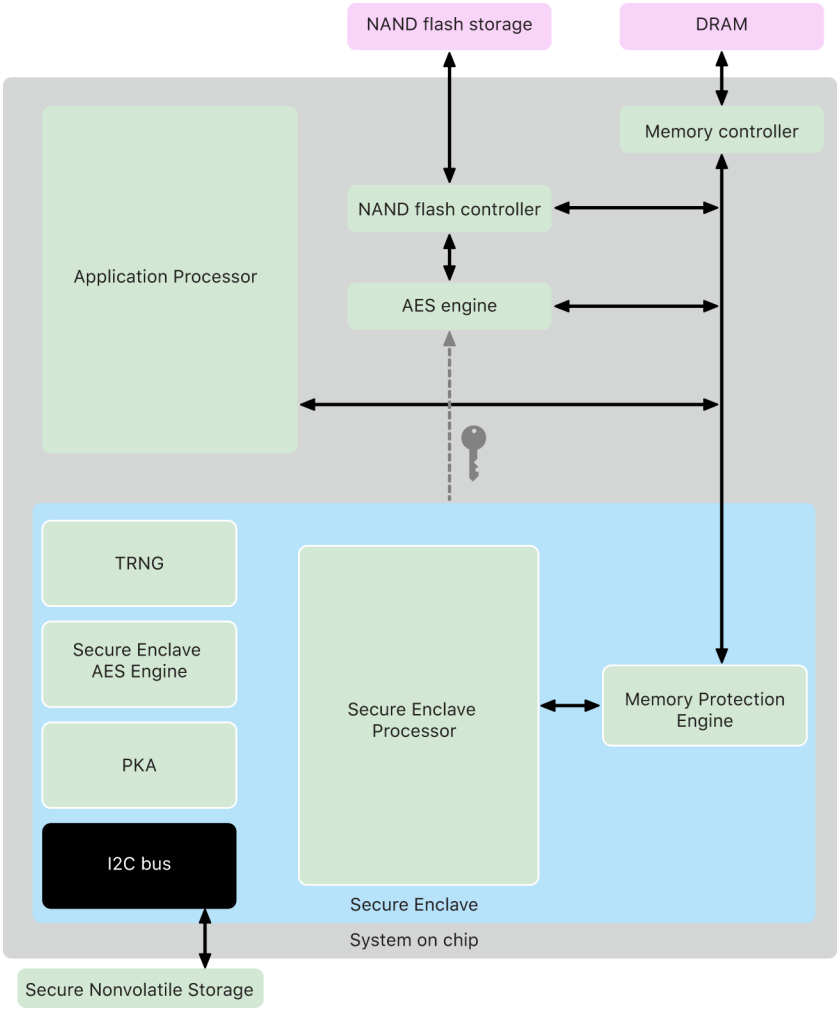


Figure 4.1: Secure Enclave architecture design - ©Apple Inc.

4.1.4 When a device passcode or password is required

While the device can always been unlocked with either the password or the PIN, biometric authentication can't always be used. For example, in these scenarios the use of PIN or password is mandatory [10]:

- Erasing the device;
- Installing configuration profiles;
- Updating the software;
- Viewing or changing passcode settings;

If the device is in one of the following states, PIN or password are once again mandatory:

- There were 5 consecutive failed authentication attempts with biometrics;
- The device has been rebooted or turned on;
- The device hasn't been unlocked for more than 48 hours;
- The device hasn't been unlocked by performing biometric authentication for at least 4 hours;
- The device has been unlocked with mechanisms other than PIN and password for at least 156 hours;
- The device has been remotely locked;
- Others...;

4.2 System security

4.2.1 Signed system volume security

In iOS 15 or later, the **Signed System Volume** (**SSV**) has been implemented to prevent tampering with software components that are part of the OS [10]. SSV introduces cryptographic hashes for protecting the integrity of file-system data and metadata. The SHA256 hash of the data from the internal storage is compared with its expected value in the file-system metadata and data will not be available to the requesting software in case of mismatch. Each hash is stored in the main file-system metadata tree, which has a structure similar to the one of a **Merkle tree**. Hence, the hash stored in the tree's root covers the whole SSV. During iOS updates, the root node signature is verified by the bootloader before allowing the kernel to run.

4.2.2 Secure data connection

On iPhone, data connections through USB cables (and others) can only be performed after authentication (either biometric or with PIN/password) [10]. If the device has been locked for more than one hour or if one hour has passed since the termination of the last data connection, a new successful authentication attempt is required. Data connections are disabled whenever the PIN or password is mandatory to unlock the device. However, this setting can be changed by enabling "always-on data connections". This helps preventing physical attacks that are carried out with, e.g., malicious chargers and/or cables.

4.2.3 Boot process for iPhone devices

The boot procedure can only be completed if the full chain of trust is verified [10]. Components like the kernel (and its extensions), the bootloaders, and the cellular baseband firmware are all digitally signed by Apple, so that it's unlikely that they get compromised. When an iOS device is turned on, the CPU executes code from the **Boot ROM**, which is always trusted as it's embedded in the chip during its manufacturing. This code contains the X.509 certificate of Apple's root certification authority, which is then used to verify that the **iBoot bootloader** has been signed by Apple, before allowing it to load. If iBoot is successfully verified, it will then verify and run the iOS kernel on its own. The Secure Enclave also performs a secure boot that verifies the integrity and authenticity of its operating system, **sepOS**.

4.2.4 Memory safe iBoot implementation

Starting from iOS 14 (A13 Bionic chip or later), the **C compiler toolchain** used to build the iBoot bootloader has been modified to prevent memory-corruption vulnerabilities exploitation [10]:

- Pointers carry runtime information regarding their type that's verified when performing casts, preventing type errors;
- Pointers include bounds information that are verified upon accessing the memory, to prevent buffer overflows;
- Dynamic memory allocations have been segregated by static type to prevent type errors caused by use after free vulnerabilities exploitation;
- Heap data has been separated from its corresponding metadata, to detect errors like segmentation faults, e.g., caused by using twice the `free` function on the same memory location (this helps preventing heap exploitation);

4.2.5 BlastDoor for Messages and IDS

Starting from iOS 14, a security feature called **BlastDoor** was introduced [10]. BlastDoor makes it harder for an attacker to mount 0-click attacks (i.e., which require no user interaction) by exploiting Messages, Apple **Identity Services** (**IDS**), and other possible attack vectors. This is achieved by properly performing data validation and by employing sandbox restrictions.

4.2.6 Lockdown Mode

Starting from iOS 16, **Lockdown Mode** has been introduced. Lockdown Mode is an extreme protection mechanism specifically designed for people (e.g., VIP) who may be targeted by advanced malware (e.g., Pegasus). Some of the device's features will be changed when this mode is active [10, 44]:

- **Configuration profiles**: Mobile Device Management (MDM) and configuration profiles can't be installed;
- **Device connections**: Authentication is always required to connect the device to an accessory or a computer;
- **Messages**: Most message attachments are blocked and links (as well as link previews) are unavailable;
- **Photos**: Location information is always excluded from pictures upon sharing;

- **Web browsing**: Certain web technologies are disabled (e.g., some fonts, images, JavaScript), which may cause a worse (but securer) user experience;
- Others...;

4.3 Operating System Integrity

iOS employs many techniques to mitigate vulnerabilities exploitation and preserve the integrity of software components. However, these security features are not always available for all devices, as they require the proper hardware to be implemented [10]:

Feature	SoCs
Kernel integrity protection	A10-17
Fast permission restrictions	A11-17
System coprocessor integrity protection	A12-17
Pointer Authentication Codes	A12-17
Page Protection Layer	A11-17
Secure Page Table Monitor	A15-17

Table 4.1: iOS platform supported security features

Note: SPTM is supported only on A15, A16, and A17 and replaces PPL on supported platforms.

4.3.1 Kernel Integrity Protection

Kernel Integrity Protection (**KIP**) is enabled after the kernel is initialised [10]. iBoot uses a protected physical memory region, provided by the memory controller, to load the kernel and its extensions. Upon completion of the loading process, no write operations are allowed to that memory region anymore. The **Memory Management Unit** (**MMU**) is configured to prevent mapping privileged code and writeable memory portions to, respectively, physical memory outside the protected region and kernel memory. After the boot process is complete, the all the hardware which has been used for enabling KIP gets locked to prevent being reconfigured.

4.3.2 Fast Permission Restrictions

The **Fast Permission Restrictions** (**FPR**) hardware primitive has been introduced after the A11 SoC series [10]. FPR uses hardware CPU registers called "**APRR registers**" to remove the execute permissions from memory with a small overhead on a per-thread basis. These registers help mitigate attacks which are started by just-in-time compiled code, because the memory can't be executed at the same time read and write operations are being performed.

4.3.3 System Coprocessor Integrity Protection

Coprocessor firmware integrity is really important, as it handles tasks like the Secure Enclave, the image sensor processor, and others. Hence, a **System Coprocessor Integrity Protection** (**SCIP**) system, which works similarly to KIP, is implemented [10]. During the boot procedure, iBoot loads each coprocessor's firmware into a protected memory region (separate from the KIP reserved region). Each coprocessor's memory unit is configured to help prevent executable mappings outside the protected memory region and writeable mappings inside the protected region itself. The Secure Enclave OS (seOS) is used to configure SCIP at boot time. Upon completion of the boot, the hardware used to enable SCIP is locked to prevent being reconfigured.

4.3.4 Pointer Authentication Codes

Pointer Authentication Codes (**PACs**) are used to prevent memory corruption vulnerabilities exploitation [10]. All function pointers and return addresses from the system software and apps are signed by using five 128-bit long secret values.

Item	Key	Salt
Function Return Address	IB	Storage address
Function Pointers	IA	0
Block Invocation Function	IA	Storage address
Objective-C Method Cache	IB	Storage address + Class + Selector
C++ V-Table Entries	IA	Storage address + Hash (mangled method name)
Computed Goto Label	IA	Hash (function name)
Kernel Thread State	GA	None
User Thread State Registers	IA	Storage address
C++ V-Table Pointers	DA	0

Table 4.2: PACs signatures computation data

The signatures are stored in the unused padding bits of the 64-bit pointers and are always verified before using the pointers. After verifying the signature, padding is restored to ensure the proper functioning of each pointer. If signature verification fails, the process is aborted. This security measure mitigates attacks like ROP.

4.3.5 Page Protection Layer

The **Page Protection Layer** (**PPL**) prevents user space code from being modified after verifying the code signature [10]. PPL uses both KIP and FPR to make sure it is the only one able to write protected pages containing user code and page tables. This system works even if the kernel gets compromised.

4.3.6 Secure Page Table Monitor and Trusted Execution Monitor

Secure Page Table Monitor (**SPTM**) and **Trusted Execution Monitor** (**TXM**) work together to prevent page tables modification for both user and kernel processes, even if the attacker has kernel privileges [10]. That is possible because the SPTM uses a higher privilege level than the kernel, while the TXM is only used to enforce code execution policies and it is hence lower-privileged. Thanks to this separation of privileges, if the TXM gets compromised, the SPTM can't be compromised too. Starting from A15 SoCs, SPTM and TXM replace PPL.

4.4 Encryption and Data Protection

4.4.1 Overview

iOS devices use an encryption technique which is called "**Data Protection**" [10]. Key chains are always rooted in the HW RoT of the Secure Enclave and encryption keys are never disclosed to untrusted components. The kernel also employs **sandboxing** to restrict applications from accessing data which is stored on the device, and the **Data Vault** system is used for restricting apps from accessing data that belongs to another app.

4.4.2 Passcodes and Passwords

Time delays are used to mitigate brute-forcing PIN and/or password [10]. **Data Protection** is automatically enabled as soon as a PIN/password is set up. PIN or password provides entropy for some encryption keys, so that data which are protected with those keys can't be decrypted without knowing it. Time delays are calibrated so that it would take more than five years to brute force six-digits PIN. The delays are enforced by the Secure Enclave and the timer starts over again upon device reboot. An interesting, yet dangerous, setting can be enabled to erase all data on the iPhone after 10 consecutive failed authentication attempts. This setting is also available to admins that install an MDM profile on a supervised device.

4.4.3 Data Protection

Data Protection protects data stored in the device's flash storage [10]. While system apps use Data Protection by default, third-party apps automatically receive this protection. Data Protection uses a hardware-enforced key hierarchy and assigns a class to each stored file. The file can only be decrypted if the corresponding class keys are unlocked. The Apple File System even allows different portions of the same file to be encrypted with different keys. Every time a new file is created, it is encrypted by the AES engine while it's getting written to the storage, with a 256-bit **per-file key** generated at the moment. Starting from A14 SoCs, AES-256-XTS is used and the per-file key is given as input to a Key Derivation Function (NIST Special Publication 800-108 [45]) to produce a 256-bit tweak and a 256-bit cipher key. However, from A9 to A13 SoCs, AES-128-XTS is used instead and the per-file key is just split in half to get a 128-bit tweak and a 128-bit cipher key, without using a KDF.

The per-file key gets wrapped and stored in the file's metadata. Metadata, which also contain information regarding the class of the file, are encrypted by using a different key called "**file system key**". When a file is opened, metadata are decrypted and the per-file key is unwrapped and sent to the AES engine for decrypting the file while it's being read from the storage.

The Secure Enclave manages wrapped keys and never discloses them. That's only possible because the Secure Enclave negotiates an ephemeral key with the AES Engine at startup. So, when a key gets unwrapped inside the Secure Enclave, it is then re-wrapped with that ephemeral key before being sent to the CPU.

4.4.4 Data Protection Classes

Here the classes which can be assigned to a file when it's first created [10]:

- **Complete Protection** (`NSFileProtectionComplete`): The class key is protected with a key derived from the user PIN/password and the device UID. After the device gets locked, the decrypted class key is discarded and must be derived again by performing authentication.
- **Protected Unless Open** (`NSFileProtectionCompleteUnlessOpen`): Files that need to be written while the device is locked are protected by a per-file key which is encrypted with an ephemeral key established with ECDH over the Curve25519 (NIST SP 800-56A [46]).
- **Protected Until First User Authentication** (`NSFileProtectionCompleteUntilFirstUserAuthentication`): This class behaves like the Complete Protection class, except that the decrypted class key isn't removed from memory when the device is locked. This is the default class for all third-party app data and it helps mitigate attacks that exploit reboot.
- **No Protection** (`NSFileProtectionNone`): This class key is protected only with the UID, and is kept in the Effaceable Storage.

4.4.5 Keybags for Data Protection

Class keys for Data Protection are stored in **keybags** [10]:

- **User keybag**: It is a `.plist` file that stores the wrapped class keys used in normal operation of the device. When the PIN is entered, the complete protection class key is loaded from this keybag and unwrapped. This keybag is protected by the user's PIN.
- **Device keybag**: It is used for storing the wrapped class keys used for operations involving device-specific data. Class keys from this keybag are used to wrap per-file keys. This keybag is protected by the user's PIN.
- **iCloud Backup keybag**: It is created when an encrypted iCloud backup is performed. All the class keys in this keybag are asymmetric and use Curve25519. Another asymmetric keybag is also used to protect the backed-up keychain for iCloud Keychain recovery.

4.4.6 Protecting app access to user data

iOS devices prevent apps from accessing a user's personal information without permission by using **Data Vault** [10]. In iOS Settings, users can see and manage the permissions of apps. Starting from iOS 13.4, all third-party apps automatically have their data protected from unauthorised access in a Data Vault. If the user signs into iCloud, apps are granted access by default to the **iCloud Drive**, unless they're explicitly denied by the user from the Settings. Apps and accounts installed by an MDM profile are restricted from communicating with apps installed by the user.

4.4.7 Protecting keys in alternate boot modes

Access to user data during alternate boot modes like **Device Firmware Update** (**DFU**) and Recovery mode is restricted [10]. These modes are often used during forensics analyses as a workaround to acquire part of the device's data without knowing the PIN [47].

Feature	SoCs
Recovery: All Data Protection Classes protected	A10-A17
Alternate boots of DFU mode, Recovery, and software updates: Class A, B, and C data protected	A11-A17

Table 4.3: Alternate boot modes data protection

The Secure Enclave's AES Engine embeds lockable software seed bits which are used by the KDF when creating new keys. The seed bits are used in different manners on different SoCs:

- Starting from A10 SoC, a seed bit is set for keys that require the user's passcode. From iOS 13, all user data are inaccessible, by using an additional seed bit, when the device is booted into Diagnostics mode. This bit is used for accessing the **media key**, that is needed for decrypting the metadata and, consequently, the content of all files.
- On A12 SoCs, the Secure Enclave's Boot ROM locks the passcode seed bit if the DFU or Recovery mode is entered.

Restoring a device after it enters DFU mode ensures that it is returned to a state where it only contains trusted signed code.

4.4.8 Sealed Key Protection (SKP)

Starting from A11 SoCs, devices support **Sealed Key Protection** (**SKP**) [10]. This security measure ensures that all cryptographic material is unavailable off device. SKP is implemented by **hardware registers** that exist at a lower and distinct level than the Secure Enclave and provides extra protection to the keys which are needed to decrypt user data which is independent from the enclave itself. If Data Protection is supported, the key which is derived from the user's password, the long-term SKP key, and the Hardware key 1 (where 1 is the UID for the Secure Enclave), is used to protect the user keybag.

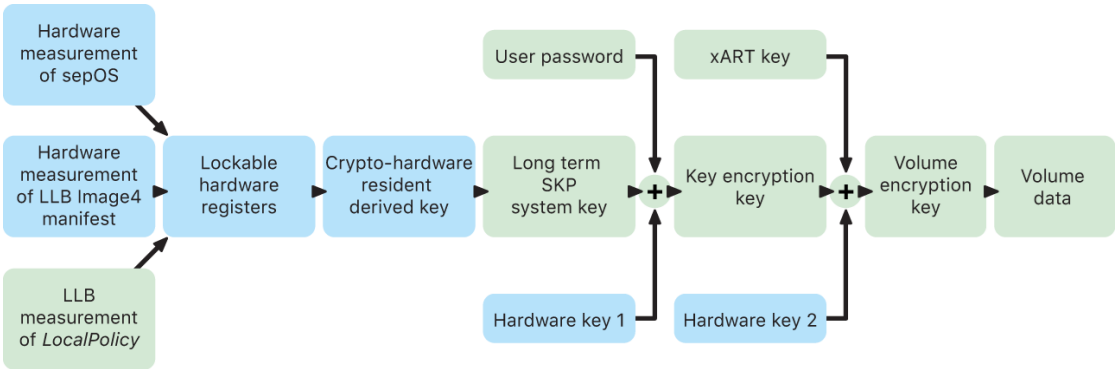


Figure 4.2: Sealed Key Protection scheme - ©Apple Inc.

4.5 Security of runtime process in iOS

In iOS, runtime security is enforced with application sandbox, declared entitlements, and Address Space Layout Randomization (ASLR) [10, 48].

4.5.1 Sandboxing

All third-party apps are sandboxed [10]. Upon installation, an app is assigned to a **unique and random home directory** where to store its files. Third-party apps must use iOS services to access data of other apps, as well as system files. All third-party apps run as the **mobile** user, which is non-privileged and the OS partition is mounted as read-only. Moreover, APIs are specifically designed to help preventing privilege escalation attacks.

4.5.2 Use of entitlements

Entitlements are key-value pairs that are digitally signed into an app and allow to perform authentication beyond runtime factors (e.g., UID) [10]. They are generally used by system apps and daemons to perform privileged operations without having the process running as root. If a system app or daemon gets compromised, it will not be able to perform privilege escalation attacks because of this security measure.

4.6 iMessage

iMessages are texts, photos, or videos that can be sent send to another iPhone, iPad, iPod touch, or Mac over Wi-Fi or cellular-data networks [10, 49].

4.6.1 iMessage security overview

iMessage uses the **Apple Push Notification service** (**APNs**) and the content of messages and attachments is protected by **end-to-end encryption** [10]. When iMessage is turned on, the device generates:

- Encryption keys: RSA 1280-bit long key and EC 256-bit long key (NIST P-256 curve);
- Signing keys: ECDSA 256-bit long keys;

All private keys are saved in the device's keychain and they're only available after unlocking the device for the first time. On the other hand, all public keys are sent to the Apple Identity Service (IDS), where they are associated with the user's phone number (or email address), along with the device's APNs address.

Despite it providing secure end-to-end communication, iMessage has always been used by attackers for installing malware on target devices. The attack vector is often a crafted attachment that exploits a vulnerability in the iMessage app, as further discussed in Chapter 8.

4.7 iCloud

4.7.1 iCloud encryption

There are two modes for encrypting data stored on iCloud [10, 50]:

- **Standard data protection** (default setting): User's iCloud data is encrypted and encryption keys are securely stored in Apple data centers;
- **Advanced Data Protection**: If Advanced Data Protection is turned on, only the devices which are trusted by the user can access the encryption keys [51];

System apps store data on iCloud by using the CloudKit frameworks and APIs [52]. CloudKit allows developers to store key-value data and assets (e.g., images and videos) in:

- Public databases, accessible to everyone and not encrypted (used for generic assets);
- A private database, available for each distinct user to store their data;

Databases are grouped in **containers**. Each container's private database is protected by a key hierarchy, rooted in an asymmetric **CloudKit Service key**, unique to each user and generated on their trusted device. When writing data to CloudKit, record keys are generated on the user's device and wrapped to the corresponding key hierarchy before uploading data on iCloud.

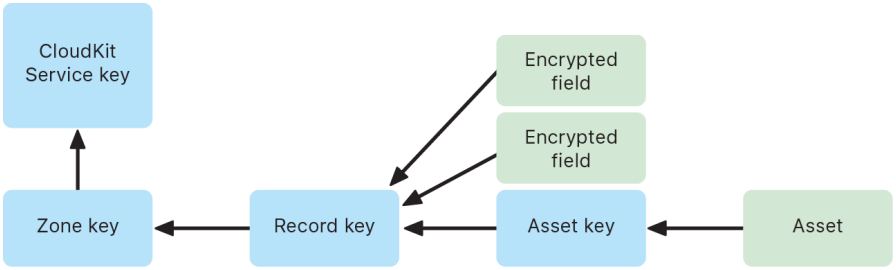


Figure 4.3: iCloud key hierarchy - ©Apple Inc.

The security of encrypted data in CloudKit relies on the security of the corresponding encryption keys. There are two classes of encrypted service keys:

- **End-to-end encrypted**: Created locally on a user's trusted device and transferred to the user's other devices using **iCloud Keychain security**, which prevents anyone from accessing the data. If access to iCloud Keychain and its corresponding recovery mechanisms is lost there's not way to recover the data.
- **Available-after-authentication**: Stored in iCloud HSMs in Apple data centers, they can be accessed by some Apple services. When a user signs in to iCloud on a new device and authenticates with their Apple ID, these keys can be accessed by Apple servers. This is the case of the Photos service.

4.7.2 Security of iCloud Backup

iCloud Backup can be disabled by IT administrators using mobile device management (MDM) configuration profiles [10]. By default, the **iCloud Backup service key** is backed up to iCloud HSMs in Apple data centers as an "available-after-authentication" service key. When Advanced Data Protection for iCloud is enabled, the iCloud Backup service key is protected with end-to-end encryption. The backup consists of a copy of the user's files and the **iCloud Backup keybag**, which is protected by a random key (stored in the backup set as well). Since the user's iCloud password isn't used for encryption, changing it won't affect the backups.

The following content is backed up using iCloud Backup:

- Contacts, calendar events, reminders, and notes;
- Device settings, App data, Home Screen and app organization;
- Photos and videos stored on the user's devices. However, starting from iOS 8.1, if iCloud Photos is enabled, these assets will already be stored in iCloud they won't be included in backups;
- SMS and MMS messages, which may requires physical access to the SIM card that was in use when performing the backup;
- Others...;

iCloud Backup is also used to back up the local device keychain, encrypted with a key derived from the Secure Enclave UID root cryptographic key of the device (unique for each device). In this way, only the device which created the backup can restore the database.

4.8 Updates and security patches

Apple has long supported prior iPhone models with software updates, even phones that are several years old. While it was always assumed older models would get some updates, an exact timeline wasn't ever given. Older devices seem to receive software updates and security patches for about 5 years [53], even though this time window has been extended sometimes. The iPhone 6S came out in 2015 and shipped with iOS 9 but received regular software updates through iOS 15, which came out in 2021. It even got a security update in 2023, which meant it got nearly eight years of security updates. According to the official updates history [54], it looks like Apple releases security updates almost every month, with the last update for iOS 17.5.1 being released on the 20th of May 2024.

4.8.1 Rapid Security Responses

Rapid Security Responses [55] are a new type of software release that deliver important security improvements between software updates (e.g., improvements to the Safari web browser, the WebKit framework stack, or other critical system libraries). They may also be used to mitigate some security issues more quickly, such as issues that might have been exploited or reported to exist "in the wild". New Rapid Security Responses are delivered only for the latest versions of iOS, starting from iOS 16.4.1. By default, the device automatically applies Rapid Security Responses but such setting can be disabled. If the user chooses to turn off this setting or not to apply Rapid Security Responses when they're available, the device will receive relevant fixes or mitigations when they're included in a subsequent software update.

4.9 Mobile Device Management in iOS

4.9.1 How MDM works

MDM capabilities are built on OS technologies, such as configurations, and **Apple Push Notification service** (**APNs**) [10]. By installing an MDM profile on a device, it is possible to remotely configure and update its settings and apps, as well as remotely wipe the device's memory or lock the device itself. Starting from iOS 13, an enrollment option has been added for those scenarios where a **BYOD** policy is being adopted by a company. Enrollment provides more autonomy for users on their own devices and separates private user data from enterprise data by means of cryptography. Starting from iOS 17, a similar data separation mechanism has been added for account-driven Device Enrollments as well. The following enrollment types are supported:

- **User Enrollment**: Designed for devices owned by the user. Managed Apple IDs are required to initiate the enrollment and authentication by the user is mandatory for terminating the process. Managed Apple IDs, used by managed apps only, can be used alongside the user's personal Apple ID.
- **Device Enrollment**: The user can manually enroll their device and then manage options like remote memory wipe. When a user removes an enrollment profile, all configurations, settings, and managed apps based on that enrollment profile are removed. Similar to User Enrollment, Managed Apple IDs are integrated in this profile and kept separated from the user's personal Apple ID and data.
- **Automated Device Enrollment**: It lets organizations configure and manage devices from the moment the devices are removed from the box. These devices are known as **supervised**, and users have the option to prevent the MDM profile from being removed by the user. Automated Device Enrollment is designed for devices owned by the organization.

Administrators can restrict users from specific apps and services. Restrictions are sent to devices in a **restrictions payload**, which is part of a configuration.

4.9.2 Configuration enforcement

Configurations are the primary way that an MDM solution delivers and manages policies and restrictions on managed devices [10]. A **configuration** is an XML profile or json formatted file following a certain structure and consists of payloads that load settings and authorization information onto Apple devices. These files can be created by an MDM solution which uses an enrollment profile. Configuration profiles can be digitally signed and encrypted. Configuration profiles for iOS are encrypted using the **Cryptographic Message Syntax** (**CMS**) specified in RFC 5652 [56], supporting 3DES and AES128.

Some possible settings which could be specified by configuration payloads are:

- Passcode and password policies;
- Restrictions on device features (e.g., disabling the camera);
- Network and VPN settings;
- Account settings;
- Credentials and identities;
- Certificates;
- Software updates;
- Others...;

4.9.3 Managed Lost Mode and remote wipe

Starting from iOS 9, a supervised device which gets lost or stolen can be locked by an MDM admin if the "Managed Lost Mode" is activated [10]. If that happens, the user will also be logged out from the device. The admin can also get the device's location, even if Location Services are disabled. iPhone devices can also be erased remotely by an administrator or user. When a **remote wipe command** is triggered by MDM or iCloud, the device sends an acknowledgment back to the MDM solution and performs the wipe. Remote wipe is not possible:

- With User Enrollment;
- Using Microsoft Exchange ActiveSync if the device is supervised;
- Using Microsoft Exchange ActiveSync if the account was installed with User Enrollment;

4.10 Screen Time features and security

Screen Time is an iOS feature which is generally used by parental control apps, as further discussed in Chapter 12. Screen Time is available in iOS 12 or later and its main features are [10]:

- View usage data;
- Enforce additional restrictions;
- Set web usage limits;
- Set app limits;
- Configure Downtime;

Children are informed when their parents turn on Screen Time. Parents can also set a passcode yo prevent their children from changing the Screen Time settings. Usage data and configuration settings, used to monitor the child's activities, are transferred between the parent's and child's devices with the end-to-end encrypted Apple IDS protocol.

Chapter 5

Google Play Store security

5.1 App code signing

APKs must always be signed by developers and installation of unsigned apps gets blocked. Upon app installation, the package manager verifies the digital signature of the APK thanks to the certificate included in the APK itself. Apps can be signed by third parties (e.g., OEM) or they can even be **self-signed** by developers who create their own certificate without needing any external approval. Moreover, Android doesn't perform CA verification for app certificates [57].

5.2 Google Play Protect (GPP)

Before apps are allowed to be published on the store, they are automatically scanned by the **Google Play Protect** tool [58]. This tool performs both static and dynamic analysis of apps, looking for any known malicious behavior, and then classifies them on a scale from safe to harmful [59]. Harmful apps are blocked. Apps which are difficult to classify are considered as **Potentially Harmful Applications** (**PHAs**) and they're manually reviewed by the Android security team. Misbehaving developers are banned from the store and can't publish new apps.

Google Play Protect uses Google's ML algorithms to detect PHAs. Thanks to Google's app database, these algorithms are able to recognise when an application performs suspicious actions like, e.g., unauthorised data access, access to malicious websites, installing other apps, or even bypassing Android's security measures [59].

5.2.1 GPP cloud-based security

Before an app becomes available for download in the store, it gets analyzed along its developers by using automated detection mechanisms (e.g., ML) and human analysts [59]:

- **Developer review**: Google Play's internal risk engine analyzes all information of each developer (e.g., Google account, actions, etc...) and, if it detects something suspicious, the developer's transactions are manually reviewed by analysts to make sure the developer is compliant to the Developer Distribution Agreement [60].
- **Internal app review**: Compliance with Google Play policies is checked for each app, which gets analysed by Google's application risk analyser. Security analysts may manually review an app if it gets flagged by the risk engine.

5.2.2 GPP machine learning model training

ML models of GPP are trained by using [59]:

- **Static analysis**: "The app's code is analyzed and the features are extracted and compared against expected good behavior and potential bad behavior".
- **Dynamic analysis**: Applications are run and their behaviour is checked.
- **Signatures**: Used to compare each app against a database of known malicious apps.
- **Heuristic and similarity analysis**: Apps are compared with each other, looking for trends that identify malicious apps.
- **Developer relationships**: GPP checks if an app's developer has been previously associated with the creation of potentially harmful apps.
- **Third-party reports**: Submitted by third party researches which evaluate apps.
- **SafetyNet**: "A privacy preserving sensor network that spans the Android ecosystem and identifies apps and other threats that can harm devices".

5.2.3 GPP on-device protection

GPP checks all apps on a device, regardless of where the app was downloaded [61]:

- **Daily PHA scan**: GPP scans devices once everyday and notifies the user if a PHA is found. The user can then decide whether to remove the app or not. However, if GPP finds out that the app has no benefit to the user, it can automatically remove it and block all future installs. During daily scanning, GPP will also contact Google servers for further verification if a PHA is detected.
- **On-demand PHA scan**: Users can start a full-device scan on their own, if they want to. The device will contact Google servers to retrieve the latest info and it will then scan all apps on the device.
- **Offline PHA scan**: GPP also implements offline scanning to prevent known PHAs from being installed offline (e.g., with ADB tool via USB cable). A full scan will be issued for when the device regains network connectivity.
- **Automatically disable PHAs**: While the most harmful PHAs are automatically removed from the device, less dangerous PHAs are just disabled. In other words, they become unusable but they're kept on the device with all their data. In case that happens, the user will be notified and will be able to take action (i.e., re-enable the app or remove it).

5.2.4 GPP real-time protections for non-Play installs

GPP offers protection from apps which are installed from **third-party stores** [61]:

- When a user tries to install an app, GPP checks it against known malicious samples. Further checks are performed on-device by using ML and other techniques.
- If the app is identified as malicious or suspicious, either users will be warned or the installation will be automatically blocked (in extreme cases).

GPP also offers protections from unknown threats:

- When no malicious code is recognised in a third party app samples, the user can perform a real-time code-level scan of the app to extract signals and update them on Google servers for evaluation. The user will be later notified about the results of the analysis.

5.3 Threats through Google Play Store

According to [62], Google has detected and blocked many Android malware families over the years. However, incidents involving the Google Play Store still occur. There are several techniques by which attackers can upload and propagate malware through the store, such as:

- **Dynamic code loading**: Consists in loading and executing code at runtime. On one hand, this technique can be used for performing runtime updates but, on the other, it can also introduce security risks. As a matter of fact, attackers use dynamic code loading to load and execute malicious code on a target device. Runtime-loaded libraries are not accessible by GPP. Hence, this technique may allow to avoid malware detection.
- **Incremental malicious updates attack** (**IMUTA**): used to gradually add malicious code to an application over time by using a dropper app which appears legitimate at first, but later downloads and installs malicious code.
- **Code obfuscation**: Involves the usage of one or more techniques, e.g., variables renaming, junk code addition, or code encryption, to hide the real purpose of the malicious code.
- **Repackaging**: Consists in decompiling a legitimate app, adding malicious code to it, and then recompile it and resign it. This strategy easily tricks users as the application, available on the Play Store, looks legitimate.

5.3.1 A powerful malware evasion technique

In 2021, a research group proposed a malware evasion technique which successfully avoided detection by GPP and VirusTotal's anti-malware tools (AMTs) by combining obfuscation, encryption and steganography [63]:

1. The malicious application was obfuscated (e.g., with ProGuard, AAMO, or AVPass).
2. AES was used for encrypting the malicious code, along with some extra data chunks regarding the target PNG image where to embed the code, plus the padding bytes.
3. Steganography was used to embed the malware into the image.
4. Finally, the resulting image was placed into the assets folder of a legitimate application, which only served as a wrapper to the malware.
5. After installation of the legitimate app, the `onClick` event listener was used to trigger the inverse process and the malicious APK was dynamically loaded to the phone storage and installed as an update of the wrapping application at run-time.

This technique was applied on the Dendroid spyware, with an original detection rate of 32/56 on VirusTotal, and it dropped the detection rate to 0. Moreover, the application was also successfully installed on the victim's device and it performed as intended.

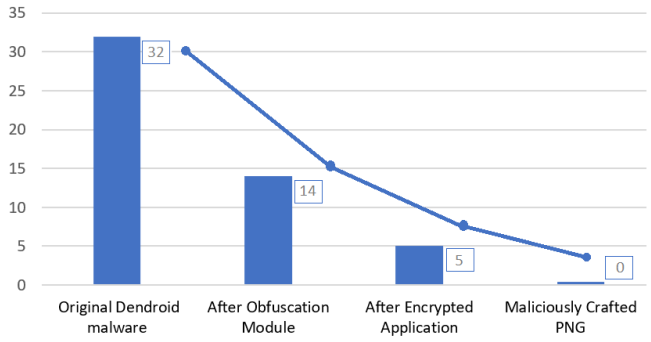


Figure 5.1: Stagewise VirusTotal detection rate

Chapter 6

App Store security

6.1 App security in iOS

iOS does not allow installation of third party apps from websites and untrusted stores [10]. All apps must be downloaded from the App Store, where all apps come from identified developers and have passed automated and human review. Executable memory pages of an app undergo signature verification while they're being loaded. After an app is verified to be from an approved source, iOS enforces security measures to prevent it from compromising the system or other apps.

6.2 About App Store security

The App Store keeps track of developers who have agreed to Apple guidelines, and makes sure that applications are digitally signed before being published [10,64]. Each app (and its updates) is reviewed to evaluate if it meets Apple requirements for privacy, security, and safety. Moreover, apps designed for children must follow strict guidelines for data collection and security, and must be integrated with iOS parental control features.

App Store security protections include:

- **Automated scans** for known malware: To prevent malware spreading through the store.
- **Human reviews**: Consist in reviewing the app description, marketing text and screenshots. Prevents scenarios where attackers misrepresent a malware as a popular app, or claim to offer features that aren't actually provided.
- **Manual checks**: To ensure the app does not request unnecessary access to sensitive data. Apps designed for children are further checked.
- **User reviews**: They are a valuable resource and the App Store is constantly improving to detect fraudulent reviews.
- **Processes for correction and removal**: If an app gets published on the App Store but is then discovered to violate guidelines, the app can be removed from the store and users who downloaded it will be notified of the its malicious behavior.

Unfortunately, security measures are not designed to protect a user from some wrong choices they could be tricked into making.

6.3 App code signing process

6.3.1 Mandatory code signing

Executable code for iOS devices must be signed with an **Apple-issued certificate** [10, 65]. System apps like Safari are already signed by Apple, but third-party apps must also be signed by using a certificate issued by Apple. In this way, the chain of trust is extended to apps.

6.3.2 How developers sign their apps

There are two steps to sign code for iOS apps [10, 65]:

1. **Certificate validation**: To develop apps, developers must register with Apple and join the Apple Developer Program. So, the real-world identity of each developer is verified by Apple before issuing any certificate [66]. This certificate will then allow the developer to sign apps and publish them on the store.
2. **Code signature validation**: iOS developers may embed third party frameworks in their apps. To prevent loading of unauthorised third-party code, the system digitally signs all the dynamic libraries that a process links against at launch time. Signatures are then verified by using a **Team ID** (i.e., a 10 characters long alphanumeric string), which is extracted from an Apple issued certificate. So, a program can only use dynamically linked libraries that either belong to the system or that have the same Team ID of its code signature. System apps don't have a Team ID and hence can only use system libraries.

6.3.3 Verifying proprietary in-house apps

Companies that want to develop and distribute their iOS apps to their employees must apply to the **Apple Developer Enterprise Program** (**ADEP**), and must register to obtain a **provisioning profile** [10,65]. The profile permits in-house apps to be executed on all the (authorised) devices where it gets installed. Apps installed through an MDM profile are implicitly trusted because the relationship between the company and the device is already established. Even in this case, in-house apps must be reviewed and allowed to run by Apple.

Chapter 7

Android attacks and countermeasures

7.1 Vulnerabilities overview

As stated by [62], in recent years Android has been reported with several vulnerabilities that may be exploited by an attacker to:

- Gain access to sensitive information;
- Install malware or other malicious software;
- Launch attacks on other devices or systems;

Since 2009, around 5000 vulnerabilities have been discovered, with the most common ones causing buffer overflows and code execution:

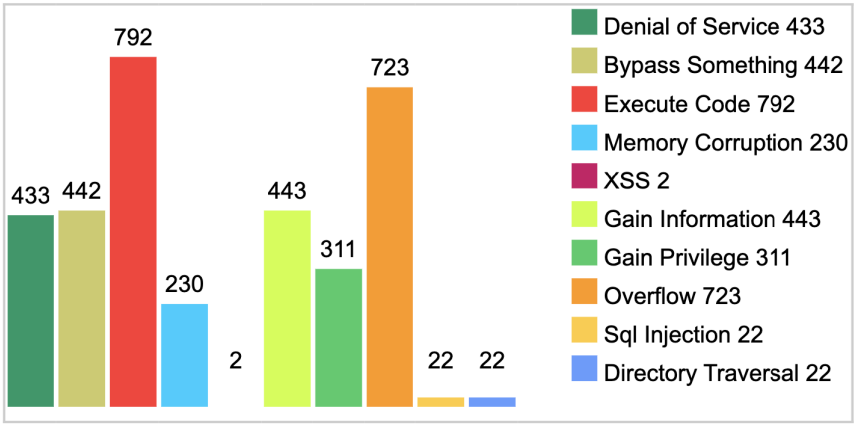


Figure 7.1: Classification of Android vulnerabilities by type

As observed by [67], vulnerabilities are mostly present in the Linux kernel and in the native libraries of Android:

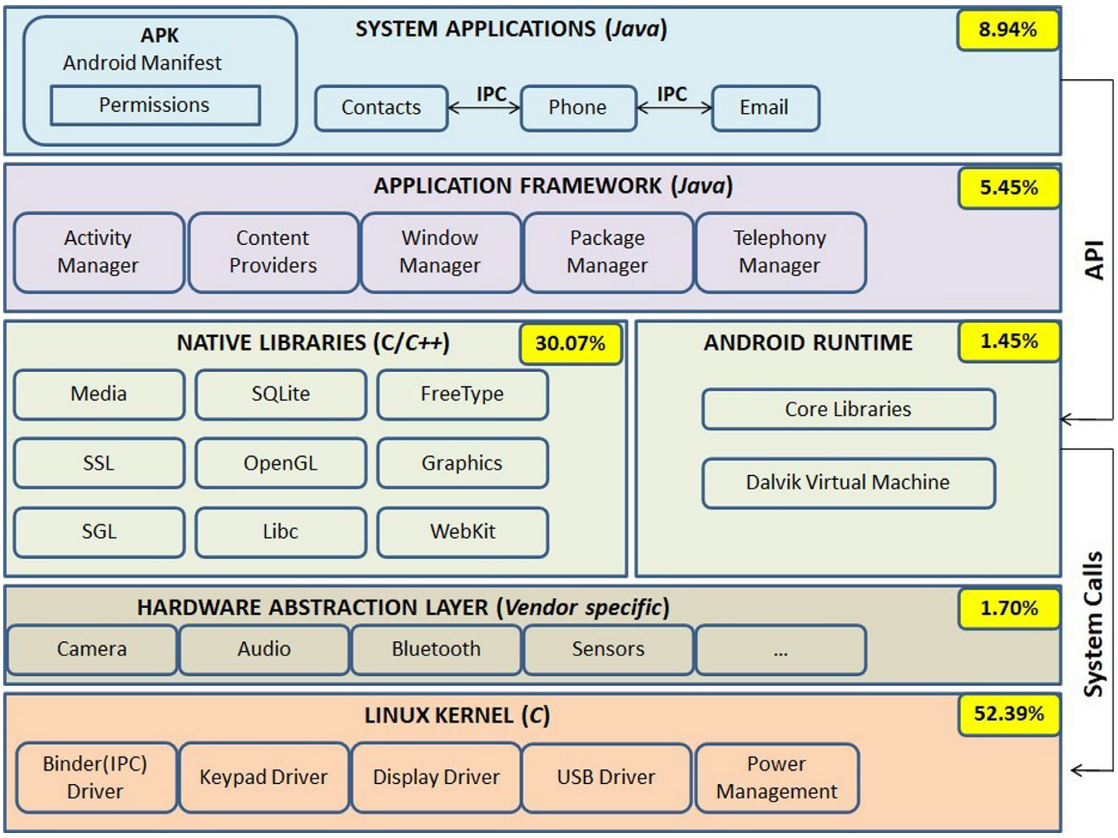


Figure 7.2: Distribution of vulnerabilities in Android framework layers

7.2 Macro categories of attacks

As described by [68], these are the most relevant categories of attacks against Android:

- **Hardware-based**: target HW components vulnerabilities, which are very difficult to patch, as patches are available with the distributors and carriers only;
- **Kernel-based**: aimed at getting root access to the system, they exploit vulnerabilities in the kernel components like Android Runtime (ART) and device drivers. Even though Android has come out with many security features (e.g., Application Sandbox, secure IPC, Encrypted File System, etc...), many vulnerabilities continue to be exposed and others can be found in all these security features as well;
- **HAL-based**: they target the interfaces exposed by the HAL for camera, microphone, GPS, etc...;
- **Applications-based**: they exploit vulnerabilities of default applications, as well as the applications installed into the system from Google Play Store and third-party stores;

7.2.1 Hardware-based attacks

Also according to [68], most of the vulnerabilities exist because of the preinstalled vulnerable components in the device provided by the manufacturers. However, other vulnerabilities can also be found in repaired components (e.g., touchscreen, NFC reader, etc...) from unauthorized vendors. To prevent this from happening, the system should be able to check the integrity of the communication between the replaced component and the CPU as much as the component driver's source code should not implicitly trust the component.

- **Rowhammer attacks**: By hamming a specific row in memory inside a DRAM chip, it is possible to cause a small amount of electrical current to leak from the target location, resulting in a change in the state of memory bit or memory leak [69] (e.g., could be used for leaking the base address of a dynamic library). It requires the installation of a malicious app on the target device and access to the device GPU. The GPU cache can be easily administered by an attacker to get access to target rows, which is otherwise tricky if the CPU is used instead. Mobile devices could be targeted by using malicious JavaScript code in the browsers [70].
- **Sensor-based attacks** [62]: Designed to exploit vulnerabilities of the device's sensors (e.g., GPS, accelerometer, etc...) to extract sensitive information (e.g., device's location, audio/video recordings), gain unauthorized access to the device (e.g., biometrics attacks) or even cause DoS (e.g., battery-draining attacks).
- **Side-channel attacks** [62]: Used to extract sensitive information (e.g., encryption keys, passcodes, and personal information) from a covert channel. This may include analyzing physical characteristics of the device like electromagnetic radiation, temperature and power consumption while the device is operating.

7.2.2 Kernel-based attacks

Privilege escalation attacks

Attacks targeting **root privilege** may exploit the following vulnerabilities [68]:

- CVE-2016-0728 [71]: a use-after-free flaw was found in the way the Linux kernel's key management subsystem handled keyring object reference counting in a certain error path of the `join_session_keyring()` function. A local, unprivileged user could use this flaw to escalate her privileges on the system.
- CVE-2013-6282: `get_user()` and `put_user()` functions do not validate certain addresses, allowing attackers to read or modify the contents of arbitrary kernel memory locations via a crafted application, and escalate their privileges or execute malicious code.
- CVE-2009-1185 [72]: `udev`, which is a root-level-code, does not verify whether a `NETLINK` message originates from kernel space or not, which allows local users to gain privileges by sending a `NETLINK` message from user space. Netlink is a datagram oriented service, used to transfer information between kernel modules and user-space processes.
- CVE-2010-EASY [73]: allows to build the RageAgainstTheCage (RATC) exploit which exploits the **Android Debug Bridge** (`adb`) daemon's functionality. The `adb` process starts with root privileges, and it has to call `setuid()` to lower its privileges to shell privileges. The exploit uses the `fork()` function to start multiple processes. When the number of processes for the malware reaches the maximum permissible number `RLIMIT_NPROC`, the exploit kills the `adb` process. The `adb` process again restarts with root privilege and tries to call `setuid()` to lower its privilege, however, at that point, the exploit has already started `RLIMIT_NPROC-1` processes other than `adb` process. So the call to `setuid()` fails and `adb`

keeps running as root. The DroidKungFu malware uses this exploit to bypass the Android Sandbox security mechanism and perform actions such as downloading additional applications and adware without user's permission. The Trojan can also add new service and receiver into the application. Once the system is booted, the malware sends a notification to the receiver. The receiver can then initiate the service without user permission.

The vulnerabilities presented above have been patched over the years and now SELinux prevents such attacks from exploiting the Linux kernel. However, malwares like Gooligan [74] and DroidKungFu still remain an interesting study case due to their wide spread.

Attacks targeting memory

Attacks targeting **memory** usually exploit memory corruption vulnerabilities to add malicious content in the kernel address space or redirect the execution of the kernel to the address space where malicious content is present [68]:

- **Return-oriented programming** (**ROP**) [75] attack is a complex yet powerful code reuse attack which doesn't require the attacker to inject any malicious code as it reuses existing code. Reusable code is a set of "gadgets", i.e., machine-level instructions that precede the "return" instruction. **Gadgets** are executed indirectly by controlling the flow of stack. Buffer Overflow vulnerabilities can be exploited to mount such attack. ROP circumvents the security offered by **DEP**, and other techniques which are used to protect memory. **Stack Canaries** [76] can be used to detect ROP and similar attacks that are based on buffer overflow of the stack. In the Stack Canary approach, a small canary (integer) value is selected and placed before the address of stack return pointer. Canary value selection takes place at the beginning of the program. Whenever buffer overflow occurs, the Canary value gets overwritten along with the stack return pointer. Every time a routine calls the stack return pointer, it checks the first Canary value and, if the value is different from the initial value at the beginning of the program, it indicates that buffer overflow has occurred, i.e., the value of the stack return pointer has changed. This method will stop the execution of any malicious activity thus making it challenging for attackers to execute the buffer overflow attack in the presence of stack canaries. To breach the security offered by Stack Canary, attackers have to look for some source for information leakage. Some other defensive techniques are **ASLR** [77], instruction set randomization, etc....
- **Jump-oriented programming** (**JOP**) [78] attack uses a gadget dispatcher to send functional gadjets and then control their execution, and the program counter to point to the dispatcher table (which contains addresses and contents of the gadgets). ROP is usable when the buffer location is on the stack, but in the case where the buffer is located on the heap then JOP is used instead. **Control-flow integrity** (**CFI**) [79], **Data-flow integrity** (**DFI**) [80], **Program shepherding** [81], and a few other similar techniques can be used to defend against ROP and JOP attack. However, these techniques require a significantly large number of resources and codebase, which makes it cumbersome to deploy such defensive techniques.
- **String-oriented Programming** (**SOP**) [82] extends ROP, JOP and uses a format string exploit to overwrite the **Global Offset Table** (**GOT**) [83] entries or function pointers.

- **Blind ROP** (**BROP**) [84] attack is able to exploit a stack buffer remotely without knowing the target binaries. This method can hack servers that are based on both open-source and closed-source software. The attack's requirements are:

  (a) knowing the vulnerability in the stack and how to exploit it;

  (b) a server application that will restart every time after it crashes;

  The attacker must be able to crash the server n times (e.g., with buffer overflow), and every time the server is supposed to restart. The period during which server restarts is used to find enough gadgets to trigger and control arguments of "write" syscall of the victim system. With that syscall, attackers can send over the socket the server's binaries to get sufficient knowledge for exploiting the system. Randomizing the address space of the server when it restarts, sleep on crash, and Control-flow integrity (CFI) are a few techniques that can be used to mitigate this attack.

- **Return-to-user** (**ret2usr**) attack overwrites kernel pointers to redirect the execution flow to the address of arbitrary code in the user space. It is essential to isolate the kernel and user address space to increase kernel security. In Android 8.0, a **Kernel-ASLR** is used to reduce attacks at kernel level by randomizing the location of kernel code at boot time. However, a timing attack named **DrK**, which exploits a hardware feature called **Intel Transactional Synchronization Extension** (**TSX**) found in latest CPUs. This attack can de-randomize the location of kernel code. Usually, OS is supposed to intervene when a hardware exception occurs, but TSX suppresses such exceptions from being notified to the OS. It results in abortion of the transaction and invoking of the abort handler. The time difference in accessing mapped and unmapped memory locations in the kernel during abort handling is calculated by the DrK to de-randomize the kernel code.

- **Return-to-Libc** (**ret2libc**) [85] is a variant of ROP that uses a stack overflow method to overwrite the return address and jump to a libc function. If the right arguments are passed to the function, then the function will execute its code bypassing the security provided by DEP. ASLR can safeguard the system from this attack.

- **Sigreturn-oriented Programming** (**SROP**) is a variant of ROP which uses only one gadget for exploitation. The attack mechanism involves placing a fake structure of signal context on the stack and then overwriting a return address with the address of a known gadget. This gadget will help the attacker in accessing either `syscall()` or `sigreturn()`. SROP is platform independent, and it can attack different OSs. ASLR, Signal cookies, Vsyscall emulation, Return-Address Protection (RAP), and Control-flow Enforcement Technology (CET) can defend the system from this attack.

Attacks targeting the bootloader

The **bootloader** is responsible for initiating the OS and some other kernel processes. In current mobile phones, bootloaders are also responsible for ensuring the integrity of the Chain of Trust (CoT) policy of the system. According to the CoT policy, the bootloader has to verify the integrity of processes in each stage during booting before the execution of the processes. The booting process is supposed to be invulnerable to kernel attacks where attackers gain complete control over the kernel, and it should be able to maintain the CoT policy of the system even in a compromised environment.

Mostly, vulnerabilities exist due to flaws found in code/implementation of bootloaders [68]:

- CVE-2014-9798: `platform/msm_shared/dev_tree.c` in the Qualcomm bootloader in Android before 2016-07-05 on Nexus 5 devices does not check the relationship between tags addresses and aboot addresses, which allows attackers to cause a DoS (OS outage) via a crafted application;
- CVE-2015-8893: `app/aboot/aboot.c` in the Qualcomm bootloader in Android before 2016-07-05 on Nexus 5 and 7 (2013) devices allows attackers to cause a DoS (OS outage or buffer over-read) via a crafted application;

BOOTSTOMP [86] is a multi-tag taint analysis hybrid method (static + dynamic) that defends the system's bootloader from attacks by identifying hidden vulnerabilities. BOOTSTOMP was able to identify CVE-2014-9798 in previous versions of Qualcomm devices and other vulnerabilities in the Huawei and NVIDIA'S bootloader.

Attacks targeting device drivers

Device drivers are interfaces that enable communication between software and hardware. In Android, bugs are mostly found either in a root driver or drivers of applications like Camera, WiFi, or Bluetooth, which are pre-installed in the device [68]:

- **Time-of-check to time-of-use** (**TOCTOU**) vulnerability: it arises when driver and user have direct access to user data buffer. So, there are chances that both may be accessing and updating data buffer simultaneously (race condition).
- CVE-2016-2435: the NVIDIA video driver in Android before 2016-05-01 on Nexus 9 devices allows attackers to corrupt memory values and, finally, launch a privilege escalation attack against the system.
- CVE-2016-2411: a Qualcomm Power Management kernel driver in Android 6.x before 2016-04-01 allows attackers to gain privileges via a crafted application that leverages root access.

7.2.3 HAL-based attacks

Key Reinstallation Attack (KRAK)

The **Key Reinstallation Attack** (**KRACK**) [87] targets a vulnerability in the WPA2 4-way handshake. The attacker needs to be within the Wi-Fi range of the target system [68]. Each time the client receives the message 3 of the handshake, it will reinstall the same session key, and thereby reset the incremental transmit packet number (nonce) and receive replay counter used by the data-confidentiality protocol. The attacker can force these nonce resets by collecting and replaying retransmissions of message 3. Hence, the data-confidentiality protocol can be attacked, e.g., packets can be replayed, decrypted, and/or forged.

This attack is particularly devastating on Android 6.0 and higher versions because of an implementation flaw in the `wpa_supplicant` module. Such vulnerability appears to be caused by a remark in the 802.11 standard that indirectly suggests to clear the TK from memory once it has been installed. The affected devices install an all-zero encryption key (TK) when receiving a retransmitted message 3.

Keyloggers

An interesting case study is the **Invisible Man malware** [68, 88]. Disguised as an authentic Flash player update for Android devices, the app uses the granted accessibility privilege to do its dirty work, rather than rely on exploiting software vulnerabilities. It can get the bank details of the user by acting like a key-logger. It grants itself device administrator rights, draws itself over other apps, installs itself as a default SMS app, and grants itself some dynamic permissions that include the ability to send and receive SMS, make calls, and read contacts. Furthermore, using its newly-gained abilities the Trojan can block any attempt to remove device administrator rights – thereby preventing its uninstallation. It is interesting that in doing so it also blocks any attempt to add or remove device administrator rights for any other app too.

Audio interface

Malicious applications, installed into the system, can also use the Audio channels to steal user's credentials [68]. Such applications send deceitful commands, through the speaker of the victim's system, to its microphone, which behaves as confused deputy. TalkBack (i.e., Google screen reader included on Android devices) Accessibility Service is used to accomplish the purpose of this attack, as it reads out the password typed by the victim to the attacker.

HAL security has been improved in Android Oreo: different device drivers have **separate HALs**. Each driver is controlled only by the HAL to which it is assigned. When a process requests for a device driver, it gets a particular HAL, and together they run in a sandbox. Thus, the process gets only necessary privileges that are required to complete its job. Processes won't be able to access device drivers directly and to access device drivers that they haven't requested. **AuDroid** is a policy enforcement technique that is employable with SELinux for preventing and detecting attacks that use audio channels for exploiting the system. **SemaDroid** is a method to protect from attacks that target sensor data (e.g., Camera, GPS, Microphone, Motion Sensors, etc...) by enforcing user policies that put restriction on the applications from collecting critical information from sensors, thereby protecting the user's private information.

7.2.4 Application-based attacks

Runtime Information Gathering (RIG)

A **Runtime Information Gathering** (**RIG**) attack is used to accumulate information from the victim's system, during the run time of the malicious application [68]. The malicious app exploits the permissions given at installation time. Researchers developed **App Guardian**, that keeps a check on the suspicious application running and stops all other processes running in the background. Once the process stops the executing, it restarts the stopped processes and cleans the memory used during runtime by the suspicious application so that no trace of information is left for the attackers to steal from the system.

Library-based attacks

An app's security depends from the libraries (both statically and dynamically linked). Any vulnerability in the libraries will be included in the utilizing application. In March 2017, Android Security Bulletin rated a vulnerability (CVE-2016-2182 [89]) in OpenSSL and BoringSSL as critical, since it enabled an attacker using a specially crafted file to cause memory corruption during file and data processing, leaving room for remote code execution within the context of a privileged process. Libraries are included in the application code (inside the `.apk` file) to ask for the permissions that are required to run the app. The inconsistency between an application description and its functionalities may be the result of a large number of permissions requested by the third-party libraries, thus opening a channel for the malicious applications to intrude into the system [68].

It is essential for the Android security mechanism to be able to detect and analyze the library code separately from the application code. There are many techniques available for detecting library code, but, like any other technique, malware developers can evade them with the help of code obfuscation. Using an updated version of libraries in application code is of prime importance, as most of the security flaws arise because developers often use the outdated version of libraries or they slowly adapt the updated version. To tackle the problem of code obfuscation, malware detectors should use library detection techniques that employ machine-learning approaches.

Intra-library collusion

A library obtains a combined set of privileges assigned to multiple applications because it's shared among them (mostly third-party apps). Malware attackers are exploiting this security vulnerability as such flaws allow access to the user's private information, with the help of the collective privileges, gained by shared libraries. Android does not perform privilege separation in-between applications and their in-built libraries or Ad-libraries. So, all the permissions allowed to an application are by default accessible to its in-built libraries [68].

Privilege escalation attacks

Third-party libraries are the easiest approach that attackers are using for exploiting privilege escalation vulnerability at different levels in the Android system. For this purpose, it is necessary to provide the applications and libraries with limited privileges, and they should not be allowed to use each other's permissions or any other permission not assigned to them. Broadly, there are two types of privilege escalation attacks: confused deputy attack [90] and collusion attack [91]. The former type, exploits the vulnerability of the target system, and the latter can implement the malicious application, attacking the system [68].

7.3 Tools for preventing privileges escalation attacks

According to [68], these tools might be used for preventing privileges escalation attacks:

- **eXtended Monitoring on Android** (**XManDroid**) prevents privilege escalation attacks dynamically based on the system policy already defined in the system database. Researchers have designed the **Policy Check Algorithm**, which is embedded in the Decision Maker component of the XMandroid framework, to keep track of Inter-component Communication (ICC) among the applications. With the help of policies defined in the policy database that are the security rules, it decides whether it should allow the action or reject it. User confirmation is also taken into account whether to allow ICC call. XMandroid helps in preventing an ICC-based privilege escalation attack.
- **Extending Android Permission Model** (**Apex**) helps users grant permission to applications for accessing resources during runtime selectively. An app installer named **Poly** is implemented, which is an extension to the already existing installer in the system. It provides users with an easy interface to imply constraints on access to resources requested by the applications at installation time.
- **AdSplit** separates the advertising libraries from the host application. Ad-libs and app run in a completely **separate environment**. **Unique UID** and separate permissions are given to app and ad-libs so that the app doesn't need to request additional privileges for the ad-libs. An **advertising service** is used for coordination between processes. It keeps track of user actions performed at UI and sends them to related **advertising activity**. With the help of a **centralized advertising service** component, AdSplit can identify any fake UI event, and this component is also responsible for proper display of advertisements.
- **AdDroid** separates the advertising component from the parent application. The component is integrated into the Android framework rather than the parent app. This helps in restricting it from gaining access to any security-critical info, even if the parent app has the privilege of accessing that same info. Introduces the **advertising API**, which is an extension of the Android API, responsible for data and application. Includes **two new permissions**, `ADVERTISING`, and `LOCATION_ADVERTISING`. For using AdDroid, applications need to request for any of these two permissions. These permissions give requesting application access to the advertising API calls. AdDroid handles all the decisions regarding advertising and events related to user interactions.
- **Component-Level Access Control** (**Compac**) puts a restriction on the privileges, given to the third-party elements, at component level. It achieves this by letting the system decide which requests from third-party elements for accessing the private information need to be accepted and which not. The system can make the decision based on the information excavated from the application's components, using runtime information of Java package. Compac gives privilege to the users and system to assign a subset of permission to the application component rather than giving similar access rights to all the components of the application.
- Unlike AdDroid, in **Privilege De-escalation** (**PEDAL**) researchers have included a module, called separator that allows users to assign privileges to ad libraries selectively. This module forbids ad-libraries from inheriting permissions, which are granted to their parent applications by the system at the time of installation.

7.4 Techniques for malware detection and analysis

The research made by [68] also lists some techniques for malware detection and analysis.

7.4.1 Static and dynamic approaches to analyze and detect attacks

There are two main approaches for detection and prevention of malicious activity or application:

- **Static** detection:
  - extracts features from the application file without executing it;
  - is resource and time efficient, as the application is not required to be executed;
  - monitors features like permissions, API calls, `.dex` files for opcodes, and metadata;
- **Dynamic** detection:
  - analyzes features while the application is running;
  - may be resource and time consuming;
  - it includes network traffic, battery usage, CPU utilization, IP addresses, and opcodes;
  - is not defeated with code obfuscation;

Most of the dynamic and static detection mechanisms incorporate machine-learning (ML) techniques for fine results. Once the model is trained, using a machine-learning algorithm, it can work automatically in the detection of attacks and human intervention is not required. It helps in improving the detection of attacks that are otherwise not detectable during manual analysis. Moreover, data mining helps in identifying the class of attacks that are previously unknown, with the help of ample experience.

7.4.2 Static malware detection

There are broadly two categories of static detection:

- **Op-Code** analysis: n-gram opcode analysis, the detection tool extracts n-gram opcodes from the dataset of malware and benign applications. The dataset is used to classify application using Pattern Recognition and machine-learning techniques. One of the most widely used classifier for this purpose is Support Vector machine.
  - Note: an n-gram is a sequence of n adjacent symbols in particular order.
- **Manifest and API Calls** analysis: the tools use machine learning to learn features such as permissions, intents, component deployment from the Manifest file, and API Calls details from the dataset of malicious and benign applications, these data are then used to classify the target applications.

Static malware detection through permissions analysis

Malware can be detected by analyzing the permissions they request when installed on an Android device. Analyzing the permissions requested by a particular application makes it possible to determine whether it is potentially malicious. For example, if an app requests permissions unrelated to its intended functionality, such as the ability to access the user's contacts or call logs, it may be a sign that the app is malicious. Additionally, if an app requests permissions that give it access to sensitive information, such as location data, it may be a sign of malware. Not all apps that request these permissions are malicious, but it can be a red flag that needs to be investigated further.

7.4.3 Dynamic malware detection

Dynamic analysis is of two types [68]:

- **in-the-box** analysis:
  - Data collection and analysis is performed on the same privilege level as the malware. The detection tools use ART for this purpose.
  - Data collection and analysis can be meddled with by malware, as it shares the same privileges as the analyzer.
  - Allows interception of data at the OS level, access to memory architecture, libraries, API, and other methods. However, at the same time, it makes the critical data vulnerable to attacks.
- **out-of-the-box** analysis:
  - Analysis is done inside a virtual environment to deceive the attackers and allow the analyzing technique to understand the nature of the attack securely.
  - Attackers have found methods to evade such emulated environment and use strategies to remain undetected.

7.4.4 Host-based vs. Network-based detection

The two possible malware detection models are [68]:

- **Host-based** detection:
  - The detection tool is incorporated in the system itself and can better analyze malicious activities occurring within the system as well as defend it from outside attacks.
  - As host-based detection mechanisms are located on the system itself, they are easy targets for attackers.
- **Network-based** detection:
  - Is responsible for monitoring the whole network to which host is connected and has access to complete network as well as to the system.
  - Can't analyze malicious activities occurring within the system, like at the kernel level.
  - Requires more software and hardware resources as compared to the host-based detection mechanism and is costlier as well.
  - A network-based detection mechanism has to analyze heavy network traffic, so its performance suffers from the intense workload and leads to a poor detection rate and fails against previously unknown attacks.

7.4.5 Machine learning techniques for malware detection

Machine learning models can also be used for malware detection [68]:

- **Supervised Learning** (**SL**):
  - This class of algorithms takes known set of input data along with known responses to the output data and trains the model to generate predictions for the new dataset. Classification and regression are some of the well-known SL techniques. Classification is used to predict the discrete outcomes.
  - Examples of classification algorithms are Naïve Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LR), Decision Trees (DT), k-Nearest Neighbor (kNN) and Neural Networks (NN).
- **Unsupervised Learning** (**UL**):
  - This class of algorithms draws inferences from the unlabeled input dataset. Clustering is a commonly used UL technique.
  - Common clustering algorithms are k-means, k-medoids, Hidden Markov Models (HMV).
- **Semi-Supervised Learning** (**SSL**):
  - This class of algorithms combines both supervised and unsupervised learning techniques with some labeled and unlabeled data.
- **Reinforcement Learning** (**RL**):
  - RL algorithms enable the agent to learn to achieve a goal in a potentially complex and uncertain environment by maximizing the cumulative reward.
  - Some examples of RL are Deep Q Network (DQN), Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC).
- **Deep Learning** (**DL**):
  - This class of algorithms relates to the functioning of brain in the form of Artificial Neural Network (ANN) and has the capability to process large volumes of data.
  - Well known DL algorithms are Deep Neural Networks (DNN), Convolutional Neural Network (CNN), Recurrent Neural Networks (RNN), Deep Belief Networks (DBN).

According to the "**No Free Lunch**" (**NFL**) **theorem**, there is no single ML model that is best suited for every problem. The assumptions of one model for a particular problem may not hold for another problem. Therefore, multiple ML models can be tried and tested to find the one that works best for a particular problem.

Chapter 8

iOS spyware and attacks

8.1 Vulnerabilities overview

Over the years, it has been proven that iOS is not free from vulnerabilities. Despite the fact that the number of such vulnerabilities has dropped from 2014 to 2024, attackers still succeed in exploiting 0-days for compromising iPhone devices. Most vulnerabilities regard overflow (i.e., buffer overflow, integer overflow, etc...) and memory corruption, with the newest ones being patched in the latest system updates (i.e., iOS 17.2, 17.3, 17.4, and 17.5) [92].

Year	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	File Inclusion	CSRF	XXE	SSRF	Open Redirect	Input Validation
2014	35	43	0	1	1	0	0	2	0	0	9
2015	184	196	0	0	5	0	1	3	0	0	26
2016	83	99	0	3	0	0	0	0	0	0	10
2017	210	205	0	14	0	0	0	0	0	1	30
2018	54	53	0	1	0	0	0	0	0	1	13
2019	81	182	2	14	0	0	0	0	0	0	48
2020	22	106	0	8	2	0	0	0	0	0	20
2021	23	98	0	6	3	0	0	0	0	1	7
2022	16	81	0	0	0	0	0	1	0	0	4
2023	17	27	0	1	0	0	0	0	0	0	2
2024	3	11	0	0	0	0	0	0	0	0	0
Total	728	1101	2	48	11		1	6		3	169

Figure 8.1: iOS vulnerabilities by types/categories

The most impactful vulnerabilities allow to mount DoS attacks and to execute code:

Year	Code Execution	Bypass	Privilege Escalation	Denial of Service	Information Leak
2014	51	1	1	51	11
2015	211	8	8	232	56
2016	78	4	4	113	28
2017	221	1	1	241	48
2018	63	0	0	63	18
2019	17	3	3	10	17
2020	46	0	0	18	8
2021	39	6	6	16	5
2022	72	0	0	10	3
2023	43	2	2	12	1
2024	14	1	1	8	1
Total	855	26	26	774	196

Figure 8.2: iOS vulnerabilities by impact types

8.2 Jailbreak and forensics implications

Jailbreaking is the process of bypassing restrictions on iOS to install other applications and tweaks that are not allowed by Apple's party. The main purpose of jailbreaking is to get a superuser right that allows root access to system files, so users can use the device without any limitations by Apple (e.g., installing an app that is not available on the App Store).

There are three types of jailbreak:

- **Tethered Jailbreak**: It requires a computer-assisted tool. The device has to be connected to a computer with a lightning cable. Tethered jailbreak is temporary and, because of that, it can only be used in a single boot. A tethered jailbreak exploits the **iBoot stage**.
- **Untethered Jailbreak**: It doesn't need a computer except when jailbreaking the device for the first time. It is permanent, because every time the device restarts it no longer needs to be connected to the computer to activate the jailbreak, including when running out of battery. Untethered jailbreak uses an exploit on the **Boot ROM** as well as installing additional exploitation on the root file system.
- **Semi-Tethered Jailbreak**: It can be done with or without using a computer. This type of jailbreak usually performs attacks on the **kernel stage** and it is performed after the iPhone is live and start running the application and should be done every time after the iPhone restarts/turns off.

Jailbreaking modern versions of iOS is extremely complex as it requires exploiting multiple vulnerabilities in various parts of the OS to defeat the systems' security measures [93]. In general terms, a jailbreak performs the following steps:

1. **Sandbox Escape**: the jailbreak tool utilizes an exploit allowing it to access components it does not have the permissions to;
2. **Privilege Escalation**: the jailbreak tool gains elevated privileges allowing it to access protected resources (e.g., mount the root file system, patch the kernel, inject code etc...);
3. **KPP Bypass**: the jailbreak tool disables or works around the code signing check, which allows modifications to the file system without making the device unbootable, causing a bootloop or random reboots;

Starting with iOS 11, all jailbreaks are using the same installation procedure. A failed jailbreak does not cause system instability and does not require reinstalling iOS in order to perform another attempt. Jailbreaking on iOS violates Apple's EULA and eliminates the warranty on its device. However, in 2010, the Electronic Frontier Foundation (EFF) was able to obtain certain exceptions that were converted into the **Digital Millenium Copyright Act** (**DMCA**) [94].

The research conducted by [95] shows how to extract forensically-relevant data from a jailbroken iPhone 6s running iOS 12.1.1. Semi-tethered jailbreak on such device has been performed by using the Unc0ver tool [96], as the exploits available at that time could only affect the kernel stage. More powerful exploits, targeting the Boot ROM stage, were available in iOS 4.3.3 by using the Redsn0w tool [97]. The research also shows that, even though the device has been jailbroken, the user data in the Media directory are not altered. The integrity of the files stored in such directory remains unchanged, as shown by their matched hash values before and after the jailbreak. However, system data alteration is inevitable as the Unc0ver tool uses the "voucher swap" exploit to create a fake kernel task which replaces the original one that is running in memory. By dumping the kernel cache with Jtool2, it is shown that its hash value changes after the jailbreak.

A similar research was conducted by [98] on an iPhone 7 Plus running iOS 14.8.1. This research points out that there are some possible concerns to jailbreaking an iPhone when performing a forensic investigation:

- The law of the nation where jailbreaking is been performed should be followed.
- Hardware may be damaged, e.g., as a result of an overload.
- Jailbreaking leaves room for malware installation.
- The jailbreak tool may function in an unreliable manner and cause data loss.
- At the research time (2021) Internet connectivity was required for jailbreaks during installation. The reason for this is that installing a program outside of the Apple Store required a verification of the used account and password from Apple. Remote deletion or blockage of the iPhone can be initiated when the device is connected to the internet.

The research also explains how to perform the jailbreak:

1. The iPhone must be unlocked and linked to a trusted computer. The passcode is required to complete the jailbreak when a passcode is active. Manually testing a passcode will result in a timeout and, at the very least, lock the phone after several tries. At the very worst, the phone will be reset to factory settings after numerous attempts. In case the PIN is not known, using a security breach to allow brute forcing it could be a possibility (the only known vulnerability which makes this possible, CVE-2014-4451 [99, 100], has been patched). If the ultimate goal of jailbreaking is acquiring forensically relevant data, another way to "bypass" the PIN and acquire a backup of the device involves the usage of lockdown records [101].

2. Entering DFU mode after using Checkra1n tool [102]. The DFU process requires pushing and holding several physical buttons on the device for specific periods of time. The DFU mode may fail after numerous attempts if one or more of these buttons fails (even an oxidized contact can violate the timings). Disassembling the device and shortening some pins to induce DFU are possible methods. Some test pins on the mainboard of iOS devices are accessible for troubleshooting reasons. A pin called FORCE DFU is one of them. The bootloader probes the pin at boot, and if a voltage of 3.3V is applied, DFU mode is activated.

The MOBILedit forensic tool has been used to extract data from the device. However, the research only highlights that the number of acquired data is significantly higher on the jailbroken device than on the jailed device but no information regarding the data integrity is given (supposedly, it shouldn't been altered).

Other articles and tools regarding jailbreak on latest iOS versions are available but they haven't been verified nor tested [103–106].

8.3 Macro categories of attacks

- Attacks that exploit vulnerabilities of iOS (some examples):
  - **Operation Triangulation attack**: Designed to work on iOS versions up to iOS 16.2, this attack exploits four 0-days to install spyware on the target device;
  - **Predator spyware** privileges escalation attack;
  - **Pegasus spyware**: It's one of the most advanced spyware, developed by the Israeli cyber-arms company NSO Group and capable of targeting both Android and iOS devices with a series of **zero-click exploits** (i.e., which don't require any user interaction to compromise the system);
  - Further references to iOS vulnerabilities exploitation are given by Google Project Zero team [107–109], which also provided details of a complex iOS 16.4.1 Safari WebContent vulnerability exploit for remote code execution [110];
- **Private APIs exploitation**: This category of attacks does not require the target device to be jailbroken and consists in exploiting iOS private APIs to access information which are usually reserved to system apps. However, applications that use private APIs must pass the App Store vetting process to be published on the store itself. Due to the restrictions imposed by Apple, private APIs usage is not allowed and such applications are often rejected from the store, except when properly designed to bypass verification.

8.3.1 Operation Triangulation attack

In 2023, Kaspersky team designed this attack targeting iOS [111], after studying the TriangleDB spyware [112]. The attack chain is really complex, due to the high number of 0-day vulnerabilities to be exploited:

1. Attackers send a **malicious iMessage attachment**, which the application processes without showing any signs to the user. This attachment exploits the remote code execution vulnerability CVE-2023-41990 [113] in the undocumented, Apple-only ADJUST TrueType font instruction (removed with an iOS 16.3 patch). It uses ROP/JOP and multiple stages written in the NSExpression/NSPredicate query language, patching the JavaScriptCore (JSC) library environment to execute a privilege escalation exploit written in JavaScript.

2. The **JavaScript exploit** is mainly dedicated to JavaScriptCore and kernel memory parsing and manipulation:

   - Exploits the JSC debugging feature DollarVM ($vm) to gain the ability to manipulate JSC's memory from the script and execute native API functions.
   - Uses a Pointer Authentication Code bypass for exploitation of recent iPhone models.
   - Uses the integer overflow vulnerability CVE-2023-32434 [114] in XNU's memory mapping syscalls (`mach_make_memory_entry` and `vm_map`) to obtain read/write access to the entire physical memory of the device at user level.
   - It uses hardware memory-mapped I/O (MMIO) registers to bypass the Page Protection Layer (PPL). This was mitigated as CVE-2023-38606 [115].

   After exploiting all the vulnerabilities, the JavaScript exploit can do whatever it wants to the device including running spyware, but the attackers chose to:

   (a) Launch the IMAgent process and inject a payload that clears the exploitation artefacts from the device;

   (b) Run an invisible Safari process and forward it to a web page with the next stage;

3. The web page has a script that verifies the victim and, if the checks pass, uses a **Safari exploit** based on CVE-2023-32435 [116] to execute a shellcode.

4. The **shellcode** executes another kernel exploit in the form of a Mach object file. It uses the same vulnerabilities: CVE-2023-32434 and CVE-2023-38606. It is also massive in terms of size and functionality, but completely different from the kernel exploit written in JavaScript. Certain parts related to exploitation of the above-mentioned vulnerabilities are all that the two share. Still, most of its code is also dedicated to parsing and manipulation of the kernel memory. It contains various post-exploitation utilities, which are mostly unused. The exploit obtains root privileges and proceeds to execute other stages, which load spyware.

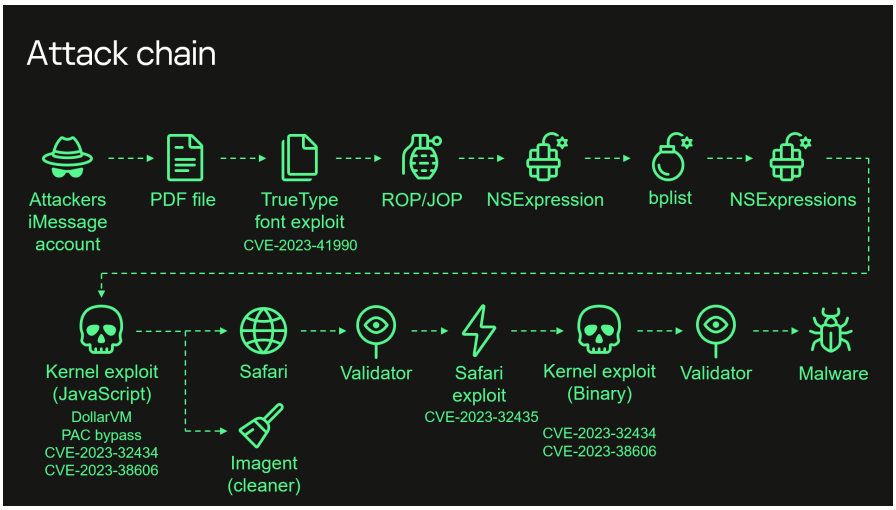


Figure 8.3: Operation Triangulation attack chain

8.3.2 Predator spyware privileges escalation

According to Apple, the following vulnerabilities may have been actively exploited against versions of iOS before iOS 16.7 by the **Predator spyware** attack chain [117–119]:

1. CVE-2023-41993 [120]: A maliciously crafted webpage could exploit this vulnerability to remotely execute malicious code on the target device;
2. CVE-2023-41991 [121]: A malicious app may be able to bypass signature validation due to a certificate validation issue;
3. CVE-2023-41992 [122]: This kernel-level privilege escalation flaw allows rogue applications and users to gain the necessary privileges for taking full control of a device;

The attack chain then ran a small binary to decide whether or not to install the full Predator implant on the target device. Predator's campaign targeted Egyptian politicians with either WhatsApp messages or SMS messages containing a link which redirected the victim to the maliciously crafted webpage.

8.3.3 Pegasus spyware

Pegasus is considered one of the most dangerous spyware due to its ability to infect devices by using advanced zero-click exploits. Unfortunately, the lack of research papers and articles regarding newer versions of Pegasus does not allow to fully understand how its attack chain is structured. Here's a summary of all information that were collected during this research:

- According to the Citiziens Lab [123], in 2023 Pegasus used the so called **blastpass exploit** to infect devices up to iOS 16.6. The exploit was achieved by sending malicious `.pkpass` files as attachments over iMessage. These files contained an embedded image which would be automatically loaded and parsed on the device without any user interaction. The image is a zero-click exploit which attempts to gain remote code execution by exploiting either CVE-2023-41061 [124] or CVE-2023-41064 [125]. Similar vulnerabilities, like CVE-2021-30860 [126] which is related to maliciously crafted PDF files, had already been exploited to compromise previous iOS versions, as stated by the 2021 article of Google Project Zero [127].
- This 2024 research paper [128] provides a comprehensive overview of Pegasus features, thanks to a meticulous literature review:
  - Undetectability: Pegasus is difficult to detect because of the employed zero-click exploits and its capability to self-distruct in case of need. Pegasus can also receive updates from the command and control server, which helps it adapt to changes in the device's environment and avoid detection;
  - Data Extraction: Pegasus can access a wide variety of data including SMS messages, emails, contacts information, call logs, browsing history, social media apps monitoring (e.g., Facebook, Instagram), instant messaging apps monitoring (e.g., WhatsApp), GPS location, key strokes, real-time audio, video recordings, etc...;

8.3.4 Private APIs in iOS

Private APIs are embedded in iOS frameworks and can only be used by built-in applications to provide access to device resources and sensitive information. The major issue with private APIs is that they provide information which are not usually regulated at runtime. A 2015 research made by [129] showed the feasibility of using private APIs in iOS applications without being detected by the App Review vetting system, and proposed a detection mechanism called iRiS based on the combined use of both static and dynamic analyis techniques. By constructing the names of private APIs at runtime, it was possible to invoke private APIs in third-party applications and still be able to pass the vetting process (i.e., no jailbreak needed).

iOS provides a rich set of frameworks for building user-level applications. These frameworks are essentially directories that contain dynamic shared libraries and resources. The dynamic shared libraries expose APIs for applications in two forms:

- Traditional C functions that are explicitly exported by the shared libraries;
- Methods in Objective-C classes that are managed and dispatched by the Objective-C runtime;

Frameworks and APIs can be classified into two macro categories:

- **Public** frameworks/APIs: Used by third-party apps;
- **Private** frameworks/APIs: Reserved to system apps;

Public frameworks can use private APIs for their internal implementation. However, private APIs and frameworks offer a set of features that could put the system security at risk, if made available to third-party apps (e.g., `SpringBoardServices` for launching and terminating apps). Despite private APIs not being documented, they can still be reverse-engineered.

Another research from 2013 goes more in depth by explaining a possible technique which could lead to private APIs usage in iOS applications [130]. The attacker creates a normal app in which he plants vulnerabilities and hides code gadgets along side the normal functionalities. After the app passes Apple's app review and gets installed on victims' devices, the attacker exploits the vulnerabilities and assembles the gadgets in a particular order to perform malicious operations.

Chapter 9

Spyware principles and functionalities

9.1 Installation

Spyware, like any other malware, can be installed on the victim's device in several ways:

- **Misleading marketing** [131]: Spyware authors will often disguise their malicious software as a legitimate tool, such as a hard disk cleaner, download manager, or new web browser. Even after uninstalling such tool, the spyware may still remain hidden on the target system;
- **Phishing and social engineering** [132]: Spyware is often delivered with intense phishing campaigns, using either e-mails with malicious links and/or attachments or SMS messages with malicious links (smishing) which the user is tricked into clicking;
- **Security vulnerabilities** [133,134]: Attackers often target code and hardware vulnerabilities to gain unauthorized access to devices and install spyware. Exploitable vulnerabilities may not necessarily be in the target device but they could also be on a third party web site which the victim is visiting (e.g, XSS [135]);
- **Software bundles** [136]: Bundleware sees users unknowingly install spyware within a bundle of software they believe to be legitimate. In this case, spyware may be delivered as an add-on/extension of the original software and could still remain on the target device after uninstalling the main application;
- **Trojans** [137–140]: A Trojan is a type of malware that pretends to be another piece of software. Cyber criminals use Trojans as a method for delivering malware strains, such as spyware, cryptojackers, and viruses, onto devices;
- **Others**: Accepting cookie consent requests and/or pop-ups from insecure/untrusted websites [141], downloading games, movies, or music from pirated or spoofed websites, downloading malicious mobile apps from third party stores;

9.2 Permissions of a spyware

After being installed on the victim's device, the spyware could try to escalate its privileges for obtaining a set of useful permissions which will allow it to silently monitor the system. This doesn't necessarily mean to try and get root privileges, as in most cases that is not even needed. In some cases, it is the user who grants permissions to the malicious application. Specifically for Android, here's a set of permissions which are most likely to be used by a spyware:

Type of permission	Name	Ref
Calls and contacts	READ_CALL_LOG	[142]
	READ_CONTACTS	[143]
Camera and audio	CAMERA	[144]
	RECORD_AUDIO	[145]
GPS	ACCESS_BACKGROUND_LOCATION	[146]
	ACCESS_COARSE_LOCATION	[147]
	ACCESS_FINE_LOCATION	[148]
Network	ACCESS_NETWORK_STATE	[149]
	ACCESS_WIFI_STATE	[150]
	INTERNET	[151]
	NEARBY_WIFI_DEVICES	[152]
Persistency	FOREGROUND_SERVICE	[153]
	RECEIVE_BOOT_COMPLETE	[154]
	REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	[155]
SMS	READ_SMS	[156]
	RECEIVE_SMS	[157]
Storage	READ_EXTERNAL_STORAGE	[158]
	READ_MEDIA_AUDIO	[159]
	READ_MEDIA_IMAGES	[160]
	READ_MEDIA_VIDEO	[161]
Others	BIND_ACCESSIBILITY_SERVICE	[162]
	GET_ACCOUNTS	[163]
	READ_CALENDAR	[164]
	READ_PHONE_STATE	[165]
	REQUEST_INSTALL_PACKAGES	[166]

Table 9.1: Android permissions which may be used by a spyware

While some of the above permissions might be self-explanatory, some others are not so straightforward. Here's a brief explanation of how a spyware could use such permissions:

- `RECEIVE_BOOT_COMPLETE`: Allows the spyware to detect when the device is either turned on or restarted, so that it can start spying on the user;
- `FOREGROUND_SERVICE`: It allows the malicious app to use the `Service.startForeground` method. Thanks to it, when the spyware service is started, it also runs in the foreground. By default, started services are background, meaning that if the system needs to kill them, they can be killed without too much harm. By calling this method, the application basically tells the system that the spyware service must be kept running;
- `REQUEST_INSTALL_PACKAGES`: May be used for installing additional malicious APKs from untrusted sources;
- `BIND_ACCESSIBILITY_SERVICE`: Once accessibility service is granted to the malicious app, the attackers will have the ability to read elements on the device's screen and interact with the device through virtual taps, swipes, and entering text. This is used by the SpyNote malware to simulate a fake touch on the device's screen and grant itself all other requested permissions, as shown in this article [167];

All permissions must be declared in the application manifest. Each permission is assigned to a specific **protection level** which expresses the potential risk implied in the permission itself and indicates the procedure to be followed by the system for determining whether the application should be granted the permission or not. The previously listed permissions are assigned to one of the following protection levels:

- `dangerous`: A higher-risk permission that gives a requesting application access to private user data or control over the device that can negatively impact the user. The system might not automatically grant it to the requesting application. Any dangerous permission requested by an application might be displayed to the user and require confirmation before proceeding, or some other approach might be taken to avoid the user automatically granting the use of such facilities;
- `signature`: A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval;

9.3 Information gathering and reporting

After gaining enough permissions, the spyware starts monitoring the victim's device. In Android, this can be done in different ways, as the literature explains:

- Use the `WorkManager` class [168] to schedule periodic tasks, as explained by [169]. This guarantees that the spyware carries out deferrable persistent work while remaining stealthy, as this kind of procedure doesn't require the display of any notification. Scheduled work is stored in an internally managed SQLite database and `WorkManager` takes care of ensuring that this work persists and is rescheduled across device reboots. `WorkManager` is intended for work that is required to **run reliably** even if the user navigates off a screen, the app exits, or the device restarts. That's why it is often used for sending logs or analytics to backend services. In this case, however, it is used by the spyware to periodically send collected data to a remote server controlled by the attacker;
- Use services and broadcast receivers that detect when the device sends or receives messages and calls. This is briefly explained by [170], where they present the design of a malicious application to target Android 4.1.2. Even though there are some discrepancies with newer versions of Android, like the removal of the `READ_HISTORY_BOOKMARKS` starting from Android 6.0, which allowed to read browser history, the previous paper still presents a good overview on the principles behind building a spyware. A more detailed explanation is given by [171], where the SpyNote malware is analysed in depth. SpyNote records incoming phone calls as `.wav` files. This is done by instantiating a broadcast receiver for the system broadcast intent `PHONE_STATE`, as incoming calls change the phone state and trigger the receiver;

The ones presented above are just some approaches adopted by spyware developers to instruct the malware on when to collect user data, but how are data actually collected assuming to have all necessary permissions?

As explained in depth by [172], Android APIs can be exploited for doing so:

- **Camera access** can be stealthy performed with one of the following techniques:
  - **Invisible preview**: Applications usually show a camera preview to the user when she wants to take a picture or record a video from within the app (e.g., WhatsApp, Instagram, etc...). Spyware can use this feature to gain access to the camera by displaying a 1x1 pixel preview, which goes unnoticed by the user;
  - **Raw frames without a preview**: Spyware can use either the `ImageReader` [173] or the `SurfaceTexture` [174] Android class to access the output of the camera without showing any preview to the user;
  - **Invisible browser**: This approach is adopted by the Spy24 app [175] and it consists in using the `WebView` [176] Android class, which allows users to browse web content in an app while providing advanced configuration options to developers. By using a 1x1 pixel size in-app browser window, the app can stream live videos to a remote server;
- **Microphone access** can be exploited for recording both phone calls and voice calls, i.e., calls made in third party apps (e.g., WhatsApp):
  - **Phone call recording**: Before Android 6, it was easier to record phone calls by setting the audio source of the `MediaRecorder` class to the `VOICE_CALL` [177] value with the `setAudioSource` method [178], which provided access to both uplink and downlink audio sources. Starting from Android 6, this was no more possible, as setting the audio source to `VOICE_CALL` required the `CAPTURE_AUDIO_OUTPUT` permission [179] which is reserved to system components. After Android 6, it was still possible to force the usage of the `VOICE_CALL` audio source by exploiting the `libmedia.so` library. However, starting from Android 9, this was not possible anymore. Now, spyware set the audio source to `MIC` [180] for capturing uplink audio and use alternative methods for capturing downlink audio (e.g., turning off noise-cancelling and listening to the device's speaker);
  - **Voice call recording**: It became possible starting from Android 10 by exploiting the `AccessibilityService`. By using either `MediaRecorder` or the equivalent class `AudioRecord` [181], a malicious app registered as an accessibility service can concurrently access the microphone with the third-party app used for voice calling [182]. However, with Android 11 it's possible to use the new method `setPrivacySensitive` of the classes above for disabling the ability to capture audio concurrently, regardless of the scenario [183]. After Android 11 (i.e., after API level 30) no further changes have been applied to `setPrivacySensitive` (diff reports here [184]) and, by searching in the literature, it doesn't look like there's a way to bypass this security measure yet;
- **Reading protected data of other apps** is possible by:
  - Exploiting the accessibility service for reading what is rendered on screen (e.g, WhatsApp chats). From the point of view of an `AccessibilityService`, a window's content is presented as a tree of `AccessibilityNodeInfo` [185] components. It is possible to get the window's content [186], traverse the tree and call the `getText` method on each node (e.g., WhatsApp messages);
  - Using notifications as a side-channel for collecting information (e.g., WhatsApp messages). Notifications can be read even if the spyware app is not active. However, the spyware must be registered as an accessibility service. By listening to the `TYPE_NOTIFICATION_STATE_CHANGED` [187] event, which represents the event showing a transient piece of information to the user (i.e., `Notification` [188]), it is possible to intercept and read the content of notifications [189, 190];
- **Recording keystrokes** can be done by using the accessibility service. Specifically, the spyware can instantiate a call back for the `AccessibilityEvent` [191] `TYPE_VIEW_TEXT_CHANGED` [192]. The type of `AccessibilityEvent` can be retrieved with the `getEventType` method.

The main purpose of an accessibility event is to communicate changes in the UI to an `AccessibilityService`. In this case, the `TYPE_VIEW_TEXT_CHANGED` event is triggered whenever the text inside an `EditText` [193] component is changed. `EditText` is just a user interface element for entering and modifying text. So basically, whenever the desired event occurs, by calling the `getText` [194] method it is possible to retrieve whatever text the user has typed so far, independently from which app she is using.

This can be prevented by setting the `importantForAccessibility` [195] attribute of the `EditText` component to false (value=2), with the `setImportantForAccessibility` method. The `EditText` component inherits this method from the `TextView` component, which inherits it from the `View` component. When that attribute is false, the component doesn't fire any `AccessibilityEvent`, making it impossible to call the `getText` method;

- **Taking screenshots** can be done in different ways:
  - Before Android 10, apps could take screenshots by using the `MediaProjection` [196] class. A screen capture session could be started by creating the proper intent and by passing it to an activity. However, spyware generally run as services and don't have any activity. So, some spyware create temporary activities which are removed after taking the screenshot [197, 198];
  - With Android 10, an application can start an activity only if certain conditions are met [199]. A spyware could still be able to use the `MediaProjection` class by either getting the `SYSTEM_ALERT_WINDOW` permission or by registering as an accessibility service;
  - Starting from Android 11, instead of using the `MediaProjection` class (which is not so trivial because of the aforementioned problems), spyware can take screenshots by simply using the `takeScreenshot` [200] method of `AccessibilityService`. So, even in this case, the spyware must be registered as an accessibility service (i.e., it must get the `BIND_ACCESSIBILITY_SERVICE` permission);
- **Reading SMS messages** can be done by using the `contentResolver` [201] class as shown in these tutorials [202, 203];
- **Reading the contacts list** can be done by using the contacts provider [204];
- **Reading stored files** is possible by using the corresponding content provider [205];

9.4 Sending commands

Remote commands can be sent to the spyware in different ways:

- **Firebase Cloud Messaging** (**FCM**) [206]: A third party library which can be used for sending data messages, with a payload of up to 4096 bytes, to a client app;
- **Short Message Service** (**SMS**) messages [207]: Sending commands through SMSs makes it possible to control the target device even when it's not connected to the Internet. However, the malicious app must require and get the `RECEIVE_SMS` permission. Moreover, this techniques requires the C&C to authenticate (e.g., by leaving a specific passcode in each SMS) in order to avoid accidental commands execution due to SMSs received from other sources. This method is also risky, as it requires SMS messages to appear harmless to the end user, in order not to arise any suspect;
- **Remote Procedure Call** (**RPC**): RPC protocols over WebSockets, like JSON-RPC [208], may be used to invoke commands execution on the vitim's device;
- **Custom solutions**: Attackers can invent their own communication protocol to send commands from a remote C&C server to the target device. Commands are usually obfuscated or even encrypted;

Chapter 10

Spyware applications

10.1 Some available tools

The tools which we will present have been designed as applications for parental-control and/or employees' mobile phones monitoring. Most of these tools are not free to install, however:

- Some of them (e.g., mSpy and SpyX) provide a brief demo which can be tried on their websites. Such demos simply show the dashboard which is used to monitor the target device, just to give the idea of which functionalities are available and what kind of information can be collected after installing the application on a device;
- Others (e.g., Hoverwatch) provide a demo APK with a limited set of functionalities;

10.1.1 mSpy

mSpy [209] (Figure 10.1a) supports Android and iOS, but with some OS version limitations.

Main requirements

- Physical access to the target device. A rooted Android phone is needed to read chats on Viber, Kik, LINE, and Skype, and to monitor the target device's emails (emails can be monitored without a rooted device as well although not to the fullest extent);
- Android with OS 5 or higher. All Android target devices are supported, although some may have aspects regarding permissions due to the fact that they have their own customized OS. Besides, install steps may be slightly different depending on the make and model of the target device's;

How to install?

1. Download the installation file;
2. Install mSpy on the target device and grant all permissions when asked;
3. Log into mSpy on PC to see the target device's activity;

Declared features

Category	Feature	Description
General features	Call logs	State (i.e., incoming, outgoing, missed), number, name (if any) associated to that number, duration, and date of each call. **Note**: In case the call details get deleted within 2 minutes after having been completed, they might not be fetched.
	Contacts list	Phone number(s) and, if any, email(s) and home address of each contact.
	Device info	Model, OS, IMEI, ISP, battery %, Wi-Fi status.
	Events	Title, description, location (if provided), timestamps for the beginning and the end of the event.
	Keylogging	Provides the text, the app where it was typed and the timestamp.
	Keyword Tracking	Send an alert message to the specified e-mail, when a specific keyword is typed. Provides the context of the keyword (e.g., browser search, social media).
	SMS/MMS	Logs include every sent/received text, picture and video with the corresponding phone numbers and timestamps.
	Social Media Chats	WhatsApp, Facebook, Instagram, Telegram, Snapchat, Line, Skype, TikTok, Viber, Tinder, and Kik.
	Wi-Fi Networks	BSSID of the access point, name of the network, last connection timestamp, latitude and longitude.
Internet Usage	Browser Bookmarks	Website title and URL of the saved web page.
	Browser History	Website title, URL of the web page, last visit timestamp and total number of visits.
	Emails	Sender, recipient, subject, date, body and attachments.
Locations	Geo Fencing	Define allowed and restricted areas. Send e-mail notification when the target device either exits an allowed area or enters a restricted area.
	GPS Locations	Latitude, longitude, accuracy, address, timestamp and Google Maps URL.
Recording	Ambient recording	Supports both audio live hearing and recording.
	Call recording	Available for iOS only.
	Remote camera	Supports both camera live watching and recording.
	Screen recording	Visual snapshot of phone activities, with timestamp and application currently in use.
Restriction	Block Applications	Possibility to block installed applications.
	Block WiFi	Deny access to specific networks.
	Block Websites	Deny access to specific websites.
Storage	Installed APPs	App logo, name and version.
	Media	Photos and videos with the corresponding creation/download date.

Uninstallation

According to mSpy's website FAQ section: "In order to uninstall mSpy™, one needs to follow the specific instructions that only our customers get after the purchase. Even if there is an attempt to remove the application, you will get notified about it in your Control Panel".

10.1.2 SpyX

SpyX [210] (Figure 10.1b) collects data from the cloud drive of the target mobile phone, including location, SMS, call logs and so on, and then transfers the data to the registered SpyX account.

Based on what's declared on SpyX website: "SpyX tracks data in real time, so you can see the latest online activities on your target phone and get the most up-to-date information possible. You can not only monitor the real-time location of your target users but also spy on their dynamic real-time chats".

Main requirements

The only requirement is:

- iCloud/Google account password, and verification code of the target device. If the same account has been signed in on other devices before, you can use those devices to complete the verification steps. Physical access to the target phone is not always necessary;

How to install?

Actually, there's no need to install any app on the target device. The data collection and transfer process of SpyX is solely based on cloud technology:

1. Log in to the SpyX account;
2. Insert the iCloud/Google account credentials of the target device;

Declared features

By reading the features declared on SpyX's website, it looks like SpyX possesses all the features of mSpy, except: ambient recording, remote camera, Wi-Fi block, websites block. Moreover, while mSpy declares to be able to show photos and videos only, SpyX declares to be able to recover all the target device's files stored on cloud and not just photos and videos.

Uninstallation

SpyX's website talks about an "uninstallation process" in the FAQ section: "Because SpyX is a completely invisible mobile phone tracker, it is difficult to be detected and removed by the monitor. If you encounter any problems during the uninstallation process, you can always contact SpyX's customer service team for assistance". However, no further details on how to actually remove SpyX are provided.

10.1.3 Hoverwatch

Hoverwatch [211] (Figure 10.1c) does not support iOS, but only Android, Windows and Mac (no OS versions are specified on the website).

Main requirements

- Physical access on target device;

How to install?

The full installation process is pretty long [212] and extra steps are needed for installing Hoverwatch on Samsung devices [213] (the guide only covers Samsung devices with Android versions from 7 up to 11, plus Android 13), so we'll simply list the main steps:

1. Disable Google Play Protect and notifications from Google Play Store;
2. Allow downloads from unknown sources;
3. Download and launch Hoverwatch installer;
4. Grant all requested permissions;
5. Log in to Hoverwatch's website to monitor the target device;

Declared features

As previously stated, the demo version of the app only provides a limited set of functionalities:

- Call Data and Recording;
- Monitoring WhatsApp, Facebook, Viber, SMS and MMS: the app stores all messages;
- Front Camera Photos;
- SIM card changes: a notification will be sent every time the SIM changes;
- Location tracking;
- Internet Activity: list of visited websites;

Additional features of the official version include call recording, contacts list tracking, periodical screenshots, calendar events tracking, and some others...depending from the activation plan.

Uninstallation

Hoverwatch can be uninstalled by triggering the hidden application screen on the target device and by clicking on the uninstallation button [214]. The hidden screen can only be triggered by knowing a secret application PIN which is assigned to the target device after the installation [215].

10.1.4 MobileTracker free

MobileTracker free [216] (Figure 10.1e) supports Android versions up to 14, even though there are some limitations on that specific version [217]. It doesn't look like iOS is supported.

Main requirements

- Have created a Mobile Tracker Free account with a valid email address;
- Have access to the target phone;

How to install?

The main installation [218] steps are the following:

1. Enable downloads from unknown sources;
2. Disable Google Play Protect;
3. Login on the MobileTracker website and app download + installation;
4. Grant all permissions and configure the app based on the desired functionalities;

Some extra steps may be required, based on the phone model and OS version. For example, in case of a Samsung mobile phone with Android 14 [219]:

1. Disable battery saving features for background apps;
2. Disable memory consumption optimizations for background apps;
3. Allow background mobile data usage;
4. Allow access to location all the time;
5. Turn off app protection settings;
6. Hide the app on the home screen;

Declared features

MobileTracker offers a wide variety of features [220]: SMS/MMS tracking, SMS alert, call logs tracking, call recording, calls blocking, GPS position history, real-time GPS tracking, access to stored pictures, social media messages tracking, ambient recording (both audio and video), screen recording, remote access to camera for taking pictures, SMS commands, stored files explorer and downloader, schedule phone use restriction, installed apps monitoring and blocking, browser history and websites blocking, calendar events monitoring, access to contacts list and analysis tool for gathering statistics about most used apps and most performed actions with the possibility of downloading a report. However, with the free plan, some of these features present some limitations [221].

Uninstallation

There are two possible methods [222]:

- Like with Hoverwatch, use the secret application PIN to display the hidden app screen and uninstall it from there;
- Revoke the device admin privileges of the app and uninstall it like a normal app;

10.1.5 iKeyMonitor

iKeyMonitor [223] (Figure 10.1d) is a parental control app for both Android and iOS.

Main requirements

iKeyMonitor fully works on non-rooted Android devices with all the features [224]:

- Android 2.3 or above;
- Physical access to the target phone;

How to install?

Installation steps are identical to the ones of MobileTracker free [225, 226]

Declared features

The features vary based on the plan [227, 228]:

- The free plan has a limited amount of features: call history, SMS tracking, GPS location tracking, geofencing, contacts list, calendar events, notes, reminders/voice memos, WiFi history;
- The full plans includes also: keylogger, capture screenshots, browser history, social media monitoring, call recording, ambient recording, block apps, photos and videos, remote access to camera, clipboard monitoring (including copied and pasted texts), email monitoring, and some others;

Uninstallation

There are two methods:

- Use the secret app PIN to display the hidden app screen and uninstall it from there [229];
- Unhide the app icon [230] and uninstall it like a normal app;

10.1.6 Hammer Security

Hammer Security [231] (Figure 10.1f) is an anti-theft application for Android, available on the PlayStore.

Main requirements

For Hammer Security to work, the following points have to be taken care of:

- Grant all the requested permissions, and enable the security features;
- Location services should be enabled, along with GPS access;
- Hammer subscription should be active, either via ads or premium subscription;
- Network data should remain enabled, i.e., app should not be on airplane mode;
- Have enough data to send commands and media to Internet;

How to install?

It can be installed for free from the PlayStore.

Declared features

- Fake shutdown: Simulates the phone's shutdown state with a fake shutdown button upon attempting to switch it off, allowing the user to remotely track and fetch data from her device via Hammer's web account;
- Intruder alerts: If someone enters the wrong PIN several times, a selfie of the person trying to unlock the device will be emailed to the user;
- Lost phone tracking: Fast and effective recovery of lost device via real-time location tracking from Hammer's web dashboard;
- Fake airplane mode: Tricks thieves into thinking they've disconnected the phone, but it stays connected and trackable. Even with the feature activated, allowing you to track its location in real-time;
- Emergency SOS: In case of emergency, send automated alerts to the designated contacts via WhatsApp. User's location and audio messages can be shared to provide personal safety;
- Low battery alerts: Send alerts when running out of battery;

Uninstallation

Hammer Security has a camouflage feature which, if used, makes it appear as a calculator app. Moreover, it can only be uninstalled with one of the following methods:

- Uninstall from PlayStore;
- Click on the "uninstall" button inside the app to remove it from Device Admin and uninstall it normally from the phone, without going to the PlayStore;

10.1.7 Cerberus app

Cerberus app [232] (Figure 10.1g) is available on the PlayStore in many different versions:

- Cerberus Phone Security (Antitheft);
- Cerberus Personal Safety (Persona);
- Cerberus Child Safety (Kids);

Main requirements

- Grant the app the asked permissions. There's no need to disable Play Protect;

How to install?

All versions can be installed for free either from the PlayStore or from the app website. However, after the free-trial period, they require an account license to be bought from within the app.

Declared features

- Cerberus Phone Security (Antitheft): GPS real-time tracking, intruder selfie, fake shutdown, remote alarm, remote lock, geofences, smartwatch integration (powered by Wear OS), forced unlock help;
- Cerberus Personal Safety (Persona): Allows to specify some trusted contacts, a place to reach, and the time by which it should be reached. If the destination is not reached by that time, the app will automatically send SMS messages to the trusted contacts with a link that they can use to track the device's location in real time;
- Cerberus Child Safety (Kids): GPS real-time tracking, app usage insights, geofences, permission monitoring;

Other features, i.e. SMS and calls monitoring, have been removed because of Google policies [233]. Anyway, the app is still evolving and it's constantly updated [234].

Uninstallation

Cerberus app can only be uninstalled after disabling Device Administration [235]:

- from within the app itself, after performing login;
- or from the phone settings;

10.1.8 FlexiSPY

FlexySPY [236] (Figure 10.1h) is a monitoring app used for parental control and employee monitoring that supports both Android and iOS. Unfortunately, it doesn't provide a free-trial app.

Main requirements

- Physical access to the target phone;
- Android OS up to 14 [237, 238]. Rooting is required only for using advanced features [239];
- iOS version up to 14.8. iOS 15 and above are not supported. Jailbreak is required;

How to install?

For Android there are two modes of installation:

- Normal mode has the simplest installation, however some features may not be available;
- Advanced mode is a more complex installation and requires device rooting before installing FlexiSPY on the device. Automatic rooting can be performed by an operator after installing the remote installation service [240];

But in any case, the main installation steps are always the same (i.e., disabling PlayProtect and device security services, enable installation from unknown sources, grant device admin permissions, etc...).

Declared features

FlexiSPY disguises itself as an innocuous "Sync Services" app and provides different features based on the purchased plan [241].

Uninstallation

On Android, it can be performed either from the online dashboard (remote uninstallation [242]) or from the target phone after disabling the accessibility service [243].

10.2 Tools summary and testing preview

10.2.1 Spy apps OS compatibility and pricing

Table 10.1 illustrates compatibility details and pricing of all the presented tools plus some extra ones. The "Cheapest plan" column always refers to the cheapest plan available, whether it is monthly (/mo often refers to a single-month plan) or annual (/ye). All prices have been converted in an equivalent amount of US Dollars.

App	Android	iOS	Free trial	Cheapest plan	Ref
Bark	✓	✓	✓	$ 5.00/mo	[244]
iKeyMonitor	✓	✓	✓	$ 10.00/mo	[227]
Key24	✓	✓	✓	$ 24.45/mo	[245]
MMGuardian	✓	✓	✓	$ 4.99/mo	[246]
TiSpy	✓	✓	✓	$ 11.73/mo	[247]
Cerberus	✓	✗	✓	$ 5.00/ye	[248]
Hammer Security	✓	✗	✓	$ 3.70/mo	[231]
Hoverwatch	✓	✗	✓	$ 32.73/mo	[249]
MeuSpy	✓	✗	✓	$ 12.20/mo	[250]
MobileTracker free	✓	✗	✓	$ 6.56/mo	[221]
Spapp	✓	✗	✓	$ 16.39/mo	[251]
SpyHuman	✓	✗	✓	$ 9.99/mo	[252]
Eyezy	✓	✓	✗	$ 36.71/mo	[253]
FlexiSPY	✓	✓	✗	$ 49.95/mo	[254]
KidsGuard Pro	✓	✓	✗	$ 49.99/mo	[255]
MobileSpy	✓	✓	✗	$ 9.99/mo	[256]
MobiStealth	✓	✓	✗	$ 59.99/mo	[257]
mSpy	✓	✓	✗	$ 45.00/mo	[258]
Spyera	✓	✓	✗	$ 97.27/mo	[259]
Spyic	✓	✓	✗	$ 39.99/mo	[260]
SpyX	✓	✓	✗	$ 49.98/mo	[261]
Spyzie	✓	✓	✗	$ 39.99/mo	[262]
TheOneSpy	✓	✓	✗	$ 35.00/mo	[263]
uMobix	✓	✓	✗	$ 49.99/mo	[264]
Xnspy	✓	✓	✗	$ 29.99/mo	[265]
TheWispy	✓	✗	✗	$ 15.99/mo	[266]
Qustodio	✓	✓	*	*	[267]

Table 10.1: Spy apps OS compatibility and pricing

* Qustodio offers a free version app with limited features and a premium app whose basic plan costs € 42.95/ye.

10.2.2 Android supported versions and Play Store reviews

Android spy apps do not require a **rooted device** to be installed, but in some cases rooting may allow the app to provide extra features. On the other hand, all apps except SpyX require physical access to the target device for performing installation. Information regarding OS version compatibility, when unavailable on the application's website, has been collected with VirusTotal by checking the `minSdkVersion` and the `targetSdkVersion` fields of the app manifest [184, 268]. Additional information regarding apps availability on the **Play Store** has been added on a second table, along with the total **number of reviews** and the **average rating**.

Note that **Cerberus** (standard + disguised versions) declares API level 26 for both fields (which corresponds to Android 8.0) but, as explained in Section 11.2, it works well on Android 14. As a matter of fact, on the official website of the application, updates history reveals that Cerberus has been updated to perform on higher Android versions as well [234].

FlexiSpy, **SpyX**, and **MobiStealth** do not report version compatibility details on their websites and, since they do not offer a free trial, such information is nowhere to be found. The research made by [172] successfully tested FlexiSpy on Android 12. All websites declare that the respective apps are compatible with "different versions of Android".

SpyX is the only app that doesn't require **physical access** to devices, as it is cloud-based, but it requires Google/iCloud credentials.

KidsGuard Pro, despite having over 500K downloads, has no reviews. The reasons for that may be the ones explained by [269].

The **Bark** app is only available in United States, South Africa, Guam, and Australia [244]. Supported Android versions have been checked by downloading the app through a VPN [270].

App	Android Version	Ref
Bark	5.0 - 13.0	[271]
iKeyMonitor	2.3 - 14.0	[224]
Key24	4.4 - 14.0	[272]
TiSpy	5.0 - 14.0	[273]
Cerberus	8.0 - 14.0	[274, 275]
Hammer Security	7.0 - 14.0	[276]
Hoverwatch	4.4 - 13.0	[277]
MeuSpy	5.0 - 14.0	[278]
MobileTracker free	5.0 - 14.0	[279]
Spapp	4.4 - 12.0	[280]
SpyHuman	3.0 - 14.0	[281]
FlexiSPY	?	[282]
KidsGuard Pro	6.0 - 14.0	[283]
MobileSpy	4.0 - 14.0	[284]
mSpy	5.0 - 14.0	[285]
Spyic	4.0 - 14.0	[286]
SpyX	?	[287]
uMobix	4.0 - 14.0	[288]
TheWispy	4.0 - 14.0	[289]
Eyezy	8.0 - 14.0	[290]
MMGuardian	8.0 - 13.0	[291]
Xnspy	4.0 - 14.0	[292]
TheOneSpy	5.0 - 14.0	[293]
MobiStealth	?	[294]
Spyera	? - 14.0	[295]
Spyzie	4.0 - 14.0	[296]
Qustodio	7.0 - 14.0	[297]

Play Store App	N. Reviews	Avg. Rating	Ref
Bark	≈ 7380	4.1/5.0	[298]
Cerberus Anti-theft	508	4.0/5.0	[299]
Eyezy	1963	2.0/5.0	[300]
Hammer Security	154.547	4.7/5.0	[301]
KidsGuard Pro	0	`null`	[302]
MMGuardian Child	11803	2.3/5.0	[303]
MMGuardian Parent	25674	3.7/5.0	[304]
Qustodio Kids	47755	2.1/5.0	[305]
Qustodio Parents	15881	3.3/5.0	[306]

10.2.3 iOS supported versions and App Store reviews

Differently from Android spy apps, iOS spy apps require (in most cases) a **jailbroken device** to perform installation on the device itself. Such applications do not require **jailbreak** (**JB**) if they either use the **iCloud sync** (i.e., credentials are needed), **Wi-Fi sync**, or if they are available on the App Store. In the former case, supposedly, all iOS versions should be supported as the iCloud backups are getting monitored instead of the physical device (in the table below, such entries are labeled with "All"). Some applications even suggest ways and/or tools to perform iOS jailbreak but none of them works on newer iOS versions [307] (i.e., iOS 17 and above). The first table below lists, for each app, the supported iOS versions (if any, otherwise the `---` symbol is used) with (i.e., ✓) and without (i.e., ✗) jailbreak. When unable to retrieve such information, the '?' has been used instead. For some apps (e.g., MMGuardian, Eyezy, etc...) version details have been retrieved from the App Store, under the "compatibility" section.

It's interesting to notice how **KidsGuard Pro reviews** are visible on the browser [308] but not from the App Store iOS application because the summary rating has been reset [309]. Available reviews are mostly 1 star reviews, claiming the app is a scam and money are not refunded.

mSpy is actually available on the App Store. However, that application has been developed by a different company than the one which owns the mSpy website [310]. As a matter of fact, no reference to the App Store application can be found on the website.

The **Bark** app is only available in United States, South Africa, Guam, and Australia [244].

App	iOS (JB ✗)	iOS (JB ✓)	Ref
Bark	≥ 12	---	[311]
iKeyMonitor	---	≤ 14.x	[224]
Key24	≥ 7	6.x - 17	[272]
MMGuardian	≥ 11	---	[312]
TiSpy	?	?	[313]
Eyezy	≥ 14	?	[314]
FlexiSPY	?	?	[282]
KidsGuard Pro	≥ 13	12 - 14	[283]
MobileSpy	≥ 7	---	[315]
MobiStealth	?	---	[294]
mSpy	All	11 - 14.8.1	[285]
Qustodio	≥ 12	---	[316]
Spyera	---	≤ 14.x	[295]
Spyic	All	---	[317]
SpyX	All	---	[318]
Spyzie	All	---	[319]
TheOneSpy	---	15.2 - 17	[320]
uMobix	≥ 7	---	[288]
Xnspy	≥ 6	6 - 9.0.2	[292]

App Store App	N. Reviews	Avg. Rating	Ref
Bark Kids	≈ 7200	1.3/5.0	[321]
Bark Parents	≈ 24300	4.2/5.0	[311]
Eyezy	28	3.6/5.0	[314]
KidsGuard Pro	38	3.6/5.0	[308]
MMGuardian Child	961	1.4/5.0	[312]
MMGuardian Parent	809	4.3/5.0	[322]
mSpy	≈ 9200	4.1/5.0	[323]
Qustodio Kids	≈ 7300	2.7/5.0	[316]
Qustodio Parents	≈ 6300	4.3/5.0	[324]

10.2.4 Android and iOS spy apps testing preview

Note for Android 14 testing (Chapter 11):

- Table 10.2 lists all Android spy apps which have been tested in depth, but not all of them have been included in this document due to the reasons below. All remaining Android applications listed in Table 10.1 have not been tested because they haven't been considered significant, in terms of features, stealth, and persistence, for this analysis. Selected applications already represent an exemplary sample of all possible categories of Android spy apps which do not exploit any system vulnerability for achieving their purpose and which offer a free trial, whether it is time-limited or it offers a limited set of features.

Tested App	Section
Hammer Security	11.1
Cerberus Anti-theft	11.2
MobileTracker free	11.3
iKeyMonitor	none
TiSpy	none

Table 10.2: Android tested spy apps

- **Hoverwatch** hasn't been tested due to the extremely poor set of features and because of its compatibility issues with Android 14.
- **iKeyMonitor**, despite having set up the phone as required, fails at running in the background and no remote command works. The app also fails at hiding from the app drawer and it can be uninstalled like any normal app. It doesn't ask for device admin privileges but only for accessibility service and some more common permissions.

- **TiSpy** is very similar to MobileTracker free, as it offers the same exact features (with a different GUI). All features work as intended. The only real differences among the two are:
  - TiSpy free trial expires after 24 hours while MobileTracker free offers a free trial with no time restrictions but a limited number of actions per day.
  - Stealth: TiSpy can also close the background apps list whenever the user wants to display it. However, this doesn't prevent the user from seeing the malicious app in that list because of a small delay which, by the way, could be exploited to forcefully stop the app by pressing the corresponding "stop" button.
  - Persistence: TiSpy implements the same protection mechanism of MobileTracker free by displaying a text box on the screen which asks for a secret PIN when trying to uninstall the app. However, TiSpy's implementation is worse because this security measure is not triggered when trying to uninstall the app from the app drawer nor when trying to revoke the device admin privileges from the app, but only when the user displays the application's info screen.

  Due to these reasons, TiSpy testing details haven't been included in this document.

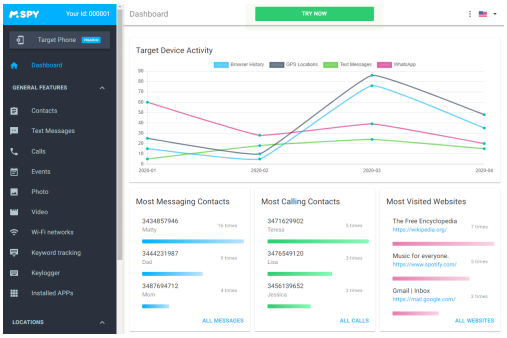
- As explained in Chapter 11, no application successfully hides from the app drawer on an unrooted device. However, each app could be set to hide its icon from the app drawer by manually enabling the corresponding Android setting from the app drawer itself [325]. Clearly, this does not prevent any user from unhiding the app icon by disabling such setting.

Note for iOS 17.6.1 testing (Chapter 12):

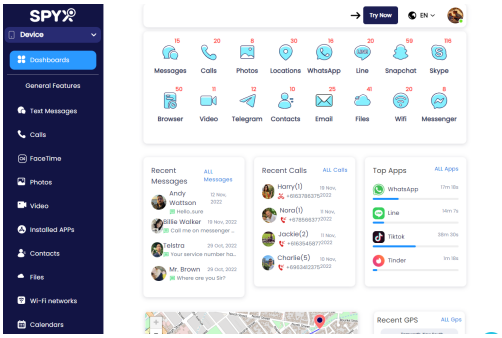
- Table 10.3 lists all iOS spy apps which have been tested in depth. Selected applications are downloadable for free from the App Store and don't require a jailbroken device nor the usage of an iCloud account. However, their features are very limited.

Tested App	**Section**
MMGuardian	12.1
Qustodio	12.2
Kidslox	12.3

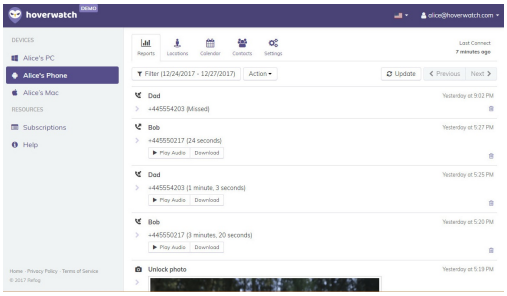
Table 10.3: iOS tested spy apps



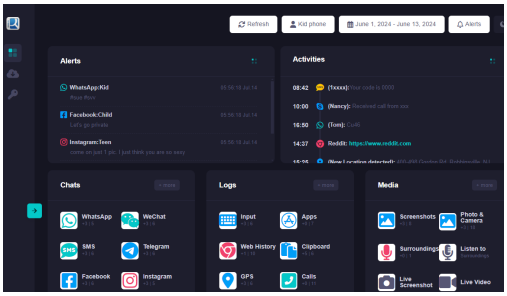
(a) mSpy (Section 10.1.1)



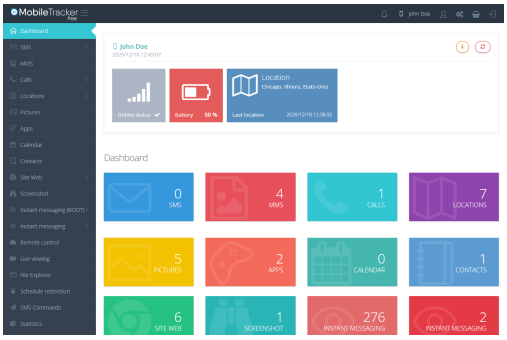
(b) SpyX (Section 10.1.2)



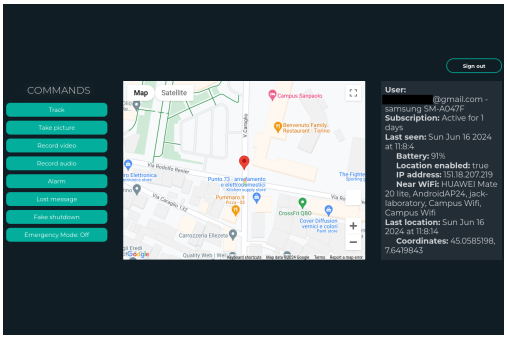
(c) Hoverwatch (Section 10.1.3)



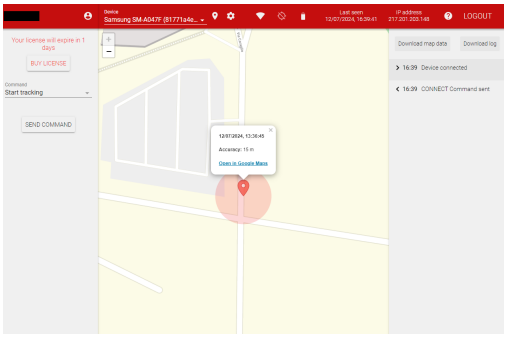
(d) iKeyMonitor (Section 10.1.5)



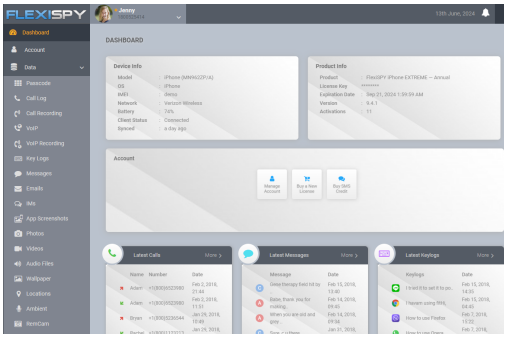
(e) MobileTracker free (Section 10.1.4)



(f) Hammer Security (Section 10.1.6)



(g) Cerberus app (Section 10.1.7)



(h) FlexiSPY (Section 10.1.8)

Figure 10.1: Spy apps overview

10.3 A deeper analysis

The 2023 research conducted by [172] assessed the functionalities and the insecurities of 14 different Android spyware apps. Researchers performed static analysis by downloading each APK and by decompiling the corresponding bytecode. Further testing has been done by installing those apps on a Pixel 4 mobile phone running Android 11. In this way, researchers have been able to confirm the results of the static analysis. Here, we're going to focus on the aforementioned apps by gathering all results obtained by this research.

First of all, we'll have a look at the apps versions:

App Name	APK Version	SHA-1 Hash of the APK	Last version
Cerberus	3.6.9	`40345be0287e47224d951cd3644ac0bd7f49e150`	3.8.0
FlexiSPY	4.16.1	`07786dc314f8cab968d0c5a310a71601543bea0e`	5.3.3
HoverWatch	7.2.338	`33c12f3fbe2b510ceb3111f8198f974040ab05ba`	7.4.363
iKeyMonitor	9.8	`2f6f807f2ac1b5a423e006a667db15c3f7229c6d`	15.1
Mobile-tracker-free	153	`e18637c9576a295b02ab3dc8282eb4ca242942dd`	154
mSPY	6.3.2	`4e675734487baaa93533f5c187c376d37ce28eb3`	?

Table 10.4: APK's version number (2023 research) with its SHA-1 + last version available today

Unfortunately, version information is not always available without installing the APK.

10.3.1 Analyzed features

Here the most peculiar features which have been analysed during the research [172]:

- Cerberus employs a series of mechanisms to stop users from deactivating it as a Device Admin app or uninstalling it. These include trying to lock the device by invoking the `lockNow` [326] method of the `DevicePolicyManager` [327] class and starting an activity that blocks users from clicking on any buttons;
- Mobile-tracker-free tries to stop users from uninstalling it by starting an activity that blocks the uninstallation screen and requests a password set by the abuser to proceed;
- Hoverwatch is the only app that creates a widget that has an `EditText` control that takes user input. If users enter the correct code, it launches itself;
- iKeyMonitor offers the ability to build apps that have customized icons and names. This helps for disguising the app as a legitimate one;
- Hoverwatch and iKeyMonitor abuse accessibility to automatically click buttons (e.g., automatically clicking on the grant permission button when the app is requesting `MediaProjection` permission), a behavior much like that of **malware**;
- Mobile Tracker free uses a loader app to install the actual app;
- mSPY has implemented functions to unhide its icon via a remote command;

10.3.2 Analyzed insecurities

The research has also highlighted some insecurities in the applications above [172]:

- Cerberus uses public URLs that are a combination of device ID and Unix timestamp (e.g., `https://domain.com/<DeviceID>-<timestamp>.ext`) and stores audio files insecurely by using the device IMEI as the device ID. An attacker could obtain the device ID from data dumps made public in breach attacks [328] or with physical access to the device and successfully retrieve data from the server by iterating over possible timestamp values;
- FlexiSPY uses HTTP with a custom encryption protocol with a flaw that makes it possible to intercept personal data sent over the network [329]. Moreover, images and audio can be retrieved from public URLs even though such endpoints are temporary;
- Hoverwatch audio files stored on the server can be retrieved through public temporary URLs;
- Mobile Tracker free authenticates its SMS commands with a default password. Unless the abuser explicitly changes it, an attacker can successfully send commands with the default SMS password. Moreover, unauthenticated access to streaming data is possible;
- mSPY leaks the upload path for its images in an HTTP response back to the victim's device when the image upload succeeds;

10.4 Legitimate app or spyware?

Let's try to draw some conclusions based on what we've learnt so far. What is it that it makes an app, like the ones presented above, look more like a spyware?

- As we previously mentioned, using the accessibility service for self-granting new permissions by automatically clicking on the grant permission button is a feature that belongs to spyware (e.g., the already discussed SpyNote);
- Collecting information which are not necessary for the application's declared purpose is another suspect behavior;
- SMS commands are definitely a spyware feature. That's why Google decided to change their policy [330] on SMS permission for the PlayStore apps. Commands could be used to perform malicious actions (e.g., launching DoS attacks, etc...);
- Applications that try to hide themselves (e.g., icon hiding) and to be more persistent (e.g., resisting to device reboot or even factory reset) exhibit spyware-like behavior;

Keep in mind that most of the aforementioned applications can be downloaded only from third-party stores, after enabling downloads from unknown sources and after disabling the device's security measures (e.g., PlayProtect, etc...). That's because, otherwise, they would probably be flagged as "spyware". Companies that distribute such apps always try to reassure customers about the fact that their applications are not malicious, but should we really trust them?

10.4.1 Stealth and persistence of an app

The [331] research performed a detailed analysis of the techniques employed by spyware developers to enforce stealth and persistency, two essential features for a spyware.

Hiding app icon

Researchers observed that apps can hide their icon from the application launcher:

- At runtime, by invoking the `setComponentEnabledSetting` [332] method with `COMPONENT_ENABLED_STATE_DISABLED` [333] and `DONT_KILL_APP` [334]. This method works only with Android 9 and earlier versions. Since Android 10, an app's icon will still appear in the app launcher even if the launcher activity is disabled at runtime. Because of this, most spyware apps now try to disguise themselves as legitimate apps by using innocuous names (e.g., "Wi-Fi", "SyncService", "InternetService") and icons;
- By deleting the `CATEGORY_LAUNCHER` [335] intent category from the app's manifest main activity (i.e., launcher activity gets disabled by default and not just at runtime). Some spyware apps that use this technique only declare an activity with an intent containing the `ACTION_MAIN` [336] action and the `LEANBACK_LAUNCHER` [337] category (other apps just use an intent with `ACTION_MAIN`). This method has been proven to work up to Android 12 but it hasn't been tested on the latest Android versions. Anyway, by doing so, spyware apps need a different way to launch themselves (e.g., using a loader to install the real app without a launcher activity);

Launching a hidden app

When a spyware app hides its icon from the launcher, there are some techniques which could be used to lunch it anyway:

- If the attacker has physical access to the target phone, he can send a remote command to unhide the application icon and then launch it manually. After doing so, another remote command is used for hiding the app icon again;
- The malicious app creates a `BroadcastReceiver` and listens for a trigger event (e.g., `SMS_RECEIVED`, `BOOT_COMPLETED`, etc...). When the event occurs, the receiver launches the app as a background service while keeping its icon hidden. However, starting from Android 3.1, an app cannot create a service without having an activity associated to it. So, an activity must be created first and then (after starting the service) either made transparent or destroyed. In this way, the app doesn't even appear in the list of running apps;

Hiding app from recents screen

Hiding a malicious app from the recents screen [338] can be done in different ways:

- By setting the `android:excludeFromRecents` [339] attribute to `true` for the activities to be hidden in the app manifest;
- By adding the flag `FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS` [340] to an activity;

Deleting communication traces

Spyware apps that support remote communication with the C&C exhibit similar behaviors:

- SMS/MMS reception notification is blocked by registering a `BroadcastReceiver` which listens for the reception event and, then, by calling the `abortBroadcast` [341] method;
- SMS/MMS deletion upon reception and sending by using the

  `delete(Uri.parse("content://sms/"), null, null)` [342]

  method of the `ContectResolver` class;
- Call block by listening for a `PHONE_STATE` change event and then by calling the `endCall` [343] method to cancel the incoming call. Starting from Android 10, this method has been deprecated. However, the `respondToCall` [344] method of the `CallScreeningService` class can be used instead. By passing to this method the proper `CallResponse` [345] object, it is possible to reject/block incoming calls [346];
- Call logs deletion follows the same reasoning as SMS/MMS deletion. Here's an example that uses a query string to select a call log with a specific number to be deleted:

  ```
  delete(Uri.parse("content://call_log/calls"), "number=?", "target_number");
  ```

- System logs deletion by using Android Runtime (ART) and the `adb logcat` command line tool `getRuntime().exec("logcat -c")` [347, 348] (only if the phone is rooted or before API version 16). This action deletes traces of hidden system activities;

Preventing stop and uninstallation

To avoid being stopped and/or uninstalled, a malicious app can:

- Disable the "Force Stop" and "Uninstall" buttons. Before Android 7.1, that was possible by granting Device Admin (DA) privileges to the app [349]. Starting from Android 7.1, the Uninstall button will remain enabled even if the app registers as DA;
- Other possible techniques are the ones employed by Cerberus and Mobile Tracker free;

Creating diehard services

These services automatically restart after being stopped by the Android system or after reboots:

- Apps can schedule to be restarted either when they are first started or when they are being terminated by the system. To schedule themselves to be restarted shortly after they are terminated, they override the `onDestroy` [350] function, which is called before the app is terminated by the system. For this purpose, scheduling frameworks like `JobScheduler` [351] and `AlarmManager` [352] can be used;
- Listening for `BOOT_COMPLETED` allows services to restart themselves after reboots;
- Apps that register as an `AccessibilityService` or `NotificationListenerService` can also survive device reboots. However, `AccessibilityService` is less reliable because it can be turned off by Android for battery saving reasons, unless the application is granted the `REQUEST_IGNORE_BATTERY_OPTIMIZATIONS` permission;

10.5 Techniques for spyware detection

Here are some possible approaches to detect spyware at runtime (i.e., dynamic analysis), as in most cases it is difficult to find where the malicious APK is stored [353]:

- Network traffic analysis [354, 355] and/or SMS interception [356, 357] are useful methods for intercepting communications between the compromised device and the C&C server;
- System calls and API calls interception [358];

Other techniques involve static analysis:

- Application manifest and permissions analysis;
- System logs analysis (if not deleted, they are a valuable resource);
- Taint analysis and pair action (PAPI) analysis [331];
- Device memory image analysis (e.g., with frameworks like Volatility [359, 360]);

Some of the aforementioned techniques could adopt machine learning algorithms for fulfilling their purpose.

Chapter 11

Testing spy apps on Android 14

All tests have been performed on a Samsung Galaxy A04s with:

- Android version: `14` (API level 34);
- Android security patch level: May 1, 2024;
- Kernel version: `4.19.198-27972583-abA047FXXS7DXE5`;

Note: No app was fully compatible with Android 14, as they were developed for previous OS versions. After having installed an app, the device displayed the following message: "Your device is not compatible with this version". Despite that, most features of the apps still worked.

All apps require biometric authentication for enabling the accessibility service.

11.1 Hammer Security

11.1.1 Installation and account setup

After getting installed, the application asks for a subscription. After creating an account by providing email and password, one among three plans must be chosen:

- **Free plan**: Requires the user to watch ads once per day in order to enable the app functionalities for that specific day only. Ads must be watched from the device where the app has been installed, by opening the app itself;
- Monthly plan: No ads but the subscription must be renewed every month;
- Premium plan: It's the most expensive one but, once payed, it doesn't need to be renewed;

All plans give access to the same functionalities. After choosing a plan, the application must be granted with accessibility service and device admin permissions, along with access to camera, microphone, geolocation and notifications. The application asks for the phone number of an **emergency contact** which will be notified on WhatsApp whenever a command is sent to the target device. It is possible to add more emergency contacts after the app is configured.

We chose the free plan and tried all the application's features.

11.1.2 Application's dashboard and commands testing

By going to the application's website, after logging in (with the same email and password which have been used for creating an account from within the app), it is possible to send commands to the target device by clicking on the following dashboard buttons:

- **Track**: Even though the application has been granted with geolocation access permission, this command works only if GPS is active on the target device. Once the command is sent, the location of the device is displayed on the map panel. The last location timestamp and coordinates get updated as well;
- **Take picture**: Takes a picture either from the front camera or the back camera;
- **Record video**: Records a 30 seconds video (audio included) either from the front camera or the back camera;
- **Record audio**: Records either 1 minute or 5 minutes of audio but quality is much worse than audio from recorded videos;
- **Alarm**: Triggers an alarm sound at maximum volume which lasts for about 30 seconds. If the user tries to lower the volume, the app will bring it to maximum level again (until the alarm ends);
- **Lost message**: Displays a message on the device's screen and, if the user tries either to turn the device off or to simply turn the screen off, the application will take a picture from the front camera (selfie);
- **Fake shutdown**: If the user tries to turn off the device, a fake "turn off" button is displayed. The device doesn't really get turned off and it can be kept in use for spying. There are some issues with this feature:
  - Before the fake button is displayed, the real "power off", "reboot", and "emergency call" Android buttons are displayed for a split second, even though there's not enough time to double click one of them before the fake button is displayed;
  - After the fake shutdown button is clicked, the device looks like it has been turned off. However, by repetitively trying to unlock the phone either with fingerprint or by pressing the side unlock button, some times it happens that the unlock screen gets displayed for a split second;
  - After fake shutdown is performed, there's no "fake boot" procedure which turns the device's screen back on after keeping the side unlock button pressed for some seconds. The only way to exit the fake shutdown state is by sending the **fake shutdown off** command from the website. After doing so, the unlock screen of the device can be displayed again after pressing the side unlock button and the device can only be unlocked by inserting the password;
  - If the phone gets charged while in the fake shutdown state, the application displays a message with the current level of battery. However, the original Android charging screen (with the battery percentage) is displayed as well;

  - Sending the `fake shutdown off` command before the device undergoes the fake shutdown procedure has no effect. This command does not prevent the fake shutdown button from appearing but it only serves for exiting the fake shutdown state. As the application's website suggests, if this command doesn't work, it is possible to:
    * Charge the phone: After plugging in the charger cable, the unlock screen is automatically displayed (if the `fake shutdown off` command has been sent);
    * Keep pressed the volume button and the unlock button simultaneously for some seconds: This triggers device reboot. The **major issue** here is that, after reboot, **all features of the app are disabled** until the device gets unlocked by inserting the password. Rebooting the device like that is possible even if the device is in the fake shutdown state and no `fake shutdown off` command has been sent;
  - The website includes also a **fake shutdown on** command which automatically triggers the fake shutdown procedure. However, even in this case, the unlock screen can be displayed for a split second by pressing the side unlock button;

- **Emergency Mode off**: It doesn't look like this command does anything other than turning off fake shutdown (which is already done by the `fake shutdown off` command);

Inside the right panel of the application's website GUI, some extra info are available:

- **User**: Email used for the subscription and target device's model;
- **Subscription**: Subscription's duration;
- **Last seen**: Information about the last time the device had an active connection (battery percentage, GPS status, IP address, near Wi-Fi access points);
- **Last location**: Timestamp and coordinates of the last location which was tracked with the `track` command;

The app does not prevent the user from enabling airplane mode or, generally, from cutting all connections with the network but it still works via SMS. Some of the aforementioned commands, instead of being sent from the application's website, can be sent by SMS too. However, there's **no SMS authentication** and commands can be triggered by SMS messages sent by any device. After a command is sent via SMS, the corresponding result is returned to the emergency contacts via WhatsApp. Here are all the available SMS commands:

Name	SMS string	Does it work?
Track	HS1 TRACK	✓
Alarm	HS1 ALARM	✓
Lost message	HS1 MESSAGE	✓
Fake shutdown on	HS1 FS ON	✓
Fake shutdown off	HS1 STOP	✓
Emergency Mode off	HS1 STOP	✓

Table 11.1: Hammer Security SMS commands

The TRACK command returns a Google maps link with the target's location. The MESSAGE command displays a default message on the target device's screen.

Here are some final considerations regarding Hammer Security app:

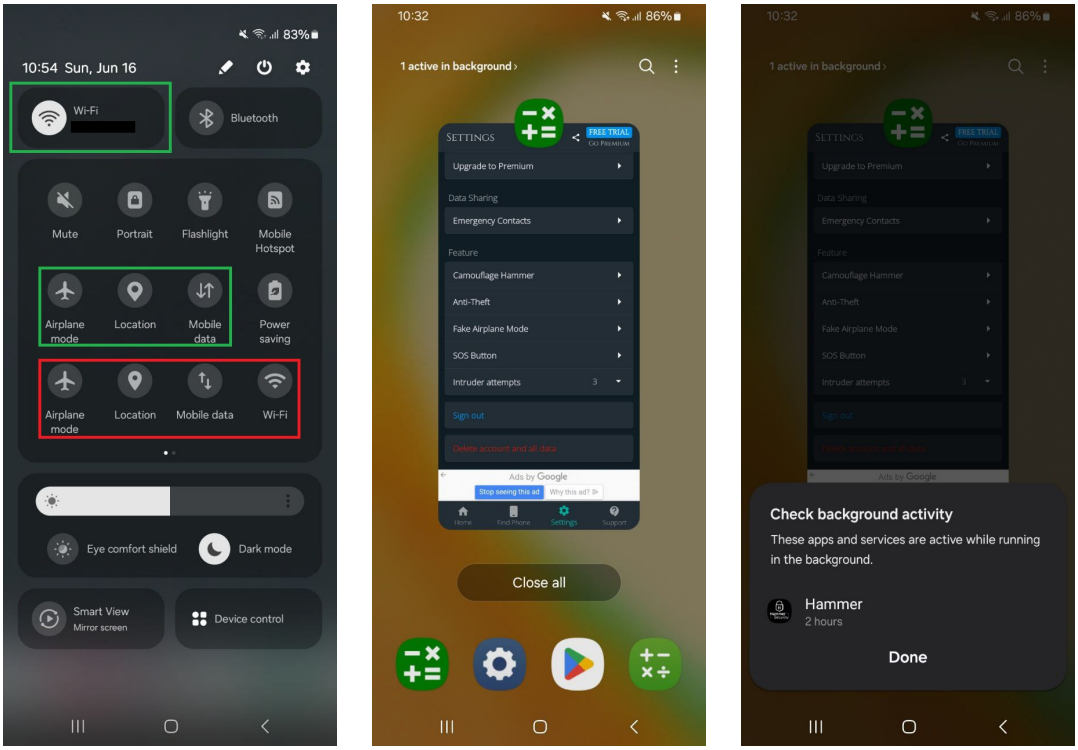
- All declared features are actually implemented;
- Location tracking is not real-time, as declared, but the last location gets updated only after sending the `track` command and only if the target device has an active GPS;
- The app uses Google Firebase APIs to send/receive commands and provides links to stored files (pictures, audios and videos) both by email (the one used for the subscription) and WhatsApp (of the emergency contacts). Provided URLs have the following structure:

  ```
  https://firebasestorage.googleapis.com/v0/b/hammer-security.appspot.com/
                  o/18%myFile.ext?alt=media&token=myToken
  ```

  where `myFile` is the name of the stored media, `.ext` is the extension of the file (`.jpg`, `.mp3`, `.mp4`) and `myToken` is the value of the authentication token, which is different for each file;

11.1.3 Stealth and persistence

- **Stealth**:
  - All SMS commands received by the target are not deleted from the SMS register.
  - The application icon can be changed into the one of a calculator app. However, the icon is visibly different from the Android system calculator app and the application's name is "HS Calculator" (Figure 11.1d). By opening the fake calculator app and inserting the application's secret 4-digit PIN, the login screen of Hammer Security is displayed.
  - The app can create a replica of the buttons for enabling/disabling airplane mode, mobile data, Wi-Fi, and GPS (which don't work as intended). Whenever one of these buttons is clicked, the application takes a picture from the front camera and records a 1 minute long audio. Everything is sent to the emergency contacts by WhatsApp. However, the buttons do not exactly look like the original Android ones and, moreover, in Android 14 the Wi-Fi button is not displayed together with the others but it has a reserved GUI area on the top left corner of the screen. In Figure 11.1a the original Android buttons have been enclosed within a green line while the fake buttons are highlighted by a red line.
  - As shown in Figures 11.1b and 11.1c, the app is still displayed in the recent apps screen and, most importantly, it is displayed in the list of background activities with its original name and icon. Moreover, as shown in Figure 11.1e, whenever the app accesses the camera and the microphone, the corresponding icons are displayed on the top right corner of the screen.
- **Persistence**:
  - The app does not work after reboot, unless the device gets unlocked.
  - The app can simply be **uninstalled** by uninstalling the "HS calculator" app from the applications list. The device will display the message: "HS Calculator is part of another app (Hammer). Do you want to uninstall it?". After accepting to uninstall the fake calculator app, the device will ask to uninstall Hammer Security itself too. By confirming the choice, the app gets completely removed from the device and all changes to the system made by it are undone.



(a) HS fake buttons (b) HS in recent apps screen (c) HS in bg activities list

(d) HS icon camouflage (e) HS access to camera and microphone

Figure 11.1: Hammer Security stealth and persistence flaws in Android 14

11.2 Cerberus App

11.2.1 App versions overview and installation

The Antitheft app offers the greatest number of features. The app is delivered in two versions:

- **Play Store version**: Requires installation of Cerberus Lock Screen Protector from the store itself and needs to be granted a huge set of permissions, including accessibility service and device admin. Gives a two days long free trial license;
- **Third party version**: Can be downloaded from the official website [361] and it can be installed only after disabling PlayProtect. It offers a disguised version of the app and requires even more permissions, but has more features. Free trial license is still granted;

Here's an overview on all permissions required by both versions:

Permission Name	Play Store App	Third Party App
ACCESS_ADSERVICES_AD_ID	✓	✓
ACCESS_ADSERVICES_ATTRIBUTION	✓	✓
ACCESS_ADSERVICES_TOPICS	✓	✗
ACCESS_BACKGROUND_LOCATION	✓	✓
ACCESS_COARSE_LOCATION	✓	✓
ACCESS_FINE_LOCATION	✓	✓
ACCESS_NETWORK_STATE	✓	✓
ACCESS_NOTIFICATION_POLICY	✓	✓
ACCESS_WIFI_STATE	✓	✓
BLUETOOTH	✓	✓
BLUETOOTH_ADMIN	✓	✓
BLUETOOTH_CONNECT	✓	✓
CALL_PHONE	✗	✓
CAMERA	✓	✓
CHANGE_NETWORK_STATE	✗	✓
CHANGE_WIFI_STATE	✓	✓
FOREGROUND_SERVICE	✓	✓
GET_ACCOUNTS	✗	✓
INTERNET	✓	✓
MANAGE_ACCOUNTS	✗	✓
MODIFY_AUDIO_SETTINGS	✗	✓
MODIFY_PHONE_STATE	✗	✓
NFC	✗	✓
POST_NOTIFICATIONS	✓	✓
PROCESS_OUTGOING_CALLS	✗	✓
READ_CALL_LOG	✗	✓
READ_CONTACTS	✗	✓
READ_PHONE_STATE	✗	✓
READ_SMS	✗	✓
RECEIVE_BOOT_COMPLETED	✓	✓
RECEIVE_SMS	✗	✓
RECORD_AUDIO	✓	✓
REQUEST_DELETE_PACKAGES	✓	✗
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	✗	✓
SEND_SMS	✗	✓
SYSTEM_ALERT_WINDOW	✓	✓
USE_BIOMETRIC	✓	✓
USE_FINGERPRINT	✓	✓
USE_FULL_SCREEN_INTENT	✗	✓
VIBRATE	✓	✓
WAKE_LOCK	✓	✓
WRITE_EXTERNAL_STORAGE	✗	✓
WRITE_SECURE_SETTINGS	✓	✓
WRITE_SETTINGS	✓	✓
WRITE_SMS	✗	✓

Table 11.2: Cerberus Antitheft permissions - versions comparison

11.2.2 Play Store application's dashboard and commands testing

By going to the application's website, after logging in (with the same email and password which have been used for creating an account from within the app), it is possible to send commands to the target device by clicking on the following dashboard buttons:

- **Start/Stop tracking**: Enable/Disable real-time geolocation tracking. By default, this feature doesn't work if GPS is not turned on. However, as explained on the application's website, Cerberus App can automatically activate/deactivate GPS without displaying any permission request on screen if it's granted the `WRITE_SECURE_SETTINGS` permission. The only way to do that is by using the Android Debug Bridge (adb) [362] and execute the following command:

  ```
  adb shell pm grant com.ssurebrec android.permission.WRITE_SECURE_SETTINGS
  ```

  An extra `-s` command option might be needed after `adb`, in case of multiple devices/emulators, to select the proper device by inserting its name [363]. Available devices can be listed with the `adb devices` command;

- **Get device info**: Displays the app name and version with the set of requested (and granted) permissions, together with the device's IP address. The WiFi network which the device is connected to and the nearby WiFi access points, however, are not displayed;

- **Lock/Unlock with code**: This feature is supposed to lock/unlock the device with a user-supplied secret PIN of at least 4 digits. However, after sending the command, the dashboard returns an error message which says that, after Android 7, it is impossible to change the locking/unlocking PIN of the device with an app. So, the lock command ends up locking the device but it can be unlocked with the device's password. The unlock command doesn't work at all;

- **Start alarm with a message**: Displays a white screen with a message on it and an alarm goes off as the phone keeps vibrating and turning on and off the flashlight. The alarm's volume cannot be changed and it's always at its maximum. The alarm gets stopped as soon as the white screen is tapped;

- **Display message**: Displays a message on a white screen. The device can also read the message and play it on the speaker, if the corresponding option is checked;

- **Start/Stop emergency mode**: When enabled, emergency mode sends an email to the account owner every 5 minutes with the updated device's location;

- **Recover from fake shutdown**: Disables the fake shutdown and makes the device unlockable again. The fake shutdown feature can be enabled in-app but only works when the device is in standby and someone tries to turn it off. If the device is unlocked and someone tries to turn it off, the fake shutdown screen does not appear. Moreover, the fake shutdown screen only shows a power off button which is really far from looking like the Android original one (it's a white rectangle with "Power off" written on it). Despite these problems, when the device is in the fake shutdown state it really looks like it's turned off as repetitively pressing the side buttons of the phone will not cause the screen to flicker at all (we'll always have a purely black screen). The device can be restarted by simultaneously pressing the lower volume and the unlock side buttons for a few seconds but, once restarted, Cerberus will still be active and functional as expected (even without unlocking the device);

- **Record audio**: Records an audio which can last from 1 to 300 seconds. The audio can then be downloaded from a link which is displayed on the dashboard. Every audio is downloaded as a video file (`.mp4`) which contains no visual frames. The link where to download the files from is always the same and allows to download the latest recorded audio. The link format is:

  `https://www.cerberusapp.com/getfile.php?d=DeviceID&t=a`

  where the `d` parameter (which presumably stands for "device") has the `DeviceID` as value, i.e., a 16 digits long hex string which is unique for each Cerberus account, and the `t` parameter (which presumably stands for "type") specifies to download an audio file (`a`);

- **Take picture**: Takes a picture with either the front camera or the back camera. The option for enabling the flash does not work. The picture is then sent as an email attachment to the account owner;

- **Grab screenshot**: Takes a screenshot of whatever is on screen at the moment. The screenshot is then sent as an email attachment to the account owner;

Other features, not displayed on the dashboard but available from within the app, are:

- **Wrong unlock code**: Consecutive wrong unlock attempts cause the device to trigger one or more of the following features: send location, send picture, start alarm, start emergency mode. An unlock attempt is considered valid from Android only if at least 4 digits/characters are entered (or a pattern which connects at least 4 dots is used);

- **Forced unlock help**: When the device is locked, a notification with a custom text is displayed. By tapping it before unlocking the phone, a fake shutdown procedure is started after 15 seconds. Emergency mode can be eventually triggered as well;

- **Smartwatch**: Connects the app to nearby wearable devices, like smartwatches. If enabled, it allows to: show a notification on the smartwatch when connection to it is lost, start an alarm from the smartwatch, start emergency mode from the smartwatch. This feature was not tested;

- **Geofences**: Defines one or multiple geofencing areas. When the target device enters or exits the area(s), the following actions can be triggered: send location, send picture, start alarm, start emergency mode;

- **Low battery alerts**: Performs the following actions when the device's battery level falls between 10% and 20%: send location, send picture, start alarm, start emergency mode;

- **Charger disconnected**: Performs the following actions when the charger is disconnected: send location, send picture, start alarm, start emergency mode;

The Play Store version of the app doesn't support SMS commands, as it doesn't even ask for the `READ_SMS` permission.

11.2.3 Third party application's dashboard and commands testing

The third party app implements all features of the Play Store app plus the following ones:

- **Get location history**: Provides a link to download a `.kml` file with the history of the device's locations. Returns a warning message if the file is empty. The file can be downloaded from a URL which has the following format:

  `https://www.cerberusapp.com/getfile.php?d=DeviceID&t=h`

- **Backup data/Stop backup**: If the backup option has been enabled from within the app, by linking Cerberus to a Google account, it is possible to backup SMSs/MMSs, call log, contacts list, photos and videos on Google Drive. Alternatively, backup on Dropbox can be configured as well;

- **Capture video**: It records a video, whose duration can go from 1 to 30 seconds, either from the front or from the back camera. The video is sent as an email attachment (as a `.3gp` file) to the account owner;

- **Capture screen recording**: This feature works only with a rooted device and couldn't been tested;

- **Get call log**: Displays on the dashboard the list of all the calls which are stored in the calls register. Each call is represented with: date, time, contact name (if any), phone number, and call type (incoming, outgoing, missed). Blocked calls type is unrecognised and displayed as `null`. If the contact is unknown, `null` is displayed instead of its name. It doesn't fetch calls which have been deleted from the register;

- **Get SMS log**: Displays on the dashboard the list of all the SMSs which are stored in the SMS register. Each SMS is represented with: date, time, contact name (if any), phone number, SMS type (sent, received) and SMS text body. If the contact is unknown, `null` is displayed instead of its name. It doesn't fetch SMSs which have been deleted;

- **Wipe device memory**: Successfully wipes the memory of the device, as if a factory reset was triggered by the user. Only the SMS version of this command has been tested and, since it works as intended, it is assumed that this version of the command works as well;

- **Wipe SD card**: Successfully erases all files inside the SD card;

- **Change app settings**: Changes the Cerberus app settings;

- **Hide from/Show in app drawer**: On Android 10 and later versions, the icon can be hidden only if the device is rooted;

- **Call phone**: Calls the specified phone number and can also activate the speaker;

- **Send SMS**: Sends an SMS with custom text to the specified phone number;

- **Enable/Disable bluetooth**: Successfully enables/disables Bluetooth;

- **Reboot device**: This feature works only with a rooted device and couldn't been tested;

- **Start shell**: Successfully opens a Linux shell on the target device. However, unless the device is rooted, you have no permission to execute most commands (Figure 11.2). The shell can be opened by submitting a GET request to the url below:

  `www.cerberusapp.com/shell/shell.php?id=DeviceID&token=TokenString`

  where `TokenString` changes every time a new shell is requested.

Figure 11.2: Cerberus Antitheft (3rd party) - Shell

- **Enable/Disable Wi-Fi hotspot**: If the mobile data option is turned on, this command successfully enables/disables Wi-Fi hotspot. However, if Wi-Fi is on (and mobile data off), the command fails unless the device is rooted. Specifically, the following message is displayed: "OVERRIDE_WIFI_CONFIG permission needed. System app required".
- **Get installed apps**: Displays on the dashboard the list of all installed applications with their corresponding package names.
- **Start application**: Successfully starts an application after providing its package name.
- **Start service**: The command requires the user to fill the "Package" and the "Service" text fields but doesn't specify what kind of syntax should be used for services. By trying to launch services for base features (e.g., GPS, Bluetooth) nothing happens and the following message is always returned: "START_SERVICE command NOT EXECUTED. Service not available".
- **Send broadcast**: It is supposed to send an Android broadcast message [364] but it doesn't look like it works. The dashboard's command interface requires the user to fill a text field named "Action". However, whenever an `android.intent.action` name is inserted, an exception like the following one (in this case, referring to `SCREEN_OFF` action) is returned:

  `SEND_BROADCAST java.lang.SecurityException: Permission Denial: not allowed to send broadcast android.intent.action.SCREEN_OFF from pid=2161, uid=10185`

  If something that doesn't start with `android.intent.action` is inserted in the action field, the dashboard says the command has been executed. However, no effect can be seen on screen. This happens even when inserting random characters in the action field. So, it's more likely the command doesn't work at all, as the application doesn't have the necessary permissions to send such broadcast messages (and unexisting actions can't have any effect).

Other features, not displayed on the dashboard but available from within the app, are:

- All features which are available in the Play Store version of the app.
- **SIM checker**: Google removed the possibility for normal apps to read the SIM card details in Android 10. This feature is available only if you root the device and install Cerberus as a system app.
- **AutoTask configuration**: Allows to define a set of rules which trigger specific events. Each rule has the following fields:
  - Rule name;
  - Event:
    - Charger connected/disconnected;
    - Wrong unlock code with N failed attempts;
    - Device switched on/off;
    - Unauthorized SIM card (not supported starting from Android 10);
    - Cerberus removed from Device Admin;
    - Low battery;
    - Bluetooth device connected/disconnected (any device or specific ones);
    - WiFi network connected/disconnected (requires Network SSID);
    - Timer (hours);
    - Geofence;
    - NFC (requires to scan an NFC tag);
    - Airplain mode enabled/disabled;
    - Device moving;
    - Shutdown attempt while device locked;
    - Forced unlock;
  - Conditions (one or more):
    - Bettery level less/greater than a specific prcentage;
    - Charging state (charging/not charging);
    - WiFi network in range/not in range (requires network SSID);
    - Screen state (on/off);
    - Time, from hh_1:mm_1 to hh_2:mm_2;
    - Day of week (one or multiple days can be selected);
    - Location inside/outside a specific zone;
    - Lock screen active/not active;
    - Emergency mode active/not active;
    - Bluetooth device connected/disconnected;
  - Actions: One or more of the actions which are also available from the website's dashboard, plus some extra ones (i.e., prevent USB debugging, block power menu).

The third party app understands SMS commands. Each command has the following format:

```
cerberus password CommandName
```

where *password* is the account's password.

Here's the full list of SMS commands (✓=yes, ∂=partially, ✗=no):

Command Name	Description	Does it work?
`find`	locate device	∂
`siminfo`	get phone number and operator	✓
`lock code`	lock device with a password (replace *code* with the desired lock code)	✗
`unlock`	unlock device	✗
`alarm text`	display a message and play alarm (replace *text* with the message)	✓
`message text`	display a message (replace *text* with the message)	✓
`speak text`	make the device speak a message (replace *text* with the message)	✓
`call number`	call a phone (replace *number* with the phone number to be called or don't write a number to make Cerberus call the sms sender)	✓
`takepicture`	take a picture and send it to your email address (front camera will be used if available)	✓
`capturevideo`	capture a video and send it to your email address	✓
`screenshot`	grab a screenshot and send it to your email address	✓
`wipe`	wipe your device memory	✓
`wipesd`	wipe the SD card	✓
`startemergency hours`	make the device send its location periodically (replace *hours* with the number of hours from one alert to the next)	✓
`stopemergency`	stop emergency mode	✓
`enabledata`	enable data access on your device	✗
`enablewifi`	enable WiFi and automatically connect to open networks	✓
`disabledata`	disable data access on the device	✓
`disablewifi`	disable WiFi	✓
`enableroaming`	enable data roaming	✗
`enablebluetooth`	enable Bluetooth	✓
`disablebluetooth`	disable Bluetooth	✓
`reboot`	reboot the device (works only on rooted devices)	✗

Table 11.3: Cerberus Antitheft third party app SMS commands list

The `siminfo` command doesn't work if GPS is turned off and the app doesn't hold the permission to automatically turn it on (`WRITE_SECURE_SETTINGS`). The command sends an SMS which is supposed to report the cell tower codes of the target device, to be inserted on the `cellphonetrackers.org/gsm/gsm-tracker.php` web page, but no codes are actually provided. The `lock` and `unlock` commands fail because of the previously explained reasons (starting from Android 7 the PIN code can't be replaced by apps). The remaining commands that fail require a rooted device to (presumably) work. All other commands work as described.

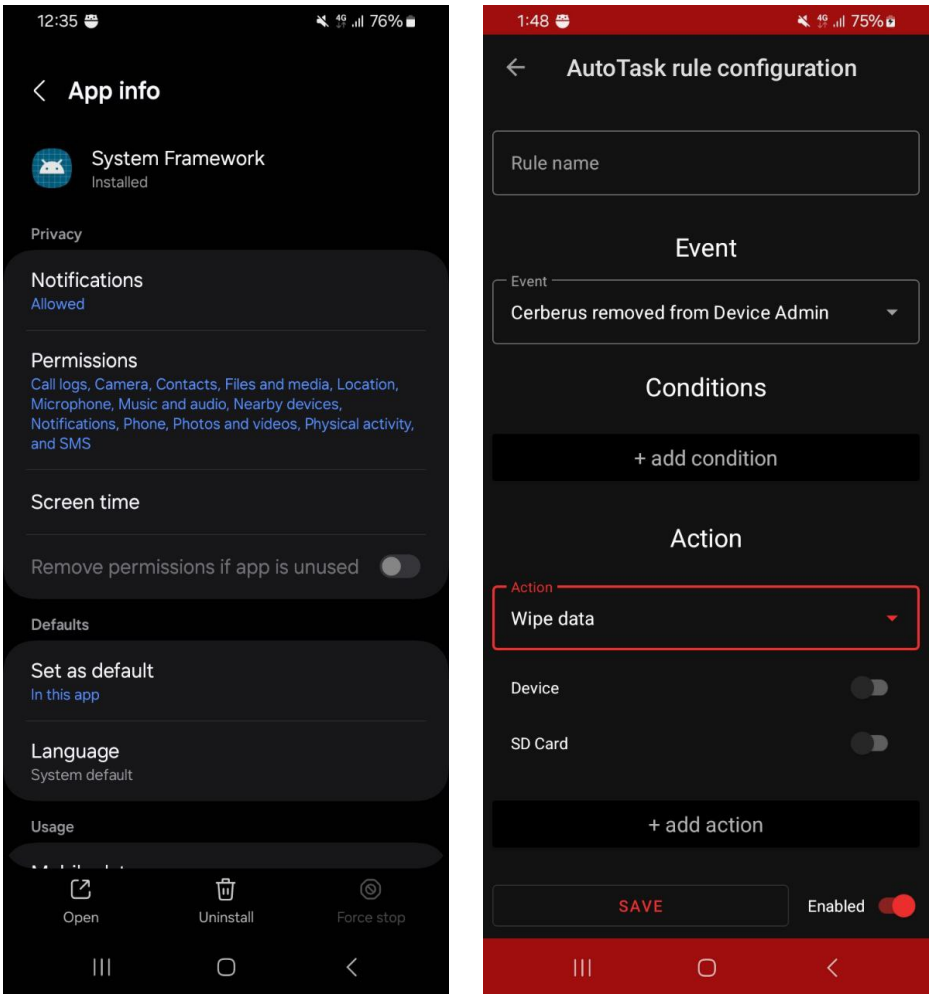
11.2.4 Stealth and persistence

Play Store version of Cerberus Antitheft:

- **Stealth**: It has no stealth at all, as the app icon is displayed in the applications list. Moreover, the app is visible in both recent apps and background apps lists. Whenever the app accesses the microphone and the camera, the corresponding green icons are displayed on the top right corner of the screen. Whenever the device gets unlocked, a notification is displayed which says "This device is protected with Cerberus".
- **Persistence**: The app cannot be uninstalled as any other regular app but it requires the user to revoke its device admin privileges. Doesn't survive factory reset but keeps working after reboot without the need to unlock the device.

Third party version of Cerberus Antitheft:

- **Stealth**: It camouflages as a system app called "System Framework". However, it exhibits the same flaws of the Play Store version. There's a function to hide the app from the app drawer (i.e., apps list) but it (presumably) works only if the device is rooted.
- **Persistence**: Same as for the Play Store version.



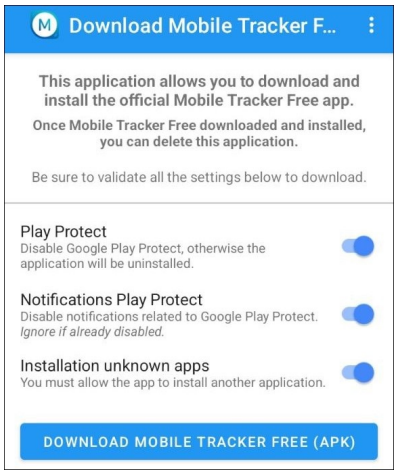
(a) Application camouflage (b) AutoTask rule configuration

Figure 11.3: Cerberus Antitheft 3rd party app version

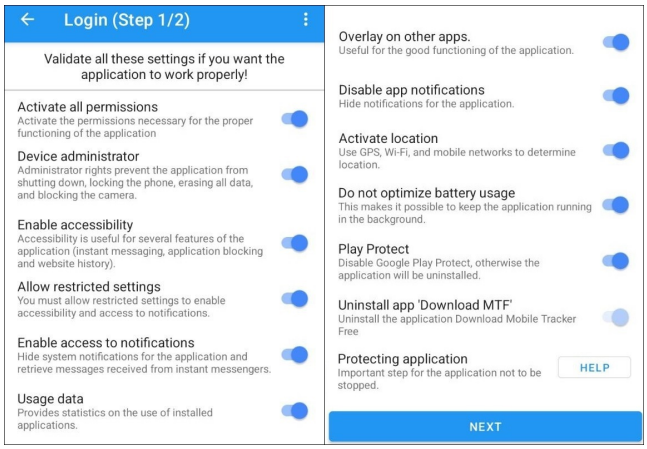
11.3 MobileTracker free

11.3.1 Installation and setup

In order to install MobileTracker free, the user must download and install the "Download Mobile Tracker free app" first. This app serves as a downloader and installer of the real spy app. As a matter of fact, it only asks for the following permissions: `INTERNET`, `ACCESS_NETWORK_STATE`, `REQUEST_INSTALL_PACKAGES`, `REQUEST_DELETE_PACKAGES`, `WRITE_EXTERNAL_STORAGE`, `READ_EXTERNAL_STORAGE`. However, PlayProtect still recognises it as malicious (so it has to be turned off). After installing the downloader app and enabling a few options (Figure 11.4a), the real app is downloaded and installed. The app requires to turn off all the security measures of the target device and it requires a huge set of permissions (Figure 11.4b)



(a) Download Mobile Tracker free



(b) Permissions setup

Accessibility	✔
Access to notifications	✔
Location	✔
Disable app notifications	✔
Device administrator	✔
Display over other apps	✔
Don't optimize battery usage	✔
Permissions	CALENDAR ✔ CAMERA ✔ CONTACTS ✔ LOCATION ✖ MICROPHONE ✔ PHONE ✔ SMS ✔ STORAGE ✔
Root	✖

(c) Configuration overview

Figure 11.4: MobileTracker free - installation and setup

11.3.2 Application's dashboard and commands testing

The app doesn't store the data (e.g., SMS, emails, calls, etc...) which was already in the phone before the installation. The following features of MobileTracker free have been tested.

SMS monitoring

Data: Access to the last 15 SMS and SMS older than 3 days are deleted. Each entry in the list shows type (i.e., outgoing, incoming), contact name (if any), SMS body, phone number, date, time and geolocation of the device with the corresponding address;

SMS Alert: By installing "MobileTracker free client" (from the app website) on the account owner's device, it is possible to trigger a notification whenever the target device receives an SMS which contains a specific keyword. This feature was tested by installing the client on the target device instead (Figure 11.5).

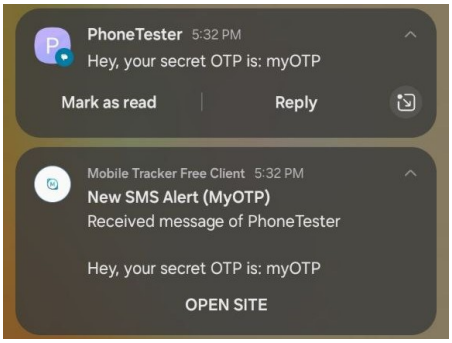


Figure 11.5: MobileTracker free - SMS alert (keyword: myOTP)

Calls monitoring

Data: Access to the last 15 calls and calls older than 3 days are deleted. Each entry of the list allows to play back/download the call and provides information regarding the call itself (Figure 11.6). However, the audio quality is really poor and it's difficult to hear the victim's voice. Only 5 listenings/downloads per day are available with the free version of the app. From the account settings it is possible to change audio source for recording the calls (e.g., Voice UpLink, Voice DownLink, MIC, Voice Recognition, etc...) but there's no configuration which allows to hear the call clearly (most of them don't even record any audio at all). The only configuration that records something (despite the poor quality) is the one which uses Media Recorder + MIC.

Block a number: When the user tries to call the blocked number, the app immediately closes the call without letting the recipient's phone ring. This feature doesn't work for incoming calls.

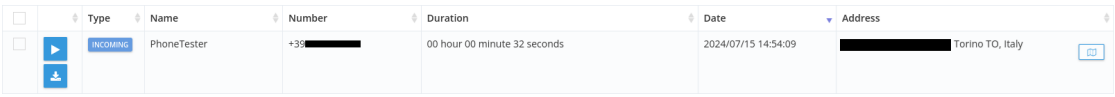


Figure 11.6: MobileTracker free - call logs entry

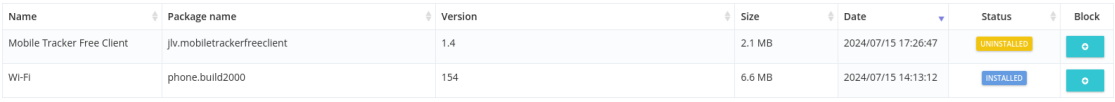
Location monitoring

Data: Access to the last 15 locations. Locations older than 3 days are deleted. Each entry shows the latitude, longitude, accuracy, date, time, and address of the corresponding location.

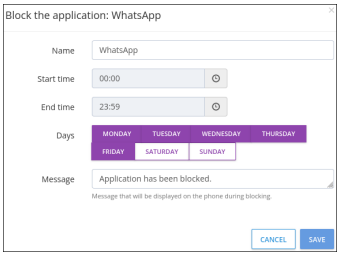
Live: Works only if GPS is turned on. Otherwise, it requires a rooted device.

Apps monitoring

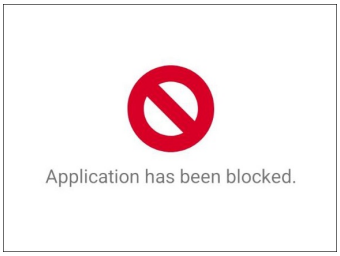
Shows all installed apps but also apps which have been installed and successively uninstalled when MobileTracker was already monitoring the device (Figure 11.7a). Each app has its name, package name, version, size, installation date/time, and status. It is possible to selectively block apps so that the victim cannot use them (Figure 11.7b). If the victim tries to launch a blocked app, MobileTracker takes over the screen (Figure 11.7c) and replaces the currently running application. The blocked app is actually launched but the user has no time to interact with it before the block screen appears. It is also possible to see some stats regarding apps usage (Figure 11.7d).



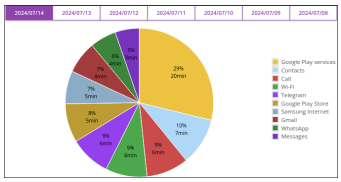
(a) Installed apps list



(b) Blocking WhatsApp



(c) Blocked application screen



(d) Apps usage

Figure 11.7: MobileTracker free - installed apps monitoring

Pictures

Access to the last 15 pictures, screenshots included. Pictures older than 3 days are deleted. Each entry displays the picture itself, along with date, time, file name + size, geolocation of where the picture has been taken (Figure 11.8). Screenshots of the app's blocking screen are named after the app's camouflaged name "Wi-Fi" (it could be a hint for detection of the malicious app).

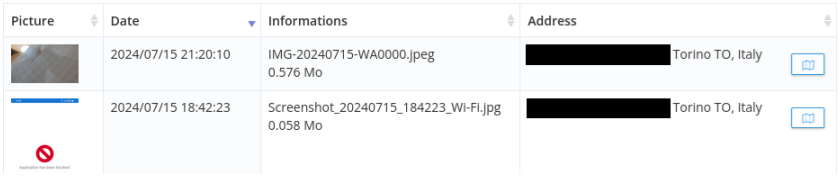


Figure 11.8: MobileTracker free - pictures storage

Calendar events

Access to the last 15 events. Events which have been created more than 3 days ago are deleted. Each entry shows the event title, description, start date, end date, location (if any), and time.

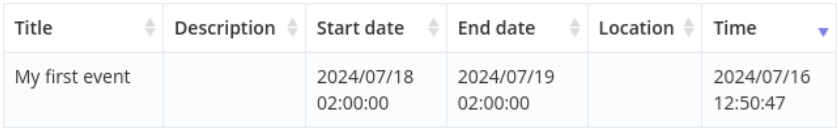


Figure 11.9: MobileTracker free - calendar events monitoring

Contacts list

Displays the contacts list. Each entry shows the contact name, number, and creation date.

Name	Number	Date
PhoneTester	+39[redacted]	2024/07/13 17:19:58

Figure 11.10: MobileTracker free - contacts list monitoring

Web sites monitoring

Historique: Access to the last websites. Websites visited more than 3 days ago are deleted. Each entry of the list shows the URL of the web page, the browser which was used to access it, the date, and the time of the access.

Blocking: Allows to block access to specific URLs. As soon as one of the blocked URLs is accessed, the browser returns to the main Google search page. Blocking `www.google.com` creates an infinite loop where the browser keeps refreshing Google's search page.

URL	Browser	Date
mobile-tracker-free.com	Samsung Internet Browser	2024/07/15 17:27:51
www.virustotal.com	Samsung Internet Browser	2024/07/15 14:34:43
www.google.com	Samsung Internet Browser	2024/07/15 14:32:47

(a) Web sites historique

URL	Date	Block
WWW.GOOGLE.COM	2024/07/15 14:34:58	✓
WWW.VIRUSTOTAL.COM	2024/07/15 14:34:11	✓

(b) Web sites blocking

Figure 11.11: MobileTracker free - web sites monitoring

Screenshot command

Periodically takes a screenshot. If the phone is idle, the screenshot will be taken as soon as it exits the idle state. If the phone is not rooted, an icon will be displayed during capture. Despite the fact that the "screenshots" feature has been enabled in-app, the app never requested such permission. Hence, this feature does not work.

Instant messaging and social media monitoring

WhatsApp: Access to the last 15 messages. Messages older than 3 days are deleted. Each entry of the messages list shows the message type (i.e., incoming, outgoing), contact name (if any), group name (if any), message text, date, and time. However, the app does not distinguish medias (i.e., audio, video, pictures) from text messages. If medias are exchanged, the app will view them as text messages whose content is the same of the last exchanged text message. In Figure 11.12, only the first message is actually a text message whose content is "Hi". All the other messages are medias. Files sent and received via WhatsApp can be fetched with the "file explorer" function of the app.

Type	Name	Message	Date
INCOMING	PhoneTester	Hi	2024/07/15 21:20:13
INCOMING	PhoneTester	Hi	2024/07/15 21:19:54
INCOMING	PhoneTester	Hi	2024/07/15 14:29:23
INCOMING	PhoneTester	Hi	2024/07/15 14:28:46

(a) WhatsApp direct messages

Type	Name	Message	Date
INCOMING	Phone testing group (3 messages): PhoneTester	Welcome! This is the phone testing group!	2024/07/16 12:05:17
INCOMING	Phone testing group (3 messages): Phone testing group	PhoneTester added you	2024/07/16 12:05:17
INCOMING	Phone testing group	2 new messages	2024/07/16 12:05:17
INCOMING	Phone testing group (2 messages): Phone testing group	You were added	2024/07/16 12:05:17

(b) WhatsApp group messages

Figure 11.12: MobileTracker free - WhatsApp monitoring

Telegram: Same as WhatsApp except that medias exchange goes unnoticed, i.e., there's no corresponding entry in the messages list (not even multiple copies of the last exchanged text message like in WhatsApp) and group chats are not as detailed (i.e., no group name nor message sender's name).

Gmail: Access to the last 15 emails. Emails older than 3 days are deleted. Each entry of the mail list shows the email type (i.e., incoming, outgoing), sender/recipient name, email body, date, and time. Attachments are not intercepted by the app.

YouTube: Each entry represents an info regarding the interactions between the user and the YouTube app. The "info" field can either be the title of a video the user watched or an action performed by the user. Each entry, as always, is associated to date and time (Figure 11.13).

Info	Date
Application icon	2024/07/15 21:29:28
Eminem - Till I Collapse [HD]	2024/07/15 21:30:10
Skip	2024/07/15 21:30:10
Close miniplayer	2024/07/15 21:30:11
Close miniplayer	2024/07/15 21:30:22
Eminem - The Real Slim Shady (Official Video - Clean Version)	2024/07/15 21:30:22

Figure 11.13: MobileTracker free - YouTube app monitoring

Live viewing

Only 5 live sessions of 1 minute each are available per day. When a session is started, camera and microphone of the phone are accessed in real-time.

File explorer

Maximum 5 files downloaded per day. Maximum 10 MB per file. Allows to browse the device's file system and to download files from it.

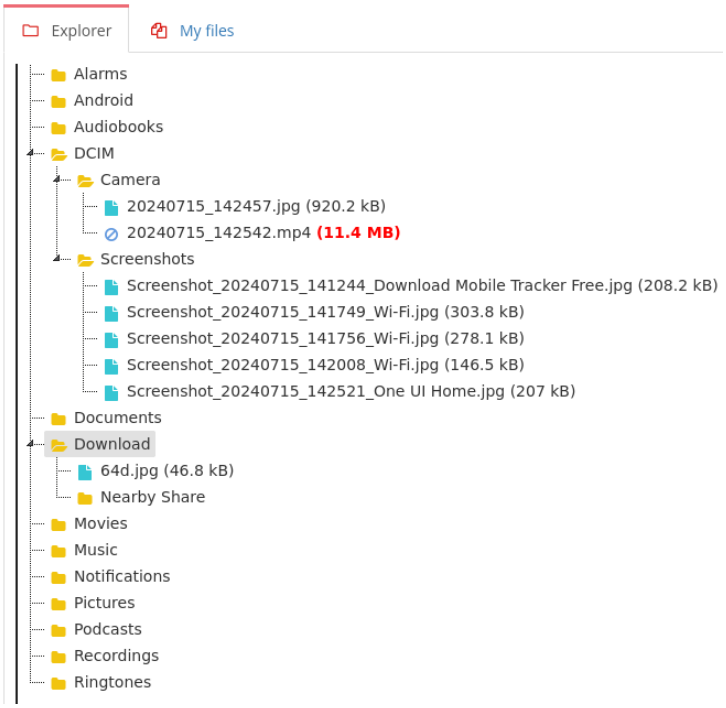


Figure 11.14: MobileTracker free - file explorer

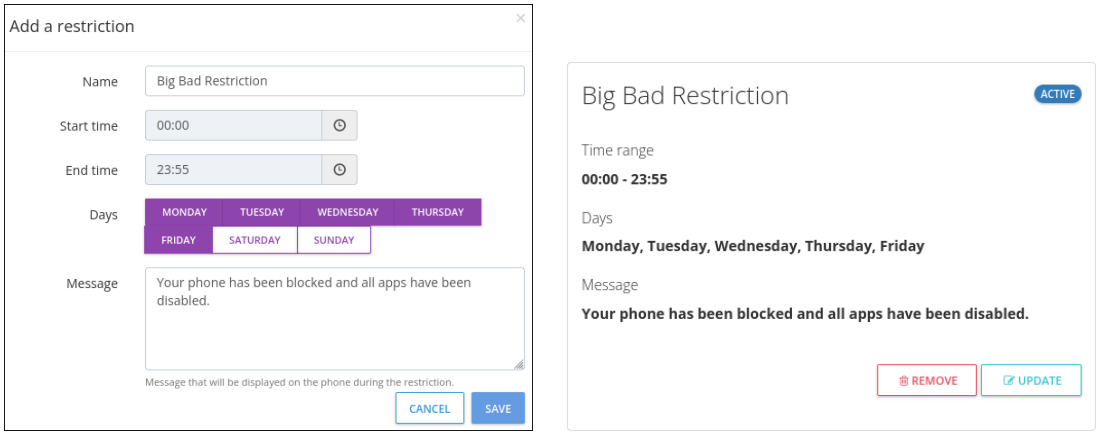
This feature has some filtering options which allow to immediately retrieve WhatsApp medias (i.e., pictures, audio, voice notes, videos, documents) from the storage. Files are retrieved from the following directories:

- `/storage/emulated/0/Android/media/com.whatsapp/WhatsApp/Media`
  - `/WhatsApp Images`
  - `/WhatsApp Videos`
  - `/WhatsApp Audio`
  - `/WhatsApp Voice Notes`
  - `/WhatsApp Documents`

These directories can also be accessed when connecting the device to a computer, with a USB cable, to navigate its file system.

Schedule restriction

Prevents the victim from launching any app during the selected week days and time windows.

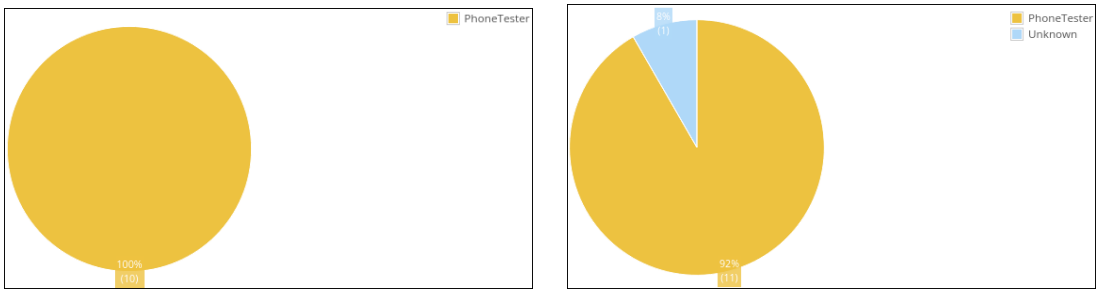


(a) Restriction creation (b) Restriction dispatching

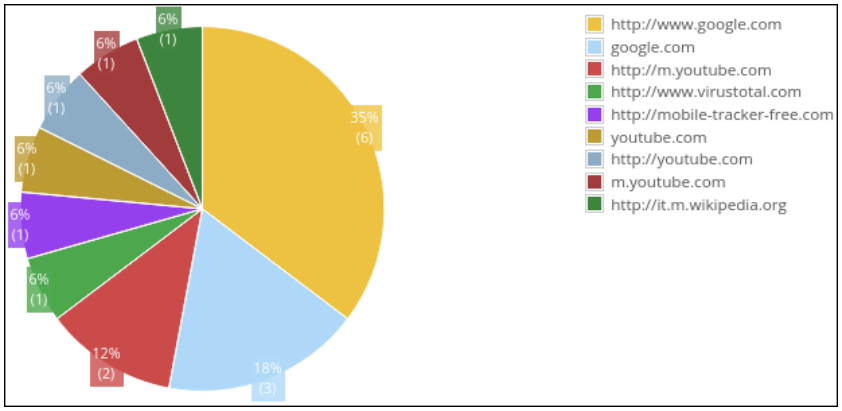
Figure 11.15: MobileTracker free - restriction scheduling

Statistics

Displays statistics regarding which contacts the victim has exchanged SMS with (Figure 11.16a) and most called contacts (Figure 11.16b), along with most visited websites (Figure 11.16c) and most used apps (Figure 11.7d).



(a) SMS stats (b) Calls stats



(c) Web sites stats

Figure 11.16: MobileTracker free - statistics

Remote control

Only 10 remote commands per day can be executed.

Non-root commands:

- **Get location**: Returns the current geolocation of the device, if GPS is active;
- **Recover data**: Recovers up to 30 SMS messages or call logs whose creation happened before MobileTracker was installed;
- **Take a picture**: Either from the front or the back camera. Flash doesn't work;
- **Record audio**: Records either 1 or 2 minutes of audio as an MP3, AMR or WAV file;
- **Vibrate**: Makes the phone vibrate for the specified amount of seconds;
- **Send a message to the phone**: Either a text to be displayed on screen or a voice message to be played by the device's speaker;
- **Get contacts list**: Returns contacts list;
- **Send SMS**: Sends an SMS with custom text to the specified phone number;
- **Enable/Disable camera**: This command doesn't work;
- **Take screenshot**: A message is displayed on screen to ask the user for permission to take screenshots on her phone (Figure 11.17). The message displays the camouflaged app's name "Wi-Fi". After granting the permission, the command has to be sent a second time in order to actually take a screenshot;
- **Get SIM info**: It doesn't return IMEI nor IMSI. It only returns the phone number, SIM software version and mobile operator. It also returns information regarding the device's software, i.e., Kernerl version, Android API level, and device model;
- **Delete all phone data**: This feature hasn't been tested;
- **Hide/Unhide app icon**: This command doesn't really hide the app icon but it simply disguises it as a fake Wi-Fi app icon. Further explanation in Section 11.3.3.

Figure 11.17: MobileTracker free - permission for taking screenshots

These commands have not been tested as they require a rooted device:

- **Enable/Disable GPS**;
- **Enable/Disable mobile data**;
- **Recover data**: Deleted social media messages, emails, browser history;
- **Social media calls history**: WhatsApp, Skype, Facebook, and Viber;
- **Restart the phone**;
- **Take screenshot**;

SMS commands

The app supports SMS commands. Each command has the following syntax:

`PassPhrase--CommandName`

where `PassPhrase` is a secret string of characters which can be changed by the user from within the app. The default string is `MobileTrackerFreeSMS`. These are the available commands:

Command name	Does it work?
`ring--nSeconds`	✓
`getlocation`	✓
`datamobile`	✗
`nodatamobile`	✗
`gps`	✗
`restart`	✗

Table 11.4: MobileTracker free - SMS commands

All commands which have been marked with ✗ require a rooted device and have not been tested.

11.3.3 Stealth and persistence

Stealth

- By default, the app disguises itself as a Wi-Fi application (Figure 11.18). Despite it being shown in the apps list (unless the device is rooted), whenever its icon is clicked the real Android Wi-Fi management screen appears. The real app can be launched only by writing a specific PIN on the Keypad and hitting the call button. The PIN can be set from within the app and it must be enclosed within `*` (i.e., `*NumericDigits*`). Default pin is `*1234*`.
- The real MobileTracker icon can be displayed after launching the "unhide icon" remote command (Section 11.3.2). When the real app icon is clicked, the user is not redirected to the android Wi-Fi management screen anymore. App's name changes as well. However, in the background apps list, MobileTracker is still displayed as the fake "Wi-Fi" application.
- Whenever the app performs an action (e.g., blocking other apps) which causes it to display something on screen (e.g., messages, text boxes, etc...), it then removes itself from the recent apps list whenever the user tries to look at the list itself. Here's the execution flow which brings to this scenario:
  1. MobileTracker displays something on screen;
  2. The user goes to the recent apps list;
  3. MobileTracker closes the recent apps list;
  4. Once the user re-opens the recent apps list, MobileTracker is gone.
- Whenever the app accesses the microphone and the camera, the corresponding green icons are shown on the top right corner of the display.
- Received SMS commands are not deleted from the target device.

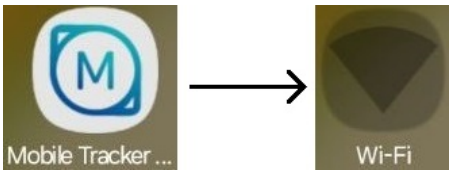


Figure 11.18: MobileTracker free - icon camouflage

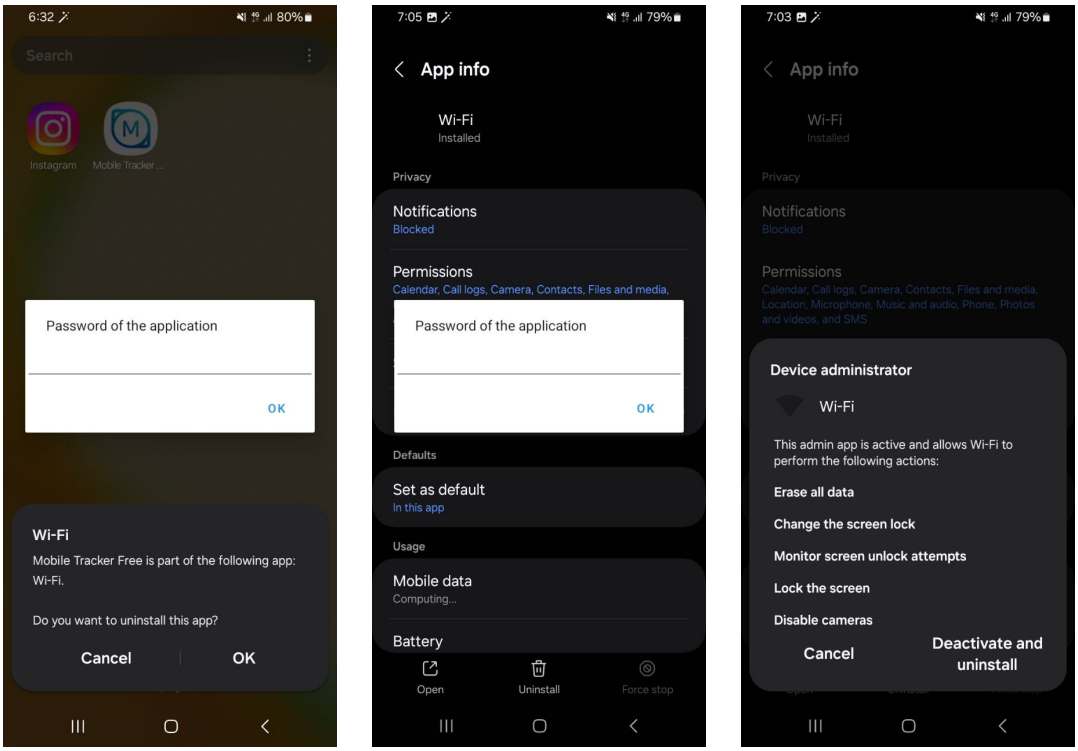
Persistence

- If the phone is either restarted or turned on after being turned off, MobileTracker doesn't work until the phone gets unlocked for the first time after reboot.
- If, instead of simply clicking on the app icon (either camouflaged or not), the user clicks and holds her finger on the icon for a brief second (as if she wanted to move the icon elsewhere, to read the app info, or to uninstall the app), a text box which asks for the app secret PIN appears. Whenever a wrong PIN is inserted, the text box is re-displayed. When the right PIN is inserted, the text box disappears. If the text box is on screen, it is impossible to click any button (e.g., uninstall, info, etc...) other than the confirmation button of the text box itself for checking the PIN.

By rapidly clicking on the text box "OK" button and on the uninstall button, it is possible to display the confirmation message for uninstalling the app (Figure 11.19a). However, the text box immediately re-appears and takes over the screen, making it impossible to click on the confirmation button for uninstalling the app. The only thing the user can do is to click on the text box "OK" button but, by doing so, the confirmation message for uninstalling the app will disappear. As long as the user doesn't know the secret PIN, it is impossible to uninstall the app from the apps drawer.

- The PIN text box appears even when trying to uninstall the app from its settings screen (Figure 11.19b). By clicking on the "OK" button, application settings are closed and the user is brought back to the home screen. However, device settings are not removed from the recent apps list! By going back to the device settings through the recent apps list, all the device settings are displayed (not the application settings for the fake "Wi-Fi" app). Starting from this point, clicking again on the "apps" button and going to the fake "Wi-Fi" application settings would make us fall again in the initial scenario (Figure 11.19b). If, instead of doing so, the user clicks twice on the back arrow button of the phone, she is brought back to the fake "Wi-Fi" application settings screen but this time the PIN text box is not displayed! It is then possible to uninstall the application (Figure 11.19c).

  Uninstallation methods suggested by the application website (Section 10.1.4) work as well.



(a) Apps drawer (b) Settings screen (c) Successful uninstallation

Figure 11.19: MobileTracker free - uninstallation scenarios

11.4 Comparing the tested tools

Based on our tests, we can make some final considerations regarding these spy apps:

- Hammer Security and the Play Store version of Cerberus Antitheft are the **easiest to install**, as they're downloadable from the official Google store and do not require to turn off PlayProtect, as well as they don't require to enable the "Allow installations from unknown source" Android feature;
- In terms of **availability**, MobileTracker free is the best one because its free plan has no expiration date but it only limits the number of actions the attacker can perform each day. On the other hand, Hammer Security and Cerberus free plans have a limited duration which can only be extended by watching ads from within the respective apps, i.e., from the victim's device. The remaining tools, which haven't been tested, do not provide any free plan (except Hoverwatch which, however, has a very limited set of features);
- In terms of **functionality**, Cerberus and MobileTracker free are the best ones. However, some features, like taking screenshots, are better implemented in Cerberus and do not require a rooted device. On the other hand, MobileTracker free exhibits amazing capabilities when it comes to monitoring WhatsApp and explore the target device's file system (such features are not offered by Cerberus);
- In terms of **stealth**, there's no app which successfully hides from the applications drawer and the background apps list. All apps provide icon camouflage but MobileTracker free can also remove itself from the recents screen. Cerberus isn't really stealthy because of the constantly displayed notification "This device is protected by Cerberus";
- In terms of **persistence**, Cerberus is the only one which can still work properly after the device gets rebooted, while the others require the user to unlock the device with the password for the first time after reboot (otherwise, they won't work until the device is unlocked). However, MobileTracker free is the only one which tries to prevent the user from uninstalling it, even though this protection mechanism is not 100% effective;

Chapter 12

Testing spy apps on iOS 17.6.1

All tests have been performed on an iPhone SE 2nd generation (2020) with iOS 17.6.1.

As anticipated in Section 10.2, there are three categories of spy apps for iOS:

- Apps that require the target device to be jailbroken;
- Apps that don't require a jailbreak but need iCloud credentials instead;
- Apps that don't require any of the previous things but have a very restricted set of features;

Since jailbreak cannot be performed on newer iOS versions (including 17.6.1, which is the latest version of iOS currently available) the first category of apps had to be excluded. The second category was excluded too, because the use of iCloud would imply no app to be installed on the target device and this scenario is not meaningful for our analysis, as the tools used in Chapter 14 can only detect malicious applications which have been installed on the device. Hence, applications available for free on the App Store have been selected for testing.

The testing sections below are structured like the ones for Android except for the "Stealth and Persistence" section which has not been included. None of the tested apps presents intrinsic features which grant it such properties:

- MMGuardian icon is always visible, unless the remote command for blocking all apps is executed. In that case, the icons of all the downloaded apps (including MMGuardian) would be hidden from the App Library. However, this would immediately arise suspicions. Moreover, every remote command sent from the dashboard, including this one, displays a notification on the target device because of the Mobile Device Management (MDM) profile which has been created when configuring the app.
- The inability to uninstall an app depends exclusively from the change of the device's settings and not from an intrinsic feature of the app.

12.1 MMGuardian

12.1.1 Installation and setup

MMGuardian is available on the App Store [312] and offers a 14 days long free trial. In order to download the app, the Apple ID account's password is required. After downloading and installing the app, the configuration phase consists of changing the "Screen Time" settings [365]:

- Permanent use of location;
- Don't allow deleting apps;

A "Screen Time secret PIN" will be created and it will be requested in the future for modifying Screen Time settings. Without knowing the Screen Time PIN, it's not possible to uninstall the app. To change Screen Time settings in the first place, the device's secret PIN is required.

MMGuardian also asks to define a Mobile Device Management (MDM) profile to connect the iPhone to its MDM server located at `family.mmguardian.com/apple/server`. This is the only way for the app to manage the device remotely.

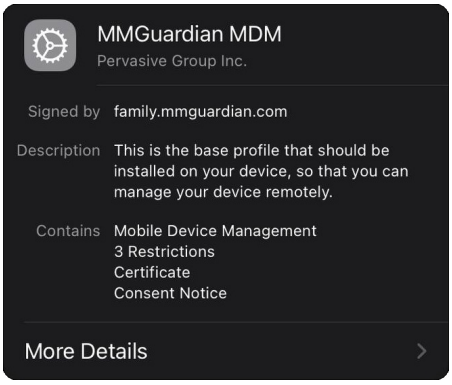


Figure 12.1: MMGuardian MDM profile

Moreover, in order to access SMS and call logs, the MMGuardian Sync App must be installed on a computer. This app periodically performs a backup of the iPhone through Wi-Fi (every 15 minutes, minimum). In other words, the only way to retrieve SMS and call logs is to be connected to the same network of the target device.

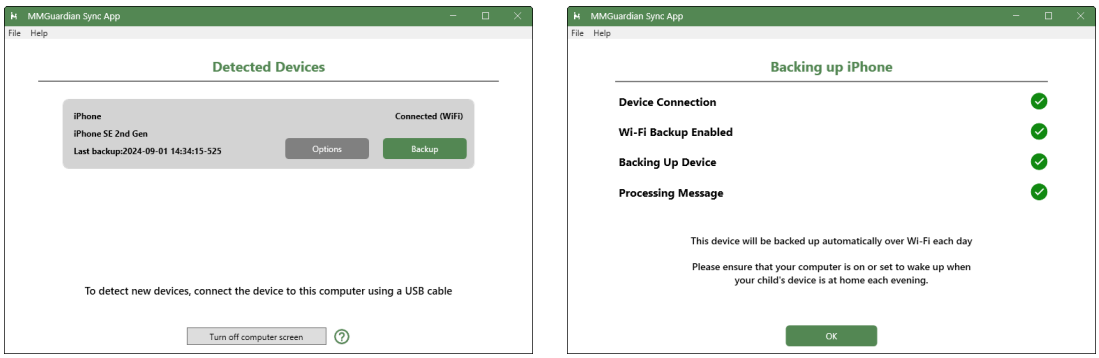


Figure 12.2: MMGuardian Sync App device registration and backup

12.1.2 Application dashboard and features

MMGuardian's dashboard provides some really basic features:

- **Device usage**: Displays statistics regarding data collected through the Wi-Fi backup, i.e., SMS messages, call logs, and web browsing. SMS messages content can only be seen from the Sync App;
- **Location map**: Retrieves current geolocation of the target device;
- **Quick app control**: Disables/Enables all downloaded apps for a fixed amount of time;
- **Device functions**: Disables/Enables Siri, camera, voice dialing, and/or screen capture;
  - Note: The aforementioned features are not being actively used by the app, they're just disabled/enabled for the victim;
- **Device lost/stolen**: Allows to lock the device with its PIN;
- **Settings**: Among some customization settings, there's a command for calling the target device from the MDM server;
- All the remaining features are mainly filtering options to block inappropriate content on the Internet and they are far from being considered possible features of a spyware.

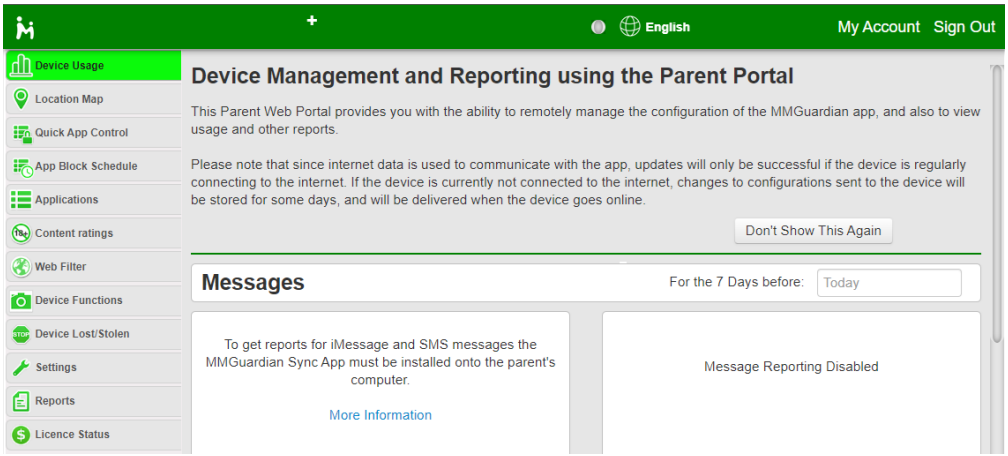


Figure 12.3: MMGuardian's dashboard

Name/Number		Total	Sent	Received:
KENA MOBILE	iMessage	1	0	1
Phone Tester	iMessage	1	1	0
Name/Number		**Calls**		**Total Time**
Phone Tester_+393716838653		1		0.1 mins
Website		**Page Views**		**Duration**
family.mmguardian.com		1		0 mins
www.mmguardian.com		1		0 mins
www.google.com		6		0 mins
support.apple.com		4		0 mins

Figure 12.4: MMGuardian stats

12.2 Qustodio

12.2.1 Installation and setup

Qustodio is available on the App Store [316] and offers a 3 days long free trial. The app installation and configuration are very similar to the ones of MMGuardian:

- The app creates an MDM profile for the device.
- The app requires installation of a third party computer application for monitoring SMS and call logs [366]. However, after connecting the iPhone to the computer and completing the registration procedure of the "Calls and Messages" app, those information do not get periodically updated despite both devices being connected to the same network. This seems unintended, as Qustodio automatically activates access to its private VPN on the target device, so that updates should be received even without having access to the target's Wi-Fi network. Hence, due to some implementation flaw, SMS and call logs monitoring is not really possible.
- The app requires installation of another third party computer application [367] for activating anti-tampering protection which prevents anyone from uninstalling Qustodio. After connecting the iPhone to the computer and completing the procedure, the iPhone automatically reboots and, from that moment, it is impossible to uninstall Qustodio, unless by undoing the procedure from the "Qustodio Advance" computer app.

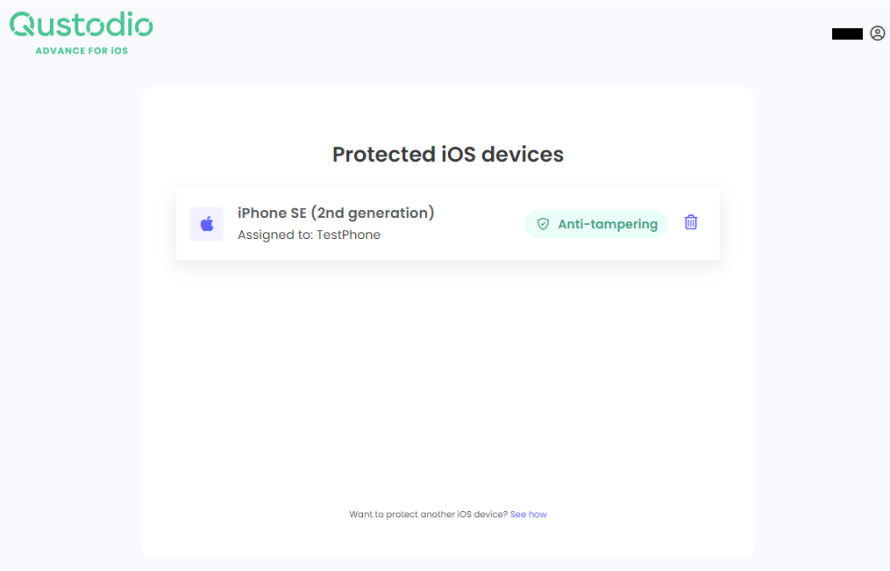


Figure 12.5: Qustodio Advance anti-tampering

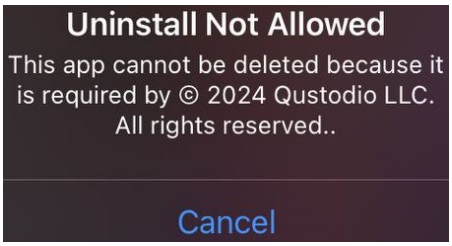


Figure 12.6: Qustodio uninstallation attempt with anti-tampering setup

12.2.2 Application dashboard and features

The features provided by Qustodio are, more or less, the same of MMGuardian:

- **Screen time**: Statistics regarding screen time.
- **Games & Apps**: Allows, blocks, or sets time limits to applications usage.
- **Location tracking**: Real time location tracking. Location only gets updated if the target significantly moves away from its last position.
- **Calls & Messages**: Call logs and SMS monitoring. This feature doesn't work properly as explained in the previous paragraph.
- **YouTube**: YouTube history monitoring. This feature always gets updated in real time and provides links to watched videos.

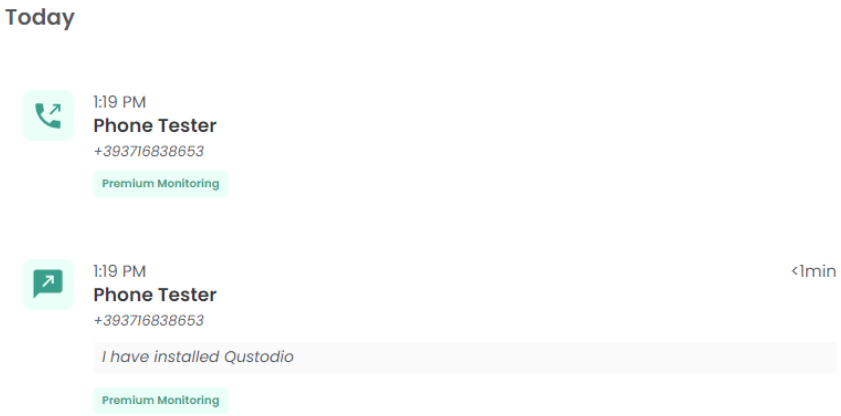


Figure 12.7: Qustodio call logs and SMS messages

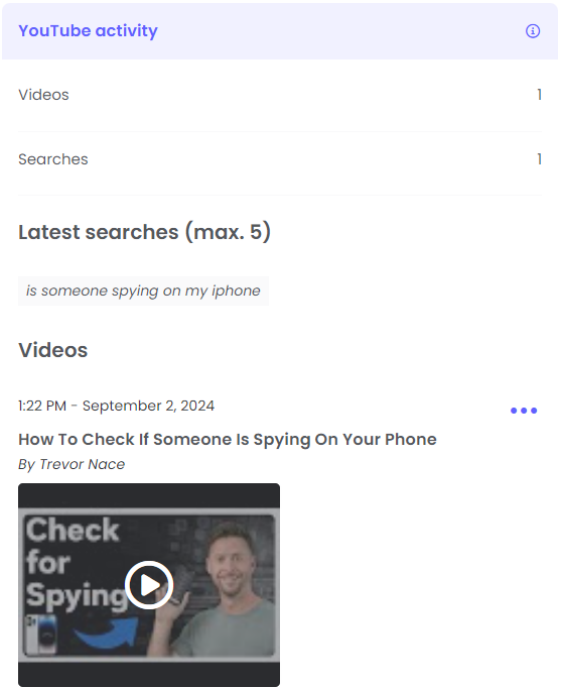


Figure 12.8: Qustodio YouTube monitoring

12.3 Kidslox

12.3.1 Installation and setup

Kidslox is available on the App Store [368] and offers a 3 days long free trial. Installation and configuration procedures are similar to the ones of the previous iOS apps:

- The app creates an MDM profile and connects the device to its VPN;
- A third party computer application can be installed for adding anti-tampering protection and block uninstallation attempts;

12.3.2 Application dashboard and features

Even in this case, available features are more or less the same ones provided by the previous apps. The ones which could be used for "spying" on someone are:

- **Web History**: Logs all visited websites and their corresponding URLs;
- **Web Searches**: Logs whatever the victim has searched with the browser;
- **YouTube**: Saves YouTube history and watched videos links;
- **Telescope**: Periodically takes a screenshot of the target device. This feature requires the user to explicitly enable screen broadcasting, which doesn't go unnoticed at all, and doesn't really work automatically but only when an explicit screenshot request is sent with the dashboard;

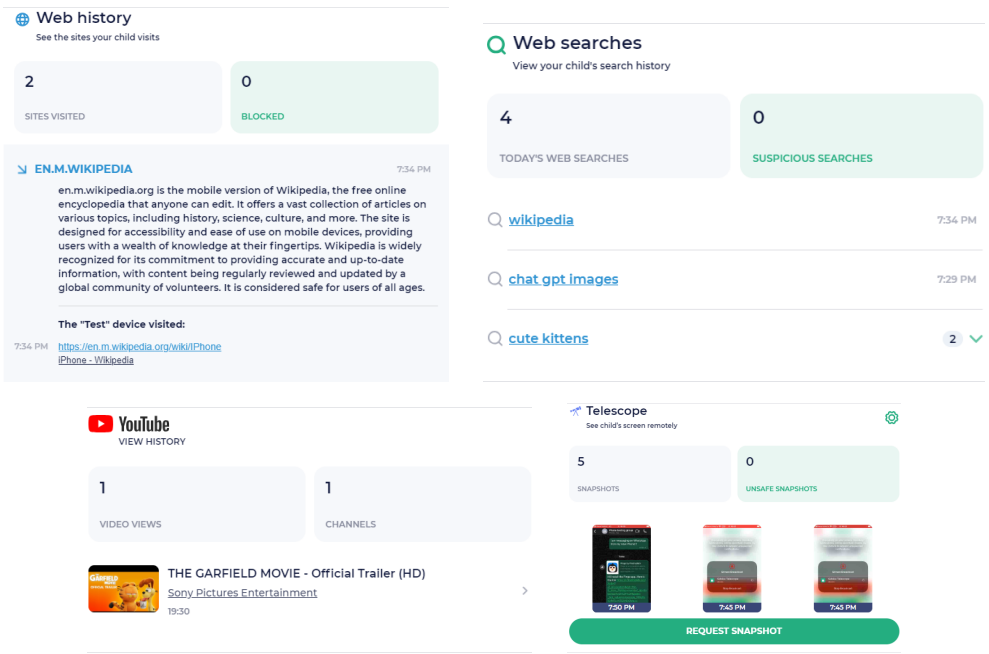


Figure 12.9: Kidslox main features

Chapter 13

Installation attack on Android

The goal of this chapter is trying to design an attack which could lead to successful installation and execution of a spyware on a target mobile device. After studying and testing spy apps (Chapters 10 and 11), we've learnt how crucial it is for applications of this kind to get all the necessary permissions. However, we've also seen how doing so is not always trivial as most critical permissions, in order to be granted, require the user's consent.

We're going to describe our attack as the composition of two phases:

1. **Installation** of the malicious app;
2. **Configuration** of the app (i.e., permissions, etc...);

We have to consider that the design of the attack may vary based on what kind of users the attacker means to target and why:

- Employees of a company, or even federal agencies, for stealing confidential data (e.g., private documents) and re-sell them? [369, 370]
- A single person (e.g., a VIP, someone the attacker knows directly, etc...), maybe to follow her movements [371, 372], to acquire private/sensible data and ask something in return (e.g., a ransom) by threatening to disclose them [373], or even to get revenge?
- Random people, to steal banking information? [374]

Clearly, we cannot put ourselves into really specific scenarios. As a matter of fact, scenarios where the target are not just some random people will probably require the attacker to gather some prior knowledge about the victims and to have good social engineering skills.

Due to all these factors, we will try to define an attack model which could have success in the most general scenario possible.

13.1 Installation phase

13.1.1 Possible scenarios and attack premises

Let's try to reason about how a malicious application could be installed on a device. Here we present some possible scenarios:

1. The attacker has **physical access** to the victim's device.

   This is more of an isolated case, as it's not that frequent for an attacker to gain physical access to a target device, but it's still something which is possible and which could easily lead to the installation of the malicious app on the device.

2. The application is installed by the victim **willingly**.

   The user may decide to install the app on her own because she finds it interesting. Such application must look legit but it must contain, sooner or later, some malicious code in order to behave differently from what the user is expecting. Clearly, if the application is available on the official stores and has a good amount of positive reviews, it will be easier to fool the victim(s). It is unlikely that users install applications from third party stores on their own, unless they're forced or tricked into doing so [375, 376].

   So, another question arises: How can the user develop interest towards the application?

   If you want to convince someone into installing something, the most common (yet effective) way to do so is by making good publicity for it (i.e., ads). However, for an attacker, this could require a considerable amount of time and money. Supposing the attacker decides to develop the application from scratch and to successfully publish it on an official store, the app should become popular enough for it to be installed by as many users as possible.

3. The application is installed on the victim's device without her knowing (**unwillingly**) and without having physical access to the device itself.

   This scenario often requires the attacker to exploit some vulnerability in the target system to install and execute the malicious app [377, 378]. However, it is not that trivial.

Since the first scenario is too accommodating, while the third one is too complex, we will focus on the second scenario. Trying to convince people to install the app from an unofficial store may be ineffective for the following reasons:

- The attack vector, i.e. an e-mail or an SMS message containing a malicious link, would probably arise suspicions in the user;
- The user should be convinced into enabling the "Allow installations from unknown source" option, which is not really something to fall for;
- Hosting the malicious app on a third party store doesn't really solve the detection problem, as tools like PlayProtect scan apps even on the device and not just on the official store, unless the user is convinced to disable such security measures (which is unlikely);

So, our attack design will involve the publication of the malicious app on the official stores.

13.1.2 Choosing the attack vector

We're not considering ads as an option, because the process is slow and costly [379,380]. Hence, the easiest way for trying to promote the app would be starting e-mail and/or SMS campaigns. Each e-mail/SMS would contain a legitimate link to the app inside the Play Store. An additional link to a YouTube video could be added for showing a preview of the application. The main goal here is to create interest in the end user. Each e-mail/SMS will have to contain also some text which tries to promote the app and invites the user to try it and, maybe, even get some rewards in return. Rewards could be, for example, free access to exclusive features of the app. Keep in mind that the app itself must be interesting. By looking at the most downloaded app categories from the Play Store in 2022 [381], we can see that Gaming, Education, and Business are at the top of the list.

13.1.3 PlayProtect detection evasion

How can the malicious app go undetected by PlayProtect?

The app must look legit at first and it must get enough good reviews for it to be trusted and to become popular. Then, when the desired number of downloads is reached, an update can be pushed to add the malicious code for the spyware features. This technique is called **IMUTA** (**Incremental Malicious Updates Attack**) and it can be combined with other techniques such as dynamic code loading, code obfuscation, and encryption [62]. A good example of detection evasion is presented by [63], where the malicious code is obfuscated, encrypted, and embedded into an image by means of steganography. The malicious version of the app should be tested against tools like VirusTotal [382] and, most importantly, PlayProtect itself. Testing against PlayProtect can be done by installing the application on a test device with (possibly) the latest Android version, as we've done in Chapter 11. The app should be installed on the test device with the Android Debug Bridge tool [362,383] as it can't be downloaded from the Internet yet. Clearly, the more testing is performed against anti-malware tools, the better it is.

13.2 Configuration phase

If the installation phase has been successful, there's still one final layer of security which should be bypassed: permissions. The only way for the app to gain the necessary permissions to start behaving like a spyware, without considering any vulnerability exploit of the target system, is by having the user granting such permissions to the app. There is no other way. We have seen examples like Hammer Security (Section 11.1) and Cerberus Antitheft (Section 11.2), which are downloadable from the Play Store and are verified by PlayProtect, but still ask for critical permissions like accessibility service and device admin (along with many others). That proves that it is possible to publish similar applications on the Play Store. However, if the app asks for more permissions than the ones which would be necessary for fulfilling its purpose, the user could get suspicious and uninstall the app. One way to try mitigating this, could be by using the accessibility service for self-granting all the other permissions, like some malwares do [171]. However, it must be the user to grant the accessibility service permission to the app in the first place. Hence, the app should try to justify to the user why it needs to be granted with such permission. If the user falls for this trap, the attack finally succeeds.

Chapter 14

Tools for spyware detection

14.1 SpyGuard

SpyGuard [384] is a Linux tool that monitors the network flows transmitted by a device to detect signs of compromise. SpyGuard requires the host where it gets installed to have one free (unused) wireless interface to create an ephemeral access point where to connect the target device for analysis. SpyGuard also needs another network interface, which can either be wireless or wired, for connecting to the network. SpyGuard relies on **Suricata** [385], an open source network IDS, IPS and network security monitoring engine, and uses **heuristic methods** and **Indicators Of Compromise** (**IOC**) for detecting suspicious traffic.

14.1.1 SpyGuard requirements and issues

In order to create an access point, the host network card must support AP mode. This can be verified by running the command below:

```
iw list | grep -E "* AP$" | wc -l
```

If the result is greater than 0, it means there's at least one wireless interface that supports AP mode. Unfortunately, in our case, it's been necessary to buy a USB Wi-Fi adapter compatible with Ubuntu 24.04 which supports AP mode (BrosTrend AC1L). The adapter's drivers installation has been easy to complete and SpyGuard setup has been performed without any issues. It is **highly recommended** to disable the "auto reconnect" setting from any known network on the tested device, before connecting to the ephemeral AP.

On average, for each spy app, SpyGuard has been kept running for 15 minutes in order to collect the traffic generated by the execution of remote commands.

14.1.2 Testing SpyGuard on Android spy apps

MobileTracker free SpyGuard analysis

SpyGuard detected MobileTracker free and generated the following alerts:


```
{
    "title": "A DNS request have been done to live.mobile-tracker-free.com which is tagged as
        STALKERWARE.",
    "description": "The domain name live.mobile-tracker-free.com seen in the capture has been
        explicitly tagged as malicious. This indicates that your device is likely compromised and
        needs to be investigated deeply.",
    "host": "live.mobile-tracker-free.com",
    "level": "High",
    "id": "IOC-03"
},
{
    "title": "A DNS request have been done to fileexplorer.mobile-tracker-free.com which is
        tagged as STALKERWARE.",
    "description": "The domain name fileexplorer.mobile-tracker-free.com seen in the capture has
        been explicitly tagged as malicious. This indicates that your device is likely
        compromised and needs to be investigated deeply.",
    "host": "fileexplorer.mobile-tracker-free.com",
    "level": "High",
    "id": "IOC-03"
},
{
    "title": "A DNS request have been done to mobile-tracker-data.com which is tagged as
        STALKERWARE.",
    "description": "The domain name mobile-tracker-data.com seen in the capture has been
        explicitly tagged as malicious. This indicates that your device is likely compromised and
        needs to be investigated deeply.",
    "host": "mobile-tracker-data.com",
    "level": "High",
    "id": "IOC-03"
}
```


We can clearly recognise the domains which are used by the app for:

- Live streaming from the device's camera (`live.mobile-tracker-free.com`);
- File system exploration of the device (`fileexplorer.mobile-tracker-free.com`);
- Presumably, data upload, e.g., photos, videos, etc... (`mobile-tracker-data.com`);

Connection to the domains above has been performed with TLS 1.3.

Cerberus (Play Store) SpyGuard analysis

Due to Cerberus free license expiration, without having the possibility to create a new account for obtaining a new 3-days free trial on the third party version of the app, the Play Store version of Cerberus has been used for this analysis. Supposedly, the results obtained from the Play Store app are a sub-set of the results which would have been obtained by using the third party app, due to the reduced number of available remote commands. Anyway, SpyGuard successfully detected Cerberus as a stalkerware.

Here the alert produced by the tool:


```
{
    "title": "A DNS request have been done to www.cerberusapp.com which is tagged as
        STALKERWARE.",
    "description": "The domain name www.cerberusapp.com seen in the capture has been explicitly
        tagged as malicious. This indicates that your device is likely compromised and needs to
        be investigated deeply.",
    "host": "www.cerberusapp.com",
    "level": "High",
    "id": "IOC-03"
}
```


Even in this case, traffic is encrypted by TLS 1.3.

Hammer Security SpyGuard analysis

Hammer Security is the only Android app which didn't get detected by SpyGuard as no alert was produced. However, by looking at the `records.json` file generated by the tool, we can clearly see the domains which have been reached by the app:


```
{
    "ip_dst": "216.239.36.54",
    "whitelisted": true,
    "suspicious": false,
    "protocols": [
        {
            "name": "TLS",
            "port": 443
        },
        {
            "name": "TCP",
            "port": 443
        }
    ],
    "domains": [
        "us-central1-hammer-security.cloudfunctions.net"
    ]
},
{
    "ip_dst": "34.120.206.254",
    "whitelisted": true,
    "suspicious": false,
    "protocols": [
        {
            "name": "TLS",
            "port": 443
        }
    ],
    "domains": [
        "hammer-security.firebaseio.com"
    ]
}
```


Just like with the previous reports, SpyyGuard produces a `.pcap` file, with all the captured packets, which can be opened with Wireshark. However, all traffic is encrypted by TLS 1.3.

14.1.3 Testing SpyGuard on iOS spy apps

MMGuardian SpyGuard analysis

SpyGuard did not detect MMGuardian but connection via TLS to the MDM server has been successfully intercepted and logged by the tool:


```
{
    "ip_dst": "142.250.180.147",
    "whitelisted": true,
    "suspicious": false,
    "protocols": [
        {
            "name": "TLS",
            "port": 443
        }
    ],
    "domains": [
        "family.mmguardian.com"
    ]
}
```


Qustodio SpyGuard analysis

SpyGuard did not detect Qustodio but connection via TLS to the MDM server has been successfully intercepted and logged by the tool:


```
{
    "ip_dst": "3.211.130.217",
    "whitelisted": false,
    "suspicious": false,
    "protocols": [
        {
            "name": "TLS",
            "port": 443
        }
    ],
    "domains": [
        "mdm-service.qustodio.com"
    ]
},
{
    "ip_dst": "34.204.240.131",
    "whitelisted": false,
    "suspicious": false,
    "protocols": [
        {
            "name": "TCP",
            "port": 443
        },
        {
            "name": "TLS",
            "port": 443
        }
    ],
    "domains": [
        "api.qustodio.com"
    ]
}
```

Kidslox SpyGuard analysis

SpyGuard did not detect Kidslox but connection via TLS to the MDM server has been successfully intercepted and logged by the tool:


```
{
    "ip_dst": "104.16.51.111",
    "whitelisted": true,
    "suspicious": false,
    "protocols": [
        {
            "name": "TLS",
            "port": 443
        }
    ],
    "domains": [
        "kidsloxsupport.zendesk.com"
    ]
}
```

14.2 Mobile Verification Toolkit (MVT)

MVT [386] is a tool developed by Amnesty International Security Lab to gather forensic traces helpful to identify a potential compromise of Android and iOS devices.

MVT uses public **Indicators Of Compromise** (**IOC**) to scan mobile devices for potential traces of targeting or infection by known spyware campaigns. Specifically, MVT uses IOC related to the following campaigns:

- Pegasus;
- RCS Lab;
- Quadream KingSpawn;
- Operation Triangulation;
- WyrmSpy & DragonEgg;
- Intellexa Predator;
- Other minor campaigns;

IOC include information related to accessed domains, file paths, Android packages names and others. Each campaign data set contains a **STIX2** file [387] which lists all the indicators [388] and the relationships [389] of a specific malware family.

14.2.1 MVT Android analysis

IOC commands

With the `mvt-android download-iocs` command, all available public IOS have been downloaded. A full backup of the Android device has been performed with the `mvt-android check-adb` command after installing the Android Debug Bridge and enabling USB debugging on the unlocked device. All results have been stored in several JSON files and analysed with the `mvt-android check-iocs` command. Note that, as stated by MVT docs, public IOC alone are insufficient to determine that a device is not targeted by spyware.

Here we list all JSON files which are generated and analysed by MVT:

- `dumpsys_accessibility.json`, `dumpsys_activities.json`, `dumpsys_appops.json`, `dumpsys_battery_daily.json`, `dumpsys_battery_history.json`, `dumpsys_dbinfo.json`, `dumpsys_receivers.json`, `files.json`, `getprop.json`, `info.json`, `packages.json`, `processes.json`, `selinux_status.json`, `settings.json`, `sms.json`;

MVT generates other files as well:

- `command.log`: Sequence of commands executed by MVT during the analysis;
- `dumpsys.txt`: Detailed system dump done with Dumpsys tool [390];
- `logcat.txt`: Log of system messages done with Logcat tool [348];
- `timeline.csv`: Timeline of system events with corresponding UTC timestamp, plugin, event type and description;

Plugin	**Available Events**
Files	`file_modified`
DumpsysBatteryDaily	`battery_daily`
DumpsysAppOps	`Access`, `Reject`
Packages	`package_install`, `package_first_install`, `package_last_update`

Table 14.1: MVT timeline - (some of the) available events

During the analysis, suspicious events are flagged as WARNING. However, it is interesting to notice how events flagged as INFO, which are not necessarily suspicious, may still **leak information regarding spyware activity** and must not be ignored! Moreover, since the `check-iocs` command only analyses JSON files, some other useful information may be found inside the `dumpsys.txt` and the `timeline.csv` files.

APKs download

By using the `download-apks` command, MVT downloads all non-system installed application packages and performs a preliminary analysis. Notice how applications which have been downloaded either from the Play Store or from the manufacturer store (Samsung store in this case) are usually installed by `com.android.vending`. Here we have some examples:

- INFO Found non-system package with name "`com.whatsapp`" installed by "`com.android.vending`" on 2024-07-13 16:30:49;
- INFO Found non-system package with name "`org.telegram.messenger`" installed by "`com.android.vending`" on 2024-07-13 16:30:56;
- INFO Found non-system package with name "`com.instagram.android`" installed by "`com.android.vending`" on 2024-07-13 16:31:38;
- INFO Found non-system package with name "`com.samsung.android.app.notes`" installed by "`com.android.vending`" on 2024-07-13 15:41:25;
- INFO Found non-system package with name "`com.google.android.apps.photos`" installed by "`com.android.vending`" on 2024-07-13 15:38:57;
- INFO Found non-system package with name "`com.sec.android.app.sbrowser`" installed by "`com.android.vending`" on 2024-07-13 15:40:40;
- INFO Found non-system package with name "`com.sec.android.app.popupcalculator`" installed by "`com.android.vending`" on 2024-07-13 15:40:13;
- INFO Found non-system package with name "`com.google.android.apps.youtube.music`" installed by "`com.android.vending`" on 2024-07-13 15:39:43;

It's also interesting to notice the following reports for WhatsApp and Facebook:

- INFO Third-party package "`com.whatsapp`"
  requested 11 potentially dangerous permissions;
- INFO Found non-system package with name "`com.facebook.katana`"
  installed by "`None`" on 2022-01-01 01:00:00;

Downloaded APKs can also be checked against VirusTotal, by performing a **SHA256 lookup**, with the following command (requires VirusTotal API key):

```
MVT_VT_API_KEY=<key> mvt-android download-apks --output /outFolder --virustotal
```

The previous command both downloads the APKs from the device and performs the lookups on VirusTotal. In case the APKs have already been downloaded and only the VirusTotal lookup is needed, the command syntax slightly changes as follows:

```
MVT_VT_API_KEY=<key> mvt-android download-apks --from-file apks.json --virustotal
```

The `download-apks` command also generates a more detailed report in the `apks.json` folder. Each entry of the JSON object list is structured as follows:


```
{
    "package_name": "com.example",
    "file_name": "/data/app/.../base.apk",
    "installer": "com.example.installer",
    "disabled": false,
    "system": false,
    "third_party": true,
    "files": [
        {
            "path": "/data/app/.../base.apk",
            "md5": "md5_hash",
            "sha1": "sha1_hash",
            "sha256": "sha256_hash",
            "sha512": "sha512_hash",
            "local_path": "/pathToFolderWhereAPKsHaveBeenDownloaded"
        }
    ],
    "uid": "",
    "version_name": "123.0.0.40.567",
    "version_code": "12345678 minSdk=28 targetSdk=34",
    "timestamp": "2024-07-13 16:31:38",
    "first_install_time": "2024-07-13 16:31:39",
    "last_update_time": "2024-07-13 16:31:39",
    "permissions": [
        {
            "name": "android.permission.EXAMPLE",
            "granted": true,
            "type": "install/declared/runtime"
        }
    ],
    "requested_permissions": [
        "android.permission.EXAMPLE"
    ]
}
```

14.2.2 MobileTracker free MVT analysis

IOC checking

Here the results of the `check-iocs` command:

- `settings.json`
  - WARNING Suspicious system settings
    - `samsung_errorlog_agree = 0` (disabled sharing of crash logs with manufacturer);
    - `send_security_reports = 0` (disabled sharing of security reports);
  - WARNING Suspicious secure settings
    - `accessibility_enabled = 1` (enabled accessibility services);
    - `install_non_market_apps = 1` (enabled installation of non Google Play apps);
  - WARNING Suspicious global settings
    - `package_verifier_user_consent = -1` (disabled Google Play Protect);
- `dumpsys_battery_daily.json`, `dumpsys_battery_history.json` (2 matches)
  - WARNING Found a known suspicious app with ID "`com.mtf.download`" matching indicators from "MobileTrackerFree".

    NOTE: This is actually the installer app "DownloadMobileTrackerFree" and not the real spy app. Moreover, the installer app had already been removed from the device at the time of the analysis, but MVT still managed to find traces of it.
- `dumpsys_accessibility.json`
  - INFO Found installed accessibility service:
    - `phone.build2000/com.app.service.AccessService`

  NOTE: `phone.build2000` is the package name for MobileTracker free;
- `dumpsys_receivers.json`
  - INFO Found a receiver monitoring telephony state/incoming calls:
    - `phone.build2000/com.app.received.CallReceived`;
  - INFO Found a receiver monitoring telephony outgoing calls:
    - `phone.build2000/com.app.received.CallReceived`;
  - INFO Found a receiver to intercept incoming SMS messages:
    - `phone.build2000/com.app.received.SMSReceived`;

Timeline checking

Since we know the package name of MobileTracker free, we can still extract other information from the non-JSON files. In a real scenario, the package name would be unknown, but this wouldn't stop one from looking for details like the ones below, which may give a hint on the spyware presence inside the system.

Timestamp	Plugin	Event	Description
2024-07-15	DumpsysBatteryDaily	battery_daily	Recorded update of package phone.build2000 with vers 510
2024-07-16	DumpsysBatteryDaily	battery_daily	Recorded update of package phone.build2000 with vers 0
2024-08-23	DumpsysBatteryDaily	battery_daily	Recorded update of package phone.build2000 with vers 510
2024-08-23 19:29:01	Packages	package_install	phone.build2000 (system: False, third party: True)
2024-08-23 19:29:04	Packages	package_last_update	phone.build2000 (system: False, third party: True)
2024-08-23 19:29:04	Packages	package_first_install	phone.build2000 (system: False, third party: True)
2024-08-23 19:29:12.276000	DumpsysAppOps	Access	phone.build2000 access to WAKE_LOCK
2024-08-23 19:31:00.593000	DumpsysAppOps	Reject	phone.build2000 access to ACCESS_RESTRICTED_SETTINGS
2024-08-23 19:31:01.257000	DumpsysAppOps	Reject	phone.build2000 access to ACCESS_RESTRICTED_SETTINGS
2024-08-23 19:31:26.430000	DumpsysAppOps	Access	phone.build2000 access to TOAST_WINDOW
2024-08-23 19:31:30.223000	DumpsysAppOps	Access	phone.build2000 access to TOAST_WINDOW
2024-08-23 19:31:45.738000	DumpsysAppOps	Access	phone.build2000 access to REQUEST_DELETE_PACKAGES
2024-08-23 19:31:49.285000	DumpsysAppOps	Access	phone.build2000 access to BIND_ACCESSIBILITY_SERVICE
2024-08-23 19:31:49.375000	DumpsysAppOps	Access	phone.build2000 access to ACCESS_RESTRICTED_SETTINGS
2024-08-23 19:40:48.251000	DumpsysAppOps	Access	phone.build2000 access to READ_EXTERNAL_STORAGE
2024-08-23 19:41:07.815000	DumpsysAppOps	Access	phone.build2000 access to START_FOREGROUND
2024-08-23 19:41:08.001000	DumpsysAppOps	Access	phone.build2000 access to MONITOR_LOCATION
2024-08-23 19:41:08.449000	DumpsysAppOps	Reject	phone.build2000 access to USE_ICC_AUTH_WITH_DEVICE_IDENTIFIER
2024-08-23 19:41:08.451000	DumpsysAppOps	Reject	phone.build2000 access to READ_DEVICE_IDENTIFIERS
2024-08-23 19:41:08.506000	DumpsysAppOps	Access	phone.build2000 access to GET_USAGE_STATS
2024-08-23 19:41:09.634000	DumpsysAppOps	Access	phone.build2000 access to FINE_LOCATION
2024-08-23 19:41:12.753000	DumpsysAppOps	Access	phone.build2000 access to READ_CONTACTS
2024-08-23 19:41:14.028000	DumpsysAppOps	Access	phone.build2000 access to ACCESS_ACCESSIBILITY
2024-08-24 17:25:06.652000	DumpsysAppOps	Access	phone.build2000 access to BIND_ACCESSIBILITY_SERVICE
2024-08-24 17:26:39.032000	DumpsysAppOps	Access	phone.build2000 access to ACCESS_RESTRICTED_SETTINGS
2024-08-24 17:26:47.839000	DumpsysAppOps	Access	phone.build2000 access to ACCESS_RESTRICTED_SETTINGS
2024-08-24 17:27:00.660000	DumpsysAppOps	Access	phone.build2000 access to ACCESS_ACCESSIBILITY
2024-08-24 17:27:01.581000	DumpsysAppOps	Access	phone.build2000 access to START_FOREGROUND
2024-08-24 17:27:01.606000	DumpsysAppOps	Access	phone.build2000 access to START_FOREGROUND
2024-08-24 17:27:01.800000	DumpsysAppOps	Reject	phone.build2000 access to USE_ICC_AUTH_WITH_DEVICE_IDENTIFIER
2024-08-24 17:27:01.802000	DumpsysAppOps	Reject	phone.build2000 access to READ_DEVICE_IDENTIFIERS
2024-08-24 17:27:02.591000	DumpsysAppOps	Access	phone.build2000 access to MONITOR_LOCATION
2024-08-24 17:27:53.133000	DumpsysAppOps	Access	phone.build2000 access to READ_CONTACTS
2024-08-24 17:28:22.778000	DumpsysAppOps	Access	phone.build2000 access to READ_CALENDAR
2024-08-24 17:37:47.986000	DumpsysAppOps	Access	phone.build2000 access to FINE_LOCATION
2024-08-24 17:39:28.145000	DumpsysAppOps	Access	phone.build2000 access to ACCESS_RESTRICTED_SETTINGS
2024-08-24 17:42:41.408000	DumpsysAppOps	Access	phone.build2000 access to READ_SMS
2024-08-24 17:43:13.586000	DumpsysAppOps	Access	phone.build2000 access to ACCESS_ACCESSIBILITY

Table 14.2: MobileTracker free MVT timeline

Not only we can see the exact moment where each permission has been granted/denied to the spy app, but we can also determine when the application has been installed.

APK MVT analysis

- INFO Third-party package "`phone.build2000`" requested 15 potentially dangerous permissions;
- INFO Found non-system package with name "`phone.build2000`" installed by "`com.google.android.packageinstaller`" on 2024-08-23 19:29:01;

The only **important details** which should be taken into account are the number of potentially dangerous permissions and the name of the application which installed the suspect APK. In this case, since MobileTracker free is downloaded and installed from the installer app "DownloadMobileTrackerFree", its APK is the only one in the list which has been installed by the Android package installer `com.google.android.packageinstaller`.

14.2.3 Cerberus Antitheft (3rd party) MVT analysis

IOC checking

- `settings.json`
  - WARNING Suspicious system settings
    - `samsung_errorlog_agree = 0` (disabled sharing of crash logs with manufacturer);
    - `send_security_reports = 0` (disabled sharing of security reports);
  - WARNING Suspicious secure settings
    - `accessibility_enabled = 1` (enabled accessibility services);
    - `install_non_market_apps = 1` (enabled installation of non Google Play apps);
  - WARNING Suspicious global settings
    - `package_verifier_user_consent = -1` (disabled Google Play Protect);
- `processes.json`, `dumpsys_battery_history.json`, `dumpsys_receivers.json`, `dumpsys_activities.json`, `dumpsys_accessibility.json`, `dumpsys_dbinfo.json`, `dumpsys_appopps.json`, `packages.json` (8 matches)
  - WARNING Found a known suspicious app with ID "`com.ssurebrec`" matching indicators from "Cerberus";
- `dumpsys_receivers.json`
  - INFO Found a receiver to intercept incoming SMS messages:
    - `com.ssurebrec/com.surebrec.SmsReceiver`;
  - INFO Found a receiver monitoring outgoing calls:
    - `com.ssurebrec/com.surebrec.StartReceiver`;

  Other receivers which have not been reported by MVT but are listed inside the file are `AdminReceiver`, `BootReceiver`, `AirplaneModeReceiver`.
- `dumpsys_accessibility.json`
  - INFO Found installed accessibility service:
    - `com.ssurebrec/com.surebrec.AccService (A11yTool)`

NOTE: Cerberus (3rd party) is the app that matched most IOC and `com.ssurebrec` is its package name.

APK MVT analysis

- INFO Third-party package "`com.ssurebrec`" requested 12 potentially dangerous permissions;
- INFO Found non-system package with name "`com.ssurebrec`" installed by "`None`" on 2024-08-26 21:15:58;

NOTE: Installation of Cerberus, this time, was performed with ADB. That's probably the reason for MVT to list it as installed by "None".

Logcat report

The `logcat.txt` file produced by Cerberus contains far more data than usual. However, it's not really easy to interpret. Let's go over some log data...

Call phone command used to call a number in speaker mode:

```
08-26 22:06:20.788 1063 3491 I Telecom:
SamsungTelecomServiceImpl: placeCall - handle: *** / extras: Bundle[{
    android.telecom.extra.PHONE\_ACCOUNT_HANDLE=ComponentInfo{
        com.android.phone/com.android.services.telephony.TelephonyConnectionService
    },
    1, UserHandle{0}, android.telecom.extra.START_CALL_WITH_SPEAKERPHONE=true
}]
/ callingPackage: com.ssurebrec / callingUser: UserHandle{0}

08-26 22:06:20.841 1063 3491 I Telecom:
Event: RecordEntry TC@1: CREATED, com.ssurebrec;
requestedAcct: ComponentInfo{
    com.android.phone/com.android.services.telephony.TelephonyConnectionService
},
1, UserHandle{0}, selfMgd: false, OUTGOING, false: TSI.pC(cs)@AY4
```

Here we can see the use of the `Telecom` framework [391], the `ConnectionService` API [392], and the `placeCall` function [393].

Take picture command used for taking a picture from the front camera:

```
08-26 22:03:15.977 818 2923 I CameraService:
CameraService::connect call (PID 17493 "com.ssurebrec",camera ID 1) and Camera API version 1

08-26 22:03:15.991 818 2923 I CameraService:
CameraService::validateClientPermissionsLocked is ok :calling pid 17493, calling uid 10258,
client com.ssurebrec, cameraservice pid=818, device user 0, currently allowed device users:0

08-26 22:03:15.999 818 2923 I Camera2ClientBase: Camera 1: Opened.
Client: com.ssurebrec (PID 17493, UID 10258)

08-26 22:03:16.003 818 2923 I CameraService: service.camera.client set to com.ssurebrec

08-26 22:03:16.151 818 2923 I CameraService: startCameraOps:
Start camera ops, package name = com.ssurebrec, client UID = 10258

08-26 22:03:16.190 711 974 I ExynosCameraConfigurationsSec:
[CAM(1)][Front_0]-(checkClientPackageName[1836]):
packageName (com.ssurebrec) cameraClient (0) m_isLowPowerApp (0)

08-26 22:03:16.460 818 19937 I CameraService: startCameraStreamingOps:
Start camera streaming ops, package name = com.ssurebrec, client UID = 10258

08-26 22:03:20.891 818 2923 I CameraService: finishCameraOps:
Finish camera ops, package name = com.ssurebrec, client UID = 10258

08-26 22:03:20.891 818 2923 I CameraService: finishCameraStreamingOps:
finish camera streaming ops, package name = com.ssurebrec, client UID = 10258
```

Capture video command used for recording a 10 seconds video from the back camera:

```
08-26 22:03:32.395 818 2923 I CameraService:
CameraService::connect call (PID 17493 "com.ssurebrec",camera ID 0) and Camera API version 1

08-26 22:03:32.397 818 2923 I CameraService:
CameraService::validateClientPermissionsLocked is ok :calling pid 17493, calling uid 10258,
client com.ssurebrec, cameraservice pid=818, device user 0, currently allowed device users:0

08-26 22:03:32.398 818 2923 I Camera2ClientBase: Camera 0: Opened.
Client: com.ssurebrec (PID 17493, UID 10258)

08-26 22:03:32.399 818 2923 I CameraService: service.camera.client set to com.ssurebrec

08-26 22:03:32.411 818 2923 I CameraService: startCameraOps:
Start camera ops, package name = com.ssurebrec, client UID = 10258

08-26 22:03:32.426 711 20152 I ExynosCameraConfigurationsSec:
[CAM(0)][Back_0]-(checkClientPackageName[1836]):
packageName (com.ssurebrec) cameraClient (0) m_isLowPowerApp (0)

08-26 22:03:32.513 818 20153 I CameraService: startCameraStreamingOps:
Start camera streaming ops, package name = com.ssurebrec, client UID = 10258

08-26 22:03:45.016 818 6377 I CameraService: finishCameraOps:
Finish camera ops, package name = com.ssurebrec, client UID = 10258

08-26 22:03:45.016 818 6377 I CameraService: finishCameraStreamingOps:
finish camera streaming ops, package name = com.ssurebrec, client UID = 10258
```

NOTE: Both in this and the previous command we can recognise a pattern which makes us understand how Cerberus is accessing the camera.

1. Connect call to camera and permissions validation;
2. Opening the camera and setting the camera service to the app;
3. `startCameraOps` and `startCameraSteamingOps`;
4. `finishCameraOps` and `finishCameraStreamingOps`;

Start application command used to launch WhatsApp:

```
08-26 22:05:20.822 1063 3491 I wm_create_activity: [
    0,15330618,25,com.whatsapp/.Main,android.intent.action.MAIN,NULL,NULL,
    268435456,startActivityAsUser:com.ssurebrec:-1
]
```

When creating the activity for launching WhatsApp, the `FLAG_ACTIVITY_NEW_TASK` [394] flag is used (value 268435456), which brings WhatsApp to the screen.

Get SMS log command used for retrieving SMS messages from the register:

```
08-26 22:04:26.137 1417 1430 D SmsApplication: No PackageName Pattern: com.ssurebrec
```

NOTE: This is the only trace of SMS messages being associated somehow to `com.ssurebrec` package. The letter `D` means this is a debug log. Since the log timestamp corresponds to the time when the command was received by the application (confirmed by checking Cerberus logs from the app), this may be the only trace of the "Get SMS log" command execution.

Record audio command for recording a 10 seconds long audio:

```
08-26 22:02:32.834 1063 3465 I ActivityManager:
Background started FGS: Allowed [
    callingPackage: com.ssurebrec; callingUid: 10258;
    uidState: TOP ; uidBFSL: [BFSL];
    intent: Intent {cmp=com.ssurebrec/com.surebrec.RecordAudioService (has extras)};
    code:PROC_STATE_TOP; tempAllowListReason:
    <,reasonCode:SYSTEM_ALLOW_LISTED,duration:9223372036854775807,callingUid:-1>;
    targetSdkVersion:26; callerTargetSdkVersion:26; startForegroundCount:0;
    bindFromPackage:null: isBindService:false
]

08-26 22:02:32.834 1063 1099 I am_wtf: [
    0, 1063, system_server, -1, ActivityManager,
    Background started FGS: Allowed [
        callingPackage: com.ssurebrec; callingUid: 10258;
        uidState: TOP ; uidBFSL: [BFSL];
        intent: Intent {cmp=com.ssurebrec/com.surebrec.RecordAudioService (has extras)};
        code:PROC_STATE_TOP; tempAllowListReason:
        <,reasonCode:SYSTEM_ALLOW_LISTED,duration:9223372036854775807,callingUid:-1>;
        targetSdkVersion:26; callerTargetSdkVersion:26; startForegroundCount:0;
        bindFromPackage:null: isBindService:false
    ]
]

08-26 22:02:32.854 1063 3131 I am_foreground_service_start: [
    0,com.ssurebrec/com.surebrec.RecordAudioService,
    1,PROC_STATE_TOP,26,26,0,0,0,1,UNKNOWN,0
]

08-26 22:02:33.421 1063 3465 V AudioService.RecordingActivityMonitor:
updateSoundAppPolicy packageName = com.ssurebrec

08-26 22:02:43.509 1063 3497 V AudioService.RecordingActivityMonitor:
updateSoundAppPolicy packageName = com.ssurebrec

08-26 22:02:45.060 1063 3494 I am_foreground_service_stop: [
    0,com.ssurebrec/com.surebrec.RecordAudioService,
    1,PROC_STATE_TOP,26,26,1,1,12206,1,STOP_SERVICE,0
]
```

dumpsys_appops checking

The `dumpsys_appops.json` file is worth of notice because it reports every single moment during which the app used its permissions. This file confirms all the results obtained from Logcat. First of all, we can confirm when the app executed the "Get SMS log" command:

```
{
    "name": "READ_SMS",
    "entries": [
        {
            "access": "Access",
            "type": "fgsvc-s",
            "timestamp": "2024-08-26 22:04:26.110000"
        }
    ],
    "access": "allow"
}
```

Then, we can also confirm access to camera and microphone:


```
{
    "name": "CAMERA",
    "entries": [
        {
            "access": "Access",
            "type": "fgsvc-s",
            "timestamp": "2024-08-26 22:03:32.514000"
        }
    ],
    "access": "allow"
},
{
    "name": "RECORD_AUDIO",
    "entries": [
        {
            "access": "Access",
            "type": "top-s",
            "timestamp": "2024-08-26 22:02:33.384000"
        },
        {
            "access": "Access",
            "type": "fgsvc-s",
            "timestamp": "2024-08-26 22:03:33.731000"
        }
    ],
    "access": "allow"
}
```


However, as we can see from the timestamps, it looks like only the "Capture video" (first `CAMERA` entry + second `RECORD_AUDIO` entry) and the "Record audio" (first `RECORD_AUDIO` entry) commands of Cerberus have been recorded by this file. There are no traces of another access to the camera for executing the "Take picture" command here (but Logcat found them).

We can also see when the app tracked the device's location via GPS:


```
{
    "name": "FINE_LOCATION",
    "entries": [
        {
            "access": "Access",
            "type": "top-s",
            "timestamp": "2024-08-26 22:02:42.674000"
        },
        {
            "access": "Access",
            "type": "fgsvc-s",
            "timestamp": "2024-08-26 22:10:13.486000"
        }
    ],
    "access": "allow"
}
```

14.2.4 Hammer Security MVT analysis

IOC checking

- `dumpsys_accessibility.json`
  - INFO Found installed accessibility service:
    - `com.hammersecurity/.Services.AccessibilityService1`
  - INFO Found installed accessibility service:
    - `com.hammersecurity.Services.AccessibilityService1`

Since it was pretty unusual to find two almost identical entries, further analysis has been conducted by extracting the `AndroidManifest.xml` file from the APK by using `apktool`. In the manifest, only one accessibility service is declared:

```
<service android:canRequestFingerprintGestures="true" android:directBootAware="true"
    android:exported="true" android:label="Hammer"
    android:name="com.hammersecurity.Services.AccessibilityService1"
    android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE">
    <intent-filter>
        <action android:name="android.accessibilityservice.AccessibilityService"/>
    </intent-filter>
    <meta-data android:name="android.accessibilityservice"
               android:resource="@xml/accessibility"/>
</service>
```

APK MVT analysis

- INFO Found non-system package with name "`com.hammersecurity`" installed by "`com.android.vending`" on 2024-08-25 20:28:50;

Timeline checking

Along with installation events and permissions granting, there are traces of (what looks like) the app storing the last recorded video (from the back camera) on the device:

```
2024-08-25 19:30:28.379479 Files file_modified
/sdcard/Android/data/com.hammersecurity/files/Hammer/back_video.mp4

2024-08-25 19:30:28.379479 Files file_modified
/storage/emulated/0/Android/data/com.hammersecurity/files/Hammer/back_video.mp4
```

When looking for spyware traces, this could be considered as a suspect behavior because why would an application ever store an MP4 file on the device without the user knowing?

Logcat report

In the `logcat.txt` file we can find some other details which should arise suspicions:

```
08-25 21:33:28.197 1064 3451 W PackageManager:
Cannot suspend package "com.hammersecurity": has an active device admin
```

We can also see some information related to the usage of the camouflaged calculator icon:

```
08-25 21:29:58.779 1832 23790 I HoneySpace.IconCache:
add icon DB: [com.hammersecurity/.Alias_2#UserHandle{0}]
IconAndLabel(icon=android.graphics.Bitmap@7a0f7ed, label=HS Calculator)
```

There are also traces of commands execution, like camera access:

```
08-25 21:22:24.899 822 4041 I CameraService:
CameraService::connect call (PID 20647 "com.hammersecurity", camera ID 1) and Camera API version 2

08-25 21:22:24.900 822 4041 I CameraService:
isUidActiveLocked E: uid 10257, callingPackage com.hammersecurity, isRegistered true

08-25 21:22:24.900 822 4041 I CameraService:
isUidActiveLocked: start to wait until com.hammersecurity gets into active.

08-25 21:22:24.902 822 4041 I CameraService:
isUidActiveLocked X: uid 10257, callingPackage com.hammersecurity, isActive true.

08-25 21:22:24.910 822 4041 I CameraService:
CameraService::validateClientPermissionsLocked is ok :calling pid 20647, calling uid 10257,
client com.hammersecurity ,cameraservice pid=822, device user 0, currently allowed device users: 0

08-25 21:22:24.917 822 4041 I Camera2ClientBase:
Camera 1: Opened. Client: com.hammersecurity (PID 20647, UID 10257)

08-25 21:22:24.917 822 4041 I CameraService:
isNoSALoggingPackage: com.hammersecurity is a third party package.

08-25 21:22:24.919 822 4041 I CameraService:
service.camera.client set to com.hammersecurity

08-25 21:22:25.085 822 4041 I CameraService:
startCameraOps: Start camera ops, package name = com.hammersecurity, client UID = 10257

08-25 21:22:25.111 822 4041 I CameraServicePoxyWrapper:
updateProxyDeviceState: notifyCameraState for Camera ID 1, newState 0, facing 1,
clientName com.hammersecurity, apiLevel 2

08-25 21:22:25.112 822 4041 I CameraService:
isNoSALoggingPackage: com.hammersecurity is a third party package.

08-25 21:22:26.049 822 23726 I CameraService:
startCameraStreamingOps: Start camera streaming ops,
package name = com.hammersecurity, client UID = 10257

08-25 21:22:26.054 822 23726 I CameraServicePoxyWrapper:
updateProxyDeviceState: notifyCameraState for Camera ID 1, newState 1, facing 1,
clientName com.hammersecurity, apiLevel 2

08-25 21:22:26.958 822 23726 I CameraService:
finishCameraStreamingOps: finish camera streaming ops,
package name = com.hammersecurity, client UID = 10257

08-25 21:22:26.959 822 23726 I CameraServiceProxyWrapper:
updateProxyDeviceState: notifyCameraState for Camera ID 1, newState 2, facing 1,
clientName com.hammersecurity, apiLevel 2
```

Dumpsys receivers checking

Since Hammer Security does not instantiate any receiver for SMS messages nor calls, MVT does not flag it as suspicious. However, we can still see what receivers does it use from the `dumpsys_receivers.json` file. The most relevant ones are probably:

- `com.google.firebase.iid.FirebaseInstanceIdReceiver`;
- `BroadcastReceivers.BootReceiver`;
- `BroadcastReceivers.AdminReceiver`;

While the first two receivers are used by other applications too, `com.hammersecurity` is the only one to instantiate the `AdminReceiver` broadcast receiver. However, this is not flagged by MVT.

dumpsys_appops checking

Despite MVT not flagging Hammer Security as suspicious, due to IOC mismatches, in this file we can find everything we need to track Hammer Security's suspicious activity.

Here we can see the app enabling GPS and accessing the device's location:


```
{
    "name": "GPS",
    "entries": [
        {
            "access": "Access",
            "type": "top-s",
            "timestamp": "2024-08-25 21:21:21.846000"
        }
    ],
    "access": "allow"
},
{
    "name": "FINE_LOCATION",
    "entries": [
        {
            "access": "Access",
            "type": "top-s",
            "timestamp": "2024-08-25 21:21:30.864000"
        }
    ],
    "access": "allow"
}
```

Here's also access to camera and microphone:


```
{
    "name": "CAMERA",
    "entries": [
        {
            "access": "Access",
            "type": "fgsvc-s",
            "timestamp": "2024-08-25 21:29:58.501000"
        },
        {
            "access": "Access",
            "type": "fg-s",
            "timestamp": "2024-08-25 21:30:26.620000"
        },
        {
            "access": "Access",
            "type": "cch-s",
            "timestamp": "2024-08-25 21:30:27.701000"
        }
    ],
    "access": "allow"
},
{
    "name": "RECORD_AUDIO",
    "entries": [
        {
            "access": "Access",
            "type": "fgsvc-s",
            "timestamp": "2024-08-25 21:29:58.518000"
        },
        {
            "access": "Access",
            "type": "fg-s",
            "timestamp": "2024-08-25 21:30:26.621000"
        },
        {
            "access": "Access",
            "type": "cch-s",
            "timestamp": "2024-08-25 21:30:27.702000"
        }
    ],
    "access": "allow"
}
```

14.2.5 MVT iOS analysis

MVT analysis of an iOS device requires either the device to be jailbroken to execute a full filesystem dump, which is not our case, or a backup to be analysed. Backups can be produced either with iTunes or by installing the `libimobiledevice` utilities [395] and running the command:

```
idevicebackup2 backup --full /path/to/backup/
```

Encrypted backups will contain more data but will also require the device backup password to be decrypted. To enable backup encryption, the following command can be run:

```
idevicebackup2 -i encryption on
```

Backup decryption can then be performed by running the following commands:

```
mvt-ios decrypt-backup -p DecryptionPassword BACKUP_PATH
```

```
mvt-ios check-backup --output /path/to/output/ /path/to/backup/udid/
```

14.2.6 MMGuardian MVT analysis

IOC already provide all possible information which could otherwise be extracted from the generated JSON files [396]. Such information are sufficient to detect the malicious app:

- INFO `[mvt.ios.modules.backup.backup_info]` Installed Applications:
  - `[com.google.ios.youtube, com.mmguardian.childapp.ios]`;
- INFO `[mvt.ios.modules.backup.profile_events]` On 2024-09-01 13:30:39.576131 process "`com.apple.dmd`" started operation "install" of profile:
  - `com.mmguardian.profile.restrictions.blockapps-xxx`

  Note: `xxx` has been replaced to a sequence of pseudo-random-looking characters;
- WARNING `[mvt.ios.modules.mixed.safary_history]` Found a known suspicious domain:
  - `https://family.mmguardian.com/apple/enroll?token=xxx`

  matching indicator "`family.mmguardian.com`" from "MMGuardian";
- WARNING `[mvt.ios.modules.mixed.safary_history]` Found a known suspicious domain:
  - `https://www.mmguardian.com/screen-time-mmguardian`

  matching indicator "`mmguardian.com`" from "MMGuardian";
- WARNING `[mvt.ios.modules.mixed.webkit_resource_load_statistics]` Found a known suspicious domain:
  - `mmguardian.com`

  matching indicator "`mmguardian.com`" from "MMGuardian";

14.2.7 Qustodio MVT analysis

IOC already provide all possible information which could otherwise be extracted from the generated JSON files [396]. Such information are sufficient to detect the malicious app:

- INFO `[mvt.ios.modules.backup.backup_info]` Installed Applications:
  - `[com.google.ios.youtube, com.qustodio.mdm.app.family.pro]`;
- INFO `[mvt.ios.modules.backup.profile_events]` On 2024-09-02 12:14:14.479883 process "`com.apple.purplebuddy`" started operation "install" of profile:
  - `com.qustodio.profile.mobileconfig-xxx`

  Note: `xxx` has been replaced to a sequence of pseudo-random-looking characters;
- INFO `[mvt.ios.modules.backup.profile_events]` On 2024-09-02 process "`com.apple.dmd`" started operation "install" of profile:
  - `com.qustodio.profiles.restrictions-xxx` (12:14:21.588256)
  - `com.qustodio.profiles.certificates-xxx` (12:14:22.580748)
  - `com.qustodio.mdm.app.family.pro.xxx` (12:14:23.553012)

  Note: `xxx` has been replaced to a sequence of pseudo-random-looking characters which is different for each installed profile;
- WARNING `[mvt.ios.modules.mixed.webkit_resource_load_statistics]` Found a known suspicious domain:
  - `qustodio.com`

  matching indicator "`qustodio.com`" from "Qustodio";
- WARNING `[mvt.ios.modules.mixed.applications]` Found a known suspicious app with ID
  - `com.qustodio.mdm.app.family.pro`

  matching indicators from "Qustodio";
- WARNING Malicious application `com.qustodio.mdm.app.family.pro` identified;

14.2.8 Kidslox MVT analysis

IOC already provide all possible information which could otherwise be extracted from the generated JSON files [396]. This time, MVT didn't detect any suspicious activity. However, we can still see the logs related to the MDM profile creation:

- INFO `[mvt.ios.modules.backup.backup_info]` Installed Applications:
  - `[com.google.ios.youtube, com.kidslox.main]`;
- INFO `[mvt.ios.modules.backup.profile_events]` On 2024-09-02 17:21:54.008573 process "`com.apple.Preferences`" started operation "install" of profile:
  - `com.kidslox.mdm-xxx`

  Note: `xxx` has been replaced to a sequence of pseudo-random-looking characters;
- INFO `[mvt.ios.modules.backup.profile_events]` On 2024-09-02 18.03:07.107579 process "`com.apple.dmd`" started operation "install" of profile:
  - `com.kidslox.settings-aaaaaaaa-bbbb-cccc-dddd-000000000002`

14.2.9 MVT post-uninstall analysis

Detection becomes more difficult after the malicious apps get uninstalled.

Android spy apps

No spy app is installed on the device, which has been turned off for 1 month.

Almost all traces of the apps presence are gone. By using the `grep` command to search the name of each APK in all the report files (`*.json`, `*.txt`, `*.csv`), we obtain some matches.

Note:

- `MyAndroidPackage` can be replaced with any of the apps' package names
  - i.e., `com.hammersecurity`, `com.ssurebrec`, and `phone.build2000`;
- Dates and version numbers in `timeline.csv` have been replaced with generic strings
  - i.e., `YYYY-MM-DD`, and `xxx`;

```
dumpsys_battery_daily.json: "package_name": "MyAndroidPackage"
dumpsys_battery_history.json: "package_name": "MyAndroidPackage"
dumpsys_battery_history.json: "service":"MyAndroidPackage/com.google.android.datatransport
.runtime.scheduling.jobscheduling.JobInfoSchedulerService"

timeline.csv:"YYYY-MM-DD","DumpsysBatteryDaily","battery_daily",
    "Recorded update of package MyAndroidPackage with vers xxx"

dumpsys.txt:mIsQuickReclaimEnabled: true, heavy apps :Camera:{
    org.telegram.messenger, com.hammersecurity, com.whatsapp, com.android.internet.a20240801,
    com.polzhfjo.folwsgar, com.ssurebrec, phone.build2000, com.sec.android.app.camera
}

grep: dumpsys.txt: binary file matches
```

`dumpsys_battery_daily` is useless because it contains records for all the devices which have ever been analysed with MVT, but there's no information which links an application to the device where it has been installed. As a matter of fact, there are records for iOS apps too but they are indistinguishable from records for Android apps. Moreover, the file doesn't report any useful info which could be helpful to classify an app as spyware. So, the `timeline` file is useless too.

`dumpsys_battery_history` reports only the records for Android apps of the current device, but in terms of provided info is no different from the `dumpsys_battery_daily` file, hence useless.

The `dumpsys.txt` file is difficult to interpret as the Android Developer team does not provide any article which fully explains the output of the `dumpsys` command. A possible explanation of the displayed record is that listed apps heavily consume system resources when using the device's camera. Among these apps there are all the malicious APKs which have been installed. This information is not completely useless, but it can't really help when it comes to classify apps as spyware (legitimate apps like WhatsApp make intensive use of camera too).

Actually, the `grep` command finds other matches in the `dumpsys.txt` file, but they are not displayed unless the `--text` option is used. Some of these matches refer to records which may suggest the presence of spyware, but that can't be really confirmed.

This record, present for Hammer Security only, suggests the use of the accessibility service:

```
com.hammersecurity(10259)com.android.server.accessibility.AccessibilityServiceConnection
   #findAccessibilityNodeInfoByAccessibilityId,5259,1971(6)
```

These records expose which application packages are associated to the "persistent direct boot aware API service", which is used for implementing persistence at boot time. Third party apps like WhatsApp and Telegram are not listed here, as well as MobileTracker free (`phone.build2000`). As a matter of fact, none of these applications works after a device reboot, unless the user unlocks the device first. On the other hand, Hammer Security and Cerberus (`com.ssurebrec`) appear in the list, along with other system applications whose package names all start with `com.google.android` and some exceptions like `com.android.vending` (Play Store) and a package which is called `com.google.audio.hearing.visualization.accessibility.scribe`. However, Hammer Security doesn't really work after reboot without unlocking the device first.

```
Association com.google.android.gms.chimera.PersistentDirectBootAwareApiService:
  Process: com.google.android.gms.persistent
  Active count 4916: +2h16m25s689ms / 36%
  Total count 16: +6h16m49s120ms / 98%
  ...
  <- com.hammersecurity/u0a259 (com.hammersecurity):
       Active count 27 (BTop): +29s590ms / 0.13%
       Total count 4: +1m1s273ms / 0.27%
  <- com.ssurebrec/u0a241 (com.ssurebrec):
       Active count 12 (BFgs): +19s15ms / 0.08%
       Total count 3: +38s49ms / 0.17%
  ...
  <- com.ssurebrec/u0a258 (com.ssurebrec):
       Active count 84 (BTop): +14s94ms / 0.06%
       Total count 1: +16s775ms / 0.07%
  ...
  <- com.ssurebrec/u0a260 (com.ssurebrec):
       Total count 2: +30s452ms / 0.13%
```

These records expose which apps are associated to the "Google location manager service", which suggests that these apps are probably using the `LocationManager` class. As stated by Android Developer: "This class provides access to the system location services. These services allow applications to obtain periodic updates of the device's geographical location, or to be notified when the device enters the proximity of a given geographical location" [397]. All the malicious APKs are listed here, as they all constantly monitor the device's location. This list is surprisingly shorter than the previous one.

The remaining packages that use the location manager service are:
`com.sec.location.nsflp2`, `com.google.android.googlequicksearchbox`, `com.google.android.gms`, `com.google.android.apps.messaging`, `system`, `com.google.android.apps.maps`, and `org.telegram.messenger`.

```
Association com.google.android.location.internal.GoogleLocationManagerService:
  Process: com.google.android.gms.persistent
  Active count 3970: +1h49m4s650ms / 28%
  Total count 9: +6h15m56s362ms / 98%
  ...
  <- com.ssurebrec/u0a258 (com.ssurebrec):
       Active count 334 (multi-state): +8m59s816ms / 2.3%
           Bnd Top: +1m24s177ms / 0.37%
               Fgs: +7m35s639ms / 2.0%
       Total count 4: +12m57s329ms / 3.4%
  <- com.hammersecurity/u0a259 (com.hammersecurity):
       Active count 105 (multi-state): +6m20s914ms / 1.7%
           Bnd Top: +5m6s561ms / 1.3%
           Bnd Fgs: +1m14s353ms / 0.32%
       Total count 5: +14m48s249ms / 3.9%
  <- com.ssurebrec/u0a260 (com.ssurebrec):
       Active count 37 (multi-state): +4m19s602ms / 1.1%
           Bnd Top: +3s485ms / 0.02%
               Fgs: +4m16s117ms / 1.1%
       Total count 1: +19m42s475ms / 5.1%
  ...
  <- phone.build2000/u0a263 (phone.build2000):
       Active count 7 (BTop): +9s717ms / 0.04%
       Total count 4: +29m49s786ms / 7.8%
```

These record expose which apps access the contacts list. Both MobileTracker free and Hammer Security are listed. Strangely, Cerberus is not listed.

```
Association com.android.providers.contacts.ContactsProvider2:
  Process: android.process.acore
  Active count 1171: +7m12s630ms / 1.9%
  Total count 17: +3h31m3s912ms / 55%
  <- phone.build2000/u0a263 (phone.build2000):
       Active count 253 (multi-state): +4m20s120ms / 1.1%
           Bnd Top: +2s134ms / 0.01%
           Bnd Fgs: +4m17s986ms / 1.1%
       Total count 9: +12m22s802ms / 3.2%
  ...
  <- com.hammersecurity/u0a259 (com.hammersecurity):
       Active count 12 (BTop): +5s945ms / 0.03%
       Total count 3: +18s130ms / 0.08%
```

These records expose MobileTracker free's usage of the call log provider. Strangely, Cerberus is not listed, despite it being able to access call logs.

```
Association com.android.providers.contacts.CallLogProvider:
  Process: android.process.acore
  Active count 351: +31s154ms / 0.14%
  Total count 15: +5m45s85ms / 1.5%
  <- phone.build2000/u0a263 (phone.build2000):
       Active count 3 (BFgs): +42ms / 0.00%
       Total count 1: +6s257ms / 0.03%
```

These records expose MobileTracker free's usage of the SMS provider. Strangely, Cerberus and Hammer Security are not listed, despite them being able to read SMS messages.

```
Association com.android.providers.telephony.SmsProvider:
  Process: com.android.phone
  Total count 15: +4m39s384ms / 1.2%
  <- phone.build2000/u0a263 (phone.build2000):
       Total count 1: +6s429ms / 0.03%
```

Further inspection of the `dumpsys.txt` file reveals extra details which are interesting. The file is structured as a list of reports, where each report corresponds to the dump of a specific service.

The first record extracted with `grep` (i.e., `mIsQuickReclaimEnabled:...`) is part of the dump of the `ChimeraManagerService`. Although it is not so clear what this service does, it is possible to discover which package it belongs to with ADB. By using the command `adb shell service list`, the tool says that the service is provided by an application package which is proprietary Samsung software: `com.samsung.android.chimera.IChimera`. This APK, just like all system packages, can't be extracted, for further investigation, without a rooted device. However, the dump of the `ChimeraManagerService` includes the following records:

```
Chimera enabled: true
UseDynamicFreeMem: false
MemFreeTarget: 778240

[Chimera Stats]
Using aggressive mode
QuickReclaimKillCnt: 0
QuickReclaimDynamicThreshold: 153600
```

This suggests that `ChimeraManagerService` is probably related to memory management and the "quick reclaim" feature probably involves reclaiming memory from applications which have high memory consumption. In this case, the record extracted with `grep` probably addresses all the malicious apps as "heavy" (in terms of memory) when using the camera. This information may not be available on all devices, as this service probably runs on Samsung devices only.

The record for Hammer Security, related to the accessibility service, belongs to the dump of the service `binder_calls_stats`. All the remaining records belong to the `procstats` service dump.

The `procstats` service dump is actually very helpful, as it reports per-package statistics. Every group of stats refers to one specific "`packageName / UID / version`" tuple. Among the available info, there are the services of each APK. All the malicious APKs declair their services (in their manifest) with human-readable names which are, sometimes, self-explanatory.

```
* com.hammersecurity / u0a259 / v375:
    Service com.hammersecurity.Services.XCameraService:
      Process: com.hammersecurity
    Service com.hammersecurity.Services.NotificationListener:
      Process: com.hammersecurity
    Service com.hammersecurity.Services.AccessibilityService1:
      Process: com.hammersecurity
    Service com.hammersecurity.Services.MessaginService1:
      Process: com.hammersecurity

* com.ssurebrec / u0a258 / v333800:
    Service com.ssurebrec.FCMMessagingService:
      Process: com.ssurebrec
    Service com.ssurebrec.AccService:
      Process: com.ssurebrec
    Service com.google.firebase.sessions.SessionLifecycleService:
      Process: com.ssurebrec
    Service com.ssurebrec.TrackServiceFused:
      Process: com.ssurebrec

* phone.build2000 / u0a263 / v510:
    Service com.app.service.Localization:
      Process: phone.build2000
    Service com.app.service.UploadDataFileExplorer:
      Process: phone.build2000:fileexplorerupload
    Service com.app.service.UploadRecordAudioService:
      Process: phone.build2000
    Service com.app.service.ServiceLocalization:
      Process: phone.build2000
    Service com.app.service.AudioRecordService:
      Process: phone.build2000
    Service com.app.service.UploadPictureRemoteService:
      Process: phone.build2000:uploadpicture
    Service com.app.activity.liveViewing.LiveRTCService:
      Process: phone.build2000
    Service com.app.service.FileExplorerRemote:
      Process: phone.build2000:fileexplorer
    Service com.app.service.AccessService:
      Process: phone.build2000
    Service com.app.service.UploadDataService:
      Process: phone.build2000
    Service com.app.service.NotificationListener:
      Process: phone.build2000
    Service com.app.service.MyFirebaseMTFMessagingService:
      Process: phone.build2000
```

These information are useful to understand what an application may do. There are accessibility services, Google Firebase messaging services (as some spy apps use it to communicate with the C&C server), services for location tracking, services for taking pictures, and more. However, as service names are just strings, they could be obfuscated.

iOS spy apps

The analysis has been performed with no spy app installed on the device and after resetting all device settings. All MDM profiles have been removed as well. By comparing the new decrypted backup with the older one, there's already a major change in the number of occurrences of the apps' names. By running the command `grep --text -r -i "AppName" Path/To/Decrypted/Backup | wc -l`, which is case insensitive, we obtain the following results:

- Occurrences of the string "mmguardian" dropped from 39 to 14;
- Occurrences of the string "qustodio" dropped from 88 to 14;
- Occurrences of the string "kidslox" dropped from 234 to 14;

Moreover, since a full system dump cannot be performed without a jailbreak, detection is even more difficult. By checking the backup with MVT for IOC matches, we find no match and a detection rate of 0. However, there are still INFO records from the "profile_events" module of MVT that show when the Questodio MDM profile was installed and removed from the device. There are no records regarding the other MDM profiles which were previously created and removed. This is interesting because this analysis has been performed three days after installing and removing the Qustodio app and its MDM profile, while all the other apps had already been uninstalled 1 month earlier. This suggests that data regarding removed MDM profiles is deleted from the device after some time.

By inspecting the MVT report files, the only occurrences of "mmguardian" and "kidslox" appear to be in the `Manifest.db` file in the decrypted backup folder. Occurrences are present in relative paths which are stored in such file. Strangely, there are no occurrences of "qustodio".

```
Media/PhotoData/private/com.mmguardian.childapp.ios
Media/PhotoData/external/com.mmguardian.childapp.ios
Media/PhotoData/private/com.kidslox.main.KidsloxPacketTunnelProxyExtension
Media/PhotoData/external/com.kidslox.main.KidsloxPacketTunnelProxyExtension
```

By manually checking the decrypted backup, we can find the remaining occurrences in some backup files. iOS backup files are mostly binary files and there are not so many human-readable strings. There are many binary files which are actually SQLite files and can be dumped with the `sqlite3` command. Other files are Apple binary property list files, which have been further inspected with the `plistutil` command but they don't carry any useful information. Understanding what the malicious apps do is impossible and hence there's no way to tell they are spy apps, unless knowing their name is enough to do so. Here all the strings which have been found in SQLite files:

```
INSERT INTO bundle_info VALUES(64,"com.mmguardian.childapp.ios",48);
INSERT INTO bundle_info VALUES(67,"com.qustodio.mdm.app.family.pro",48);
INSERT INTO bundle_info VALUES(68,"com.kidslox.main",48);

INSERT INTO ZINAPPREVIEWREQUEST VALUES(1,1,1,NULL,746990642.43482995,"com.kidslox.main","9.7.0");

INSERT INTO application_identifier_tab VALUES(231,"com.qustodio.mdm.app.family.pro");
```

There's also a `.bak` file which is classified as an SQLite 3 database disk image by the `file` utility. However, trying to dump it results in an error message showing up, as the "disk image is malformed". Attempts to recover it have been unsuccessful. Manual inspection does not provide any benefit, other than the human-readable strings which had already been found with `grep`.

We can conclude that detection post uninstallation in iOS is not possible with MVT, as the decrypted device backup doesn't carry enough information to classify spy apps as malicious, and a full system dump can't be performed because of the impossibility to jailbreak the device.

Chapter 15

Results

The purpose of this chapter is to make a comparison of the results obtained during the test phase (Chapters 11 and 12) and the detection phase (Chapter 14) of both Android and iOS spy apps, highlighting the differences in terms of spying capabilities and operating system security. All results can be safely considered valid for systems which run Android 14 and iOS 17.6.1, which are the latest available versions of Android and iOS. Results may slightly change when considering older minor versions of Android and iOS, but could be greatly different on previous major versions of these operating systems, as documented in the literature and in this thesis.

15.1 Spy apps testing

15.1.1 Installation and setup

While Android allows to download third party applications which are not published on (nor verified by) the Google Play Store, there's no such thing on iOS, where all the apps must be downloaded from the official App Store. Installation of third party spy apps on Android is only possible after allowing downloads from unknown sources and disabling the Play Protect security measure, which requires fingerprint authentication. Apps which are downloaded from the Play Store don't require to do so.

On Android, spy apps always need device admin privileges and accessibility service permission. In case of third party apps, accessibility service permission can only be granted after enabling restricted settings, which also requires fingerprint authentication. All other permissions can simply be granted by having physical access to the unlocked target device, as no further authentication is needed. On iOS, parental control apps available on the App Store always require the installation of a Mobile Device Management (MDM) profile, which can only be completed by inserting the device's PIN. Permissions do not require any authentication to be granted, but the set of permissions which are available for an iOS app is restricted to location services access and some other minor (less dangerous) permission. Hence, iOS apps have far less features.

15.1.2 Available features

Android spy apps have proven to be able to implement far more features, compared to their iOS alternatives. In that sense, Android spy apps' behaviour is much more similar to a real mobile spyware's behaviour. This difference is due to the fact that the iOS operating system imposes more restrictions on installed apps, making it impossible to implement features like real-time monitoring of camera, microphone, SMS messages and call logs. Such features may only be achievable with a jailbroken device, but there's no way to perform a jailbreak on newer iOS versions. On the other hand, Android spy apps are able to implement all the features above (and even more) without the need to root the target device.

Tested iOS apps have been able to monitor real-time geolocation and browser history (or web searches) but SMS messages and call logs could only be retrieved by performing a device backup over Wi-Fi. This, not only completely defeats the purpose of real-time monitoring, but also makes it more difficult to retrieve such information, as the only way to do so is by being connected to the same network of the target device and perform a backup every single time. No iOS app provided remote access to camera nor microphone.

15.1.3 Stealth and persistence

On Android, spy apps generally employ techniques of icon camouflage and removal from the recents screen. However, all apps are still visible in the app drawer and in the background apps list. App icons can only be hidden from the app drawer by manually checking the corresponding setting in the home screen. Access to peripherals like microphone and camera is always reported, on the top right corner of the device's screen, with the corresponding green icons. Some apps may be able to remotely hide their icon from the app drawer on a rooted device only, as this would allow to install an app as a system application. If properly engineered, Android spy apps can keep operating normally after a system reboot, without the need to unlock the device for the first time by inserting the password (e.g., Cerberus). Sometimes, they can even prevent the user from uninstalling them by taking over the device's screen and overlaying drawable elements on the uninstallation panel, which actively stops the user from pressing the "uninstall" button. However, examples like MobileTracker free do not provide a flawless implementation of such feature. In order to completely block uninstallation (except with factoring reset), an Android app should be able to take over the screen:

- In the home screen, after clicking and holding the app's icon;
- In the app's settings page, where the "uninstall" button is always visible on the bottom of the screen (but disabled if the app has device admin privileges);
- On the device admin apps page, as disabling device admin privileges would allow to uninstall the app;
- On the accessibility page, to prevent disabling the accessibility service;

Additionally, the app should be able to block USB debugging to prevent uninstallation with the Android Debug Bridge (ADB) tool.

On iOS, apps are always visible in the App Library, unless they get restricted by the "Screen Time" settings or by an MDM profile (but in that case they just stop running and can't be launched anymore). Access to peripherals is shown on the top right corner of the device's screen:

- Camera access is shown with a green dot;
- Microphone access is shown with an orange dot;

Tested apps work after reboot, without unlocking the device, but there's actually no feature which they can use for spying on the user while the phone is locked except location tracking. Parental control apps downloaded from the App Store require an additional computer application to be downloaded for providing anti-tampering and preventing uninstallation. Hence, the only way these apps can be setup for anti-tampering is by connecting the mobile device to a computer (which requires to insert the device's PIN) and to run the anti-tampering software. The same procedure must be repeated for disabling such feature.

15.2 Spy apps detection

We have tested Android and iOS apps against the following tools:

- SpyGuard: Dynamic analysis tool which employs heuristic methods and Indicators Of Compromise (IOC) for analysing the intercepted network flows of a target device to detect traces of malware;
- Mobile Verification Toolkit (MVT): Static analysis tool which performs and then analyses with IOC a backup of the device, trying to detect infection traces;

SpyGuard performed worse than MVT, detecting only 2 apps out of 6. MVT, maybe because of the higher number of available IOC (≈ 9000 against the ≈ 7000 used by SpyGuard), detected 4 apps out of 6. It was interesting to notice the differences between Android and iOS backups. On Android, MVT was able to retrieve a full system dump and system logs, thanks to the `dumpsys` and `logcat` tools developed by Android team. This allowed to gain additional information regarding the apps performing maliciously. For example, despite Hammer Security not being detected by MVT, it was still possible to retrieve logs that showed the app accessing camera and microphone, and even storing MP4 files on a private directory before sending them to the command and control server. On iOS, on the other hand, much less information were available and detection was possible only by relying on IOC, as there were no system logs to be "manually checked". However, it should also be considered that iOS apps only accessed location services and web history. Hence, detection may have been harder because of their reduced capabilities.

App	OS	SpyGuard	MVT
Hammer Security	Android	✗	✗
Cerberus	Android	✓	✓
MobileTracker free	Android	✓	✓
MMGuardian	iOS	✗	✓
Qustodio	iOS	✗	✓
Kidslox	iOS	✗	✗

Table 15.1: Spy apps detection results

Further tests have been performed with MVT after having all the spy apps uninstalled. Detection on Android is difficult, but in the `dumpsys.txt` file there are still traces of uninstalled apps and it is possible to understand, up to some extent, what their behaviour is. On iOS, since a full system dump can't be performed without a jailbreak, detection is impossible. Decrypted iOS backup files still have the names of uninstalled apps stored somewhere, but nothing more.

Chapter 16

Conclusions

The goal of this thesis was to study mobile spyware targeting Android and iOS operating systems, and to understand the role they play in digital forensics. Research on iOS has been more difficult to carry out, due to the lack of resources both in the literature and online. On the other hand, as Android is an open source OS, the greater availability of resources and documentation made it possible to study Android spyware more in depth. Different mobile spyware have been presented and discussed, along with some of the most critical vulnerabilities which have been discovered over the years in Android and iOS. Legal parental control and anti-theft mobile applications, often used (illegally) as spy apps, have been tested on the latest versions of Android and iOS. These applications do not exploit any system vulnerability but they behave as intended only if granted with all the permissions they require.

It has been highlighted how Android spy apps offer far more features than iOS apps, due to the softer restrictions imposed by Android. Hence, it would be easier to setup a real scenario where an attacker with physical access to a victim's Android device can perform installation and configuration of a spy app without being easily detected.

After discussing the security measures employed by Google Play Store and App Store, the design of an attack which could lead to spyware installation and configuration on an Android device, without having physical access to it, has been presented. It has been shown how it could be possible to avoid malware detection and successfully publish a malicious application on the store.

Finally, spy apps have been tested against some digital forensics tools and most of them have been successfully detected. The Mobile Verification Toolkit (MVT) has proven to perform significantly better than the network analysis tool SpyGuard. However, it must also be considered that SpyGuard uses heuristic methods to detect the well-known malware families, but it was used to detect applications which are not exactly spyware. Moreover, detection post-uninstallation with MVT is feasible on Android, if certain conditions are met (e.g., application manifests not obfuscated, etc...), but it's impossible on iOS without being able to jailbreak the device and, consequently, to perform a full system dump.

16.1 Future work

The knowledge gained through this document could be used for developing a proof-of-concept Android application with (most of) the features implemented by the tested spy apps. The application should be as stealthy and as persistence as possible, ideally resisting to system reboots and blocking the user from performing uninstallation by adopting techniques similar to the ones implemented by Cerberus and MobileTracker free. The application should avoid malware detection by Play Protect and VirusTotal, possibly adopting the approach described in Section 5.3.1. The main goal would be to develop a malicious application, capable of spying on users, which goes undetected by the anti malware tools which have been described and tested during this thesis (including the digital forensics tools which have been discussed in Chapter 14).

Despite it being far more difficult, it would be interesting to find out if something similar could be done on iOS. All the applications which have been tested, plus others which have not been cited in this document, do not implement remote access to a device's camera nor microphone. Hence, the goal here would be to discover if it's possible to design an iOS app which can be published on the App Store and which can provide either of those features [398].

Other detection tools could be tested to see if they perform better than SpyGuard and MVT.

Bibliography

[1] The StatCounter GlobalStat project, https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] Android developer, "Platform architecture", May 2024, retrieved from https://developer.android.com/guide/platform

[3] Andrew S. Tanenbaum, Herbert Bos, "Modern operating systems", Pearson, 2015

[4] Google LLC, "Android debug bridge (adb)", February 2024, retrieved from https://developer.android.com/tools/adb

[5] Madihah Mohd Saudi, Muhammad Afif Husainiamer, Azuan Ahmad, and Mohd Yamani Idna Idris, "iOS mobile malware analysis: a state-of-the-art", Journal of Computer Virology and Hacking Techniques, May 2023, DOI 10.1007/s11416-023-00477-y

[6] Apple Inc., "Cocoa fundamentals guide", retrieved from https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CocoaFundamentals/Introduction/Introduction.html

[7] Apple Inc., "File Metadata Attributes Reference", retrieved from https://developer.apple.com/library/archive/documentation/CoreServices/Reference/MetadataAttributesRef/MetadataAttrRef.html

[8] Apple Inc., "Media Playback Programming Guide", retrieved from https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/MediaPlaybackGuide/Contents/Resources/en.lproj/Introduction/Introduction.html

[9] Apple Inc., "iOS & iPadOS 17 Release Notes", retrieved from https://developer.apple.com/documentation/ios-ipados-release-notes/ios-ipados-17-release-notes

[10] Apple Inc., "Apple Platform Security", May 2024, retrieved from https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf

[11] Apple Inc., "About Apple File System", retrieved from https://developer.apple.com/documentation/foundation/file_system/about_apple_file_system/

[12] Apple Inc., "Role of Apple File System", May 2022, retrieved from https://support.apple.com/guide/security/role-of-apple-fi le-system-seca6147599e/web

[13] Wikipedia, ".ipa", retrieved from https://en.wikipedia.org/wiki/.ipa

[14] Appdome, "Structure of an iOS App Binary (.ipa)", retrieved from https://www.appdome.com/how-to/devsecops-automation-mobile-cicd/appdome-basics/structure-of-an-ios-app-binary-ipa/

[15] Fileformats, "What is an IPA File?", retrieved from https://fileformats.org/file-types/ios-application/

[16] Apple Inc., "About Information Property List Files", retrieved from https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html

[17] Apple Inc., "Nib Files", retrieved from https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/LoadingResources/CocoaNibs/CocoaNibs.html

[18] Android developer, "System and kernel security", August 2024, retrieved from https://source.android.com/docs/security/overview/kernel-security

[19] Android developer, "Application Sandbox", August 2024, retrieved from https://source.android.com/docs/security/app-sandbox

[20] Android developer, "Security-Enhanced Linux in Android", August 2024, retrieved from https://source.android.com/docs/security/features/selinux
[21] Android developer Google blog, "Here comes Treble: A modular base for Android", May 2017, retrieved from https://android-developers.googleblog.com/2017/05/here-comes-treble-modular-base-for.html
[22] Android developer, "File-Based Encryption", August 2024, retrieved from https://source.android.com/docs/security/features/encryption/file-based
[23] "Ieee standard for cryptographic protection of data on block-oriented storage devices", IEEE Std 1619-2007, 2008, pp. 1–40, DOI 10.1109/IEEESTD.2008.4493450
[24] Google LLC, "Enabling Adiantum", April 2024, retrieved from https://source.android.com/docs/security/features/encryption/adiantum
[25] Android developer, "Metadata Encryption", August 2024, retrieved from https://source.android.com/docs/security/features/encryption/metadata
[26] Android developer, "Hardware-backed Keystore", September 2024, retrieved from https://source.android.com/docs/security/features/keystore
[27] Android developer, "Adoptable storage", August 2024, retrieved from https://source.android.com/docs/core/storage/adoptable
[28] Google LLC, "Configure on-device developer options", April 2024, retrieved from https://developer.android.com/studio/debug/dev-options
[29] Android developer, "Back up user data with Auto Backup", May 2024, retrieved from https://developer.android.com/identity/data/autobackup
[30] Google LLC, "Android 14 compatibility definition", April 2024, retrieved from https://source.android.com/docs/compatibility/14/android-14-cdd#7310_biometric_sensors
[31] Android developer, "Measuring Biometric Unlock Security", August 2024, retrieved from https://source.android.com/docs/security/features/biometric/measure
[32] Google LLC, "Android enterprise security paper", 2023, retrieved from https://services.google.com/fh/files/misc/android-enterprise-security-paper-2023.pdf
[33] Android developer, "Measuring Biometric Unlock Security", August 2024, retrieved from https://source.android.com/docs/security/features/biometric/measure
[34] Android developer, "Fingerprint HIDL", August 2024, retrieved from https://source.android.com/docs/security/features/authentication/fingerprint-hal
[35] C. Scott Brown, "Here are the phone update policies from every major android manufacturer", January 2024, retrieved from https://www.androidauthority.com/phone-update-policies-1658633/
[36] Android developer, "OTA updates", August 2024, retrieved from https://source.android.com/docs/core/ota
[37] Android developer, "Security updates and resources", September 2024, retrieved from https://source.android.com/docs/security/overview/updates-resources
[38] A. Acar, G. S. Tuncay, E. Luques, H. Oz, A. Aris, and S. Uluagac, "50 shades of support: A device-centric analysis of android security updates", 2024, DOI 10.14722/ndss.2024.24175
[39] Google LLC, "Integrate android into your emm solution", January 2024, retrieved from https://developers.google.com/android/work/overview#integrate_android_into_your_emm_solution
[40] Google LLC, "Android devices: management use cases", January 2024, retrieved from https://developers.google.com/android/work/overview#android_devices_management_use_cases
[41] Google LLC, "Build a device policy controller", February 2024, retrieved from https://developer.android.com/work/dpc/build-dpc
[42] Google LLC, "Iam overview", April 2024, retrieved from https://cloud.google.com/iam/docs/overview
[43] Google LLC, "Manage identity and access", August 2023, retrieved from https://cloud.google.com/architecture/framework/security/identity-access#use_a_single_identity_provider

[44] Apple Inc., "About Lockdown Mode", January 2024, retrieved from https://support.apple.com/en-us/105120

[45] National Institute of Standards and Technology, "Recommendation for Key Derivation Using Pseudorandom Functions", Tech. Rep. NIST Special Publication 800 NIST SP 800-108r1-upd1, U.S. Department of Commerce, Washington, D.C., 2022. DOI 10.6028/NIST.SP.800-108r1-upd1

[46] National Institute of Standards and Technology, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", Tech. Rep. NIST SP 800-56A Rev. 3, U.S. Department of Commerce, Washington, D.C., 2018. DOI 10.6028/NIST.SP.800-56Ar3

[47] Paolo Dal Checco, "Learning iOS Forensics", March 2015, retrieved from https://www.dalchecco.it/learning-ios-forensics/

[48] Apple Inc., "Security of runtime process in iOS", February 2021, retrieved from https://support.apple.com/guide/security/security-of-runtime-process-sec15bfe098e/web

[49] Apple Inc., "What is the difference between iMessage and SMS/MMS?", May 2024, retrieved from https://support.apple.com/en-us/104972

[50] Apple Inc., "iCloud security overview", December 2022, retrieved from https://support.apple.com/guide/security/icloud-security-overview-secacde2d0da/web

[51] Apple Inc., "iCloud Data categories and encryption", January 2024, retrieved from https://support.apple.com/en-us/102651#:~:text=Data%20categories%20and%20encryption

[52] Apple Inc., "Designing apps using CloudKit", retrieved from https://developer.apple.com/icloud/cloudkit/designing/

[53] Eli Blumenthal, "Apple Reveals Its iPhone Gets at Least 5 Years of Security Updates", June 2024, retrieved from https://www.cnet.com/tech/mobile/apple-reveals-its-iphone-gets-at-least-five-years-of-security-updates/

[54] Apple Inc., "Apple security releases", June 2024, retrieved from https://support.apple.com/en-us/HT201222

[55] Apple Inc., "About Rapid Security Responses for iOS, iPadOS, and macOS", August 2023, retrieved from https://support.apple.com/en-us/102657

[56] R. Housley, "Cryptographic Message Syntax (CMS)", RFC-5652, September 2009, DOI 10.17487/RFC5652

[57] Android developer, "App signing", September 2024, retrieved from https://source.android.com/docs/security/features/apksigning

[58] Google developer, "Google Play Protect", retrieved from https://developers.google.com/android/play-protect

[59] Google developer, "Cloud-based protections", retrieved from https://developers.google.com/android/play-protect/cloud-based-protections

[60] Google Play, "Google Play Developer Distribution Agreement", retrieved from https://play.google.com/about/developer-distribution-agreement.html

[61] Google developer, "On-device protections", retrieved from https://developers.google.com/android/play-protect/client-protections

[62] Z. Muhammad, Z. Anwar, A. R. Javed, B. Saleem, S. Abbas, and T. R. Gadekallu, "Smartphone security and privacy: A survey on apts, sensor-based attacks, side-channel attacks, google play attacks, and defenses", Technologies, vol. 11, no. 3, 2023, DOI 10.3390/technologies11030076

[63] S. Mirza, H. Abbas, W. B. Shahid, N. Shafqat, M. Fugini, Z. Iqbal, and Z. Muhammad, "A malware evasion technique for auditing android anti-malware solutions", 2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2021, pp. 125–130, DOI 10.1109/WETICE53228.2021.00034

[64] Apple Inc., "About App Store security", May 2024, retrieved from https://support.apple.com/guide/security/about-app-store-security-secb8f887a15/web

[65] Apple Inc., "App code signing process", February 2021, retrieved from https://support.apple.com/guide/security/app-code-signing-process-sec7c917bf14/web

[66] Apple Inc., "Identity Verification", retrieved from https://developer.apple.com/support/identity-verification/

[67] S. Garg and N. Baliyan, "Android security assessment: A review, taxonomy and research gap study", Computers & Security, vol. 100, 2021, p. 102087, DOI 10.1016/j.cose.2020.102087

[68] P. Bhat and K. Dutta, "A survey on various threats and current state of security in android platform", ACM Comput. Surv., vol. 52, feb 2019, DOI 10.1145/3301285

[69] D. Fisher, "Rowhammer, android and the future of hardware attacks", 2018, retrieved from https://duo.com/decipher/rowhammer-android-and-the-future-of-hardware-attacks

[70] S. Khandelwal, "Glitch: New 'rowhammer' attack can remotely hijack android phones", 2018, retrieved from https://thehackernews.com/2018/05/rowhammer-android-hacking.html

[71] Red Hat, Inc., "Use after free vulnerability in linux kernel keychain management", 2016, retrieved from https://access.redhat.com/articles/2131021

[72] Red Hat, Inc., "Rhsa-2009:0427 - security advisory", 2009, retrieved from https://access.redhat.com/errata/RHSA-2009:0427

[73] Daniel R. Thomas, "Rageagainstthecage adb", 2013, retrieved from https://androidvulnerabilities.org/vulnerabilities/RageAgainstTheCage_adb

[74] The ZER0-DAY tracking project, "Zero-day vulnerability in google android", 2013, retrieved from https://www.zero-day.cz/database/253/

[75] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-oriented programming: Systems, languages, and applications", ACM Trans. Inf. Syst. Secur., vol. 15, mar 2012, DOI 10.1145/2133375.2133377

[76] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "StackGuard: Automatic adaptive detection and prevention of Buffer-Overflow attacks", 7th USENIX Security Symposium (USENIX Security 98), San Antonio, TX, January 1998. https://www.usenix.org/conference/7th-usenix-security-symposium/stackguard-automatic-adaptive-detection-and-prevention

[77] V. Parikh and P. Mateti, "Aslr and rop attack mitigations for arm-based android devices", Security in Computing and Communications (S. M. Thampi, G. Martínez Pérez, C. B. Westphall, J. Hu, C. I. Fan, and F. Gómez Mármol, eds.), Singapore, 2017, pp. 350–363, DOI 10.1007/978-981-10-6898-0_29

[78] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, "Jump-oriented programming: a new class of code-reuse attack", Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, New York, NY, USA, 2011, p. 30–40, DOI 10.1145/1966913.1966919

[79] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications", ACM Trans. Inf. Syst. Secur., vol. 13, nov 2009, DOI 10.1145/1609956.1609960

[80] M. Castro, M. Costa, and T. Harris, "Securing software by enforcing data-flow integrity", Proceedings of the 7th Symposium on Operating Systems Design and Implementation, USA, 2006, p. 147–160. https://dl.acm.org/doi/10.5555/1298455.1298470

[81] V. Kiriansky, D. Bruening, and S. P. Amarasinghe, "Secure execution via program shepherding", Proceedings of the 11th USENIX Security Symposium, USA, 2002, p. 191–206. https://dl.acm.org/doi/10.5555/647253.720293

[82] M. Payer and T. R. Gross, "String oriented programming: when aslr is not enough", Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop, New York, NY, USA, 2013, DOI 10.1145/2430553.2430555

[83] OSIRIS Lab & CTFd, "GOT", January 2024, retrieved from https://ctf101.org/binary-exploitation/what-is-the-got/

[84] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazières, and D. Boneh, "Hacking blind", 2014 IEEE Symposium on Security and Privacy, 2014, pp. 227–242, DOI 10.1109/SP.2014.22

[85] H. Shacham, "The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)", Proceedings of the 14th ACM Conference on Computer and Communications Security, New York, NY, USA, 2007, p. 552–561, DOI 10.1145/1315245.1315313

[86] N. Redini, A. Machiry, D. Das, Y. Fratantonio, A. Bianchi, E. Gustafson, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "BootStomp: On the security of bootloaders in mobile devices", 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, August 2017, pp. 781–798. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/redini

[87] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in wpa2", Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 2017, p. 1313–1328, DOI 10.1145/3133956.3134027

[88] Iain Thomson, "'Invisible Man' malware runs keylogger on your Android banking apps", 2017, retrieved from https://www.theregister.co.uk/2017/08/02/banking_android_malware_in_uk

[89] Google LLC, "Android security bulletin", March 2017, retrieved from https://source.android.com/docs/security/bulletin/2017-03-01#rce-in-openssl-&-boringssl

[90] N. Hardy, "The confused deputy: (or why capabilities might have been invented)", SIGOPS Oper. Syst. Rev., vol. 22, oct 1988, p. 36–38, DOI 10.1145/54289.871709

[91] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones", Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011, 2011. https://www.ndss-symposium.org/ndss2011/soundcomber-stealthy-and-context-aware-sound-trojan-smartphones/

[92] CVEdetails, "Apple iPhone OS", retrieved from https://www.cvedetails.com/product/15556/Apple-Iphone-Os.html?vendor_id=49 (2024). Accessed on 26 jul 2024

[93] Forensic Focus, "Jailbreaking iOS 11 And All Versions Of iOS 10", March 2018, retrieved from https://www.forensicfocus.com/articles/jailbreaking-ios-11-and-all-versions-of-ios-10/

[94] Electronic Frontier Foundation, "Unintended Consequences: Fifteen Years under the DMCA", March 2013, retrieved from https://www.eff.org/id/pages/unintended-consequences-fifteen-years-under-dmca

[95] A. Ali, N. Cahyani, and E. Jadied, "Digital Forensic Analysis on iDevice : Jailbreak iOS 12.1.1 as a Case Study", Indonesia Journal on Computing (Indo-JC), vol. 4, September 2019, pp. 205–218, DOI 10.34818/INDOJC.2019.4.2.349

[96] iRemove Tools, "Unc0ver Jailbreak Guide", retrieved from https://iremove.tools/unc0ver-jailbreak-guide

[97] TweakBox, "Redsn0w Jailbreak", retrieved from https://tweak-box.com/redsn0w/

[98] M. R. Laayu, A. Kurniawan, N. D. W. Cahyani, and G. B. Satrya, "Comparison of acquisition results on iphone 7 plus (ios 14.8.1) between jailbreaking vs non-jailbreaking device", 2022 10th International Conference on Information and Communication Technology (ICoICT), 2022, pp. 402–406, DOI 10.1109/ICoICT55009.2022.9914862

[99] NIST, "CVE-2014-4451", retrieved from https://nvd.nist.gov/vuln/detail/cve-2014-4451

[100] Graham Cluley, "CVE-2014-4451", March 2015, retrieved from https://www.intego.com/mac-security-blog/iphone-pin-pass-code/

[101] Forensic Focus, "Forensic Implications of iOS Lockdown (Pairing) Records", November 2016, retrieved from https://www.forensicfocus.com/articles/forensic-implications-of-ios-lockdown-pairing-records/

[102] Kim Jong Cracks, "checkra1n", retrieved from https://checkra.in/

[103] mayedamakoto - GitHub, "Leading Jailbreak Solutions for iOS 17 - iOS 17.5.1", June 2024, retrieved from https://github.com/mayedamakoto/ios-17-5-jailbreak

[104] GetRevanced - GitHub, "Fugu17", November 2023, retrieved from https://github.com/GetRevanced/Fugu17

[105] GeoSn0w - iDevice Central, "iOS 17 – 17.4.1 JAILBREAK: The Most Complete Guide", November 2023, retrieved from https://idevicecentral.com/jailbreak-guide/how-to-jailbreak-ios-17/

[106] GeoSn0w - iDevice Central, "iOS 17.0 – 17.1 Jailbreak News: New WebKit / Safari Vulnerability Released!", October 2023, retrieved from https://idevicecentral.com/jailbreak-news/new-ios-17-webkit-exploit-released-ios-17-jailbreak/

[107] Brandon Azad - Google Project Zero, "A survey of recent iOS kernel exploits", June 2020, retrieved from https://googleprojectzero.blogspot.com/2020/06/a-survey-of-recent-ios-kernel-exploits.html

[108] Samuel Groß - Google Project Zero, "Remote iPhone Exploitation Part 1: Poking Memory via iMessage and CVE-2019-8641", January 2020, retrieved from https://googleprojectzero.blogspot.com/2020/01/remote-iphone-exploitation-part-1.html

[109] Samuel Groß - Google Project Zero, "Fuzzing ImageIO", April 2020, retrieved from https://googleprojectzero.blogspot.com/2020/04/fuzzing-imageio.html

[110] Ian Beer - Project Zero, "An analysis of an in-the-wild iOS Safari WebContent to GPU Process exploit", October 2023, retrieved from https://googleprojectzero.blogspot.com/2023/10/an-analysis-of-an-in-the-wild-ios-safari-sandbox-escape.html

[111] Boris Larino - Kaspersky, "Operation Triangulation: The last (hardware) mystery", December 2023, retrieved from https://securelist.com/operation-triangulation-the-last-hardware-mystery/111669/

[112] Jessica Lyons - The Register, "Apple squashes kernel bug used by TriangleDB spyware", June 2023, retrieved from https://www.theregister.com/2023/06/21/apple_patches_triangledb_spyware/

[113] NIST, "CVE-2023-41990", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2023-41990

[114] NIST, "CVE-2023-32434", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2023-32434

[115] NIST, "CVE-2023-38606", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2023-38606

[116] NIST, "CVE-2023-32435", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2023-32435

[117] Chris Williams - The Register, "Apple squashes security bugs after iPhone flaws exploited by Predator spyware", September 2023, retrieved from https://www.theregister.com/2023/09/22/apple_emergency_patches/

[118] Maddie Stone - Google Threat Analysis Group (TAG), "0-days exploited by commercial surveillance vendor in Egypt", September 2023, retrieved from https://blog.google/threat-analysis-group/0-days-exploited-by-commercial-surveillance-vendor-in-egypt/

[119] B. Marczak, J. Scott-Railton, D. Roethlisberger, B. A. Razzak, S. Anstis, and R. Deibert - The Citiziens Lab, "PREDATOR IN THE WIRES", September 2023, retrieved from https://citizenlab.ca/2023/09/predator-in-the-wires-ahmed-eltantawy-targeted-with-predator-spyware-after-announcing-presidential-ambitions/

[120] NIST, "CVE-2023-41993", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2023-41993

[121] NIST, "CVE-2023-41991", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2023-41991

[122] NIST, "CVE-2023-41992", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2023-41992

[123] The Citiziens Lab, "BLASTPASS: NSO Group iPhone Zero-Click, Zero-Day Exploit Captured in the Wild", September 2023, retrieved from https://citizenlab.ca/2023/09/blastpass-nso-group-iphone-zero-click-zero-day-exploit-captured-in-the-wild/

[124] NIST, "CVE-2023-41061", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2023-41061

[125] NIST, "CVE-2023-41064", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2023-41064

[126] NIST, "CVE-2021-30860", retrieved from https://nvd.nist.gov/vuln/detail/CVE-2021-30860

[127] I. Beer and S. Groß - Google Project Zero, "A deep dive into an NSO zero-click iMessage exploit: Remote Code Execution", December 2021, retrieved from https:

//googleprojectzero.blogspot.com/2021/12/a-deep-dive-into-nso-zero-click.html

[128] K. Mustafa Kareem, "A comprehensive analysis of pegasus spyware and its implications for digital privacy and security", International Journal of Intelligent Systems and Applications in Engineering, vol. 12, March 2024, p. 1360–1373. retrieved from https://ijisae.org/index.php/IJISAE/article/view/5527

[129] Z. Deng, B. Saltaformaggio, X. Zhang, and D. Xu, "iRiS: Vetting Private API Abuse in iOS Applications", Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 2015, p. 44–56, DOI 10.1145/2810103.2813675

[130] T. Wang, K. Lu, L. Lu, S. Chung, and W. Lee, "Jekyll on iOS: When benign apps become evil", 22nd USENIX Security Symposium (USENIX Security 13), Washington, D.C., August 2013, pp. 559–572. Retrieved from https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/wang_tielei

[131] S. Hutchinson, B. Zhou, and U. Karabiyik, "Are we really protected? an investigation into the play protect service", 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 4997–5004, DOI 10.1109/BigData47090.2019.9006100

[132] Telsy, "REMCOS and Agent Tesla loaded into memory with Rezer0 loader", September 2021, retrieved from https://www.telsy.com/en/remcos-and-agent-tesla-loaded-into-memory-with-rezer0-loader/

[133] Telsy, "Spyware: a growing global threat", May 2024, retrieved from https://www.telsy.com/en/spyware-a-growing-global-threat/

[134] G. Kucherin, L. Bezvershenko, I. Kuznetsov, "Dissecting TriangleDB, a Triangulation spyware implant", June 2023, retrieved from https://securelist.com/triangledb-triangulation-implant/110050/

[135] Shad Sluiter, "Create a malware keylogger with JavaScript cross site scripting XSS attack", August 2023, retrieved from https://www.youtube.com/watch?v=w8pwVx_kIig

[136] S. Shevchenko, ""Double agent": a MacOS bundleware installer that acts like a spy", March 2020, retrieved from https://news.sophos.com/en-us/2020/03/17/double-agent-a-macos-bundleware-installer-that-acts-like-a-spy/

[137] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, and J. Teng, "Stealthy video capturer: a new video-based spyware in 3g smartphones", Proceedings of the Second ACM Conference on Wireless Network Security, New York, NY, USA, 2009, p. 69–78, DOI 10.1145/1514274.1514285

[138] N. Xu, W. Jia, Y. Luo, F. Zhang, D. Xuan, and J. Teng, "An opened eye on you", IEEE Vehicular Technology Magazine, vol. 6, no. 4, 2011, pp. 49–59, DOI 10.1109/MVT.2011.942809

[139] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones.", Network and Distributed Systems Symposium, 2011, pp. 17–33. retrieved from https://homes.luddy.indiana.edu/xw7/papers/schlegel2011soundcomber.pdf

[140] J. Soo, "SpyNote Android Trojan Builder Leaked", July 2016, retrieved from https://unit42.paloaltonetworks.com/unit42-spynote-android-trojan-builder-leaked/

[141] Norton, "Remove Fake pop-up or tech support scam messages that warn the computer is infected", June 2023, retrieved from https://support.norton.com/sp/en/en/home/current/solutions/v122764455

[142] Android developer, "READ_CALL_LOG", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#READ_CALL_LOG

[143] Android developer, "READ_CONTACTS", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#READ_CONTACTS

[144] Android developer, "CAMERA", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#CAMERA

[145] Android developer, "RECORD_AUDIO", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#RECORD_AUDIO

[146] Android developer, "ACCESS_BACKGROUND_LOCATION", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#ACCESS_BACKGROUND_LOCATION

[147] Android developer, "ACCESS_COARSE_LOCATION", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#ACCESS_COARSE_LOCATION
[148] Android developer, "ACCESS_FINE_LOCATION", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#ACCESS_FINE_LOCATION
[149] Android developer, "ACCESS_NETWORK_STATE", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#ACCESS_NETWORK_STATE
[150] Android developer, "ACCESS_WIFI_STATE", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#ACCESS_WIFI_STATE
[151] Android developer, "INTERNET", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#INTERNET
[152] Android developer, "NEARBY_WIFI_DEVICES", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#NEARBY_WIFI_DEVICES
[153] Android developer, "FOREGROUND_SERVICE", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#FOREGROUND_SERVICE
[154] Android developer, "RECEIVE_BOOT_COMPLETE", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#RECEIVE_BOOT_COMPLETED
[155] Android developer, "REQUEST_IGNORE_BATTERY_OPTIMIZATIONS", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
[156] Android developer, "READ_SMS", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#READ_SMS
[157] Android developer, "RECEIVE_SMS", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#RECEIVE_SMS
[158] Android developer, "READ_EXTERNAL_STORAGE", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#READ_EXTERNAL_STORAGE
[159] Android developer, "READ_MEDIA_AUDIO", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#READ_MEDIA_AUDIO
[160] Android developer, "READ_MEDIA_IMAGES", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#READ_MEDIA_IMAGES
[161] Android developer, "READ_MEDIA_VIDEO", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#READ_MEDIA_VIDEO
[162] Android developer, "BIND_ACCESSIBILITY_SERVICE", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#BIND_ACCESSIBILITY_SERVICE
[163] Android developer, "GET_ACCOUNTS", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#GET_ACCOUNTS
[164] Android developer, "READ_CALENDAR", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#READ_CALENDAR
[165] Android developer, "READ_PHONE_STATE", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#READ_PHONE_STATE
[166] Android developer, "REQUEST_INSTALL_PACKAGES", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#REQUEST_INSTALL_PACKAGES

[167] d*classified, "Android RATs - The Sneaky Malware Rodent", February 2024, retrieved from https://medium.com/d-classified/android-rats-the-sneaky-malware-rodent-4a3272a790ef
[168] Android developer, "Schedule tasks with WorkManager", January 2024, retrieved from https://developer.android.com/topic/libraries/architecture/workmanager
[169] C. M. Mwange and E. C. Cankaya, "Android trojan horse spyware attack: A practical implementation", 2024 12th International Symposium on Digital Forensics and Security (ISDFS), 2024, pp. 1–5, DOI 10.1109/ISDFS60797.2024.10527296
[170] H. M. Salih and M. S. Mohammed, "Spyware injection in android using fake application", 2020 International Conference on Computer Science and Software Engineering (CSASE), 2020, pp. 100–105, DOI 10.1109/CSASE48920.2020.9142101
[171] Amit Tambe, "Take a note of SpyNote!", October 2023, retrieved from https://blog.f-secure.com/take-a-note-of-spynote/
[172] E. Liu, S. Rao, S. Havron, G. Ho, S. Savage, G. Voelker, and D. McCoy, "No privacy among spies: Assessing the functionality and insecurity of consumer android spyware apps", Proceedings on Privacy Enhancing Technologies, vol. 2023, 2023, pp. 207–224, DOI 10.56553/popets-2023-0013
[173] Android developer, "ImageReader", April 2024, retrieved from https://developer.android.com/reference/android/media/ImageReader
[174] Android developer, "SurfaceTexture", April 2024, retrieved from https://developer.android.com/reference/android/graphics/SurfaceTexture
[175] Last Click Media LLC, "Spy24 Monitoring App", 2024, retrieved from https://spy24.io/
[176] Android developer, "WebView", May 2024, retrieved from https://developer.android.com/reference/android/webkit/WebView
[177] Android developer, "VOICE_CALL", April 2024, retrieved from https://developer.android.com/reference/android/media/MediaRecorder.AudioSource#VOICE_CALL
[178] Android developer, "setAudioSource", April 2024, retrieved from https://developer.android.com/reference/android/media/MediaRecorder#setAudioSource(int)
[179] Android developer, "CAPTURE_AUDIO_OUTPUT", May 2024, retrieved from https://developer.android.com/reference/android/Manifest.permission#CAPTURE_AUDIO_OUTPUT
[180] Android developer, "MIC", April 2024, retrieved from https://developer.android.com/reference/android/media/MediaRecorder.AudioSource#MIC
[181] Android developer, "AudioRecord", April 2024, retrieved from https://developer.android.com/reference/android/media/AudioRecord#AudioRecord(int,%20int,%20int,%20int,%20int)
[182] Android developer, "Sharing audio input (Android 10)", May 2024, retrieved from https://developer.android.com/media/platform/sharing-audio-input#voice_call_ordinary_app
[183] Android developer, "Sharing audio input (Android 11)", May 2024, retrieved from https://developer.android.com/media/platform/sharing-audio-input#concurrent-capture-r
[184] Android developer, "What is API level?", May 2024, retrieved from https://developer.android.com/guide/topics/manifest/uses-sdk-element#ApiLevels
[185] Android developer, "AccessibilityNodeInfo", May 2024, retrieved from https://developer.android.com/reference/android/view/accessibility/AccessibilityNodeInfo
[186] Android developer, "Retrieving window content", May 2024, retrieved from https://developer.android.com/reference/android/accessibilityservice/AccessibilityService#retrieving-window-content
[187] Android developer, "TYPE_NOTIFICATION_STATE_CHANGED", April 2024, retrieved from https://developer.android.com/reference/android/view/accessibility/AccessibilityEvent#TYPE_NOTIFICATION_STATE_CHANGED
[188] Android developer, "Notification", May 2024, retrieved from https://developer.android.com/reference/android/app/Notification

[189] Android developer, "Notification EXTRA_TEXT", May 2024, retrieved from https://developer.android.com/reference/android/app/Notification#EXTRA_TEXT
[190] StackOverflow, "How to read Notifications using Accessibility Services", June 2021, retrieved from https://stackoverflow.com/questions/64415650/how-to-read-notifications-using-accessibility-services
[191] Android developer, "AccessibilityEvent", April 2024, retrieved from https://developer.android.com/reference/android/view/accessibility/AccessibilityEvent
[192] Android developer, "TYPE_VIEW_TEXT_CHANGED", April 2024, retrieved from https://developer.android.com/reference/android/view/accessibility/AccessibilityEvent#TYPE_VIEW_TEXT_CHANGED
[193] Android developer, "EditText", May 2024, retrieved from https://developer.android.com/reference/android/widget/EditText
[194] Android developer, "getText()", April 2024, retrieved from https://developer.android.com/reference/android/view/accessibility/AccessibilityRecord#getText()
[195] Android developer, "android:importantForAccessibility", May 2024, retrieved from https://developer.android.com/reference/android/view/View#attr_android:importantForAccessibility
[196] Android developer, "MediaProjectiony", April 2024, retrieved from https://developer.android.com/reference/android/media/projection/MediaProjection
[197] StackOverflow, "Android MediaProjectionManager in Service", May 2020, retrieved from https://stackoverflow.com/questions/32381455/android-mediaprojectionmanager-in-service#32849121
[198] StackOverflow, "How do I get MediaProjectionManager without disturbing the current foreground process, except to ask for permission?", May 2021, retrieved from https://stackoverflow.com/questions/33398211/how-do-i-get-mediaprojectionmanager-without-disturbing-the-current-foreground-pr
[199] Android developer, "When apps can start activities", May 2024, retrieved from https://developer.android.com/guide/components/activities/background-starts#exceptions
[200] Android developer, "takeScreenshot()", May 2024, retrieved from https://developer.android.com/reference/android/accessibilityservice/AccessibilityService#takeScreenshot(int , %20java.util.concurrent.Executor , %20android.accessibilityservice.AccessibilityService.TakeScreenshotCallback)
[201] Android developer, "ContentResolver", April 2024, retrieved from https://developer.android.com/reference/android/content/ContentResolver
[202] Stevdza-San, "Complete Guide on Reading SMS (Inbox & Sent) in Android - Easy Tutorial", July 2023, retrieved from https://www.youtube.com/watch?v=9fIiQ9YQ7BI
[203] StackOverflow, "How can I read SMS messages from the device programmatically in Android?", December 2022, retrieved from https://stackoverflow.com/questions/848728/how-can-i-read-sms-messages-from-the-device-programmatically-in-android
[204] Android developer, "Retrieve a list of contacts", January 2024, retrieved from https://developer.android.com/training/contacts-provider/retrieve-names
[205] Android developer, "Access media files from shared storage", May 2024, retrieved from https://developer.android.com/training/data-storage/shared/media
[206] Google, "Firebase Cloud Messaging", May 2024, retrieved from https://firebase.google.com/docs/cloud-messaging
[207] Y. Zeng, K. G. Shin, and X. Hu, "Design of sms commanded-and-controlled and p2p-structured mobile botnets", Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks, New York, NY, USA, 2012, p. 137–148, DOI 10.1145/2185448.2185467
[208] R. Saeki, L. Kitayama, J. Koga, M. Shimizu, and K. Oida, "Smishing strategy dynamics and evolving botnet activities in japan", IEEE Access, vol. 10, 2022, pp. 114869–114884, DOI 10.1109/ACCESS.2022.3217795
[209] mSpy, "The Best Phone Tracker for Parental Control", retrieved from https://www.mspy.com/

[210] SpyX, "The best phone monitoring software for parental control", retrieved from https://spyx.com/
[211] Hoverwatch, "Free mobile tracker", 2024, retrieved from https://www.hoverwatch.com/
[212] Hoverwatch, "How to Install Hoverwatch for Android", April 2024, retrieved from https://support.hoverwatch.com/hc/en-us/articles/207595389-How-to-Install-Hoverwatch-for-Android
[213] Hoverwatch, "Additional installation instructions", retrieved from https://support.hoverwatch.com/hc/en-us/sections/360003448453-Additional-installation-instructions
[214] Hoverwatch, "How to Uninstall Hoverwatch for Android", September 2023, retrieved from https://support.hoverwatch.com/hc/en-us/articles/207610149-How-to-Uninstall-Hoverwatch-for-Android
[215] Hoverwatch, "How to open a hidden Hoverwatch program on Android phone", April 2024, retrieved from https://support.hoverwatch.com/hc/en-us/articles/209298845-How-to-open-a-hidden-Hoverwatch-program-on-Android-phone
[216] MobileTracker free, "The best monitoring solution", retrieved from https://mobile-tracker-free.com/
[217] MobileTracker free, "Restrictions/Limitations Android 14", retrieved from https://support.mobile-tracker-free.com/hc/en-us/articles/9435164993052-Restrictions-Limitations-Android-14
[218] MobileTracker free, "Installation guide", retrieved from https://mobile-tracker-free.com/help/
[219] MobileTracker free, "Samsung Android 14.x Guide", retrieved from https://support.mobile-tracker-free.com/hc/it/articles/11800922959516-Samsung-Android-14-x-Guide
[220] MobileTracker free, "The best way to monitor your phone or tablet.", retrieved from https://mobile-tracker-free.com/features/
[221] MobileTracker free, "Offers", retrieved from https://mobile-tracker-free.com/offers/
[222] MobileTracker free, "How to uninstall Mobile Tracker Free?", retrieved from https://support.mobile-tracker-free.com/hc/en-us/articles/360003110633-How-to-uninstall-Mobile-Tracker-Free
[223] iKeyMonitor, "Ultimate Parental Control App for Android, iPhone", retrieved from https://ikeymonitor.com/
[224] iKeyMonitor, "iKeyMonitor Compatibility", retrieved from https://ikeymonitor.com/compatibility
[225] iKeyMonitor, "iKeyMonitor Android Installation Guide", retrieved from https://users.ikeymonitor.com/knowledge-base/ikeymonitor-android-installation-guide
[226] iKeyMonitor, "Samsung Android 14.x Guide", retrieved from https://users.ikeymonitor.com/knowledge-base/samsung-android-14-guide
[227] iKeyMonitor, "Plans", retrieved from https://ikeymonitor.com/purchase
[228] iKeyMonitor, "Features", retrieved from https://ikeymonitor.com/features
[229] iKeyMonitor, "Uninstall/Remove iKeyMonitor for Android", retrieved from https://support.ikeymonitor.com/hc/en-us/articles/115006327888-Uninstall-Remove-iKeyMonitor-for-Android
[230] iKeyMonitor, "How to hide iKeyMonitor?", retrieved from https://community.ikeymonitor.com/t/how-to-hide-ikeymonitor/55
[231] Hammer, "Shield Your Phone, Prevent Theft", retrieved from https://hammer-security.ca/
[232] LSDroid, "Cerberus Phone Security (Antitheft)", retrieved from https://www.cerberusapp.com/home/en
[233] LSDroid, "FAQ: What happened to Cerberus app? Why is Cerberus Anti-theft not available on Google Play Store?", retrieved from https://www.cerberusapp.com/help/en/28
[234] LSDroid, "FAQ: Changelog", retrieved from https://www.cerberusapp.com/help/en/15
[235] LSDroid, "FAQ: I'm not able to uninstall Cerberus", retrieved from https://www.cerberusapp.com/help/en/10

[236] FlexiSPY, "The World's Most Powerful Monitoring Software for Computers, Mobile Phones and Tablets", retrieved from https://www.flexispy.com/
[237] FlexiSPY, "Android OS 14 Limitations", retrieved from https://support.flexispy.com/portal/en/kb/articles/android-os-14-limitations#Normal_Mode_Limitations
[238] FlexiSPY, "FlexiSPY Android (Normal) Version History", retrieved from https://support.flexispy.com/portal/en/kb/articles/flexispy-android-normal-version-history
[239] FlexiSPY, "Android Device Limitations", retrieved from https://support.flexispy.com/portal/en/kb/articles/android-device-limitations#Limitations_for_All_Devices
[240] FlexiSPY, "Want An Easier Way To Install FlexiSPY?", retrieved from https://www.flexispy.com/en/flexispy-remote-installation-service.htm
[241] FlexiSPY, "Check out all our great monitoring features!", retrieved from https://www.flexispy.com/en/features-overview.htm
[242] FlexiSPY, "Remotely uninstall FlexiSPY from a device", retrieved from https://www.flexispy.com/en/features/uninstall-flexispy-remotely.htm
[243] SpyDrill, "How To Uninstall FlexiSPY On Android", 2022, retrieved from https://www.youtube.com/watch?v=WUP5TMtzdnE
[244] Bark, "Bark FAQs", retrieved from https://www.bark.us/#:~:text=11-,FAQs,-How%20much%20does
[245] Key24, "Key24 Pricing", retrieved from https://key24.app/faqs/
[246] MMGuardian, "MMGuardian Pricing", retrieved from https://www.mmguardian.com/pricing
[247] TiSpy, "TiSpy Pricing", retrieved from https://tispy.net/pricing/
[248] LSDroid, "Cerberus License Pricing", retrieved from https://www.cerberusapp.com/pricing/en
[249] Hoverwatch, "Hoverwatch Pricing", retrieved from https://www.hoverwatch.com/pricing
[250] Meuspy, "Meuspy Pricing", retrieved from https://meuspy.com/#pricing
[251] Spapp, "Spapp Pricing", retrieved from https://www.spappmonitoring.com/spy/promo_billing
[252] SpyHuman, "SpyHuman Pricing", retrieved from https://spyhuman.com/pages/price-list.html
[253] Eyezy, "Eyezy Pricing", retrieved from https://www.eyezy.com/step-3-choose-plan
[254] FlexiSPY, "FlexiSPY Plans", retrieved from https://www.flexispy.com/en/buy-flexispy.htm
[255] KidsGuard Pro, "KidsGuard Pro Pricing", retrieved from https://www.clevguard.org/kidsguard-pro-pricing/
[256] MobileSpy, "MobileSpy Pricing and Plans", retrieved from https://mobilespy.io/pricing-plans/
[257] MobiStealth, "MobiStealth Pricing", retrieved from https://www.mobistealth.com/products.php
[258] mSpy, "mSpy plans", retrieved from https://www.mspy.com/blog/mspy-extreme-vs-premium/#what-subscription-plans-does-mspy-offer
[259] Spyera, "Spyera Pricing", retrieved from https://spyera.com/pricing/
[260] Spyic, "Spyic Pricing", retrieved from https://spyic.com/pricing.html
[261] Inga Valiaugaite, "SpyX Review 2024", July 2024, retrieved from https://www.techopedia.com/spy/spyx-review
[262] Spyzie, "Spyzie Pricing", retrieved from https://spyzie.io/pricing.html
[263] TheOneSpy, "TheOneSpy Pricing", retrieved from https://securepay.theonespy.com/buy-now.php
[264] uMobix, "uMobix Plans", retrieved from https://umobix.com/prices.html?platform=android&split_group=umobix_mobile_price_page&split_variant=prices
[265] Xnspy, "Xnspy Pricing", retrieved from https://xnspy.com/buy-now.html
[266] TheWiSpy, "TheWiSpy Pricing", retrieved from https://www.thewispy.com/pricing/

[267] Qustodio, "Qustodio Pricing", retrieved from https://www.qustodio.com/en/premium/
[268] Android Developer, "<uses-sdk>", 6 2024, retrieved from https://developer.android.com/guide/topics/manifest/uses-sdk-element
[269] StackOverflow, "Reviews are not displayed in Google Play", June 2023, retrieved from https://stackoverflow.com/questions/70592386/reviews-are-not-displayed-in-google-play
[270] Google Play, "Secure VPN - Safer Internet", retrieved from https://play.google.com/store/apps/details?id=com.fast.free.unblock.secure.vpn&hl=en
[271] VirusTotal, "Bark Parents SDK versions", retrieved from https://www.virustotal.com/gui/file/e1056f179d9c2aa57ad9f925fff292b6f242a0be2f9ed5b509956f9bd652832f/details
[272] Key24, "Key24 Compatibility", retrieved from https://key24.app/compatibility/
[273] TiSpy, "TiSpy Android Installation Guide", retrieved from https://tispy.net/install-guide/
[274] VirusTotal, "Cerberus Standard SDK versions", retrieved from https://www.virustotal.com/gui/file/05ad6752683b7d59488f45d1ea32945ae4e48e1f63819e850f2002d11efeec28/details
[275] VirusTotal, "Cerberus Disguised SDK versions", retrieved from https://www.virustotal.com/gui/file/d8212ea364999eb28ba71ed1d65cde089933c6a949498896c0d19ef53fb0dbb8/details
[276] VirusTotal, "Hammer Security SDK versions", retrieved from https://www.virustotal.com/gui/file/860bdb15107bcbf3f1da2056c161e63ef772b2298cf252174c3f355d5572f44f/details
[277] VirusTotal, "Hoverwatch SDK versions", retrieved from https://www.virustotal.com/gui/file/49eb2fb7f5968909c06d75d71cf8be0849038e213fddbc1e899c835367c63b50/details
[278] VirusTotal, "Meuspy SDK versions", retrieved from https://www.virustotal.com/gui/file/aefde2e59094eab0228cb6a2e562c76dd283e0969332ce1a72c702aa0198030f/details
[279] VirusTotal, "MobileTracker free SDK versions", retrieved from https://www.virustotal.com/gui/file/2de500618c8ba340dbbe615cef8df65d9864e8b6bee7fa78c16421a38b1bd185/details
[280] VirusTotal, "Spapp SDK versions", retrieved from https://www.virustotal.com/gui/file/0b70317cbb1e0149108b78b2da741f0dc1283c0c55a4e83e3601e4feb8147b5b/details
[281] SpyHuman, "SpyHuman FAQ", retrieved from https://spyhuman.com/pages/faq.html
[282] FlexiSPY, "FlexiSPY Compatibility", retrieved from https://www.flexispy.com/en/compatibility.htm
[283] SpyHuman, "KidsGuard Pro Compatibility", retrieved from https://www.clevguard.org/phone-monitoring/#:~:text=to%20more%20people.-,Compatibility,-KidsGuard%20Pro%20for
[284] MobileSpy, "MobileSpy FAQ", retrieved from https://mobilespy.io/faq/
[285] mSpy, "mSpy Compatibility Policy", retrieved from https://www.mspy.com/compatibility-policy.html
[286] Spyic, "Spyic FAQ", retrieved from https://spyic.com/faq.html
[287] SpyX, "SpyX FAQ", retrieved from https://spyx.com/faq
[288] Arooba Shahzadi, "uMobix Review 2024", March 2024, retrieved from https://www.linkedin.com/pulse/umobix-review-unveiling-features-performance-parental-arooba-shahzadi-jhzgf
[289] TheWiSpy, "TheWiSpy FAQ", retrieved from https://www.thewispy.com/faqs/
[290] VirusTotal, "Eyezy SDK versions", retrieved from https://www.virustotal.com/gui/file/cd58859c01e386b527b022346c7c9331534061c16aec82388b4f715e9d494789/details
[291] VirusTotal, "MMGuardian SDK versions", retrieved from https://www.virustotal.com/gui/file/0c90bd528f364cc43b8ca42d2a725f39522be7e03ba227aa5b2a41e556de0657/details

[292] Xnspy, "Xnspy Compatibility", retrieved from https://xnspy.com/compatibility.html
[293] TheOneSpy, "TheOneSpy Android Compatibility", retrieved from https://www.theonespy.com/android-spy-software/#:~:text=time%20issue%20(Fixed)-,Supported%20OS,-Compatibility%20with%20Android
[294] MobiStealth, "MobiStealth FAQs", retrieved from https://www.mobistealth.com/how-it-works#faqs
[295] Spyera, "Spyera Compatibility", retrieved from https://spyera.com/compatibility/
[296] Spyzie, "Spyzie FAQs", retrieved from https://spyzie.io/faq.html
[297] VirusTotal, "Qustodio SDK versions", retrieved from https://www.virustotal.com/gui/file/21c3cae94737c4dc087fbe0822d011c5a1e5ec33203ca686df0a501660f738a8/details
[298] Google Play, "Bark - Parental Controls", retrieved from https://play.google.com/store/apps/details?id=cm.pt.barkparent&hl=en_US
[299] Google Play, "Cerberus Phone Anti-theft", retrieved from https://play.google.com/store/apps/details?id=com.eyezy.android&hl=en
[300] Google Play, "Eyezy – GPS Location Tracker", retrieved from https://play.google.com/store/apps/details?id=com.ssurebrec&hl=en
[301] Google Play, "Hammer Security: Find my Phone", retrieved from https://play.google.com/store/apps/details?id=com.hammersecurity&hl=en
[302] Google Play, "KidsGuard Pro-Phone Monitoring", retrieved from https://play.google.com/store/apps/details?id=com.clevguard.kidsguardpro.parent&hl=en
[303] Google Play, "MMGuardian Child Phone App", retrieved from https://play.google.com/store/apps/details?id=com.mmguardian.childapp&hl=en
[304] Google Play, "MMGuardian Parental Control", retrieved from https://play.google.com/store/apps/details?id=com.mmguardian.parentapp&hl=en
[305] Google Play, "Kids App Qustodio", retrieved from https://play.google.com/store/apps/details?id=com.qustodio.qustodioapp&hl=en
[306] Google Play, "Qustodio Parental Control App", retrieved from https://play.google.com/store/apps/details?id=com.qustodio.family.parental.control.app.screentime&hl=en
[307] iJailBreakGuide, "Jailbreak Compatibility Check", retrieved from https://ijailbreakguide.com/
[308] App Store, "KidsGuard Pro-Phone Monitoring", Accessed on Aug 7th 2024 at https://apps.apple.com/us/app/kidsguard-pro-phone-monitoring/id6449023572
[309] Apple Developer, "Reset an app summary rating", retrieved from https://developer.apple.com/help/app-store-connect/monitor-ratings-and-reviews/reset-an-app-summary-rating/
[310] mSpy, "mSpy Terms of Use", retrieved from https://www.mspy.com/terms-of-use.html#:~:text=are%20provided%20by-,Altercon%20Group%20s.r.o.,-%2C%20registered%20address%3A%20Londynska
[311] App Store, "Bark - Parental Controls", retrieved from https://apps.apple.com/us/app/bark-parental-controls/id1477619146
[312] App Store, "MMGuardian Parental Control", retrieved from https://apps.apple.com/us/app/mmguardian-parental-control/id951476346
[313] TiSpy, "TiSpy iOS Installation Guide", retrieved from https://tispy.net/install-guide-ios/
[314] App Store, "Eyezy: find my friends tracker", retrieved from https://apps.apple.com/in/app/eyezy-find-my-friends-tracker/id1575693779
[315] MobileSpy, "MobileSpy iOS Compatibility", retrieved from https://mobilespy.io/spy-iphone/#:~:text=Frequently%20Asked%20Question%0AFor%20iOS%20Spy
[316] App Store, "Kids App Qustodio", retrieved from https://apps.apple.com/us/app/kids-app-qustodio/id990229433
[317] Spyic, "Spyic iOS FAQ", retrieved from https://spyic.com/faq-ios-no-jailbreak.html
[318] SpyX, "SpyX iOS FAQs", retrieved from https://spyx.com/iphone-spy#:~:text=FAQs%20about%20spying%20on%20iPhone

[319] Spyzie, "Spyzie iPhone installation", retrieved from https://spyzie.io/how-to-track-someone-iphone-location.html
[320] TheOneSpy, "TheOneSpy iOS Compatibility", retrieved from https://www.theonespy.com/iphone-spy-software/#:~:text=Without%20them%20Knowing%3F-,Supported%20OS,-Compatibility%20with%20iOS
[321] App Store, "Bark Kids", retrieved from https://apps.apple.com/us/app/bark-kids/id1500629600
[322] App Store, "MMGuardian Parent App", retrieved from https://apps.apple.com/us/app/mmguardian-parent-app/id1226900784
[323] App Store, "mSpy: Find my Friends Phone", retrieved from https://apps.apple.com/us/app/mspy-find-my-friends-phone/id1182397829
[324] App Store, "Qustodio Parental Control App", retrieved from https://apps.apple.com/us/app/qustodio-parental-control-app/id1501720596
[325] A. Valiyathara, "Hiding apps on Android devices: A step-by-step guide", June 2024, retrieved from https://www.croma.com/unboxed/how-to-hide-apps-on-android#:~:text=on%20their%20devices.-,Built%2Din%20options%20for%20hiding%20apps,-Many%20Android%20devices
[326] Android developer, "lockNow", May 2024, retrieved from https://developer.android.com/reference/android/app/admin/DevicePolicyManager#lockNow(int)
[327] Android developer, "DevicePolicyManager", May 2024, retrieved from https://developer.android.com/reference/android/app/admin/DevicePolicyManager
[328] Rithvik, "Cerberus Acknowledges Data Breach, States Some Usernames and Encrypted Passwords Stolen", March 2014, retrieved from https://www.droid-life.com/2014/03/26/cerberus-data-breach/
[329] A. Langton, "Stalking stalkerware: A deep dive into flexispy", December 2019, retrieved from https://blogs.juniper.net/en-us/threat-research/stalking-stalkerwarea-deep-dive-into-flexispy-2
[330] Ryne Hager, "Google's new SMS and call permission policy is crippling apps used by millions", January 2019, retrieved from https://www.androidpolice.com/2019/01/05/googles-new-sms-and-call-permission-policy-is-crippling-apps-used-by-millions/
[331] Z. Shan, I. Neamtiu, and R. Samuel, "Self-hiding behavior in android apps: Detection and characterization", 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), 2018, pp. 728–739, DOI 10.1145/3180155.3180214
[332] Android developer, "setComponentEnabledSetting", April 2024, retrieved from https://developer.android.com/reference/android/content/pm/PackageManager#setComponentEnabledSetting(android.content.ComponentName,%20int,%20int)
[333] Android developer, "COMPONENT_ENABLED_STATE_DISABLED", April 2024, retrieved from https://developer.android.com/reference/android/content/pm/PackageManager#COMPONENT_ENABLED_STATE_DISABLED
[334] Android developer, "DONT_KILL_APP", April 2024, retrieved from https://developer.android.com/reference/android/content/pm/PackageManager#DONT_KILL_APP
[335] Android developer, "CATEGORY_LAUNCHER", May 2024, retrieved from https://developer.android.com/reference/android/content/Intent#CATEGORY_LAUNCHER
[336] Android developer, "ACTION_MAIN", May 2024, retrieved from https://developer.android.com/reference/android/content/Intent#ACTION_MAIN
[337] Android developer, "CATEGORY_LEANBACK_LAUNCHER", May 2024, retrieved from https://developer.android.com/reference/android/content/Intent#CATEGORY_LEANBACK_LAUNCHER
[338] Android developer, "Recents screen", January 2024, retrieved from https://developer.android.com/guide/components/activities/recents
[339] Android developer, "excludeFromRecents", May 2024, retrieved from https://developer.android.com/guide/topics/manifest/activity-element#exclude
[340] Android developer, "FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS", May 2024, retrieved from https://developer.android.com/reference/android/content/Intent#FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS

[341] Android developer, "abortBroadcast", April 2024, retrieved from https://developer.android.com/reference/android/content/BroadcastReceiver#abortBroadcast()
[342] Android developer, "ContentResolver.delete", April 2024, retrieved from https://developer.android.com/reference/android/content/ContentResolver#delete(android.net.Uri,%20java.lang.String,%20java.lang.String[])
[343] Android developer, "endCall", May 2024, retrieved from https://developer.android.com/reference/android/telecom/TelecomManager#endCall()
[344] Android developer, "respondToCall", May 2024, retrieved from https://developer.android.com/reference/android/telecom/CallScreeningService#respondToCall(android.telecom.Call.Details,%20android.telecom.CallScreeningService.CallResponse)
[345] Android developer, "CallScreeningService.CallResponse.Builder", April 2024, retrieved from https://developer.android.com/reference/android/telecom/CallScreeningService.CallResponse.Builder
[346] StackOverflow, "I want to get hold of phone number Programmatically as the phone is ringing in android", January 2024, retrieved from https://stackoverflow.com/questions/77711455/i-want-to-get-hold-of-phone-number-programmatically-as-the-phone-is-ringing-in-a/77762680
[347] Android developer, "Runtime", June 2023, retrieved from https://developer.android.com/reference/java/lang/Runtime
[348] Android developer, "logcat", January 2024, retrieved from https://developer.android.com/tools/logcat
[349] Z. Shan, R. Samuel, and I. Neamtiu, "Device administrator use and abuse in android: Detection and characterization", The 25th Annual International Conference on Mobile Computing and Networking, New York, NY, USA, 2019, DOI 10.1145/3300061.3345452
[350] Android developer, "onDestroy", May 2024, retrieved from https://developer.android.com/guide/components/activities/activity-lifecycle#ondestroy
[351] Android developer, "JobScheduler", April 2024, retrieved from https://developer.android.com/reference/android/app/job/JobScheduler
[352] Android developer, "AlarmManager", April 2024, retrieved from https://developer.android.com/reference/android/app/AlarmManager
[353] Pierazzi Fabio, Mezzour Ghita, Han Qian, Colajanni Michele, Subrahmanian V. S., "A data-driven characterization of modern android spyware", ACM Trans. Manage. Inf. Syst., vol. 11, apr 2020, DOI 10.1145/3382158
[354] S. Chowhan and A. K. Saxena, "Advanced techniques in network traffic analysis: Utilizing wireshark for in-depth live data packet inspection and information capture", 2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI), 2023, pp. 843–847, DOI 10.1109/ICCSAI59793.2023.10421631
[355] Liu Yabing, Song Han Hee, Bermudez Ignacio, Mislove Alan, Baldi Mario, Tongaonkar Alok, "Identifying personal information in internet traffic", Proceedings of the 2015 ACM on Conference on Online Social Networks, New York, NY, USA, 2015, p. 59–70, DOI 10.1145/2817946.2817947
[356] S. Holtmanns and I. Oliver, "Sms and one-time-password interception in lte networks", 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–6, DOI 10.1109/ICC.2017.7997246
[357] K. Ullah, I. Rashid, H. Afzal, M. M. W. Iqbal, Y. A. Bangash, and H. Abbas, "Ss7 vulnerabilities—a survey and implementation of machine learning vs rule based filtering for detection of ss7 network attacks", IEEE Communications Surveys & Tutorials, vol. 22, no. 2, 2020, pp. 1337–1371, DOI 10.1109/COMST.2020.2971757
[358] D. Javaheri, M. Hosseinzadeh, and A. M. Rahmani, "Detection and elimination of spyware and ransomware by intercepting kernel-level system routines", IEEE Access, vol. 6, 2018, pp. 78321–78332, DOI 10.1109/ACCESS.2018.2884964
[359] I. Shakhsheer, M. Abdallah, I. Mashaqi, and A. Awad, "Automated forensic analysis following memory content using volatility framework", 2023 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), 2023, pp. 369–374, DOI 10.1109/3ICT60104.2023.10391513

[360] Google Code Archive, "volatility - AndroidMemoryForensics.wiki", retrieved from https://code.google.com/archive/p/volatility/wikis/AndroidMemoryForensics.wiki
[361] LSDroid, "Cerberus Phone Security (Antitheft) downloads", retrieved from https://www.cerberusapp.com/downloads/en
[362] How-To Geek, "How to Install and Use ADB, the Android Debug Bridge Utility", December 2022, retrieved from https://www.howtogeek.com/125769/how-to-install-and-use-abd-the-android-debug-bridge-utility/
[363] Stack Overflow, "How to use ADB Shell when Multiple Devices are connected?", June 2024, retrieved from https://stackoverflow.com/questions/14654718/how-to-use-adb-shell-when-multiple-devices-are-connected-fails-with-error-mor
[364] Android developer, "Broadcasts overview", June 2024, retrieved from https://developer.android.com/develop/background-work/background-tasks/broadcasts
[365] MMGuardian, "Configuration of Apple Screen Time with MMGuardian", retrieved from https://www.mmguardian.com/screen-time-mmguardian
[366] Qustodio, "Qustodio: How do I set up calls & messages monitoring for iOS devices?", retrieved from https://help.qustodio.com/hc/en-us/articles/4986837688465-How-do-I-set-up-calls-messages-monitoring-for-iOS-devices
[367] Qustodio, "What is Qustodio Advance and how do I set it up?", retrieved from https://help.qustodio.com/hc/en-us/articles/20084558366482-What-is-Qustodio-Advance-and-how-do-I-set-it-up
[368] App Store, "Parental Control App - Kidslox", retrieved from https://apps.apple.com/us/app/parental-control-app-kidslox/id914825567
[369] The Hacker News, "Pakistani Hackers Use DISGOMOJI Malware in Indian Government Cyber Attacks", June 2024, retrieved from https://thehackernews.com/2024/06/pakistani-hackers-use-disgomoji-malware.html
[370] The Hacker News, "CapraRAT Spyware Disguised as Popular Apps Threatens Android Users", July 2024, retrieved from https://thehackernews.com/2024/07/caprarat-spyware-disguised-as-popular.html
[371] Independent UK, "My abusive ex-boyfriend secretly stalked me for months with hidden tracking app on my phone", June 2024, retrieved from https://www.independent.co.uk/news/uk/crime/stalking-tech-companies-police-tracking-device-b2557016.html
[372] L'Unione Sarda, "Turetta killed Giulia Cecchettin with 75 stab wounds. A spy app to "control" her", May 2024, retrieved from https://www.unionesarda.it/en/italy/turetta-killed-giulia-cecchettin-with-75-stab-wounds-a-spy-app-to-control-it-dhwzqdli
[373] Spamfighter, "One Fresh Android Spyware Exaspy Attacks Company Executives", November 2016, retrieved from https://www.spamfighter.com/News-20581-One-Fresh-Android-Spyware-Exaspy-Attacks-Company-Executives.htm
[374] The Hacker News, "SpyNote: Beware of This Android Trojan that Records Audio and Phone Calls", October 2023, retrieved from https://thehackernews.com/2023/10/spynote-beware-of-this-android-trojan.html
[375] Cyber Swachhta Kendra, "SecuriDropper", November 2023, retrieved from https://www.csk.gov.in/alerts/SecuriDropper.html
[376] Sophos, "Android APT spyware, targeting Middle East victims, enhances evasiveness", November 2021, retrieved from https://news.sophos.com/en-us/2021/11/23/android-apt-spyware-targeting-middle-east-victims-improves-its-capabilities/
[377] The Citizen Lab, "BlastPass: NSO Group iPhone Zero-Click, Zero-Day Exploit Captured in the Wild", September 2023, retrieved from https://citizenlab.ca/2023/09/blastpass-nso-group-iphone-zero-click-zero-day-exploit-captured-in-the-wild/
[378] The Citizen Lab, "ForcedEntry: NSO Group iMessage Zero-Click Exploit Captured in the Wild", September 2021, retrieved from https://citizenlab.ca/2021/09/forcedentry-nso-group-imessage-zero-click-exploit-captured-in-the-wild/
[379] Google LLC, "About App campaigns", retrieved from https://support.google.com/google-ads/answer/6247380

[380] Google LLC, "Create an App campaign for installs", retrieved from https://support.google.com/google-ads/answer/12575501
[381] Statista, "Most popular Google Play app categories as of 3rd quarter 2022, by share of available apps", December 2023, retrieved from https://www.statista.com/statistics/279286/google-play-android-app-categories/
[382] VirusTotal, "Virustotal home page", retrieved from https://www.virustotal.com/gui/home/upload
[383] Stack Overflow, "Install an apk file from command prompt?", retrieved from https://stackoverflow.com/questions/7076240/install-an-apk-file-from-command-prompt
[384] GitHub, "SpyGuard: Intercept Detect Report", retrieved from https://github.com/SpyGuard
[385] Open Information Security Foundation (OISF), "Suricata", retrieved from https://suricata.io/
[386] GitHub, "Mobile Verification Toolkit (MVT)", retrieved from https://github.com/mvt-project/mvt
[387] OASIS Open, "Introduction to STIX", retrieved from https://oasis-open.github.io/cti-documentation/stix/intro.html
[388] OASIS Open, "STIX Indicator", retrieved from https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html#_muftrcpnf89v
[389] OASIS Open, "STIX Relationship", retrieved from https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html#_e2e1szrqfoan
[390] Android developer, "dumpsys ", July 2023, retrieved from https://developer.android.com/tools/dumpsys
[391] Android developer, "Telecom framework overview", January 2024, retrieved from https://developer.android.com/develop/connectivity/telecom
[392] Android developer, "How to Place Outgoing Calls", June 2024, retrieved from https://developer.android.com/reference/android/telecom/ConnectionService#how-to-place-outgoing-calls
[393] Android developer, "placeCall", June 2024, retrieved from https://developer.android.com/reference/android/telecom/TelecomManager#placeCall(android.net.Uri,%20android.os.Bundle)
[394] Android developer, "FLAG_ACTIVITY_NEW_TASK", July 2024, retrieved from https://developer.android.com/reference/android/content/Intent#FLAG_ACTIVITY_NEW_TASK
[395] libimobiledevice, "libimobiledevice: A cross-platform FOSS library written in C to communicate with iOS devices natively", retrieved from https://libimobiledevice.org/
[396] MVT, "Records extracted by mvt-ios", retrieved from https://docs.mvt.re/en/latest/ios/records/
[397] Android Developer, "LocationManager", April 2024, retrieved from https://developer.android.com/reference/android/location/LocationManager
[398] Apple Developer, "Develop apps for Apple platforms", retrieved from https://developer.apple.com/tutorials/app-dev-training/