BACHELOR'S THESIS

Unauthorized access to mobile data

How can third-party mobile applications be used to access and acquire data?

Bachelor	Applied Computer Science
Degree	Computer & Cyber Crime Professional
Academic year	2020 - 2021
Student	Kyra Van Den Eynde
Internal supervisor	Koen Koreman (Howest)
External promotor	Jonas Bauters & Jean-François Maes (NVISO)

howest.be

BACHELOR'S THESIS

Unauthorized access to mobile data

How can third-party mobile applications be used to access and acquire data?

Bachelor	Applied Computer Science
Degree	Computer & Cyber Crime Professional
Academic year	2020 - 2021
Student	Kyra Van Den Eynde
Internal supervisor	Koen Koreman (Howest)
External promotor	Jonas Bauters & Jean-François Maes (NVISO)

howest.be

Admission to loan

22/06/2021

Foreword

First of all, I would like to thank Jeroen Beckers, manager of mobile security at NVISO, and Koen Koreman, my internal mentor and lecturer of mobile security at Howest, for helping me with this bachelor thesis and my Proof of Concept. They have been a continuous help for me and I learned a lot about Android from them.

Finally, I would also like to thank Wim Van Renterghem, researcher and lecturer at Howest, for guiding me in my research on iOS.

I found this bachelor project very educational and eye-opening, certainly considering the practical part, where I could try out the abilities of accessibility services in Android. It has been a challenge, as I knew very little about mobile security, but that made it only more interesting.

This thesis was written for everyone in IT who has an interest in mobile security.

Please enjoy.

Kyra Van Den Eynde, Bruges, 08/06/2021.

Abstract

This bachelor thesis is about the impact mobile malware can have and how malware might access sensitive data on a mobile device. This thesis describes the different types of malware and compares how Android and iOS handle security threats.

As a theoretical introduction, the term *mobile application* and some key differences between Android and iOS are explained. To end this introduction, this thesis also briefly describes seven different types of mobile malware.

The research part of this thesis focuses on mobile spyware. Firstly, this part describes the types of data that mobile malware might access. Then, it also explains how mobile malware might get on a victim's device and how it might access sensitive data. Then, this research will also explain how real this threat is by giving some examples from the past and explaining the possible impact. Lastly, this research will demonstrate how different Android and iOS handle this threat.

The last part describes the Proof of Concept that was made for NVISO. This Proof of Concept is a simple spyware app that uses Android's accessibility services to steal MFA tokens from MFA apps.

In the end rests the conclusion to the research question, my personal experiences with this project, and of course, some possibilities for further research. As an addendum, there are some appendices.

Keywords: mobile malware – spyware – Android – iOS – accessibility services

Samenvatting

Deze bachelorproef gaat over de impact die mobiele malware kan hebben en hoe malware toegang kan krijgen tot gevoelige gegevens op een mobiel apparaat. Deze scriptie beschrijft de verschillende soorten malware en vergelijkt hoe Android en iOS omgaan met veiligheidsbedreigingen.

Als theoretische inleiding wordt de term mobiele applicatie en enkele belangrijke verschillen tussen Android en iOS toegelicht. Ter afsluiting van deze inleiding worden in deze scriptie ook zeven verschillende soorten mobiele malware kort beschreven.

Het onderzoek van dit proefschrift richt zich op mobiele spyware. Als eerste worden in dit deel de soorten gegevens beschreven waartoe mobiele malware toegang kan hebben. Vervolgens wordt ook uitgelegd hoe mobiele malware op het apparaat van een slachtoffer kan komen en hoe deze toegang kan krijgen tot gevoelige gegevens. Daarnaast zal dit onderzoek ook uitleggen hoe reëel deze dreiging is door enkele voorbeelden uit het verleden te geven en de mogelijke gevolgen uit te leggen. Ten slotte zal dit onderzoek laten zien hoe verschillend Android en iOS met deze bedreiging omgaan.

Het laatste deel beschrijft het Proof of Concept dat gemaakt is voor NVISO. Deze Proof of Concept is een eenvoudige spyware app die gebruik maakt van Android's toegankelijkheidsdiensten om MFA tokens te stelen van MFA apps.

Aan het eind rest de conclusie met betrekking tot de onderzoeksvraag, mijn persoonlijke ervaringen met dit project, en natuurlijk enkele mogelijkheden voor verder onderzoek. Als addendum zijn er enkele bijlagen.

Sleutelwoorden: mobiele malware – spyware – Android – iOS – toegankelijkheidsdiensten

List of figures

List of tables

Glossary

BYOD	Bring Your Own Device is a principle in which employees are allowed to bring their personal device to work and also use it for their work-related tasks.
Red teaming	Red teaming is the practice of testing plans, policies, systems, and assumptions by adopting a hostile approach.
MFA	Multi-Factor Authentication is a security principle in which users have to authenticate using at least 2 of the following types of factors: something the user knows (e.g.: password or PIN), something the user has (e.g.: badge), and/or something the user is (e.g.: fingerprint). Other factors might be the location of the user or the time on which the user tries to authenticate.
Open-source	Open-source is a term that originally referred to open source software (OSS). Open-source software is software whose source code has been published and is freely available to the public, allowing anyone to freely copy, modify and redistribute it without incurring copyright and surcharge costs. Open-source hardware is hardware whose design specifications, usually in a software format, are published and available to the public allowing anyone to freely copy, modify and distribute the hardware and source specifications.
ROM (custom)	Read-Only Memory (ROM) is a type of non-volatile memory. The ROM stores data from the manufacturer and prevents the data from being electronically modified. ROM is useful for storing software that rarely changes during the life of the system. The term “ROM” can also refer to a ROM device containing specific software or a file containing software. So are files modifying or replacing the Android operating system called “custom ROMs”.
Bootloader	A bootloader, or “bootstrap loader”, is the system software that is responsible for booting a computer.
Firmware	Firmware is a specific class of computer software that provides the low-level control for a device's specific hardware. Examples of firmware are the BIOS and the UEFI.
Root (user)	The root user is the Linux equivalent of the Windows administrator. It is a privileged account that has full control on the computer or network.
Sandboxed apps	Sandboxed apps are apps that are completely isolated from the rest of the system’s resources unless the user specifically grants them access to other features. Sandboxing, or containerization, is done by default by both Android and iOS.

OEM	The Original Equipment Manufacturer (OEM) of a device is the company that manufacturers and sells that device. The term is mainly used in the context of end products that contain subsystems from other manufacturers. In that context, the OEM in the main manufacturer of the device.
Trojan	A Trojan is a legitimate looking app that, when installed, is used to get remote access to the infected device. That way, hackers can access private data or use the infected device for other attacks.
Clickjacking	Clickjacking is a technique hackers use to trick a user into clicking on something that looks innocent, but is not. It is often used to steal sensitive information of the user, to download malicious apps, or to give malicious apps root permissions.
Overlay attack	An overlay attack is a mobile security attack in which the mobile malware is able to overlay its own windows on top of another programs. By doing this, the malware can intercept information or trick the user into clicking on specific buttons.
Signing certificates	Signing certificates are some sort of contract that iOS developers need before they can start developing apps for iOS.
C&C or C2 server	Command-and-Control (C&C) servers are controlled and used by attackers to maintain communications and send commands to systems inside a target network compromised by malware.
MITM attack	A Man-In-The-Middle (MITM) attack is a cyberattack where the attacker relays and possibly alters the communications between two parties who believe that they are directly communicating with each other.
Toast notification	A toast notification is some kind of popup that appears at the bottom of the screen and disappears automatically after a few seconds. It usually informs the users of non-critical information that does not require specific attention. Toast notifications should be non-interactive.
Social engineering	Social engineering is an umbrella term for all kinds of malicious activities that are accomplished through human interaction. It uses psychological manipulation to trick users into making security mistakes or giving away sensitive information.
REST API	A REST API or RESTful API is an Application Programming Interface (API) that is conform the REST architectural constraints. The term “REST” stands for representational state transfers. REST APIs are used to transfer representations of the state of the resource to the requester or endpoint. This information is delivered in one of several formats via HTTP.

SAST	Static application security testing (SAST) is a white box method of testing. This means the tester has access to the underlying framework, design, and implementation. The application is tested from the inside out. During SAST, the code is examined to find software weaknesses and flaws.
DAST	Dynamic application security testing (DAST) is a black box testing method. This means that the tester has no knowledge of the technologies or frameworks that the application is built on. The application is tested from the outside in. During DAST, the application is examined as it's running to find vulnerabilities that an attacker could exploit.

Table of contents

1 Topic introduction

1.1 Mobile security

As the popularity of devices like smartphones and laptops, and principles like BYOD grow, so does the importance of mobile security.

Almost everyone has at least one smartphone and one laptop. Some even have one for personal use and one for work. While most companies still provide their own devices to their employees, more and more companies implement the BYOD principle. But that principle can entail high risks.

This thesis was written during a red teaming internship at NVISO Security for people with knowledge in IT and a passion for mobile security.

1.2 Unauthorized access to mobile data

Mobile phones hold a lot of very private data, more than people realize sometimes. Malicious apps accessing that data might have a high impact.

Luckily, when installing a mobile app, that app has to ask permission for every type of data it wants to access. But many people do not read that list nor think about its fairness. Another problem is that some malicious apps are smart enough to hide that list of permissions with tricks. That way, users are tricked to click on a button, without knowing that they give access to their whole mobile device.

1.2.1 Android vs. iOS

Just like there is Linux and Windows for desktop, there are multiple operating systems for mobile devices as well. The biggest players in mobile operating systems are Android and iOS.

Because both work in a different way, it is important to talk about those differences in this thesis as well.

1.2.2 Protecting ourselves

Besides talking about the ways this threat can occur, this thesis will also talk about how one can protect himself against this threat from a developer's and a user's perspective, as well as from a company's perspective.

1.3 Research question

This thesis will thus talk about mobile security and unauthorized access to mobile data, with as main research question: *How can third-party mobile applications be used to access and acquire data?*

The following chapters will clarify how malicious applications might access data, what they might access, how high the impact might be, and of course, how to mitigate this threat.

1.4 Structure of this thesis

This thesis starts with a more theoretical introduction to the topics of *mobile security* and *mobile malware*. This chapter will talk about the different types of mobile apps and the different methods by which malware can be installed. The most important mobile operating systems, Android and iOS, will be compared with each other. The theory will end with the different types of mobile malware with a short description and some real life examples.

As research, this thesis will focus on mobile spyware. The research chapter exists of four main parts: first and foremost, the different types of mobile data, then, the different methods to get unauthorized access, after that, some examples of spyware from the past 10 years and the severity of this threat, and last but not least, the difference between Android and iOS in handling this threat.

After the research chapter, there is a specific use case of this threat. In this chapter, the working of a malicious app that steals MFA codes by exploiting the Accessibility service of Android will be explained.

In the last part, there will be some guidelines on how to mitigate this threat as a user, a company, or a developer.

2 Theoretical introduction

2.1 What are third-party applications?

2.1.1 Applications

The terms *application*, *app*, and *program* are often used interchangeably. Nonetheless, they are not synonyms, although *app* and *application* are. The term *app* is the shortened version of the term *application* but can also mean *mobile app*, while desktop and web apps are often simply called *applications*. A *program*, on the other hand, is something completely different.

SearchSoftwareQuality explains this very well:

"An application – also referred to as an application program or application software – is a computer software package that performs a specific function directly for an end-user or, in some cases, for another application. An application can be self-contained or a group of programs. The program is a set of operations that runs the application for the user [1]."

Applications often interact with the operating system of the device it is running on and can be distributed as proprietary or open-source.

2.1.2 Third-party

A good description of a third-part application would be:

"A third-party app is an application created by a developer that is not the manufacturer of the device the app runs on or the owner of the website that offers it [2]."

2.2 What are mobile applications?

Applications can be divided into three main groups: desktop applications, mobile applications, and web applications. Those three groups differ mainly in function, value, and capabilities. A combination of two or more of those types is called a hybrid app.

2.2.1 Mobile apps vs. desktop apps

Desktop applications and mobile applications differ mainly in their function and their value.

"A mobile app generally performs a single function or serves a single purpose, while a desktop application is designed to perform several functions simultaneously.

Some users feel that mobile apps are disposable. When you no longer need the app, you simply click the uninstall button. Conversely, if a desktop application malfunctions, gets lost, or is otherwise unavailable, it could negatively impact your work output for the day [3]."

"Desktop apps are usually much fuller than mobile apps and consist of all the features of a program, whereas the mobile equivalent is a simpler and easier-to-use version.

This makes sense when you consider that most desktop and web apps are built to be used with a mouse and keyboard along with a large display, but mobile apps are intended to be accessed with a finger or stylus on a small screen [4]."

2.2.2 Mobile apps vs. web apps

The differences between mobile apps and web apps lie more in their function and their capabilities:

Mobile apps have a limited amount of features. Web apps, on the other hand, are full of features but still need to be able to connect to the internet and run in a web browser. For those reasons, most web apps are lightweight.

2.2.3 Types of mobile applications

There exist three types of mobile applications: native apps, hybrid apps, and Progressive Web Apps (PWAs).

Native apps are mobile apps written for a specific OS. They can function without internet connectivity, but according to a study of the Pew Research Center in 2015, 83% still interacts with the internet or with web views. Today, this number might be even higher. Native apps are usually downloaded via an app store, but Android users can also download them via a website and install them locally. iOS users can use different certificates to bypass the app store. The advantages of native apps are that they can use all features of the mobile devices (if permitted) and that they are must faster. An example of a native app is the Facebook mobile app.

Progressive Web Apps (PWAs) can be used on every platform and are downloaded from a web browser. They also use web standards and thus need to be able to connect to the internet. Other disadvantages are that they cannot use all features of the mobile device and that they are much slower. Twitter and Pinterest both have a PWA version of their native app.

Lastly, hybrid apps are a combination of both. They can be installed like a native app, but they use web standards like PWAs, although they can also be used offline. An example of hybrid apps is the Google app.

The table on the next page compares the three types of mobile applications by comparing the characteristics of each type.

Table 1 - Types of mobile applications

	Native app	Hybrid app	Progressive Web App
OS-specific?	Yes	Depends	No
Downloads user elements from the internet?	No	Yes, but can be used offline as well	Yes, but can be used offline as well
Only downloadable via an app store?	No, but app store is main distributor	No, but app store is main distributor	No, also via web browsers
Can use all features of the mobile device?	Yes	Depends	No
Speed?	Faster	Intermediately	Slower

Security-wise, native apps can be considered the most secure as app stores are a more secure source for apps than web browsers. Also, apps that do not connect to the internet cannot be attacked remotely. The only downside is that a malicious native app might get access to the whole device.

2.3 How are mobile apps installed on your device?

Mobile apps can be installed in 2 different ways: with or without an app store.

Each mobile operating system has its official app store for mobile apps. While Android has the Google Play Store, iOS users have to use the App Store. iOS users with a jailbroken device can use Cydia. Both official app stores are also available on desktops, giving the user the ability to remotely install an app on his mobile device.

These official app stores have their own security mechanisms to ensure only legitimate apps get downloaded. Other app stores entail some risk. Those security mechanisms will be explained shortly in the next part.

Both operating systems have alternative app stores. While they might be less secure and prone to malware, they also bring benefits, when comparing them to the official app stores. So will developers be charged less app listing fees and can the alternative app stores promote apps as app of the day. A specific advantage for developers of iOS apps is the ability to target specific countries. Advantages for the users, on the other hand, are getting access to apps that are not available on the official app store. For Android users this also means having more privacy as Google logs every app that ever got installed on all devices with the same Google account.

There exist quite a lot of alternative app stores. Some well-known alternatives for the App Store are GetJar, Appland, and Builds.io. Android users can use the Amazon App Store, SlideME, or 1Mobile. Since China blocked Google in 2012, China now offers MyApp, Baidu, and 360 Market as alternative app stores for Android. Finally, there also exist some cross-platform alternative app stores that offer apps for various operating systems. Cross-platform app stores that offer apps both for Android and iOS are the Opera Mobile Store, neXva, and Appszoom.

One way of installing an app without an app store is via a desktop. This way, users can download any app on their mobile devices. Both Android and iOS users will need additional software tools. Android users will also need to put their phones into developer mode.

Android users also have a second option to install apps without the app store. They can simply download the APK file of the app via a web browser and install it via a file manager. That way, users do not need adb or a desktop computer.

Installing apps without an app store gives users the freedom to install any app they want, even self-programmed apps, but users should only do this at their own risk. Rooting or jailbreaking the device gives users even more freedom but is even more insecure.

2.4 What are the main differences between Android and iOS?

There are quite a few operating systems for mobile devices, but for the sake of the scope, this thesis will only talk about Android and iOS.

“Android is a mobile operating system based on a modified version of the Linux kernel and other open-source software, designed primarily for touchscreen mobile devices such as smartphones and tablets [5].”

Android also has a variant for wearables, Wear OS. Wear OS is compatible with phones running Android 6.0+ or iOS 10.0+. Supported features may vary between platforms and countries. Wear OS by Google for iOS is available for iPhone 5 and later.

“iOS (formerly iPhone OS) is a mobile operating system created and developed by Apple Inc. exclusively for its hardware. It is the operating system that powers many of the company's mobile devices, including the iPhone and iPod Touch; the term also included the versions running on iPads until the name iPadOS was introduced with version 13 in 2019. It is the world's second-most widely installed mobile operating system, after Android. It is the basis for three other operating systems made by Apple: iPadOS, tvOS, and watchOS. It is proprietary software, although some parts of it are open source under the Apple Public Source License and other licenses [6].”

2.4.1 Open source and availability

As explained in [5] and [6], Android is open source as its kernel is based on Linux, while iOS is mainly proprietary. The iOS kernel itself is not open source but is based on the open-source Darwin OS. For this reason, Android is available on devices of many manufactures like Samsung, Oppo, and Xiaomi. iOS, on the other hand, is only available on iPod Touch, iPhone, iPad, and Apple TV.

Although Android itself is open-source, Google Play Services is not. Google Play Services is a proprietary app that is installed on all Android devices by default. Play Services runs in the background at all times and can be seen as an extension to Android itself. The app makes it possible to update all other apps separately, without needing a system update. It also gives apps access to Google APIs. That way, apps that need to know the user’s location, for example, might use the Google Maps API. Google Play Services also hides sensitive information from apps and manages background tasks for battery efficiency.

Most open-source projects allow full control. Android does allow this as well, but not by default. To get more control, users have to root their devices or flash a custom ROM. But both actions have different impacts. Rooting the device gives the user full control, while flashing a custom ROM allows you to bypass some key checks.

A custom ROM is an alternative Android version, based on the Android Open Source Project (AOSP) or an already existing ROM. By installing a custom ROM, users can alter the source code of Android as desired.

Installing a custom ROM is easy, but creating one is not. That is why developers publish their own custom ROMs. Users can also combine their favorite functions of multiple custom ROMs. This is called ‘cherry-picking’.

Custom ROM comes from a third party, while stock ROM is customized by the mobile manufacturer (e.g.: Samsung). Flashing ROMs containing only the open-source parts is completely legal, although users will lose their warranty.

Advantages of custom ROMs are better performance, getting OS updates before the device manufacturer releases them, and being able to install otherwise incompatible apps. On the other hand, some manufacturers stop updating the device after a certain age. From that moment, custom ROMs are considered the most secure option.

A disadvantage of custom ROMs is that they have fewer free premium features than stock ROMs from the mobile manufacturer.

To flash a custom ROM, unlocking the bootloader is always necessary. A bootloader, or “bootstrap loader”, is the system software that is responsible for booting a computer. During the start-up of the device, the bootloader gets loaded into the working memory bootable medium like a hard drive, a CD/DVD, or a USB stick. The boot medium receives information from the firmware of the computer, i.e. the BIOS or UEFI.

The manufacturer might prohibit the unlocking of the bootloader for various reasons. One of them is money. Every device comes with some pre-installed apps. Some of those are third-party apps for which the manufacturer gets revenue. Without unlocking the bootloader, uninstalling these apps is impossible. Another reason is that by unlocking the bootloader the manufacturer loses control of the device and therefore cannot, for example, collect user data anymore.

If the manufacturer does not allow unlocking of the bootloader, one could try to 'live-flash' a ROM by using an exploit to get root privileges. Rooting the Android device is thus not always necessary to install a custom ROM. It all depends on the device manufacturer.

One can root the custom ROM itself after its installation, just like with the stock ROM.

Rooting and jailbreaking will be explained in the next part.

2.4.2 Rooting vs. jailbreaking

“Despite the two different names, rooting and jailbreaking are essentially the same; the first term is used for Android devices and the second for iPhones. On a new smartphone, many operations are only available to software created by the manufacturer or the creator of the operating system; all programs downloaded from app stores have limited access rights. Rooting or Jailbreaking describes the acquisition of complete administrator rights on the device, allowing third-party programs to perform operations that were not originally available to them, such as controlling CPU clock speed or overwriting system files [7].”

Rooting an Android device can be done in two different ways. The first way is exploiting vulnerabilities in the firmware via a tool that users can install by connecting their device to a computer with a USB cable. This option used to be popular, but because exploits are very unreliable users now tend to use the second option, which is unlocking the bootloader with the approval of the manufacturer. To do this, “users must type their smartphone’s unique number (IMEI) and download the necessary software. It will unlock the smartphone’s bootloader and enable the upload of a modified operating system that will grant a user’s full rights [8]”.

Apple does not approve jailbreaking iOS devices and is thus more restricted. To jailbreak an iOS device, users have to exploit vulnerabilities.

Rooting or jailbreaking a device can be very useful if the user knows what he is doing. Using a rooted or jailbroken device for normal use is generally not a security problem. As long as the user keeps installing software updates and security patches regularly and does not engage in unsafe behaviors, such as installing pirated applications or malware – even unintentionally – the security risk of rooting or jailbreaking a device is quite low.

iOS users must change the default root password because hackers might otherwise try to get remote control of the jailbroken device.

2.4.3 Default security settings

Android and Apple have different approaches when it comes to checking the integrity of apps and users.

Both Android apps and iOS apps are by default completely sandboxed. This means they are isolated from the rest of the system’s resources unless a user explicitly grants an application access to other features. This is a very secure approach as long as the app does not ask for unnecessary permissions.

Before an iOS app is available in the App Store, Apple must approve and even sign the app. The App Store then asks the user for an Apple ID before installing the requested app. This Apple ID is the combination of an email address and a password.

Before Android 6.0, most Android apps asked for certain permissions before their installation. This meant Android users could only choose between accepting all the permissions or not installing the app at all. After installation, the user could deny some permissions again.

iOS apps and Android apps since Android 6.0, on the other hand, have granular permissions. This means they only ask for permission when they need it. That way, its users can be selective in picking the app permissions while using the app. But this also means that the app usage will be interrupted with notifications and warning popups every time the app needs a permission that has not been granted yet. Users can choose to grant permissions for one time or for all future times.

Android explains the app permissions in much better detail, while iOS explains them more user-friendly to help its users understanding them without any confusion. Based on their app permissions, Android can thus be considered more transparent and informative than iOS.

Another security difference between Android and iOS is their approach to security patching. This is where iOS shines. Apple releases software updates and makes them available to all iOS devices at the same time. Google, on the other hand, only releases software updates and security patches to Pixel devices. Android devices from other manufacturers lay behind because they must take these security updates from Google and apply them to their own devices. To mitigate this delay, Google firstly distributes its patches to the manufacturers before releasing them to the Pixel devices.

Luckily, Android changed its approach to security patching since Android 10 in 2019 with Project Mainline. Project Mainline extends the efforts of Project Treble, which was introduced in Android 8.0, and modularized Android by separating the OS framework from the device-specific, lower-level software. With Project Mainline, Google does not have to wait for an Original Equipment Manufacturer (OEM) anymore to roll out an update but can do the task itself. This means that Android devices with Android version 10 or newer will install security updates automatically in the background. That way, users will no longer require to reboot their phones. This is mainly done by Google Play Services, as discussed before. Not everything is updated this way, but some security-critical libraries are. This is still not ideal, but already a step forward.

Since that same year, Android also collaborates with ESET, a respected member of the cybersecurity industry. As part of this collaboration, ESET will share with Google which apps it detects as malicious, potentially harmful or unwanted, before Google will publish the app on the Google Play Store.

2.5 What types of mobile malware exist?

Mobile malware is malicious software that targets other mobile apps or the operating systems on mobile devices. Most types of malware for desktops also exist for mobile, like trojans and viruses. But some types of malware only exist on mobile, like SMS or banking trojans.

Hackers target the Android OS more than iOS for three different reasons. In the first place, because Android allows users to install apps from outside the app store. In the second place, because its flexibility allows much more malware in the Google Play Store. And lastly, because there are more Android users than iOS users and thus more possible targets. But all of that does not mean iOS does not have malware.

Nokia's threat intelligence report of 2020 states that Android users are 15 times more likely to be infected than iOS users. In 2019, the difference was even greater. In that year, Android users were even 55 times more likely to get infected by malware.

See *appendix 1 – Infection by device according to Nokia's threat intelligence report of 2020*.

2.5.1 Subscription scammers

Subscription scammers allow their users to use their app for a short period. After that, the user will have to pay a certain amount each month or year. The user can cancel those subscriptions but has to do this manually. Unlike other apps, uninstalling this type of malware will not stop the subscriptions, which is why users often have to pay high amounts even after not using the app anymore.

This way of handling subscriptions leads to many discussions because it is technically not illegal; users can cancel the subscriptions, and the app warns users about the working of the subscriptions itself. For that reason, subscription scammers are not always considered malware as they work by the law.

2.5.2 Ransomware

Ransomware is a type of malware that denies access to the device or files on the device and then asks the user to transfer money to the attacker. The attacker usually asks for Bitcoin because it is completely untraceable.

There can be considered three types of mobile ransomware.

The first type is technically not ransomware but pretends to be. Most ransomware of this type just shows popups claiming to have locked or encrypted the device or some files. Users only have to uninstall the malicious app to get rid of this popup. Examples of this type of ransomware are Safari's JS popup scareware on iOS and Koler on Android.

The second type, also called crypto-malware or encryptors, does encrypt the files on the mobile device. An example of this type is SimpleLocker on Android.

The third and last type of ransomware, the lockers, locks both the device and all files on the device itself. Examples of this type are KeyRaider on iOS and DoubleLocker on Android.

The table below gives a short overview of those types of malware.

Table 2 - Types of ransomware

	Scareware	Crypto-malware / encryptors	Lockers
Does it actually encrypt something?	No, just shows a popup	Yes, all files on the device	Yes, all files on the device and the device itself
Example(s)	• Safari's JS popup (iOS) • Koler (Android)	SimpleLocker (Android)	• KeyRaider (iOS) • DoubleLocker (Android)

Ransomware is very effective as it is maddening and panic-inducing.

2.5.3 Mobile ad fraud

Mobile ad fraud is an attempt to steal money from advertisers by exploiting mobile advertising technology. Malware can commit mobile ad fraud in many different ways, from faked impressions (view fraud) to fake clicks and fake installs.

When doing view fraud, fraudulent publishers may make the advertisements on their website invisible by stuffing them into 1 pixel or by aligning them out of view.

An example of mobile malware using click fraud is Snaptube. Snaptube was removed from the Google Play Store in 2019 but is still available in some third-party app stores.

2.5.4 Adware

Adware is malicious software that places unwanted advertisements on the screen, often in the web browser. Hackers often distribute it as legitimate software to get money from the views and clicks on the ad. Another way of getting infected with adware is via an infected browser.

Examples of adware are Muda (also called AdLord), which attacks iOS, and Shedun, which attacks Android.

Many adware gets promoted on social media platforms like TikTok and Instagram.

2.5.5 SMS Trojans

SMS trojans send SMSs to premium numbers so users have to pay for each SMS. Those Trojans can give users big phone bills.

An example of such a Trojan was Trojan-SMS.AndroidOS.FakePlayer.a. It targeted Android users by disguising itself as a harmless media player app. According to Kaspersky, it is the first malicious program classified as a Trojan-SMS to target the Android OS.

Another Android SMS Trojan is the Joker malware. The Joker malware either sends premium text messages or uses the account of its victim to make purchases using WAP billing. Besides being an SMS Trojan, the Joker malware also acts as spyware by stealing text messages, contact lists, and device info.

2.5.6 Banking Trojans

“Bank Trojans are often disguised as legitimate applications and seek to compromise users who conduct their banking business – including money transfers and bill payments – from their mobile devices. This type of Trojan aims to steal financial login and password details [9].”

Banking Trojans can also be seen as a form of clickjacking as they often use overlay attacks to steal credentials.

An example of a banking Trojan is Wroba. Besides being a banking Trojan, it acts like spyware as well. It tries to steal files, passwords, contact lists, and messages, open web pages, make calls and send SMS text messages. One thing that is different from most malware is that this one attacks both Android and iOS.

2.5.7 Spyware

Spyware is a type of malware that collects, uses, and spreads personal or sensitive information without the user’s consent or knowledge. There are four main categories of spyware: adware, system monitors, trojans, and tracking cookies. One type of Trojan is known as a Remote Access Trojan (RAT). Attackers use it to gain access to the whole device to steal the private information of the user.

The HummingBad spyware attacks of Android devices using a chain-attack tactic in a very persistent way. It even reached fourth place in ‘the most prevalent malware globally’ list. In the same year, a new variant arose, dubbed ‘HummingWhale’.

Another well-known spyware is Pegasus, which attacks both Android and iOS. Pegasus can read text messages, track calls, collect passwords, use the microphone and video camera, and gather information from apps. It is sold commercially for a huge amount of money and is used by governments to track dissidents and free speech lawyers.

The research part of this thesis will focus on spyware, and after that, there will be a specific use case of an Android app that steals MFA tokens.

3 Research

3.1 What types of data might mobile malware access?

Mobile devices are used extensively because of their variety of features and compactness. Users can use one mobile device for all tasks and use the same device to store all their data in one place. That way, people always have everything at hand, as long as they have their phone with them.

But that flexibility has a downside. People sometimes forget how much data their phone holds. People often use their phones to surf the internet, take pictures with them, read their emails on them, etc. A malicious app that can gain access to all that data might have a high impact on the user.

The biggest problem is that often the most critical mobile data is overlooked. Users' browser history, pictures, and emails are evident. But mobile devices also know their owner's location, contact list, and behavior. Besides that, they are used for MFA as well. Stealing that kind of data from a mobile device might have a high impact as well. Especially if the data comes from a corporate device or from a device that is being used for both work and personal life.

Personal data on mobile devices can be allocated into three main categories:

- data about the use of the device (e.g. used apps and duration of usage);
- data on the device (e.g. contacts and photos);
- data from the device (e.g. location and movements).

The next chapter of this thesis will dive deeper into the use case of a malicious app that steals MFA tokens.

3.2 How does mobile malware get on your phone?

3.2.1 Users downloading malicious apps from an app store

Hackers might upload a legitimate-looking app to an app store. Users then think they are installing a legitimate app because it looks promising or looks exactly like a well-known app. An example of this is *appendix 2 – Result of looking for 'BNP Paribas fortis' in Google Play*.

Uploading a malicious app to the official marketplaces of Android or iOS is quite hard as those marketplaces are secured. iOS has its app review process, while Android has its Play Protect feature. But there are a lot of ways hackers try to circumvent these security measurements.

Sometimes hackers succeed in hacking the account of a developer with a good reputation. In that case, they can alter the apps of that developer as adding new ones could be noticed by the developer.

In 2015, some hackers even succeeded in distributing a malicious version of Xcode, the primary development tool used by iOS developers. They took advantage of users using third-party websites because of the slow download speed from Apple servers. The spiked version, dubbed XcodeGhost, would insert malicious functionality into applications that it built and prepared for distribution. According to FireEye, XcodeGhost was able to infect over 4000 apps.

But hackers or malicious companies can also upload the malware with their own accounts. Android's Google Play Store is an easy target for this type of attack, as it accepts self-signed applications. Apple's App Store, on the other hand, is more secure. iOS apps can only be downloaded from the App Store. To get in the App Store, iOS apps need to be signed by Apple itself and thus must go through the vetting process of Apple.

But developers can publish iOS apps with code-signing certificates as well. With those certificates, publishers do not need the App Store. The different types of certificates are enterprise certificates for companies to distribute apps to their employees, developer ID certificates for short-term testing purposes, or distribution certificates for long-term testing purposes. Distribution certificates can only be requested by Account Holders and Admins.

In 2019, the Italian company Connexxa released a spyware app called "Assistenza SIM". This malicious app abused the iOS enterprise certificate to bypass the security measures of the App Store.

When publishing the app themselves, hackers often reuse an already existing app and republish it under a slightly different name. An example of a republished app is the MilkyDoor malware, a Trojan that posed as a recreational app but allowed attackers to bypass firewalls. It also concealed its malicious activities within ordinary network traffic. For this, it used SSH on port 22. By using SSH, it was able to encrypt malicious traffic to make its detection even trickier.

Another example of a republished app is Triout. In 2018, it posed as the privacy tool called Psiphon to trick users into downloading it. Users could barely see the difference between the legitimate and the malicious version, as seen in the picture below. But after installation, Triout recorded phone calls, monitored text communications, tracked the user's location, stole photos, and even took pictures.

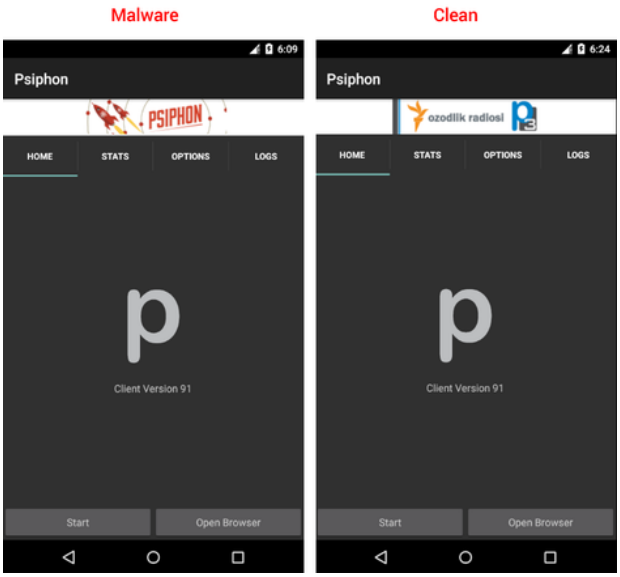


Figure 1 - The malicious version of Psiphon compared with the legitimate version
Source: https://www.zdnet.com/article/now-this-android-spyware-poses-as-a-privacy-tool-to-trick-you-into-downloading

Malware can remain unnoticed in the marketplace for quite some time when using advanced obfuscation techniques. Another technique to stay unnoticed is sleeping for a specific period after the installation or only activating itself after an update. That way, the user will not know for sure what app is causing the problems. The malware might also trick the marketplace by downloading the malicious code only after installation.

In 2019, hackers used simple but smart checks to bypass the security screenings before uploading 18 malicious iOS apps to the App Store. After installation, the app checked whether the phone on which it is installed has a SIM card. The presence of a SIM card would indicate that the phone belongs to a normal user rather than a security researcher or someone that screens apps for App Store approval. After ensuring the device was of a normal user, the apps would turn the device into an invisible click farm.

An easier way is uploading the malware to a third-party marketplace. Those marketplaces often do not check the integrity of the apps that get uploaded. The only downside is the less broad public.

3.2.2 Users downloading legit apps that use malicious third-party services

Attackers can use the same techniques as described before to add malicious dependencies to a legitimate app. That way, they do not have to change the whole functionality of the compromised app. Those techniques again include hacking the account of a legitimate developer, malicious developer tools, and republishing.

These types of malware also use the same techniques to remain unnoticed as any other type of malware.

An example of this is the MalBus spyware. It hides in a legitimate South Korean Transit app by hacking the original developer's Google Play account. After getting into the account, the hackers added a malicious library to two versions of the app. After installation of the infected app, MalBus phishes for the Google user id and password of the victim with a fake login page, drops a malicious Trojan on the device, and searches the device for specific military and political keywords, and exfiltrates files.

Another example is CamScanner, a free app that converts images into PDF documents. The app itself is legit, but in 2019, it had problems with a malicious third-party advertising library that secretly installed malware on the victims' phones. The library did this by decrypting a Zip archive in the app, which then downloaded additional files from several C2 servers. The app also collected infected files.

3.2.3 Users downloading malicious apps from other sources

App stores are not the only source of malware. Attackers also use social media and gaming platforms like TikTok, Instagram, and Discord to distribute malware. In its mobile threat report of 2020, McAfee explained how LeifAccess was distributed via malvertising and was uploaded to the Discord chat service.

Attackers might also distribute mobile applications via mail or SMS instead of uploading them to an app store. By clicking the link in the suspicious message, the user downloads malware on his phone. To install the app, users still has to allow other sources to install apps on their mobile device. Users are often tricked into allowing this with an overlay attack.

Other ways malware might get installed are via backup and restore operations, via file sharing apps, via malvertising, or even via Mobile Device Management (MDM) servers.

Malvertising does not require user interaction (like clicking) to install the malware. Seeing the malicious ad is often enough. An example of malvertising is the Svpeng malware, which targeted Google Chrome for Android users in Russia via Google AdSense ads.

3.2.4 Users using non-secure connections or URLS

Users surfing to insecure websites are likely to be a victim of MITM attacks and malware.

The browser itself could also be a source of vulnerabilities. All major desktop browsers allow plugins to extend functionality. Most mobile browsers do not allow plugins, although some do. Some Android browsers that allow plugins are Kiwi Browser, Firefox, Yandex Browser, or even the default Samsung browser for Samsung users.

Plugins add extra functionality to the browser, but can also introduce vulnerabilities or have malicious intends. Some examples of malicious plugins are Nyoogle, Copyfish, Particle, and Video Downloader for Facebook.

These vulnerabilities can lead to browser-based attacks. Some types of browser-based attacks are Man-In-The-Browser, clickjacking, drive-by downloads, and address bar spoofing.

Man-In-The-Browser (MITB) is a proxy Trojan related to Man-In-The-Middle (MITM). It takes advantage of vulnerabilities in the browser to modify web pages, modify transaction content, or insert additional transactions. Boy-In-The-Browser (BITB) is a simpler version of MITB.

Clickjacking is a technique of tricking a user into clicking on something different from what he perceives. A similar attack is a malware that automates some clicks in the browser.

Drive-by downloads are unintended software downloads from the internet, either through ignorance of the consequences or without the knowledge of the user. Hackers can accomplish this kind of attack by exploiting API calls for various plugins or writing shellcode to memory to exploit the browser or a plugin.

In address bar spoofing, an attacker succeeds in exploiting a bug in a browser that allows a malicious site to modify its real URL and show a fake one instead.

A famous example of a browser-based attack is JailbreakMe. JailbreakMe uses an exploit in the browser's PDF parser to execute unauthorized code and gain access to the underlying operating system.

3.2.5 Hackers exploiting OS vulnerabilities

The mobile operating system might also have some vulnerabilities that a hacker can exploit. Those vulnerabilities are usually quickly discovered and patched up, but the device will remain vulnerable as long as the user does not update the software on his phone.

Android is proved to be quite vulnerable. In 2019, it had even more vulnerabilities than Linux and Windows. But that does not mean Android is insecure. Many vulnerabilities are normal for open-source projects. That said, the number of vulnerabilities found in Android is going down over the years.

Examples of attacks exploiting OS vulnerabilities are Zero Day attacks, bloatware, and malware that exploits trusted OS features like the fingerprint scanner.

An example of malware that exploited the Apple Touch ID of iOS was the Fitness Balance app. The app asks for the user's fingerprint to authenticate, but while scanning the fingerprint, the app shows a popup for a payment of $99.99 or $119.99. The payment gets immediately verified with the Apple Touch ID.

If the hacker succeeds in compromising the mobile device, malware can be downloaded directly onto the device from the command and control (C&C) center.

3.2.6 Hackers attacking the supply chain

Sometimes the Original Equipment Manufacturer (OEM) wants to include additional features on top of those of the AOSP. The OEM might hire a third-party vendor to do this. Attacking the supply chain means that the third-party vendor adds those additional features and some malware to the system image that it got from the OEM before sending it back to the ignorant OEM.

This process can be seen in the picture below, with Triada as an example for the malware that gets injected:

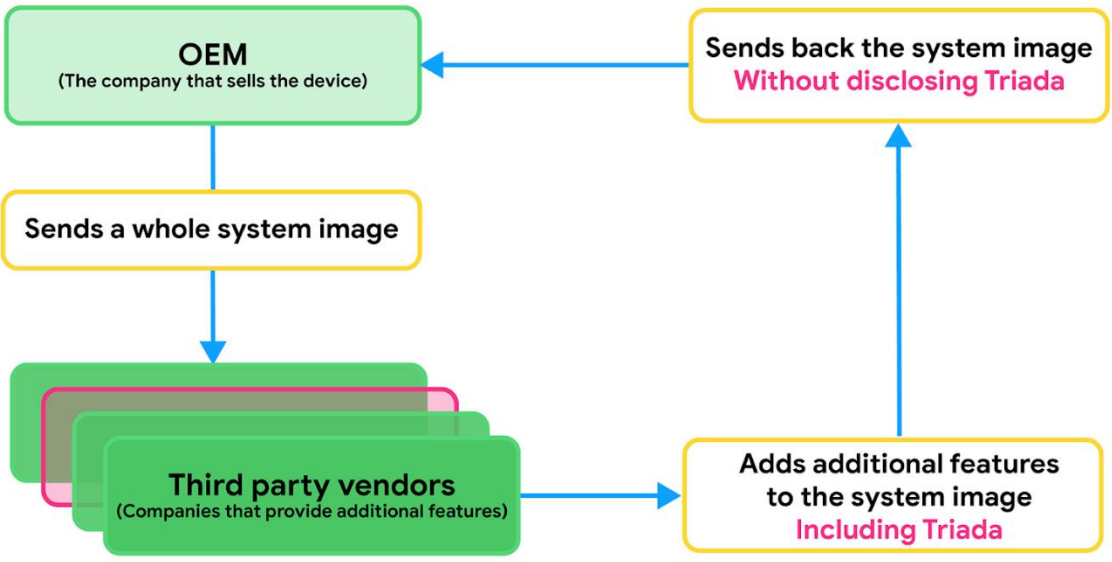


Figure 2 - Process of attacking the supply chain
Source: https://security.googleblog.com/2019/06/pha-family-highlights-triada.html

Triada used this technique in 2017 when Google Play Protect began strengthening its defenses against rooting exploits. The early versions of Triada worked on Android versions older than Marshmallow (version 6.0). The newer version targeted Android Nougat (version 7.0) as well.

Triada backdoored the log function to execute its code every time an app on the device tried to log something. That way, it could execute code in another app's context. Triada targeted the System UI app to get the GET_REAL_TASKS permission. With that permission, it could determine what the active app was. If the active app were a browser, Triada would display ads. That way, the user did not suspect anything. Triada would also abuse the Google Play app to install other apps in its context. The malicious apps that got installed then communicated with the backdoor in the log function.

3.3 How can mobile malware access your data?

Most malware does not root or jailbreak the mobile device it is attacking. Rooting is quite difficult because each device is different. Jailbreaking, on the other hand, is pretty easy but is almost never done by malware because it is simply unnecessary.

There are, however, some types of mobile malware that do root the targeted devices. They do this to persist factory resets and other efforts to remove it by installing themselves within the system partition of the operating system. Examples of malware that roots the device are the DroidKungFu spyware and the Brain Test rootkit. Both exploit Android devices.

Some mobile malware does require root privileges, and for that matter a rooted or jailbroken device, but does not root or jailbreak the device itself. This type of malware is also quite rare as it can only attack rooted or jailbroken devices. An example of type of mobile malware is the KeyRaider ransomware for iOS.

Not rooting or jailbreaking your device does not mean you are safe from malware.

This also applies to spyware. Spyware mostly asks for unnecessary permissions like accounts access, SMS permissions, microphone access, contacts access, or even device admin permissions. That way, spyware might get access to contact lists, email addresses, and text messages. It might also record conversations or give the attacker remote control of the whole device.

Spyware might also use advanced techniques to get access to data on mobile devices. Since more than 5 years, attackers are exploiting the accessibility services of Android. An example of this is the LeifAccess spyware. To remain undetected, malware might use techniques like file hiding, code obfuscation, and encryption. WireLurker uses these techniques to steal a variety of information from mobile devices.

LeifAccess is a Android Trojan that abused OAuth in 2019 by leveraging accessibility services to automatically create accounts and steal data. After installation, it does not show any icon and runs in the background. LeifAccess then showed a toast notification claiming the device is at risk if a certain service is not enabled. This service then performs malicious activities without any user interaction.

Google now limits the number of apps with accessibility service permissions in its Play Store in the most recent versions of Android. Google also moved some functionality to new APIs to minimize the abuse of accessibility services.

3.4 How real is this threat?

3.4.1 Is it possible and has it happened before?

Spyware on mobile device does happen. This thesis already cited some cases. Below, there is a short overview of this cases and some other examples of mobile spyware of the past 10 years. Most cases are already explained in previous parts of this thesis.

First, there was DroidKungFu on Android in 2011 and XAgent targeting iOS in 2015.

Then, in 2016, there were Wroba on Android and Pegasus on both Android and iOS, although Pegasus might have been active since 2013. In the same year, there were also some types of military spyware like SmeshApp, attacking the Android devices of the Indian military, and an Android-variant of XAgent, tracking Ukrainian units.

In 2017, there were GO Keyboard and HummingBad with its variant HummingWhale on Android. In that same year, there was also the ViperRAT spyware targeting Android devices of Israeli soldiers and AhMyth, an open-source RAT. Also in 2017, the first version of the Joker took residence. As of today, new versions of the Joker still get submitted to the Play Store.

The next year, in 2018, there was the Triout spyware that posed as a privacy tool for Android to trick users into installing it.

In 2019, Android was targeted by LeifAccess (Shopper), MalBus, and the RB Music spyware app that was based on the AhMyth RAT. In the same year, there were also the Assistenza SIM app, attacking iOS users, and the Exodus spyware platform, attacking Android since 2016. The CamScanner spyware attacked both Android and iOS in 2019.

Lastly, in 2021, a type of spyware posing as a system update of Android was discovered. It can steal sensitive information by abusing the accessibility service of Android.

See *appendix 3 – Mobile spyware over the past 10 years* for a quick overview of occurrences of this type of malware.

3.4.2 What are the risks of this threat?

Should attackers succeed in uploading spyware on a mobile device, they might have access to personal conversations, pictures, documents, location data, etc. Should the mobile device be a corporate one, the attacker might even have access to corporate emails and documents.

The impact and the risk of spyware attacks is very personal and different for each case. It all depends on what the user is using the device for and what value the user gives to his data. A user that uses his mobile device only for phone calls and sending text messages might find this type of attack acceptable. Someone who uses his mobile device for payments might be worried when this attack happens. For companies this type of attack might even have a higher impact as corporate secrets might get out.

One cannot give this type of attack a score based on the type of malware. One must decide for each device how high the risk would be and should, based on that score, take mitigations.

3.5 How different do Android and iOS handle this threat?

As explained before, Android and iOS use different methods to secure their operating system. This chapter will not compare those methods to see which operating system uses the most secure methods, but it will simply explain some of those methods in great detail.

Some methods are only used by Android, like using separate APIs and Google Play Protect. Other methods are only used by iOS, like enforcing signing certificates. But there are also some methods that are used by both operating systems. Those are enforcing apps to ask the user to grant certain permissions and encrypting all stored data.

On top of the methods explained above, this chapter will briefly explain some additional methods used by iOS. Most of those methods are used to secure the integrity of apps, like sandboxing them, scanning them for private APIs before publishing them to the App Store, and using code signatures to detect sudden changes in de app after an update. The last method that will be explained very briefly is iOS's efficient updating and patching scheme.

3.5.1 How do Android's separate APIs work?

Android puts its functionality in separate APIs. As a result, Android has separate APIs for Google Maps and the Calendar Provider. Android also uses a separate web API to get information about the user's social media accounts and several dedicated APIs for security features. That way, apps do not directly interact with the underlying Android system.

Apps can also use such APIs to communicate with Google Play Services for In-Process Controls (IPC). An example of this is resolving any issues at runtime, such as missing, disabled, or out-of-date services. Via Google Play Services, automatic updates are delivered independent of carrier, OS, or OEM system image updates.

Those APIs – also called *Google APIs* or *inaccessible APIs* – can be divided into two main categories: hidden APIs and internal APIs.

Hidden APIs are classes, methods, and resources that Google hides from the user for stability reasons and because their features may be changed in the next API version. Hidden APIs are located in the android.jar – also called the *development library* – and are used in a development environment.

Internal APIs are classes, methods, and resources that are reserved to system apps, whose development is more rigorous. Internal APIs give apps access to sensitive resources, e.g., low level access to hardware. They are located in the framework.jar – also called the *runtime library* – and are used in a runtime environment, after the apps are installed.

Although both libraries are built from the same source code, the runtime library is much richer than the development library as it includes an additional set of APIs which are inaccessible for the development of third party apps. This is also illustrated by the figure on the next page.

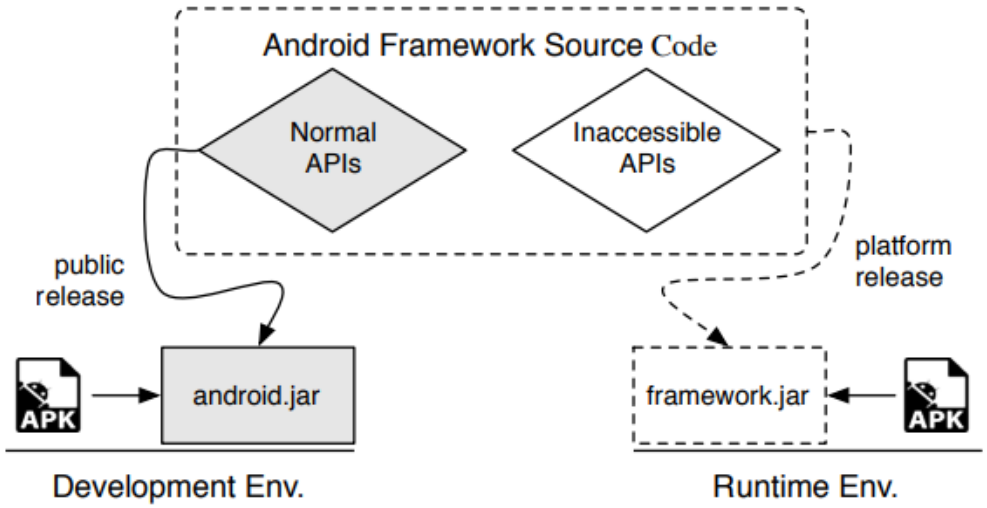


Figure 3 - The mechanism of Android APIs
Source: https://tngotran.wordpress.com/2018/08/23/relationship-between-android-sdk-aosp-code-framework-code-how-they-communicate-with-each-other

To have some sort of version control, Android also works with API levels. Each version has an API level which indicates which sets of APIs are available. For instance, Android 1.0 is API level 1 and Android 4.4 is API level 19. That way, it manage app compatibility across different versions of the operating system. Using this API levels, developers can easily declare which Android version their app minimally supports.

3.5.2 How does Android's Google Play Protect work?

Google Play Protect is an anti-malware feature of Android that is built-in in the Google Play Store since 2017. It warns the user about any detected Potentially Harmful Apps (PHAs) and removes known harmful apps. It runs a safety check on apps from the marketplace before the user downloads them and checks whether the mobile device has PHAs installed from other sources as well. Google Play Protect only checks installed apps from other sources if the user explicitly allows it.

Google Play Protect uses machine learning to constantly adapt and improve itself. It provides both on-device protection as cloud-based security to scan installed apps and test apps before they appear in Google Play.

Android's cloud-based security consists of an analysis and review of both the app and its developer before the app becomes available for download. Developers must sign the Google Play Developer Distribution Agreement. This contract contains a variety of policies the developer has to comply with. This compliance is then checked by Google Play's internal risk engine. The engine even checks the developer's relationships to check whether he has be associated with the creation of PHAs. The app review is done via both static and dynamic analysis. The detection is done both signature-based and heuristic.

Figure 4, on the next page, gives a clean overview on Android's cloud-based security.

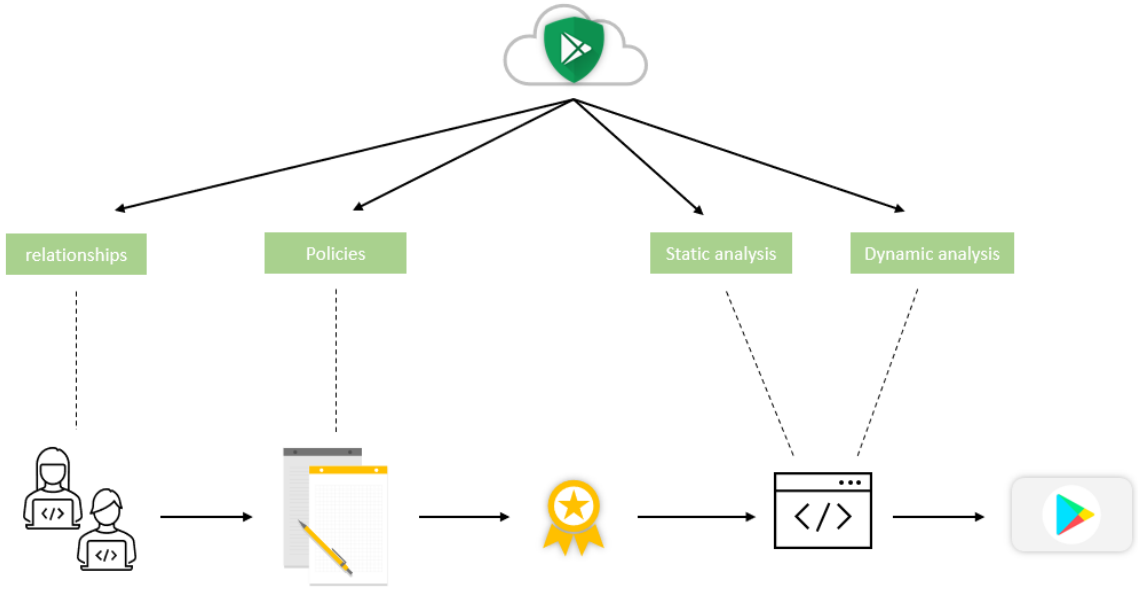


Figure 4 - Android's cloud-based security
Source: Kyra Van Den Eynde ©2021

For its on-device protection, Google Play Protect scans the installed apps every day. It removes PHAs and blocks future installs based on the app's behavior. To conserve the user's data, Google Play Protect only contacts the Google servers to request verification when a suspected PHA is detected. The on-device protection constantly runs in the background, but users can manually start a scan as well. As some PHAs are installed when the device is offline, Google Play Protect also provides offline scanning. Additionally, the on-device protection includes some SafetyNet APIs. These APIs allow developers to analyze the devices that installed their apps, use reCAPTCHA to protect their apps from bots, and verify whether a device is compatible with Google Play Protect. They also protect users against threats by allowing apps to verify the integrity of URLs.

In January 2020, AV-TEST compared Google Play Protect with 16 other well-know Android security apps, e.g., Bitdefender, Kaspersky, and Avast. The test revealed that Play Protect detected only 37% of all 6700 malware samples. The other security apps, both free and paid apps, all detected at least 98.9%. Besides letting malware slip through, Play Protect also branded some harmless apps as a PHA. AV-TEST concluded that Play Protect gives users a false sense of security and that users need to install an additional security app to be truly protected.

The results of this test can easily be explained as most accomplishments of Google Play Protect have actually been part of Android for ages. Play Protect is just a rebrand and separation to make Android's security more visible. This can easily be proven by looking at each functionality individually. Scanning the Play Store and the mobile device for malware is already done since 2012. Remotely wiping a mobile device can be done since 2013 and, since 2015, it warns users about potential harmful URLs. No extra security measures were added.

3.5.3 How do the permissions of Android work?

App permissions protect the user's privacy by protecting access to restricted data, such as the system's state, and restricted actions, such as recording audio. Android has three types of permissions, including install-time permissions, runtime permissions, and special permissions.

The first type are install-time permissions. These permissions are automatically granted when the app is installed and only give limited access to restricted data and actions. The app store shows a list of these permission in the app's detail page.

There are two groups of install-time permissions defined by Google; normal permissions and signature permissions.

Normal permission allow access to data and actions beyond the app's sandbox that entail very little risk to the user's privacy and the operation of other apps. Google assigns these permissions the "normal" protection level. Examples of normal permissions are VIBRATE to let the device vibrate and USE_FINGERPRINT to use the fingerprint hardware.

Signature permissions are only granted if the app that requests the permission is signed with the same certificate as the app that defines the permission. Google assigns these permissions the "signature" protection level. Examples of signature permissions are REQUEST_INSTALL_PACKAGES to request the installation of new packages and WRITE_VOICEMAIL to modify and remove voicemails.

The second type are the runtime permissions. Apps have to request the user to grant those permissions at runtime at the moment that they need them. Runtime permissions affect the user's privacy and other apps more than install-time permissions. Google assigns these permissions the "dangerous" protection level. Examples of runtime permissions are CAMERA to use the camera of the device and CALL_PHONE to initiate phone calls without needing the confirmation of the user.

Lastly, Android also has some special permissions. Special permissions correspond to particular app operations and can only be defined by the platform and manufacturers. Google assigns these permissions the "appop" protection level. Examples of special permissions are LOADER_USAGE_STATS to allow a data loader to read a package's access logs and SYSTEM_ALERT_WINDOW to create overlays on top of other apps.

Permissions might have multiple protection levels. Those permissions will be granted if the app qualifies for any of the permission levels. For that reason, many permissions have the "signature" level to make sure that the platform-signed apps can be granted these permissions automatically.

3.5.4 How do Android and iOS encrypt their data?

Data encryption will not stop spyware that uses accessibility services, like the use case in the next part of this thesis. However, it will stop most other forms of spyware and hackers that get access to someone's device.

Both Android and iOS implement only File-Based Encryption (FBE), although many previous versions of Android once supported Full-Disk Encryption (FDE) as well. With file-based encryption, each file on the device is encrypted individually. With full-disk encryption, every bit of data stored on the disk gets encrypted. Both encryption methods are only engaged as soon as the device gets locked. When unlocking the device with the device password, the data gets decrypted.

Android's full-disk encryption was introduced in Android 4.4 and was based on dm-crypt. It used an 128-bit AES algorithm with Cipher-Block Chaining (CBC) and ESSIV:SHA256. The master key was encrypted with 128-bit or 256-bit AES by using the OpenSSL library. This key was then protected by the user's device password.

While full-disk encryption is very secure, it may cause some core functionality of the device not being available directly after a reboot. These functionalities include alarms and accessibility service. That is why full-disk encryption is no longer allowed on devices running Android 10 and higher. These devices thus have to depend on file-based encryption to secure their data.

Android's file-based encryption is supported since Android 7.0 and allows different files to be encrypted with different keys that can be unlocked independently. This type of encryption uses AES or Adiantum as encryption algorithm.

One of the biggest advantages of file-based encryption is that with this encryption method and APIs that make apps aware of the encryption, apps can operate within a limited context before the user fills in his device password. Thanks to a feature called Direct Boot, installed apps can get access to the Device Encrypted (DE) storage before the user has unlocked the device. After the device is unlocked, the installed apps get access to the Credential Encrypted (CE) storage as well.

Since Android 9, also the metadata of files can be encrypted if the hardware supports that functionality. With metadata encryption, things like directory layouts and permissions are encrypted. The key for this type of encryption is protected by Keymaster, which in turn is protected by verified boot. Metadata encryption is enabled by default on adoptable storage whenever FBE is enabled, but it can also be enabled on internal storage.

Android describes this Keymaster as "the software running in a secure context, most often in TrustZone on an ARM SoC, that provides all of the secure Keystore operations, has access to the raw key material, validates all of the access control conditions on keys, etc. [10]"

Both Android's file-based encryption and full-disk encryption use symmetric encryption keys.

iOS's file-based encryption is used to secure apps' files and to prevent unauthorized access to them. It is automatically enabled when the users set a passcode for their devices since iOS version 8.0. Every time a file is created, a new 256-bit key is created. This key is then sent to the hardware AES engine, which uses the key to encrypt the file as it is written to flash storage.

For each file that gets created, the user can choose between 4 protection levels. This protection level determines when one may access the file. The first level, called “No protection”, does not encrypt the file at all, making the file accessible at all times. The second level, called “Complete until first user authentication”, makes the file inaccessible until the first time the user unlocks the device. After the first unlocking of the device, the file remains accessible until the device shuts down or reboots. Then, the third level, called “Complete unless open”, gives access to closed files only when the device is unlocked. Files that are open remain accessible, even after the device gets locked. The last protection level is called “Complete”. This level makes files inaccessible until the device gets unlocked.

If no protection level is chosen by the user, Apple applies the default protection level, “Complete until first user authentication”.

iOS’s file-based encryption is covered by a feature that is called “Data Protection”. It has enhanced a lot since Apple started using Secure Enclave.

The Secure Enclave is a hardware chip that manages all encryption and authentication functionalities. The Secure Enclave is used in iOS devices since iPhone 5S in 2013 to secure the private data of the user. The purpose of this chip is to store and manage encryption keys and biometrical information that is too privacy-sensitive to be stored into memory. It is separated hardware-based from the normal processor so that the normal processor cannot access the sensitive information. This means that the Secure Enclave has its own processor, called the Secure Enclave Processor. The Secure Enclave can be found in the A7 chip, T1 chip, S2 chip, and the recent proprietary chips from Apple. The chip has access to a special part of the RAM memory that is encrypted by the Memory Protection Engine. This part of the RAM memory is called TZ0.

The Secure Enclave makes it very difficult for hackers to hack iOS, but in 2017, some hackers revealed that they had managed to crack the Secure Enclave. By doing that, they might have gotten some insight into the working of the chip. Fortunately, they were not able to retrieve the encryption keys and could only decrypt the firmware itself.

As Android and iOS both use very similar encryption algorithms, one can conclude that their encryptions are of the same strength as well. The only big differences are the implementation method and the way they store their encryption keys.

3.5.5 How do iOS’s permissions work?

Just like with Android’s permissions, the purpose of iOS’s permissions is to protect the privacy of the user. But in contrary to Android, iOS only has 2 types of permissions: basic permissions that are automatically granted and permissions that are granted at runtime.

The basic permissions of iOS are similar to the install-time permissions of Android. They are all granted be default. An example of such permission for iOS is internet access.

The runtime permissions of iOS and Android work in the same way. As soon as the app needs such permission, it will show a popup with a request. The user can then decide if it will grant the permission or not. Since iOS 10, it is required to provide a clear description that explains why the app needs the permission in question. When no description is included, the app will crash. The description itself must contain a brief active sentence that clearly describes how and why the app collects data. Examples of runtime permissions on iOS are Contacts, Microphone, and Speech recognition.

3.5.6 How do iOS's signing certificates work?

As described before, all iOS apps have to be signed before they can be available in the App Store. Signing can be done by Apple via TestFlight or the App Store, or by the developer itself for test versions or the app through specific certificates.

TestFlight is an online service owned by Apple that is used for over-the-air installation and testing of mobile apps. Initially, it supported testing both iOS and Android apps, but since 2014, support for Android was retracted. TestFlight is now only available to developers within the iOS Developer Program. Via this service, developers can receive remote logs, crash reports, and tester feedback.

To develop an iOS app, a developer must request a certificate from Apple. This can only be done if the developer is a member of the Apple Developer Program. To request a certificate, the developer has to generate his private key. With that key, a signing request can be uploaded that in turn can be signed by Apple. That way, the certificate is stored locally without ever being sent over the internet. Recent versions of Xcode can automate this flow.

Paying developers can manually add multiple certificates that are all valid for 1 year. There are 2 paid programs: the Apple Developer Program for 99 USD per year and the Apple Developer Enterprise Program for 299 USD per year. The prices may vary by region and are listed in local currency during the enrollment process.

Non-paying developers get certificates that are only valid for 7 days. That way developers can test versions of their app locally on their phone without having to pay to Apple. These certificates do have to be rebuild, resigned, and reinstalled every 7 days.

To release an iOS app, developers can choose between releasing to TestFlight for testing purposes and the App Store to distribute their app to the public. For both options, the app has to be signed by a certificate and validated by Apple itself.

To use a certificate, developers need a provisioning profile to sign their apps with. That profile holds a Bundle ID and a list of certificates.

The Bundle ID uniquely identifies an app and must be registered to the developer's account. It is defined in Xcode and is written in a reverse DNS notation (e.g. com.myCompany.myApp). It also forms the App ID together with the Team ID as prefix.

The list of certificates indicates which certificates can be used with that provisioning profile. When generating a new certificate, for example when an old one has expired, a new provisioning profile has to be generated as well.

When using a development certificate, the provisioning profile also contains a list of devices. This list contains all UDIDs of the devices that can run the app. When releasing the app, these devices do not have to be re-entered. Apple will just resign the app with a releasing certificate and publish the app to the App Store and make it available to the whole public.

The provisioning profile is also illustrated in the figure on the next page.

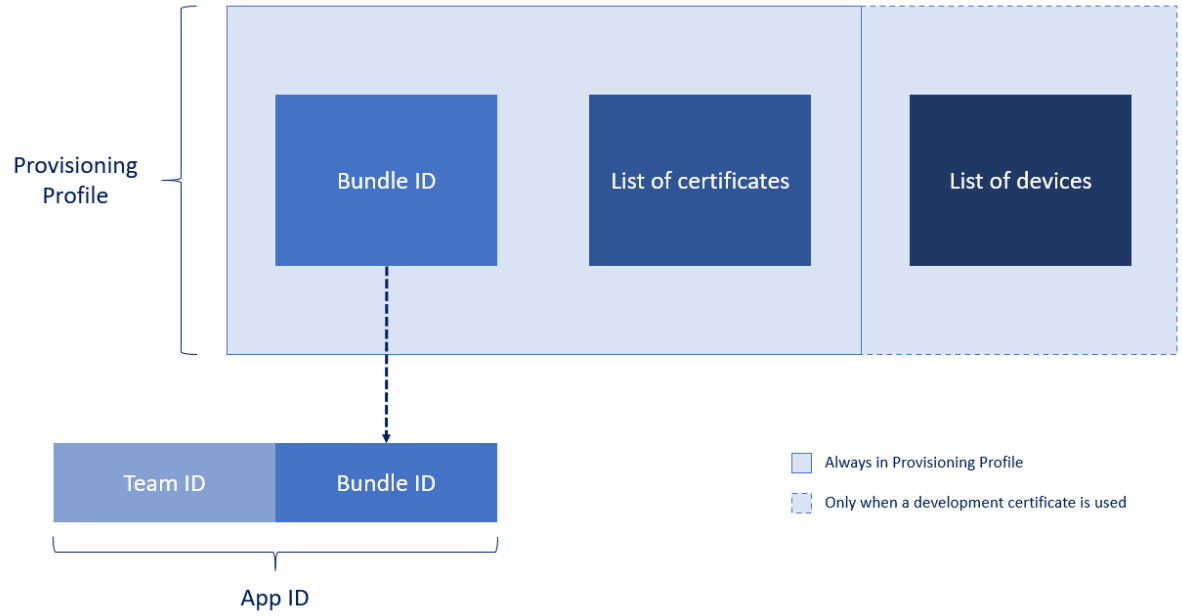


Figure 5 - Schematic illustration of provisioning profiles in Apple
Source: Kyra Van Den Eynde ©2021

3.5.7 What does iOS do besides enforcing certificates?

Apple does many other things to protect its operating systems and installed apps on its devices besides forcing developers to use certificates. These section will list the most important measures against spyware.

One of those things is running all iOS apps in a sandbox. This is also done by Android. Application sandboxing, or containerization, is used to limit the environments in which certain code can execute. Since iOS 13.4, all third-party apps have a Data Vault that protects all their data even from processes that are not sandboxed. This way, apps should not be able to access data of other apps. Unfortunately, some malicious apps do succeed in escaping those sandboxes. In 2019, Google Project Zero found an iPhone malware implant that could escape the iOS sandbox and run as root.

Additionally, all iOS apps get scanned for private APIs before they get approved by Apple and get published in the App Store. Only development version of apps do not get held back for using private APIs. Private APIs are classes and methods used by Apple itself to get device info or a list of installed apps to show in the settings app. Those APIs can make it possible for malicious apps to bypass permissions and are thus prohibited in the App Store. Abusing the private APIs of Apple can result in the removal of the app from the App Store and possibly even a banishment as a developer.

On top of that, Apple uses code signatures to ensure app have not been modified since they were installed or last updated. This is done by checking if all executable memory pages are made as they are loaded.

Lastly, possibly one of the most important things that Apple does to secure its iOS devices is maintaining an efficient updating and patching scheme. This results in 80% of all iPhones having the latest iOS version – iOS 14 – as of May 2021, according to Apple. On top of that, Apple's update system can even help preventing downgrade attacks so that iOS devices cannot be rolled back to an older version of the operating system.

4 Use case: Android app that steals MFA tokens

This part describes the use case of a specific type of spyware. This type of spyware steals MFA tokens from mobile devices by using an Android accessibility service.

As the practical part of this thesis, I made an Android app that uses an accessibility service to steal MFA tokens from Google Authenticator, Microsoft Authenticator, and Authy. The malicious app targets devices with Android version 6.0 or newer.

First, the decisions that were taken will be explained. Then, the working of the Proof of Concept of this app will be explained. After that, my PoC will be put to the test to find its strengths and weaknesses. Lastly, I will brainstorm around how NVISO could get this app on mobile devices of their future clients in future pentest projects.

4.1 Decisions

4.1.1 Why this specific use case?

Mobile devices are already part of the day-to-day life and Multi-Factor Authentication (MFA) is getting more popular every day.

People use MFA because it should make it more difficult for hackers to get access to their accounts. As some malware already can bypass MFA, it could be very useful to check how easy it is to hack MFA and where its weaknesses are.

This use case is also partially on behalf of NVISO as it was already determined that such an app could be beneficial to red teaming engagements, but one was not yet available.

4.1.2 Why targeting Android?

The reason why this specific malware app will target Android is because mobile hackers already target Android more than iOS and because there are more Android users than iOS users and thus more possible targets. This use case would thus be more relevant if it focuses on Android.

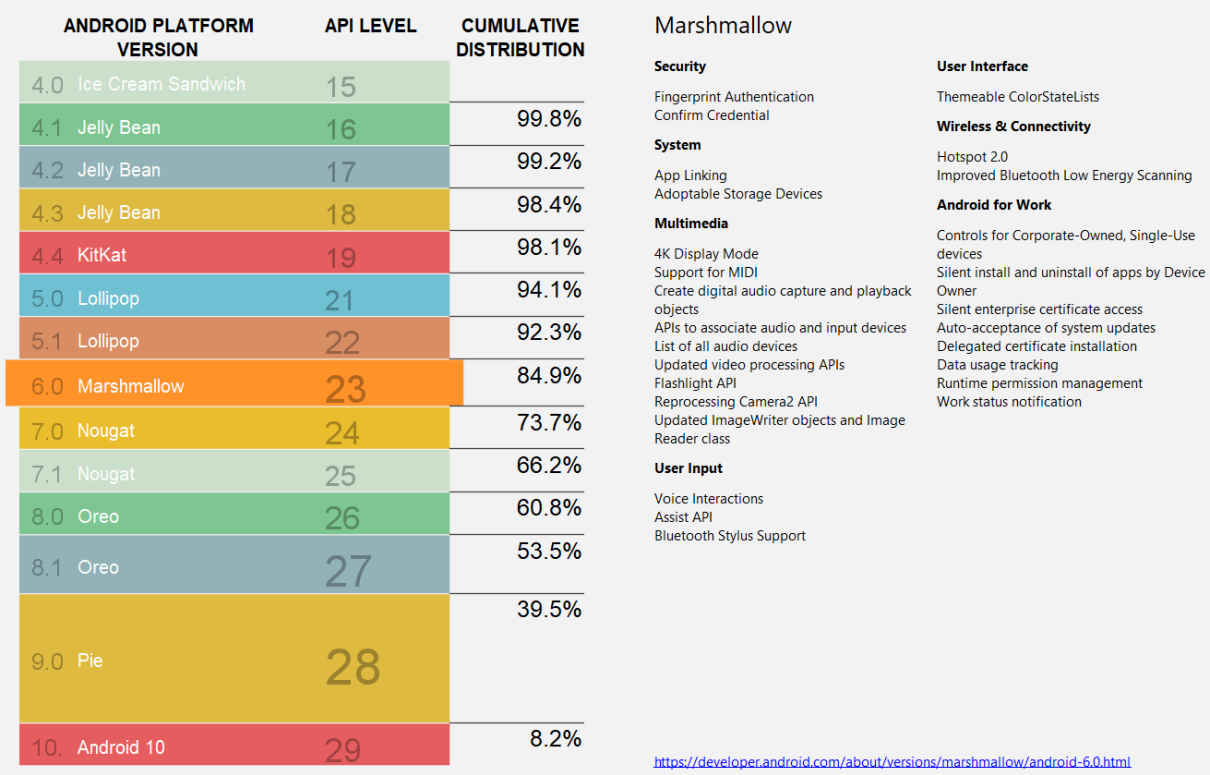
The reasons why hackers target Android more often than iOS is described in chapter 2.5 of the theoretical introduction.

4.1.3 Why targeting Android version starting from Android 6.0?

As described before, many Android devices might still have an older version of the operating system installed. In the screenshot of Android Studio 4.1.2 on the next page one can see what percentage of Android users use which version as of January 2021.

As one can also see in that screenshot, 11.2% of all Android users use Android 6.0. When focusing on Android 6.0, this malicious app would be able to abuse some old vulnerabilities and target at least 11% of all Android users.

Should the app be able to target Android 7, 8, 9, or even 10, this app would be able to target even 24.1%, 45.4%, 76.7%, or even 84,9% respectively of all Android users.



ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.8%
4.2 Jelly Bean	17	99.2%
4.3 Jelly Bean	18	98.4%
4.4 KitKat	19	98.1%
5.0 Lollipop	21	94.1%
5.1 Lollipop	22	92.3%
6.0 Marshmallow	23	84.9%
7.0 Nougat	24	73.7%
7.1 Nougat	25	66.2%
8.0 Oreo	26	60.8%
8.1 Oreo	27	53.5%
9.0 Pie	28	39.5%
10. Android 10	29	8.2%

Figure 6 - Android platform versions.
Source: Android Studio. Copyright 2021 by Android Studio 4.1.2

4.1.4 Why using an accessibility service?

Accessibility services are added to Android since Android 1.6 in 2009. Since Android 4.0 in 2012, accessibility services can act on behalf of users, including opening and closing apps, pressing buttons, and interacting with text fields.

Accessibilities are meant to assist users with disabilities, or who may temporarily be unable to fully interact with a device. But hackers found this feature also very useful as they now only have to get their spyware with their own accessibility service on the victim's device and then have to convince the victim to enable the attacker's accessibility service through social engineering. As soon as the accessibility service is enabled, it is practically game over because then, the malicious accessibility service can not only read the content of TextViews, but also block users from accessing pages and automatically grant itself additional runtime permissions.

4.1.5 Which MFA apps is the PoC targeting?

This spyware app does not focus on stealing MFA tokens from text messages as stealing tokens from MFA app would be more difficult and relevant to the research on hacking with Android's accessibility services. If this app could steal tokens from well-known MFA apps, stealing tokens from text messages would be child's play.

For the relevance of this use case, it was important to focus on the most used MFA apps. This specific use case focusses on Google Authenticator, Authy, and Microsoft Authenticator, whose logos can be seen in the same order in the figure below.

Figure 7 - Logos of Google Authenticator, Authy, and Microsoft Authenticator
Source: https://www.twilio.com/blog/2018/01/google-authenticator-support-in-authy-api.html

In the Google Play Store, Google Authenticator has more than 50 million downloads, while Authy and Microsoft Authenticator have more than 10 million downloads.

4.2 The Proof of Concept

The Proof of Concept is created from scratch as no open-source project already existed that did a similar thing. The PoC poses as an app that will monitor the privacy of the user and is completely written in Java.

The working of the PoC can be explained in a few general steps. First, the user has to install the app. Then, the malicious accessibility service that is used by the app will ask the C2 server for a command. If the C2 server has a command ready, the malicious service will begin to steal tokens. When it is done, it will send all captured tokens to the C2 server and will wait for the next command.

4.2.1 The user installs the app

First, the user has to download and install the app. When opening the app, the user will be shown a warning popup explaining that the app needs a certain service to be enabled in order for the app to be able to run. As soon as the user closes this popup, he gets redirected to the settings app where he can enable this specific service. This service is of course the malicious service that will steal the MFA tokens in the background.

As long as the malicious service is enabled, the popup will not be shown anymore when the app is opened. But as soon as the user disables the service again, the warning popup will be shown again as well.

4.2.2 The service asks the C2 server for a command

In the background, the app will make a call to the C2 server every 10 accessibility events. An accessibility event is triggered when the state of the screen changes. If the app gets a command back, it will do some malicious actions based on that command. If the app does not get a command back, it will wait 10 accessibility events before asking again.

There are 4 different commands in this PoC: *Google*, *Microsoft*, *Authy*, and *All*. For the first three, the app will just open the respective app and steal the tokens from there. For the last command, the app will first steal all tokens it can find in the Google Authenticator, then in the Microsoft Authenticator, and lastly in Authy.

If an MFA app is not found, the app will let the C2 server know via an extra message.

4.2.3 The service steals the tokens

Stealing tokens from the Google Authenticator is the easiest. For that, the malicious accessibility service just has to open the MFA app. All accounts and their corresponding token will get displayed on the main screen. The malicious service can thus scrape the tokens all at once. No other actions are needed.

Stealing tokens from the Microsoft Authenticator is a little bit harder. When opening the app, only the account names are shown. To see their corresponding token, one has to click on the account name. Then, the MFA app opens another view. The malicious accessibility service thus has to click on every account and has to go back to the home screen after every token capture. This can be done by clicking on the back button of the app itself.

Stealing tokens from Authy is even harder. The user can in fact choose between two different layouts of the app and can switch between them at any time.

The first option is called the List Mode and works just like how the Microsoft Authenticator works. The only difference is that Authy does not has a back button in its app. To go back to the overview screen, one has to trigger a ACTION_UP event or has to temporally close the app.

To find out what event to use, one should decompile the apk with jadx-gui. Under *resources* > *res* > *layout* one can find an interesting xml file, *view_tokens_list.xml*. This file explains that a certain SlidingLayer is clickable (see *appendix 4 – Screenshot of view_tokens_list.xml*). This SlidingLayer can be found under *com* > *authy* > *authy* > *p016ui*. When interpreting the code (see lines 20, 23, and 37 of the code in *appendix 5 – Screenshot of SlidingLayer's simplified onTouchEvent method*), one can conclude that the value of motionEvent has to be 1, which corresponds to ACTION_UP (see the documentation on MotionEvent[1]).

The other option, however, is completely different from the Google and Microsoft Authenticator and is called the Grid Mode. This Grid Mode shows the accounts in a list of tiles at the bottom of the screen and is actually the default mode as well. In the middle of the screen, Authy highlights one of those accounts. For the highlighted account, both the account name and the corresponding token are shown, while for the tiles, only the account name is shown. To scrape the tokens of all accounts when Authy is in Grid Mode, one has to click on every tile and then scrape the account name and the token before going to the next.

Figure 8 on the next page illustrates the possible screens the accessibility service might encounter when attacking Authy, considering this app holds at least one account. When Authy is in List Mode, the view on the left is the overview screen and the view in the middle is the detail screen. When Authy is in Grid Mode, the right view is the overview screen and also the only possible screen.

[1] https://developer.android.com/reference/android/view/MotionEvent

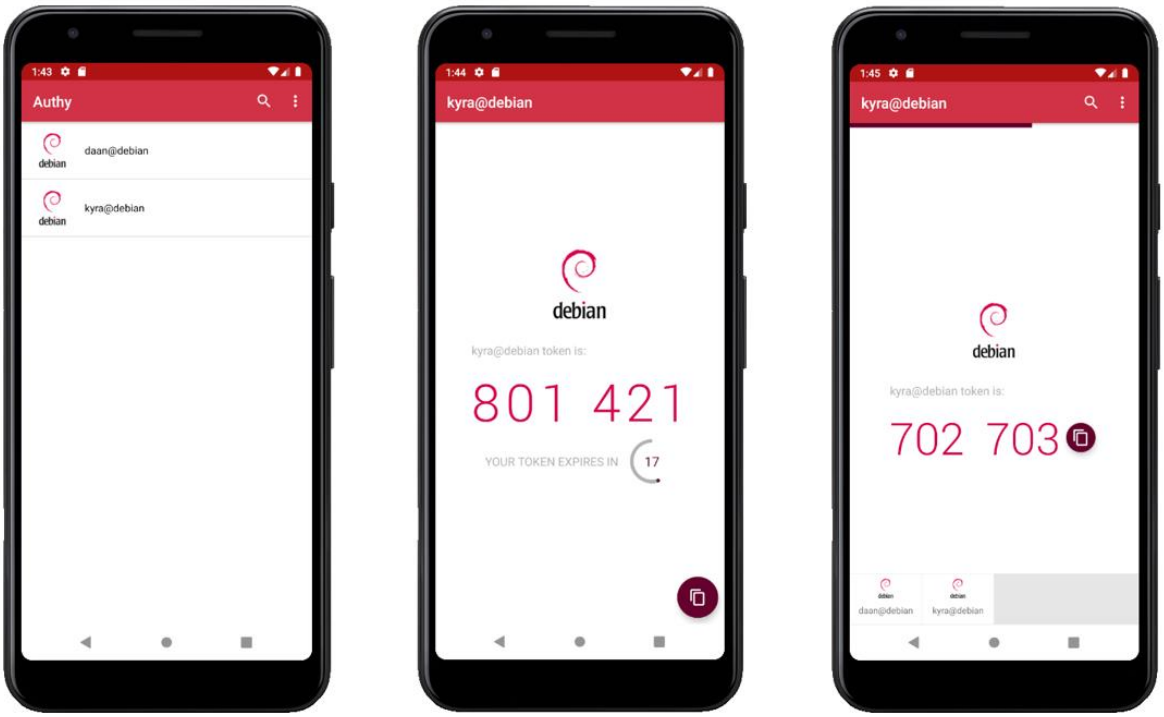


Figure 8 - Possible displayed views when attacking Authy via accessibility service
Source: Kyra Van Den Eynde ©2021

At the moment of writing this thesis, stealing tokens from Authy in List Mode is completely supported. The support for the Grid Mode is almost finished, but still misses some functionality.

On top of that, both the Microsoft Authenticator and Authy can also be secured with a PIN code or fingerprint. If this is the case, the malicious service will have to show an overlay to trick the user into filling in his PIN code or to use his fingerprint. Unfortunately, this functionality has not been added yet to the PoC due to time constraints, although some checks for Microsoft Authenticator are ready.

After having stolen all tokens from the MFA app, the malicious accessibility service will minimize the MFA app as it can not properly close apps. In case the malicious service got the "All" command from the server, it will just go to the next MFA app until all MFA apps are scraped.

If an MFA app without accounts is found, the app will let the C2 server know via an extra message if the *All* command was used. In case any of the other commands was used, it will send an error message.

The original goal was to show an overlay view before the (first) MFA app would be opened. This overlay view would show a message like "Looking for an update…" and would only be closed after the attack is over. That way, users would not notice anything. But due to time constraints, this PoC does not support this functionality yet, although it will be added in the near future.

4.2.4 The service waits for another command

When the app is finished with the received command, it will send its results to the C2 server and again wait 10 accessibility events before asking for the next command. The whole cycle gets repeated as long as the user does not disable the malicious accessibility service.

The parameters of that get send to the C2 server are the accounts and a few messages for extra info. Each account contains info about the app it came from, the name of the account, the stolen token, and the remaining time that the tokens will be valid. A timestamp will be added at the server-side for now to ignore any possible differences in time settings. The messages contain info like the command from the C2 server and a list of what apps where not found by the malicious service.

All communications between the app and the C2 server are in JSON format and the C2 server itself is a basic PHP REST API that handles all requests from the app. The global command that is valid for any device that calls this API is just a hardcoded variable.

All attacks take less than 30 seconds. Unfortunately, the attack cannot be accelerated. The malicious service needs to wait on the MFA app to load, and that might take some time in the case of the Microsoft Authenticator and Authy. Google Authenticator, on the other hand, loads in less than a second. This means that this type of attack cannot be executed unnoticed without the use of overlays.

4.3 Putting the PoC to the test

4.3.1 Does it work on more recent versions of Android?

No. Unfortunately, it does not. Or at least, not completely. This PoC was developed while using an Android 9 emulator. On this version, every scenario was tested and now works.

To test if the malicious app would also be able to run on other versions of Android, an emulator was created for each version of Android starting from Android 6. Then, all three MFA apps were installed without any accounts. After that, the app was run. All emulators got the “All” command to test all MFA apps in the fastest way possible. If the app could successfully open each MFA app, notice that it did not hold any account, and inform the server of its findings, the Android version would pass the test. In any other case, the Android version would fail the test.

The results were quite surprising. In Android 6, 7, and 11 the current version of the PoC does not work. In Android 11 the app crashed when the second MFA app, the Microsoft Authenticator, was opened. In Android 6 and 7, In Android 6 and 7, the app already crashed when the first MFA app, the Google Authenticator, is opened. Due to tie constraints, it was not tested if adapting the code for those three versions would help as that might have taken quite some time. Fortunately, this PoC does work on Android 8, 9, and 10. According to the statistics one can find in Android studio (see figure 6), this would mean that this PoC can target approximately 60.8% of all Android users, which is still a lot.

Of course, this was not the most complete test one could do. To make sure all versions that passed the test actually work, one should test if the Android version would also let the app steal the tokens. Due to time constraints, a complete test was not possible.

4.3.2 Does it work when the phone is in sleeping mode?

No. First and foremost, the malicious service depends on Accessibility events. During this sleeping mode, no accessibility events are triggered. On top of that, no apps can be opened when the app is in sleeping mode. And last but not least, the accessibility service cannot read text from the screen when the screen is black.

I tested this by powering off an emulator running the PoC.

4.3.3 Does it work when the lock screen is shown?

Probably not. The PoC will not work for the same reasons as it will not work in sleeping mode. When the device is still locked, no apps can be opened and no text can be read from the MFA apps.

But, contrary to the sleeping mode, the lock screen does trigger the malicious accessibility service. This means some extra checks will be needed before this PoC can be used on a real mobile device. Unfortunately, there was no time to add these checks, but they will be added in a later stadium.

4.4 Concluding questions

4.4.1 How would NVISO get this app on a device of a victim?

Should NVISO ever use this malicious app for its red team assessments, the company has two options to distribute this app; via an app store or via phishing links.

The first option, uploading the app to an app store, might be really challenging as most official app stores already have some measures against this specific kind of threat. To circumvent those measures, NVISO could encrypt and obfuscate the malicious app so that the app store would not recognize its malicious behavior. But as soon as app would be detected, the app would be deleted from the app store and NVISO account would possibly be blocked. Non-official app stores might be a solution to that. But a big disadvantage of using any app store at all is that NVISO would not be able to specifically target the employees of the company that they are doing an assessment for.

A solution to that might be the second option, sending phishing emails or text messages that contain a malicious link that automatically installs the malicious app. Using this method, only people who get the email or text message might possibly click on the link and install the malware. However, a disadvantage of this option is that it might need some additional overlays and social engineering to trick the employees enabling the malicious accessibility service that come with the app as they should not know that they just installed an app. A solution to that might be posing as a C-level colleague explaining that all employees need to install a certain privacy app to monitor their work devices. Of course, NVISO then would need the consent of the C-level person in question to be allowed to pose as that person.

To let the mobile malware remain undetected until the end of the assessment, NVISO could try to let the malware disable Google Play Protect, just like LeifAccess did in 2019.

4.4.2 Can MFA apps secure themselves against these types of attacks?

No. Both accessibility attacks and overlay attacks are almost impossible to prevent as an Android app developer.

Accessibility services are a feature of Android and thus are part of the operating system itself. Developers are not allowed and are not able to disable accessibility services as that might be considered a discrimination against disabled people.

What developers can do to protect their app against accessibility attacks, is removing all accessibility data from the app. This can be done by setting all android:contentDescription attributes to "@null". This will make it much more difficult to track the user as it will effectively remove all meta information from the app. However, displayed text and input text will still be available to the malware. To solve this, one could change every input text to a password field. Depending on the user's settings, this won't work either as Android shows the last entered character in a password field by default.

The only downside to removing all accessibility data from the app is that this will of course not only stop malware, but stop legitimate accessibility software as well from analyzing the app.

Luckily, detecting accessibility services and their capabilities is possible by using the AccessibilityManager class and its getEnabledAccessibilityServiceList method. Some downsides of this technique are the requirement of a dataset of known good services to compare against, many false positives, and of course some privacy related issues as well, since users might not want the app to know they have some disabilities.

But even overlay attacks, especially full overlays, are almost impossible to prevent. There is a research that suggests adding visual indicators to inform the user that an overlay attack is taking place. Another study is trying to find suspicious patterns during app-review to identify overlay malware.

The developers of MFA apps could try to detect such overlay attacks, but one cannot detect an overlay attack very easily. One can scan for installed apps to check if suspicious apps are installed, but this requires actively tracking mobile malware campaigns and since Android 11 it even is impossible to scan for apps that are not defined in the Android Manifest. Developers could also use accessibility services themselves to monitor which views are created by the Android OS and trigger an error if specific scenarios occur, but this could give users the idea that certain apps do require accessibility services, which would play into the hands of malware developers. But the only real feasible implementation is detecting if a screen has been obfuscated by listening for onFilterTouchEventForSecurity events, although this can lead to many false negatives and false positives. For that reason it can only be used in conjunction with other information during a risk assessment.

But even if the app succeeds in detecting an overlay attack, it might result in many false positives as most overlays are normal behavior. An example of normal overlay behavior is the Facebook Messenger chat head.

But even after detecting overlay attacks, the next steps developers can take will not help.

The first option is automatically closing the app after the detection of an overlay. This will not help at all if the malicious app is using an accessibility service. Then, the spyware can just reopen the MFA app again and try again. This would also mean that users using Messenger would not be able to open their MFA apps.

The other option is showing a warning to the users. This might be useful, unless the spyware is using a full overlay. Then, the MFA app is not the top-level view and users might not even see the warning.

4.4.3 Are these MFA apps still secure then?

Yes. They do store tokens in a secure way and do not send tokens on-air, like SMS tokens. They also use a principle known as Time-based One Time Passwords (TOTP).

MFA apps cannot change anything about accessibility services being a feature and cannot do anything about installed apps that might be malicious. It is up to the user to be cautious when installing other apps.

The user is, and forever will be, the weakest link when it comes to cybersecurity.

One thing that MFA apps can do is securing their app with a passcode, fingerprint (Touch ID for iOS), or Face ID (only on iOS). This is already supported by Microsoft Authenticator and Authy, although it is not enabled by default. Google Authenticator does not support this functionality and thus might be considered less secure.

However, securing the app with a passcode or fingerprint will not stop malware completely as they can use overlay attacks to trick the user into filling in his passcode or scanning his fingerprint.

But although MFA apps and other legit apps cannot stop this type of malicious apps, Google might. Since 2017, Google is working on a strategy to remove apps from the Play Store that use the accessibility services for anything except helping disabled users. In 2019, Google announced that they would deprecate the overlay functionality of accessibility services in Android 10. Additionally, Android will automatically revoke the accessibility permission if it is requested after 30 seconds. However, should the malicious app request permissions immediately after launch, this remediation would be useless. In 2020, Google has removed the deprecated status of the overlay permission in Android 11. Since Android 10, some functionality of the old overlay API called SYSTEM_ALERT_WINDOW is replaced by the new Bubbles API. In a future release, the SYSTEM_ALERT_WINDOW API will be fully deprecated and replaced by the Bubbles API.

The only problem that then will remain is that not all Android devices will ever be upgraded to the latest Android version. For that reason, many devices will remain vulnerable to these types of attacks.

5 Guidelines

Mobile malware can be a real threat. Some types are just annoying, like adware that spams the user with a flood of advertisements. Others are much more dangerous, like SMS trojans that send SMSs to premium numbers so users get huge phone bills, or spyware that can steal sensitive information from the user or the user's company.

Fortunately, there are some ways to mitigate this threat, although there is no way to completely prevent mobile malware attacks. In this chapter, some measures are listed that one can take as a user, a company, or a developer.

5.1 Guidelines for users

General security

- **Do not root or jailbreak your device;**
- **Disable the developer mode on Android;**
- **Encrypt sensitive data;**
- **Lock your device with a strong password, a fingerprint, or a Touch ID;**
- **Update your device regularly;**
- **Use unique passwords for every account;**
- **Use MFA for all important accounts;**
- **Use the "Log in with Apple" feature whenever possible on iOS.**

Do not root or jailbreak your device. These techniques remove a lot of the device's built-in security and leave you more vulnerable to attacks. If you do root or jailbreak your device, make sure it still gets system updates.

Disable the developer mode on Android. With the developer mode on, attackers might be able to install spyware via USB. That way, attackers can circumvent the security measures from app stores.

Encrypt sensitive data. By encrypting your sensitive data malware cannot use it if it gets access to it. Be sure to encrypt corporate emails and documents.

Lock your device with a strong password, a fingerprint, or a Touch ID. Do not use PIN codes or symbol passwords as those are considered less secure because they are more brute-forceable. Even face recognition is considered less secure than using a fingerprint because face similarities are less random and because face recognition needs some fuzziness so users can shave their head or beard and can still be recognized. But, fortunately, malware cannot bypass face recognition or fingerprints. But, when using one of those, one does have to be careful of social engineering techniques.

Update your device regularly. By updating your device regularly to the latest version all available patches get installed. That way, attackers cannot exploit known vulnerabilities of your device to get malware on it.

Use unique passwords for every account. That way, when one account gets hacked, hackers will not be able to pivot to another account using the same credentials.

Use MFA for all important accounts. Without MFA, spyware only has to get access to your passwords. When using keylogger functionalities, this is easily done. A big advantage of MFA is that MFA tokens are only temporary valid. When you are given options on how to configure the MFA, choose dedicated apps like Google Authenticator or Microsoft Authenticator over SMS tokens or phone calls, as the last two can be intercepted by attackers.

Use the "Log in with Apple" feature whenever possible on iOS. Many apps and websites work with Log in with Apple to configure accounts. Signing in with Apple limits the amount of information shared about you. The feature works with the Apple ID you already have and employs secure MFA. If this option is not possible, you can let your iPhone generate a strong password for you without you having to memorize it.

Additional security

- **Perform mobile device vulnerability scanning;**
- **Use third-party virus scanners as an additional security layer;**

Perform mobile device vulnerability scanning. Install vulnerability scanners like SecurityMetrics Mobile.

Use third-party virus scanners as an additional security layer. Let those virus scanners run on top of the built-in security features of the device's app store. Examples are Avast Mobile Security & Antivirus, Bitdefender Antivirus, and ESET Mobile Security.

Prevent malware installation

- **Do not open links in suspicious emails or text messages;**
- **Do not download apps from third-party app stores if not necessary;**
- **Be careful when downloading an app;**
- **Do not install banking apps via an app store.**

Do not open links in suspicious emails or text messages. If you do not the sender, it is even more secure to delete the message without opening it because sometimes, opening the message is enough to get infected.

Do not download apps from third-party app stores if not necessary. These third-party marketplaces are often less secure and contain more malware than the official marketplaces of the operating system.

Be careful when downloading an app. Check how many times the app has been downloaded. Low downloads counts might suggest it is malware. Also be sure to check the reviews, certainly the bad ones, as those normally are written by legitimate users. Check if the permissions it asks for are relevant.

Do not install banking apps via an app store. Banking apps are very popular with malware. Even the official marketplaces are full of them. Ensure you install the correct app by following the link on the official website of the bank.

Wireless connections

- **Disable interfaces that are not in use;**
- **Set Bluetooth-enabled devices to non-discoverable;**
- **Avoid unknown Wi-Fi networks and public Wi-Fi hotspots;**
- **Use a VPN to access and share information over public Wi-Fi networks.**

Disable interfaces that are not in use. Enable Wi-Fi, Bluetooth, and infrared only when using them. Attackers might exploit those interface to upload malware on your device.

Set Bluetooth-enabled devices to non-discoverable. That way, nearby attackers or infected will not get alerted to target you. In non-discoverable mode, your device is invisible to other unauthenticated devices.

Avoid unknown Wi-Fi networks and public Wi-Fi hotspots. Attackers might use this networks or hotspots to attack your device.

Use a VPN to access and share information over public Wi-Fi networks. A Virtual Private Network ensures online privacy and anonymity by creating a private network from a public internet connection. VPNs mask the IP address of the device to make online actions virtually untraceable. On top of that, VPNs also establish secure and encrypted connections.

Mitigate accessibility attacks

- **Hide your notifications;**
- **Hide your passwords.**

Hide your notifications. Ensure the content of your messages are not readable on the lock screen. Some messages may in fact contain sensitive information, like MFA tokens. Malware abusing accessibility services might read these messages to sent that sensitive information back to the C2 server of the hacker.

Hide your passwords. By default, your mobile device shows you the last entered character of your password. Malware abusing accessibility services might read this characters one by one to later reconstruct your password to get access to your account. Of course, this option is needed for users who actually need accessibility services and can be re-enabled by malicious accessibility services.

5.2 Guidelines for companies

General security

- **Train your employees;**
- **Enforce strict mobile device policies;**

Train your employees. Ensure your employees know what types of malware exist and how they can avoid them. Also train them on phishing and tell them to avoid public networks when using their work device outside of the office.

Enforce strict mobile device policies. Enforce those policies on all mobile devices, even if employees are allowed to use their own devices. This can be done by using Mobile Device Management software. On the next page, there are some of the most important examples of mobile device policies.

- Password policies
  - Enforce strong passwords or the use of fingerprints;
  - Enforce Multi-Factor Authentication;
  - Enforce the use of encrypted password stores for user-saved passwords;
  - Enforce all passwords to be unique from all credentials within the company;
  - Forbid the use of PIN codes or symbol passwords.
- System security policies
  - Enforce updates and patches as soon as they are available;
  - Enforce mobile devices to use virus scanners that scan the device regularly;
  - Enforce the remote wipe function on all mobile devices;
  - Prohibit users from installing apps on their own or from untrusted sources;
  - Prohibit users from rooting or jailbreaking their device;
  - Prohibit users from using personal accounts on their work device;
  - Use whitelists of allowed sites.
- Network security policies
  - Employ a least privilege model;
  - Only allow devices managed by IT to connect to the corporate network.

Additional security

- **Create backups routinely of all important data;**
- **Use VPNs for remote connections.**

Create backups routinely of all important data. This minimalizes the data loss in case a mobile security threat destroys your data or makes it inaccessible. To help make this process easier, consider a solution that creates automated backups of mobile data.

Use VPNs for remote connections. A Virtual Private Network ensures online privacy and anonymity by creating a private network from a public internet connection. VPNs mask the IP address of the device to make online actions virtually untraceable. On top of that, VPNs also establish secure and encrypted connections.

5.3 Guidelines for developers

General security

- **Avoid processing sensitive data at the client-side;**
- **Do mobile app vulnerability scanning;**
- **Detect malicious activities;**
- **Leverage platform security;**
- **Use the OWASP Mobile AppSec Verification Standard checklist;**
- **Use the OWASP Mobile Top 10 to mitigate the most common risk on your app.**

Avoid processing sensitive data at the client-side. Assume the users can be tricked and the mobile device can be hijacked. Let the server process sensitive data and give users the least privileges possible.

Do mobile app vulnerability scanning. Install vulnerability scanners like ImmuniWeb MobileSuite, Zed Attack Proxy (ZAP), and QARK to scan the app before deploying it.

Detect malicious activities. Ensure most work is done on the server-side. That way, better insights are possible. Double-check every client request for malicious activities and abnormal activity patterns.

Leverage platform security. Just because the client is mobile does not mean the server must not be hardened. Thus, ensure unnecessary services are turned off and employ a strong configuration and patch management.

Use the OWASP Mobile Application Security Checklist for Android and iOS. This checklist can be found as an Excel file[2] in English, French, Japanese, Korean, and Spanish or as a GitBook[3] in English only. The Excel file has different checklists for Android and iOS, while the GitBook uses just one list.

Use the OWASP Mobile Top 10 to mitigate the most common risk on your app. This list can be found here[4] on OWASP's website.

Additional security

- **Monitor app stores;**
- **Support and recommend MFA;**
- **Use Interactive Application Security Testing (IAST);**
- **Use Runtime Application Self-Protection (RASP).**

Monitor app stores. Look in both official and unofficial app stores for fake clones of your app. Those are very likely malware. Make sure to report those to the app stores and inform the users of this malicious version.

Support and recommend MFA. Make sure MFA is supported by the app and recommend it to the users. This strengthens the authentication of the app.

Use Interactive Application Security Testing (IAST). IAST is focused on identifying vulnerabilities within the apps. It combines Static Application Testing tools (SAST) and Dynamic Application Testing tools (DAST) approaches. IAST thus both identifies issues dynamically during production, like DAST, and evaluates code, like SAST.

Use Runtime Application Self-Protection (RASP). RASP takes advantage of insight into an app's internal data and state to enable it to identify threats at runtime that may have otherwise been overlooked by other security solutions like firewalls that do not have any contextual awareness. RASP makes it more difficult for attackers to republish the app.

[2] https://github.com/OWASP/owasp-mstg/tree/master/Checklists
[3] https://mobile-security.gitbook.io/masvs
[4] https://owasp.org/www-project-mobile-top-10

Mitigate accessibility and overlay attacks

- **Detect and log overlay attacks and show warnings to the user;**
- **Detect accessibility attacks and their capabilities;**
- **Remove all accessibility data from the app;**
- **Change input fields into password fields;**

Detect and log overlay attacks and show warnings to the user. This can be done by listening for onFilterTouchEventForSecurity events, although this can lead to many false negatives and false positives. For that reason, it can only be used in conjunction with other information during a risk assessment and to show warnings to the user. Unfortunately, if the malware causes a full overlay, this warning will not help as the overlay will remain on top.

Detect accessibility attacks and their capabilities. This can be done by using the AccessibilityManager class and its getEnabledAccessibilityServiceList method. Some downsides of this technique are the requirement of a dataset of known good services to compare against, many false positives, and of course some privacy related issues as well, since users might not want the app to know they have some disabilities.

Remove all accessibility data from the app. This can be done by setting all android:contentDescription attributes to “@null”. This will make it much more difficult to track the user as it will effectively remove all meta information from the app. One thing to consider with this measure is that this will of course not only stop malware, but stop legitimate accessibility software as well from analyzing the app.

Change input fields into password fields. That way, malware abusing accessibility services will not be able to read displayed texts. Depending on the user’s settings, this won’t work either as Android shows the last entered character in a password field by default.

6 Conclusion

Malware can get on a victim's device in six different ways. Users can download them from an app store, official or otherwise. But users can also download malware from other sources, like social media, backups, and phishing emails or text messages. Then, there is also the possibility that users download legitimate apps that use malicious third-party services or use insecure websites or malicious browser plugins. Lastly, hackers can also exploit OS vulnerabilities or can attack the supply chain of the app itself to get their malware on the device of a victim.

When finally installed on a victim's device, malware might get access to sensitive information that is stored on that device. This can be done through rooting or jailbreaking as this would give the malware full control over the device, or by asking numerous permissions, like access to the accounts or the microphone of the device. But malware nowadays is using more advanced techniques to get access to data. One of those techniques is using accessibility services, among other things.

Mobile devices can hold a lot of data, depending on its usage. Spyware could steal information about the use of the device, or stored data like contacts, photos, locations, etc. But should the device be a corporate one, the spyware might also get access to corporate emails and documents. The impact and the risk of spyware attacks is thus very personal and different for each case. It all depends on what the user is using the device for and what value the user gives to his data.

Android and iOS are already doing numerous things to protect their devices and operating systems against mobile malware. Some measures are done by both Android and iOS, while others are specific to the operating systems.

Both operating systems can thus be considered equally secure, although Android is more often the target of malware attacks. This is in fact for three different reasons. Firstly, because Android allows apps to be installed from outside the app store. Secondly, because its flexibility allows much more malware in the Google Play Store. And lastly, because there are more Android users than iOS users and thus more possible targets.

The most difficult step of stealing information from a victim's device is getting the malware installed. This is mainly because uploading it to an app store is nearly impossible because of all its security measures. The other option is then sending a malicious link via phishing, but then the attacker depends on the ignorance of the user.

But as soon as the malware is installed, stealing information via accessibility services is child's play, which has been proven by the Proof of Concept described in this thesis.

Unfortunately, apps cannot really protect themselves against accessibility attacks, as accessibility services are a feature of the operating system to help disabled users. Fortunately, there are some ways to detect accessibility attacks and to protect oneself against mobile malware in general. The protection against mobile malware can be done both on a technical and a non-technical level as a users, as a company, and as a developer.

7 Critical reflection

7.1 Critical self-reflection

I have evolved a lot as a person since the beginning of my research. I have learned to just go for something if it really interests me, no matter how challenging it might be. As long as you accept that you are not perfect and you dare to ask questions, everything is possible if you really want it.

On a professional level, I learned how the accessibility services of Android work and what their possibilities are. It was certainly not easy as there is much but unclear documentation on those accessibility services. You get lost very easily if you are a total beginner. On top of those accessibility services, I learned how to set up an app that could communicate with a server using three program languages: Java, JSON, and PHP. All of them I had already used before, but this was my first time combining the three.

But most importantly, I am very proud of the results of this thesis and I will definitely continue my research in private as well. I will definitely try to improve my use case by adding some functionality to give different mobile devices different commands. On top of that, I will also try to improve the app by giving the app some overlay views when it is stealing tokens. And of course, I would like to see what more I can do with such accessibility services and how long this PoC will remain working on the different versions of Android.

7.2 Possibilities on further research

There are quite some possibilities on further research. Both on the topic of the research as on the specific use case.

This thesis focused solely on Android and iOS. In further research, one could try to focus on lesser known mobile operating systems, like Chrome OS and Windows. One could even compare those lesser know OSs with Android or iOS or could compare iPadOS with iOS.

This thesis also focuses more on how mobile spyware generally works and what its impact could be. In further research, one could try to focus on finding detection and recovery methods.

For the use case there are even more opportunities for further research.

Firstly, the use case of this thesis is a mobile spyware app for Android. Of course, one could try to make the exact same PoC for iOS. It will probably be a lot harder than on Android as iOS is more strict, but that would only make it more relevant if one actually succeeds. And even if it should turn out that this use case would not be possible, it would still be extremely relevant as this would proof that iOS is way more secure than Android on that front.

Secondly, this PoC supports only three MFA apps. Of course, there are way more than that. One could extend the functionalities of this PoC so it would support other MFA apps as well. Examples of those MFA apps are Authenticator Plus, LastPass Authenticator, and Duo Mobile.

Thirdly, this PoC has only one hardcoded command and one result file for all possible devices that connect to the server. This PoC can thus be improved be letting the hacker change the command for every connected device dynamically. The devices that are connected would then send their device ID along with the results of the attack to the server. The server can then put the result in different files for each device or could just add that ID to every result.

On top of that, one could extend the functionalities of the PoC by also supporting features to steal tokens from text messages. Stealing from those text messages on itself should be child's play, but one could try to force the fallback from tokens via phone calls to text messages. This would be very relevant as accessibility service probably would not be able to get the token from the phone call. And certainly not without the user noticing. Forcing a fallback could be a solution to that.

One could also try to capture the PIN code by which the Microsoft Authenticator or Authy might be protected. That way, the malicious service will only have to wait for some user interaction once. After that, the attack might again be run independendly.

Lastly, other possibilities include finding a way to capture the duration in which the token will remain valid for the Google Authenticator, and showing an overlay view for the duration of the attack. Finally, one could try to find a way to also support Android 6, 7, and 11. Should one have to conclude that this is not possible, one could try to detect the Android version and ensure that the app would only run if the Android version is supported.

Resource & literature list

Citations

[1] A. S. Gillis, “Definition – application,” *searchsoftwarequality.techtarget.com*, para. 1, Sept., 2018. [Online]. Available: https://searchsoftwarequality.techtarget.com/definition/application. [Accessed March 5, 2021].

[2] R. L. Midrack, “What is a third-party app?” *lifewire.com*, para. 1, Feb. 4, 2020. [Online]. Available: https://www.lifewire.com/what-is-a-third-party-app-4154068. [Accessed March 5, 2021].

[3] “What are applications?,” *indeed.com*, para. 8-9, Feb. 4, 2020. [Online]. Available: https://www.indeed.com/career-advice/finding-a-job/what-are-applications. [Accessed March 5, 2021].

[4] M. Karch, “A beginner’s guide to mobile apps,” *lifewire.com*, para. 5-6, Dec. 2, 2020. [Online]. Available: https://www.lifewire.com/what-are-apps-1616114. [Accessed March 5, 2021].

[5] Wikipedia, “Android (operating system),” *Wikipedia*, para. 1, Mar. 7, 2021. [Online]. Available: https://en.wikipedia.org. [Accessed March 10, 2021].

[6] Wikipedia, “iOS,” *Wikipedia*, para. 1, Mar. 4, 2021. [Online]. Available: https://en.wikipedia.org. [Accessed March 10, 2021].

[7] S. Malenkovich, “Rooting and jailbreaking: what can they do, and how do they affect security?” *kaspersky.com*, para. 3, May 31, 2013. [Online]. Available: https://www.kaspersky.com/blog/rooting-and-jailbreaking/1979. [Accessed March 10, 2021].

[8] S. Malenkovich, “Rooting and jailbreaking: what can they do, and how do they affect security?” *kaspersky.com*, para. 8, May 31, 2013. [Online]. Available: https://www.kaspersky.com/blog/rooting-and-jailbreaking/1979. [Accessed March 12, 2021].

[9] K. Baker, “What is mobile malware?” *crowdstrike.com*, para. 6, Jan. 25, 2021. [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/malware/mobile-malware. [Accessed March 16, 2021].

[10] Android, “Hardware-backed Keystore,” *source.android.com*, para. 16, n.d. [Online]. Available: https://source.android.com/security/keystore. [Accessed May 26, 2021].

References

Glossary

"Open source." nl.wikipedia.org. https://nl.wikipedia.org/wiki/Open_source. [Accessed June 3, 2021].

"Read-only memory." *en.wikipedia.org*. https://en.wikipedia.org/wiki/Read-only_memory. [Accessed June 3, 2021].

"Firmware." *en.wikipedia.org*. https://en.wikipedia.org/wiki/Firmware. [Accessed May 31, 2021].

"Original equipment manufacturer." *nl.wikipedia.org*. https://nl.wikipedia.org/wiki/Original_equipment_manufacturer. [Accessed June 3, 2021].

"Overlay attack." *encyclopedia.kaspersky.com*. https://encyclopedia.kaspersky.com/glossary/overlay-attack. [Accessed May 31, 2021].

R. Larsen. "What are command-and-control (C2) callbacks?" *dualog.com*. https://www.dualog.com/blog/what-are-command-and-control-c2-callbacks. [Accessed May 31, 2021].

S. Munot. "Toast Notification or Dialog Box?" *uxplanet.org*. https://uxplanet.org/toast-notification-or-dialog-box-ae32ad53106d. [Accessed March 29, 2021].

"Social Engineering." *imperva.com*. https://www.imperva.com/learn/application-security/social-engineering-attack. [Accessed June 2, 2021].

"What is a REST API?" *redhat.com*. https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Accessed June 2, 2021].

A. Phadke. "SAST vs. DAST: What's the best method for application security testing?" *synopsys.com*. https://www.synopsys.com/blogs/software-security/sast-vs-dast-difference. [Accessed June 3, 2021].

2.2 What are mobile applications?

M. Karch. "A beginner's guide to mobile apps." *lifewire.com*. https://www.lifewire.com/what-are-apps-1616114. [Accessed March 5, 2021].

K. Koreman. Mobile Security. (2020, Summer). Basic Android. Bruges: Hogeschool West-Vlaanderen (Howest)

2.3 How are mobile apps installed on your device?

Pallavi. "A Descriptive List Of Alternative App Stores For 2021." *mobileappdaily.com*. https://www.mobileappdaily.com/app-stores-list. [Accessed May 19, 2021].

B. King. “Why You Should Replace Google Play With an Alternative App Store.” *makeuseof.com*. https://www.makeuseof.com/tag/replace-google-play-app-store. [Accessed May 19, 2021].

“Google China.” *en.wikipedia.org*. https://en.wikipedia.org/wiki/Google_China. [Accessed May 19, 2021].

2.4 What are the main differences between Android and iOS?

“Android vs iOS.” *diffen.com*. https://www.diffen.com/difference/Android_vs_iOS. [Accessed March 10, 2021].

“Wear Check.” *wearos.google.com*. https://wearos.google.com/wearcheck. [Accessed March 19, 2021].

2.4.1 Open source and availability

J. Hindy. “Google Play Services - Everything you need to know” *androidauthority.com*. https://www.androidauthority.com/google-play-services-1094356. [Accessed May 19, 2021].

M. van ‘t Klaphek. “Uitleg: Waarom Google Play Services zo belangrijk is voor Android (en Huawei).” *androidplanet.nl*. https://www.androidplanet.nl/nieuws/google-play-services. [Accessed March 19, 2021].

“Custom rom: alles over alternatieve Android-versies.” *androidplanet.nl*. https://www.androidplanet.nl/spotlight/custom-rom. [Accessed March 12, 2021].

“Is flashing ROMs legal? Well I went straight to the big guys.” *forum.xda-developers.com*. https://forum.xda-developers.com/t/is-flashing-roms-legal-well-i-went-straight-to-the-big-guys.598449. [Accessed March 19, 2021].

Madhumintha. “Android Stock Rom VS Custom Rom: Which one is better?” *droidtechknow.com*. https://droidtechknow.com/tips-and-tricks/android-stock-rom-vs-custom-rom. [Accessed March 19, 2021].

“Bootloader: What you need to know about the system boot manager” *ionos.com*. https://www.ionos.com/digitalguide/server/configuration/what-is-a-bootloader. [Accessed May 19, 2021].

“Why is root access required for a custom ROM?” *android.stackexchange.com*. https://android.stackexchange.com/questions/37877/why-is-root-access-required-for-a-custom-rom. [Accessed March 19, 2021].

U. Gupta. “Why do some mobile companies refuse to unlock bootloaders, like Huawei and Realme?.” *quora.com*. https://www.quora.com/Why-do-some-mobile-companies-refuse-to-unlock-bootloaders-like-Huawei-and-Realme. [Accessed May 19, 2021].

2.4.2 Rooting vs. jailbreaking

C. Hoffman. “What’s the difference between jailbreaking, rooting, and unlocking?” *howtogeek.com*. https://www.howtogeek.com/135663/htg-explains-whats-the-difference-between-jailbreaking-rooting-and-unlocking. [Accessed March 10, 2021].

J. Snyder. “What Are the Security Risks of Rooting Your Smartphone?” *insights.samsung.com*. https://insights.samsung.com/2019/05/29/what-are-the-security-risks-of-rooting-your-smartphone. [Accessed March 12, 2021].

2.4.3 Default security settings

M. Atkinson. “Chapter 3: An Analysis of Android App Permissions.” *pewresearch.org*. https://www.pewresearch.org/internet/2015/11/10/an-analysis-of-android-app-permissions. [Accessed April 23, 2021].

Souvik. “Android vs. iOS: Which One Owns a Better App Permissions System?” *rswebsols.com*. https://www.rswebsols.com/tutorials/software-tutorials/android-ios-better-app-permissions-system. [Accessed March 26, 2021].

S. Cuthbertson. “Sharing what’s new in Android Q.” *blog.google*. https://blog.google/products/android/android-q-io. [Accessed March 16, 2021].

A. Siddiqui. “Everything you need to know about Android’s Project Mainline” *xda-developers.com*. https://www.xda-developers.com/android-project-mainline-modules-explanation. [Accessed April 23, 2021].

“ESET Becomes Founding Member of Google’s App Defense Alliance.” eset.com. https://www.eset.com/int/about/newsroom/press-releases/company/eset-becomes-founding-member-of-googles-app-defense-alliance-eset-to-proactively-protect-mobile-ap-4. [Accessed March 17, 2021].

2.5 What types of mobile malware exist?

“Threat Intelligence Report 2020,” *Nokia*, Espoo, Uusimaa, Finland, Rep. CID210088, Feb. 2021. [Accessed March 16, 2021].

“Malware for iOS.” *theiphonewiki.com*. https://www.theiphonewiki.com/wiki/Malware_for_iOS. [Accessed March 29, 2021].

“Mobile Ad Fraud | Definition.” *adjust.com*. https://www.adjust.com/glossary/mobile-ad-fraud. [Accessed March 16, 2021].

“Snaptube.” *wikipedia.com*. https://en.wikipedia.org/wiki/Snaptube. [Accessed March 16, 2021].

“Adware.” *malwarebytes.com*. https://nl.malwarebytes.com/adware. [Accessed March 16, 2021].

D. Palmer. “Mobile security: These seven malicious apps have been downloaded by 2.4m Android and iPhone users.” *zdnet.com*. https://www.zdnet.com/article/mobile-security-these-seven-malicious-apps-have-been-downloaded-by-2-4m-android-and-iphone-users. [Accessed March 17, 2021].

“First SMS Trojan detected for smartphones running Android.” *kaspersky.com*. https://www.kaspersky.com/about/press-releases/2010_first-sms-trojan-detected-for-smartphones-running-android. [Accessed March 16, 2021].

“Smartphones en apps.” *autoriteitpersoonsgegevens.nl*. https://autoriteitpersoonsgegevens.nl/nl/onderwerpen/internet-telefoon-tv-en-post/smartphones-en-apps. [Accessed March 15, 2021].

D. Palmer. “Android security: Six more apps containing Joker malware removed from the Google Play Store.” *zdnet.com*. https://www.zdnet.com/article/android-security-six-more-apps-containing-joker-malware-removed-from-the-google-play-store. [Accessed March 16, 2021].

P. Wagenseil. “Nasty malware attacks iPhones and Android – what to do now.” *tomsguide.com*. https://www.tomsguide.com/news/wroba-mobile-trojan-ios-android. [Accessed March 16, 2021].

“Mobile malware.” *en.wikipedia.org*. https://en.wikipedia.org/wiki/Mobile_malware. [Accessed March 16, 2021].

“A Whale of a Tale: HummingBad Returns.” *blog.checkpoint.com*. https://blog.checkpoint.com/2017/01/23/hummingbad-returns. [Accessed March 16, 2021].

T. Seals. “HummingWhale Breaches the Surface of Google Play.” *infosecurity-magazine.com*. https://www.infosecurity-magazine.com/news/hummingwhale-google-play. [Accessed March 16, 2021].

3.2 How does mobile malware get on your phone?

D. Page. “5 Ways Your Mobile Device Can Get Malware.” *securitymetrics.com*. https://www.securitymetrics.com/blog/5-ways-your-mobile-device-can-get-malware. [Accessed March 25, 2021].

E. Shein. “Almost half of mobile malware are hidden apps.” *techrepublic.com*. https://www.techrepublic.com/article/almost-half-of-mobile-malware-are-hidden-apps. [Accessed March 19, 2021].

R. Samani. “McAfee Mobile Threat Report – Mobile Malware Is Playing Hide and Steal,” *McAfee*, Santa Clara, CA, USA, Rep. Q1 - 2020, 2019.

C. Cimpanu. “Play Store identified as main distribution vector for most Android malware.” *zdnet.com*. https://www.zdnet.com/article/play-store-identified-as-main-distribution-vector-for-most-android-malware. [Accessed March 18, 2021].

P. Hannay. “Explainer: how malware gets inside your apps.” *theconversation.com*. https://theconversation.com/explainer-how-malware-gets-inside-your-apps-79485. [Accessed March 22, 2021].

J. Finkle. “Apple's iOS App Store suffers first major attack” *reuters.com*. https://www.reuters.com/article/us-apple-china-malware-idUSKCN0RK0ZB20150920. [Accessed March 19, 2021].

“XcodeGhost.” *wikipedia.com*. https://en.wikipedia.org/wiki/XcodeGhost. [Accessed March 19, 2021].

Z. Whittaker. “A powerful spyware app now targets iPhone owners” *techcrunch.com*. https://techcrunch.com/2019/04/08/iphone-spyware-certificate. [Accessed March 29, 2021].

“Apple suspends another spyware app from App Store.” *livemint.com*. https://www.livemint.com/technology/apps/apple-suspends-another-spyware-app-from-app-store-1554801819996.html. [Accessed March 29, 2021].

C. Castillo. “Android/TimpDoor Turns Mobile Devices Into Hidden Proxies.” *mcafee.com*. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/android-timpdoor-turns-mobile-devices-into-hidden-proxies. [Accessed March 19, 2021].

D. Palmer. “Now this Android spyware poses as a privacy tool to trick you into downloading.” *zdnet.com*. https://www.zdnet.com/article/now-this-android-spyware-poses-as-a-privacy-tool-to-trick-you-into-downloading. [Accessed March 21, 2021].

B. Barret. “How 18 Malware Apps Snuck Into Apple's App Store.” *wired.com*. https://www.wired.com/story/apple-app-store-malware-click-fraud. [Accessed March 19, 2021].

M. Kan. “Malware Discovered in Popular Android App CamScanner.” *pcmag.com*. https://www.pcmag.com/news/malware-discovered-in-popular-android-app-camscanner. [Accessed March 22, 2021].

K. Haider. “3 Android Browsers With Extensions Support To Install Your Favorite Extensions.” *gtricks.com*. https://www.gtricks.com/android/3-android-browsers-with-extensions-support-to-install-your-favorite-extensions. [Accessed May 19, 2021].

D. Goodin. “Google Chrome extensions with 500,000 downloads found to be malicious.” *arstechnica.com*. https://arstechnica.com/information-technology/2018/01/500000-chrome-users-fall-prey-to-malicious-extensions-in-google-web-store. [Accessed May 19, 2021].

A. Perekalin. “Why you should be careful with browser extensions.” *kaspersky.com*. https://www.kaspersky.com/blog/browser-extensions-security/20886. [Accessed May 19, 2021].

E. Rejthar. “Search for malicious code among add-ons.” *blog.nic.nz*. https://blog.nic.cz/2020/11/19/hledani-skodliveho-kodu-mezi-doplnky. [Accessed May 19, 2021].

“Man-in-the-browser.” *wikipedia.com*. https://en.wikipedia.org/wiki/Man-in-the-browser. [Accessed March 22, 2021].

“Drive-by download.” *wikipedia.com*. https://en.wikipedia.org/wiki/Drive-by_download. [Accessed March 22, 2021].

C. Cimpanu. “Seven mobile browsers vulnerable to address bar spoofing attacks.” https://www.zdnet.com/article/seven-mobile-browsers-vulnerable-to-address-bar-spoofing-attacks. [Accessed March 22, 2021].

“Browser exploit.” en.*wikipedia.org*. https://en.wikipedia.org/wiki/Browser_exploit. [Accessed March 22, 2021].

C. Singh. “Android: Most Vulnerable OS Of 2019; Followed By Linux & Windows 10.” *fossbytes.com*. https://fossbytes.com/android-most-vulnerable-os-2019-followed-by-linux-windows-10. [Accessed March 22, 2021].

K. O’Flaherty. “These Scam Apple Apps Use The Fingerprint Scanner To Steal Cash.” *forbes.com*. https://www.forbes.com/sites/kateoflahertyuk/2018/12/03/scam-apple-apps-use-the-fingerprint-scanner-to-steal-cash/?sh=199496c111ab. [Accessed March 22, 2021].

L. Siewierski. “PHA Family Highlights: Triada.” *security.google.com*. https://security.googleblog.com/2019/06/pha-family-highlights-triada.html. [Accessed March 22, 2021].

3.3 How can malware access your data?

“DroidKungFu.” *en.wikipedia.org*. https://en.wikipedia.org/wiki/DroidKungFu. [Accessed March 29, 2021].

A. Barbaschow. “More Google Play apps infected with Brain Test malware: Lookout.” *zdnet.com*. https://www.zdnet.com/article/more-google-play-apps-infected-with-brain-test-malware-lookout. [Accessed March 29, 2021].

“WireLurker.” *en.wikipedia.org*. https://en.wikipedia.org/wiki/Wirelurker. [Accessed March 29, 2021].

F. Ruiz. “Android/LeifAccess.A is the Silent Fake Reviewer Trojan.” *mcafee.com*. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/android-leifaccess-a-is-the-silent-fake-reviewer-trojan. [Accessed March 29, 2021].

“Shedun.” *en.wikipedia.org*. https://en.wikipedia.org/wiki/Shedun. [Accessed May 19, 2021].

3.4.1 Is it possible and has it happened before?

“Andr/Wroba-B.” *sophos.com*. https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/Andr~Wroba-B/detailed-analysis.aspx. [Accessed March 29, 2021].

“An Invasive Spyware Attack on Military Mobile Devices.” *blog.checkpoint.com*. https://blog.checkpoint.com/2018/07/05/an-invasive-spyware-attack-on-military-mobile-devices. [Accessed March 29, 2021].

A. Meyers. “Danger Close: Fancy Bear Tracking of Ukrainian Field Artillery Units.” *crowdstrike.com*. https://www.crowdstrike.com/blog/danger-close-fancy-bear-tracking-ukrainian-field-artillery-units. [Accessed March 29, 2021].

"What is Spyware?" *avg.com*. https://www.avg.com/en/signal/what-is-spyware. [Accessed March 29, 2021].

M. Adams. "GO Keyboard is spying on millions of users according to researchers." *androidauthority.com*. https://www.androidauthority.com/go-keyboard-caught-spying-802270. [Accessed March 29, 2021].

L. Stefanko. "First-of-its-kind spyware sneaks into Google Play." *welivesecurity.com*. https://www.welivesecurity.com/2019/08/22/first-spyware-android-ahmyth-google-play. [Accessed March 29, 2021].

C. Cimpanu. "Google details its three-year fight against the Bread (Joker) malware operation" *zdnet.com*. https://www.zdnet.com/article/google-details-its-fight-against-the-bread-joker-malware-operation. [Accessed March 31, 2021].

"Exodus: New Android Spyware Made in Italy." *securitywithoutborders.org*. https://securitywithoutborders.org/blog/2019/03/29/exodus.html. [Accessed March 29, 2021].

A. Yaswant. "New Advanced Android Malware Posing as System Update" *blog.zimperium.com*. https://blog.zimperium.com/new-advanced-android-malware-posing-as-system-update. [Accessed March 29, 2021].

3.5.1 How do Android's separate APIs work?

"Android Hidden APIs." *github.com*. https://github.com/anggrayudi/android-hidden-api. [Accessed April 1, 2021].

A. Sinicki. "How to use a web API from your Android app." *androidauthority.com*. https://www.androidauthority.com/use-web-api-android-1152645. [Accessed April 1, 2021].

"Overview of Google Play services." *developers.google.com*. https://developers.google.com/android/guides/overview. [Accessed April 1, 2021].

T. F. Bissyandé, Y. Le Traon, & J. Klein. "Accessing Inaccessible Android APIs: An Empirical Study." thesis, IDC for Security, Univ. Luxembourg, Esch-sur-Alzette, Luxembourg, 2016. [Online]. Available: https://core.ac.uk/download/pdf/78371334.pdf.

"Device compatibility overview." *developer.android.com*. https://developer.android.com/guide/practices/compatibility. [Accessed April 1, 2021].

3.5.2 How does Android's Google Play Protect work?

"Help protect against harmful apps with Google Play Protect." *support.google.com*. https://support.google.com/googleplay/answer/2812853?hl=en. [Accessed April 1, 2021].

"Google Play Protect." *developers.google.com*. https://developers.google.com/android/play-protect. [Accessed April 1, 2021].

"Cloud-based protections." *developers.google.com*. https://developers.google.com/android/play-protect/cloud-based-protections. [Accessed April 1, 2021].

"On-device protections." *developers.google.com*. https://developers.google.com/android/play-protect/client-protections. [Accessed April 1, 2021].

"Here's how well 17 Android Security Apps Provide Protection." *av-test.org*. https://www.av-test.org/en/news/here-s-how-well-17-android-security-apps-provide-protection. [Accessed April 1, 2021].

"The big secret behind Google Play Protect on Android." *computerworld.com*. https://www.computerworld.com/article/3210587/google-play-protect-android.html. [Accessed April 1, 2021].

3.5.3 How do the permissions of Android work?

"Permissions on Android." *developer.android.com*. https://developer.android.com/guide/topics/permissions/overview. [Accessed May 19, 2021].

"Manifest.permission." *developer.android.com*. https://developer.android.com/reference/android/Manifest.permission. [Accessed May 19, 2021].

"Android permissions for system developers." *android.googlesource.com*. https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/permission/Permissions.md. [Accessed May 19, 2021].

3.5.4 How do Android and iOS encrypt their data?

E. Pederson & H. Kawakami. "Is Android disk encryption as strong as the encryption used for iOS?" *quora.com*. https://www.quora.com/Is-Android-disk-encryption-as-strong-as-the-encryption-used-for-iOS. [Accessed May 31, 2021].

A. Fortuna. "Full Disk Encryption: tools and setup suggestion for personal data protection." *andreafortuna.org*. https://www.andreafortuna.org/2020/02/21/full-disk-encryption-tools-and-setup-suggestions-for-personal-data-protection. [Accessed May 31, 2021].

"Full-Disk Encryption." *source.android.com*. https://source.android.com/security/encryption/full-disk. [Accessed May 31, 2021].

"Encryption." *source.android.com*. https://source.android.com/security/encryption. [Accessed May 31, 2021].

"File-Based Encryption." *source.android.com*. https://source.android.com/security/encryption/file-based. [Accessed May 31, 2021].

"Metadata Encryption." *source.android.com*. https://source.android.com/security/encryption/metadata. [Accessed May 31, 2021].

"Encrypting Your App's Files." *developer.apple.com*. https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files. [Accessed May 31, 2021].

"Data Protection overview" *support.apple.com*. https://support.apple.com/nl-be/guide/security/secf6276da8a/web. [Accessed May 31, 2021].

"Secure Enclave: zo werkt Apple's speciale beveiligingschip in de iPhone, iPad en Mac." *iculture.nl*. https://www.iculture.nl/uitleg/secure-enclave. [Accessed May 31, 2021].

"Secure Enclave." *support.apple.com*. https://support.apple.com/nl-be/guide/security/sec59b0b31ff/web. [Accessed May 31, 2021].

3.5.5 How do iOS's permissions work?

Apple Platform Security May 2021. (2021). Accessed June 4, 2021. [Online]. Available: https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf

"Contacts." *developer.apple.com*. https://developer.apple.com/documentation/contacts. [Accessed June 4, 2021].

"Accessing User Data." *developer.apple.com*. https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/accessing-user-data. [Accessed June 4, 2021].

3.5.6 How do iOS's signing certificates work?

J. Beckers, private communication, March 2021.

W. Van Renterghem, private communication, May 2021.

"TestFlight." *en.wikipedia.org*. https://en.wikipedia.org/wiki/TestFlight. [Accessed May 31, 2021].

"Certificates." *developer.apple.com*. https://developer.apple.com/support/certificates. [Accessed May 31, 2021].

"Purchase and Activation." *developer.apple.com*. https://developer.apple.com/support/purchase-activation. [Accessed May 31, 2021].

B. Bronosky. "What is the difference between the app ID and the bundle ID? Where is the app ID in the Xcode project?" *stackoverflow.com*. https://stackoverflow.com/questions/4271884/what-is-the-difference-between-the-app-id-and-the-bundle-id-where-is-the-app-id. [Accessed May 31, 2021].

3.5.7 What does iOS do besides enforcing certificates?

Apple Platform Security May 2021. (2021). Accessed June 4, 2021. [Online]. Available: https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf

K. Haider. "How To Sandbox Android Apps For Ultimate Data Privacy." *gtricks.com*. https://www.gtricks.com/android/how-to-sandbox-android-apps-for-privacy. [Accessed May 31, 2021].

T. Reed. “Unprecedented new iPhone malware discovered.” *blog.malwarebytes.com*. https://blog.malwarebytes.com/mac/2019/08/unprecedented-new-iphone-malware-discovered. [Accessed May 31, 2021].

“App security overview for iOS and iPadOS.” *support.apple.com*. https://support.apple.com/nl-be/guide/security/secf49cad4db/1/web/1. [Accessed May 31, 2021].

“App Store.” *developer.apple.com*. https://developer.apple.com/support/app-store. [Accessed May 31, 2021].

“System security overview.” *support.apple.com*. https://support.apple.com/nl-be/guide/security/sec114e4db04/1/web/1. [Accessed May 31, 2021].

4.1.4 Why using an accessibility service?

“Create your own accessibility service.” *developer.android.com*. https://developer.android.com/guide/topics/ui/accessibility/service. [Accessed May 31, 2021].

“Android Donut.” *en.wikipedia.org*. https://en.wikipedia.org/wiki/Android_Donut. [Accessed May 31, 2021].

“Android Jelly Bean.” *en.wikipedia.org*. https://en.wikipedia.org/wiki/Android_Jelly_Bean. [Accessed May 31, 2021].

E. Cebuc. “How are we doing with Android's overlay attacks in 2020?” *labs.f-secure.com*. https://labs.f-secure.com/blog/how-are-we-doing-with-androids-overlay-attacks-in-2020. [Accessed May 31, 2021].

4.4 Concluding questions

F. Ruiz “Android/LeifAccess.A is the Silent Fake Reviewer Trojan.” *mcafee.com*. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/android-leifaccess-a-is-the-silent-fake-reviewer-trojan. [Accessed June 2, 2021].

J. Beckers. “New mobile malware family now also targets Belgian financial apps.” *blog.nviso.eu*. https://blog.nviso.eu/2021/05/11/new-malware-family-now-also-targets-belgian-financial-apps. [Accessed May 31, 2021].

D. Tuffley. “Can I still be hacked with 2FA enabled?” *theconversation.com*. https://theconversation.com/can-i-still-be-hacked-with-2fa-enabled-144682. [Accessed May 31, 2021].

J. Knight. “Google Authenticator Is NOT the Best 2FA App Anymore.” *smartphones.gadgethacks.com*. https://smartphones.gadgethacks.com/news/google-authenticator-is-not-best-2fa-app-anymore-0186776. [Accessed May 31, 2021].

C. Davenport. “Google will remove Play Store apps that use Accessibility Services for anything except helping disabled users.” *androidpolice.com*. https://www.androidpolice.com/2017/11/12/google-will-remove-play-store-apps-use-accessibility-services-anything-except-helping-disabled-users. [Accessed May 31, 2021].

E. Cebuc. “How are we doing with Android's overlay attacks in 2020?” *labs.f-secure.com*. https://labs.f-secure.com/blog/how-are-we-doing-with-androids-overlay-attacks-in-2020. [Accessed May 31, 2021].

M. Rahman. “Bubbles in Android Q will fully replace the overlay API in a future Android version.” *xda-developers.com*. https://www.xda-developers.com/android-q-system-alert-window-deprecate-bubbles. [Accessed May 31, 2021].

5 Guidelines

J. Beckers, private communication, March 2021.

P. Ruggiero & J. Foote. “Cyber Threats to Mobile Phones,” *US-CERT*, Washington DC, USA, Rep. 2011, 2011.

K. Conner. “Over 1,000 Android apps were found to steal your data. Here's what you can do.” *cnet.com*. https://www.cnet.com/how-to/over-1000-android-apps-were-found-to-steal-your-data-heres-what-you-can-do. [Accessed March 30, 2021].

L. Stefanko. “Android ransomware is back.” *welivesecurity.com*. https://www.welivesecurity.com/2019/07/29/android-ransomware-back. [Accessed March 30, 2021].

R. Kamp. “Test: virusscanners voor Android.” *consumentenbond.nl*. https://www.consumentenbond.nl/virusscanner/virusscanners-voor-android. [Accessed March 30, 2021].

“10 Best Mobile APP Security Testing Tools In 2021.” *softwaretestinghelp.com*. https://www.softwaretestinghelp.com/mobile-app-security-testing-tools. [Accessed March 30, 2021].

“New mobile malware family now also targets Belgian financial apps.” *blog.nviso.eu*. https://blog.nviso.eu/2021/05/11/new-malware-family-now-also-targets-belgian-financial-apps. [Accessed May 27, 2021].

“Ingebouwde beveiliging en privacybescherming gebruiken op de iPhone.” *support.apple.com*. https://support.apple.com/nl-be/guide/iphone/iph6e7d349d1/ios. [Accessed May 27, 2021].

D. Hein. “7 Essential Mobile Security Best Practices for Businesses.” *solutionsreview.com*. https://solutionsreview.com/mobile-device-management/7-essential-mobile-security-best-practices-for-businesses. [Accessed May 31, 2021].

“Sample Mobile Device Security Policy,” Sophos, Abingdon, Oxfordshire, UK, White Paper, 2013. Accessed: May 31, 2021. [Online]. Available: https://www.sophos.com/en-us/medialibrary/Gated%20Assets/white%20papers/Sophos-sample-mobile-device-security-policy.pdf.

“OWASP Mobile Security Testing Guide.” *owasp.org*. https://owasp.org/www-project-mobile-security-testing-guide. [Accessed May 27, 2021].

“OWASP Mobile Top 10.” *owasp.org*. https://owasp.org/www-project-mobile-top-10. [Accessed May 27, 2021].

S. Symanovich. “What is a VPN?” *us.norton.com*. https://us.norton.com/internetsecurity-privacy-what-is-a-vpn.html. [Accessed May 31, 2021].

“Runtime application self-protection.” *en.wikipedia.org*. https://en.wikipedia.org/wiki/Runtime_application_self-protection. [Accessed May 31, 2021].

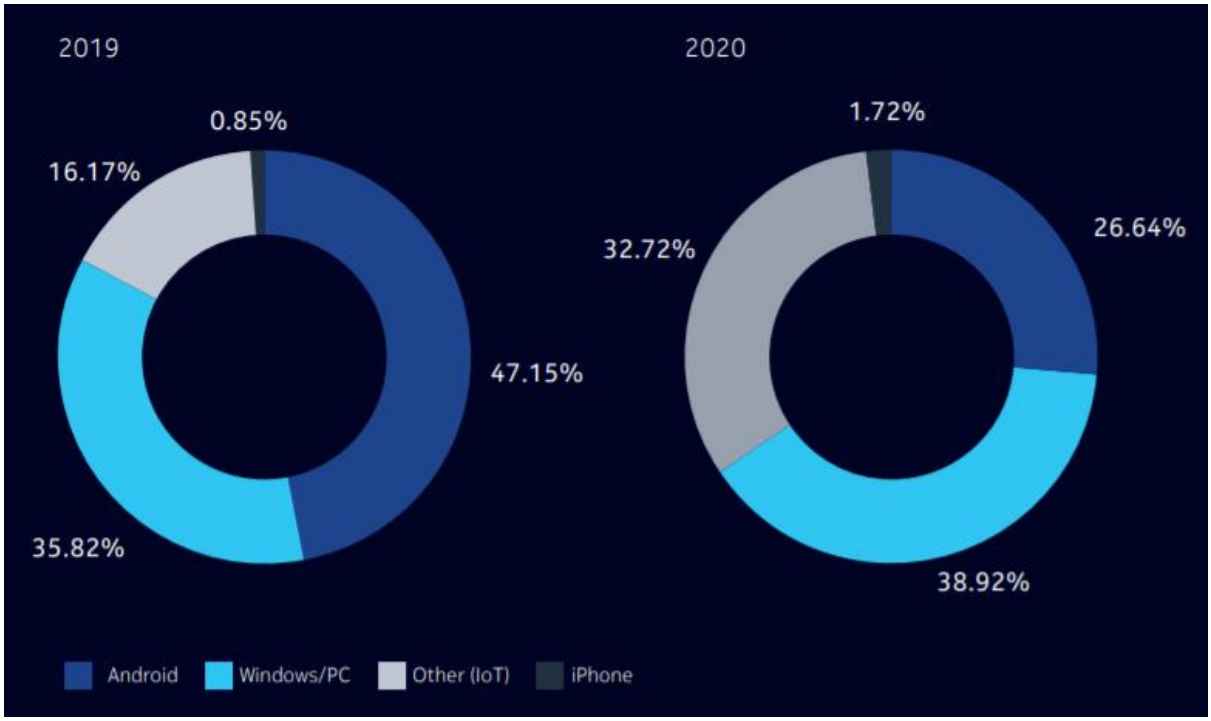
G. Maayan. “What is Interactive Application Security Testing?” *developer.ibm.com*. https://developer.ibm.com/recipes/tutorials/what-is-interactive-application-security-testing. [Accessed May 31, 2021].

“What Is Runtime Application Self-Protection (RASP)?” *checkpoint.com*. https://www.checkpoint.com/cyber-hub/cloud-security/what-is-runtime-application-self-protection-rasp. [Accessed May 31, 2021].

J. Beckers. “New mobile malware family now also targets Belgian financial apps.” *blog.nviso.eu*. https://blog.nviso.eu/2021/05/11/new-malware-family-now-also-targets-belgian-financial-apps. [Accessed May 31, 2021].

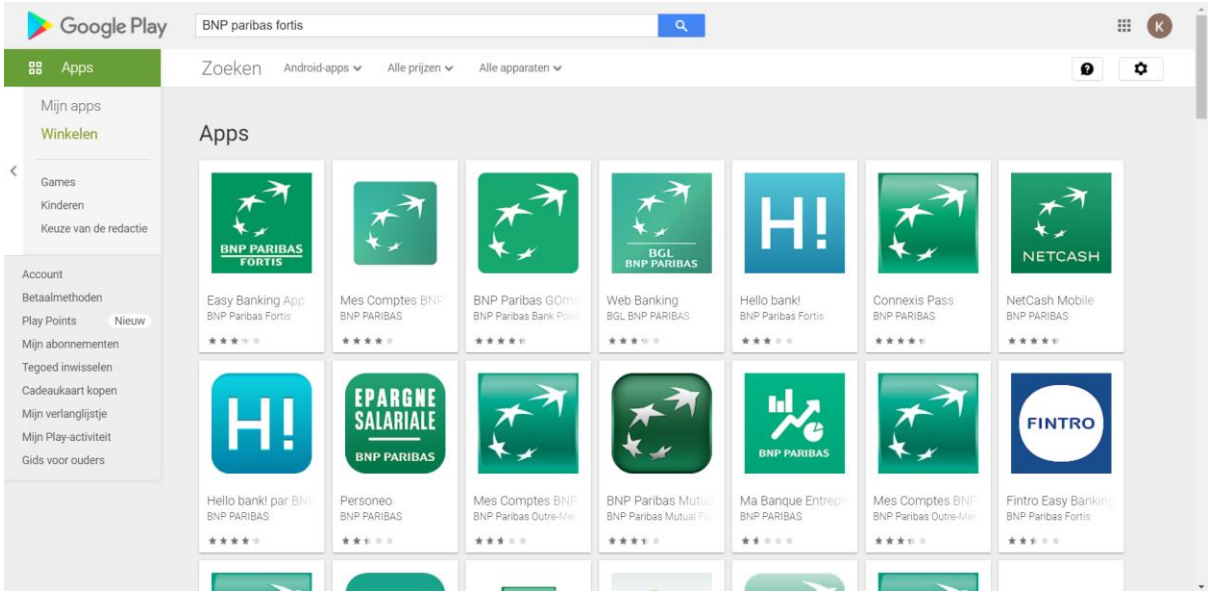
Overview of appendices

Appendix 1: Infection by device according to Nokia's threat intelligence report of 2020



Source: Threat Intelligence Report 2020 – Nokia (Rep. CID210088)

Appendix 2: Result of looking for ‘BNP Paribas fortis’ in Google Play



Source: https://play.google.com/store/search?q=BNP%20paribas%20fortis&hl=nl&gl=US

Appendix 3: Mobile spyware over the past 10 years

Table 3 - Mobile spyware over the past 10 years (2011-2021)

	Name	Description	Targets
2011	**DroidKungFu**	Spyware that roots the infected device	Android
2013	**Pegasus**	Discovered in 2016	Android & iOS
2015	**XAgent**	Spyware that targets specific people (government, military, journalists, etc.)	iOS
2016	**Wroba**	Banking Trojan that also acts as spyware	Android
	XAgent	Android-variant of XAgent tracking Ukrainian military	
	Exodus	Spyware platform	
	SmeshApp	Military spyware tracking Indian military	
2017	**GO Keyboard**	Malicious keyboard app for emojis, themes, and GIFs that secretly steals user's sensitive information	Android
	HummingBad	Attacks mobile devices using a chain-attack tactic	
	HummingWhale	Recent variant of HummingBad	
	AhMyth	Open-source RAT	
	ViperRAT	Spyware targeting Israeli soldiers	
	Joker	SMS Trojan that also acts as spyware (Still active!)	
2018	**Triout**	Posed as a privacy tool	Android
2019	**LeifAccess**	Trojan that abused OAuth by leveraging accessibility services to create accounts and steal data	Android
	MalBus	Hides in a legitimate South Korean Transit app and phishes for Google credentials	
	RB Music	Spyware app based on the AhMyth RAT	
	Assistenza SIM	Italian app abusing iOS enterprise certificates	iOS
	CamScanner	Free app that converts images into PDF documents with a malicious third-party library stealing user's data	Android & iOS
2021	**?**	Spyware posing as a system update by abusing accessibility service	Android

Source: Kyra Van Den Eynde ©2021

Appendix 4: Screenshot of view_tokens_list.xml

view_tokens_list.xml

res > layout > view_tokens_list.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout android:id="@id/viewTokensList" android:focusable="true" android:focusableInTouchMode="true" android:clickable="true" android:layout_width="fill_parent" android:l
3    xmlns:android="http://schemas.android.com/apk/res/android">
4      <TextView android:gravity="center" android:id="@id/noTokensFound" android:visibility="gone" android:layout_width="fill_parent" android:layout_height="wrap_content" android:la
5      <androidx.recyclerview.widget.RecyclerView android:id="@id/listViewTokens" android:layout_width="fill_parent" android:layout_height="fill_parent" />
6      <include layout="@layout/view_no_tokens" />
7      <include layout="@layout/view_encrypted_tokens" />
8      <com.authy.authy.ui.SlidingLayer android:id="@id/slidingLayer" android:layout_width="fill_parent" android:layout_height="fill_parent" slidingLayer:closeOnTapEnabled="true" sl
9          <include layout="@layout/view_token" />
10     </com.authy.authy.ui.SlidingLayer>
11 </RelativeLayout>
```

Source: Kyra Van Den Eynde ©2021

Appendix 5: Screenshot of SlidingLayer's simplified onTouchEvent method

```
public boolean onTouchEvent(MotionEvent motionEvent) {
    float f;
    float f2;
    float f3;
    if (!this.mEnabled || (!this.mIsDragging && !this.mLastTouchAllowed && !allowSlidingFromHereX(motionEvent, this.mInitialX) && !allowSlidingFromHereY(motionEvent, this.mInitialY))) {
        return false;
    }
    int action = motionEvent.getAction();
    if (action == 1 || action == 3 || action == 4) {
        this.mLastTouchAllowed = false;
    } else {
        this.mLastTouchAllowed = true;
    }
    if (this.mVelocityTracker == null) {
        this.mVelocityTracker = VelocityTracker.obtain();
    }
    this.mVelocityTracker.addMovement(motionEvent);
    int i = action & 255;
    if (i == 0) {
        // motionEvent = ACTION_DOWN (finger is touching the screen)
    } else if (i != 1) {
        if (i == 2) {
            // motionEvent = ACTION_MOVE (finger is touching the screen and is moving around)
        } else if (i != 3) {
            if (i == 5) {
                // motionEvent = ACTION_POINTER_1_DOWN
            } else if (i == 6) {
                // motionEvent = ACTION_POINTER_1_UP
            }
        } else if (this.mIsDragging) {
            // finger is dragging
        }
    } else if (this.mIsDragging) {
        // finger is dragging
    } else if (this.mIsOpen && this.closeOnTapEnabled) {
        closeLayer(true);
    } else if (!this.mIsOpen && this.openOnTapEnabled) {
        openLayer(true);
    }
    if (this.mActivePointerId == -1) {
        this.mLastTouchAllowed = false;
    }
    return true;
}
```

Source: Kyra Van Den Eynde ©2021