On the security of pre-installed Android apps in low-cost devices

Alioune DIALLO[1], Anta DIOP[2], Abdoul Kader KABORE[3], Aleksandr PILGUN[3], Jordan SAMHI[3], Tegawendé F. BISSYANDE[3], and Jacques KLEIN[3]

[1] SnT/TruX, University of Luxembourg, Luxembourg
{alioune.diallo}@uni.lu
[2] ESP, Université Cheikh Anta Diop de Dakar, Senegal
antadiop1@esp.sn
[3] SnT/TruX, University of Luxembourg, Luxembourg
{firstname.lastname}@uni.lu

Abstract. Pre-installed system and vendor applications on low-cost Android devices can run with elevated privileges yet receive little independent scrutiny. In this work, we present *PiPLAnD*, a pipeline that extracts APKs from physical devices and applies static analysis to detect sensitive-data leaks, manifest misconfigurations, and suspicious behaviors in pre-installed apps. Using *PiPLAnD*, we analyzed 1544 pre-installed APKs collected from seven devices (Infinix, itel, Tecno). Our findings show that 145 apps (9%) leak sensitive information, 249 apps (16%) export sensitive components without adequate protection, and numerous apps exhibit risky behaviors (226 execute dangerous commands, 79 access/send/delete SMS, 33 perform silent installation actions). We also identified a vendor-shipped package that appears to exfiltrate device identifiers and location to a third-party vendor. These results indicate that pre-installed software on widely distributed, low-cost devices can pose real privacy and security risks to end users.

Keywords: pre-installed apps · sensitive data · Android · static analysis · low-cost devices · Africa.

1 Introduction

The Android operating system has enabled affordable smartphones for millions of people worldwide. Devices typically ship with manufacturer- or vendor-installed systems and third-party apps, which can become a distribution vector for malware and privacy-invasive functionality, especially in developing regions [24]. Low-cost smartphones remain widely used across Africa [36] and have helped reduce the digital divide by expanding access to services [31, 15]. For example, roughly 20–22 million people in South Africa use smartphones, accounting for one third of the population [36]. Feature phones and inexpensive devices are still popular in Africa, which preserves the market opportunity for the expansion of low-cost Android devices. Although low-cost Android initiatives have

improved device accessibility [3], the pre-installation process can be abused to embed harmful apps that reach large user populations.

Multiple reports document dangerous apps shipped with devices [7, 27, 34, 25, 38, 4]. These apps have been observed exfiltrating personal data to remote servers [7], harvesting financial credentials via clipboard monitoring [27], performing silent installations [38], or enrolling users in paid services without consent [4]. Several of the reported apps appear specifically on low-cost devices, including phones sold in Africa [7, 4].

A few studies collected firmware images from vendor websites and online forums, extracting pre-installed apps for analysis [38, 11, 21, 33]. The extracted apps are then examined for suspicious permissions, misconfigurations in manifest files [38, 21, 33], malware [38], and data leaks [11, 14, 6]. Other studies have directly extracted pre-installed apps from physical devices [14, 6], allowing for a more comprehensive assessment across various Android brands. Yet, research on pre-installed applications remains limited since Android device manufacturers are expected to strictly follow compliance guidelines enforced by Google [20]. Unlike globally known brands, Android devices sold in African markets are often very low-priced. This raises concerns about the trade-offs manufacturers make between cost and security.

We studied seven low-cost device models from Infinix, Tecno, and itel brands — all produced by TRANSSION and together accounting for 51% of the African smartphone market in 2023 [8]. We extracted a dataset of 1544 pre-installed APKs for analysis. Pre-installed apps on these devices have received little systematic analysis despite their prevalence. The most notable work is by Elsabagh et al. [11] who examined pre-installed apps from Infinix and Tecno, focusing on privilege escalation. To the best of our knowledge, no prior work has systematically analyzed pre-installed apps across these low-cost Android devices commonly sold in Africa. To explore this gap, we developed *PiPLAnD*, a pipeline that extracts APKs from physical devices and applies taint-based leak detection, pattern-driven behavior scanning, and manifest/component inspection to find data leaks, insecurely exported components, and suspicious behaviors. Our study builds upon existing work by taking a more in-depth approach. For instance, previous studies identified exported components by analyzing only the application manifests. In contrast, we extended this analysis by examining the corresponding code to detect components that provide access to sensitive data. Moreover, while prior research on data leaks has mainly focused on those occurring through Android or Java API methods, our analysis also considers leaks introduced via third-party libraries.

Our analysis uncovered multiple suspicious pre-installed apps across the tested devices. A substantial fraction of these apps leak sensitive information (for example, IMEI, IMSI, location) and exhibit manifest misconfigurations that expose components without adequate protection. In a notable case, *PiPLAnD* identified the package *com.transsion.statisticalsales*, which was not flagged by VirusTotal. These findings underscore the need for independent audits of pre-installed software on low-cost devices in developing regions.

Below, we summarize our main contributions:

① We present *PiPLAnD*, a pipeline to systematically inspect pre-installed apps from Android devices.
② We collected a dataset of 1544 pre-installed APKs from seven low-cost device models (Infinix, Tecno, itel).
③ We identify manifest misconfigurations: 249 app versions (16%) export sensitive components without adequate protection.
④ We quantify sensitive-data leakage: 145 apps (9%) leak identifiers or location data.
⑤ We document widespread suspicious behaviors in pre-installed apps.

2 Background

This short background defines terms used in the paper and clarifies our measurement scope.

Android firmware and pre-installed apps. Firmware boots device hardware and the Android OS[4]. Devices ship with *system apps* and *manufacturer-supplied* pre-installed apps [23]. *System apps* can run with elevated privileges or reside in privileged locations (e.g., `/system/priv-app`). Some devices in our dataset use Android Go, a lightweight Android configuration for entry-level hardware. Android Go was designed to optimize its running on low-end devices; however, the security implications of this optimization are not fully understood [17].

Exported components and misconfigurations. Apps declare components in AndroidManifest.xml. A component is "exported" if other apps can start or bind to it (via `android:exported` or an `intent-filter`). Exported components without proper permissions or access checks form an attack surface and are treated here as security misconfigurations [18].

PII, sources, sinks, and leaks. PII includes identifiers and user data (IMEI, IMSI, phone number, location) [26]. A SOURCE is a method that reads sensitive data (e.g., `getLastKnownLocation()`); a SINK is a method that can exfiltrate data (network send, SMS, world-readable storage). When an app allows to get this data and send it outside the app itself, in this case, we talk about data leakage (or data leak). Static taint analysis tools like FlowDroid are well known to detect leaks in Android apps [2].

Low-cost device. In this study, we often use the terms "low-cost", "low-end", "cheap", and "affordable" when talking about devices that are not expensive but affordable, specifically targeting devices primarily used in Africa.

3 Related Work

To the best of our knowledge, this study is the first one that focuses on analyzing Android apps pre-installed on the devices primarily used in Africa. However,

[4] `https://www.androidpolice.com/what-is-firmware/`

there are some existing works about the analysis of pre-installed apps in Android devices.

Indeed, Zheng et al. [38] presented a tool named DroidRay that extracts statically and dynamically pre-installed apps from 250 firmware downloaded from forums and websites. The static extraction consists of extracting pre-installed apps directly from the firmware images. They also flashed the image into a device and then dynamically extracted the pre-installed apps using ADB commands. DroidRay performs pre-installed app analysis and system analysis by extracting the "SharedUserId" attribute from the AndroidManifest.xml and the signature information from the RSA file before comparing them with the default signatures they found from the AOSP. They also analyzed the apps in VirusTotal to detect potential malware. It performs static and dynamic analysis of the Android firmware by doing a system signature vulnerability detection, a network security analysis, and a privilege escalation vulnerability detection. In our proposed pipeline, we leveraged the technique of dynamic extraction to directly extract pre-installed apps from a physical device using ADB commands rather than collecting firmware and flashing it into a device.

Mitchell et al. [30] designed DexDiff, a system for assessing the security impacts of vendor customization to the official Android system. DexDiff helps the security analyst, who first retrieves the pre-installed apps and libraries from the phone and then builds their corresponding base binaries from the release branch in AOSP on which the phone is based. DexDiff compares each pair of these binaries obtained and evaluates the security impacts of individual modifications. The only similarity between this study and our approach is that it extracts pre-installed apps from the phone. However, the proposed tool did not automate this process. The apps are supposed to be extracted before using DexDiff. Contrary to our approach, PiPLAnD automated the process of extraction and analysis.

Elsabagh et al. [11] proposed a static analysis tool named FIRMSCOPE to identify unwanted functionality in pre-installed apps by analyzing the Android firmware. FIRMSCOPE extracts pre-installed apps from the firmware and then performs a taint analysis with context-sensitive, flow-sensitive, field-sensitive, and partially object-sensitive. Specifically, it focuses exclusively on identifying the increase in privileges.

Gamba et al. [14] presented a large-scale study of Android pre-installed using crowd-sourcing methods. In this study, the authors built an Android app, Firmware Scanner, that looks for and extracts pre-installed apps when installed on a device. This study performs permission analysis using Androguard, static analysis leveraging existing tools such as Androwarn, FlowDroid, and Amandroid, as well as apktool and Androguard frameworks to identify unwanted behaviors, and traffic analysis using the crowd-sourced Lumen mobile traffic dataset to see app real-world behaviors. Compared to this work, we did the same by extracting pre-installed apps from the physical device, but using ADB commands rather than an installed app. We also performed a taint analysis using FlowDroid, but by considering third-party libraries as well.

Blázquez et al. [6] proposed FOTA (Firmware-Over-The-Air) Finder to automatically classify a given APK as FOTA or not based on Androguard, using the dataset of pre-installed apps from Firmware Scanner [14]. Then, they performed behavior analysis relying on FlowDroid and Amandroid for a taint analysis and a modification of Androwarn to analyze the use of API calls. This part of the work is a bit similar to our data leak detection using FlowDroid. However, we considered all the pre-installed apps extracted from devices, and also performed malware detection.

Hou et al. [21] performed a study in which they collected firmware images from vendors, official websites, and open source repositories, and CVE data to link them with pre-installed apps. In this study, they proposed a tool named AndScanner that automates the extraction of pre-installed apps from firmware images before analyzing them. AndScanner proposes an analysis of the security patches of the firmware to know if it has been patched in time and if the security issues have been fixed. It also performs app analysis by analyzing the pre-installed apps using Androguard to identify misconfiguration in the manifest file and CryptoGuard to detect cryptography misuse. Our study is different since we did not only focus on analyzing the manifest files, but also analyzed the cryptography misuse. However, we leveraged on Androguard framework in our pipeline.

More recently, Sutter and Tellenbach [33] proposed FirmwareDroid, an automated static analysis tool for pre-installed apps. This tool automates the process of extracting pre-installed apps from firmware images and their analysis using existing tools, including Androgurad and Exodus. The study identifies the advertising tracker libraries used with Exodus and the permissions pre-installed apps inherited with Androguard. The authors have integrated 8 open source static analysis tools in FirmwareDroid, which could be used for further analysis.

Almost all of these studies are a bit similar since they follow almost the same approach, such as collecting firmware from the Internet, extracting the pre-installed apps, and analyzing them. Our approach allows the extraction of pre-installed apps from physical devices, the detection of apps having suspicious behaviors, the detection of data leakage, and security misconfiguration on the Manifest files. Furthermore, our approach is particularly tested on low-cost devices sold in Africa. However, with this approach, every brand and every Android device can be inspected and analyzed. Consequently, we do not have a limitation related to missing some brands or firmware.

4 Low-cost Android devices in Africa

Upon careful investigation of the African mobile device market, we identified three popular brands shipping Android devices under 100 US dollars price – Infinix, itel, and Tecno. All these devices run Android Go Edition. We acquired 7 devices in total from these brands for our study.

4.1 Android Go Edition

Android Go Edition is a lightweight configuration of standard Android designed for entry-level devices with limited memory (≤ 2 GB) and storage [17]. Android Go ships with resource-optimized "Go" versions of Google apps, but users can still install apps from the Play Store. Android Go may receive updates less frequently than standard Android [32, 17]. To reduce resource consumption, Android Go disables several features by default [17]:

- Picture-in-picture support;
- `SYSTEM_ALERT_WINDOW` permission (display over other apps);
- Split-screen / multi-window;
- Live wallpapers;
- Multi-display;
- Launcher shortcuts (deep shortcuts);
- Reduced maximum width/height for images in remote views;
- VR mode.

Android Go is not necessarily less secure than standard Android. In fact, users may gain security benefits from the removal of the `SYSTEM_ALERT_WINDOW` permission, which has often been abused by malicious apps [13]. We did not find literature that evaluates the security implications of the other optimizations introduced in Android Go. However, Android Go devices typically receive fewer updates and have a shorter support lifecycle, which can leave users without critical security patches [1].

Many pre-installed apps on some device brands are not available on Google Play. They may come from unverified third-party vendors that may potentially distribute malicious apps to the end users.

4.2 Devices and Pre-installed apps

From our seven devices (Infinix, itel, and Tecno) we extracted 1544 pre-installed APK files, including both system and third-party apps, which form the dataset used in this study. Most of the pre-installed apps extracted cannot be found in the Google Play Store, as shown in Figure 1. Several brands provide their own app stores. In particular, the Palm Store is reported to be the official app distribution platform for Infinix, Tecno, and itel[5]. Palm Store is preinstalled on the devices we examined and allows users to install, uninstall, and update apps. According to the provider, the store performs automated and manual security checks, compatibility testing, and content-compliance monitoring.

System apps are signed by multiple different certificate authorities, and the dominant signer varies by device. For example, Table 1 shows that on the Infinix SMART8, 68% of system apps are signed by Infinix; 20% by Google; 4% by Transsion; 2% by Tecno; 1% by Facebook; 1% with the default AOSP certificate; and 4% by other authorities. System apps are usually found on */system/app*

[5] Palmstore: `https://www.palmplaystore.com/`

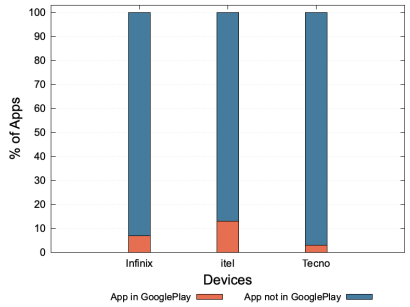

Fig. 1: App present in Google Play vs. App not present in Google Play

Table 1: System apps grouped by certificate authority for some devices.

Signed by	***Infinix SMART8***	***itel A50***	***Tecno POP8***
Infinix	68%	0%	0%
itel	0%	71%	0%
Tecno	2%	1%	68%
Transsion	4%	2%	3%
Google	20%	22%	21%
Facebook	1%	1%	2%
Default	1%	1%	1%
SW	0%	0%	2%
Others	4%	2%	3%

and */system/priv-app* folders. However, when extracting the APK files, we have found that the apps have been distributed not only in these two folders, but in several others as well. Figure 2 shows an example of the folders on Infinix. Many

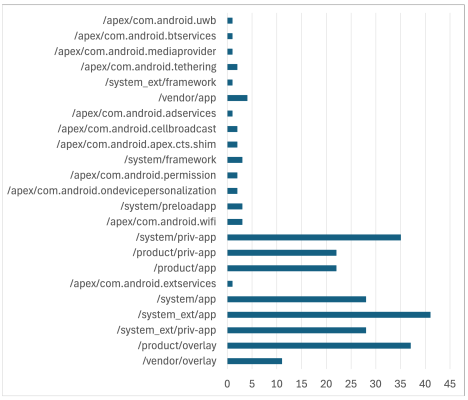

Fig. 2: Location folders of the system apps in Infinix

apps for some brands have been pre-installed based on the regional needs. For example, the supply chain of Transsion devices suggests that Transsion works with partners in Africa, which could pre-install apps and services tailored for the regional needs [22]. Since many steps exist to pre-install apps, this could be an entry point for introducing malicious apps on these devices [37].

5 Methodology

5.1 Research questions

Understanding the potential risks of pre-installed apps is essential. To guide our research, we have established the following key questions.

RQ1: To what extent do pre-installed apps leak sensitive data on low-cost devices? This question focuses on checking whether pre-installed apps perform activities, such as leaking sensitive data via the Internet or using other ways. This helps us to understand not just if these apps pose risks, but also how they do so.

RQ2: To what extent do pre-installed apps exhibit suspicious behaviors on low-cost devices? The aim of this question is to determine how widespread suspicious pre-installed apps are on affordable devices. Although previous research has shown that some of these apps may contain harmful code [38], a more comprehensive assessment is needed to quantify the extent of the problem in different manufacturers and regions. Through this question, we will explore the devices to identify apps having suspicious behaviors (including malware) as well as those using malicious URLs.

RQ3: How prevalent are security misconfigurations in the manifest files of pre-installed apps on low-cost devices? Through this question, we explore apps to identify those exposing sensitive data across the exported components.

By addressing these research questions, our study provides a clearer picture of the security landscape around pre-installed apps on low-cost mobile devices and highlights potential areas where better security measures are needed.

5.2 PiPLAnD Design

This section outlines the workflow of *PiPLAnD*, a pipeline designed to inspect pre-installed apps using static analysis approaches. Figure 3 gives an overview of the pipeline. The source code of *PiPLAnD* is available on our GitHub repository[6]. We present the details of the workflow of each module in the following sections.

Data collection. We have used seven (7) low-cost Android devices in this study. *PiPLAnD* first automatically extracts the pre-installed apps from a physical device using ADB commands when we plug the device into the computer. After extraction, we got a dataset of 1544 APK files from the overall devices. This dataset includes system and third-party apps.

[6] PiPLAnD source code: `https://github.com/liounea/PiPLAnD`

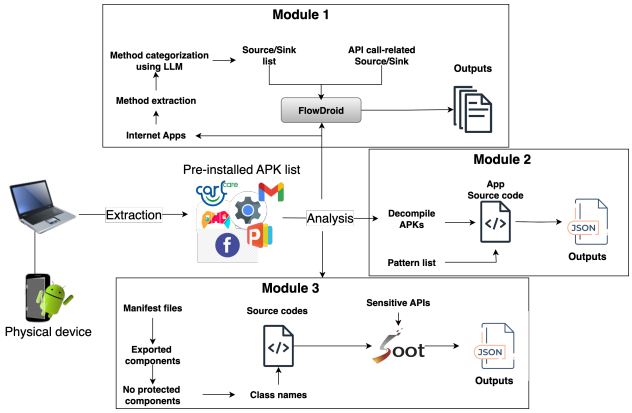

Fig. 3: Overview of *PiPLAnD*'s workflow.

Module 1: Data leak detection.
We integrate FlowDroid into *PiPLAnD* to identify pre-installed apps that leak sensitive data. FlowDroid is based on a list of sources and sinks for the detection. This list contains Android API calls and Java methods. Thus, by default, it cannot detect data leaked from methods other than Android API calls and Java methods. During our study, we have noticed that there are pre-installed apps that use custom methods from third-party libraries to send data over the Internet. Because of that, we have identified apps that access to Internet by looking for *android.permission.INTERNET* permission on their manifest files. For these apps, we extracted all their methods that we have given to LLMs for source and sink categorization. The resulting list is used, in combination with the default list of sources and sinks, to identify Internet apps leaking sensitive data over the Internet. The default Source and Sink list is also used for detecting other types of data leakage in other apps.

Module 2: Behavior analysis.
The behavior analysis focuses on pattern detection. More specifically, we looked for patterns such as *"pm install"*, *"installPackage"*, etc., to detect installation behaviors, *"logcat"* for apps collecting log data, *"content://sms", "delete", "sendTextMessage", "Telephony.SMS_RECEIVED"* for accessing SMS, deleting, sending, and listening to received SMS, etc. Table 2 shows the full list of the patterns we used. *PiPLAnD* helps with this analysis by decompiling the APK file using the Androguard framework and by checking for patterns that match our list in the app code. We only consider these patterns since they are the most obvious and are based on the existing literature, as well. Then, it reports the findings in a JSON file. This file is used for a manual analysis to confirm the behavior of the app based on the patterns detected in its code.

Module 3: Security misconfigurations on Manifest files.
This module allows for the identification of security misconfigurations in the Android manifest files. In this study, we only focus on identifying components

Table 2: The list of patterns

Behaviors	**Patterns**
Access / Delete / Send SMS	content://sms delete sendTextMessage Telephony.SMS_RECEIVED Telephony.Sms
Dangerous commands	"am start " "chmod " "su " "sudo " "rm -rf "
Log collection	logcat
Installation indicators	"pm install " installPackage

exported that allow access to sensitive information. Android allows restricting access to an exported component by using permissions. We first extracted exported components from the manifest files for each app. Then, we filter out and keep those who did not have permission to protect and restrict their access. This gives us a list of class names with their full path in the app code. Besides that, we have constructed a list of sensitive API calls based on the source and sink list from FlowDroid, and from the Androwarn study [10], in which authors look for sensitive data accessed. We explored the list containing the classes of the exported components, and for each class, we analyzed it using the SOOT framework [28]. For the analysis with SOOT, we used the sensitive API list and looked for each of them in the code of the corresponding class to know whether this component allows access to sensitive data. If we did not find the presence of a sensitive API in the code, our analysis program constructs a call graph (CG) of the corresponding class to search whether the sensitive API is used in a method called in the code of the exported component. The results are put into a JSON file, in which we have the corresponding class, the sensitive API, and the method where we found this API.

6 Analysis Results

In this section, we present the findings from seven (7) identified affordable devices that are primarily used in Africa. The main goal of this study is to evaluate the security of the pre-installed apps and to investigate the prevalence of suspicious ones on low-cost devices. Our analysis has revealed interesting findings. Table 3 summarizes these findings. This section shows the results of the analysis by answering the different research questions.

6.1 Apps leaking sensitive data

Mobile apps often handle sensitive data on the device. With the privileges that pre-installed apps have, they can perform harmful activities, including leaking

Table 3: Summary of the results.

Analysis Modules	**Behaviors**	**# of apps (%)**
Exported sensitive components	Components that allow to access sensitive data	249 (16%)
Leak of sensitive data	Leak of sensitive data	145 (9%)
Suspicious behaviors	Dangerous commands	226 (15%)
	Log collection	10 (0.7%)
	Silent installation behaviors	33 (2%)
	Access / Send / Delete SMS	79 (5%)

this data. Our analysis, consisting of identifying apps that leak sensitive data, has identified several pre-installed apps on seven 7 different devices leaking data, such as Mobile Country Code (MCC), user location (longitude and latitude), device info, International Mobile Subscriber Identity (IMSI), IMEI (International Mobile Equipment Identity), etc. Sensitive data is leaked in different ways, such as in SharedPreferences, in logs, in Intents, as well as on the network. Overall, we have found around 9% of the pre-installed apps on the devices leaking sensitive data, which represent 145 pre-installed apps. As an example, the app (*com.transsion.statisticalsales*) sends sensitive information to a remote host by using third-party libraries with customized methods. As illustrated in Listing 1, the app collects location info, IMSI, IMEI, phone version, etc., and sends them to a remote server. Pre-installed apps also leak data using other ways, such as SharedPreferences storage, device logs, etc. These practices expose the user to many security and privacy problems.

Answer to RQ1

The results show that several pre-installed apps on the three devices exhibit harmful activities such as leaking sensitive data, exposing the users to severe risks.

6.2 Suspicious behaviors on pre-installed apps

Since mobile devices come with pre-installed apps, these may contain harmful code that can compromise the security of users [38]. We have looked for apps having suspicious behaviors. Several pre-installed malware have been identified to have silent installation behaviors [38]. Others steal sensitive data using different ways [38, 35, 9]. To identify these kinds of malware, we focus the analysis on detecting patterns. When a pre-installed app declares the *INSTALL_PACKAGES* permission, it has the ability to install an app without the user's knowledge. It can use *android.content.pm.PackageManager.installPackage()* or *Runtime.exec('pm install')* to silently install the app [38]. Our analysis has revealed 33 pre-installed apps having this silent installation behavior on the overall devices.

```
[...]
public class SSHttpClient {
    public static final String BASE_URLFLAG = "PCHttpClient";
    private static final String DEFAULT_BASE_SERVER =
    ↪ "https://asv.transsion.com:443/SaleStatistics/sendsale/sendSale";
    private static final String DEFAULT_INDIA_SERVER =
    ↪ "https://asvin.transsion.com:8080/SaleStatistics/sendsale/sendSale";
    protected static final String TAG = "SSHttpClient_";
    [...]
    public void RegisterInformation(final HttpCallback<HttprequestResult> httpCallback,
    ↪ boolean z) {
        RequestParams requestParams = new RequestParams();
        requestParams.put("ua", this.requestInfo.getUa());
        requestParams.put("screen", this.requestInfo.getScreen());
        requestParams.put("imsi", this.requestInfo.getImsi());
        requestParams.put("imei", this.requestInfo.getImei());
        requestParams.put("phone_version", this.requestInfo.getPhone_version());
        requestParams.put("platform", this.requestInfo.getPlatform());
        requestParams.put("device", this.requestInfo.getDevice());
        requestParams.put("lang", this.requestInfo.getLang());
        requestParams.put("timeStamp", this.requestInfo.getTimeStamp());
        requestParams.put("auth", this.requestInfo.getAuth());
        requestParams.put("lat", this.requestInfo.getLat());
        requestParams.put("lng", this.requestInfo.getLng());
        requestParams.put("client_type", this.requestInfo.getClient_type());
        requestParams.put("phone", this.requestInfo.getPhone());
        requestParams.put("client_version", this.requestInfo.getClient_version());
        requestParams.put("lac", this.requestInfo.getCELL_LAC());
        requestParams.put("cid", this.requestInfo.getCELL_CID());
        mClient.post(z ? DEFAULT_INDIA_SERVER : DEFAULT_BASE_SERVER, requestParams, new
        ↪ AsyncHttpResponseHandler() {
            [...]
        });
    }
}
```

Listing 1: App sending sensitive data to a remote server

Several pre-installed apps have access to the SMS provider *content://sms*. Some of them delete SMS or declare the *SEND_SMS* and *RECEIVE_SMS* permissions with the broadcast action *Telephony.SMS_RECEIVED*, allowing them to listen and send messages. We have found around 79 apps pre-installed on these low-cost devices, sending, deleting, and/or reading SMS contents. Furthermore, at least 10 pre-installed apps can access and collect the logcat content. Since they have déclared the *READ_LOGS* permission, they can access the overall logcat content, including logs from other applications. In addition to this, we have found several apps that execute dangerous commands (226 apps).

Answer to RQ2

Low-cost Android devices ship pre-installed apps with suspicious behaviors, including sending/deleting/reading SMS, executing dangerous commands, accessing overall logcat memory, and having silent installation behaviors.

6.3 Security misconfigurations on the manifest files

Exported sensitive components. Android apps often export components, such as activities, services, receivers, and providers, explicitly by setting *android:exported="true"* or implicitly by declaring an *intent-filter* in the manifest file. When it is the case, the app allows other apps to launch the component [18]. The access to exported components is often restricted using permissions [19]. If there is permission for an exported component, the app that wants to launch it should declare this permission. If an app exports a component without properly enforcing permission, any app could launch it or access sensitive data it contains[7]. When a component allows access to sensitive data, we call it a sensitive component. Our analysis has revealed that several pre-installed apps have sensitive components exported. As illustrated in Table 3, we have found around 16% of the pre-installed apps, representing 249 different app versions, that have exported sensitive components on the low-cost devices, without any restriction or protection mechanism. It means that these devices embed apps that allow other apps to potentially access sensitive information, exposing the user to security and privacy problems. For example, we have found a pre-installed app (*com.transsion.carlcare*) having this method (Listing 2), from an exported activity (*com.transsion.carlcare.WarrantyCardActivity*). This method allows access to location information (lines 13 and 14). In the Android manifest file of the app, this component is clearly exported; however, it is protected by no mechanism, facilitating its easy access and its potential exploitation. Another example shows an exported ContentProvider (*com.sprd.providers.photos.SpecialTypesProvider*) found in the app (*com.sprd.providers.photos*) that allows access to media files, contained in an external storage, from a URI (Listing 3, line 10). This exported component does not have a mechanism that restricts its access by other apps.

Answer to RQ3

The results have revealed several pre-installed apps exporting sensitive components, including activities, services, content providers, and receivers, on low-cost devices. This potentially puts users at risk, as their data could be accessed by third parties.

[7] CWE-926: `https://cwe.mitre.org/data/definitions/926.html`

```
public void R1() {
    [...]
    HashMap<String, String> map = new HashMap<>();
    if (TextUtils.isEmpty(this.f15263g0)) {
        map.put("imei", listA.get(0));
    } else {
        map.put("imei", this.f15263g0);
    }
    map.put("imsi", wd.c.g());
    map.put("lang", getResources().getConfiguration().locale.
                                        toString());
    if (this.f15282z0 != null) {
        map.put("lat", this.f15282z0.getLatitude() + "");
        map.put("lng", this.f15282z0.getLongitude() + "");
    }
    map.put("phone_version", a2());
    map.put("screen", getResources().getDisplayMetrics().widthPixels + "*" +
    ↪  getResources().getDisplayMetrics().heightPixels);
    map.put("ua", Build.BRAND + "-" + Build.MODEL);
    [...]
}
```

Listing 2: Exported activity accessing sensitive information

```
[...]
public final class SpecialTypesProvider extends ContentProvider {
    [...]
    private static final Uri EXTERNAL_CONTENT_URI = MediaStore.Files.getContentUri("external");
    private static final String[] SPECIAL_TYPE_PROJECTION = {"_data", "owner_package_name"};
    [...]
    private int getCameraType(long j) {
        Log.d(TAG, "mediaStoreId = " + j);
        boolean z = true;
        Cursor cursorQuery = getContext().getContentResolver().query( EXTERNAL_CONTENT_URI, SPECIAL_TYPE_PROJECTION, "_id=?",
        ↪  new String[]{String.valueOf(j)}, null);
        if (cursorQuery != null) {
            try {
                if (cursorQuery.moveToFirst()) {
                    String string = cursorQuery.getString(0);
                    Log.d(TAG, "mediaPath = " + string);
                    String string2 = cursorQuery.getString(1);
                    if (!GOOGLE_PHOTOS_PACKAGE_NAME.equals( string2) && !GOOGLE_GALLERY_PACKAGE_NAME.equals( string2)) {
                        z = false;
                    }
                    Log.d(TAG, "ownerPackageName = " + string2 + ", isGoogleCreatedImage = " + z);
                    try {
                        ExifInterface exifInterface = new ExifInterface();
                        exifInterface.readExif(string);
                        iIntValue = z ? 0 : exifInterface.getTagIntValue( ExifInterface.TAG_CAMERATYPE_IFD). intValue();
                        Log.d(TAG, "getCameraType cameraType = " + iIntValue);
                    } catch (Exception e) {
                        Log.d(TAG, "Exception occurs, mediaPath = " + string + ". ex = " + e);
                    }
                }
            } finally {
                if (cursorQuery != null) {
                    cursorQuery.close();
                }
            }
        }
        return iIntValue;
    }
    [...]
}
```

Listing 3: Exported ContentProvider accessing external storage files

7 Discussion

In this section, we discuss the results from the analysis of the devices.

Sensitive data leakage. During our analysis, we have found several pre-installed apps leaking sensitive data. Several of them send the data to a remote host by using Android API calls. Others use third-party libraries that use customized methods to avoid existing detection (e.g, *com.transsion.statisticalsales*). We have considered these customized methods and added them to FlowDroid, and we have detected apps sending sensitive data over the Internet with these methods. Considered as malware by some blog posts [7], the app (*com.transsion.statisticalsales*) is not detectable by malware scanners such as VirusTotal. It silently collects data (phone version, IMSI, user location, CID (Cell Tower ID), LAC (Location Area Code), etc.) and sends it to a remote host. This suspicious behavior compromises users' security and privacy. Several other apps get and store sensitive data in internal storage, such as SharedPreferences, or log it in the logcat memory. These leaked data can be used by malicious actors for user profiling, user tracking [5], or accessed by other apps since pre-installed apps can share each other's data when they have the same *sharedUserId* and have signed with the same certificate [38].

Suspicious behaviors. We have found several pre-installed apps having suspicious behaviors, including silent installation behavior, sending/deleting SMS, etc. These are done without the user's knowledge and may have negative consequences. Indeed, when an app is able to install another app silently, it may install a malicious app from a malicious remote server or via dynamic code loading. This technique is often used by malicious actors to infect Android devices and avoid detection [38]. This is possible only with system apps, since non-system apps cannot declare the required permission. When a pre-installed app has the ability to send and delete messages without the user's interaction, this can lead to the theft of sensitive data. Several malware are known to use this technique. For instance, a variant of the Triada malware has been found pre-installed in Android devices [35]. This malware performs several actions, among which we have enabling premium SMS services, intercepting, sending, and deleting messages.

Security misconfiguration. Our analysis has revealed several apps exporting components that allow other apps to potentially access sensitive data, without any protection. The Android framework proposes permission levels, including normal, dangerous, and signature, to protect and restrict access to components [16]. The absence of protecting exported components is a known vulnerability from the community [16]. When an app exports a component without any restriction, it allows other apps to access it. From this, a malicious app may access sensitive resources, as mentioned in the Common Weakness Enumeration (CWE - 926). The app can be victim to an attack named confused deputy attack [12, 14], as well as a malicious app can abuse the privileges that these components have to gain unauthorized access to these resources [16, 29].

8 Conclusion

This study investigates and analyzes Android pre-installed apps, in particular, those shipped with phones sold in Africa. For this purpose, we have proposed

a pipeline that allows the extraction of APK files from a physical device and inspects them to look for different suspicious behaviors, including pre-installed malware, apps exposing sensitive data, and apps sending personal data to remote hosts. The pipeline is tested by inspecting three different low-cost Android devices bought in Africa. The results show interesting findings, such as (1) the leak of sensitive data, (2) sensitive data exposure through exported components, and (3) apps having suspicious behaviors. As future research, we planned to go deeper into these pre-installed apps by analyzing the URLs they use to identify malicious ones, the native libraries used, and by looking for further suspicious behaviors. Our future study will extend the number of low-cost devices and compare the results from the analysis of pre-installed apps with those from European devices, as well. In addition, we planned to perform a dynamic analysis by implementing a solution that monitors the system log and detects suspicious behaviors using AI models.

References

1. Acar, A., Tuncay, G.S., Luques, E., Oz, H., Aris, A., Uluagac, S.: 50 shades of support: A device-centric analysis of android security updates. In: Netw. Distrib. Syst. Security Symp., San Diego, CA, USA (2024)
2. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P.: Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. p. 259–269. PLDI '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2594291.2594299, https://doi.org/10.1145/2594291.2594299
3. BARTON, J.: Orange releasing android-powered ultra-low cost smartphone for africa (2020), https://developingtelecoms.com/telecom-technology/telecom-devices-platforms/10053-orange-releasing-android-powered-ultra-low-cost-smartphone-for-africa.html
4. BBC: Chinese phones with built-in malware sold in africa (2020), https://www.bbc.com/news/technology-53903436
5. Becenti, M.: Leaky apps: How they're stealing your data without you knowing (2024), https://www.quokka.io/blog/leaky-apps-security-risks
6. Blázquez, E., Pastrana, S., Feal, A., Gamba, J., Kotzias, P., Vallina-Rodriguez, N., Tapiador, J.: Trouble over-the-air: An analysis of fota apps in the android ecosystem. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1606–1622 (2021). https://doi.org/10.1109/SP40001.2021.00095
7. Bobe, B.: Statisticalsales — the malware pre-installed on your phone (2024), https://medium.com/@threatspotlight/episode-5-statisticalsales-417a14e0f75a
8. Canalys: African smartphone market surges 24% in q4 2023, despite macro headwinds (2024), https://www.canalys.com/newsroom/africa-smartphone-market-Q4-2023
9. Cimpanu, C.: Triada trojan found in firmware of low-cost android smartphones (2017), https://www.bleepingcomputer.com/news/security/triada-trojan-found-in-firmware-of-low-cost-android-smartphones/

10. Debize, T.: Androwarn: Yet another static code analyzer for malicious android applications (2012), https://github.com/maaaaz/androwarn
11. Elsabagh, M., Johnson, R., Stavrou, A., Zuo, C., Zhao, Q., Lin, Z.: FIRMSCOPE: Automatic uncovering of Privilege-Escalation vulnerabilities in Pre-Installed apps in android firmware. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 2379–2396. USENIX Association (Aug 2020), https://www.usenix.org/conference/usenixsecurity20/presentation/elsabagh
12. Felt, A.P., Wang, H.J., Moshchuk, A., Hanna, S., Chin, E.: Permission re-delegation: attacks and defenses. p. 22. SEC'11, USENIX Association, USA (2011)
13. Fratantonio, Y., Qian, C., Chung, S.P., Lee, W.: Cloak and dagger: from two permissions to complete control of the ui feedback loop. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 1041–1057. IEEE (2017)
14. Gamba, J., Rashed, M., Razaghpanah, A., Tapiador, J., Vallina-Rodriguez, N.: An analysis of pre-installed android software. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 1039–1055 (2020). https://doi.org/10.1109/SP40000.2020.00013
15. GOODIN, D.: Potentially millions of android tvs and phones come with malware preinstalled (2023), https://arstechnica.com/information-technology/2023/05/potentially-millions-of-android-tvs-and-phones-come-with-malware-preinstalled/
16. Google: Permission-based access control to exported components (2024), https://developer.android.com/privacy-and-security/risks/access-control-to-exported-components
17. Google: Android (Go edition) (2025), https://developer.android.com/guide/topics/androidgo
18. Google: android:exported (2025), https://developer.android.com/privacy-and-security/risks/android-exported
19. Google: App components (2025), https://developer.android.com/guide/topics/manifest/manifest-intro#components
20. Google: Play Protect Certified Android devices: safe and secure (2025), httphttps://www.android.com/certified/
21. Hou, Q., Diao, W., Wang, Y., Liu, X., Liu, S., Ying, L., Guo, S., Li, Y., Nie, M., Duan, H.: Large-scale security measurements on the android firmware ecosystem. In: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). pp. 1257–1268 (2022). https://doi.org/10.1145/3510003.3510072
22. Hu, A.: From mobile king in africa to e-scooter giant? transsion's next big move (2025), https://www.exportsemi.com/company-post/from-africa-phone-king-to-e-bikes-can-transsion-work-another-miracle/
23. Hutchinson, R.: How to delete pre-installed android apps from your smartphone (2024), https://www.geeky-gadgets.com/how-to-delete-pre-installed-android-apps-from-your-smartphone/
24. kaspersky: Malware attacks in africa are increasing, reaching 85 million in only 6 months (2021), https://kaspersky.africa-newsroom.com/press/malware-attacks-in-africa-are-increasing-reaching-85-million-in-only-6-months?lang=en
25. Kurt Knutsson, C.R.: Fbi warns over 1 million android devices hijacked by malware (2025), https://www.foxnews.com/tech/fbi-warns-over-1-million-android-devices-hijacked-malware.amp
26. LABOR, U.D.O.: Guidance on the protection of personally identifiable information (pii), https://www.dol.gov/general/ppii

27. Lakshmanan, R.: Chinese android phones shipped with fake whatsapp, telegram apps targeting crypto users (2025), https://thehackernews.com/2025/04/chinese-android-phones-shipped-with.html
28. Lam, P., Bodden, E., Lhoták, O., Hendren, L.: The soot framework for java program analysis: a retrospective. In: Cetus Users and Compiler Infastructure Workshop (CETUS 2011). vol. 15 (2011)
29. Maqsood, H.M.A., Qureshi, K.N., Bashir, F., Islam, N.U.: Privacy leakage through exploitation of vulnerable inter-app communication on android. In: 2019 13th International Conference on Open Source Systems and Technologies (ICOSST). pp. 1–6 (2019). https://doi.org/10.1109/ICOSST48232.2019.9043935
30. Mitchell, M., Tian, G., Wang, Z.: Systematic audit of third-party android phones. In: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy. p. 175–186. CODASPY '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2557547.2557557, https://doi.org/10.1145/2557547.2557557
31. Ng, A.: Android malware that comes preinstalled is a massive threat (2019), https://www.cnet.com/tech/mobile/android-malware-that-comes-preinstalled-are-a-massive-threat/
32. Sharma, S.: Android go vs regular android – which is right for your business app? (2025), https://www.appventurez.com/blog/android-go-apps-vs-regular-apps
33. Sutter, T., Tellenbach, B.: Firmwaredroid: Towards automated static analysis of pre-installed android apps. In: 2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft). pp. 12–22 (May 2023). https://doi.org/10.1109/MOBILSoft59058.2023.00009
34. TIMESOFINDIA: These android phones come with dangerous pre-installed apps (2019), https://timesofindia.indiatimes.com/gadgets-news/these-android-phones-come-with-dangerous-pre-installed-apps/articleshow/72110567.cms
35. Toulas, B.: Counterfeit android devices found preloaded with triada malware (2025), https://www.bleepingcomputer.com/news/security/counterfeit-android-devices-found-preloaded-with-triada-malware/
36. TRUSTONIC: Closing the digital divide with android smartphones in africa (2021), https://www.trustonic.com/opinion/closing-the-digital-divide-with-android-smartphones-in-africa/
37. upstream: xhelper/triada malware pre-installed on thousands of low cost chinese android devices in emerging markets (2020), https://www.upstreamsystems.com/press/press-releases/xhelper-triada-malware-pre-installed-on-thousands-of-low-cost-chinese-android-devices-in-emerging-markets/
38. Zheng, M., Sun, M., Lui, J.C.: Droidray: a security evaluation system for customized android firmwares. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. p. 471–482. ASIA CCS '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2590296.2590313, https://doi.org/10.1145/2590296.2590313