Certified Copy? Understanding Security Risks of Wi-Fi Hotspot based Android Data Clone Services

Siqi Ma
The University of Queensland
Australia
slivia.ma@uq.edu.au

Hehao Li, Wenbo Yang, Juanru Li[1]
Shanghai Jiao Tong University
China
<lihehao,loccs,jarod>@sjtu.edu.cn

Surya Nepal
CSIRO
Australia
surya.nepal@data61.csiro.au

Elisa Bertino
Purdue University
USA
bertino@purdue.edu

ABSTRACT

Wi-Fi hotspot-based data clone services are increasingly used by Android users to transfer their user data and preferred configurations while upgrading obsolete phones to new models. Unfortunately, since the data clone services need to manipulate sensitive information protected by the Android system, vulnerabilities in the design or implementation of these services may result in data privacy breaches. In this paper we present an empirical security analysis of eight widely used Wi-Fi hotspot-based data clone services deployed to millions of Android phones. Our study evaluates those services with respect to data export/import, data transmission, and Wi-Fi configuration with respect to security requirements that the data clone procedure should satisfy. Since data clone services are closed source, we design Poirot, an analysis system to recover workflows of the data clone services and detect potential flaws. Our study reveals a series of critical security issues in the data clone services. We demonstrate two types of attacks that exploit the data clone service as a new attack surface. A vulnerable data clone service allows attackers to retrieve sensitive user data without permissions, and even inject malicious contents to compromise the system.

KEYWORDS

Data clone service, Platform app, Proprietary protocol, Android diversification

ACM Reference Format:
Siqi Ma, Hehao Li, Wenbo Yang, Juanru Li[1], Surya Nepal, and Elisa Bertino. 2020. Certified Copy? Understanding Security Risks of Wi-Fi Hotspot based Android Data Clone Services. In *Annual Computer Security Applications Conference (ACSAC 2020), December 7–11, 2020, Austin, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3427228.3427263

1 INTRODUCTION

Mobile phones nowadays store large amounts of sensitive user data such as personal e-mails and account passwords. To protect against security threats such as data theft and malware, mobile phones and their operating systems (OSes) have introduced a number of security protection mechanisms such as full disk encryption and app data isolation to prevent unauthorized access. However, when a user wants to transfer her data from an obsolete mobile phone to a new one, the transfer process suffers from both usability and privacy issues. With respect to usability, official data backup/restore mechanisms of Android and iOS systems require either a cloud drive (e.g., Google Drive and iCloud) or a desktop computer as an auxiliary medium. Although users do not frequently replace their mobile phone, 55.56% of them update to a new phone every 12 or 24 months. Such data backup/restore mechanisms are not only inefficient but also inconvenient. With respect to privacy, exporting sensitive user data to either a cloud drive or a hard disk raises concerns of data abuse or data leakage.

To ensure a secure and smooth data migration, many Android phone manufacturers developed a new cross-device data migration solution — **Wi-Fi hotspot-based data clone**. By establishing a private and temporary Wi-Fi network connection between two mobile phones, user data are transferred between these two phones directly. This approach achieves a high data transmission speed without requiring the transferred user data to be stored at a third-party medium. More importantly, system privileges are granted to these data clone services so that they are allowed to access sensitive user data (e.g., app login credentials stored at app sandboxes). Hence, a large number of manufacturers embed the data clone service as a default service in their mobile phones. Table 1 lists the number of devices produced by the well-known manufacturers that would be affected by the vulnerabilities we have identified.

Nonetheless, designing a secure data clone service is challenging because developers are not security experts and they focus more on the usability of the software [36] instead of its security complied with the Android security model [24]. Although most data clone services claim to have addressed security issues, their implementations are closed source and often lack a comprehensive assessment. In response, we conducted a systematic study against the popular Wi-Fi hotspot-based data clone services published by the eight mainstream Android manufacturers. To the best of our knowledge, we are the first to analyze this application scenario.

Our target is to examine whether the analyzed data clone services satisfy essential security goals such as confidentiality and integrity checks for transferred data, and authentication of connected mobile phones. In order to carry out our analyses, we build Poirot, an analysis system to recover the undocumented workflows and proprietary data transmission protocols. Poirot statically analyzes executables of each data clone service and dynamically inspects its behavior (including code execution and relevant network traffic). The data clone services are analyzed in three perspectives: data

[1]Juanru Li is the corresponding author.

Table 1: Estimated number of devices affected by vulnerabilities in Wi-Fi hotspot-based data clone

Data Clone Service	Number of vulnerable devices
Huawei PhoneClone	≥ 100 million
OPPO BackupAndRestore	≥ 70 million
Vivo EasyShare	≥ 70 million
Xiaomi Backup	≥ 50 million
Gionee GdataGhost	≥ 15 million
OnePlus BackupRestore	≥ 4 million
Motorola Migrate	≥ 1 million

In the rest of this paper we refer to each data clone service by the name of the company providing it (e.g., Huawei PhoneClone is referred to as Huawei)

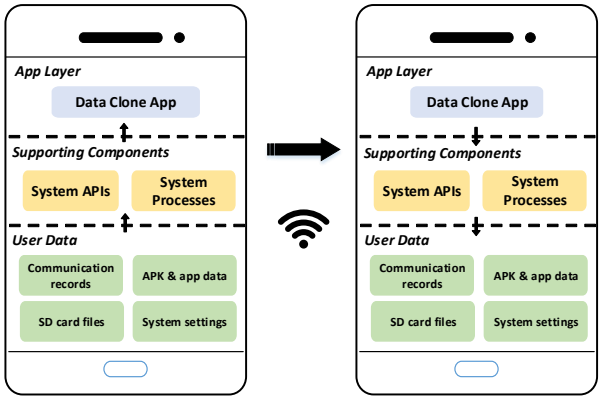

Figure 1: Internal architecture of a Wi-Fi hotspot based data clone service

export and import, data transmission, and Wi-Fi configuration. The results show that those data clone services contain many serious design flaws that have never been reported. To protect end-users, we have contacted the relevant manufacturers and reported the identified flaws. We expect our study becomes a guide to revising similar functions as well as developing new services securely.

To prove that those flaws can lead to actual attacks, we designed two types of attacks. The attacks can either be launched remotely through wireless communication or locally via a malicious app on either the old phone or the new one. We found that seven out of eight investigated services suffer from at least one attack, and since those data clone services are implemented as pre-installed system components, we could estimate the number of mobile phones affected according to the sales data released by each manufacturer.

The rest of the paper is organized as follows. Section 2 introduces relevant background on Wi-Fi hotspot-based data clone. Section 3 introduces our security analysis approach and describes Poirot. It also describes in details the different features we analyze for each data clone service. Section 4 reports the results of our analysis and introduces the attacks we have designed to exploit some of the flaws identified by our analysis. Finally Section 5 discusses related works and Section 6 outlines a few conclusions.

2 WI-FI HOTSPOT-BASED DATA CLONE

The **Wi-Fi hotspot-based data clone** is a data migration procedure between two Android mobile phones, in which a temporary, private Wi-Fi network is built connecting the phones. In comparison to data clone solutions relying on a USB cable (e.g., Android's ADB backup and restore), this wireless data clone procedure is convenient. Unlike a cloud-based data clone, which uses a remote cloud server (e.g., Google Drive) as the intermediate data storage, such a Wi-Fi hotspot-based data migration is fast – around 10MB/s relying on a private Wi-Fi network between two involved phones, and is not much affected by the Internet connection. In addition it does not "consume" any mobile data.

Figure 1 shows the internal architecture of a Wi-Fi hotspot-based data clone. A data clone service typically consists of a **data clone app** and several **supporting components** implemented as either system libraries or system processes. Specifically, a data clone procedure between two phones consists of four steps:

1) Wi-Fi Setup: The private Wi-Fi hotspot is started by the data clone app on the old mobile phone[1]. The app sets up the Wi-Fi network and then encodes the Wi-Fi information (i.e., SSID and password) into an QR code. The QR code might include a connection port, which is allocated either randomly or it is fixed. The new mobile phone can join the WLAN by using the camera to scan the QR code.

2) Data Export: According to the permissions granted to the data clone app, the app on the old mobile phone lists all the exportable data for users to select. The app then packs the selected data into different files with customized transfer formats. Note that the large size data, such as photos and videos, are compressed when their size exceeds a certain limit.

3) Data Transmission: To transfer the packed files between two mobile phones, most manufacturers customize the private application protocols based on TCP. Others rely on some existing protocols such as FTP and HTTP. During the actual transfer, files are usually transferred through multiple threads. Data clone services adopt multiple strategies to protect the transferred data. First, the privately built Wi-Fi network is protected by a security standard protocol (e.g., WPA2 protocol) and excludes unauthorized devices. Second, data clone services rely on a secure protocol (e.g., TLS) to avoid potential attacks (e.g., eavesdropping) from third parties. Finally, many data clone services deploy proprietary data encoding and encryption schemes to further guarantee the confidentiality and integrity of the transferred data.

4) Data Import: When all packed files are transferred to the new mobile phone, its data clone app first unpacks the data and then restores each to the specific directory. Obviously, permissions to operate on these data are required.

In the following, we detail the four types of user data that can be migrated.

2.1 Cloneable User Data

User data are stored at different places in the mobile phone, such as SD cards and system databases. With respect to their purposes,

[1]In fact, the Wi-Fi hotspot can be set up by either the old mobile phone or the new mobile phone.

storage locations, and access permissions, we can classify the transferred user data into four categories: **SD card data**, **communication records**, **system settings**, and **app related data**.

2.1.1 SD card data. The Android system supports SD cards to expand the storage of the phone. The specific user data (e.g., photos) are stored in the partition of the SD card, i.e., `/sdcard`[2]. The SD card data include: 1) **Media Files**, which are digital camera image (DCIM) files and audio files that are created by apps such as camera, video records, or music players; and 2) **Downloaded Files**, which are the files downloaded via Internet and stored at the download folder `/sdcard/Downloads/`. Typical formats of the downloaded files include Microsoft office documents (.docx, .xlsx, etc.) and portable documents (.pdf)[3].

Data stored on the SD card are accessible by an app if it is authorized with `READ_EXTERNAL_STORAGE` and `WRITE_EXTERNAL_STORAGE` permissions, respectively. As these two permissions are classified into the `STORAGE` permission group with a *dangerous* protection level, the apps must prompt the users to grant the permissions at runtime instead of directly requesting them at installation time. Such dynamic prompt for those two permissions has been introduced in Android starting from version 6.0.

2.1.2 Communication Records. Communication records represent the information created during communication, mainly *SMS messages*, *contacts*, and *call logs*. The Android system stores these records in SQLite databases and any app with the relevant permissions (e.g., SMS permission) granted is able to access them. Notice that SMS messages are stored in different "stores" in different Android versions. They are stored in the system database[4] from Android 4.4 to Android 6.0. Since Android version 7.0, SMS messages are stored in an encrypted database protected by a hardware-bound key.

To access communication records, starting from Android version 6.0 an app needs runtime permissions classified as *dangerous*.

- **SMS Messages:** apps need the `READ_SMS` permission to read, and only the default SMS app indicated by the user is allowed to obtain the `WRITE_SMS` permission to insert or modify SMS messages.
- **Contacts:** apps need runtime permissions `READ_CONTACTS` and `WRITE_CONTACTS` in the `CONTACTS` permission group to retrieve contact records.
- **Call Logs:** apps require runtime permissions `READ_CALL_LOG` and `WRITE_CALL_LOG` to read and write call logs, respectively.

2.1.3 System Settings. The function `SettingsProvider` is used by the Android system to manage various system settings such as `HTTP_PROXY` and `BLUETOOTH_DISCOVERABILITY`. System settings are classified into three categories: `System`, `Global`, and `Secure`. The `Global` (e.g., Bluetooth on and off) and `Secure` categories (e.g., Location) contain app-read-only system preferences. To modify them, the user must explicitly operate the system UI. For settings in the `System` category, apps could request the `WRITE_SETTINGS` permission (a *dangerous* protection level runtime permission) to read and write them. In addition, some sensitive data, such as Wi-Fi passwords and passcodes, are stored in private directories to which the special SEAndroid types are assigned. For instance, the SSID and the password for Wi-Fi network connection in `wifiConfigStore.xml` are `wifi_data_file` type objects, and thus a normal platform app cannot access them. To directly operate on system settings, data clone services often utilize a supporting process with the root privilege to circumvent the restriction of Android systems.

2.1.4 App-related Data. While executing an app, app-related data are generated and only accessible by the host app. There are two main types of data: **APK Files** and **App Data**. APK files are the installation packages of apps stored in the `/data/app/` directory. They are readable by any apps without requesting any permissions. App data include app database, app settings, and all other user data, which are isolated from the other apps via the app sandbox. App data are only accessible by the host app stored in the private app folder, i.e., `/data/data/<packageName>`.

Note that users can access the private folder through the `ADB` tool [3] when its attribute `android:allowBackup` is set as `True`. However most apps nowadays disable this attribute.

3 SECURITY ANALYSIS

To protect user data against leakages, Android security policies [1] are designed to restrict data export/import. Hence, manufacturers need to modify the Android system by embedding customized components to bypass policy restrictions on data export/import. However, such modifications may introduce security threats.

In what follows, we discuss the attack models against customized Wi-Fi hotspot-based data clone services, and then introduce our security analysis system and our approaches to detect flaws in these services.

3.1 Attack Model

In our attack model, the main target of the adversary is the transferred cloned data. A customized data clone service mainly introduces two attack surfaces: 1) new sensitive data export/import interfaces in the Android system; 2) a potentially unprotected wireless data transmission.

We assume that an attacker holds the same brand of Android mobile phone and thus he can reverse engineer the data clone app to retrieve the required information. Then we consider the following types of attack:

A1: On-device Data Extraction. In this attack, given two mobile phones, we assume that the attacker has managed to install a malicious app on one of the two mobile phones before the data clone procedure. Although the malicious app is isolated by Android security policies, it can access the sensitive data without breaking the security model of the Android system due to the insecure implementation of the data clone service. When the user starts to execute the data clone service, the malicious app can detect such an execution and exploit the data clone service to obtain the transferred sensitive user data.

A2: Network-level Eavesdropping and Tampering. In this attack, we assume that the attacker does not install any malicious app

[2]The partition of the SD card can be either a physical SD card or an emulated one using part of the internal storage.

[3]A new data storage, scoped storage, has been introduced in Android version 10 or higher. It makes easier to maintain the external storage and allows an app to access the app-specific storage on the external storage.

[4]`com.android.providers.telephony/databases/mmsms.db`

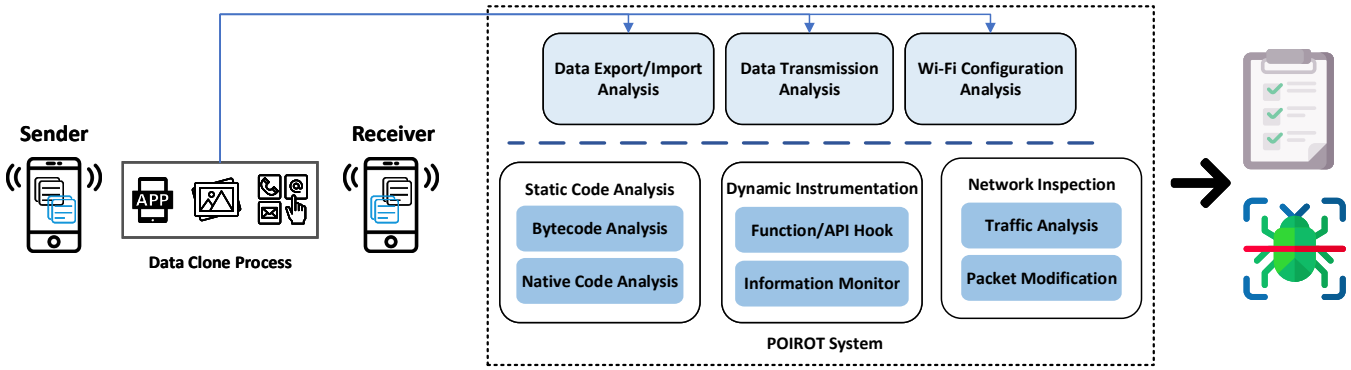

Figure 2: A high-level overview of our analysis workflow

on any of the mobile phones. Thus, instead of monitoring executions of the mobile phones, the attacker continuously monitors the network to identify and exploit the insecure Wi-Fi network established for the data clone service. Then during the execution of the data clone operation, the attacker could launch man-in-the-middle attacks to eavesdrop and tamper the transferred data.

3.2 Approach Overview

Most implementations of the customized Wi-Fi hotspot-based data clone services are neither documented nor open-source. To fully understand a data clone service and assess whether the modified Android system violates the Android security policies, we execute the workflow shown in Figure 2 to analyze the entire procedure of a data clone service from various aspects. We first build Poirot, an analysis system with three components: a **static code analysis**, a **dynamic instrumentation**, and a **network inspection**. Relying on Poirot, we check whether the data clone service is implemented properly by analyzing: 1) **data export/import**; 2) **data transmission**; and 3) **Wi-Fi configuration**. Details about Poirot and the analysis approach are introduced below.

3.3 Poirot Analysis System

The design of a system for properly analyzing the security data cloning services requires addressing the following challenges:

1) How to understand the complicated functionalities provided by a data clone service? The Wi-Fi hotspot-based data clone service aims at transferring various types of user data. Thus, a number of privileges are required and a wide variety of system functions and components are invoked to grant these privileges. Hence, locating all the involved system functions and components in ARM binary code and Dalvik bytecode is a challenge.

2) How to conduct an effective dynamic analysis on new models of Android phones? A typical dynamic analysis needs a high privilege to debug the involved executables, to hook critical system APIs and to extract sensitive data such as app data and network traffic. However, most of the new models of Android phones do not allow users to obtain the root privilege, and thus hinder dynamic analysis.

3) How to analyze the data transmission protocol? Proprietary protocols are adopted for data transmission and thus transferred data are packed in customized formats. It is difficult to design a technique to determine whether these protocols are secure.

In response to these challenges, we design and implement Poirot which consists of three components: 1) a static code analysis to identify and analyze the executables that are relevant to the data clone service; 2) a dynamic instrumentation to conduct a runtime information analysis during the execution of the data clone procedure; 3) a network inspection to monitor the transferred data and support active network traffic inspection.

3.3.1 Static Code Analysis. As data clone services are implemented by data clone apps, Poirot analyzes the data clone apps to understand how each data clone service is implemented. It searches and locates bytecode and binary executables that are correlated to the data clone service (solution of challenge 1). To differentiate the data clone apps developed by mobile phone manufacturers and third parties, we refer to the data clone apps that are pre-installed as **platform apps**.

Bytecode Analysis. The bytecode analysis aims at parsing the platform app and checking the relevant functions in the Android system framework; thus Poirot first extracts the APK file of the platform app. According to the app name (i.e., the name displayed on the mobile phone), Poirot obtains all APK files from the mobile phone and executes `aapt` to parse the APK files to identify the one that contains the app name. The APK file is regarded as the APK of the data clone app. Then it collects DEX files in the APK file, i.e., ODEX files in the platform app directory, manufacturer-specific VDEX files (e.g., `boot-framework.vdex` and `wifi-service.vdex`), and framework resource files (e.g. `framework.apk`). Next Poirot combines all those files and disassembles them using APKTool. With the generated bytecode, it then leverages JEB [7] to decompile it to high-level source code. Data and control dependencies can be retrieved for further analysis (see Section 3.4). Note that Poirot uses an existing callback control flow analysis approach [35] to deal with multi-thread programming and inter-component communication.

Native Code Analysis. Many customized components of platform apps are implemented as native code executables, and thus Poirot conducts native code reverse engineering. It first analyzes the platform app and searches for specific APIs (e.g., `exec`) and inter-process communications (e.g., local socket); then it collects the involved executables under the system folder (i.e., `/system/bin/`). Finally, Poirot reverses the binary code by using `IDA` and `IDAPython`. We

then manually analyze these system services to understand how a data clone app accesses and backs up/restores system data (e.g., data in system folders) through a native executable.

3.3.2 Dynamic Instrumentation. With the help of the Frida dynamic code instrumentation framework, Poirot provides an instrumented system to execute the data clone operation. By rooting an Android mobile phone (e.g., unlocking the bootloader and flashing a customized kernel image), Poirot instruments sensitive APIs in the identified executables (e.g., data clone apps, system libraries). While executing the data clone procedure, Poirot collects information including the accessed user data, the invoked system APIs (especially those related to network I/O and cryptography) and the used parameters, as well as the involved permissions (app permissions and SEAndroid types).

Since some models of Android mobile phones cannot be rooted, we install the universal version of the data clone app[5] on a rooted mobile phone (solution of challenge 2). Although the universal data clone app is not granted *signature* permissions to access the specific sensitive data, user data (e.g., contacts information) that are protected by *dangerous* and *normal* permissions are still accessible. Poirot can obtain adequate information for further analysis because the data transmission protocol for data transfer is the same.

3.3.3 Network Inspection. Through the network inspection component, Poirot monitors network traffic between two Android mobile phones relying on the tcpdump data-network packet analyzer [2]. For the mobile phones that cannot be rooted, the universal versions of the data clone apps are installed on the other rooted mobile phones for data transmission. If the universal version is not provided, we run the data clone apps on two unrooted Android phones to execute the data clone procedure and then leverage Poirot to intercept network packets by launching an address resolution protocol (ARP) spoofing (solution of challenge 3). The captured network packets are analyzed and modified by a laptop through the man-in-the-middle attack.

To reduce noise, Poirot drops ARP, DNS, and ICMP packets from the captured network traffics because data transmission over the private Wi-Fi LAN does not need these protocols. For data transmission that use common protocols, such as HTTP, FTP, and WebSocket, Poirot parses the packets with the help of Wireshark [9]. Otherwise, Poirot simply records the raw transferred data for further analysis.

Note that Poirot also supports network traffic differential analysis because we can select the data type before transferring the data. Thus, the packet format for transferring different types of data can be determined.

3.4 Analysis Process

Relying on the above three components, we use Poirot to conduct the following analyses to examine data clone services.

3.4.1 Data Export/Import Analysis. We investigate whether a data clone service (unintentionally) exposes the operated user data to a third-party app. Given a data clone app, Poirot first queries the `requestedPermissionsFlags` array in the class `PackageInfo` class to obtain a complete list of its used permissions directly. Apart from the listed permissions, Poirot dynamically monitors the data clone procedure because some permissions are granted dynamically in the new versions of Android systems. To retrieve a completed permission list, Poirot monitors file I/O operations, API invocations, involved system services (e.g., `BackupManagerService`).

To ensure the effectiveness of data transmission, the transferred user data are temporarily stored at the flash storage as intermediate data. Besides, most data clone services adopt an asynchronous mechanism, which uses different threads to handle data transmission and data export/import, respectively. Therefore, Poirot monitors file I/O operations to identify where the intermediate data are stored and checks whether these intermediate data are properly cleaned after transmission.

For the system services, Poirot determines the external system services that are invoked by the data clone app and then identifies the required permissions for these services. As user data are operated on during the data clone procedure, Poirot determines the services that are requested and checks which system privilege is invoked for the data operation. For instance, a data clone app utilizes the `PackageManager` service to restore user data and then leverages the `adb backup` functionality to move user data into the sandbox. Accordingly, Poirot first identifies the service that is invoked to restore and move user data. It then learns the permissions required to operate on the user data.

After having identified the permissions used by the data clone service, Poirot analyzes whether the user data are protected properly. First, it examines the *dangerous* permissions and identifies the unintentionally exposed components by executing Drozer [4] [22]. We manually analyze the functionalities defined in the components and determine whether the components are vulnerable. Third-party apps can access sensitive data illegally after exploiting these exposed components. As manufacturers embed customized system components into the standard Android system to support user data import and export, Poirot then checks the embedded system components to verify whether a third-party app will execute them.

3.4.2 Data Transmission Analysis. Poirot further checks the confidentiality and integrity of the transferred data. It analyzes data transmission from three perspectives: application-level transmission protocols, transmission formats, sender/receiver identity and data integrity checks. Although Poirot can analyze data transmission between two Android mobile phones by reverse engineering data clone apps (static analysis), the information obtained from the network-related code snippet is incomplete. Hence Poirot uses a hybrid methodology.

First, Poirot relies on the network inspection component to capture network traffics while data are being migrated. Two steps are followed to analyze the network traffics:

(1) Poirot leverages Wireshark [9] to parse network traffic and identify whether any known protocol (e.g., HTTP, FTP) is used.
(2) If a proprietary protocol is used, we manually execute the data clone procedure by transferring each data type in two comparable groups. Then Poirot intercepts the transmitted

[5] Apart from the platform apps, the manufacturers usually develop the universal versions of the data clone apps that can be installed on the phones published by the other manufacturers.

network packets and conducts a differential analysis to distinguish **data payload** and **meta data payload** for further inspections. Consider an example of transferring media files. We first transfer data in group A (with photos only) and intercept the network traffic. Then we send data in group B (with audio audio files and photos) and intercept the corresponding network traffics. Finally, Poirot identifies the differences between the network traffic of these two groups. By analyzing the network traffic generated by group A only, we pinpoint the data payload in the proprietary protocol.

Second, Poirot uses a heuristics to statically analyze the data clone app. Since the I/O operations are involved during the data clone procedure, Poirot identifies the I/O related network functions such as `getOutputStream` and `write()`. When third-party network application frameworks (e.g., Netty [8], Apache MINA [5]) are used, Poirot locates the critical encoding APIs that will be applied before network I/O (e.g., `encode()` in Apache MINA). Besides, protection operations for user data, such as data encryption, might also be used before data transfer. Therefore, Poirot identifies the typical cryptographic APIs (e.g., `Cipher.getInstance()`, `BigInteger.modPow`, and `MessageDigest.getInstance`) and infers whether checksum algorithms are implemented.

Finally, Poirot verifies whether user data are transferred properly by conducting an active penetration test. Given the data payload, it executes format identification to check data confidentiality. If the format of the data payload is text-based (e.g., HTTP-plaintext/base64, json), Poirot directly extracts its contents. If the data format is binary, Poirot leverages binwalk [6] to extract the potential contents (e.g., a JPEG file). Once meaningful contents are recognized, Poirot reports a violation against data confidentiality in the data clone service. In addition, Poirot tampers either the data payload (as well as the attached checksum) or the meta data to check whether the data clone app on the receiver side verifies the user data. If a data payload (e.g., an APK file) can be replaced or a checksum can be forged, Poirot further reports a violation against data integrity in the data clone service.

3.4.3 Wi-Fi Configuration Analysis. For the Wi-Fi hotspot-based data clone service, the protection strategies (e.g., authentication) of the WLAN are essential. Poirot analyzes the WLAN protection strategies from two aspects: generation rules of SSID/password and connection restriction.

Generation rules of SSID/password. Poirot executes the static code analysis component to locate the code snippets that are relevant to the Wi-Fi hotspot setup and recover the generation rules of the SSID/password declared in each app. To set up the Wi-Fi hotspot, the `wifiConfiguration` API needs to be invoked. Hence, Poirot locates the code snippet with `wifiConfiguration` and then retrieves the value assigned to the variables `SSID` and `preSharedKey`[6].

For Android version 7.0 and below, the `setWifiApEnabled` API in `WifiManager` class is invoked to start the Wi-Fi hotspot. Poirot pinpoints `setWifiApEnabled` to recognize where variables `SSID` and `preSharedKey` are declared and their assigned values. Starting from the values of `SSID` and `preSharedKey`, Poirot performs backward program slicing to track the statements that are (directly/indirectly) data dependent on the values of `SSID` and `preSharedKey`. The correlated statements are regarded as the generation rule to generate SSIDs/passwords. Within those correlated statements, Poirot identifies whether there is any fixed string that is used to generate either SSID or password. If so, Poirot labels such a generation rule as insecure.

It is important to mention that from Android version 7.1 to version 9.0, the `startTethering` API in `ConnectivityManager` class should also be used together with the `setWifiApEnabled` API. Hence, Poirot locates `startTethering` when analyzing these Android versions.

Connection restriction. We argue that for security only two Android phones are allowed to join the WLAN because the data clone process is a peer-to-peer data migration. Any other WLAN connection requests should be declined even if the correct Wi-Fi password is provided. To test this, we manually execute a data clone app to set up the Wi-Fi hotspot and use another data clone app to join the private network. Then, we use one or more Android phones to join the network. If these Android phones can join the network successfully, we define the setting of data transmission as highly risky.

4 EVALUATION

In this section, we report our analysis results against eight customized data clone services developed by different Android phone manufacturers. We first obtained implementation details of those data clone services with the help of Poirot and then identified related attacks against insecurely implemented services. We also check the new versions of the data clone services to identify whether the vulnerabilities still exist.

4.1 Experimental Targets

We investigated the popular Android phones and discovered eight customized data clone services supported by well-known manufacturers, including Gionee, Huawei, Nokia, Motorola, OnePlus, Oppo, Vivo, and Xiaomi. All phones are off-the-shelf products released during 2015-2019. Details about the tested phones are given in Table 2.

The Android systems installed on these Android phones are from version 5.0 to the latest mainstream version 9.0 (by 2019). The corresponding platform app developed by each manufacturer for implementing the data clone service is also listed. It is worthy noting that the mobile phones by Gionee are security-enhanced Android phones with an EAL4+ certificate (No. ISCCC-2016-VP-304).

In our experiments, five data clone services (i.e., Huawei, Motorola, Nokia, OnePlus, and Xiaomi) are analyzed by using at least one rooted phone. For the other three data clone services (Gionee, OPPO, and Vivo), we extracted the universal versions of the data clone apps and installed them on the rooted Motorola (Android 5.0), Huawei (Android 7.0), and Xiaomi phone (Android 8.0) for analysis.

To conduct the security analysis for testing each data clone service, we prepared two Android mobile phones developed by each manufacturer, phone A (p_A) and phone B (p_B) with the built-in data clone services, to simulate the phone-to-phone data clone

[6]Note that since Android version 10, Google has suggested to use `WifiNetworkSpecifier.Builder` to create `NetworkSpecifier` and `WifiNetworkSuggestion.Builder` to create `WifiNetworkSuggestion`. However in our study most Android phones still use Android version 9.0 and below, and thus in this paper we focus on the `WifiConfiguration` class.

procedure. Before the test, we also stored user data in p_A including contacts, SMS messages, installed apps, etc., and executed a factory reset for p_B. Then we ran the data clone apps on p_A and p_B to transfer data.

Table 2: Analyzed data clone services and their related Android devices

Device	Clone App	Phone A	Phone B	System
Gionee	*Ami GdataGhost*	GN8002S	GN8003	Android 6.0
Huawei	*PhoneClone*	P9*	Mate8	Android 7.0
Motorola	*Migrate*	XT1079*	XT1085*	Android 5.0
Nokia	*PhoneCloner*	X5*	X5*	Android 9.0
OnePlus	*BackupRestore*	5T*	5*	Android 9.0
OPPO	*BackupAndRestore*	A37m	R9 Plusm A	Android 5.1
Vivo	*EasyShare*	U1	U1	Android 8.1
Xiaomi	*Backup*	MI5	MI6*	Android 8.0

*: the phone is rooted

4.2 Analysis Results

Having the details about the Android phones, we first analyzed the data types that can be transferred. The transferred data types supported by each manufacturer are listed in Table 3. We found that all the services, except for the Motorola service, support transferring all types of sensitive data (i.e., communication records, files on SD card, app-related data, and system settings). Next, we utilized Poirot to assess the security risks these services may suffer from. The analysis results are discussed in what follows.

Table 3: Supported data types for each data clone service

Service	On-device Attack	Network-level Attack
Gionee	vulnerable	vulnerable
Huawei	-	partially vulnerable
Motorola	vulnerable	-
Nokia	-	-
OnePlus	vulnerable	vulnerable
OPPO	vulnerable	-
Vivo	-	vulnerable
Xiaomi	vulnerable	vulnerable

4.2.1 Data Export & Import. We inspected all data clone services supporting data export/import. When the data clone app is granted with *normal* and *dangerous* permissions, communication records, SD card data, and APK files can be operated on. Since all the data clone apps are installed as platform apps on their own brands of Android phones, *signature* permissions are granted, and thus system settings and install apps are accessible without requiring the access to be granted by the users. Note that the data clone services of Motorola and Nokia are not allowed to operate on app data and system settings, even though they possess the `signature` level permissions. In addition, we found that six manufacturers implemented supporting system components to help data clone services access the protected data (e.g., WLAN history and app data). Among them, Xiaomi and Vivo modified the original Android `BackupManagerService` to ignore the `android:allowBackup="false"` flag and thus their data clone apps can backup private data of **ALL** the installed apps. The customized processes with root privilege were introduced by Huawei[7], OnePlus[8] and Oppo[9], whereas Gionee integrated a built-in `superuser` executable to grant root privilege to its data clone service. For these services requiring sensitive permissions, Poirot did not detect explicitly exposed components, which indicates that a third-party app could not utilize interfaces of these services directly.

The code analysis and the network traffic analysis demonstrated that all eight data clone services transfer data separately. That is, data are first packed into files of different formats and then each file is sent individually. By utilizing the dynamic instrumentation of Poirot, we found that three data clone services (that is, Huawei, Motorola, and Vivo) store intermediate files at the app sandbox, while the other five data clone services (that is, Gionee, Nokia, OnePlus, OPPO, and Xiaomi) use the SD card as buffer for intermediate data. This leads to a temporary file retrieving attack (see details in Section 4.3).

4.2.2 Data Transmission. We found that there are significant differences between data transmission protocols adopted by each data clone service (see the last two columns of Table 4). The data clone services of Gionee, Nokia, OnePlus, OPPO, and Xiaomi adopt their proprietary, TCP based protocols. The data clone services of Huawei, Motorola, and Vivo use multiple protocols to transfer different types of data. The Huawei data clone service uses the File Transfer Protocol (FTP) to transfer user data, a customized UDP-based heartbeat protocol to keep a long-lived connection, and a proprietary TCP-based protocol to send control commands. The Motorola data clone service utilizes the standard HTTP protocol to transfer communication records and a TCP-based proprietary protocol to transfer files stored on the SD card. The Vivo data clone service utilizes the HTTP protocol to transfer user data while an additional WebSocket-based heartbeat protocol is used to maintain the connection.

With the help of Poirot, we successfully located the code snippets related to data packing and data sending/receiving in all eight data clone apps. We then conducted a manual reverse engineering to recover the formats of transferred data. For the data clone services of Nokia and Gionee, the transferred data are serialized using `writeObject`. The Nokia data clone service separates communication sessions into metadata sessions and raw content sessions, while the Gionee data clone service combines the metadata and the content of transferred data in the same session. The data clone services of OnePlus and OPPO share a similar solution that transfers user data without compressing them. The difference is that the OPPO data clone service uses only one session to send all data while the OnePlus data clone service establishes several sessions to transfer data payload and other metadata, respectively. The data clone service of Xiaomi transfers user data in both the text-format and the binary-format. The text-format user data contain JSON-serialized control commands and file metadata, and the binary-format files contain the raw content of the transferred data. The data clone

[7] `/system/bin/filebackup`
[8] `/system/bin/br_app_data_service`
[9] `/system/bin/br_app_data_service`

Table 4: Wi-Fi and protocol features of data clone services (Strings in red are constant SSIDs)

Service	Hotspot AP	SSID		Passwd		Protocol	Server Port
		Fixed	Pattern	Fixed	Pattern		
Gionee	sender	✓	AmiClone_**DN**	✓	"Lss19900716"	TCP	5024
Huawei	receiver	✗	**DN**%*nn*%CloudClone	✗	*nnnnnnnn*	FTP/TCP/UDP	-
Motorola	receiver	✗	DIRECT-*pp*-**DN**_*iiii*[a]	✗	*ppppppp*	TCP/HTTP	6000
Nokia	receiver	✗	AndroidShare_*nnnn*	✗	RandomUUID	TCP	8988
OnePlus	receiver	✗	**DN**_co_ap*llll*	✗	*lllnnnn*	TCP	8940
OPPO	receiver	✗	**DN**_co_ap*llll*	✓	*iiiiiii*	TCP	8939
Vivo	sender	✓	Vivo#**DN**#*ii*	-	-	HTTP/WebSocket	10178
Xiaomi	receiver	✓	*str*[b] **DN**	✓	SSID-related	TCP	57383-57386

DN: device name or device model; *n*: a random number; *p*: a random printable character such as number or letter; *l*: a random lower case letter; *i*: a fixed character once generated at first time.
[a] an example is "DIRECT-2k-XT1085_8e6e" in which "DIRECT" is a fixed string, "XT1085" is device model, "2k" is a random string that changes for each time of execution and "8e6e" is a fixed string.
[b] *str* is a device-specific, base64-encoded string, which is constant for a certain smartphone.

services of Motorola transfers data (SD card files with TCP, communication records with HTTP) without any packing. The Vivo data clone service adopts a data transmission with HTTP to transfer different data files separately, and part of the data are packed (media data are compressed as a .zip file, and app data are packed as files with Android Backup format). The Huawei data clone service packs most user-generated data (communication records, app data, system settings) into SQLite3 (.db) files, but it sends APK files and media files directly or archives them as a .tar file.

In short, we found that exported data and metadata (i.e., file paths, file sizes, and checksums) are encoded into different files with customized packing format. But **NONE OF THEM** encrypts these files. All eight services only rely on the protection of the established private Wi-Fi network, and thus fail to guarantee data confidentiality during their data clone process. In addition, we checked how each service validates the integrity of transferred data. Surprisingly, none of those data clone services executes a robust data integrity validation: they either miss the data integrity check or incorrectly implement it. Only Huawei and Xiaomi employ data checksum validations (HMAC-SHA256 and CRC-32, respectively) in their protocols, while the other services do not use any checksum for integrity validation. And even though the Xiaomi service attaches a CRC checksum to each transferred file, for some unknown reasons the data clone app just ignores the check. Thus, even if the transferred data does not match its CRC checksum, the data clone app on p_B still accepts it.

4.2.3 Wi-Fi Configuration. Our experiments inspected the following three aspects of the established Wi-Fi network:

SSID randomness: By reverse engineering the data clone platform apps, we found that most apps generate an SSID in a certain format. This SSID can be used as a fingerprint of its corresponding data clone service. The SSID generation rules are shown in Table 4. Among them, the services of Gionee, Vivo, and Xiaomi always generate constant SSIDs on the same device. The SSIDs generated by other services follow certain rules; thus one can learn the patterns of those services in advance and continuously scan the Wi-Fi signals to wait for a matched SSID. When an expected SSID is scanned, it indicates that a data clone process is starting. Subsequently, the attacker can try to circumvent the password authentication (see next paragraph) and join the network and intercept messages exchanged by the two vulnerable devices.

Password strength: We found that many data clone services do not randomly generate Wi-Fi passwords. Instead, they adopt insecure password generation rules. Table 4 lists the recovered Wi-Fi password generation rules. In detail, only the data clone service of Nokia implements a secure password generation (that is, it uses the default configuration provided by the Android system). For other data clone services, we found the following flaws:

a) Predictable Passwords: The data clone service of Xiaomi generates the password by hashing the SSID and fetching the first four bytes in hex format (e.g., "A1B2C3D4"). Any attacker could directly calculate the password since both the SSID and the hash algorithm are publicly known. The password used by the Gionee data clone service is a hard-coded constant string in its platform app. The Wi-Fi network established by Vivo data clone service is an open WLAN without any password protection. The passwords created by OnePlus and Huawei data clone services suffer from several issues. First, both services provide partially-random passwords. Only four digits (10^4 candidates) are random in the OnePlus data clone service, while eight digits (10^8 candidates) are random in the Huawei data clone service. A brute-force attack against the Wi-Fi network handshake packet can allow the attacker to obtain the password within at most 50 seconds, using a state-of-the-art GPU such as RTX 2080 Ti. Second, theOnePlus data clone service uses the system time as the random seed, which has inadequate information entropy and is easily to be guessed.

b) Passwords Leakage: We found that although the passwords generated by the Oppo and Motorola data clone services are random and unpredictable. the Android systems of the tested devices (version below 8.0) provide a reflection mechanism to invoke the system API `getWiFiApConfiguration` that allows any apps to query the SSID and the password of the established Wi-Fi network. A third-party app on the same device thus can easily obtain the password of the private WLAN [23].

Table 5: Security overview of the analyzed data clone services

Service	Unpredicatable SSID	Secure Wi-Fi Password	Connection Restriction	Protected Temporary Data	Encrypted Transmission	Integrity Check	Device-to-device Authentication
Gionee	✗	✗	✗	✗	✗	✗	✗
Huawei	✗	✗	✓	✓	✗	✓	✗
Motorola	✗	☹	✗	✓	✗	✗	✗
Nokia	✓	✓	✗	✗	✗	✗	✗
OnePlus	✗	✗	✗	✗	✗	✗	✗
OPPO	✗	☹	✗	✗	✗	✗	✗
Vivo	✗	✗	✗	✓	✗	✗	✗
Xiaomi	✗	✗	✗	✗	✗	☹	✗

symbol ☹ denotes that under certain circumstances (e.g., a malicious app is installed) the requirement may be violated.

Connection restriction: We argue that the private Wi-Fi network used in a data clone service should strictly validate the connected devices. Unfortunately, we found that all the Wi-Fi networks except for the one established by Huawei do not restrict the number of connected devices. In addition, none of the eight data clone services employ a phone-to-phone authentication and thus any mobile phone with the password could join the WLAN. This could lead to a sandbox data extraction attack (see Section 4.3).

4.3 Attacks

To demonstrate how these insecure implementations threat user data privacy, we designed two types of attack: a on-device attack and a network-level attack. We first defined seven essential requirements that a secure Wi-Fi hotspot based data clone service should satisfy. Table 5 lists for each considered data clone service the requirements satisfied by the data clone service. As shown in the table, none of the analyzed services satisfies all requirements, and some of them do not even satisfy one of those requirements. We also report in Table 6 whether each data clone service is vulnerable to our proposed attacks, and detail how data clone services are threatened by these attacks.

4.3.1 On-device Attack. An on-device attack is launched by a malicious app installed on either the old Android phone or the new one. The only requirement for this attack is that the malicious app has a permission belonging to the `STORAGE` permission group to read the files on the SD card. The security issue here is that data clone services do not protect the intermediate files generated and stored on the SD card. Even though the intermediate files are deleted right after transmission, data packing is in general much faster than data transmission and thus there exists a relatively long time window for a malicious app to copy files from this directory.

As Table 5 shows, we found that five data clone services (i.e., OnePlus[10], Xiaomi[11], OPPO[12], Gionee[13], Nokia[14]) are vulnerable to this attack. We have developed a proof-of-concept (PoC) malicious app to conduct this attack (installed on either p_A or p_B). The app keeps monitoring the data buffer directories to check whether a temporary file is written. If such a file is written, the app immediately copies it to a new place on the SD card to store the data permanently. We found that after a complete data clone procedure, our PoC app collected all temporary files, and then extracted sensitive user data from those files.

We argue that this attack is particularly applicable to a digital forensic scenario, in that a forensic analyst can force one Android phone to export its personal data by using another phone with our PoC app as the receiver. This is a typical security risk when the user data (especially the app-related data) are assumed to be forensic-resistant (i.e., even though a forensic analyst knows the unlock password, he cannot retrieve the data in the sandbox).

4.3.2 Network-level Attack. In this attack, the attacker monitors the privately built Wi-Fi network instead of installing a malicious app on the user's mobile phone. The steps of this attack are as follows. The attacker first has to detect these privately built Wi-Fi networks by identifying the specific SSID patterns. Then the attacker persistently monitors the Wi-Fi signals until a certain Wi-Fi is established. The attacker has then to crack the password and connect to the WLAN; as we have seen in our analysis, the passwords used to protect these Wi-Fi networks are often not strong enough and thus the attacker can quickly crack them. Once the password is cracked, the attacker can eavesdrop on the Wi-Fi and tamper data using tools such as Ettercap. We notice that all data transmission protocols we have analyzed in this work are implemented with an incorrect data integrity validation, and even without any data confidentiality protection. It is important to note that this attack requires the attacker to be physically close to the attacked phones and continuously monitor Wi-Fi signals in order to detect when a data clone activity is taking place[15]. Therefore such an attack is unlikely to be carried at a large scale. Rather it is an attack that is more likely to be carried out against targeted parties (e.g., individuals under surveillance).

As we can see from Table 5, the SSID of the Wi-Fi hotspot generated by all data clone services but Nokia can be predicted, and four Wi-Fi networks (that is, Gionee, OnePlus, Vivo, and Xiaomi) are vulnerable to network-level attacks because of the predictable SSIDs and insecure Wi-Fi passwords they use. As a result, attackers could circumvent authentication and connect to the WLAN to conduct further attacks against the legitimate phones. Note that although Huawei service uses predictable SSID and insecure passwords, it

[10] /sdcard/opbackup/ChangeOver
[11] /sdcard/MIUI/backup/Transfer/
[12] /sdcard/Backup/ChangeOver/
[13] /sdcard/amihuanji/temp/
[14] /sdcard/backup/

[15] Notice of course that the attacker can install a hidden device with the same capabilities in the proximity of the targeted party.

restricts the number of connected clients. An attack could only be conducted if the legal receiver is excluded and replaced by a malicious receiver. Thus we label Huawei service as "partially vulnerable" to network-level attack.

We utilized the network inspector of Poirot to conduct man-in-the-middle attacks as mentioned in Section 3.3.3. Despite passive data eavesdropping, the attacker could actively tamper the transferred data to inject malicious contents. We found that in our investigation, p_B unconditionally trusts all the cloned data. If an attacker controls p_A and installs malicious apps, those malicious apps are transferred and installed to p_B without prompting warning or scanning malicious code. More seriously, the attacker could modify the app data instead of the APK file. In this situation, a benign app would be exploited easily (e.g., by trusting a file with a malformed format in the app sandbox).

Table 6: Security evaluation regarding different attacks against data clone services

Service	Communication Records	Files on SD card	App-related Data		System Settings
			APK	App data	
Gionee	✓	✓	✓	✓	✓
Huawei	✓	✓	✓	✓	✓
Motorola	✓	✓	✗	✗	✗
Nokia	✓	✓	✓	✗	✗
OnePlus	✓	✓	✓	✓	✓
OPPO	✓	✓	✓	✓	✓
Vivo	✓	✓	✓	✓	✓
Xiaomi	✓	✓	✓	✓	✓

We also discovered a particular network hijacking case in the OPPO data clone serive. The data transmission thread on p_B does not check whether its used TCP port (i.e., 8939) is available. Even worse, the UI of the platform app does not prompt any error information when the data clone service fails to bind the TCP. As a result the data clone service on the other device will continue transfer data without verifying the identity of the receiver. Suppose that a malicious app on p_B occupies the 8939 port before a data clone process starts, p_A will communicate with the malicious app on p_B and thus sends user data to the malicious receiver.

4.4 Manufacturer Feedback

We have reported the discovered vulnerabilities and the consequences to the corresponding manufacturers in September 2019. Followed by our report, we also provided the suggestions to fix the vulnerabilities. Among the seven manufacturers whose data clone services are vulnerable, three of them (OnePlus, Vivo, and Xiaomi) have recognized the vulnerabilities and fixed them[16]. We then checked the most recent version of their data clone services. We found that OnePlus has addressed the data leakage issue by encrypting the sensitive data (e.g., contacts and SMS messages) before data transmission. Instead of establishing an open WLAN, Vivo now protects the WLAN by adding the password requirement. The password consists of 8-12 digits of numbers and letters (in upper/lower cases), which is difficult to crack via brute force attacks. Xiaomi now generates a pseudo-random SSID for each device comprised by `DeviceName_` and 4 digits of pseudo-random numbers. For the other data clone services, we observe that some manufacturers (i.e., Oppo, Motorola, Gionee and) stopped updating the older version of the systems (Android 5.1, 6.0, and 7.0) since 2018, and thus the data clone services were not updated either.

[16]We also earned bounty awards from those companies.

5 RELATED WORK

App Analysis. As functionalities of mobile apps become abundant, a large number of efforts have focused on large-scale security analysis of Android apps. Identifying data leakages is the goal of many such analysis efforts and tools have been developed to facilitate the analysis. FlowDroid [12] and PiOS [14] statically analyze app code to track sensitive data flow. FlowDroid [12] optimizes previously proposed static taint-analysis approaches relying on context, flow, field and object-sensitivity information. To detect potential privacy leaks, FlowDroid draws a complete Android lifecycle by including callbacks handling and UI widgets within the apps. Instead of analyzing Android apps, PiOS targets iOS apps by analyzing whether the proposed vetting process may leak sensitive data. Through static analysis, it checks code paths of each app and pinpoints where the app first accesses sensitive information and then transmits it over the network. Due to the lack of source code, PiOS analyzes apps that are developed in Objective-C code. However, app code analysis is insufficient when analyzing the data clone service because the data transmission procedure is not covered by code analysis.

Apart from analyzing the entire Android apps, some approaches focus on certain issues, such as cryptographic misuses, unauthorized access, and backdoor functionalities. As cryptographic primitives are sometimes implemented incorrectly, many approaches have been proposed to identify cryptographic vulnerabilities. By gathering the cryptographic vulnerabilities identified by the other detection tools, such as FixDroid [25]. CrySL [20] and CryptoLint [13], CryptoGuard [26] is proposed with a set of detection algorithms. To address the false positive issues, a number of refinement algorithms based on empirical observations about common programming idioms and language restrictions have been proposed so that the irrelevant statements are removed to reduce false alarms. Kratos [28] finds security issues in access control systems implemented in Android systems. By constructing a precise call graph including all execution paths, it identifies the paths that third-party apps with insufficient privilege are able to access sensitive resources. Similarly, InputScope [38] focuses on the hidden functionalities (e.g., backdoors and blacklists to block unwanted content) implemented in Android apps. Through static taint analysis and backward slicing, it checks whether the input data match with the data stored in the app or retrieved over the network to exploit the hidden secrets. Instead of detecting a general type of vulnerability from the Android app, we identify the potential vulnerabilities based on the potential threats against the data clone service.

Apart from the static analysis, tools such as TaintDroid [15] and Charm [31] produce realtime results by executing certain components dynamically. TaintDroid detects misbehaving apps by tracking the flow of privacy-sensitive data through third-party apps. By labeling data from privacy-sensitive sources, it identifies sensitive data propagation through program variables, files, and inter-process

messages dynamically. Charm [31] identifies vulnerabilities and bugs in device drives. It executes the remote device driver in a virtual machine for only servicing the low-level and infrequent I/O operations through the USB channel. Then analysts are able to use Charm for manual interactive debugging, record-and-replay, and enhancing fuzzing. Since both app code and data transmission need to be analyzed, we combine the static and dynamic analysis to provide a complete analysis of the data clone procedure. To the best of our knowledge, none of these security researches have analyzed data clone apps.

OS Customization Analysis. In order to support a functionality, manufacturers might need to modify the standard Android system by embedding support components. However, such customized components often result in vulnerabilities due to incorrect implementation [11, 17]. Some previous works have focused on different customized components for vulnerability detection including customized Android phone driver [39], permission re-delegation [16], vendor-specific certification [32], and insecure validation [37]. In addition, the corresponding platform apps developed by manufacturers might be vulnerable [18, 34]. Our study extends those works by revealing the insecurity of data clone services.

Specifically, Woodpecker [19] identifies leakages of permissions or capabilities. By conducting data flow analysis, it explores the reachability of each dangerous permission from a public interface. Furthermore, Woodpecker exploits publicly-accessible interfaces and services with and without requesting permissions from the other apps to check the explicit capability leaks and implicit capability leaks. Harehunter [10] detects hanging attribute references (Hares) vulnerability. A Hares vulnerability occurs when an inter component communication (ICC) call refers to a non-existing attribute (e.g., package, activity, service) due to the customization of Android system. A malicious app could claims itself as the definition party of such attributes and hijack the ICC call. In response, Harehunter compares all the attributes defined by the system app and their corresponding references to find potential null-reference flaws. Nonetheless, because more customized components are embedded to support the data clone procedure, security analysis of data export/import cannot be accomplished by code analysis only. In contrast, our analysis combines code reverse engineering and data analysis. Specifically, we extend the study of Shu *et al.* [30] by discovering temporarily stored data could also be a source of sensitive information retrieving.

Device-to-device (D2D) Communication Analysis. Data clone process is related to D2D communication. A survey investigating potential security threats of D2D communications is by Wang *et al.* [33]. Similarly, Liu *et al.* [23] presented an in-depth empirical security analysis on mobile D2D network between two Android devices. A particular case of smart config Wi-Fi provisioning has been revealed to be very insecure and could lead to the Wi-Fi password leakage [21]. Our analysis further studies security issues of Wi-Fi hotspot, Wi-Fi direct, and Bluetooth used in customized data clone services.

Several approaches have been proposed to secure the D2D network communications. Shen *et al.* [29] first discussed several attacks against Wi-Fi D2D communications such as man-in-the-middle attacks and denial-of-service attacks. They also proposed two protocols, authentication-string-based key agreement protocol and SAS-based key agreement protocol, to secure the communication between Android mobile phones. Besides, Raju *et al.* [27] proposed a security protocol to protect individual confidentiality. To address the vulnerabilities in existing public Wi-Fi hotspots, such as weak encryption and lack of confidentiality, they designed a solution to eliminate the dependency on pre-shared information. Unlike previous work, we focus on the practical aspects (implementation level) to assess whether the Wi-Fi hotspot is securely set up (i.e., SSID/password) and whether the integrity and confidentiality of the transferred data are assured.

6 CONCLUSION

In this paper, we have investigated the security of popular Wi-Fi hotspot based data clone services provided by Android phone manufacturers. We developed an analysis system to help analysts understand implementation details of closed source data clone services, and proposed three security analysis approaches to detect security flaws in those services. We evaluated our proposed system and approaches by assessing eight data clone services designed by mainstream Android phone manufacturers. The results show that those data clone services are vulnerable to four specific attacks; millions of released Android phones would thus be vulnerable if they were to execute such a data clone procedures. We have reported the discovered security issues to corresponding manufacturers and helped some of them to fix the flaws. We also claim that developers should be aware of those risks when designing a similar service.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their feedback, and our shepherd, Manuel Egele, for helping improve this paper. This work was partially supported by National Natural Science Foundation of China (Grant No.62002222 and No. U1636217), the Major Project of the Ministry of Industry and Information Technology of China (Grant No.2018-36). We especially thank Ant Financial Services Group for the support of this research within the *SJTU-AntFinancial Security Research Centre*.

REFERENCES

[1] 2012. Android. http://www.android.com/.
[2] 2019. tcpdump. http://www.tcpdump.org/.
[3] 2020. ADB (Android Debug Bridge) - Android Developers. https://developer.android.google.cn/studio/command-line/adb.
[4] 2020. Drozer. https://github.com/FSecureLABS/drozer.
[5] Accessed 2020. Apache MINA. http://mina.apache.org/.
[6] Accessed 2020. binwalk. https://github.com/ReFirmLabs/binwalk.
[7] Accessed 2020. JEB. https://www.pnfsoftware.com/.
[8] Accessed 2020. Netty Project. https://netty.io/.
[9] Accessed 2020. Wireshark. https://www.wireshark.org/.
[10] Yousra Aafer, Nan Zhang, Zhongwen Zhang, Xiao Zhang, Kai Chen, XiaoFeng Wang, Xiao-yong Zhou, Wenliang Du, and Michael Grace. 2015. **Hare Hunting in the Wild Android: A Study on the Threat of Hanging Attribute References**. In *Proc. 22nd ACM Conference on Computer and Communications Security (CCS)*. ACM, Denver, CO, USA.
[11] Yousra Aafer, Xiao Zhang, and Wenliang Du. 2016. **Harvesting Inconsistent Security Configurations in Custom Android ROMs via Differential Analysis**. In *Proc. 25th USENIX Security Symposium (Usenix Security)*. USENIX Association, Austin, TX, USA.
[12] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick D. McDaniel. 2014. **FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps**. In *Proc. 35th Conference on Programming Language Design and Implementation (PLDI)*. ACM, Edinburgh, United Kingdom.

[13] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. **An empirical study of cryptographic misuse in android applications**. In *Proc. 20th ACM Conference on Computer and Communications Security (CCS)*.

[14] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. 2011. **PiOS: Detecting Privacy Leaks in iOS Applications**. In *Proc. 18th Annual Network and Distributed System Security Symposium (NDSS)*. Internet Society, San Diego, California, USA,.

[15] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick D. McDaniel, and Anmol N. Sheth. 2014. **TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones**. *ACM Trans. Comput. Syst.* 32, 2 (2014), 5:1–5:29.

[16] Adrienne Porter Felt, Helen J. Wang, Alexander Moshchuk, Steve Hanna, and Erika Chin. 2011. **Permission Re-Delegation: Attacks and Defenses**. In *Proc. 20th USENIX Security Symposium (Usenix Security)*. USENIX, San Francisco, CA, USA.

[17] Roberto Gallo, Patricia Hongo, Ricardo Dahab, Luiz C. Navarro, Henrique Kawakami, Kaio Galvão, Glauber Junqueira, and Luander Ribeiro. 2015. **Security and System Architecture: Comparison of Android Customizations**. In *Proc. 8th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. ACM, New York, NY, USA.

[18] J. Gamba, M. Rashed, A. Razaghpanah, J. Tapiador, and N. Vallina-Rodriguez. 2020. **An Analysis of Pre-installed Android Software**. In *Proc. 41th IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA.

[19] Michael C. Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. 2012. **Systematic Detection of Capability Leaks in Stock Android Smartphones**. In *Proc. 19th Annual Network and Distributed System Security Symposium (NDSS)*. Internet Society, San Diego, California, USA.

[20] Stefan Krüger, Johannes Späth, Karim Ali, Eric Bodden, and Mira Mezini. 2019. **Crysl: An extensible approach to validating the correct usage of cryptographic apis**. *IEEE Transactions on Software Engineering* (2019).

[21] Changyu Li, Quanpu Cai, Juanru Li, Hui Liu, Yuanyuan Zhang, Dawu Gu, and Yu Yu. 2018. **Passwords in the Air: Harvesting Wi-Fi Credentials from SmartCfg Provisioning**. In *Proc. 11thACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. ACM, Stockholm, Sweden.

[22] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel. 2015. **IccTA: Detecting Inter-Component Privacy Leaks in Android Apps**. In *Proc. 37th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Florence, Italy.

[23] Kecheng Liu, Wenlong Shen, Yu Cheng, Lin X Cai, Qing Li, Sheng Zhou, and Zhisheng Niu. 2018. **Security Analysis of Mobile Device-to-Device Network Applications**. *IEEE Internet of Things Journal* 6, 2 (2018), 2922–2932.

[24] René Mayrhofer, Jeffrey Vander Stoep, Chad Brubaker, and Nick Kralevich. 2019. **The Android Platform Security Model**. *CoRR* abs/1904.05572 (2019).

[25] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. **A stitch in time: Supporting android developers in writingsecure code**. In *Proc. 24th ACM Conference on Computer and Communications Security (CCS)*. Dallas, USA, 1065–1077.

[26] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng (Daphne) Yao. 2019. **CryptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects**. In *Proc. 26th ACM Conference on Computer and Communications Security (CCS)*. ACM, London, UK.

[27] Laiju K Raju and Reena Nair. 2015. **Secure Hotspot a novel approach to secure public Wi-Fi hotspot**. In *Proc. 3rd International Conference on Control, Communication and Computing India (ICCC)*. IEEE, Trivandrum India.

[28] Yuru Shao, Qi Alfred Chen, Zhuoqing Morley Mao, Jason Ott, and Zhiyun Qian. 2016. **Kratos: Discovering Inconsistent Security Policy Enforcement in the Android Framework**. In *Proc. 23rd Annual Network and Distributed System Security Symposium (NDSS)*. Internet Society, San Diego, California, USA.

[29] Wenlong Shen, Bo Yin, Xianghui Cao, Lin X Cai, and Yu Cheng. 2016. **Secure Device-to-Device Communications over WiFi Direct**. *IEEE Network* 30, 5 (2016), 4–9.

[30] Junliang Shu, Juanru Li, Yuanyuan Zhang, and Dawu Gu. 2018. **Burn After Reading: Expunging Execution Footprints of Android Apps**. In *Proc. 12thInternational Conference on Network and System Security (NSS)*. Springer, Hong Kong, China.

[31] Seyed Mohammadjavad Seyed Talebi, Hamid Tavakoli, Hang Zhang, Zheng Zhang, Ardalan Amiri Sani, and Zhiyun Qian. 2018. **Charm: Facilitating Dynamic Analysis of Device Drivers of Mobile Systems**. In *Proc. 27th USENIX Security Symposium (Usenix Security)*. USENIX, Baltimore, MD, USA.

[32] Narseo Vallina-Rodriguez, Johanna Amann, Christian Kreibich, Nicholas Weaver, and Vern Paxson. 2014. **A Tangled Mass: The Android Root Certificate Stores**. In *Proc. 10th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, Sydney, Australia.

[33] Mingjun Wang and Zheng Yan. 2017. **A Survey on Security in D2D Communications**. *Mobile Networks and Applications* 22, 2 (2017), 195–208.

[34] Lei Wu, Michael Grace, Yajin Zhou, Chiachih Wu, and Xuxian Jiang. 2013. **The Impact of Vendor Customizations on Android Security**. In *Proc 20th ACM Conference on Computer and Communications Security (CCS)*. ACM, Berlin, Germany.

[35] Shengqian Yang, Dacong Yan, Haowei Wu, Yan Wang, and Atanas Rountev. 2015. **Static Control-Flow Analysis of User-Driven Callbacks in Android Applications**. In *Proc. 37th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Florence, Italy.

[36] Ka-Ping Yee. 2004. **Aligning security and usability**. *IEEE Security & Privacy* 2, 5 (2004), 48–55.

[37] Lei Zhang, Zhemin Yang, Yuyu He, Zhenyu Zhang, Zhiyun Qian, Geng Hong, Yuan Zhang, and Min Yang. 2018. **Invetter: Locating Insecure Input Validations in Android Services**. In *Proc. 25th ACM Conference on Computer and Communications Security (CCS)*. ACM, Toronto, ON, Canada.

[38] Qingchuan Zhao, Chaoshun Zuo, Brendan Dolan-Gavitt, Giancarlo Pellegrino, and Zhiqiang Lin. 2020. **Automatic Uncovering of Hidden Behaviors From Input Validation in Mobile Apps**. In *Proc. 41th IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA.

[39] Xiaoyong Zhou, Yeonjoon Lee, Nan Zhang, Muhammad Naveed, and XiaoFeng Wang. 2014. **The Peril of Fragmentation: Security Hazards in Android Device Driver Customizations**. In *Proc. 35th IEEE Symposium on Security and Privacy (SP)*. IEEE, Berkeley, CA, USA.