Clone2Pwn: A Systematic Security Analysis of Data Migration Tools in the Android Ecosystem

Florian Draschbacher[1,2], Lukas Maar[1], Lorenz Schumm[1], Rene Denifl[3], Lukas Treffner[3], and Stefan Mangard[1]

[1] Graz University of Technology {first.last}@tugraz.at [2] A-SIT Austria
[3] Graz University of Technology {denifl,lukas.treffner}@student.tugraz.at

Abstract. To simplify the transfer of user data to a new device, all major smartphone manufacturers offer data migration (or *phone clone*) tools. Despite the sensitivity of the involved data, the security of current-generation data migration tools has not been analyzed comprehensively.
In this paper, we systematically analyze state-of-the-art data migration tools from seven major Android vendors (Google, Samsung, Vivo, Xiaomi, Oppo, Huawei, Honor) with a combined download count exceeding 12 billion. We identify generic attack scenarios, and check concrete apps for realizability of these scenarios based on a list of requirements. Our analysis uncovers severe security vulnerabilities in all tools that allow attackers within Wi-Fi range to eavesdrop sensitive user data from data migrations. The eavesdropped data includes communication records, personal media and documents, and login credentials, effectively representing a sizable dump of a user's digital life. In practice, attackers can exploit these vulnerabilities by placing cheap equipment near places such as mobile carrier stores, where data migrations are typically done multiple times per day. Moreover, five tools are vulnerable to data injection attacks, allowing attackers within Wi-Fi range to achieve code execution on a freshly set-up device. Finally, three tools are also susceptible to data extraction or injection attacks from local attackers. We responsibly disclosed our results to affected vendors, so far leading to eight CVEs, five of which are high-severity.

1 Introduction

To encourage device sales, mobile device manufacturers provide tools that facilitate data migration from an old device to a newly purchased one. Through tight OS integration, these tools can bypass sandbox boundaries to, e.g., seamlessly place imported messages into the suitable data store on the new device.

The data transmitted in a device migration typically represents a sizable dump of a person's digital life. In the wrong hands, this data could be misused to cause considerable harm. Data migration tools must therefore make sure that transmissions cannot be eavesdropped or manipulated. All data migration tools from major device vendors operate without Internet connection, either transmitting data through a peer-to-peer Wi-Fi connection or a cable. However, simply omitting a remote server from the system does not ensure secure transmission.

Despite the security sensitivity of data migration tools, they remain largely underexplored. The only prior study, conducted by Ma et al. [25] in 2019, reported that five of 8 tested tools leaked transmission data to disk and four were susceptible to wireless eavesdropping. While it provided an important first look at data migration security, it (i) excluded Android Switch (Google) and Smart Switch (Samsung)—the current market leaders—and (ii) reflected the landscape from about six years ago. More broadly, no subsequent work has re-evaluated these tools at scale or examined their security in today's ecosystem. Still, data migration tools have evolved substantially in recent years and expanded their user bases by orders of magnitude, rendering any vulnerabilities relevant to a large portion of smartphone users.

In this paper, we close this research gap through a comprehensive security analysis of today's data migration tool landscape in the Android ecosystem. We reveal that tools from the major vendors suffer from multiple vulnerabilities that can lead to leakage of sensitive data, and in some cases even code execution. Methodologically, we go beyond Ma et al. [25] by defining generic attack scenarios against these tools as cross-products of attacker goals and attack surfaces, enabling a systematic and extensible evaluation across tools. For each attack scenario, we then identify the attack requirements, i.e., necessary conditions that cause susceptibility to the scenario in a data migration tool. For example, wirelessly eavesdropping all transmitted user data requires a combination of weak link-level protection and lack of app-level encryption.

Brand	Tool	Downloads
Google	Android Switch 1.0.740410803	10B+
Samsung	Smart Switch 3.7.64.10	1B+
Vivo	EasyShare 5.10.5.3_gp	1B+
Xiaomi	Mi Mover 4.2.8	100M+
Oppo	Clone Phone 15.9.3 / 14.7.22_third_gp	10M+
Huawei	Phone Clone 14.5.0.550	10M+
Honor	Device Clone 14.8.5.306 / 11.6.1.320	1M+

Table 1: Data migration tools evaluated. Differing app versions for third-party devices are indicated separately.

Brand	Device	OS	Year
Google	Pixel 8a[s,t]	15	2024
Samsung	Galaxy S23[s,t]	14	2025
Samsung	Galaxy A54[s]	14	2024
Vivo	V40 SE 5G[t]	15	2025
Vivo	Y36[s]	13	2024
Xiaomi	Redmi Note 12 5G[t]	14	2024
Xiaomi	12[s]	14	2023
Oppo	A58[t]	15	2024
Oppo	A94 5G[s]	11	2023
Huawei	nova 12i[s,t]	14	2024
Honor	90 Lite[s,t]	14	2024

Table 2: Devices for evaluation, their Android version and purchase year. s/t = migration source/target.

Next, we evaluate seven of the most popular data migration tools in the Android ecosystem based on the identified attack scenarios. As shown in Table 1, these tools have been downloaded between one million and 10 billion times[4]. Most Android devices ship preinstalled with two of these tools. Besides Android Switch,

[4] Data according to Google Play or Huawei AppGallery listings.

which is shipped with every Google-certified Android device independent of manufacturer [20], the analysed tools account for $75\,\%$ of the Android market [9]. For each tool, we determine whether our attack scenarios can be realized, i.e., whether the tool's implementation fulfills the attack requirements causing susceptibility. Since most tools are vendor-specific and only preinstalled (with privileged data access) on that vendor's devices, we consider two constellations for each tool. These are first-party migrations between two devices from the same manufacturer, and third-party migrations, where the tool is only privileged on the new device. First-party migrations usually include more sensitive data only accessible to privileged apps, such as, e.g., Wi-Fi credentials, app-private data, and more. Throughout this paper, we use the vendor name (e.g., Google) to refer to their respective data migration tool (e.g., Android Switch).

Our analyses reveal that all inspected apps are susceptible to wireless attacks. This is because six of seven apps use no or trivially bypassable app-level encryption or integrity checks together with a weak WPA2 Pre-Shared Key (PSK). The remaining app uses Wi-Fi Direct with a peer identification scheme that can be spoofed. The resulting wireless attacks either allow eavesdropping on the transmitted user data, or even manipulating it on the fly, e.g., for code execution in the context of transmitted apps. An attacker can carry out these attacks by placing cheap equipment within Wi-Fi range of places like mobile carrier stores, where data migrations are a frequent occurrence [43, 42, 34]. Moreover, for three of seven apps, a local attacker, e.g., a malicious app, can also extract or manipulate user data due to lack of authentication for open TCP sockets. Lastly, we implement end-to-end exploits for all identified attacks and detail two of them for Samsung Smart Switch, a particularly popular data migration tool.

Contributions. The main contributions of our work are

(1) We define attack scenarios for data migration tools and derive analysis procedures for checking susceptibility of a tool based on attack requirements.

(2) Through these analysis procedures, we identify attacks against 7 data migration tools from the most popular Android manufacturers.

(3) We implement end-to-end exploits for all identified attacks and discuss particularly interesting cases in Samsung Smart Switch.

Disclosure. We responsibly disclosed 16 variants of 14 end-to-end exploits to all analysed vendors. They acknowledged 13 of our reports (1 response still pending)[5], and either already rolled out or are rolling out fixes. So far, Samsung, Oppo, Vivo, and Huawei issued us 8 CVEs (5 high, 3 moderate severity)[6].

2 Background and Related Work

This section briefly introduces Wi-Fi and Android background, and discusses related work.

[5] Oppo could not reproduce S4 despite our PoC; Honor did not acknowledge S2.

[6] CVE-2025-{21060, 21061, 21062, 21064, 21078, 27387, 15515, 68963}. CVEs promised by Google, Xiaomi still pending as of the final manuscript of this paper.

Background. *Wi-Fi* refers to protocols defined in IEEE 802.11 for wireless local data transmission over radio [21]. In a typical local network setup, one router functions as the station (AP) while connected client devices operate in STA mode. Wi-Fi operates primarily in $2.4\,\mathrm{GHz}$ and $5\,\mathrm{GHz}$ bands, which exhibit different propagation characteristics. In an outdoor setting, $2.4\,\mathrm{GHz}$ transmissions can achieve ranges of more than 100 meters [47]. In practice, the expected Wi-Fi range in indoor settings is commonly estimated between 20 and 45 meters for the $2.4\,\mathrm{GHz}$ band and about $75\,\%$ of that for the $5\,\mathrm{GHz}$ band [28, 39]. WPA/WPA2/WPA3 are successors to the Wired Equivalent Privacy (WEP) incorporated in the original 802.11 standard, which is considered broken nowadays [17]. Wi-Fi Protected Access (WPA) addresses WEP's flaws through improved key management and authentication mechanisms. The succeeding WPA2 protocol uses the AES block cipher and a four-way handshake to establish secure communications. However, WPA2 does not offer forward secrecy, such that a compromised pre-shared key enables decryption of previously captured communication. The most recent WPA3 finally added forward secrecy. Wi-Fi Direct is an 802.11 standard extension for peer-to-peer communication between Wi-Fi devices, without requiring the traditional AP infrastructure. This protocol establishes ad-hoc networks through group owner negotiation, where one device assumes coordination as the group owner, while connected peers operate as clients. Wi-Fi Direct implementations inherit the security protocols of the underlying Wi-Fi standard, typically WPA2 or WPA3.

Android is an open-source mobile OS developed by Google that dominates the smartphone industry [36]. To ensure a coherent ecosystem despite vendor customizations, devices have to undergo Google's Compatibility Test Suite [1]. Devices that pass are eligible for a license to ship with Google Mobile Services, which includes ecosystem components such as the Google Play Store. The Android Platform Security Model [26] mandates that apps on Android are sandboxed. Their access to system resources such as the file system is tightly restricted. The restrictions can only be selectively relaxed by an app requesting permissions. Particularly powerful permissions can only be granted to privileged apps that are signed with the same certificate as the system. Every app is assigned a private data folder that no other app can access. Apps typically use this folder to store information such as login tokens or other app-specific user data. Usually, not even privileged apps can directly access data in other apps' private data folder. However, privileged apps with the `BACKUP` permission can request a dump of the contained data. Unless an app opts out, this request is granted. For the remainder of this paper, we assume Android's security model remains uncompromised by other privilege escalation attacks that have been presented before [16, 31, 11, 15]. *Wi-Fi APIs* allow Android apps to host or join Wi-Fi networks. On modern Android versions, unprivileged apps can start a Wi-Fi hotspot through the modern `LocalOnlyHotspot` API [4] or through the legacy Wi-Fi Direct Hotspot API [3]. Both establish a local Wi-Fi hotspot with a configurable network name (SSID) and PSK. The latter allows Wi-Fi Direct peers to join, in addition to Wi-Fi devices in STA mode that are also supported by the former. Privileged apps can also utilize the Wi-Fi Tethering API [6] originally intended for sharing the device's Internet connection via Wi-Fi. For joining a Wi-Fi network, unprivileged apps can choose between

the `WifiNetworkSpecifier` [7] and the `WifiNetworkSuggestion` [8] APIs. While the former only establishes a temporary app-local link without Internet access, the latter works on a device-global scale, supporting auto-reconnect and Internet access. Both require user consent through a system dialog. Privileged apps can still use the otherwise deprecated `WifiManager.addNetwork()` [2] API. It does not require user consent but otherwise operates like the `WifiNetworkSuggestion` API. Finally, apps can use Wi-Fi Direct APIs [5] for negotiating a Wi-Fi connection between two devices without needing the APIs described above.

Related Work. Prior works have explored the security and privacy implications of wireless communication and external resource interactions in mobile systems. Naveed et al. [30] highlighted the mis-binding threats from external devices in Android, showing how malicious peripherals can be misrepresented as trusted components. Building on this, Demetriou et al. [14] further dissected both mandatory and discretionary protections for external resources, demonstrating vulnerabilities in Android's security framework when managing peripheral interactions. A broader survey by Wang and Yan [44] systematically reviewed the landscape of device-to-device communication security, identifying numerous threat vectors and proposing general countermeasures. More specialized work by Li et al. [22] exploited vulnerabilities in smart provisioning protocols, specifically SmartCfg, to capture Wi-Fi credentials from air-based transmissions, revealing how initial network setups can leak sensitive information. Cristalli et al. [13] focused on the absence of authentication protocols in app-to-app communications over short-range channels, emphasizing the risks in cross-device scenarios where trust is implicitly assumed. Similarly, Liu et al. [23] analyzed security threats in device-to-device apps, discussing attack surfaces related to permission abuse and communication mismanagement in proximity-based interactions. Recent work has shifted attention to data cloning and device impersonation. Ma et al. [25] and Song et al. [35] exposed vulnerabilities in hotspot-based cloning services and Android system customization, respectively, showing how attackers could replicate or siphon user data via network configurations or system-level modifications. Extending beyond Android, Stute et al. [38, 37] examined Apple's wireless ecosystem, uncovering tracking, denial-of-service, and machine-in-the-middle vulnerabilities through BLE, AWDL, and Wi-Fi. While these studies underline a persistent pattern of insufficient authentication, weak isolation, and exploitable trust assumptions in wireless and cross-device communication, none systematically evaluate the security of current-generation data migration tools, as we do in this study.

3 Threat Models

In line with prior work [25], we assume an adversary who attacks a data migration between two devices. We observe that the new device typically starts out empty and the old device is abandoned after the migration. We therefore consider two distinct attacker goals: (1) exfiltrating data from the old device and (2) injecting (manipulated) data into the new device. Finally, we rule out implementation bugs in the OS or in the underlying protocols, such as WPA2.

We distinguish between two different attacker models based on the privileges prior to the attack. We explicitly consider hybrids between these two models (i.e., a proximity attacker who also has local access) unrealistic.

M1: Proximity Attacker. In this model, the attacker starts with no access to the target device. The only assumption is that the attacker is physically positioned in proximity to the data migration. We observe there are certain physical places that have a high occurrence of data migrations per day. For example, these include retail locations of phone carriers [43, 42, 34], electronics stores [10] or mobile phone repair shops [12, 40]. In these locations, people not only set up their newly purchased devices, but can also get this done as a service. A high volume of data migrations is also likely to be found close to a company's IT department, if the employees' mobile devices are centrally managed and maintained. We assume the distance between this attacker and the data migration does not exceed the range of Wi-Fi radio signals (cf. Section 2). The attacker can hide a small self-contained computer, such as a Raspberry Pi, anywhere within this range. This computer scans the Wi-Fi beacons of nearby Wi-Fi networks until one of them reveals a data migration app (e.g., through its SSID pattern). Given the low cost of the involved equipment (less than $100), this attack can easily be scaled. Through planting a large number of such units, an adversary can collect data from or infect many devices in a short time span. For attacks that require near-realtime PSK brute-forcing, each unit can, e.g., access cheap remote GPUs from Amazon Web Services via an Internet connection.

M2: Local Attacker. Here, the attacker already has code execution capabilities on the old device. For example, an otherwise benign app might integrate a malicious library. Due to the Android Platform Security Model [26], the app is still sandboxed, i.e., can only access limited system resources. Attacking a data migration might thus allow elevating its privileges, i.e., bypassing sandbox restrictions. This can, e.g., include gaining code execution in another app on the new device, or accessing resources not otherwise reachable for the malicious app.

Attack Impact. Through any of the described attacks, the attacker gains access to sensitive user information. All examined data migration tools transmit messages, call logs, contacts, pictures, videos, and documents. For first-party migrations (between two devices from the same vendor), transmissions often also include Wi-Fi credentials and app-private data, which commonly contains session tokens or login credentials (see Section 5.3). Any of this data can be misused to cause serious harm to the device owner, their social circle, or employer. Some of the attacks additionally allow manipulation of the transmitted data. Manipulating transmitted app packages, i.e., a variant of an app repackaging attack [27], allows gaining code execution in the context of third-party apps. As a result, the attacker can, e.g., access app-private data even if it was not included in the migration itself. Additional privilege escalations can be accomplished, e.g., by overwriting system configurations such as granted permissions for an app.

4 Attack Scenarios for Data Migration Tools

This section defines attack scenarios for data migration tools as cross-products of attacker goals and attack surfaces. Per scenario, we identify attack requirements that need to be fulfilled for the scenario to be realizable for a specific app.

General Operation of Data Migration Tools. An initial analysis indicates that all data migration tools operate in 4 basic steps. ① They start with a pairing phase, in which a direct link between the old and new device is established. In the examined apps, this link is either a Wi-Fi connection (one device acts as Wi-Fi AP), Wi-Fi Direct, or USB connection. ② Next, the app on the old device prepares the data to be transmitted. Involved steps may include data export from databases, compression, and/or copying to a central place. ③ The prepared data is sent to the new device over the previously established direct link. For links based on Wi-Fi, this involves communication over TCP. ④ Finally, the received data is reinstated on the new device, roughly inversing ②.

Attack Surfaces. The different attacker models (cf. Section 3) reach different attack surfaces of the operation described above. An **M1** (proximity) attacker can interact with steps ① and ③, i.e., the pairing and transmission procedures. The main attack surface reachable to an **M1** attacker in these operations is the link layer. It is worth noting that an **M1** attacker is only effective against the most common scenario of wireless transmissions (cf. Section 7). For **M2** (local) attackers (which we restrict to the old device as explained in Section 3), only steps ② and ③ are accessible. The main attack surfaces involved here are the file system or open TCP/IP sockets.

4.1 Definition of Attack Scenarios

Based on the attack surfaces discussed above, we define attack scenarios for data migration tools. To this end, we consider cross-products of main attack surfaces (proximity Wi-Fi, local file system and TCP/IP sockets) and attacker goals (data exfiltration and injection). The attack scenarios incorporate known attack techniques against Wi-Fi [41], TCP/IP [24], general device-to-device connectivity [13] and data transfer tools [25]. For every scenario, we list requirements, i.e., design or implementation details that need to be present in a data migration tool for the scenario to be realizable. We emphasize on realistic scenarios based on slightly refined attacker models from prior works (see Section 3) rather than on exhaustiveness. In particular, we only consider attacks that do not visibly interfere with the data migration, i.e., do not cause alertness of the user due to errors or warnings visible in the UI. Table 3 shows an overview of the requirements per scenario. In the following, we discuss the scenarios in detail.

S1: Eavesdropping on wireless communication. The **M1** attacker captures Wi-Fi frames and later (offline) extracts the data exchanged in the transmission. Many Wi-Fi chips support monitor mode, which allows passively (therefore stealthily) capturing Wi-Fi frames destined for other devices. For this scenario to be viable, the Wi-Fi beacons of the data migration app must allow identification of the specific tool, e.g., through the SSID of the hotspot it creates. For the attack to yield user

Requirement	S1	S2	S3	S4	S5	S6
R1: App is identifiable through Wi-Fi beacons	✓	✓	-	-	-	-
R2: Wireless protection weak	✓	-	-	-	-	-
R3: No app-level payload encryption	✓	-	✓	✓	-	-
R4: Wireless protection very weak (Wi-Fi AP)	-	✓[1]	-	-	-	-
R5: Pairing identification ambiguous (Wi-Fi Direct)	-	✓[1]	-	-	-	-
R6: Payload encryption not derived from out-of-band secret	-	✓	-	-	-	-
R7: Payload integrity not protected via out-of-band secret	-	✓	-	-	-	-
R8: Server socket on old device accepts extra clients	-	-	✓	-	-	-
R9: No authentication for payload connections	-	-	✓	✓	-	-
R10: Wi-Fi join is device-global	-	-	-	✓	-	-
R11: Server socket on new device accepts extra clients	-	-	-	✓	-	-
R12: No encrypted or separately transmitted checksums	-	-	-	✓	-	-
R13: User data (temporarily) stored in public file system	-	-	-	-	✓	✓
R14: No encryption for data stored in public file system	-	-	-	-	✓	✓
R15: No integrity checks for data in public file system	-	-	-	-	-	✓

[1] Either **R4** or **R5** is necessary for **S2**.

Table 3: Summary of requirements for different attack scenarios **S1** to **S6**.

data, the attacker needs to be able to decrypt the captured Wi-Fi frames. This means the app's Wi-Fi hotspot needs to be either WPA2-protected with a weak PSK or unprotected. We consider a WPA2 PSK weak if it can be brute-forced from a captured WPA2 handshake in a matter of hours. For reference, an Nvidia GeForce RTX5090 commodity GPU can brute-force WPA2 at a rate of $3.4\,\mathrm{MH/s}$. Finally, this scenario is only realizable if there is no effective app-level encryption for payloads, i.e., encryption based on an asymmetric key exchange or an out-of-band secret prevents this attack.

S2: Manipulating wireless communication. The **M1** attacker gains a mediating position between the old and new device to alter the payloads exchanged between them. For normal AP Wi-Fi connectivity, it can be accomplished, e.g., through a multi-channel machine-in-the-middle (MC-MITM) attack (cf. Appendix A), ARP spoofing, or DHCP spoofing. The attacker needs to be able to decrypt and re-encrypt the transmission in real-time, both at the link and app layers. This means the app needs to use WPA2 or no protection for its hotspot. The WPA2 key must leak to the attacker or be very weak, i.e., brute-forceable in near-realtime. Specifically, we consider a WPA2 PSK very weak if it can be realistically brute-forced within the 15 seconds that Android waits for a DHCP lease. During this time, a MC-MITM can artificially delay connection establishment by intentionally not forwarding Wi-Fi frames. A detailed description of this technique can be found in Appendix A. If Wi-Fi Direct is used, an attacker can achieve a MITM position if the target device is not unambiguously identified prior to forming a group. Finally, for this attack scenario to be realizable, any app-level payload encryption or integrity checking must not be cryptographically linked to an out-of-band (OOB) secret. If an OOB secret is only used for authentication, an attacker can still perform a relay attack without knowing the secret.

S3: Extracting data via local socket connection. A malicious app (**M2** attacker) running on the old device might be able to connect to a listening socket on

the same device opened by the data migration tool. If needed, SYN scanning [24] allows identifying dynamic ports. For a connection attempt to succeed, the listening socket must accept additional payload connections. The attacker has no information about the transmission session, so an attack can only work if payload connections are unauthenticated and unencrypted.

S4: Injecting data via socket connection to new device. If the data migration tool opens a listening socket on the new device, a malicious app (**M2** attacker) on the old device might use it to inject data. For identifying the new device's IP address and port, the attacker can, e.g., use ARP scanning and SYN scanning. As for **S3**, the tool must allow extra payload connections that are unauthenticated and unencrypted. Finally, it needs to use a joining API that establishes a device-global connection (cf. Section 2).

S5: Extracting data through file system. If the data migration tool stores migration data in public storage, an **M2** attacker can extract it from there. For tools privileged on the old device (first-party data migrations), these files may leak data otherwise inaccessible to user-installed apps. This scenario is only viable if the data migration tool does not (effectively) encrypt its files.

S6: Manipulating data through file system. In addition to **S5**, a malicious app can manipulate migration data if there are no integrity checks for the (unencrypted) files the data migration tool stores in public storage.

5 Systematic Analysis

In this section, we analyze the seven data migration apps described in Section 1 for their susceptibility to the six attack scenarios introduced in Section 4. We use dynamic testing and manual static analysis to systematically determine susceptibility through the presence of requirements **R1** to **R15**.

5.1 Evaluation Setup

Per app, we assess first-party data migration between two devices from the same manufacturer and third-party data migration between differing manufacturers. The exception is Google, where any Google-certified Android device (due to the app being preinstalled) is considered first-party (we use the Samsung Galaxy S23), and third-party migrations are not supported. We use Honor as source for first-party migrations on Huawei and vice versa for lack of a second device from these vendors. This is possible because the two apps use the same protocol, and reverse-engineering reveals strong similarities between their code bases. We thus anticipate the core findings for these apps align with first-party migrations.

Table 2 lists the Android devices we use for our analysis. For every data migration app manufacturer, we use a first-party device purchased in either 2024 or 2025 as the data migration target. We can therefore assume that our results accurately reflect real-world migration to a freshly purchased device. For analysis steps that require root access (see Section 5.2), we root the Google Pixel 8a using Magisk [45] and use it as the old device in a third-party migration.

If an analysis requires capturing Wi-Fi communication, we utilize a Raspberry Pi 5 running Kali Linux and an external Wi-Fi interface based on the Mediatek MT7921 chipset. For brute-forcing WPA2 PSKs, we use hashcat on a Ubuntu 22.04.4 LTS Linux host with an Nvidia GeForce RTX5090 commodity GPU. We use JADX [33] and Ghidra [29] for reverse-engineering managed and native components of app package (APK) files.

5.2 Analysis Methodology

We determine the susceptibility to attack scenarios **S1** through **S6** from Section 4 through the presence of respective requirements **R1** to **R15** in an app. Additionally, we evaluate the data each data migration tool can migrate. For efficiency reasons, we employ short-circuit evaluation. If one requirement for a given attack is not present in an app, we do not evaluate for the presence of other requirements for that attack. Where possible, we evaluate the presence of a requirement through dynamic testing procedures whose outcomes provide definitive proof for the presence or absence of the respective implementation detail. However, for some requirements, such a procedure cannot be found or is not feasible. In these cases, we employ reverse-engineering of the app code to arrive at a conclusion. Unless noted otherwise, we carry out every check for every tool for both first-party and third-party data migrations. As some vendors prevent rooting, we aim for analysis procedures that work without root access where possible. In the following, we describe the analysis procedure for each attack requirement, as well as for finding the per-app migration data scope.

Data included in migrations. For determining the scope of data migrations, we prepare the old device for each app with pictures, videos, PDF documents, phone calls, SMS messages, contacts, and Wi-Fi credentials. Additionally, we install a simple app that stores fake secrets in its private data folder. We then check for the presence of these data items in transmission captures (described for **R3** below). For apps where all traffic is encrypted, we confirm data scope by checking data items present on the new device after the data migration.

Wi-Fi beacons identify data migration app (R1). We complete multiple pairing phases per app, inspecting Wi-Fi beacons from captures and noting down the SSIDs of started hotspots. If we recognize identical sections in the SSIDs (beyond recognizable device model numbers), we use these strings for a search into the app code. This allows us to confirm if these strings are constants in the code, which allows identification of the app.

Weak (R2) or very weak (R4) wireless protection. We determine the used Wi-Fi protection scheme through a Wi-Fi scanner immediately after the pairing step of the data migration tool. Some apps perform connection upgrades to $5\,\mathrm{GHz}$ Wi-Fi channels, and change the Wi-Fi protection scheme in the process. We thus additionally scan during the data transmission. If we detect WPA2, we reverse-engineer the app to determine the method used for generating the PSK. For PSKs that are randomly generated, we determine the PSK length and used character set. This allows us to calculate the worst-case time it takes to brute-force the PSK on commodity hashcat setups or rentable cloud compute.

App-level payload encryption (R3). We capture the Wi-Fi traffic of a transmission covering all supported data types and (if necessary) decrypt the captured Wi-Fi frames. As needed, we either extract the PSK from the UI, from verbose Wi-Fi logs on unrooted devices, derive it from the SSID, or brute-force it. From the decrypted Wi-Fi communication, we re-assemble TCP streams, which we then manually inspect. We try to identify all transmitted data types through their content or unique format. If we cannot find some data in the capture, we reverse-engineer the app to see if the reason was encryption.

Pairing device identification (R5). For normal Wi-Fi connections between a STA and an AP, pairing is based on the knowledge of both the SSID and PSK. In these cases, the attacker can only interfere with the pairing if they know the PSK (cf. **R4**). We determine an app's use of Wi-Fi Direct through scanning for corresponding advertisement beacons. If we find such beacons, we reverse-engineer the app to identify the pairing information used for identifying the device with which to start Wi-Fi Direct group negotiation.

Cryptography based on OOB secrets (R6, R7). First, we check for OOB secrets in the implementation. We complete a data migration, taking note of all communication channels between the two devices beyond the primary link (usually Wi-Fi). If the pairing phase involves scanning a QR code, we manually extract and analyse its contents. Next, we determine the presence of payload encryption (**R6**) or integrity checks (**R7**). For encryption, we use the procedure described for **R3**, additionally checking for involvement of the OOB secret. Equally, we extend the integrity checks procedure described for **R12**, reverse-engineering involved logic and correlating it with communication captures.

Sockets accepting additional connections (R8, R11). We monitor the procfs TCP tables via the Android Debug Bridge (ADB) to identify sockets used for the data migration. ADB can be enabled on any Android device (rooted or not). For every listening socket on either the new (**R8**) or old (**R11**) device, we observe whether it accepts multiple clients during a legitimate data migration. If this is the case, we use connection captures to determine whether these clients carry data payloads. Finally, we manually reverse-engineer the involved logic to determine whether arbitrary payload clients can connect.

Authentication for additional connections (R9). We reverse-engineer the protocol used for payload connections. We then send packets for sending or requesting a single file from an additional socket client during a legitimate data migration. If the file extraction or injection succeeds, no authentication is used.

App-level payload integrity checks (R12). We use Frida [32] to monitor `MessageDigest` and `Hmac` APIs, checking if they process transmitted data. We also hook framework APIs for retrieving APK signatures. If any hook fires, we identify and reverse-engineer the relevant logic in the app's code. Due to needing root access, we only run this check for third-party migrations.

Wi-Fi joins device-global (R10). During a data migration, we check if a ping to the new device succeeds from a separate process on the old device.

Data in public file system (R13, R14, R15). We use the `FileObserver` API from a service on the old device to monitor all files written to during a data migration.

We manually inspect these files to see if they contain user data (**R13**) and are unencrypted (**R14**). For files whose contents cannot be classified, we use reverse-engineering to determine if and what encryption is used, and trace back the origin of the key. We test for integrity checks (**R15**) by replacing files during a data migration and confirming that they are restored on the new device.

5.3 Analysis Results

Attack Impact. The data types each migration tool transmits can be found in Table 5. Beyond basic media and communication data, all except Google also support migrating documents and apps (APKs). It is worth noting that for Google in particular, the transmission of certain data types is not apparent in the UI and cannot be disabled. Specifically, all Google data migrations include highly sensitive health data (including sexual activity and other reproductive health data). Google also migrates app-private data for installed apps, even if the app (APK) the data belongs to is not migrated. First-party data migrations commonly include additional information related to manufacturer-specific services that we do not examine at scale. Unless otherwise noted, the attacks described in the following concern all data included in the data migration.

S1: Eavesdropping on Wireless Communication. Every analyzed data migration app supports wireless migration. Samsung uses Wi-Fi Direct with strong WPA2 keys, preventing passive eavesdropping (i.e., **S1**). The other tools establish a Wi-Fi AP with a unique SSID pattern (**R1**) on the new device, protected only with a weak (**R2**) or very weak WPA2 PSK. Vivo starts out with a WPA2-protected $2.4\,\mathrm{GHz}$ Wi-Fi hotspot, but switches to a completely unprotected $5\,\mathrm{GHz}$ hotspot for large transfers. Google and Oppo use 8-character PSKs that allow brute-forcing within 30 minutes on a single RTX5090 GPU. Xiaomi deterministically derives its PSK from a random token embedded in the SSID, which an **M1** attacker can simply replicate. Exact Wi-Fi configurations for these apps are listed in Appendix B. Only two apps use some payload encryption (**R3**). Vivo employs TLS for all payloads. Oppo only encrypts contacts, SMS/MMS, call logs, and Wi-Fi credentials using an RSA key negotiated during handshake.

S2: Manipulating Wireless Communication. Huawei's, Honor's, and Vivo's very weak all-numeric PSKs can be realistically brute-forced within 15 seconds (**R4**). An attacker can, e.g., use 2 RTX5090 GPUs or rent a suitable AWS E2C instance for \$13/hr[7]. Samsung can be identified through vendor fields in Wi-Fi Direct beacons (**R1**), and target device identification is ambiguous (**R5**; see Section 6). Huawei and Honor do not use OOB secrets (**R6**), thus being unprotected against MITM attacks. Their pairing QR codes only contain the Wi-Fi SSID and PSK. Vivo's TLS implementation trusts any server certificate. Samsung encrypts certain payloads during transmission, but transmits the symmetric key in plain. Samsung and Vivo do not employ any integrity checks (**R7**). Xiaomi uses MD5 for some payloads, but not protected via OOB secrets.

[7] AWS EC2 g6e.12xlarge with 4 NVIDIA L40S GPUs ($9.4\,\mathrm{MH/s}$ total).

S3: Extracting data via local socket connection. Only Samsung and Vivo open listening sockets on the old device. Samsung only accepts the first connection, which is used by the legitimate client. Vivo allows arbitrary payload connections (**R8**), and its TLS server uses no client authentication (**R3**/**R9**).

S4: Injecting data via socket connection to new device. For third-party data migrations, all tools use app-local Wi-Fi connections. Oppo, Vivo, and Xiaomi additionally support manual pairing through the system's Wi-Fi settings, leading to device-global connections. For first-party migrations, all tools except Google use the legacy join API, resulting in device-global connections (**R10**). All these apps open a listening socket on the new device. Samsung only accepts a single connection. Huawei and Honor use limited parallel payload connections, but their legitimate clients exhaust the allowed number of connections. These apps are unsusceptible to **S4**, as consuming such a connection for the attack would visibly disturb the transmission, violating the goals from Section 4. Xiaomi accepts an arbitrary number of payload connections, but requires each client to prove knowledge of the old device's random UUID, which the attacker cannot know. Oppo allows arbitrary payload connections (**R11**) without effective authentication (**R9**). Although Vivo also accepts arbitrary connections on the new device, these are not for data payloads. Oppo does not implement payload encryption (**R3**) for APK files, which means they can be manipulated for code execution. Although Oppo transmits SHA256 hashes for APK files, they are never enforced on the receiving side.

S5: Extracting data through file system. Huawei, Honor, and Samsung write files to public storage during a data migration. For Huawei and Honor, these files only store insensitive analytics data. Samsung uses public storage for caching user data on the old device prior to transmission **(R13)**. Specifically, this concerns all transmitted data except pictures, videos, and documents, for both wired and wireless data migrations. For first-party migrations, the stored user data includes, e.g., Wi-Fi credentials, which would otherwise never be accessible to third-party apps. Although some files are encrypted, the used symmetric encryption key is written to public storage after each data migration **(R14)**.

S6: Injecting data through file system. Only Samsung stores user data in public storage (**R13**). Some data, such as Wi-Fi credentials, is unencrypted. Other data, including APK files, is encrypted. Although the app writes the symmetric encryption key into public storage, it only does so at the end of a data migration session. The key is randomly generated per session. However, the APK encryption scheme only encrypts the first 1 MB of every APK file (**R14**). As shown in Section 6, an attacker can exploit properties of the APK file format to construct a modified APK file whose encrypted prefix stays unchanged. No integrity checks are included (**R15**).

5.4 Summary of Results

As the overview in Table 4 shows, all tools are susceptible to at least one of our attack scenarios. Notably, all apps are vulnerable to realizations of at least one scenario under attacker model **M1** (wireless), which is the attacker model that requires no prior privileges. The most realizable attack scenarios across vendors are

S1 and **S2**, i.e., wireless eavesdropping and MITM. Both work against 5 of the 7 ($71\,\%$) examined apps. Overall, the examined apps are much less susceptible to **M2** (local) attack scenarios, which in total affect 3 of the 7 ($43\,\%$) apps.

The most vulnerable data migration tools in our test set are Samsung and Vivo, each susceptible to 3 attacks. This is concerning, as besides Google, these have the highest download count in our set (both > 1 billion). The most secure of the analyzed apps is Google, which is only affected by one attack.

	Attack Scenario					
	Proximal		Local			
App	S1	S2	S3	S4	S5	S6
Google	⚠	-	-	-	-	-
Samsung	-	⚠	-	-	⚠	⚠
Vivo	-	⚠	⚠	-	-	-
Xiaomi	⚠	⚠	-	-	-	-
Oppo	⚠[1]	-	-	⚠[2]	-	-
Huawei	⚠	⚠	-	-	-	-
Honor	⚠	⚠	-	-	-	-

[1] Except Wi-Fi Configs, SMS/MMS, Call Log, Contacts.
[2] Only first-party migrations or manual pairing.

Table 4: Attack susceptibility per data migration tool.

App	🖼	📹	📄	📞	💬	🪪	</>	🗄	📶	🧰
Google	✓	✓	-	✓	✓	✓	-	✓	✓	✓
Samsung	✓	✓	✓	✓	✓	✓	✓	✓[1]	✓[1]	-
Vivo	✓	✓	✓	✓	✓	✓	✓	-	✓[1]	-
Xiaomi	✓	✓	✓	✓	✓	✓	✓	-	✓[1]	-
Oppo	✓	✓	✓	✓	✓	✓	✓	✓[1]	✓[1]	-
Huawei	✓	✓	✓	✓	✓	✓	✓	-	-	-
Honor	✓	✓	✓	✓	✓	✓	✓	-	✓[1]	-

[1] Only first-party data migrations.

Table 5: Content included in data migrations per app. 🖼 = Pictures, 📹 = Videos, 📄 = Documents, 📞 = Call Logs, 💬 = SMS/MMS, 🪪 = Contacts, </> = Apps (APK files), 🗄 = App Data, 📶 = Wi-Fi Credentials, 🧰 = Health Data.

6 End-to-End Attacks

We implemented end-to-end attacks for all possible attack scenarios identified through our analyses (see Table 4). The implementations for **S1** attacks are very similar across tools. We capture Wi-Fi frames, derive or brute-force the weak WPA2 PSK to decrypt the frames, and extract payloads using binwalk. For **S2** attacks, we use app-specific techniques (described below) against Samsung, DHCP spoofing against Vivo, and MC-MITM (cf. Appendix A) against Xiaomi, Huawei, and Honor. Due to space constraints, we focus this section on **S2**, **S5** and **S6** against Samsung, a particularly popular yet vulnerable app.

Manipulating Wireless Communication (S2). Samsung Smart Switch uses Wi-Fi Direct as the wireless link. The app on the new device starts advertising as a Wi-Fi Direct peer and displays a QR code for scanning through the app on the old device. The code contains a 32-byte random shared secret S, as well as $H_N = HMAC_S(\textit{Suffix}(MAC_{New}, 3))$, an HMAC of the last three bytes of the new device's MAC address. After scanning the QR code, the old device starts Wi-Fi Direct peer discovery, calculating $H_C = HMAC_S(\textit{Suffix}(MAC_{Candidate}, 3))$ for each discovered MAC address. If $H_C == H_N$, the old device negotiates a Wi-Fi Direct group with this peer. Once connected, both devices prove possession of S

through an authentication handshake that includes a random nonce N and HMAC $H_A = HMAC_S(N)$ from both parties. Additionally, the new device uses S to encrypt the subsequent message that contains its IP address and used port, which the old device then tries to connect a socket to.

This pairing scheme is vulnerable to a MITM attack. The attacker scans for Wi-Fi Direct peers until a device is discovered whose Wi-Fi beacons identify it as a Samsung smartphone. At this point, the attacker spoofs the last three bytes of the MAC address in the Wi-Fi beacons and starts to advertise as a Wi-Fi Direct peer as well. Although the attacker does not know S, their MAC address will now qualify them for group negotiation with the old device. Depending on which peer the old device discovers first, it will either negotiate a group with the new device or the attacker. The attacker can detect group formation with the new device by a new group SSID being broadcast. In this case, the attacker forces the user to repeat the pairing procedure by sending Wi-Fi de-auth frames. Once negotiation with the old device has succeeded, the attacker requests group negotiation with the new device on a second Wi-Fi interface. Through its gained MITM position, the attacker then relays the authentication handshake between the new and old device. Although the attacker does not know S, they are now authenticated to both legitimate devices. The attacker replicates the IP allocation scheme of Android's Wi-Fi Direct implementation, which always assigns the same address to the group owner. Next, the attacker opens a server socket on the hardcoded port Samsung uses. As a result, the attacker can simply relay the subsequent encrypted message containing the IP address and port unaltered. There is no cryptographic link between the OOB secret S and actual payload transmission. Some data payloads in the transmission are encrypted, but using a symmetric key that is transmitted in plain before the payload. We successfully used this attack to extract pictures, videos, documents, call logs, messages, contacts, and Wi-Fi credentials from a data migration. Additionally, we demonstrated that the attacker can get code execution in any migrated app by replacing its APK data as it is transmitted.

Reading (S5) & Manipulating (S6) Cache Files. Smart Switch caches most user data in public storage during a transmission. Our **S5** attack can extract unencrypted Wi-Fi credentials, or use the key Samsung writes to disk after each transmission to, e.g., decrypt encrypted Samsung account credentials. Although cached APK files are encrypted, the encryption scheme only covers the first 2^{20} bytes ($1\,\mathrm{MiB}$) of data. Our **S6** attack constructs a special APK file that prepends the first 2^{20} bytes of the unencrypted target APK to the attacker's payload APK. The attacker can simply determine the package name of the target APK from its file name and use unprivileged framework APIs to retrieve its unencrypted APK contents. Next, the attacker adjusts the offsets in the ZIP Central Directory to account for the added bytes and signs the resulting APK. Finally, the attacker replaces the prefix with the corresponding ciphertext from the encrypted APK.

7 Discussion

This section discusses potential limitations of our analyses and solutions for mitigating attacks in device-to-device communication.

Analysis Precision. Some of our procedures for determining presence of individual attack requirements use manual reverse-engineering. Like with any static analysis technique, imprecisions cannot be ruled out. However, we implemented working end-to-end exploits for all identified attacks, which we included in vendor disclosures. We thus argue that the core findings of our analysis faithfully reflect the security posture of state-of-the-art Android data migration tools.

Wired Data Migration. Some data migration tools support wired operation via USB connections, which are insusceptible to our most powerful (**M1**) attacks. However, we anticipate that wireless connections are used in most data migrations for two reasons. (1) Only one tested tool (Samsung) supports (optional) wired migrations from Android devices. Although a legacy support document mentions wired migrations for Google [18], the rebranded Android Switch no longer offers this option [19]. (2) In a USB connection between two phones, one of them charges from the other, quickly draining its battery.

Mitigations. MITM or eavesdropping attacks on device-to-device communication can effectively be mitigated by encrypting all exchanged traffic through an OOB secret. An OOB secret of sufficient entropy can, e.g., be passed between devices through a QR code or derived from a short user-entered pin via a password-authenticated key exchange (PAKE) protocol such as Secure Remote Password (SRP) [46]. At the time of our analysis, none of the evaluated apps integrated such a mitigation.

8 Conclusion

In this paper, we presented a systematic security analysis of data migration tools in the Android ecosystem. Based on an abstracted view of the operation of these apps, we defined six attack scenarios. We then identified necessary implementation details for a specific scenario to be realizable for a given app, arriving at 15 attack requirements. These requirements allowed us to systematically evaluate the susceptibility for each app and attack scenario pair. Our evaluation for seven widely used data migration tools uncovered severe vulnerabilities across the industry. All apps were susceptible to wireless attacks from an attacker within Wi-Fi range of the victim. Of the seven apps, three were also susceptible to local attacks, e.g., from malware. We presented end-to-end attacks on all analyzed data migration tools and detailed two of them for Samsung Smart Switch, a particularly popular yet vulnerable data migration tool. We anticipate our systematic cross-vendor study will provide value for practitioners and developers and motivate lasting improvements to this critical part of the Android ecosystem.

Acknowledgements. This project has received funding from the Austrian Research Promotion Agency (FFG) via the SEIZE project (FFG grant number 888087).

A Multi-Channel Machine-In-The-Middle Attack

In this section, we document our adaptations to multi-channel machine-in-the-middle (MC-MITM) attacks as presented by Vanhoef et al. [41]. These attacks allow an attacker within Wi-Fi range to gain a MITM position between a Wi-Fi STA and AP. They work by injecting Wi-Fi control frames for enforcing a frequency switch for STAs connected to the victim AP, but not for the AP itself. The attacker can then relay between the two different frequencies, effectively granting it a MITM position between the STAs and the AP. Vanhoef et al. [41] focused their efforts on link-level attacks operating on the encrypted Wi-Fi frames, enabling, e.g., denial-of-service attacks. However, the approach can be extended to scenarios where the attacker needs to manipulate the actual data payloads within Wi-Fi frames. For this to work in general, the attacker needs to obtain the PTK, i.e., know the WPA2 PSK and observe the handshake. We further adapt the technique so that it offers the attacker a time window for brute-forcing the WPA2 PSK. This can be accomplished by intentionally dropping (i.e., not forwarding) all communication for a certain time span following the WPA2 handshake. During this time, the attacker brute-forces the PSK and PTK from the captured handshake. Once the PSK and PTK have been brute-forced, the attacker continues to relay between STA and AP, but can now decrypt, manipulate, and re-encrypt the transmitted payloads. In our experiments on multiple Android devices, we could consistently drop Wi-Fi communication for at least $15\,\mathrm{s}$ without any negative consequences. When using longer time windows, the OS (and therefore any app using the system's Wi-Fi APIs) considered the Wi-Fi network non-functional and disconnected from it. Accounting for overhead caused by the Python implementation of the attack, this observation matches the $18\,\mathrm{s}$ timeout Android implements for awaiting a DHCP lease[8]. For the attack to work, the attacker therefore needs to be able to brute-force the WPA2 PSK within $15\,\mathrm{s}$. The (simplified) steps the attacker takes are:

1. Scan for Wi-Fi beacons until a data migration is detected through the app's unique SSID pattern.
2. Start a Wi-Fi AP on a rogue channel and send Channel Switch Announcement (CSA) beacons to force STAs onto the rogue channel.
3. Capture Wi-Fi frames on the corresponding channel until a WPA2 handshake is observed.

[8] Default timeout defined at `https://cs.android.com/android/platform/superproject/main/+/main:packages/modules/NetworkStack/common/networkstackclient/src/android/net/shared/ProvisioningConfiguration.java;drc=497ee73d99bf8331974955c844581efd46e83bea;l=85`

4. Once the link-level MITM position is established, stop forwarding frames to the AP for $15\,\mathrm{s}$.
5. During this time, brute-force the WPA2 PSK based on the captured WPA2 handshake and known PSK pattern for the app.
6. Use the brute-forced PSK to decrypt, manipulate, and re-encrypt the Wi-Fi frames relayed between STA and legitimate AP.

We successfully implemented this attack technique in Python using the open-source Scapy library. For being able to carry out complex modifications to TCP streams contained in the Wi-Fi frames, we route the traffic through the kernel TCP/IP stack via TUN interfaces on the attacker machine running Linux.

B Wi-Fi Hotspot Configurations

Table 6 displays the Wi-Fi hotspot configurations used by the analysed data migration apps. Samsung was omitted due to using Wi-Fi Direct.

App	SSID Pattern	WPA2 PSK
Google	`DIRECT-[0-9A-F]{2}-[0-9A-F]{6}`	`[0-9A-F]{8}`
Vivo	`vivo#.#[0-9A-Z]{2}(_RE)?`	None or `[0-9]{8}`
Xiaomi	`._[0-9a-zA-Z]{7}_MI`	Derived from SSID
Oppo	`oplus_co_ap[a-z]{4}`	`[a-z]{4}[0-9]{4}`
Huawei	`.%[0-9]{4}%CloudClone`	`[0-9]{8}`
Honor	`.%[0-9]{4}%CloudClone`	`[0-9]{8}`

Table 6: Wi-Fi Hotspot configurations used by data migration apps. `.` represents manufacturer-specific model identifiers.

References

1. AOSP: Android Compatibility program overview (2025), https://source.android.com/docs/compatibility/overview
2. AOSP: API Reference: WifiManager.addNetwork() (2025), https://web.archive.org/web/20250729133834/https://developer.android.com/reference/android/net/wifi/WifiManager#addNetwork(android.net.wifi.WifiConfiguration)
3. AOSP: API Reference: WifiP2pManager.createGroup() (2025), https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager#createGroup(android.net.wifi.p2p.WifiP2pManager.Channel,%20android.net.wifi.p2p.WifiP2pConfig,%20android.net.wifi.p2p.WifiP2pManager.ActionListener)
4. AOSP: Use a local-only Wi-Fi hotspot (2025), https://developer.android.com/develop/connectivity/wifi/localonlyhotspot
5. AOSP: Wi-Fi Direct (peer-to-peer or P2P) overview (2025), https://web.archive.org/web/20250729122558/https://developer.android.com/develop/connectivity/wifi/wifip2p

6. AOSP: Wi-Fi hotspot (Soft AP) (2025), https://source.android.com/docs/core/connect/wifi-softap
7. AOSP: Wi-Fi Network Request API for peer-to-peer connectivity (2025), https://web.archive.org/web/20250729122716/https://developer.android.com/develop/connectivity/wifi/wifi-bootstrap
8. AOSP: Wi-Fi suggestion API for internet connectivity (2025), https://developer.android.com/develop/connectivity/wifi/wifi-suggest
9. AppBrain: Android Statistics: Top Manufacturers (2025), https://web.archive.org/web/20250702234143/https://www.appbrain.com/stats/top-manufacturers
10. BestBuy: GeekSquad: Setup for New Devices (2025), https://www.bestbuy.com/site/setup-for-new-devices/9131012.p?skuId=9131012&intl=nosplash
11. Cao, S., Zhou, H., Shi, S., Zhao, Y., Wang, H.: Parcel mismatch demystified: Addressing a decade-old security challenge in android. In: CCS (2025)
12. Cell Phone Repair by Assurant: Cell Phone Data Transfer (2025), https://www.cellphonerepair.com/common-issues/data-transfer
13. Cristalli, S., Lu, L., Bruschi, D., Lanzi, A.: Detecting (absent) app-to-app authentication on cross-device short-distance channels. In: ACSAC (2019)
14. Demetriou, S., Zhou, X., Naveed, M., Lee, Y., Yuan, K., Wang, X., Gunter, C.: What's in Your Dongle and Bank Account? Mandatory and Discretionary Protection of Android External Resources. In: NDSS (2015)
15. Draschbacher, F., Maar, L.: Manifest problems: Analyzing code transparency for android application bundles. In: ACSAC (2024)
16. 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: USENIX Security (2020)
17. Fluhrer, S., Mantin, I., Shamir, A.: Weaknesses in the key scheduling algorithm of RC4. In: International Workshop on Selected Areas in Cryptography (2001)
18. Google: Copy apps & data from an Android to a new Android device (2023), https://support.google.com/android/answer/13761358?hl=en, accessed: 10/08/2025
19. Google: How to transfer data between Android phones (2025), https://web.archive.org/web/20251008132752/https://www.android.com/transfer-data-android-to-android/, accessed: 10/08/2025
20. Google, I.: Google Play: Android Switch (2024), http://web.archive.org/web/20250805184017/https://play.google.com/store/apps/details?id=com.google.android.apps.restore, accessed: 08/05/2025
21. Institute of Electrical and Electronics Engineers: Wireless Local Area Networks (2025), https://www.ieee802.org/11/
22. Li, C., Cai, Q., Li, J., Liu, H., Zhang, Y., Gu, D., Yu, Y.: Passwords in the Air: Harvesting Wi-Fi Credentials from SmartCfg Provisioning. In: WiSec (2018)
23. Liu, K., Shen, W., Cheng, Y., Cai, L.X., Li, Q., Zhou, S., Niu, Z.: Security Analysis of Mobile Device-to-Device Network Applications. IEEE Internet of Things Journal (2019)
24. Lyon, G.: The Art of Port Scanning. Phrack Magazine (1997)
25. Ma, S., Li, H., Yang, W., Li, J., Nepal, S., Bertino, E.: Certified Copy? Understanding Security Risks of Wi-Fi Hotspot based Android Data Clone Services. In: ACSAC (2020)
26. Mayrhofer, R., Stoep, J.V., Brubaker, C., Kralevich, N.: The Android Platform Security Model. CoRR (2023)
27. Merlo, A., Ruggia, A., Sciolla, L., Verderame, L.: You Shall not Repackage! Demystifying Anti-Repackaging on Android. Computers & Security (2021)

28. Mitchell, B.: What Is the Range of a Typical Wi-Fi Network? (2020), https://www.lifewire.com/range-of-typical-wifi-network-816564, accessed: 08/05/2025
29. National Security Agency: Ghidra SRE framework (2019), https://github.com/NationalSecurityAgency/ghidra, accessed: 07/31/2025
30. Naveed, M., Zhou, X.y., Demetriou, S., Wang, X., Gunter, C.A.: Inside Job: Understanding and Mitigating the Threat of External Device Mis-Binding on Android. In: NDSS (2014)
31. Oversecured: Oversecured automatically discovers persistent code execution in the Google Play Core Library (2020), https://blog.oversecured.com/Oversecured-automatically-discovers-persistent-code-execution-in-the-Google-Play-Core-Library/, accessed: 03/11/2026
32. Ravnås, O.A.V.: Frida (2014), https://frida.re, accessed: 07/31/2025
33. Skylot: JADX (2015), https://github.com/skylot/jadx, accessed: 07/31/2025
34. SmarTone: Data Transfer Service (2025), https://www.smartone.com/en/privileges_and_support/support/device_support/data_transfer.jsp
35. Song, W., Jiang, M., Yan, H., Xiang, Y., Chen, Y., Luo, Y., He, K., Peng, G.: Android Data-Clone Attack via Operating System Customization. IEEE Access (2020)
36. statcounter: Mobile Operating System Market Share Worldwide (2025), https://web.archive.org/web/20250708194001/https://gs.statcounter.com/os-market-share/mobile/worldwide
37. Stute, M., Heinrich, A., Lorenz, J., Hollick, M.: Disrupting Continuity of Apple?s Wireless Ecosystem Security: New Tracking, DoS, and MitM Attacks on iOS and macOS Through Bluetooth Low Energy, AWDL, and Wi-Fi. In: USENIX Security (2022)
38. Stute, M., Narain, S., Mariotto, A., Heinrich, A., Kreitschmann, D., Noubir, G., Hollick, M.: A Billion Open Interfaces for Eve and Mallory: MitM, DoS, and Tracking Attacks on iOS and macOS Through Apple Wireless Direct Link. In: USENIX Security (2019)
39. TP-Link: General questions about Wi-Fi range of TP-Link wireless products (2021), https://www.tp-link.com/en/support/faq/2291/, accessed: 08/05/2025
40. ubreakifix by Asurion: Smartphone Data Transfers (2025), https://web.archive.org/web/20250430135435/https://www.ubreakifix.com/repairs/smartphones/services/data-transfer
41. Vanhoef, M., Piessens, F.: Advanced Wi-Fi attacks using commodity hardware. In: ACSAC (2014)
42. Verizon: Verizon Community: Data Transfer Fee (2024), https://community.verizon.com/t5/Other-Network-Discussions/Data-Transfer-Fee/m-p/1750512/highlight/true
43. Vodafone UK: What our Tech Team can do for you (2025), https://web.archive.org/web/20250729134247/https://www.vodafone.co.uk/help-and-information/tech-team
44. Wang, M., Yan, Z.: A Survey on Security in D2D Communications. Mobile Networks and Applications (2017)
45. Wu, J.: Magisk: The Magic Mask for Android (2018), https://github.com/topjohnwu/Magisk, accessed: 07/31/2025
46. Wu, T.D.: The secure remote password protocol. In: NDSS (1998)
47. Yamamoto, B., Wong, A., Agcanas, P.J., Jones, K., Gaspar, D., Andrade, R., Trimble, A.Z.: Received Signal Strength Indication (RSSI) of 2.4 GHz and 5 GHz Wireless Local Area Network Systems Projected over Land and Sea for Near-Shore Maritime Robot Operations. Journal of Marine Science and Engineering (2019)