Network-Level Insights into Google-Free Android Operating Systems

Bruno Mirčevski[1][0009−0007−1008−6708] and Second Author[1][1111−2222−3333−4444]

Faculty of Computer Science, Bialystok University of Technology, Wiejska 45a, 15-351, Bialystok, Poland

Abstract. This paper presents a comparative analysis of privacy in Android operating systems through network traffic inspection. Using a controlled man-in-the-middle setup and traffic decryption techniques, six Android distributions were evaluated: stock Google Android, GrapheneOS, iodéOS and LineageOS (in 3 variants). The study examined outbound connections, transmitted identifiers, and network traffic patterns to assess the impact of integrated Google services on user privacy. Results indicate that stock Android maintains the most communication with Google servers, transmitting identifiers and metadata even when idle. In contrast, privacy-focused systems minimize unsolicited traffic and offer stronger user control. The findings demonstrate that alternative Android distributions can substantially improve privacy without sacrificing core functionality, highlighting the potential of open-source ecosystems as viable, privacy-respecting alternatives to Google-dependent mobile environments.

Keywords: Android · Privacy · Network traffic.

1 Introduction

Android is the dominant mobile operating system and acts as a primary gateway to digital world for billions of users. It's crucial for core OS to be a reliable, stable and trustworthy platform, that allows users to run apps and services of their choice. While mobile app privacy has been widely studied [2, 11, 9, 15, 8, 14], less attention is typically paid to the operating system layer and to privileged service frameworks that silently enable additional functionality. Standard unprivileged apps are constrained by Android's permission model, although some attempt to circumvent it in creative ways [10, 7]. In contrast, privileged system apps operate with higher access to the device and OS, which makes them harder to understand and control. Most user-facing privacy controls focus on applications and permissions, yet a substantial amount of data exchange can occur below the app layer, through OS services, vendor components, and Google service frameworks. This work therefore compares the network behavior of multiple Android distributions and their preinstalled/privileged system components under controlled, reproducible conditions.

1.1 Android dependency on privacy-threatening services

While Android's core is the Android Open Source Project (AOSP), most consumer devices ship with additional proprietary components layered on top. They are often implemented as privileged system apps and service frameworks to provide baseline functionality such as push notifications, location assistance, app distribution/updates, device integrity checks and many more. Users and app developers commonly assume their presence and availability by default.

These components exists for a practical reason. Centralized services can significantly improve usability and efficiency, for example by reducing battery drain compared to per-app background connections, and by providing faster, more accurate location through fused network and sensor-based methods. However, this design also tightly couples many functions into a single service stack, so enabling useful capabilities may implicitly enable others that a user would not choose, such as advertising identifiers, telemetry, and behavioral tracking. In practice, the result is a large proprietary bundle that is difficult to audit and decompose into only the features a user actually wants.

On Android, these baseline capabilities are most commonly delivered through GMS (Google Mobile Services). Because Google's business model is strongly advertising-driven, many users may be unwilling to entrust this provider with broad access to device and their data. In principle, users should be able to choose which services they rely on and have confidence that these choices are enforced by the platform's permission and isolation mechanisms. This demand for transparency and controllability motivates alternative approaches such as microG and GrapheneOS sandboxed Google Play. These solutions aim to preserve device functionality and app compatibility while reducing privilege and improving user control over Google-related components.

1.2 Privacy-focused Android operating systems

Privacy-focused Android operating systems (also referred to as *custom ROMs*) are distributions derived from AOSP that aim to reduce data exposure at the OS level. They achieve this by limiting preinstalled privileged components, tightening default settings, and providing more transparent controls over permissions and network access. In contrast to *stock Android* (the factory system shipping with Google Mobile Services enabled by default) privacy-oriented systems minimize background communication, restrict persistent identifiers, and strengthen user enforcement mechanisms over non-essential components.

In this work we focus on three representative approaches. GrapheneOS prioritizes a hardened security model and strong isolation. Google components are not included by default, but can be installed in a dedicated sandboxed mode as ordinary apps, without the special privileges that Google Mobile Services (GMS) typically hold on factory systems. This design targets improved containment and user control while preserving high compatibility when the user decides to install Google services. GrapheneOS also improves low-level security, for example by replacing Android's default memory allocator with its own hardened implementation to make memory corruption exploitation harder [13].

LineageOS provides a lightweight, broadly supported AOSP-based system with minimal preinstalled software and extensive device compatibility. It can be deployed in multiple configurations: without Google services, with official proprietary Google Play Services and Play Store, or with microG as an open-source reimplementation of key Google APIs. These configurations represent different trade-offs between privacy, compatibility, and reliance on proprietary services.

Finally, iodéOS is a privacy-oriented distribution based on LineageOS that combines a de-Googled baseline with additional privacy features and preconfigured app ecosystem choices, shipping with microG. For completeness, CalyxOS and /e/OS follow a similar privacy-oriented direction (AOSP-based, microG-enabled, and designed for practical daily use). They were considered for the study, but were excluded because they did not receive updates to the current Android version during the measurement period.

Privacy-focused Android operating systems do not uniformly improve security. Some strengthen the platform through tighter defaults, hardening, and timely patching, while others may weaken it through slower updates or changes that reduce the strength of certain platform guarantees. In particular, microG support often requires signature spoofing, which weakens Android's signature-based authenticity checks. Systems that enable it typically try to minimize the risk by restricting spoofing to just microG. Some distributions do not support verified boot chain or re-locking the bootloader after installation, which weakens protection against physical tampering.

1.3 Challenges of network traffic analysis on Android

Network traffic analysis on Android is difficult by design and that is largely a security benefit. Modern protocols increasingly hide data behind encryption (TLS 1.3, QUIC/HTTP3, encrypted DNS). Even when traffic is captured, compression and binary encodings can make payloads hard to interpret without schemas or additional context. As a result, MITM-based decryption is inherently incomplete. Certificate pinning and anti-tamper mechanisms can block interception or break app functionality. Circumvention of these security mechanisms do not reliably yield a full plaintext view. While emulator-based setups enable highly reproducible traffic capture [1], this work uses a physical device to analyze behavior in a real environment, at the cost of reduced control and reproducibility. We accept that some flows will remain opaque, not all network communication will be revealed or understood. Our focus in on differences between studied Android systems and on general patterns.

In this work we captured traffic using a controlled man-in-the-middle gateway around a physical device. A Linux host acted as the Wi-Fi router and transparently redirected the device's HTTP(S) traffic through an intercepting proxy, while QUIC over UDP/443 was blocked to force TCP where decryption is feasible. To enable TLS inspection, the proxy CA was installed and promoted to a system-trusted certificate on the device (requiring root access). Where certificate pinning prevented interception, we attempted runtime unpinning via dynamic

instrumentation, acknowledging that this only worked for a subset of apps. Full packet captures were recorded and compared across Android distributions to analyze differences in network communication.

2 Related work

Prior work has established that Android privacy risks arise not only from user-installed applications but also, in some cases, from the operating system and its preinstalled components. Privacy-focused Android distributions have gained popularity as alternatives to stock vendor firmware, yet they remain comparatively understudied in the measurement literature. Consequently, there is limited empirical understanding of what practical privacy and control benefits different Android distributions provide to users.

A key OS-level measurement study addresses this gap by comparing several vendor-customized Android variants with two open-source distributions under a privacy-conscious baseline configuration [5]. It reports that all tested stock/vendor builds transmit substantial data to OS vendors and third parties even when the handset is idle, and that this collection persists without an effective opt-out. In contrast, the privacy-focused open-source /e/OS in their comparison exhibits minimal data transmission, whereas LineageOS still shows substantial communication to Google because it was evaluated with Google's proprietary app stack present. This distinction is important for studies like ours that evaluate multiple distributions and configurations (including variants with and without integrated Google services), since the presence of Google system components can dominate observed traffic and shape the overall privacy profile.

Follow-up work focusing specifically on OEM telemetry further shows that Samsung, Xiaomi, Huawei and Realme make extensive use of long-lived hardware identifiers and also collect installed-app lists and analytics data, sometimes even in connections that should not require persistent identifiers, enabling straightforward linkage to a user's identity when an OEM account is used [6].

Additional work by the same authors analyzes telemetry from two core Google system apps, Google Messages and Google Dialer, by decrypting and decoding the logging channels that forward data to Google [4]. The study reports that these apps disclose sensitive communication metadata, including when SMS messages and calls are sent or received, with timestamps and for calls duration. Messages also sends a hash derived from message content that can uniquely identify messages and link participants. It further finds that phone numbers may be transmitted and that events are tagged with persistent identifiers such as the Android ID, undermining anonymity and leaving users with no effective opt out. Taken together, these results suggest that even basic default communication apps on Google Android cannot be assumed to be trustworthy from a privacy perspective.

Complementing OS-level telemetry measurements, a recent study focuses specifically on what pre-installed Google components persistently store on-device and shows that Google's role in Android privacy extends beyond outbound traf-

fic [3]. It reports that Google Play Services, the Play Store, and other bundled Google apps receive and store multiple cookies, advertising-related identifiers, and tracking links even after a factory reset and even when the user has not opened Google apps, with no dedicated consent prompt and no practical opt-out. The study highlights the Google Android ID as a persistent identifier provisioned early and widely reused in subsequent Google connections, and documents how additional tokens and cookies can be stored and later transmitted alongside telemetry or analytics events. Together, these findings reinforce that Google's integrated service stack functions as a central privacy-relevant layer of the Android ecosystem, shaping device identification and data handling independently of user-installed applications.

A broader view of Google's privacy impact is given in a report that describes data collection as an ecosystem across platforms, apps, and advertising services, not as a single feature of Android [12]. It distinguishes "active" data sharing (when users directly use Google products) from "passive" collection that happens in the background via Android/Chrome and via tracking/analytics components embedded in many third-party apps and websites. The report argues that these different data sources can be combined to build detailed user profiles, and that "pseudonymous" identifiers (like ad IDs or cookies) can still be linked back to user accounts when they appear together in the same workflows.

Overall, prior work consistently indicates that Google's integrated service stack is a dominant driver of privacy-relevant data flow on Android. It enables persistent device identification, background telemetry, and cross-context linkage between app usage, web activity, and advertising infrastructure, often with limited user visibility and weak practical opt-out mechanisms. At the same time, the literature suggests that meaningful reductions in unsolicited communication are possible when Google components are absent or replaced with open source variants [5]. Alternative configurations and distributions remain less widely deployed, less familiar to typical users, and comparatively under-explored in reproducible measurement studies. This motivates systematic, real-device comparisons of stock systems versus privacy-oriented alternatives and configurations to quantify what privacy improvements are achievable in practice and what trade-offs they entail.

3 Experiment design

3.1 Hardware and software

We used a single Google Pixel 6a (codename *bluejay*) to install and evaluate studied Android distributions. This model was chosen due to its broad compatibility with alternative operating systems. The full measurement setup is shown in Fig. 1. All tested builds base on Android 16 and were obtained in December 2025:

- Stock Android: `BP4A.251205.006`
- LineageOS: `lineage-23.0-20251206` (`BP2A.250805.005`)

- LineageOS for microG: `lineage-23.0-20251203-microG` (`BP2A.250805.005`)
- GrapheneOS: `2025121200` (`BP4A.251205.006`)
- iodéOS: `7.0-20251129-bluejay` (`BP2A.250805.005`)

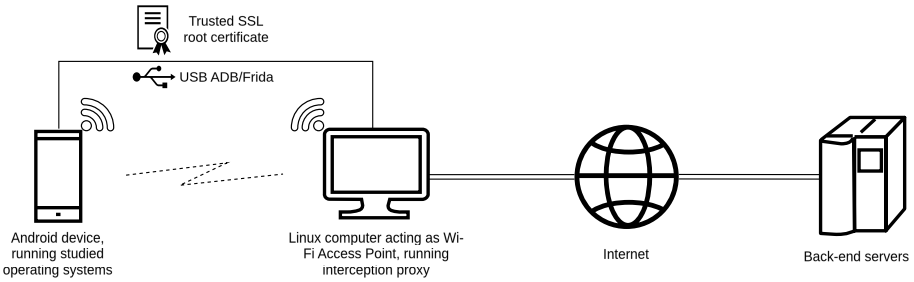

Fig. 1. Measurement setup. Android device running studied operating systems is configured to access Internet using Wi-Fi. Access point is hosted on a Linux computer, running traffic interception and capturing software. Custom system certificate is installed on the phone using root access. USB connection is used for ADB/Frida communication.

3.2 Measurement scenarios

To enable consistent, comparable measurements across Android distributions, four setup scenarios were defined. Each scenario specifies the Google account state, consent and permission posture, and which Google components are present. Not all scenarios apply to every operating system due to their characteristics. Scenarios were adapted to each system's capabilities: on GrapheneOS, sandboxed Google Play Services were installed immediately after setup; on systems with microG, the component was enabled during initial setup when prompted, or activated manually immediately afterward. As a general principle, any required configuration step not available during initial setup was performed as soon as possible thereafter.

Scenario A: Default Setup with Google Account. The device is configured to reflect a typical user path. Google account sign-in is performed during setup (or immediately after) and default recommendations are accepted (including prompted permissions).

Scenario A2: Minimal Setup with Google Account. This scenario extends *Scenario A* by adjusting settings towards privacy while retaining Google account functionality. During and after initial setup, all optional consents and permissions are declined. System and account settings are adjusted to maximize user privacy. This scenario is not applicable to systems using microG or GrapheneOS, as these distributions are already designed to minimize privacy risks and do not

implement or prompt users for privacy-threatening features characteristic of the factory system.

Scenario B: Minimal Setup without Google Account. The device is configured with maximum privacy in mind, avoiding Google account sign-in entirely. During initial setup, only strictly required prompts are accepted and optional data-sharing is declined. System settings are adjusted to maximize user privacy. Google components or their functional replacements remain present and enabled to preserve baseline device functionality and app compatibility, but operate without account.

Scenario C: Extremely Minimal Setup without Google Components. This scenario applies only to distributions that do not bundle or enable Google services by default. It is not applicable to the official factory system, which enables Google components by default. If microG is preinstalled, it remains disabled.

3.3 Measurement tasks

We performed the following measurement tasks sequentially for each scenario. Network traffic was captured using Wireshark for basic analysis.

Task 0: Initial setup. Configure the device according to the target scenario, connect to Wi-Fi, complete account sign-ins and install components as specified. Network traffic capture begins with the device's first Internet connection and continues until 10 minutes after setup completion.

Task 1: System idle. Ensure the system and apps are up to date and that no update-related background jobs are running. Reboot the device to begin capture, then keep it connected to the monitored network for at least 30 minutes. During this period, unlock the device three times and scroll through the app list. Do not open any apps.

Task 2: Basic apps interactions. Open preinstalled apps and perform simple actions that should not require an Internet connection

- Settings: disable haptic feedback; toggle dark mode; check per-app battery usage
- Clock: set an alarm
- Files: create a folder
- Contacts: create a local (on-device) contact

Beyond direct task comparisons, additional packets of interest were examined to compare data transmission practices across different system configurations and to characterize what data is sent and how. This analysis required TLS decryption via the intercepting proxy and, in some cases, runtime certificate

unpinning with Frida using scripts from HTTP Toolkit.[1] Although Frida-based unpinning failed in most attempts, it was useful in a small number of cases.

To complement network-level observations, Google Takeout[2] was used to inspect server-side data associated with the test Google account after running scenarios involving Google services (Stock Android, GrapheneOS with sandboxed Google Play Services, and microG-based distributions).

4 Results

4.1 Data usage across operating systems and scenarios

Total network data usage across all operating systems and applicable scenarios for performed tasks, is presented on charts in Figures 2–4. To keep the results comparable and avoid dominance by bulk transfers, we excluded traffic to domains used for app and system update payload delivery (large binary downloads), though this exclusion was not perfect and some residual update-related traffic remains in the dataset. Generally, we expect lower network traffic in configurations with fewer Google services present.

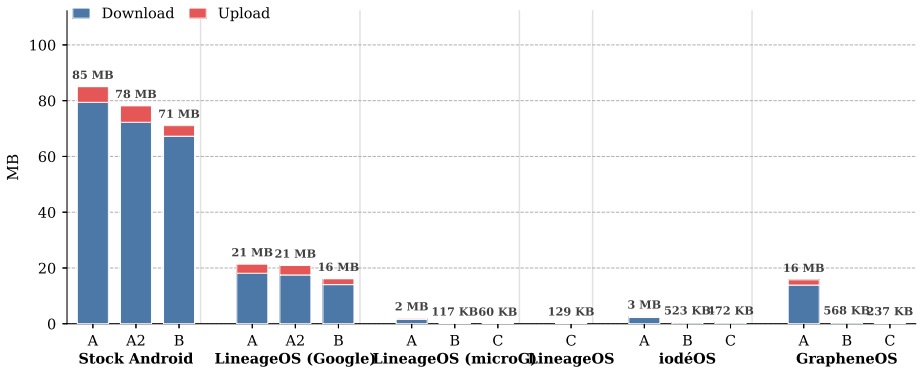

Fig. 2. Task 0: Initial setup data usage

Figure 2 illustrates network activity during initial device setup. As expected, stock Android shows the highest data usage. The modest reductions between privacy-adjusted scenarios indicate that configuration choices provide little control over automatic updates and background synchronization. Across all distributions, variations align with service architecture: configurations incorporating more Google services generate proportionally higher traffic, while de-Googled baselines demonstrate substantially reduced exchange. LineageOS with official

[1] https://github.com/httptoolkit/frida-interception-and-unpinning
[2] https://takeout.google.com/

Google Play Services generates substantially less traffic than stock Android due to its more minimal base system, yet remains considerably higher than configurations without proprietary Google services, with the microG variant and Google-free installation achieving notable further reductions. On GrapheneOS, Scenario B demonstrates that even when sandboxed Google Play Services are installed and granted network and background permissions, they remain largely inactive when not deliberately invoked, contrasting with the persistent background activity characteristic of integrated GMS deployments.

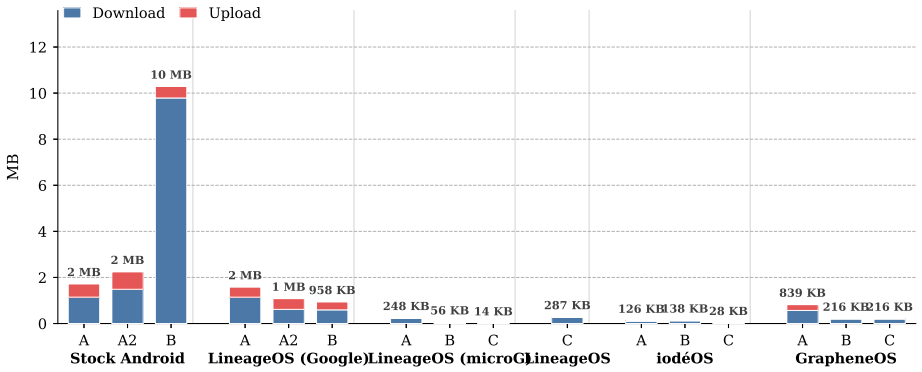

Fig. 3. Task 1: Idle data usage

Network activity observed during system idle following a reboot is presented in Figure 3. Stock Android again shows the highest data usage, though Scenario B exhibits unexpectedly elevated traffic compared to both Scenario A2 and Scenario A. This anomaly stems from substantial background communication with www.gstatic.com, likely triggered by automatic updates or application background activity during the measurement time.

LineageOS in Scenario C similarly deviate from anticipated patterns, generating more idle traffic than its microG variant. LineageOS made connections to agnss.goog (Google Assisted GNSS) and www.gstatic.com. The agnss.goog endpoint was also observed in LineageOS (microG) during other measurement tasks, indicating that the entire LineageOS family retains this Google-dependent location mechanism. Other alternative distributions either replace this component with independent implementations or allow users to disable it entirely. GrapheneOS exhibits virtually identical idle traffic in Scenarios B and C, confirming that sandboxed Google Play Services remain dormant when not actively invoked.

Figure 4 shows network activity during basic app interactions that should not require Internet connectivity. Stock Android dominates with substantial data usage mostly by uncontrolled updates. All alternative distributions demonstrate dramatically lower traffic, with de-Googled configurations remaining effectively

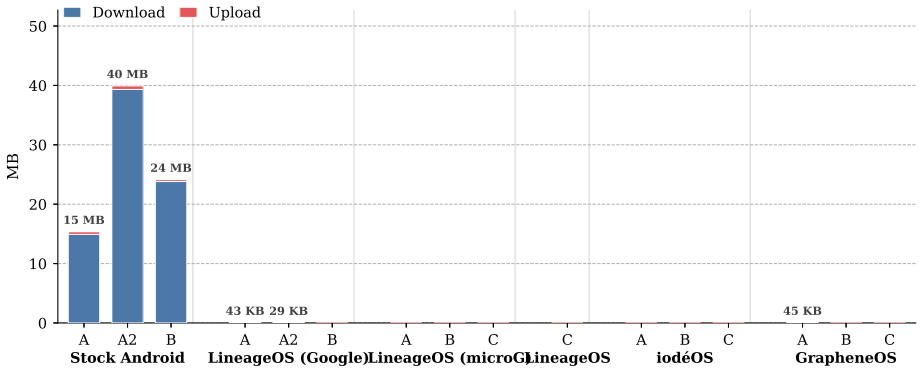

Fig. 4. Task 2: Basic apps interactions data usage

silent. Notably, whenever official Google Play Services were present and a Google account logged in at least a single request to Google infrastructure occurred.

4.2 Domain distribution and Google dependencies

Figure 5 presents the domain-level distribution of network traffic for Scenarios A and B. They were selected as the most realistic configurations incorporating essential Google functionality while differing in account linkage and privacy-related settings. Overall traffic is dominated by large content transfers, as systems download necessary resources and updates. Stock Android contacts the widest range of Google domains, spreading traffic across many different endpoints. Alternative systems show less Google communication, though patterns vary by design. GrapheneOS makes more requests than microG-based systems because it downloads Google services on demand from its own app store rather than having them preinstalled. IodéOS and LineageOS with microG reach fewer Google servers overall, as microG minimizes Google connectivity while still enabling basic features. IodéOS additionally relies on some Mozilla infrastructure not present in other systems.

Table 1 shows the infrastructure dependencies for basic system functions. LineageOS variants remain closest to stock Android, relying entirely on Google servers for connectivity checks, remote provisioning, and location assistance. IodéOS adopts an intermediate position, using independent infrastructure for connectivity checks while still depending on Google for remote provisioning and location services. GrapheneOS demonstrates the strongest de-Googling, substituting all three functions with its own or independently operated infrastructure. Both iodéOS and GrapheneOS expose these mechanisms through explicit system settings toggles, though GrapheneOS offers more granular control: users may choose between its independent proxy infrastructure and official Google servers for each function, with defaults configured to avoid Google connections entirely.

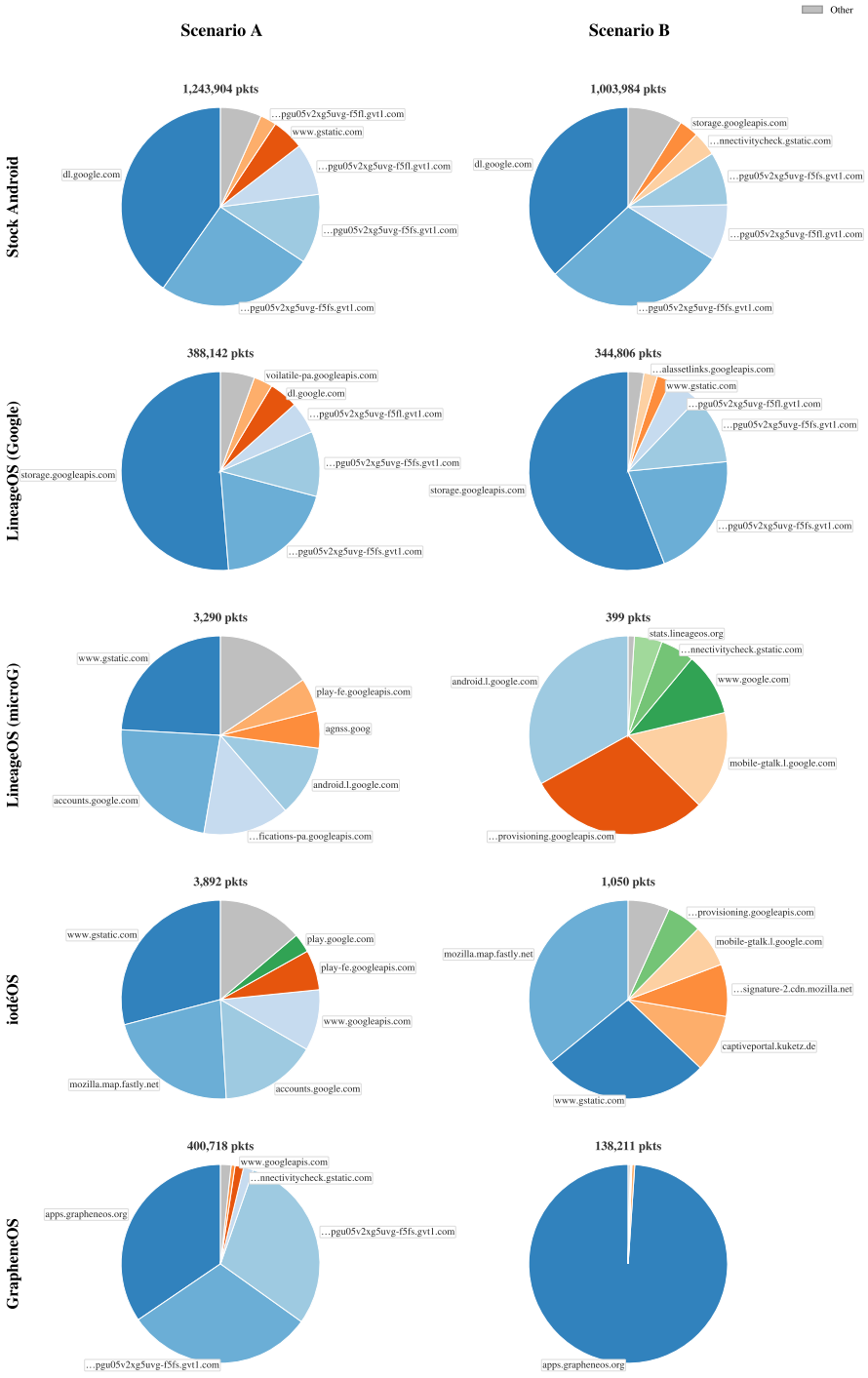

Fig. 5. Domain-level distribution of network packets for scenarios A (with Google account) and B (without Google account, but Google Services present), aggregated over all tasks. Each pie shows the top domains by packet count for a given operating system and scenario.

Table 1. Domains observed for connectivity check, remote provisioning, and location assistance.

System	Connectivity check	Remote provisioning	Location assistance
Stock Android	connectivitycheck.gstatic.com	–	supl.google.com agnss.goog
LineageOS	connectivitycheck.gstatic.com	remoteprovisioning.googleapis.com	supl.google.com agnss.goog
iodeOS	captiveportal.kuketz.de	remoteprovisioning.googleapis.com	supl.google.com
GrapheneOS	connectivitycheck.grapheneos.network	remoteprovisioning.grapheneos.org	gs-loc.apple.grapheneos.org

4.3 Additional findings

[Mention that some of these results where gathered earlier on android 14, we where not able to confirm them later in android 16 due to frida method not working anymore, but we expect general trend to remain similar]

NOTES FOR: Android "checkin" request: in stock os sends real imei and serial number. in microg fake and randomly changing imei and mac adress is used. mac addresses start with "b4:07:f9" but last part varies, all imei starts with 35503104... android id in microg requests is often empty, zero. microg sends real device information like device model, os version, timezone. microg sends serial key like 008741D1E2C1C2C7, which is also displayed in microg settings, it acts as a unique generated identifier to hide real device identifiers.

NOTES FOR: app-measurment.com/region1.app-measurement.com on os with google services we noticed requests to these domains. on lineage stock and graphene. these requests contain app name, source of installation (google play/aurora/manual), system theme (dark/light), app version, android id. Info about installed and not open apps where send too. We where not able to verify what region1.app-measurement.com on graphene os contained because grapheneos blocked frida attempts.

NOTES FOR: ADID In few requests in stock we noticed ADID (format: d72ed0d9-c534-4c76-bf42-5185a6e7850d). It was present in some google requests to different domains. value was constant across requests. It is resettable in settings, after removing it in settings we noticed it being changed into zeros.

NOTES FOR: Microg location microG is sending nearby cell-tower and Wi-Fi scan data to api.positon.xyz to estimate the phone's location (network-based geolocation). The cell-tower request succeeds and returns an approximate lat/lng with accuracy, while the Wi-Fi-based lookups return "404 Not found," meaning that service/key doesn't have matching Wi-Fi AP location data available. It happens only after users enables it and chooses location provider. No google. <request base64="false"><![CDATA[POST /v1/geolocate?key=627db168-2a5e-11ef-afbd-a75f87425aa4 HTTP/1.1 User-Agent: microG/0.3.10.250932 (Linux; Android 16; com.google.android.gms) Content-Type: application/json; charset=utf-8 Host: api.positon.xyz Connection: keep-alive Accept-Encoding: gzip, deflate, br Content-Length: 298

"considerIp":false,"homeMobileCountryCode":260,"homeMobileNetworkCode":1,"radioType":"wcdma","c
94],"fallbacks":"lacf":true,"ipf":false]]></request> <status>200</status> <responselength>532</responselength> <mimetype>JSON</mimetype> <response base64="false"><![CDATA[HTTP/2 200 OK Server: nginx/1.22.1 Date: Thu,

11 Dec 2025 10:49:24 GMT Content-Type: application/json; charset=utf-8 Content-Length: 322 X-Powered-By: Express Etag: W/"142-LUquWahK9inoXp0ExE4zfupVysQ"
"raw":["timestamp":1772928000000,"location":"lat":53.044937,"lng":21.585724,"accuracy":3294,"cellTower
94],"location":"lat":53.044937,"lng":21.585724,"accuracy":3294]]></response>

4.4 Google Takeout

5 Conclusion

[Based on requests google can determined if device is running stock os, or microg or graphene] [Adjusting Privacy settings matters, it makes difference, but for higher privacy levels alternative OS is required] [All alternative OS do not force EULA]

References

1. Jimenez-Berenguel, A., Campo, C., Moure-Garrido, M., Garcia-Rubio, C., Díaz-Sánchez, D., Almenares, F.: Parrot: Portable android reproducible traffic observation tool (Sep 2025). https://doi.org/10.48550/arXiv.2509.09537
2. Jin, H., Liu, M., Dodhia, K., Li, Y., Srivastava, G., Fredrikson, M., Agarwal, Y., Hong, J.I.: Why are they collecting my data? inferring the purposes of network traffic in mobile apps. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. **2**(4) (Dec 2018). https://doi.org/10.1145/3287051, https://doi.org/10.1145/3287051
3. Leith, D.J.: Cookies, identifiers and other data that google silently stores on android handsets. Tech. rep., Trinity College Dublin (Feb 2025), https://www.scss.tcd.ie/doug.leith/pubs/cookies_identifiers_and_other_data.pdf
4. Leith, D.J.: What data do the google dialer and messages apps on android send to google? In: Li, F., Liang, K., Lin, Z., Katsikas, S.K. (eds.) Security and Privacy in Communication Networks. pp. 549–568. Springer Nature Switzerland, Cham (2023)
5. Liu, H., Patras, P., Leith, D.J.: Android mobile os snooping by samsung, xiaomi, huawei and realme handsets (Oct 2021), https://www.scss.tcd.ie/doug.leith/Android_privacy_report.pdf
6. Liu, H., Patras, P., Leith, D.J.: On the data privacy practices of android oems. PLOS ONE **18**(1), 1–15 (01 2023). https://doi.org/10.1371/journal.pone.0279942, https://doi.org/10.1371/journal.pone.0279942
7. LocalMess: Disclosure: Covert web-to-app tracking via localhost on android (2025), https://localmess.github.io/
8. Pham, A., Dacosta, I., Losiouk, E., Stephan, J., Huguenin, K., Hubaux, J.P.: HideMyApp: Hiding the presence of sensitive apps on android. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 711–728. USENIX Association, Santa Clara, CA (Aug 2019), https://www.usenix.org/conference/usenixsecurity19/presentation/pham
9. Razaghpanah, A., Nithyanand, R., Vallina-Rodriguez, N., Sundaresan, S., Allman, M., Kreibich, C., Gill, P.: Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem (01 2018). https://doi.org/10.14722/ndss.2018.23009

10. Reardon, J., Feal, Á., Wijesekera, P., On, A.E.B., Vallina-Rodriguez, N., Egelman, S.: 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 603–620. USENIX Association, Santa Clara, CA (Aug 2019), https://www.usenix.org/conference/usenixsecurity19/presentation/reardon
11. Ren, J., Lindorfer, M., Dubois, D.J., Rao, A., Choffnes, D.R., Vallina-Rodríguez, N.: Bug fixes, improvements, ... and privacy leaks - a longitudinal study of pii leaks across android app versions. In: Network and Distributed System Security Symposium (2018), https://api.semanticscholar.org/CorpusID:4231807
12. Schmidt, D.C.: Google data collection. Tech. rep., Digital Content Next (DCN) (Aug 2018), https://digitalcontentnext.org/wp-content/uploads/2018/08/DCN-Google-Data-Collection-Paper.pdf
13. Stefanski, N.: Exploring GrapheneOS secure allocator: Hardened malloc (Sep 2025), https://www.synacktiv.com/en/publications/exploring-grapheneos-secure-allocator-hardened-malloc
14. Sutter, T., Tellenbach, B.: Firmwaredroid: Towards automated static analysis of pre-installed android apps (May 2023). https://doi.org/10.1109/MOBILSoft59058.2023.00009
15. Wang, Z., Li, Z., Xue, M., Tyson, G.: Exploring the eastern frontier: A first look at mobile app tracking in china. In: Sperotto, A., Dainotti, A., Stiller, B. (eds.) Passive and Active Measurement (PAM 2020). pp. 314–328. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020)