

What Play services knows

A static reverse engineering of **com.google.android.gms 26.40.32** as redistributed, unmodified and Google-signed, by the GrapheneOS project, and what it still means when the app is caged in a sandbox.

402

PERMISSIONS

468

LOG SOURCES

200

MODULES DECLARED

01 Executive summary

03 Anatomy of the app

05 What leaves the device

07 Where it goes

09 Recommendations

11 Downloaded code

13 Support

02 Provenance and method

04 Identity: the ladder

06 When it happens

08 The GrapheneOS delta

10 Beyond settings

12 Evidence

Sandboxing does not make Play services private. It makes it consent-gated, and revocable.

26.40.32

base.apk · 194,764,356 bytes
sha256 1bfbf0c0ddab7470...e0e9b3f
signer sha256 5f2391277b1dbd48...b5402

ANALYSIS COMPLETE

Executive summary

GrapheneOS ships Google Play services exactly as Google builds it. The privacy argument does not rest on a different APK. It rests on the operating system refusing to give it the privileges it was designed around. This report measures both halves of that claim: what the app contains, and what the sandbox takes away.

402 PERMISSIONS DECLARED	481 EXPORTED COMPONENTS	778 LITERAL HOSTNAMES	468 TELEMETRY LOG SOURCES
161,628 DECOMPILED CLASSES	200 MODULES DECLARED	12 h CHECK-IN CADENCE	132 PERMISSIONS NEVER GRANTABLE

The six findings

- Identity is minted at first check-in and survives everything except clearing app data.** The device gets a 64-bit Android ID, a check-in logging ID and a security token written to `/data/misc_ce/<user>/checkin` inside the app's own sandbox. That triple underlies every later association Google can make.
- Clearcut uploads are account-aware by design.** The logging context carries `uploadAccount`, `logAndroidId`, `isAnonymous`, the package name and version, and a PII level, and 468 distinct log sources exist in this build. [SEEN](#)
- The sandbox removes the dangerous half.** 132 of the declared permissions are `signature/privileged` and can never be granted to an unprivileged app: reading IMEI without consent, MAC addresses, logs, secure settings, other users' data. 175 more are not even defined by AOSP. [SEEN](#)
- The code that runs is decided at runtime.** 200 modules are declared in the APK, and 13 more were downloaded onto the audited phone that are not in the repository at all. Collected and decompiled from a test device, these modules turn out to be ordinary code (barcode and text recognition, a Maps renderer, an ads renderer, passkeys, launcher search, an on-device search index), but none asks for a permission of its own: each runs with the permissions of Play services, or of the app that loads it. The DroidGuard engine is also a downloaded, certificate-pinned APK. The module plan is a remote flag fetched with the device and account identifiers, so Google can deliver a different signed module to one device or account without a Play services update. Only code signed by Google loads. The GrapheneOS sandbox contains this code; it does not stop it from arriving. **So there is no single "Play services" that everyone runs and that could be audited once for all.** The device sends its configuration to Google, and Google decides which modules, in which versions, that particular device gets. This report audits the host app (the Play services installation package, the APK) and a sample of the dynamic code that one device downloaded. [SEEN](#) (section 11)
- What remains is still enough to identify you.** `READ_PHONE_STATE` (user-grantable) unlocks IMSI and SIM serial for check-in. `READ_CONTACTS` unlocks contact upload. `QUERY_ALL_PACKAGES` is a *normal* permission, so the installed-app list is available without any prompt. [SEEN](#)

6. **Network access is the only hard switch, and I would not recommend it.** On GrapheneOS, `INTERNET` itself is a dangerous permission, so the per-app network toggle severs Google's telemetry. It also severs push notifications, which is the main reason most people install Play services, so it breaks much of what you wanted Play services for. `INFERRED`

What can be done about it. Section 09 lists settings that exist today; section 10 goes further: filtering the network, modifying the APK and patching GrapheneOS. Network filtering is the practical one. Modifying the APK and patching GrapheneOS are **not recommended**: they are a theoretical discussion of whether they would make sense, and they are a much bigger pain than they look.

Reading this report. Everything is static analysis of the APK. Nothing here was captured from a live device's traffic. Claims tagged `SEEN` have a file and line in the appendix; `INFERRED` means the conclusion is reasoning over what was seen. Server-side behaviour (what Google retains, how it joins records) is outside what any APK can tell you.

02

Provenance and method

The APK was pulled from GrapheneOS's own app repository, and every byte was checked against the repository's signed metadata.

ITEM	VALUE
Repository	<code>https://apps.grapheneos.org/</code> · <code>metadata.1.0.0.sjson</code> (signify-signed)
Package	<code>com.google.android.gms</code> · <code>versionCode 264032035</code> · <code>26.40.32</code> (260400-995630914)
Channel / SDK	<code>stable</code> · <code>minSdk 35</code> · <code>arm64-v8a + armeabi-v7a</code>
base.apk	194,764,356 bytes · sha256 <code>1bfbf0c0ddab7470...</code> (matches metadata)
Signer	<code>CN=Android, O=Google Inc.</code> · sha256 <code>5f2391277b1dbd48...b5402</code>
Splits analysed	<code>AdsDynamite</code> · <code>CronetDynamite</code> · <code>DynamiteLoader</code> · <code>DynamiteModulesA/C</code> · <code>GoogleCertificates</code> · <code>MapsDynamite</code> · <code>MeasurementDynamite</code> · <code>maps_core</code> · <code>config.en</code> · <code>config.xxhdp</code>
Companion apps	<code>com.google.android.gsf 14</code> · <code>com.android.vending 53.5.48-34</code>

Why the signature check matters. GrapheneOS does not patch Play services; it patches the *operating system* so that an unmodified Google binary runs without the privileged hooks it expects. The signature match makes the phrase "official release without modifications" in the repository description verifiable.

What was done

Unpack	apktool (resources + manifest), raw zip for assets and native libraries
Strings	DEX string tables parsed directly (773,000 strings) plus a raw byte scan of every asset
Code	jadx 1.5.6, full sweep: 161,628 classes, 920 MB of Java, plus targeted reads of the check-in, Clearcut, Measurement, Phenotype, ads and verifier modules
Policy	permission protection levels diffed against GrapheneOS <code>frameworks/base</code> branch 17
Scope	GMS base + 11 splits; GSF and Play Store only for identity context
Further analysis	Module manifests decoded (200 modules, 80 module-set entries); all 39 arm64 native libraries triaged, the privacy-relevant ones decompiled in Ghidra
Modules	24 downloaded module APKs (19 builds), the DroidGuard engine and the module registry pulled from a rooted Android 13 test emulator, hash-checked against Play's own records, and decompiled (section 11)

What could not be established

- No packet capture: cadence below is from code and flag defaults, not observation.
- Field-level check-in semantics: the request protobuf is name-stripped, so individual field numbers cannot be labelled with certainty. The *inputs* to the request are visible and are what this report cites.
- The DroidGuard engine was obtained (an x86-64 build from a test emulator, section 11), but the program it runs is sent per request and was not captured, so what it measures on a given call is unknown. The native attestation verifier (`liboak`) checks Google's own servers, not the phone.
- Anything Google does after ingestion: retention, joins, model training.

03

Anatomy of the app

Play services is a module host. It declares 1,310 components, but 1,054 of them belong to 200 modules described in `assets/ChimeraManifest.pb`. A second manifest, `assets/ChimeraModuleSetList.pb`, lists 80 module-set entries: 12 of them are on-demand splits, and 8 modules ship twice, once as an APK inside `assets/chimera-modules/` and once as an install-time split. On the audited Pixel 10 Pro Fold, ART was compiling **thirteen** module APKs from `/data/user_de/0/com.google.android.gms/app_chimera/m/`, and none of them ships in the GrapheneOS repository. 17 DEX files, 801 assets, 29.5 MB of native code in base.apk plus 3 more libraries in two splits. SEEN

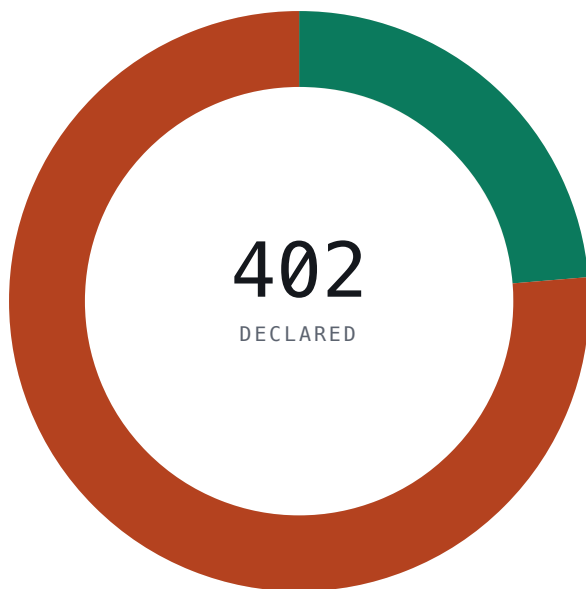


Figure 1: can it ever be granted?



■ normal · 55 ■ dangerous · 35 ■ special access · 5 ■ signature/privileged · 132 ■ not in AOSP · 175

Figure 2: the same 402 permissions, by protection level

Surface area

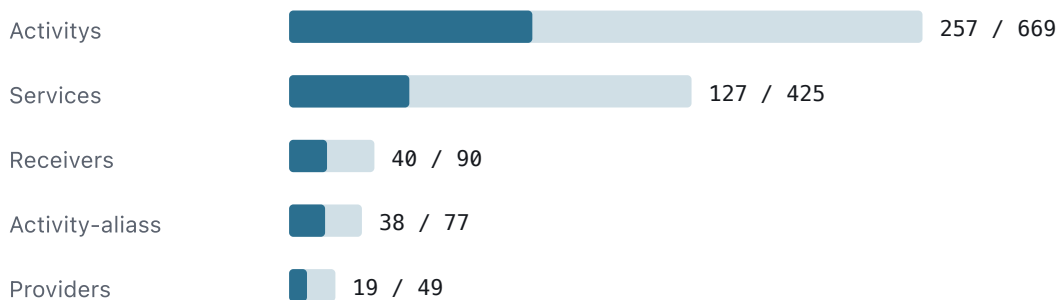


Figure 3: components declared vs. exported (exported / total)

1,310 components, 481 of them exported. Export is how other apps reach into Play services: the account provider, the Fast Pair slice provider, the Phenotype configuration provider, the Chimera module broker, the Settings search provider. Only 60 of those exported components declare an `android.permission`. The other 421 are open at the manifest level, including the App Set ID provider and the U2F/FIDO services, and must validate their callers in code.

```
AndroidManifest.xml
  android:sharedUserId="com.google.uid.shared"
  android:isolatedSplits="true"  android:forceQueryable="true"
  android:compileSdkVersion="37"

  com.google.android.gms.checkin.CheckinService
  com.google.android.gms.checkin.CheckinServiceSecretCodeReceiver
  com.google.android.gms.appset.service.AppSetIdProviderService  (exported, no permission)
```

The permission story

GMS asks for everything, because it is written to be a privileged system app. On AOSP it is one: signed with the platform key, placed in `/system/priv-app`, granted signature permissions at boot.

In the GrapheneOS model it is a normal app with a normal UID. The **132** signature/privileged permissions are simply denied, and **175** more refer to permissions no AOSP build defines, so they are denied too. What is left is a much smaller but not empty capability set.

Shared UID. `com.google.uid.shared` means Play services, Google Services Framework and the Play Store run as one Linux user. They share a data directory, a keystore namespace and an Android ID. Sandboxing isolates them from *you*; it does not isolate them from each other.

04

Identity: the ladder

"Tied to the user" covers several identifiers. GMS maintains a ladder of them, and each rung survives a different kind of reset.

01

Android ID (SSAID)

64-bit hex, scoped to the signing key + user + device. GMS and the other Google apps share one value because they are signed with the same key (and run under one UID). Survives app reinstall and OS updates; resets on factory reset. [SEEN](#) `Settings.Secure "android_id"`

02

Check-in logging ID + security token

Minted by the check-in server on first contact and persisted in the app sandbox (`/data/misc_ce/<user>/checkin`, `security_token`). This is the "GSF ID" of folklore. It is what makes an Android ID into an account-independent device identity. [SEEN](#) `bjisc.java:41-42,122-124`

03

Advertising ID (GAID) and App Set ID

Generated and stored locally by Play services, resettable by the user, exposed to apps through the `IAdvertisingIdService` binder. Limit-ad-tracking is a local flag; the identifiers stay put unless the user resets them. [SEEN](#) `adid/service/*`, `"limit_ad_tracking"`

04

Google account

When an account is added, GMS holds its name, Gaia ID and OAuth tokens, and Clearcut/Phenotype can attach events to it. The Phenotype database has an `accounts` table and an explicit anonymous row (`accounts.name = ''`). [SEEN](#) `phenotype/sync/*`, `PlayLoggerContext`

05

Device fingerprint

Build fields, screen, locale, SIM operator and, with `READ_PHONE_STATE`, IMSI and SIM serial number. Sent with check-in. [SEEN](#) `bjtv.java:131,340,347`

06

Attestation identity

DroidGuard is a downloaded APK. A signed *creation response* supplies `byteCode`, `vmUrl`, `vmChecksum` and a validity window, and its signature is checked before any field is used. The engine is fetched from `https://www.gstatic.com/droidguard/`, cached at `app_dg_cache/<key>/the.apk`, checked against two pinned signing-certificate digests and loaded with `DexClassLoader` as `com.google.ccc.abuse.droidguard.DroidGuard`. Google can replace the engine at any time; the pin is the only local control. Play Integrity verdicts then bind an app's request to a hardware-backed key. [SEEN](#) `bysw.java:90-92`, `bysr.java:14-15`, `bysq.java:53-60` · [INFERRED](#) (verdict semantics)

How the identifiers relate

IDENTIFIER	WHO ASSIGNS	LIFETIME	RESET BY	VISIBLE TO APPS?
SSAID	OS, per signing key	device	factory reset	yes, no permission
Check-in ID	Google, at check-in	install	clear app data	no
Security token	Google, at check-in	install	clear app data / 401	no
GAID	Play services, local	user-controlled	Settings → Ads → reset	yes, via binder
App Set ID	Play services, local	developer scope	reset / uninstall all	yes, via binder
Account / Gaia	Google	account	sign out	via AccountManager
RLZ / legacy client IDs	Google, legacy	install	varies	no (GServices)

Signing out. Removing your Google account does not reset rungs 01 to 03. The check-in ID persists, the Android ID persists, and the advertising ID persists. "Signed out" means "not joined to an account on the server", not "unidentified". INFERRED

What leaves the device

Thirteen subsystems carry data off the device. For each one, the table gives the endpoint found in the code, what triggers it, what it carries, and what the GrapheneOS sandbox changes about it.

SUBSYSTEM	ENDPOINT (AS SEEN)	WHAT LEAVES	UNDER GRAPHENEOS
Check-in device registration	android.googleapis.com/checkin android.clients.google.com/checkin checkin.gstatic.com/checkin	Android ID, logging ID, security token, build fingerprint, device class, locale, SIM operator; IMSI + SIM serial if Phone permission is granted; module inventory. Response can refresh the Android ID (e.g. after security-token loss).	Works. Without READ_PHONE_STATE the SIM fields are null. Everything else still identifies the device.
Clearcut play log	play.googleapis.com/log play.googleapis.com/log/batch	Batched events with package, version, log source, PII level and optional upload account + Android ID; 468 log source categories exist.	Runs. No extra permission needed beyond network. Account attached only if one is present.
Firebase / Measurement	app-measurement.com/a app-measurement.com/s/d app-measurement.com/config/app/...	App instance ID, app id/version, OS, device model, language, country, session and event stream from apps that embed Firebase.	Runs. Uses its own app-instance ID rather than the Android ID by default.
Phenotype config & experiments	cloudconfig.googleapis.com (service host resolved at runtime)	Device config, experiment tokens, per-account rows; 24 h sync. Tokens are committed and can be uploaded in heterodyne batches.	Runs. Experiment tokens are pseudonymous but long-lived.
Auth	android.clients.google.com/auth/* oauth2.googleapis.com	Account name, OAuth tokens, FRP validation, device state for account recovery.	Runs. Tokens live in the shared Google UID.
Verify Apps Play Protect	security.verifyapps. UPLOAD_APK → Google	APK bytes, package name, version code, install source for packages being scanned.	Only if Play Protect is enabled in the sandboxed Play Store. Can be turned off.
Contacts romanesco	people-pa.googleapis.com	Contact records and sync metadata when Contacts permission is granted; also contact-match lookups.	Denied unless the user grants Contacts. Default install: no access.
Location / Nearby	nearby.googleapis.com maps.googleapis.com fitness.googleapis.com	Fused location requests, BLE/UWB ranging results, Fast Pair device sightings, Place lookups.	Needs Fine/Coarse location + Bluetooth scan. GrapheneOS's own network location can replace Google's.
FCM / push	mtalk.google.com:5228 gcm.googleapis.com android.apis.google.com/c2dm/register3	Persistent socket, registration token, delivery receipts, which give a live "this device is online from this IP" signal.	Runs if online. Blocking the network kills push along with telemetry.
Ads	googleads.g.doubleclick.net pagead2.googleadsyndication.com	Advertising ID + limit flag to apps that ask; ad request pings when apps use AdMob.	GAID exists locally; user can reset/limit it. The privileged AD_ID path is unavailable.

SUBSYSTEM	ENDPOINT (AS SEEN)	WHAT LEAVES	UNDER GRAPHENEOS
Attestation	gstatic.com/droidguard/* playintegrity API via Play services	Two attestations that are often confused. Device attestation is DroidGuard: a downloaded, certificate-pinned APK (04.06) that produces evidence about the phone. Server attestation is <code>liboak_attestation_verification_jni.so</code> , which verifies AMD SEV-SNP and Intel TDX evidence, DICE chains and Rekor transparency-log entries about <i>Google's own</i> confidential-computing enclaves; it reads nothing about the device and makes no network calls.	Basic verdicts can pass; device/strong verdicts rely on certification GrapheneOS does not have.
Cobalt private analytics	shipped config (assets/cobalt_config.pb)	RAPPOR-style noised counters ("one_day_actives", histograms) rather than raw events.	Ships but is a rounding error next to Clearcut; included for completeness.
SNET / SafetyNet device-state telemetry	safebrowsing.google.com/safebrowsing/uploads/android	Its own device UUID (<code>snets_shared_uuid</code> , random, resettable by a remote flag <code>shared_uuid_reset_counter</code>); a 2 % sample of launcher apps with APK digests; DEX metadata for installed apps; root/setuid files; SELinux state; system event log; logcat; app-ops; mount options; carrier info.	Runs without a Google account. Not covered by Play Protect's upload switch, which governs only the APK upload. Logcat and the event log are denied to an unprivileged app. SEEN . INFERRED (what the sandbox blocks)

Three pieces of code

```
defpackage/jwgm.java:12
    return (String) a.h(1, "batch_server_url",
        "https://play.googleapis.com/log/batch").lw();

com/google/android/gms/clearcut/internal/PlayLoggerContext.java:67
    "PlayLoggerContext[package=" + a + ",packageVersionCode=" + b +
    ",logSource=" + c + ",logSourceName=" + f + ",uploadAccount=" + d +
    ",logAndroidId=" + e + ",isAnonymous=" + g + ",qosTier=" + h +
    ",appMobilespecId=" + i + ",scrubMccMnc=" + j + "piiLevelset=" + k + ']';
```

The second block shows the telemetry contract in one line: which app, which build, which log source, **which account**, and **whether the Android ID is attached**. `scrubMccMnc` and `piiLevelset` show that Google explicitly classifies how personal each event is, so the design is deliberate rather than an accident of logging.

```
defpackage/bjtv.java:24
    https://%s/checkin with SystemProperties "gms.checkin.sw_domain"
                        default "android.clients.google.com"

defpackage/bjtv.java:131
    checkPermission("android.permission.READ_PHONE_STATE", ...) == 0

defpackage/bjtv.java:340,347
    getSimSerialNumber() · getSubscriberId()
```

Check-in deliberately reaches for IMSI and SIM serial, the two identifiers that survive a factory reset on the carrier side. On GrapheneOS they are gated behind a runtime permission the user can refuse. On stock Android they arrive through `READ_PRIVILEGED_PHONE_STATE` without a prompt.

```
com/google/android/gms/security/verifier/ApkUploadChimeraService.java:20
    "com.google.android.gms.security.verifyapps.UPLoAD_APK"
InternalApkUploadChimeraService:
    package_name=%s, version_code=%d, state=%d    →    full APK upload
```

The Play Protect verifier queues the APK itself, not a hash. That is defensible as a security feature and indefensible as a privacy default, so the recommendations give it its own switch.

The module system is a second distribution channel. Thirteen module APKs were compiled on the audited device from `app_chimera/m/`; none of them ships in the GrapheneOS repository. Nothing in the APK you install tells you which of them a given phone will receive: the list comes from a server-issued configuration file (`CHMR` magic) in the app's own directory. Section 11 describes what these modules contain. SEEN (device `logcat`; `affx.java:35`)

Only three native libraries open sockets. `libbetocore` (Nearby, Rust/QUIC), `libjingle_peerconnection_so` (WebRTC, raw UDP) and `libopenscreen_jni` (Cast, mDNS + UDP). None of the 39 libraries contains a native HTTP client, a hard-coded IP address or native certificate pinning. A hostname filter therefore sees hostnames and not paths, as section 10.1 says, but the stronger worry that native code hides arbitrary HTTP traffic is not supported.

SEEN

What GrapheneOS can and cannot pass

Play Integrity gives an app three levels of verdict. Where GrapheneOS stands on each:

LEVEL	ON GRAPHENEOS WITH SANDBOXED PLAY SERVICES
Strong integrity (hardware attestation signed by Google)	GrapheneOS cannot produce it, so apps that require it stay blocked, and no setting or restriction of Google's code changes that. Strong integrity is built on Android's hardware key attestation, which GrapheneOS fully supports but which the Play Integrity API ties to Google-certified operating systems; the GrapheneOS attestation compatibility guide explains how apps could accept GrapheneOS's keys instead. Which banking apps are affected is tracked by the community in the PrivSec compatibility list, which GrapheneOS does not vouch for. SEEN (the documents) · UNTESTED (by me)
Device integrity and basic integrity	As far as I know, these usually pass with Play services running in the GrapheneOS sandbox, and many apps check only these levels. The GrapheneOS usage guide says it passes <code>basicIntegrity</code> but is not Google-certified, and it describes the older SafetyNet service. INFERRED · UNTESTED (by me)

The consequence for anyone tempted to strip Play services down: DroidGuard produces the inputs for these verdicts. Without it, even the levels that pass today cannot be evaluated, so an app that works now could begin to report an integrity error. Anything that disables DroidGuard or the downloaded code it needs (see item 8 in section 09) puts that at risk. INFERRED

Does Play services use Play Integrity itself?

Yes, in two senses.

- **It provides it.** The build contains the Integrity API module (`play/integrity/api/service/` : `IntegrityApiPersistentService` , a display-listener service and a location-trust service), a separate AutoProtect module whose `AipTelemetryIntentOperation` logs events with `EventType` , `PackageName` , `VersionCode` and `Payload` , and three `droidguard.*` services, two of them exported. SEEN
- **It consumes attestation for its own features.** About 25 files outside the DroidGuard package call the DroidGuard client. Concrete cases: the ads module schedules `ads.fetch_integrity_token.one_time` and

`.periodic`, stores a "device integrity token" and checks its signature and public key; the Firebase phone number verification flow `firebase_pnv_preflight` calls `getIntegrityToken` and takes a DroidGuard snapshot (`ckpu.java:51,62`); Find My Device crowdsourced location reports request DroidGuard results (`LocationReportUploadIntentOperation.java:208`, flow `finder_crowdsourcedReport`); device setup passes a `droidguardResult`; and a PoTokens module has an `IntegrityTokenRefreshTaskService`. SEEN

I did not trace whether each of these goes through the public Play Integrity API or straight to DroidGuard, the engine underneath it; the class names do not say. On GrapheneOS the verdicts are weaker than on a certified device, so features that insist on a strong result may fail or fall back. INFERRED

06

When it happens

"When" is where most privacy discussions go wrong. GMS runs as a scheduler with a dozen triggers, and most of them are ordinary Android events.

first run	Check-in. Registers the device, mints logging ID and security token; can refresh the Android ID on token loss.
every 12 h	Periodic check-in. Default <code>checkin_interval</code> = 43,200 s with 10,800 s flex; the Chimera config uses 43,201 s. SEEN
every 24 h	Phenotype sync. <code>Phenotype__checkin_interval_in_seconds</code> = 86,400. Pulls flags, commits experiment tokens. SEEN
on event	Clearcut. Events are queued and uploaded in batches by the playlog uploader; failures back off. Boot, module update and forced upload are explicit triggers. SEEN
on change	Re-check-in. SIM state change, connectivity change, Google account change, GServices change, package added/replaced, user present. SEEN <code>bjtj.java</code> , <code>CheckinServiceTriggerReceiver</code>
always	FCM socket. A persistent connection to <code>mtalk.google.com</code> (fallbacks alt1 to alt8, :443 and :5228) keeps the device reachable and reveals liveness. SEEN
per request	Measurement and ads. Analytics flushes when an app logs an event; the ads binder answers whenever an app asks for the advertising ID. SEEN

What this means in practice. A phone that has been idle for a day still talks to Google at least ~2× for check-in, ~1× for config, continuously for push, and whenever any Play-integrated app does anything. The network toggle therefore acts as a hard offline switch for the entire Play ecosystem, well beyond blocking ads. INFERRED

Where it goes

778 distinct hostnames appear as literals in this build. They cluster on a handful of Google domains. The counts are literal occurrences in the APK, not traffic volume.

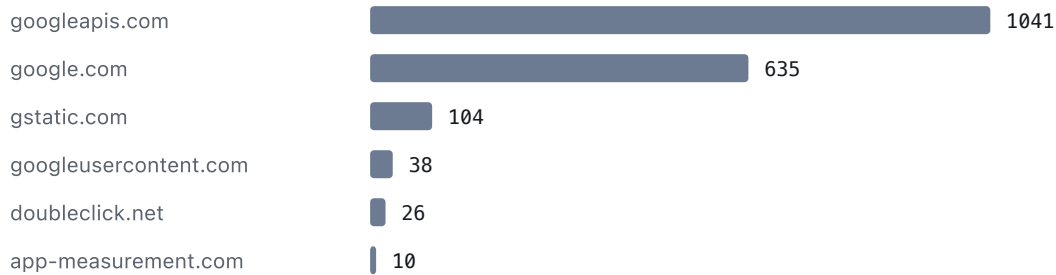


Figure 4: literal hostnames by registrable Google domain (top 6 of 155)

HOST	ROLE
www.googleapis.com	REST front door for most Google APIs (auth, people, games, ...)
support.google.com	help pages opened in WebView
accounts.google.com	sign-in, account chooser, OAuth
play.google.com	store links, Play Points, install referrer
android.googleapis.com	default check-in endpoint
android.clients.google.com	check-in, auth, FRP validation, tap-to-pay
app-measurement.com	Firebase / Google Analytics event upload
play.googleapis.com	Clearcut log upload (/log, /log/batch)
mtalk.google.com	FCM push socket (persistent connection)
gcm.googleapis.com	FCM registration / status
nearby.googleapis.com	Nearby Share / Fast Pair backend
maps.googleapis.com	Places, static maps, geocoding
people-pa.googleapis.com	Contacts & profile sync
feedback-pa.googleapis.com	in-app feedback, screenshots
firebaseinstallations.googleapis.com	Firebase installation IDs
safebrowsing.googleapis.com	Safe Browsing threat-list lookups (over OHTTP)
ohttp-relay-safebrowsing-android.google.fastly-edge.com	Oblivious HTTP relay in front of Safe Browsing

HOST	ROLE
cryptauthvault.googleapis.com	Cross-device key vault (phone-as-security-key)
findmydevice-pa.googleapis.com	Find My Device
androidpay-users-pa.googleapis.com	Google Wallet
googleads.g.doubleclick.net	ad requests / conversion pings

Two caveats apply. First, many endpoints are assembled at runtime from config (`"https://" + authority + path`), so a literal list undercounts. Second, a hostname in the APK is not a packet: some are fallbacks, some are staged (`*.sandbox.googleapis.com`), and some belong to features the user never touches.

A third caveat: one subsystem deliberately hides its destination. Safe Browsing uses Oblivious HTTP, so the SNI a filter sees is a Fastly edge host (`ohhttp-relay-safebrowsing-android.google.fastly-edge.com`), not a Google hostname, and SafetyNet and Play Protect uploads go to `safebrowsing.google.com/safebrowsing/uploads/android`. That is a real privacy property (the relay sees your IP but not the query; Google sees the query but not your IP), and it also means a blocklist built from this table is incomplete. [SEEN](#) `fykj.java:27-28`, `fykf.java:21-22`, `fysz.java:27`

```
www.googleapis.com/auth/... (scopes requested by GMS modules)
webhistory · plus.me · userinfo.email · userinfo.profile · peopleapi.readwrite
cryptauth · tapandpay · games · fitness · drive · photos.image.readonly
android_device_manager · kid.management · subscriptions · userlocation.reporting
```

The OAuth scope list covers history, profile, contacts, fitness, payments, photos, location reporting and family management. Each is reachable only when a matching account is signed in, which is the situation most users are in.

08

The GrapheneOS delta

This section covers what GMS still does on GrapheneOS.

CAPABILITY	STOCK AOSP	GRAPHENEOS SANDBOX
Read IMEI / IMSI / SIM serial	yes, silently (privileged)	only with user-granted READ_PHONE_STATE
Read MAC address, nearby APs	yes (LOCAL_MAC_ADDRESS)	denied; randomized MAC per network
Read logs, dropbox, app-op stats	yes	denied
Usage access, overlay, write settings	yes	only via Settings special access
Write secure settings	yes	denied
See all installed apps	yes	yes , because QUERY_ALL_PACKAGES is a normal permission
Contacts / SMS / call log	yes	only if user grants the dangerous permission

CAPABILITY	STOCK AOSP	GRAPHENEOS SANDBOX
Location (fine/background)	yes	only if user grants; revocable per app
Background execution	unrestricted	standard app standby; user can restrict
Network access	always	revocable , because INTERNET is dangerous on GrapheneOS
Kernel / hidden APIs	yes	blocked by GmsCompat shims or fails gracefully

What the sandbox removes

- 132 signature/privileged permissions, permanently.
- System-level hooks do not apply: privileged phone state, secure settings and other users' storage stay out of reach.
- The "trusted system app" identity used for Play Integrity and some account flows.
- Silent background privileges; GMS competes with other apps for resources.

What it does not remove

- The check-in identity ladder (SSAID, logging ID, token).
- Clearcut, Measurement, Phenotype, FCM, ads and the account plumbing.
- The installed-app list (normal permission).
- Anything you grant at runtime: contacts, phone, location, SMS.

Summary. Sandboxed Play turns Play services from a system component with permanent, silent access into an app with the same rights as any other app, plus a very long list of things it would like to have. That does not make it private, but it makes it *consent-gated and revocable*, which is a different and more useful property.

Dynamic code loading is a per-app toggle, and for Play services it is allowed by default.

GrapheneOS has three exploit-protection toggles for user-installed apps: dynamic code loading from memory, dynamic code loading from storage, and WebView JIT. When storage loading is refused it shows the blocked path. Play services is a user-installed app, so *none of this is blocked by default*: the GrapheneOS release notes say GrapheneOS made an exception for apps that depend on Play services, because the older Dynamite module system is not yet fully replaced by split APKs. Turning off storage loading stops two things: the DroidGuard engine (`DexClassLoader` over `app_dg_cache`) and module APKs loaded as files from `app_chimera` . It does **not** stop the install-time splits, which are ordinary platform splits. The GrapheneOS tracker has a crash report (issue #8714) from Play services failing with `SecurityException: DCL via storage` while loading an optional module from `app_chimera` on a phone where the restriction was on. [SEEN](#) (code, release notes, issue) ·

UNTESTED (whether Play services degrades gracefully)

Three user profiles

PROFILE	WHAT GOOGLE CAN ASSOCIATE	RISK
No Google account, no Google apps	Device fingerprint + check-in ID + SSAID, rotating Android ID on demand; coarse network metadata (IP, timing).	LOW pseudonymous device, not a person
Google account signed in, nothing else granted	All of the above joined to a real identity; app usage via Clearcut/Measurement; push token; purchase history in Play.	MEDIUM account-linked device graph
Account + Contacts + Phone + Location granted	Contact graph, IMSI/SIM, location history signals, installed apps, Play Protect uploads.	HIGH effectively stock-Android visibility

09

Recommendations

Ordered by what each step buys you per unit of inconvenience. Everything in this section is a setting or feature that exists today; nothing needs a patch. Tags show what a claim rests on: **SEEN** in the APK or in the GrapheneOS usage guide, **INFERRED** from what was seen, **UNTESTED** not tried on a live device.

Where to click

Menu names below were checked on a GrapheneOS device (build 2026102101, Android 17). Your screens may differ slightly on other builds. Most Play settings sit behind one hub: **Settings > Apps > Sandboxed Google Play**. It lists *Play services app info*, *Play services special permissions*, *Play Store app info*, *Android Auto*, *Google Settings* and, under *Geolocation*, the toggle *Reroute location requests to the OS*. **SEEN**

GOAL	PATH	SET IT TO
Take Location away from Play services	Sandboxed Google Play > Play services app info > Permissions > Location	<i>Don't allow</i> . Leave <i>Reroute location requests to the OS</i> on.
Cut Play services off the network	Play services app info > Permissions > Network	<i>Don't allow</i> makes it fully offline, but breaks push; I do not recommend it for most people (item 3)
Hide SIM identifiers from check-in	Play services app info > Permissions > Phone	<i>Don't allow</i>
Block SMS access	Play services app info > Permissions > SMS	<i>Don't allow</i>
Nearby scanning	Play services app info > Permissions > Nearby devices	<i>Don't allow</i> unless you use Fast Pair or Quick Share
Carrier authentication with device IDs	Sandboxed Google Play > Play services special permissions > <i>Allow ICC authentication with device identifiers</i>	Off, unless your carrier needs it for RCS

GOAL	PATH	SET IT TO
Contacts without giving them	Settings > Apps > (app) > Permissions > Contacts and accounts > <i>Contact Scopes</i>	Enable it instead of <i>Allow</i>
Advertising ID	Sandboxed Google Play > Google Settings > All services > Privacy & security > Ads	Tap <i>Delete advertising ID</i> (or <i>Reset advertising ID</i>)
Usage reporting	Google Settings > All services > Privacy & security > Usage & diagnostics	Switch off
Phone number verification	Google Settings > All services > Privacy & security > Phone number verification	Review; it relates to the <code>firebase_pnv_preflight</code> flow in section 05
Play Protect APK upload	Play Store > profile picture (top right) > Play Protect > Settings (top right)	Switch off <i>Improve harmful app detection</i> ("Send unknown apps to Google"). <i>Scan apps with Play Protect</i> is the scan itself. The code has two separate switches, <code>package_verifier_user_consent</code> for the scan and <code>upload_apk_enable</code> for the upload. Only the upload path was traced; the scan very likely sends at least app identifiers to Google. INFERRED
Downloaded code (DroidGuard engine, modules)	Play services app info > Exploit protection > <i>Dynamic code loading from storage</i>	Off only if you do not need Play Integrity, QR scanning or passkeys (item 8). UNTESTED
Own profile	Settings > Security & privacy > Privacy > Private space	Optional: install Play there only if few apps need it. Push stops while the space is locked.
Push stays alive	Play services app info > App battery usage	<i>Unrestricted</i> (needed for push)

Permissions with more than two choices: *Location* offers *Allow all the time*, *Allow only while using the app*, *Ask every time or allow when you share* and *Don't allow*, plus a *Use precise location* toggle. *Network*, *Phone*, *SMS* and *Nearby devices* are just *Allow* or *Don't allow*.

1. Decide what Play services is for

If you need banking apps, push notifications and Play purchases, keep it, and accept that check-in and Clearcut will run. If you do not, uninstall it. There is no configuration of GMS that is both functional and telemetry-free; the two are the same code path.

2. Decide which profile gets Play

The GrapheneOS guide says sandboxed Play can be installed in a secondary user, a work profile or Private Space, and only apps in that profile can use it **SEEN**. Android keeps package data, the Android ID and the package list per user, so what GMS can read is limited to that profile. **INFERRED** That helps if few apps need Play and you want everything else outside Google. It does not stop check-in: Play services in a running profile still contact Google. A Private Space stops when it is locked, so push arrives only while it is unlocked. **INFERRED** The guide's own advice is the simplest route: use the Owner profile only, or a second profile only if you want to choose which apps use Play

SEEN. If push is a reason you want Play, install it in the profile you use every day. A separate profile costs switching or unlocking in daily use.

3. The network toggle: it works, and I would not recommend it

GrapheneOS makes **INTERNET** a revocable permission, listed as dangerous in this analysis. The switch is *Sandboxed Google Play > Play services app info > Permissions > Network > Don't allow*. Turning off network for Play services stops check-in, Clearcut, Measurement, FCM and ads in one move. The cost is that anything using Play services stalls, so this is a per-profile decision, not a per-app tweak. **INFERRED**

Why I would not do this. Play services without a network are of little use. A few things keep working (for example scanning a QR code with a module that is already on the phone), but the main reason most people want Play services is push notifications, and push travels over the Google connection you just cut. Banking apps, messengers and couriers stop notifying; Play Integrity and sign-in fail; maps and anything else that talks to Google break. If you accept all that, you are better off not installing Play services at all. The realistic ways to lower the cost are the permission and profile settings above, not this switch.

4. Deny the permissions that buy Google the most

- **Phone.** This is the IMSI and SIM-serial path in check-in, and the code gates it on exactly this permission (`bjtv.java:131,340,347`). Deny it unless your carrier needs it (RCS and TS.43 are the cases the guide names). **SEEN**
- **Contacts.** Use **Contact Scopes** instead of the permission: the app believes it has access and gets none, and write access is blocked. **SEEN** (usage guide)
- **SMS / Call log.** Deny unless used.
- **Location.** Keep the default re-routing to the GrapheneOS geolocation service; then Play services needs no location permission for apps that ask it. **SEEN** (usage guide) The next section explains what that means for Google Maps.
- **Physical activity, Nearby devices.** Grant only for features you use (location sharing, Fast Pair).

Does Play services need Location? (and what about Google Maps)

No, not by default. The GrapheneOS guide says that apps using Google Play geolocation are redirected to GrapheneOS's own implementation on top of the standard OS location service, and that you do not need to grant any permission to Play services or change any settings for location to work in those apps. **SEEN** (usage guide) The rerouting is on in the toggle *Sandboxed Google Play > Geolocation > Reroute location requests to the OS*, and the same screen carries a *Potential issues > Geolocation* entry that warns when requests are rerouted but Play services still holds the Location permission. **SEEN** (on a device)

Google Maps keeps its own Location permission as any app does. When Maps asks the Play location API for a position, the request is answered by the OS service, so Maps should work with Location removed from Play services. **INFERRED** GrapheneOS staff and users report that Maps runs even without Play services, and some users report a grey map, so test it. No official documentation covers this. What stops working without it:

- **Location sharing in Google Maps** runs inside Play services and needs *Allow all the time* for Location plus the Physical activity permission. **SEEN** (usage guide)
- **Google's own network location service** (the one that sends Wi-Fi and cell data to Google) only works if you turn rerouting off, grant Location *Allow all the time* and Nearby devices, and opt in under *Google Location Accuracy*. The guide does not recommend it. **SEEN** (usage guide)

- **Fast Pair and similar Nearby features** use Bluetooth scanning and are tied to the Nearby devices permission.

INFERRED

The price of rerouting is power: it uses more battery than Google's service, unless you turn on GrapheneOS's opt-in network location at *Settings > Location > Location services > Network location* (Wi-Fi scanning at *Settings > Location > Location services > Wi-Fi scanning* improves it). SEEN (usage guide) On a cold start, Maps may be slower to get a first position indoors until the satellite signal arrives. UNTESTED

5. Delete the advertising ID, do not just reset it

Reset breaks the join to past ad profiles but creates a new one. Deleting it, where the Google settings offer it (the manifest contains `AdsIdentityCollapseSettingsDeleteAdIdActivity`), hands apps a zeroed value. It does not touch the check-in identity. SEEN (component) · UNTESTED (behaviour)

6. Switch off usage reporting

The build ships `usagereporting.settings.UsageReportingActivity` and a `UsageReportingIntentService`. Turn "Usage & diagnostics" off in the Google settings screen of the sandboxed Play app. It is one of the few switches Google puts in front of the Clearcut stream. SEEN (components) · INFERRED (that the toggle gates uploads)

7. Turn off Play Protect uploads if you accept the trade-off

Verify Apps queues the APK itself: `ApkUploadChimeraService` forwards an `UPLOAD_APK` intent to `InternalApkUploadChimeraService`, which keeps an upload queue and re-runs on network change and package change. If you sideload, that is a meaningful disclosure; if you only install from Play, the value is lower than it looks.

SEEN

8. If you do not need Play Integrity, block dynamic code loading from storage

Settings > Apps > Play services > App info > Exploit protection > Dynamic code loading from storage. This is the only user-facing control over the two code paths that the server decides: the DroidGuard engine and the downloaded module APKs. Expect Play Integrity verdicts, Play Protect scans, and module-backed features (QR scanning, text recognition, passkeys) to stop working. This includes the verdicts that pass today: DroidGuard generates the inputs for basic and device integrity, so an app that works now could start reporting an integrity error. Strong integrity will not pass either way (section 05). SEEN (the paths) · UNTESTED (the breakage and the exact menu path)

9. Do not sign in unless a feature needs it

The account is rung 04 of the ladder. Without it the device is a pseudonym; with it, Clearcut can attach `uploadAccount` to events.

10. Prefer apps that do not need Play services

Every app that integrates Firebase, Maps, FCM or AdMob is another stream into the same account. Avoiding such apps directly reduces the number of subsystems above that have anything to say about you.

What not to do. Do not assume that "no Google account" means "anonymous", and do not assume that clearing Play services data resets everything: it resets the check-in ID, but the SSAID survives until factory reset, and the IP and timing metadata was never yours to clear. Do not take away background execution to save battery if you want push; the guide recommends a battery exception for exactly that reason.

10

Beyond settings: blocking, patching

Settings stop at what Google chose to expose. This section goes further: filtering the network, modifying the APK, and patching GrapheneOS itself. These are engineering proposals built on the component and host inventories in this report. **None of them was tried on a device.** Each is rated by privacy gain, breakage, effort and upkeep so you can compare them.

Sections 10.2 and 10.3 are not recommended. They are here to answer the question "would this make sense?" and to think through, in theory, what it could bring. They are a thought experiment, not practical advice. In practice both are a much bigger pain than they look: you re-sign everything or build your own OS, you carry certificate pins or a verified-boot key of your own, and you redo the work after every release. The gain is small next to that cost.

10.1 Filter the network from outside

No patch, fully reversible, and the APK stays Google-signed. Use Private DNS with a filtering resolver, or a firewall VPN that is installed only in the Play profile (Android VPNs are per user, so the Owner profile is unaffected **INFERRED**). A DNS or SNI filter sees hostnames only, never paths.

HOST	SUBSYSTEM	BLOCK?	CONSEQUENCE
app-measurement.com	Firebase analytics upload SEEN	YES	Apps' analytics stops leaving the device. Apps themselves keep working.
googleads.g.doubleclick.net, pagead2.googlesyndication.com	Ads SEEN	YES	AdMob ads fail to load.
feedback-pa.googleapis.com	In-app feedback	YES	Feedback and bug reports are not sent.
people-pa.googleapis.com	Contacts and profile sync	IF UNUSED	Contact sync and lookups fail.
play.googleapis.com	Clearcut <code>/log/batch</code> SEEN , but also other Play APIs	SHARED HOST	Blocking the host stops Clearcut and also unrelated Play features. Cutting only <code>/log</code> needs path-level inspection.
cloudconfig.googleapis.com	Phenotype flags	RISKY	Flags freeze at stale values; modules may misbehave. INFERRED

HOST	SUBSYSTEM	BLOCK?	CONSEQUENCE
firebaseinstallations. googleapis.com	Firebase installation IDs	NO	Firebase-based apps need it to obtain their FCM tokens. INFERRED
android.clients.google.com	Check-in, auth, FRP	NO	Push and sign-in depend on it. It is also the identity endpoint.
mtalk.google.com, gcm. googleapis.com	FCM socket	NO	This is push. Keep.
ohhttp-relay-safebrowsing- android.google.fastly-edge. com	Safe Browsing, relayed SEEN	NO	Blocking it stops Safe Browsing lookups, which may be what you want, but it also shows the problem: the host you would have blocked (safebrowsing.googleapis.com) is not the host on the wire.

Limits. The build ships a Cronet split (`split_CronetDynamite_installtime.apk` , 590 classes, `libcronet.155.0.8059.30.so`). Cronet is a native stack, but Java drives it through `org.chromium.net.CronetEngine` , so a hook at that API would see the requests and a hook at `URLConnection` would not. What a hostname filter genuinely cannot see is UDP: WebRTC (ICE/STUN), Cast (mDNS + UDP) and Nearby (WebTransport/QUIC in `libbrotocore`). No hard-coded IPs were found in native code. Safe Browsing deliberately hides its destination behind an Oblivious HTTP relay (next row). **SEEN**

How to find out whether it works. Run PCAPdroid (an on-device VPN capture that records SNI) in the Play profile for 24 hours as a baseline, apply the filter, capture another 24 hours, and diff the host counts. That is the missing observation this report could not make.

10.2 Modify the APK (not recommended, theoretical)

Not recommended. This section is a theoretical discussion of whether repackaging Play services would make sense and what it could bring. It is not practical advice; the effort and upkeep are much bigger than they look from the list below.

Is it possible at all? Technically yes: it is an ordinary signed APK, and nothing found in this analysis stops you from unpacking, editing and signing it again (I did not test an install, and I did not look for a self-integrity check inside the app). The practical question is whether the result still works, and that is where it fails. The barriers, from this analysis:

BARRIER	WHY IT MATTERS
Signing chain	The manifest declares <code>sharedUserId="com.google.uid.shared"</code> and <code>isolatedSplits="true"</code> SEEN . A re-signed GMS needs the Play Store (and GSF, if present) re-signed with the same key, and all 11 splits too. The Google-signed install must be removed first, which deletes its data. The GrapheneOS App Store can then no longer update it. INFERRED

BARRIER	WHY IT MATTERS
Pinned certificates	Google's own code pins the package name together with the certificate SHA-1 of Play services and the Play Store (<code>gtub.java:8</code> , <code>DataService\$1.java:105</code> , used in device setup). Components that check these pins will reject a build signed with your key. SEEN (the pins) · INFERRED (the rejection)
Trust in Google's key	The build contains a <code>GoogleCertificates</code> split, and 17 files read package signatures or signing certificates (<code>signingInfo</code> in 38 files, <code>.signatures</code> in 42, <code>Signature[]</code> in 57). Other Google components and Play Integrity can be expected to treat a different signer as not Google. SEEN (the checks) · INFERRED (the consequences)
Compatibility layer	No file in the decompiled code mentions GrapheneOS or GmsCompat, which confirms the app is stock SEEN . The OS-side layer has to recognise it; I did not check whether it keys on the signer. If it does, you also need an OS patch. UNTESTED
Remote flags	Behaviour defaults are read through a flag lookup, for example the Clearcut URL in <code>jwgm.java:12</code> (<code>a.h(1, "batch_server_url", ...)</code>). Phenotype can overwrite defaults at the next sync, so editing a default is not enough; patch the call site or stop the sync. SEEN
Obfuscation churn	Code classes are renamed (<code>jwgm</code> , <code>bjtv</code> , ...) and the names are specific to a build, so a small patch must be re-found every release. Public manifest names are stable. INFERRED
Integrity	A modified Play services is likely to lower the attestation verdicts banking apps check, which are already limited on GrapheneOS. INFERRED

Options, from least to most invasive

- 1. Manifest-only patch.** Set `android:enabled="false"` on the telemetry and ads components (for example `clearcut.uploader.QosUploaderService` , the `measurement.*` brokers, `usagereporting.service.UsageReportingIntentService` , `ads.AdRequestBrokerService`) with a binary manifest editor, leave DEX untouched, zipalign, then sign every split with your own key. This avoids re-assembling a 195 MB APK and is the only option whose patch survives version churn, because the names are public. Leave `checkin.CheckinService` , the FCM and account services and `droidguard.*` alone: push, sign-in and attestation depend on them. Play services also toggles its own components at runtime (`setComponentEnabledSetting` in `brtr.java:48` , `setComponentEnabledSettings` in `gwyn.java:63` , plus two more callers), so a disabled component may be switched back on and an update may reset the list. SEEN (the names and the calls) · UNTESTED
- 2. Smali patch of egress methods.** Stub the uploader call sites. Works until the next release renames them. Not recommended.
- 3. Flag override.** Ship your own Phenotype values. This needs a server or an intercept and is a larger project than the problem.
- 4. A different implementation (microG-style).** A separate project with a different trust model that gives up the Google binaries altogether. Out of scope here.

Will Google's servers accept a modified build? (attestation)

This is the real risk of repackaging. **Nobody can promise it works, and this analysis cannot settle it.** What is known and what is not:

- Known.** Play services download a DroidGuard engine (an APK, section 11) from `gstatic.com/droidguard/` , feed it a per-request program, run it inside the Play services process, and send the result to Google. The bytecode and the server-side judgement are Google-defined and can change at any time. Play services also consume these results themselves (ads token, phone number verification, Find My Device reports). SEEN

- **Not known.** Whether the per-request program measures the host app's own signature or code. The engine was obtained and read (section 11), but the program it runs arrives with each request and was not captured.

UNKNOWN

- **Reasonable worry.** A system built to detect tampered environments has an obvious reason to look at the app it runs in, so a re-signed build is more likely to score worse than a stock one. This is a guess from the purpose of the mechanism, not an observation.

INFERRED

What a bad verdict would likely cost, from least to most serious:

AREA	LIKELY EFFECT
Apps that call Play Integrity	The device verdict reaches the app from Google's servers. On GrapheneOS only basic and device integrity are expected to pass today, as section 05 notes, so a modified build can mainly lose those. Apps that already demand device or strong integrity fail either way. INFERRED
Play services' own features	Ads token, phone number verification and crowdsourced Find My Device reports may fail or be skipped. These are the least important. INFERRED
Sign-in and account	Server-side risk scoring could ask for extra verification or limit the device. I know of no evidence either way. Use a throwaway account for any experiment. UNKNOWN
Play Store	It would also be re-signed. Whether it still downloads, updates and licenses apps is untested. UNTESTED

How to find out cheaply. In a disposable profile with a throwaway account, install the stock sandboxed Play and run an integrity-checker app; note the verdicts, push delivery, sign-in and one Play-dependent app. Then replace it with the re-signed build and repeat. Compare the two runs. Do not block `gstatic.com` during the test: it serves the DroidGuard bytecode.

The decision rule. Every option that leaves Google's signature intact (sections 09, 10.1 and 10.3 A) carries no extra attestation risk from the app side, because Google sees the same signed binary it always did. Repackaging is the only option that changes what Google can see about itself. If attestation matters to you, that is the strongest argument for the OS-level route over the APK route.

If you do this, keep the signing key as you would a release key: the shared UID means the key owner controls what runs with Play services' data. Rebuild automatically from the signed upstream file after every update, and never ship the result to others.

10.3 Patch GrapheneOS (not recommended, theoretical)

Not recommended. As with 10.2, this is a thought experiment about what an OS-level change could bring, not advice to build your own GrapheneOS. You would need your own verified-boot key, lose official over-the-air updates and carry the patches through every release.

The OS is where the signature stays intact and the fix survives GMS updates, but you must build and sign your own GrapheneOS, which means your own verified-boot key and no official over-the-air updates. I did not check which of these ideas already exist upstream; check that, and raise an issue before building anything.

PROPOSAL	WHERE	EFFECT	EFFORT / RISK
A. Per-app domain allowlist ("push-only network")	Network stack, keyed on the app's UID: an eBPF socket hook plus resolver/SNI matching	Allow check-in, auth and FCM; drop Clearcut, Measurement, ads and Phenotype. Unlike a user-space VPN it is on by default and cannot be bypassed by the app.	High. Shared hosts (<code>play.googleapis.com</code>) and ECH limit it to hostnames.
B. Package scopes	Package visibility filter in the package manager	QUERY_ALL_PACKAGES is a normal permission <code>SEEN</code> , so the installed-app list is free today, and 122 decompiled files query packages or intents. Treat GMS as not holding it, in the style of Contact Scopes.	Medium. Play Protect and Nearby lose some visibility.
C. Reset device identity	Settings plus the Android ID store	One action rotates the Android ID for Google's signing key in a profile and clears the check-in files. Today clearing app data resets the check-in ID but not the SSAID <code>INFERRED</code> .	Medium. Forces new push registration and account re-trust.
D. In-process URL filter	Compatibility-layer hooks at the Java HTTP boundary	Path-level rules without breaking TLS.	Medium and fragile: Cronet must be hooked at <code>CronetEngine</code> , and UDP (WebRTC, Cast, Nearby QUIC) is out of reach.

My order of preference is A, then B, then C, with D last because of Cronet.

10.4 Which one to pick

OPTION	PRIVACY GAIN	BREAKAGE	EFFORT	UPKEEP
Settings and permissions (09)	Medium	Low	Minutes	None
Network toggle (not recommended: breaks push)	Full, offline	Everything Play	Seconds	None
DNS / VPN filter (10.1)	Medium to high	Low to medium	Hours	List upkeep
Manifest-patched APK (10.2), theoretical	Medium	Medium to high	Days	Every release
OS patch (10.3), theoretical	High	Low	Weeks	Every OS release

Recommended path. Everyone: section 09, with Play in the profile you use every day, or in a separate one if few apps need it. If you want push without most of the telemetry: add 10.1 in that profile and measure it. Sections 10.2 and 10.3 are theory, not recommendations. If you still want to explore one, 10.3 A is the least bad on paper, because it needs no change to Google's code. Skip APK repackaging, and read its attestation discussion first.

Downloaded code: the modules

The APK in the repository is not all the code Play services runs. Google sends further module APKs to the phone after installation, and these never appear in any public artifact. They cannot be read off a phone without root, so for this report I pulled them from a rooted Android 13 test emulator running Play services 26.41, decompiled them, and checked each file against the SHA-256 that Play recorded for it. They did not come from GrapheneOS, but they are the modules Google delivers to Play services in general, so they show what Play services actually runs.

24	19	0	7
MODULE APKS COLLECTED	DISTINCT MODULE BUILDS	PERMISSIONS THEY REQUEST	MODULE GENERATIONS ON ONE DEVICE

What "dynamic loading" means: there is no single Play services

It is tempting to picture "Play services" as one component that could be audited once and then trusted or distrusted for everybody alike. That is not how it works. The APK is a host. After installation it sends Google a description of the device it runs on, and the answer says which modules this device should have and in which version. The app then downloads those modules (through the Play Store) and updates them by itself, with no action from you and no new Play services release. The request carries the device's Android ID and a token for each signed-in Google account. On the test emulator the module plan changed when the device registered again under a new identity, and one device ended up with seven module generations side by side. [SEEN](#) (section 11, below) ·

[INFERRED](#) (what the server keys on)

Earlier work points the same way. Romain Thomas's analysis of DroidGuard in the SafetyNet service (the predecessor of Play Integrity; SSTIC 2022) found that Play services download the latest DroidGuard if it is not already present, and that DroidGuard then fetches a bytecode program from Google that is "unique for each attestation request". Doug Leith of Trinity College Dublin (2025) observed that the Experiments subsystem of Play services downloads per-app settings with experiment tokens, so that Google can roll an app change out to a small subset of users first, and his list of connections that send the Google Android ID includes `play-fe.googleapis.com/fdfe/moduleDelivery`. GrapheneOS itself had to teach its runtime to load Play services' Dynamite modules from file descriptors (`platform_art` pull request 1). As far as I know, nobody has published a look at the delivered modules themselves, with a per-module list of network destinations and an analysis of what they could target. That is what this report adds.

The consequences for how you read this report:

- **Two phones with the same Play services version can run different code.** The host APK is identical; the modules, their versions and the set of features switched on can differ, depending on the device, its Google account and Google's rollout decisions.
- **An audit is a snapshot.** What I decompiled is the host APK plus the modules one test device was given on one day. It shows what kind of code Google delivers; it cannot prove what any other phone, including yours, runs today. Checking your own phone requires pulling its modules (root) and comparing their hashes.
- **Google can change it without telling anyone.** A module can be replaced or added server-side. The only check on the phone is that Google signed it.

- **The sandbox still applies.** On GrapheneOS, whatever arrives gets only the permissions you gave Play services (or the app that hosts a map or an ad).

How a module arrives

Play services asks the Play Store for a module over a private Binder interface (`BIND_PLAY_MODULE_SERVICE`), the Store downloads the APK, and Play services writes it to `app_chimera/m/<n>/` in its own data directory. Which modules a phone gets is decided by a configuration file the server issues (`current_config.fb` , a FlatBuffer with the magic `CHMR`). On the test emulator that file changed only when the device registered again under a new identity: changing the build fingerprint alone did nothing, and a fresh registration as a Pixel 8 produced a different module plan. SEEN (device logs, file sizes 578,156 → 576,380 bytes) · INFERRED (that the plan is keyed to the device registration and not to the reported hardware)

Module versions are independent of the Play services version. One emulator held seven module generations at once (builds from 23.11 to 26.41), six modules in two versions, and the live configuration referenced only nine of the fifteen files on disk. Old copies are not deleted when a newer one arrives. SEEN

What the modules contain

MODULE	WHAT IT IS	PRIVACY NOTE
<code>MlkitBarcodeUi</code>	QR and barcode scanner screen (CameraX, MediaPipe)	Uses the camera under Play services' Camera permission
<code>VisionBarcode</code>	Barcode decoder (Barhopper v3, three TFLite models)	On-device
<code>MlkitOcrCommon</code> · <code>VisionOcr</code> · <code>VisionOcrDynamite</code>	Text recognition pipeline and models	On-device
<code>TfliteDynamiteDynamite</code>	TensorFlow Lite / LiteRT runtime offered to the other ML modules as a Binder service	Hands raw native pointers to callers
<code>MapsCoreDynamite</code>	The Google Maps SDK renderer that apps with an embedded map use	Map tiles from Google
<code>AdsFdrDynamite</code>	The Google Mobile Ads rendering layer (8,758 classes, <code>ads.nonagon.*</code> , a signal generator)	Ad requests and signals for apps that embed AdMob
<code>IdentityCredentialsPlatform</code>	Android Credential Manager provider: passkeys, credential import and export, age signals; includes a WebAssembly interpreter written in Java	Handles passkeys and sign-in credentials
<code>PlayCloudSearch</code>	gRPC server behind the launcher search box, talking to <code>playgateway-pa.googleapis.com</code>	Search queries typed in the launcher, if bound
<code>Appsearch</code>	Play services' own AppSearch store (the Jetpack <code>androidx.appsearch</code> backend), with built-in indexers for installed apps and for music in MediaStore	Index stays on the device; telemetry names the calling app

All rows SEEN in the decompiled modules.

What they have in common

- **No permissions of their own, and someone else's process.** None of the 24 APKs declares a single `uses-permission`. They use the permissions of whichever process loads them, and "host" means a different thing for

the two module types. *Chimera* modules (the scanner screen, OCR, passkeys, launcher search, Appsearch) run inside Play services, with its UID, every permission you granted it and its network access. Cutting Play services off from the network cuts these off too. *Dynamite* modules (the Maps renderer, the ads renderer, Cronet, Analytics) are loaded by an app that embeds a Google SDK into *that app's* process. A map inside a banking app runs Google's downloaded code with the banking app's permissions and the banking app's network access, so denying Play services network access does not stop this code from reaching the network. The permission screen shows neither. Whether such a module also needs something from Play services before it works (the map tiles, for example) is not something I tested. On the test emulator Play services was a privileged system app with hundreds of grants; under sandboxed Play on GrapheneOS it has only what you granted. [SEEN](#) (manifests; `DynamiteLoaderV2.java`) · [INFERRED](#) (GrapheneOS)

- **No activities or services in their own manifest.** The entry points are stub classes in the main APK, and the module supplies the real code. A QR scan started from a stub screen in Play services runs code from a separately signed APK. [SEEN](#)
- **Ordinary code.** Plain DEX and plain native libraries with normal hardening, loaded the usual way. [SEEN](#)
- **Two signing keys and one source stamp.** `optional` modules are signed with Google's generic Android key, `policy` modules with a separate `gmscore_modules_percy` key. All of them, and the DroidGuard engine, carry the same source-stamp certificate. [SEEN](#)
- **Modules call each other.** The OCR and barcode modules bind the TFLite module and receive raw native function tables from it through Binder. The module boundary is a packaging boundary, not a security boundary. [SEEN](#)

Appsearch: an on-device index of your apps and music

The Appsearch module (9,028 classes, native engine `libicing_icu_with_reversejni.so`) is the storage service behind Jetpack's `androidx.appsearch` for apps that use Play services storage. It is not Android's platform AppSearch. Apart from storing what apps put into it, it ships two indexers of its own:

- **Installed apps.** Every launchable app becomes a `MobileApplication` document with its name, label, icon, signing-certificate SHA-256, launcher class and update time, refreshed on every package change. It is behind the flag `enable_appsindexer`, which defaults to off; Google can turn it on remotely. [SEEN](#) `m58/xs.java:32`, `m58/kgb.java:15`
- **Music.** Albums, artists, playlists and tracks are read from `MediaStore.Audio.Media.EXTERNAL_CONTENT_URI` into searchable documents. This needs the audio permission, which on GrapheneOS you would have to grant to Play services. Off by default. [SEEN](#) `m58/aga.java:64`

The index lives in Play services' private storage without encryption at the app layer, so it is protected by the app sandbox and file-based encryption only. Nothing in the module uploads the index. Its usage statistics do go to Clearcut: seven event types that carry the calling app's package name in clear and the database name as a truncated MD5 hash. The only download is a ~31.6 MB ICU word-segmentation dictionary from `gstatic.com`, fetched on Wi-Fi while charging. Code for a contacts indexer and an app-open-events indexer is named in settings but not present in this build, and a vector (embedding) index exists in the engine but is switched off and ships no model. [SEEN](#) `m58/aig.java:15`, `m58/ahh.java:136-146`, `m58/kgb.java:20` · [UNKNOWN](#) (which Play services features write to it)

The DroidGuard engine, read

The engine (`com.google.ccc.abuse.droidguard`, 459 KB) is a 24-class Java shim and a stripped native interpreter. It is built to resist reading. The exported `initNative` and `closeNative` are decoys that return immediately; `JNI_OnLoad` decrypts seven method names on the stack and registers the real functions. The interpreter is one

flattened 237 KB function with two banks of 256 registers, and the code maps writable-and-executable memory in three places. [SEEN](#) (offsets 0xbcc4, 0xdf97, 0x16070)

What the local code measures without help from the server:

- **A GPU fingerprint ("droidguasso").** It reads the GL vendor, renderer and version, renders two shaders seeded with a 48-byte server challenge, hashes the 32×32 result, and hashes every graphics driver in `/vendor/lib/egl` and `/system/lib/egl`. [SEEN](#)
- **Motion sensors.** Mean, deviation, minimum and maximum for two sensors, plus raw samples, at a rate and duration set by the server. [SEEN](#)
- **Touch events**, up to 32, when the embedding app feeds them. [SEEN](#) · [INFERRED](#) (only in-app use; Play services itself does not feed them)
- **Data from Play services.** The object Play services hands the engine (`RuntimeApi`) returns the Android ID, whether Verify Apps is on, and the Play Protect list of harmful apps with package name, hash and category. [SEEN](#) `droidguard/loader/RuntimeApi.java:39–110`

Beyond that, the engine's reach is defined by the program it receives. The native code contains a raw `syscall` stub and a call-any-function trampoline whose addresses are handed to the interpreter, and it resolves functions by names taken from the program. So the 64 imported symbols do not limit what it can read: a program can ask for any file or system call the Play services process may make. Checks for root, Frida or system properties are absent from the local code, which means they come with the per-request program. The engine has no network code of its own; Play services sends its output. [SEEN](#) (0x6bbc, 0x6b4c) · [UNKNOWN](#) (what a real program asks for)

Three years of the install-time modules

The emulator first ran the old Play services 23.18 that came with its system image, so the eight install-time Dynamite modules exist in both 23.18 and 26.40. The difference shows where Google's measurement code is heading:

- **Measurement (Firebase / Google Analytics)** gained Consent Mode v2 (`ad_user_data` , `ad_personalization`), DMA and TCF handling, an upload pipeline for server-side tagging (data can go to a host the app developer runs), and Privacy Sandbox conversion reporting to `google-analytics.com`. The deferred deep-link request still carries the advertising ID. Neither version reads the App Set ID or the Android ID. [SEEN](#)
- **Ads** went from no Privacy Sandbox code to Topics (off by default) and Attribution Reporting (on by default), and reads an MD5 hash of the Android ID in both versions. There is no Protected Audience code. The downloaded ads renderer (`AdsFdrDynamite`) goes further: if the advertising-ID lookup fails or times out, it puts the *unhashed* Android ID into the ad request as `pdid`. Deleting the advertising ID does not trigger this (the lookup then succeeds with zeros); a lookup that errors out does. It also sends the advertising ID together with the limit-tracking flag, keeps its own `paidv1_id` UUID, and reads the build fingerprint, carrier code, network type and motion sensors. Because it runs inside the app that shows the ad, the Android ID it reads is that app's own value. [SEEN](#) `ads/nonagon/signals/gmscore/c.java:86` , `d.java:31–46`
- **Maps** still sends a bootstrap with a logging ID and the SIM country. [SEEN](#)
- **GoogleCertificates** is a list of trusted app signers, not TLS pins. It grew from 1,635 to 2,017 digests. **Cronet** moved from Chromium 113 to 155. [SEEN](#)
- **The signing key** was rotated: the 23.18 modules are signed with the middle key of the three-key lineage in the 26.40 APK. [SEEN](#)

Who each module talks to

MODULE	RUNS IN	DESTINATIONS	WHAT IT SENDS
AdsFdrDynamite	the app showing the ad	Google ad servers, plus any host the ad response names : impression, click, video and viewability URLs, the creative in a WebView, third-party mediation SDKs	Advertising ID with the limit-tracking flag (sent even when limited), App Set ID, the raw Android ID if the ad ID lookup fails, model and build fingerprint, carrier, location if the app supplies it
MapsCoreDynamite / MapsDynamite	the app showing the map	Google only	Map tiles for the viewport, the app's API key, package and signing-certificate digest; legacy bootstrap with a logging ID and SIM country
PlayCloudSearch	Play services	<code>playgateway-pa.googleapis.com</code> (gRPC), Google image hosts	What you type into the launcher search, locale, an OAuth token. Every call is off by default and switched on by server flags
DynamiteModulesC	the calling app	Google Analytics, Tag Manager, Firebase; any host the app developer puts into a Tag Manager container or Firebase config	Analytics client ID and advertising ID in Analytics hits; whatever the developer's container defines
MeasurementDynamite	the calling app	<code>app-measurement.com</code> via Play services, <code>googleadservices.com</code> ; in 26.40 also a developer-run server-side tagging host	App events, app instance ID, the advertising ID in the deferred deep-link request
CronetDynamite	the calling app	whatever the caller asks for	Default User-Agent names the app, its version, the phone model and build
Appsearch	Play services	<code>gstatic.com</code> (dictionary download), Clearcut	Usage statistics naming the calling app
ML and vision modules, TFLite, passkeys, DynamiteModulesA	Play services or the calling app	None of their own; Clearcut through Play services	Usage counters; the vision logger allows the Android ID. Uploaded only with <i>Usage & diagnostics</i> on
DroidGuard engine	Play services	None of its own; Play services sends the result	Whatever the per-request program collects, including the Android ID handed over by Play services
GoogleCertificates, DynamiteLoader	n/a	None	Nothing

All rows SEEN in the decompiled modules, with file and line in my working notes (not published). Every connection also reveals the IP address. Which third-party hosts real ad responses name cannot be known without capturing live traffic. UNKNOWN

Two of these deserve a sentence each. The ads module calls every address the ad server hands it (to count an impression or a click, for example), follows redirects and checks nothing about the host, so an ad shown in an app can make your phone contact any number of advertisers and measurement vendors. SEEN

`ads/internal/util/client/s.java:88-131` And the map and ads modules run inside the app that uses them, so the location, network and identifiers they see are the ones that app was given.

Can Google send different code to one person?

Technically, yes. Code reaches the phone without a Play services update by five paths: Chimera modules fetched through the Play Store, on-demand Dynamite modules (same path), the DroidGuard engine, the DroidGuard program sent with each check, and dormant code that a remote flag switches on. SEEN

- **The module plan is a remote flag.** Which modules a phone should have arrives as the Phenotype flag `module_set_list` (or in the check-in response). The Phenotype request carries the Android ID, Google's pseudonymous device IDs, the build profile and an OAuth token for each signed-in account, so the server can answer one device or one account differently from everyone else. SEEN `bkaq.java:26`, `fvdd.java:70-77`, `fvvt.java:9` · INFERRED (per-device answers)
- **The download is tied to the device.** The Play Store asks `fdfe/moduleDelivery` for the module under the device's Play account and receives a URL, size and SHA-256 for each module. SEEN (Play Store 53.5) · INFERRED (that the device ID header is attached to this call)
- **Dormant code can be switched on per account.** The same flag channel controls features that ship disabled, such as Appsearch's installed-apps indexer. INFERRED
- **What stops strangers.** Before a module loads, Play services verifies its APK signature, requires exactly one signer and checks that signer against Google's built-in list. The DroidGuard engine must carry one pinned certificate, and its per-request program must be signed with a built-in RSA key. Controlling the network or the configuration is not enough to inject code; only Google's own signing keys produce code that runs. SEEN `afho.java:1464-1481`, `bysr.java:14-31`, `bysq.java:53`

So the mechanism allows Google to deliver a signed module or program to a single device registration or account without anyone else receiving it, and without an update anyone could inspect. Nothing in the code shows that this is done; that is server policy and invisible from the APK. UNKNOWN The DroidGuard request itself identifies only the build profile, though the engine can read the Android ID once it runs. SEEN

On GrapheneOS such code is still bound by the sandbox. It gets the permissions you granted to Play services (or to the app hosting a map or an ad) and nothing more. Privileged phone state, logs and other users' data stay out of reach. Turning off *dynamic code loading from storage* for Play services blocks downloaded modules and the DroidGuard engine; it does not block code already inside the installed APK that a flag switches on. INFERRED · UNTESTED (the toggle's effect on Play services)

What could not be collected

Ten modules named in the module plan never arrived on the test device: AmbientContext, AppsearchDynamite, BetterTogether, BrellaDynamite (federated learning), CameraLowLight, Dck (digital car key), Fastpair, HealthTracking, Idunderstanding and Mdocstore (mobile driving licence and ID documents). The server never listed them as pending for this device, so Play services never asked for them. They are hardware- or feature-gated and need a phone that uses those features. Thirteen modules were seen being compiled on the audited Pixel 10 Pro Fold, including Fastpair, Dck, Tensorgps, SpoonPc, MediaEffectEnhancement, Home and MlkitDocscanUi, but their files sit in app-private storage and cannot be read without root. SEEN (names) · UNKNOWN (contents)

What this means for you. The permissions you give Play services, and those you give any app with an embedded Google map or ads, also apply to code you never installed and cannot list, and Google can change that code without a Play services update. The modules examined here are mostly what their names say. The practical controls are the same as before: deny the permissions you do not need, and, if you can live without Play Integrity and scanning features, block dynamic code loading from storage (section 09, item 8).

Evidence

Every claim tagged SEEN resolves to one of these locations in the decompiled code. Paths are as jadx 1.5.6 names them when it decompiles the APKs listed in the artifact index.

Not published. The decompiled sources, the APKs and the module APKs collected from the test device are Google's proprietary software, and I do not redistribute them. To check a claim, download the same APK from the GrapheneOS repository (hash in the artifact index), decompile it yourself with jadx 1.5.6 and open the path given. The modules came off one test device, so only someone who pulls them from their own device can check the findings about them.

Cross-checks

The claims that carry the argument were checked directly in the decompiled code. The ones below hold.

- Check-in URL and the `READ_PHONE_STATE` gate before `getSimSerialNumber()` and `getSubscriberId()` : `bjtv.java:24,131,340,347` . SEEN
- Clearcut batch default URL `https://play.googleapis.com/log/batch` read through a flag: `jwgm.java:12` . SEEN
- Manifest carries `QUERY_ALL_PACKAGES` , `sharedUserId` and `isolatedSplits` , one each. SEEN
- The APK upload path is a queue: `UPLOAD_APK` intent, forwarded to `InternalApkUploadChimeraService` with network-change and package-change receivers. SEEN
- No decompiled file mentions GrapheneOS or GmsCompat, which supports "unmodified Google build". SEEN

Also checked (counts are files in the decompiled code that reference the API, not call counts):

Installed-app queries	122 files use <code>getInstalledPackages</code> , <code>getInstalledApplications</code> or <code>queryIntentActivities</code>
Usage statistics	26 files reference <code>UsageStatsManager</code> / <code>queryUsageStats</code> (gated by a special-access permission)
Install source	27 files read the installer package or install source
Android ID reads	35 files read <code>"android_id"</code>
Radio scans	16 files read Wi-Fi scan results or cell info (location stack)
SMS	3 files touch the SMS provider
Self-toggling	4 files change component enabled state (<code>brtr</code> , <code>gwyn</code> , <code>ky</code> , <code>brtq</code>)
Network stack	55 files reference Cronet; a Cronet split ships in the repository

These are counts of references, not evidence of use on a given device. Whether any path runs depends on permissions, flags and the account.

Code locations

EVIDENCE	WHAT IT ESTABLISHES
<code>defpackage/jwbq.java:6</code>	default check-in server URL

EVIDENCE	WHAT IT ESTABLISHES
defpackage/bjtv.java:24,262,501,652	check-in URL construction and gstatic fallbacks
defpackage/bjtv.java:131,340,347	READ_PHONE_STATE gate, IMSI and SIM serial reads
defpackage/bjzi.java:288–292	check-in request fields gated individually by flags
defpackage/bjsc.java:41–42,122–124	security token file and /data/misc_ce/<user>/checkin
com/.../checkin/CheckinChimeraService.java:187	43,200 s interval, 10,800 s flex
defpackage/jwpq.java:30	Chimera check-in interval 43,201 s
defpackage/jxaa.java:15	Phenotype interval 86,400 s
defpackage/jwgm.java:12	Clearcut batch endpoint
com/.../clearcut/internal/PlayLoggerContext.java:67	uploadAccount / logAndroidId / PII level
defpackage/brjn.java	468 log source enum constants
defpackage/dwrc.java:232,239	Measurement upload and SGTM endpoints
defpackage/jyqj.java:335	FCM registration endpoint
defpackage/jyqc.java:11–12	mtalk fallback host list
com/.../security/verifier/ApkUploadChimeraService.java:20	APK upload intent
com/.../core/providersettings/GoogleSettingsChimeraProvider.java:58	legacy GSF client IDs incl. rlz
defpackage/brtr.java:48, defpackage/gwyn.java:63	components switching their own enabled state
defpackage/gtub.java:8, wearable/node/DataService\$1.java:105	pinned package name and certificate SHA-1 for Play services and Play Store
assets/cobalt_config.pb	RAPPOR noised-metric configuration
AndroidManifest.xml	component names used in 10.2
defpackage/bysw.java:90–92	DroidGuard engine loaded with <code>DexClassLoader</code> as <code>com.google.ccc.abuse.droidguard.DroidGuard</code>
defpackage/bysr.java:14–15	the two pinned DroidGuard signing-certificate SHA-256 digests
defpackage/byre.java:52	<code>app_dg_cache</code> : where the engine APK is stored
defpackage/bysq.java:53–60	signed creation response, then <code>byteCode</code> / <code>vmUrl</code> / <code>vmChecksum</code>
defpackage/afgy.java:189 · afin.java:159 · afhx.java:81	<code>DelegateLastClassLoader</code> for file-backed, split-backed and foreign-package Chimera modules
defpackage/afhi.java:25 + afho.java:1144	<code>assets/chimera-modules/*.apk</code> copied out to app storage
com/.../nativeloader/NativeHotLoader.java:58	<code>hotLoad(String path)</code> → <code>dlopen</code> in <code>libisolation_mgr_jni</code>

EVIDENCE	WHAT IT ESTABLISHES
defpackage/brvn.java:84–155	reflective rewrite of <code>pathList.nativeLibraryDirectories</code>
defpackage/fymj.java:22–40	<code>snet_shared_uuid</code> minted, stored, rotated by a remote counter
defpackage/fyrt.java:55,1332 · fypo.java:51	the SNET collector list, its default-on subset, and DEX-info bundles
defpackage/fysz.java:27 · fykj.java:27–28 · fykf.java:21–22	SNET/Play Protect upload endpoint; Safe Browsing OHTTP relays and HPKE key source

Artifact index

GMS base	base.apk from apps.grapheneos.org · sha256 1bfbf0c0ddab7470...
Splits	split_*.apk · 11 files, all hash-verified against the repository metadata
GSF	base.apk, same repository · 5.7 MB
Play Store	base.apk, same repository · 63.9 MB
Repo metadata	metadata.1.0.0.sjson · 98,973 bytes
Decompiled	jadx 1.5.6 output on my machine · 920 MB · 161,628 classes (not published)
Strings	773,000 entries extracted from the APKs (not published)
Host inventory	778 hosts, 1,307 URLs extracted from the strings (not published)
Working notes	module list (280 rows), native libraries (39), dynamic-loading paths (14), claim checks (36); kept privately
Modules	24 module APKs collected from one test device, SHA-256 recorded; kept privately

Tooling

apktool · jadx 1.5.6 · aapt2 · apksigner · protoc --decode_raw · Ghidra 12.2 headless · radare2 · Python DEX string-table parser · GrapheneOS frameworks_base branch 17 (permission protection levels) · GrapheneOS app repo metadata (signify).

Sources

REFERENCE	LOCATION
App repository	https://apps.grapheneos.org/ · metadata.1.0.0.sjson
GMS artifact	packages/com.google.android.gms/264032035/base.apk
GrapheneOS docs	https://grapheneos.org/usage · sandboxed Google Play
GmsCompat	github.com/GrapheneOS/platform_packages_apps_GmsCompat
Permission levels	github.com/GrapheneOS/platform_frameworks_base · branch 17 · core/res/AndroidManifest.xml
Tools	jadx 1.5.6 · apktool · aapt2 · apksigner 36.0.0 · protoc

Limits of the evidence

This report measures an artifact and gives no verdict on a company. The code shows what is *possible* and what is *wired*. It cannot show what is retained, joined or used. Read the tags literally: the distinction between "seen" and "inferred" is the difference between engineering and speculation.

Support

If the teardown helped you and you want to support it, you can send any amount. Pick whatever suits you.



Lightning (BOLT 12)

<lno1pgvqhmrp0ys8xetjwe5kxetnypskuctv09ekjueqg48pvggz7yjsxrllfqjjju2xf4qnvqd9fjk8acdy4cf7e7kep33vspyw8kplcs>

[Pay in wallet](#)



Monero

<89jGcmA65CReki73oLFvxHUzyR45wMqTN8G93ZYRizwVF2XUQgYKbPLGQYgxj2YjX4fxAUPQ7tLr4ATP7fpR4dSeHH3r9iH>

[Pay in wallet](#)



Zcash (unified address)

u1defv75w2x9p6u82qu8je0rrz5g98gnhvs0ervckctfrjvsscvcx00e4qgfk63vm0uar587exxhtp3ps7gwd54anwm32a3sg2432htr53nv22d23agngdpdkdt75lzd8pp0e3f6s2c6n52nk9ecmnjwvdxggfcl7ndsh5uq3wqptk60z

Other ways to support: juraj.bednar.io/en/support-me/