

A Systematic Study of Security Enforcement at Android Inter-Component Boundaries

An Empirical Evaluation of First-Party Interfaces
and Platform Security Guardrails on Android 17

Sandiyo Christan A
Independent Researcher
sandichrist6@gmail.com

October 2026

Abstract

Android’s inter-process communication (IPC) mechanisms (Activities, Services, BroadcastReceivers, ContentProviders, and URI scheme handlers) form the primary interface between applications and between applications and the system. We present a systematic study of security enforcement comprising 120 evaluations (99 inter-component interface evaluations across five interface categories, 10 platform security-control evaluations, and 11 dedicated positive controls) in over 30 Google and AOSP packages on Android 17 (API 37). Using an expected-versus-observed framework under a zero-permission attacker model, we evaluate whether each boundary’s enforcement mechanism functions as designed. We find 97 enforcement gaps (80.8%) and 23 enforcement holds (19.2%). From these observations, we derive a 14-category enforcement gap taxonomy organized across manifest, code, platform, and architectural layers. Within our purposively selected corpus, observed enforcement outcomes differ substantially across interface categories: evaluated Activities and BroadcastReceivers exhibit a 100% gap rate, while ContentProviders, which benefit from framework-provided defenses such as `setStrict()`, achieve a 27.6% holds rate. We find that all seven evaluated platform compatibility changes with security intent are either disabled or gated at a `targetSdkVersion` that none of the applications in our evaluated corpus targets. We document 18 custom URI scheme registrations handling sensitive data (authentication tokens, payment credentials, medical information) despite the known unverifiability of custom schemes (RFC 8252 §8.1). Our study includes 23 positive controls, calibration cases demonstrating that the evaluation procedure distinguishes functioning enforcement from observed enforcement failures.

Keywords: Android security, inter-process communication, IPC enforcement, ContentProvider, SQL injection, URI scheme, compatibility framework, zero-permission attacker, permission model, confused deputy

1 Introduction

Android’s inter-process communication (IPC) architecture enables applications to expose functionality to other applications and to the system through four component types: Activities, Services, BroadcastReceivers, and ContentProviders. These components, along with custom URI scheme handlers, form the boundaries through which sensitive data and control flow cross between trust domains. The security of these boundaries depends on enforcement mechanisms at multiple levels: manifest-declared permissions, code-level caller identity checks, framework-provided input validation, and platform-level compatibility changes.

Despite over a decade of research on Android

IPC security, a fundamental question remains unanswered: does the enforcement at these boundaries actually work as designed? Existing research has addressed this question partially. Automated tools such as ComDroid [1], CHEX [2], and IccTA [3] detect specific vulnerability patterns at scale but do not evaluate enforcement holistically. Published security advisories document individual gaps but do not identify systemic patterns. Studies of user comprehension of the permission model [4] and compositional security analysis [5] evaluate aspects of the framework’s design but not its deployment at individual boundaries.

We address this gap with a systematic evaluation comprising 120 security-enforcement evaluations in over 30 Google and AOSP packages on Android 17. Our approach differs from prior

Table 1: Abbreviations used in this paper.

Abbr.	Full Form
2FA	Two-Factor Authentication
ADB	Android Debug Bridge
AIDL	Android Interface Definition Language
AOSP	Android Open Source Project
API	Application Programming Interface
BAL	Background Activity Launch
CA	Certificate Authority
DEX	Dalvik Executable
DoS	Denial of Service
eSIM	Embedded SIM
FRP	Factory Reset Protection
GMS	Google Mobile Services
HCE	Host Card Emulation
ICC	Inter-Component Communication
IPC	Inter-Process Communication
NFC	Near Field Communication
PKCE	Proof Key for Code Exchange
PoC	Proof of Concept
SDK	Software Development Kit
SQL	Structured Query Language
TLPE	Third-party Library Proxy Execution
TOTP	Time-based One-Time Password
UID	User Identifier
UPI	Unified Payments Interface
URI	Uniform Resource Identifier

work in several ways. First, we frame each evaluation as a security-boundary assessment (does the enforcement mechanism at this boundary function as intended?) rather than a vulnerability detection exercise. Second, we report both enforcement gaps (where observed behavior diverges from expected) and enforcement holds (where enforcement works as designed), providing calibration cases that demonstrate the evaluation procedure can distinguish functioning enforcement from failures. Third, we evaluate enforcement across five inter-component interface categories and platform security controls in a single study, enabling cross-category comparison.

Research questions.

- RQ1:** How does effective security enforcement vary across Android inter-component interfaces in a purposively selected corpus of first-party applications?
- RQ2:** What recurring implementation and architectural patterns explain divergence between intended and observed security enforcement?
- RQ3:** Are interface categories with framework-provided security defenses associated with different observed enforcement outcomes

than categories that rely primarily on application-level checks?

Contributions.

1. **Expected-versus-observed framework.** A methodology for evaluating IPC boundary enforcement that determines expected behavior from documentation, sibling comparison, and platform design intent, then tests observed behavior through proof-of-concept applications and code review (Section 3).
2. **120 evaluations with 23 positive controls.** Security-enforcement evaluations across Activities (27), ContentProviders (29), Broadcast-Receivers (11), Services (14), custom URI schemes (18), platform security controls (10), and 11 separately evaluated well-defended components. Of the 120, 23 demonstrate functioning enforcement, serving as calibration cases (Section 4).
3. **14-category enforcement gap taxonomy.** A taxonomy derived from evidence, organized across manifest, code, platform, and architectural layers (Section 5).
4. **Platform compatibility change audit.** An empirical evaluation of whether Android’s IPC-related compatibility changes are enforced on a shipping device, finding all seven disabled or unenforced on the test device (Section 4.7).
5. **Cross-category enforcement comparison.** Evidence that observed enforcement outcomes differ substantially across interface categories, with categories offering framework-provided defenses showing higher holds rates (Section 6).

The remainder of this paper is organized as follows. Section 2 provides background on Android IPC and the permission model. Section 3 describes our threat model and methodology. Section 4 presents results. Section 5 derives the enforcement gap taxonomy. Section 6 discusses implications and threats to validity. Section 7 surveys related work. Section 8 concludes.

2 Background

2.1 Android IPC Components

Android applications expose functionality through four component types, each with distinct IPC semantics:

Activities are user-interface screens launched via `startActivity()`. An exported Activity can be launched by any application, receiving attacker-controlled data through Intent extras.

Services provide background processing, accessed via `bindService()` or `startService()`. Bound services expose AIDL interfaces for inter-process method calls.

BroadcastReceivers respond to system-wide or application-specific broadcast messages. A receiver processes attacker-controlled data from the broadcast Intent.

ContentProviders expose structured data through a SQL-like query interface (`query()`, `insert()`, `update()`, `delete()`) and through file access (`openFile()`) and provider-defined method calls (`call()`). Callers supply selection predicates, projections, and sort orders that the provider incorporates into database queries.

2.2 Android Permission Model

Android controls component access through permissions at several protection levels: *normal* (auto-granted at install), *dangerous* (requires user approval), *signature* (same-certificate apps only), and *signature/privileged* (platform-signed or privileged partition). Components declare access requirements through manifest attributes: `android:permission` for Activities, Services, and BroadcastReceivers; `android:readPermission` and `android:writePermission` for ContentProviders. An `android:permission` declared at the application level provides a default permission to all components unless individually overridden.

2.3 Component Export and Intent Resolution

A component with `android:exported="true"` (or, for applications targeting API levels below 31, implicitly exported by declaring an intent filter) is accessible to any application. The `android:exported` attribute controls visibility; the `android:permission` attribute controls authorization. A component that is exported without a permission attribute is accessible to any application without restriction, unless a default permission is set at the application level.

2.4 ContentProvider Query Defenses

Android provides framework-level defenses for ContentProvider query injection. These

are methods of `SQLiteQueryBuilder` and must be explicitly enabled by the developer: `setStrict(true)` checks the selection for injected clauses and, together with a projection map, rejects requested columns that are not in that map; `setStrictColumns(true)` verifies that projections and inserted or updated values contain only columns defined in the projection map; and `setStrictGrammar(true)` rejects selection and sort-order clauses that contain unexpected SQL keywords or tokens, such as subqueries. The compatibility change `ENFORCE_STRICT_QUERY_BUILDER` (ChangeId 143231523) was introduced to enable `SQLiteQueryBuilder` validation for specific ContentProvider queries; its documented scope covers CalendarProvider2 query selections.

2.5 Custom URI Schemes and App Links

Applications can register custom URI schemes (e.g., `otpauth://`, `gpay://`) in their manifest intent-filters. Unlike HTTPS App Links, which support domain verification via `android:autoVerify`, custom URI schemes cannot be verified by the platform. Any application can register a handler for any custom scheme, enabling potential interception. RFC 8252 §8.1 [7] documents this limitation and recommends Proof Key for Code Exchange (PKCE) for public native clients, so that an intercepted authorization code cannot be redeemed by the interceptor.

2.6 Platform Compatibility Framework

Android's Compatibility Framework introduces behavioral changes gated by the target application's `targetSdkVersion`. Changes can be globally enabled, globally disabled, or conditionally enabled for applications targeting a minimum SDK version. This mechanism is intended for gradual adoption of breaking changes, including security hardening.

3 Threat Model and Methodology

3.1 Attacker Model

We adopt a **zero-permission app attacker** model. The attacker is a malicious application installed on the same device as the target, holding zero runtime or manifest permissions beyond those automatically granted at install time (normal-level permissions).

The attacker can: install via standard channels, send explicit intents to any exported component, query any exported ContentProvider, register intent filters for any custom URI scheme, and bind to any exported Service without `android.permission`. On Android 14 and later, Background Activity Launch (BAL) restrictions prevent background applications from starting Activities; all dynamic Activity evaluations in this study were executed from a visible foreground Activity in the proof-of-concept application, which reflects a realistic attacker capability.

The attacker cannot: hold signature-level or privileged permissions, modify the system partition, exploit kernel-level vulnerabilities, or interact with non-exported components (unless granted temporary URI permission).

Two further ContentProvider observations need capabilities beyond this model and are therefore excluded from the 120 evaluations. First, SettingsProvider skips its per-key read restriction for applications marked test-only, but such an application can be installed only through the Android Debug Bridge (`adb install -t`). Second, Drive’s StorageBackendContentProvider correctly rejects ordinary applications (counted as a hold, CP-18 in Section 4.3) but is reachable from the ADB shell, whose user holds the required `MANAGE_DOCUMENTS` permission.

3.2 Evaluation Categories

We organize evaluations into five inter-component interface categories and one platform security-control category (Figure 1):

- **B1** (Activity launch): Does the target Activity verify the caller before processing attacker-controlled extras?
- **B2** (ContentProvider query): Does the provider validate query parameters and enforce access control?
- **B3** (Broadcast delivery): Does the receiver verify the sender before acting on broadcast data?
- **B4** (Service binding): Does the service verify the caller’s identity and authorization?
- **B5** (URI scheme): Can the platform verify which app should handle a given custom URI scheme?
- **B6** (Platform controls): Are platform-level security hardening mechanisms active on the test device?

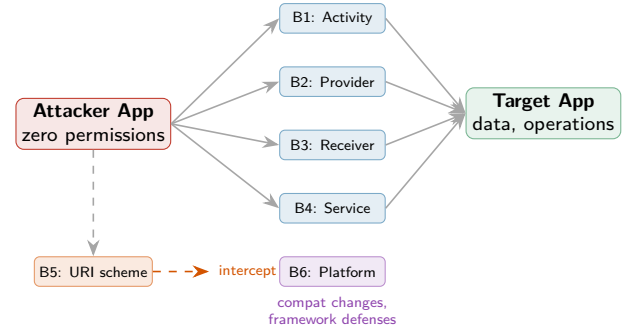


Figure 1: Threat model. Solid arrows: direct IPC (B1–B4). Dashed: URI scheme interception (B5). B6 covers platform controls.

B1–B5 evaluate enforcement at inter-component interfaces. B6 evaluates platform security guardrails that are intended to reinforce component-level enforcement; these are not IPC boundaries themselves but platform-level controls with security intent.

3.3 Expected-Versus-Observed Framework

For each evaluation, we determine the **expected enforcement** from one or more sources, listed from strongest to weakest: (1) explicit platform or security specification, (2) Android developer documentation specifying the intended behavior, (3) protection level semantics, where a permission’s level does not match the sensitivity of the protected resource, (4) sibling comparison, where a component’s siblings in the same manifest implement a security mechanism but the evaluated component does not.

We determine the **observed enforcement** through: dynamic testing (PoC app execution on a Pixel 6a running Android 17, with logcat capture), static analysis (manifest analysis and code review of decompiled APKs or AOSP source), or both.

Each evaluation is classified as a **GAP** (observed behavior diverges from expected enforcement) or **HOLDS** (enforcement works as designed). Evidence is tagged as S (static only), D (dynamic only), or S+D (both).

Each evaluation carries an identifier whose prefix names its category: A (Activity), CP (ContentProvider), BR (BroadcastReceiver), SV (Service), URI (custom URI scheme), SYS (platform control), and NEG (well-defended component used as a positive control). For example, CP-06 is the sixth ContentProvider evaluation. Section 4 discusses evaluations in functional groups and cites

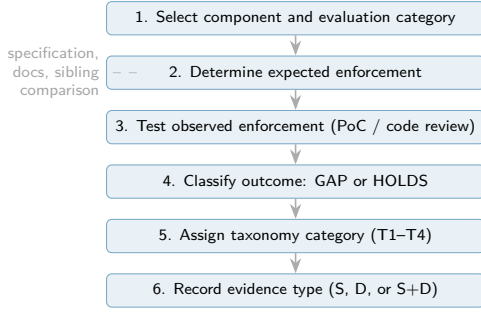


Figure 2: Per-evaluation workflow. Steps 2–6 are applied to each of the 120 evaluations. Expected enforcement is derived from specification, documentation, protection-level semantics, or sibling comparison.

these identifiers so that each statement can be matched to a specific evaluation, and Appendix A lists all 120 evaluations by identifier with the component, package, outcome, and evidence type. Identifiers are numbered consecutively within each category, except that ContentProvider identifiers run to CP-34 although 29 are counted: three duplicate positive controls NEG-06 to NEG-08, and two require ADB access (Section 3.1).

Figure 2 summarizes the study workflow.

3.4 Evaluation Scope

In scope: Android phone inter-component interfaces on a Google Pixel 6a running Android 17 (API 37, builds CP2A.260605.012 and CP31.260608.007). Google and AOSP first-party applications, including Chrome and Fitbit.

Out of scope: Wear OS components (separate security model), kernel/Binder-layer analysis (below the IPC abstraction), network-level attacks, and WebView/JavaScript bridge security, with one exception (SYS-10, Section 4.7).

3.5 Limitations of Methodology

Our evaluations are not randomly sampled; the researcher selected boundaries based on judgment about where gaps might exist, biasing the gap rate upward. The 80.8% gap rate characterizes the evaluated corpus, not all Android IPC boundaries. Twenty-one evaluations (17.5%) are based on static analysis only and lack runtime confirmation. Evaluations cover a single device (Pixel 6a) and Android version (17). The study focuses on Google/AOSP first-party apps; findings may not generalize to the broader Android app ecosystem. App versions are as installed at evaluation time; Google apps receive frequent updates.

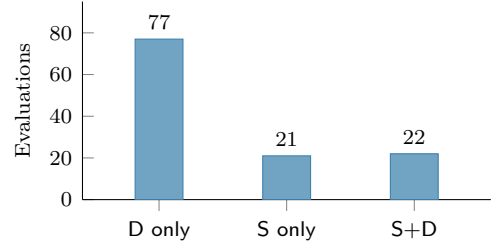


Figure 3: Evidence type distribution. D=dynamic PoC only, S=static analysis only, S+D=both. 82.5% of evaluations have dynamic evidence.

Table 2: Enforcement outcomes by evaluation category. Gap rate = Gaps / Total.

Category	N	Gaps	Holds	Gap %
B1: Activities	27	27	0	100.0
B2: ContentProv.	29	21	8	72.4
B3: Broadcast	11	11	0	100.0
B4: Services	14	10	4	71.4
B5: URI Schemes	18	18	0	100.0
B6: Platform Ctrl.	10	10	0	100.0
Positive Controls	11	0	11	0.0
Total	120	97	23	80.8

4 Results

Throughout this section, identifiers in parentheses, such as (CP-06) or (A-12), point to individual evaluations; Appendix A gives the full entry for each.

4.1 Enforcement Landscape

Table 2 summarizes enforcement outcomes across all evaluation categories. Of 120 evaluations, 97 (80.8%) exhibit enforcement gaps and 23 (19.2%) exhibit functioning enforcement. Within our purposively selected corpus, observed enforcement outcomes differ substantially: Activities, BroadcastReceivers, custom URI schemes, and platform controls show 100% gap rates, while ContentProviders and Services show lower gap rates (72.4% and 71.4%), that is, holds rates of 27.6% and 28.6%. The 100% rate for custom URI schemes is structural rather than a per-application failure: the platform cannot verify any custom scheme (Section 4.6), so every scheme selected for carrying sensitive data is a gap.

Figure 3 shows the evidence type distribution. Dynamic evidence (D or S+D) supports 99 of 120 evaluations (82.5%); 21 evaluations (17.5%) rely on static analysis only.

Figure 4 shows outcomes by evaluation category.

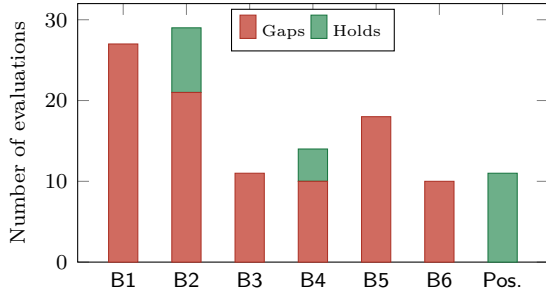


Figure 4: Enforcement outcomes by category. B1=Activities, B2=ContentProviders, B3=Broadcast-
Receivers, B4=Services, B5=URI Schemes, B6=Platform
Controls, Pos.=Positive Controls.

4.2 Activity Boundary Findings (B1)

All 27 evaluated Activity boundaries exhibit enforcement gaps, spanning eight functional groups.

OAuth/authentication callbacks (A-01–A-03). Three authentication callback Activities in Gmail and GMS accept arbitrary callers. CseRedirectUriReceiverActivity (Gmail) processes OAuth callback data from any application. BrowserSignInResponseHandlerActivity (GMS) handles Firebase Auth callbacks on both HTTPS and HTTP schemes without origin verification. AuthzenDeepLinkHandlerActivity (GMS) handles sign-in challenge links without verifying the caller.

Payment Activities (A-04–A-07). Four Activities in Google Wallet and GMS handle payment-related operations without adequate caller verification. FinishAndroidAppRedirectProxyActivity forwards caller-controlled URLs through GMS’s process context. AddNewCardThroughBrowserActivity accepts card-tokenization requests from any caller. PresentationModeActivity activates NFC tap-to-pay without lock screen authentication. Wallet’s DeepLinkActivity declares `android:autoVerify`, but that attribute only governs implicit intents, so any application can open its deep-link routes with an explicit intent.

Credential and identity Activities (A-08, A-09). SharePasswordsActivity (GMS) accepts an arbitrary requesting agent name and package in its credential-sharing flow, and a diagnostics tool’s login Activity uses an authentication check that always succeeds.

Debug Activities (A-10, A-11). CalendarDebugActivity runs in the CalendarProvider process, allowing any caller to bypass the `READ_CALENDAR` permission check. ContactsDumpActivity zips and serves the entire contacts data directory without access control.

Confused deputy (A-12–A-17). Six Ac-

tivities act on a caller’s behalf with privileges the caller lacks, the confused deputy pattern [12]. Three render protected data directly: LaunchInfoActivity renders calendar events without requiring `READ_CALENDAR`, QuickContactActivity renders full contact cards without `READ_CONTACTS`, and HostPhotoPagerActivity renders photos without `READ_MEDIA_IMAGES`. Three relay caller-controlled requests under the target application’s identity: BetterBug’s DispatchActivity forwards intents to an Activity protected by the `DUMP` permission (A-15), SystemUI’s MediaProjectionPermissionActivity reads the package name for its consent-reuse check from an attacker-controlled extra when the caller is unknown (A-16, static evidence), and a Play Store Activity forwards a caller-controlled URI from the Play Store process (A-17, static evidence).

DoS and inconsistent validation (A-18–A-23). Five Activities crash when they receive empty or malformed Intents, causing process-level denial of service (A-18, A-20–A-23). KeyChainActivity (A-20) crashes the system-UID process; BrightnessDialog (A-22) crashes the main SystemUI process. In the same group, a Dialer handler (A-19) is the only one of its sibling handlers that omits the Google signature check the others apply.

PII exposure and privileged configuration (A-24–A-27). An emergency-information screen designed for the lock screen is exported to any application and exposes the owner’s name, medical information, and emergency contacts (A-24). Three Activities perform privileged configuration without the permission their context implies: a Wi-Fi calling activation screen whose sibling Activities require `MODIFY_PHONE_STATE` (A-25), an eSIM download screen that accepts a forced-provisioning flag from any caller (A-26), and a parental-controls setup Activity reachable without a manifest permission (A-27, static evidence).

4.3 ContentProvider Boundary Findings (B2)

Of 29 evaluated ContentProvider boundaries, 21 exhibit gaps and 8 exhibit holds, the highest holds rate of any interface category.

SQL injection (CP-01–CP-13). CalendarProvider2 accepts unsanitized input in its selection, projection, and write-path (update and delete) parameters, permitting cross-table data extraction (CP-01–CP-03), and the platform change that would enforce strict query validation for it is

disabled (CP-04; see SYS-01). Selection injection also reaches ContactsProvider2, CallLogProvider, and CarrierIdProvider, the last with full UNION injection and no permission required (CP-05, CP-07, CP-08); sortOrder injection affects four providers (CP-09); and CellBroadcastContentProvider applies none of the strict-mode checks (CP-10, static evidence). Where the defenses are applied, they hold: ContactsProvider2 blocks projection injection via `setStrict(true)` (CP-06), CalendarProvider2's instances query enables all three strict-mode checks (CP-11), and MediaProvider and DownloadProvider reject SQL keywords in sortOrder (CP-12, CP-13).

Shared library gap (CP-14–CP-19). Four Google Workspace applications (Drive, Docs, Sheets, Slides) share a `LegacyStorageBackendContentProvider` with zero permission checks on `query()` (CP-14–CP-17). The modern `StorageBackendContentProvider` in the same packages correctly requires `MANAGE_DOCUMENTS` (CP-18, HOLDS), and the legacy provider's `openFile()` resists path traversal because it looks files up by database identifier rather than building a filesystem path from the request (CP-19, HOLDS).

Asymmetric permissions (CP-23, CP-24). `MmsSmsProvider` declares `readPermission` but no `writePermission`, allowing deletion of SMS/MMS records. `GoogleSettingsProvider` declares `writePermission` but no `readPermission`, exposing 18 partner settings.

Other provider gaps (CP-20–CP-22, CP-25, CP-27, CP-34). A debug provider serves a ZIP archive of the contacts database (CP-20); Settings' `SuggestionStateProvider` reveals screen-lock and biometric enrollment status (CP-21); Messages' `AvatarContentProvider` proxies arbitrary `content://` URIs under its own `READ_CONTACTS` permission (CP-22); `ServiceStateProvider` skips its permission checks for callers targeting API levels below 31 (CP-25); `GServicesProvider` is guarded only by a normal-level permission while exposing 1,404 configuration entries (CP-27); and a temporary URI grant from the contact picker permits SQL injection that reaches all internal `ContactsProvider2` tables (CP-34).

Of the identifiers CP-28 to CP-33, three duplicate positive controls (CP-28 to CP-30) and one falls outside the attacker model (CP-33, Section 3.1). The other two are counted holds: `BluetoothOppProvider` applies `setStrict(true)` and a path permission (CP-31), and the phone application's SIM contact providers require contacts

permissions (CP-32).

4.4 BroadcastReceiver Boundary Findings (B3)

All 11 evaluated `BroadcastReceiver` boundaries exhibit gaps.

GmsExternalReceiver (BR-01–BR-03). GMS uses a single exported receiver for 31 distinct Intent actions with no `android.permission` and no per-action sender validation (BR-03). Behind it, `FrpUpdateIntentOperation` writes the Factory Reset Protection (FRP) secret without sender verification (BR-01), and `AuthZenEventHandler` forces re-registration of six `CryptAuth` security keys (BR-02).

Debug receivers (BR-04–BR-06). `TestingToolsBroadcastReceiver` is present in 11 Google applications, gated by a runtime Phenotype flag (Google's server-controlled feature-flag system) rather than stripped at build time (BR-04). `CalendarProvider`'s debug receiver responds to a secret-code broadcast (BR-05), and a test receiver in `com.google.android.apps.scone` forwards simulated state changes to a privileged service (BR-06).

Safety-critical receiver (BR-08). `CrisisNotificationBroadcastReceiver` accepts broadcast intents from any application, enabling dismissal of earthquake, tsunami, and active shooter alerts.

Other receivers (BR-07, BR-09–BR-11). `SystemUI`'s media-output dialog action is not declared as a protected broadcast although its sibling action is (BR-07); an OMA-DM receiver without a permission invokes a permission-protected receiver directly as a Java method, bypassing the system's broadcast permission check (BR-09); `Wallet` receivers let any application cancel `Wallet` notifications, including fraud alerts (BR-10); and the Messages RCS provisioning receiver relies on a normal-level permission and validates the sender for only one of its seven actions (BR-11).

4.5 Service Boundary Findings (B4)

Of 14 evaluated `Service` boundaries, 10 exhibit gaps and 4 exhibit holds.

MediaBrowserService pattern (SV-01, SV-02). `YouTube`'s `MediaBrowserService` only blocks `SystemUI`, accepting all other callers with zero permissions. `Bluetooth`'s `MediaBrowserService` accepts all callers with zero validation.

Missing or weak service permissions (SV-03–SV-06, SV-09, SV-10). GMS exports

126 services without `android.permission` (SV-03), and Photos exports seven backup services without permission while keeping its internal backup services unexported (SV-04). Messages and the Google app guard services with normal-level permissions that every application receives automatically (SV-05, SV-10). FederatedCompute performs no Binder caller checks anywhere in its service (SV-06), and an on-device personalization service lacks the permission its sibling service requires (SV-09).

Inconsistent AIDL enforcement (SV-07, SV-08). KeyChainService exposes approximately 50 Binder methods; 5 lack caller identity checks, allowing enumeration of 120 system certificate authority (CA) certificate aliases. NfcService protects all but 2 of approximately 50 methods; the two unprotected methods accept forged host card emulation (HCE) data.

Positive controls (SV-11–SV-14). FelicaServiceImpl enforces `getCallingUid()` (SV-11). ZeroPartyBinder performs UID plus signature verification (SV-12). NfcService enforces NFC permissions on all its other methods (SV-13). Two GMS services throw SecurityException on unprivileged bind attempts (SV-14).

4.6 Custom URI Scheme Findings (B5)

We identified 18 custom URI scheme registrations handling security-sensitive data (URI-01 to URI-18); where one application registers several related schemes for the same purpose, such as `tez://` and `gpay://` in Google Pay, they form a single evaluation. Table 3 lists selected schemes. All 18 share the same architectural exposure: Android cannot verify which application should handle a custom scheme [7], meaning any installed application can register a competing handler for any of these schemes.

The 18 schemes span 15+ packages and handle authentication credentials, payment tokens, eSIM activation codes, medical information, and other sensitive data. Multiple applications independently chose custom schemes despite RFC 8252’s documentation of the interception risk.

4.7 Platform Security-Control Findings (B6)

All 10 platform-level evaluations exhibit gaps.

Compatibility changes (SYS-01–SYS-07). We evaluated seven Android compatibility framework changes with security intent (Table 4). On our test device (Pixel 6a, Android 17), three

Table 3: Selected custom URI scheme registrations (18 total). All share the architectural limitation that Android cannot verify ownership of non-HTTPS custom schemes [7].

Scheme	Application	Data at Risk
otpaath://	Authenticator	TOTP 2FA secret keys
LPA://	eSIM manager	eSIM activation codes
FIDO://	GMS	FIDO2 session data
genericidp://	5 apps	OAuth authorization codes
comgoogle.wallet://	GMS (Wallet)	Card tokens, digital car keys
tez://, gpay://	Google Pay	UPI payment data
safetyhub://	Safety Hub	Medical information parameters

Table 4: Platform compatibility changes with security intent. “Observed” reflects the state on the test device (Pixel 6a, Android 17): Disabled means off for all applications; SDK 37 means enabled only for applications whose `targetSdkVersion` is 37 or higher.

ID	Change	Observed
SYS-01	ENFORCE_STRICT_QUERY_BUILDER	Disabled
SYS-02	ENFORCE_INTENTS_TO_MATCH_INTENT_FILTERS	Disabled
SYS-03	BLOCK_NULL_ACTION_INTENTS	Disabled
SYS-04	ENFORCE_STRICT_SQL_CHECKS	SDK 37
SYS-05	PARCEL_HARDENING	SDK 37
SYS-06	RESTRICT_DATA_URI_COLUMNS	SDK 37
SYS-07	DETECT_IMPLICIT_URI_PERMISSION_GRANT	SDK 37

changes are disabled and four are gated at `targetSdkVersion=37`, which none of the applications in our evaluated corpus targets. Some of these changes may have different documented default states; our observations reflect the state observed via `am compat get-config` on the test device.

Parcelable mismatch regression (SYS-08). A Settings Tile Parcelable implementation writes and reads fields in different orders, enabling a self-changing Bundle attack. This class of vulnerability (Parcelable serialization/deserialization mismatch) was previously identified in CVE-2023-45777 [8]; our observation represents a similar pattern on a current platform build. The platform defense (`PARCEL_HARDENING`) is gated at `targetSdk=37` on the test device and thus unenforced.

Third-party library proxy execution

(SYS-09). InCallController in `system_server` uses `createPackageContext()` with flag 3 (`CONTEXT_INCLUDE_CODE | CONTEXT_IGNORE_SECURITY`) against an attacker-controlled package during `getInCallServiceComponents` enumeration, triggered by any app holding the auto-granted `MANAGE_OWN_CALLS` permission. According to the original report, this loads code from the attacker’s Dalvik Executable (DEX) into `system_server` (UID 1000), enabling arbitrary code execution without user interaction. This vulnerability was discovered and reported by sithi [10], assigned CVE-2026-49881 [9], and patched in the September 2026 Android Security Bulletin. We did not rediscover it; we include it because our static review confirmed that the vulnerable code path is present in the evaluated builds, which predate that patch. Our search for the same pattern elsewhere found only uses that resolve to Google-signed packages in application context (NEG-09, NEG-10).

JavaScript bridge (SYS-10). The one WebView-related evaluation in scope concerns a JavaScript bridge in Google Maps whose `openUrl()` method turns script-supplied URLs into native intent launches within the Maps process. This evaluation rests on static evidence only; we did not confirm that an attacker can inject script into the relevant WebView.

4.8 Positive Controls

Twenty-three evaluations demonstrate enforcement holds, providing calibration cases that confirm the evaluation procedure can detect functioning enforcement.

Well-defended components (NEG-01–NEG-11). These eleven evaluations target components chosen as positive controls rather than as suspected gaps. Pixel Launcher uses signature permissions combined with SHA-256 package allowlists (NEG-01). The NFC stack uses flag 0 for `createPackageContext`, avoids `FLAG_MUTABLE` for `PendingIntents`, and enforces permissions on all Binder methods (NEG-03). `ManagedProvisioning` performs multi-factor code-level caller verification despite being exported without a manifest permission (NEG-04). The strongest of these components combine manifest-level permissions, code-level caller validation, and explicit allowlists.

ContentProvider and Service defenses. The eight `ContentProvider` holds and four `Service` holds are described in Sections 4.3 and 4.5. Table 5

Table 5: The 23 enforcement holds and the mechanism that held in each case.

ID	Component	Mechanism that held
CP-06	Contacts-Provider2	<code>setStrict(true)</code> blocks projection injection
CP-11	Calendar-Provider2 (instances)	All three strict-mode checks enabled
CP-12	MediaProvider	Rejects SQL keywords in <code>sortOrder</code>
CP-13	DownloadProvider	Rejects SQL keywords in <code>sortOrder</code>
CP-18	Drive StorageBackend provider	<code>MANAGE_DOCUMENTS</code> (signature privileged)
CP-19	Drive legacy <code>openFile()</code>	Lookup by database ID; no path built from input
CP-31	BluetoothOpp-Provider	<code>setStrict(true)</code> and path permission
CP-32	SIM contact providers	Contacts permissions required
SV-11	FelicaServiceImpl	<code>getCallingUid()</code> check
SV-12	ZeroPartyBinder	UID and signature check
SV-13	NfcService (other methods)	NFC permission checks enforced
SV-14	Two GMS services	<code>SecurityException</code> on unprivileged bind
NEG-01	Pixel Launcher	Signature permissions and SHA-256 allowlists
NEG-02	Google Meet	No intent forwarding or JavaScript bridge
NEG-03	NFC stack	Safe context flags, immutable <code>PendingIntents</code> , Binder permission checks
NEG-04	Managed-Provisioning	Code-level caller verification
NEG-05	Phone application	Provider permissions; diagnostics limited to system user and debuggable builds
NEG-06	Gmail providers (10)	<code>SecurityException</code> , empty result, or not exported
NEG-07	GMS providers	Not exported or permission required
NEG-08	FileProviders	Not exported
NEG-09	<code>createPackageContext</code> uses	Resolve only to Google-signed packages
NEG-10	DynamiteModule loaders	Hard-coded GMS package name
NEG-11	17 other candidates (16 apps)	Properly secured or not exploitable

lists all 23 holds with the mechanism that held in each case.

5 Enforcement Gap Taxonomy

From the 97 enforcement gaps, we derive a taxonomy of 14 enforcement failure patterns organized across four layers. The taxonomy groups patterns by where in the enforcement stack the failure occurs; we note that individual categories represent different kinds of entities (implementation errors, deployment states, architectural limitations) unified by their shared

Table 6: 14-category enforcement gap taxonomy derived from 97 gaps. Total exceeds 97 because some evaluations exhibit multiple patterns.

ID	Category	Count	Layer
T1.1	Missing authorization	35	Manifest
T1.2	Asymmetric permission	3	Manifest
T1.3	Debug code in production	6	Manifest
T1.4	Protection level inadequacy	5	Manifest
T2.1	SQL injection	13	Code
T2.2	Missing Binder check	6	Code
T2.3	Confused deputy	7	Code
T2.4	Permission trampoline	2	Code
T2.5	DoS via input validation	4	Code
T3.1	Compat change unenforced	7	Platform
T3.2	Feature-flag-gated security	5	Platform
T3.3	targetSdk gating bypass	2	Platform
T3.4	Parcelable mismatch	1	Platform
T4.1	URI scheme unverifiable	18	Arch.

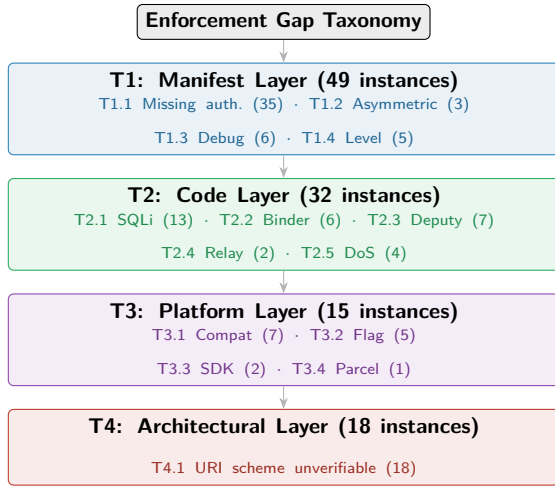


Figure 5: Taxonomy hierarchy: 14 categories across four layers. Instance counts in parentheses; total exceeds 97 due to co-occurring patterns.

effect of producing an observed enforcement gap. Table 6 summarizes the taxonomy, and Figure 5 shows the hierarchy.

T1.1 (“Missing authorization”) captures exported components that perform security-sensitive operations, such as accessing protected data, modifying system state, or forwarding privileged operations, without adequate authorization checks. An intentionally public component with no security-sensitive behavior is not classified here. The other categories are defined as follows. T1.2: a ContentProvider declares a read or a write permission but not both. T1.3: debug or test components are exported in release builds. T1.4: the guarding permission has normal protection level, so every application receives it automatically. T2.1: caller-supplied selection, projection, or sort-order text reaches SQL without validation. T2.2:

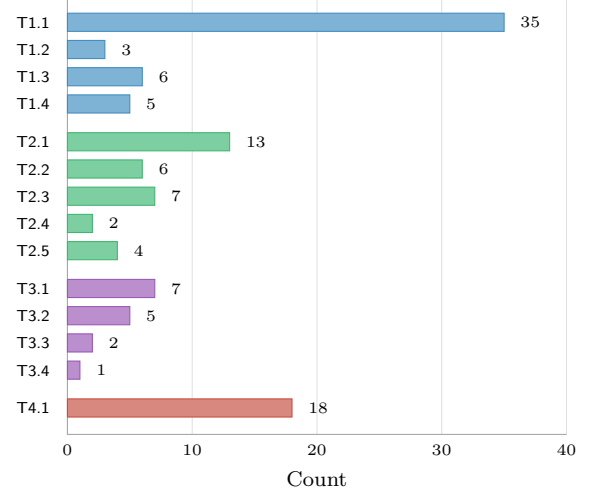


Figure 6: Taxonomy category distribution. Colors indicate layer: blue=manifest, green=code, purple=platform, red=architectural.

a Binder method performs no caller identity or permission check. T2.3: a privileged component accesses a resource on behalf of a caller without checking that the caller may access it. T2.4: an unprotected component forwards requests to a protected one under its own identity. T2.5: missing null or type checks on Intent data crash the receiving process. T3.1: a compatibility change with security intent is disabled or gated at an SDK level that applications do not yet target. T3.2: a security check depends on a server-controlled feature flag. T3.3: a permission check applies only to callers targeting a recent SDK level. T3.4: a Parcelable writes and reads fields in different orders, so a Bundle can change between parses. T4.1: the platform cannot verify which application owns a custom URI scheme.

The three most prevalent patterns, missing authorization on an exported component (T1.1, 35 instances), custom URI scheme unverifiable (T4.1, 18), and SQL injection in query parameters (T2.1, 13), account for 68% of all observed gaps.

Figure 6 shows the distribution of taxonomy categories.

Pattern co-occurrence. Several evaluations exhibit multiple patterns simultaneously. CalendarProvider2 (CP-01–CP-04) exhibits both SQL injection (T2.1) and a disabled platform defense (T3.1). The Google Workspace legacy providers (CP-14–CP-17) exhibit SQL injection (T2.1) compounded with feature-flag-gated fixes (T3.2). The Play Store redirect Activity (A-17) and the TestingToolsBroadcastReceiver debug receiver (BR-04) each combine their primary pattern with a

flag-gated fix (T3.2), `ServiceStateProvider` (CP-25) combines a missing permission with SDK-conditional checks (T3.3), and the Settings Tile Parcelable regression (SYS-08) exhibits Parcelable mismatch (T3.4) with the defending compat change, `PARCEL_HARDENING`, unenforced (T3.1).

6 Discussion

6.1 Summary of Findings by Research Question

RQ1 (enforcement variation across interfaces): Within our purposively selected corpus, observed enforcement outcomes differ substantially across interface categories. Activities and BroadcastReceivers exhibit 100% gap rates among evaluated boundaries, while ContentProviders and Services show holds rates of 27.6% and 28.6% (Section 6.2).

RQ2 (recurring patterns): The 14-category taxonomy identifies three dominant patterns: missing authorization (T1.1, 35 instances), URI scheme unverifiability (T4.1, 18), and SQL injection (T2.1, 13), accounting for 68% of observed gaps (Section 5).

RQ3 (framework-provided defenses): Interface categories where the framework offers per-request enforcement primitives are associated with higher holds rates: ContentProviders (`setStrict` family, separate `readPermission/writePermission`) at 27.6% and Services (per-call Binder caller identity) at 28.6%, compared with 0% for evaluated Activities and BroadcastReceivers (Section 6.2).

6.2 Framework-Provided Defenses and Enforcement Outcomes

Within our corpus, evaluated Activities and BroadcastReceivers show a 100% gap rate, while ContentProviders and Services show 72.4% and 71.4%. ContentProviders benefit from framework-provided defenses that developers can opt into: the three strict-mode methods of `SQLiteQueryBuilder` (Section 2.4) and separate manifest attributes for read and write permissions. When developers apply these mechanisms, enforcement holds: in `ContactsProvider2`, in `CalendarProvider2`'s instances `query`, in `MediaProvider` and `DownloadProvider`, and in `BluetoothOppProvider` (CP-06, CP-11–CP-13, CP-31). Services offer a different primitive: every Binder call carries the caller's identity, which the service can check with `getCallingUid()` or `enforceCallingPermission()`, and three

of the four `Service` holds, `FelicaServiceImpl`, `ZeroPartyBinder`, and `NfcService`, rely on such checks (SV-11–SV-13). Activities, by contrast, have no comparable framework-level caller verification mechanism: the receiving Activity is not told who started it unless the caller cooperates, and there is no framework validator for Intent extras analogous to `setStrict()`.

We observe an association between the availability of framework-provided defenses and better enforcement outcomes. A supplementary triage of all 376 Google and AOSP packages on the test device (encompassing 3,721 exported components) identified zero exported ContentProviders lacking access controls beyond those already in our corpus; each was secured through signature-certificate allowlists, calling-package verification, or runtime permission enforcement. This broader observation is consistent with the finding that ContentProvider enforcement benefits from framework-provided defenses that other categories lack. However, we note that this comparison is not controlled: the interface categories differ in many ways beyond framework support (API semantics, developer population, selection criteria), so we cannot establish a causal relationship from this data alone.

6.3 The Compatibility Change Paradox

On our test device, all seven evaluated platform compatibility changes with security intent are either disabled or gated at a `targetSdkVersion` that none of the applications in our evaluated corpus targets. The platform team recognized each problem, wrote a fix, and shipped it in the platform code; however, on this device, those fixes are not enforced. This creates a gap between the intended security behavior documented in the platform code and the actual enforcement observed on the device.

The `targetSdkVersion` gating mechanism creates an enforcement lag, and the same mechanism appears inside applications: `ServiceStateProvider` applies its permission checks only to callers targeting API 31 or later (CP-25), so an attacker application avoids them simply by declaring an older target such as API 28. This is compounded at the app level: `CalendarProvider2` has SQL injection (CP-01), and the platform defense that would catch it, the `ENFORCE_STRICT_QUERY_BUILDER` compatibility change, is disabled on the test device (SYS-01).

6.4 Custom URI Schemes: Systematic Reuse of a Known-Vulnerable Pattern

RFC 8252 §8.1, published in October 2017, documents that custom URI scheme redirect URIs are vulnerable to interception and recommends PKCE for public native clients. Despite this, we document 18 custom URI scheme registrations handling sensitive data across 15+ packages. The persistence of custom schemes nine years after the RFC suggests that awareness of an architectural limitation does not prevent its systematic reuse, particularly when the secure alternative (HTTPS App Links) requires additional infrastructure. We classify these as architectural exposure rather than necessarily exploitable vulnerabilities, since individual applications may implement compensating controls (e.g., PKCE, state validation).

6.5 Shared Code as Enforcement Gap Amplifier

A single shared library places `LegacyStorageBackendContentProvider` in four Google Workspace applications, each with the same missing permission check (CP-14–CP-17). `TestingToolsBroadcastReceiver` is present in 11 applications (BR-04). A single code-level gap in a shared library produces multiple affected applications. Conversely, a single fix in the shared library would remediate all.

6.6 The Catch-All Anti-Pattern

`GmsExternalReceiver` handles 31 Intent actions through a single exported receiver with no `android.permission`. This concentrates authorization at the per-action handler level, but at least two handlers perform no caller validation. In contrast, Pixel Launcher uses per-provider signature permissions with SHA-256 allowlists (NEG-01).

6.7 What Distinguishes Well-Defended Components

Across the 23 enforcement holds (Table 5), enforcement rests on four kinds of mechanism: (1) manifest-level permissions at signature or signature|privileged level, (2) code-level caller identity checks, (3) explicit package or signature allowlists rather than general permission checks, and (4) framework input validation. The most robust components, such as Pixel Launcher and

the NFC stack, combine several of these, applying code-level checks even where a manifest permission is already present. Others were evaluated for a single mechanism, such as the `sortOrder` validation in `MediaProvider` and `DownloadProvider`.

6.8 Threats to Validity

Selection bias. Evaluations are not randomly sampled; the 80.8% gap rate characterizes the evaluated corpus, not all Android IPC boundaries. These observations motivate a broader measurement study with systematic or random sampling. **Single device and version.** All evaluations use Pixel 6a running Android 17; compatibility change states may differ on other devices or builds. **Static-only evidence.** 21 evaluations (17.5%) lack runtime confirmation. **Google/AOSP focus.** Third-party app enforcement patterns may differ. **Temporal validity.** Specific gaps may be fixed after our evaluation window. **Confounding factors.** The observed association between framework-provided defenses and enforcement outcomes (RQ3) is not a controlled comparison; interface categories differ in multiple dimensions beyond framework support.

7 Related Work

Android IPC security tools. ComDroid [1] introduced static analysis for inter-application communication vulnerabilities and examined 20 applications in detail. CHEX [2] extended this with data-flow analysis for component hijacking across 5,486 apps. IccTA [3] performed ICC-aware taint analysis. AceDroid [6] detected inconsistent access control checks. These tools detect specific patterns at scale; we complement this with a holistic enforcement evaluation that includes positive controls, cross-category comparison, and platform-level assessment.

Permission model studies. Felt et al. [4] studied how users attend to, comprehend, and respond to Android permission prompts. Bagheri et al. [5] developed COVERT, a compositional analysis framework for detecting inter-app permission leakage. Our work evaluates enforcement of the existing permission model at individual boundaries rather than analyzing the model’s design or user comprehension.

ContentProvider security. Zhou and Jiang [11] detected passive content leaks and content pollution through exported `ContentProviders`, finding 2.3% of 62,519

Table 7: Comparison with prior work. ✓ = yes, – = no, ~ = partial.

Study	Pos. ctrl.	Exp. vs. obs.	Compat audit	Cross-type	URI survey
ComDroid [1]	–	–	–	~	–
CHEX [2]	–	–	–	✓	–
IccTA [3]	–	–	–	✓	–
AceDroid [6]	–	–	–	✓	–
Zhou & Jiang [11]	–	–	–	–	–
This study	✓	✓	✓	✓	✓

applications susceptible. We extend this with evaluation of the full enforcement chain: app-level injection, framework-level defenses (`setStrict`), and platform-level compatibility changes.

Confused deputy in Android. Felt et al. [12] defined the permission re-delegation attack model and proposed IPC Inspection. Grace et al. [13] showed that preloaded applications on stock Android phones leak permission-protected capabilities through exported components. Bugiel et al. [14] proposed XManDroid for cross-app IPC monitoring. We contribute concrete evaluations in current (2026) first-party applications, showing the pattern persists.

URI scheme security. RFC 8252 §8.1 [7] documents the custom URI scheme interception risk. To our knowledge, among the studies we reviewed, no prior work provides a systematic survey of custom URI scheme usage across a major first-party app ecosystem; we document 18 scheme registrations handling sensitive data despite the known risk.

Table 7 compares our study with prior work across key dimensions.

8 Conclusion

We presented a systematic study comprising 120 security-enforcement evaluations across five Android inter-component interface categories and platform security controls. Our expected-versus-observed framework evaluates enforcement as a security-boundary assessment, reporting both gaps (97) and holds (23). The 14-category enforcement gap taxonomy organizes failures across manifest, code, platform, and architectural layers, with three dominant patterns: missing authorization (T1.1), URI scheme unverifiability (T4.1), and SQL injection (T2.1), accounting for 68% of observed gaps (RQ2).

Three findings stand out. First, within our purposively selected corpus, observed enforcement outcomes differ substantially across interface categories (RQ1), and categories with framework-provided enforcement primitives are associated with higher holds rates (RQ3): ContentProviders and Services have holds rates of 27.6% and 28.6%, while Activities and BroadcastReceivers have 0% among evaluated boundaries. This observation motivates investigation into whether providing stronger framework-level enforcement defaults across all IPC types would improve outcomes. Second, all seven evaluated platform compatibility changes with security intent are disabled or unenforced on the test device, creating a gap between the platform team’s intentions and actual enforcement. Third, 18 custom URI scheme registrations handle sensitive data despite a nine-year-old RFC documenting the interception risk.

Our positive controls demonstrate that effective enforcement is achievable: the best-defended components combine manifest permissions, code-level caller checks, and explicit allowlists. The gap is not in the availability of enforcement mechanisms but in their consistent application.

Ethical Considerations

All evaluations were performed on the researcher’s own device (Google Pixel 6a). No user data was accessed or exfiltrated. Dynamic evaluations used proof-of-concept applications on the researcher’s own accounts. This study analyzes the design and behavior of publicly available system components; all observations are derived from manifest inspection, publicly available source code (AOSP), and standard IPC interactions on a researcher-controlled device.

References

- [1] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, “Analyzing inter-application communication in Android,” in *Proc. ACM MobiSys*, 2011, pp. 239–252.
- [2] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, “CHEX: statically vetting Android apps for component hijacking vulnerabilities,” in *Proc. ACM CCS*, 2012, pp. 229–240.
- [3] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Outeau, and P. McDaniel, “IccTA:

- detecting inter-component privacy leaks in Android apps,” in *Proc. IEEE/ACM ICSE*, 2015, pp. 280–291.
- [4] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, “Android permissions: user attention, comprehension, and behavior,” in *Proc. SOUPS*, 2012, pp. 3:1–3:14.
 - [5] H. Bagheri, A. Sadeghi, J. Garcia, and S. Malek, “COVERT: compositional analysis of Android inter-app permission leakage,” *IEEE Trans. Softw. Eng.*, vol. 41, no. 9, pp. 866–886, 2015.
 - [6] Y. Aafer, J. Huang, Y. Sun, X. Zhang, N. Li, and C. Tian, “AceDroid: normalizing diverse Android access control checks for inconsistency detection,” in *Proc. NDSS*, 2018.
 - [7] W. Denniss and J. Bradley, “OAuth 2.0 for Native Apps,” RFC 8252, Internet Engineering Task Force, October 2017.
 - [8] Android Security Bulletin, “CVE-2023-45777: Parcel mismatch in AccountManagerService intent validation,” Google, December 2023.
 - [9] Android Security Bulletin, “CVE-2026-49881: InCallController third-party library proxy execution,” Google, September 2026.
 - [10] sithi, “TLPE: third-party library proxy execution in Android InCallController,” <https://github.com/Supersonic/TLPE>, 2026.
 - [11] Y. Zhou and X. Jiang, “Detecting passive content leaks and pollution in Android applications,” in *Proc. NDSS*, 2013.
 - [12] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin, “Permission re-delegation: attacks and defenses,” in *Proc. USENIX Security Symposium*, 2011.
 - [13] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, “Systematic detection of capability leaks in stock Android smartphones,” in *Proc. NDSS*, 2012.
 - [14] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastri, “Towards taming privilege-escalation attacks on Android,” in *Proc. NDSS*, 2012.

A Index of All 120 Evaluations

This appendix lists every evaluation counted in the study, grouped by category. The identifier prefix names the category: A (Activity), CP (ContentProvider), BR (BroadcastReceiver), SV (Service), URI (custom URI scheme), SYS (platform control), and NEG (well-defended component used as a positive control). Each row gives the component and the boundary that was tested (for URI schemes, the scheme and the data it carries), the package or scope, the outcome (GAP: observed behavior diverges from expected enforcement; HOLDS: enforcement works as designed), and the evidence type (S: static analysis only; D: dynamic proof-of-concept testing on the device; S+D: both). ContentProvider identifiers are not contiguous: CP-26 and CP-33 require ADB access and are excluded (Section 3.1), and CP-28 to CP-30 duplicate positive controls NEG-06 to NEG-08.

ID	Component and boundary tested	Package or scope	Outcome	Evid.
B1: Activities (27 evaluations, all gaps)				
A-01	CseRedirectUriReceiverActivity: Caller verification on OAuth callback receiver (HTTPS scheme)	com.google.android.gm (Gmail)	GAP	D
A-02	BrowserSignInResponseHandlerActivity: Caller verification on Firebase Auth callback (HTTPS + HTTP)	com.google.android.gms	GAP	D
A-03	AuthzenDeepLinkHandlerActivity: Caller verification on Authzen challenge handler (google.authzen:// scheme, noDisplay)	com.google.android.gms	GAP	D
A-04	FinishAndroidAppRedirectProxyActivity: URL validation on redirect proxy chain (caller-controlled URL laundered through GMS context)	com.google.android.gms (Wallet)	GAP	D
A-05	AddNewCardThroughBrowserActivity: Caller verification on card tokenization handler (com.google.wallet:// scheme, singleInstance)	com.google.android.gms (Wallet)	GAP	D
A-06	PresentationModeActivity: Authentication enforcement for NFC tap-to-pay activation (QUICKDRAW action bypasses lockscreen/biometric)	com.google.android.apps.walletnfcrel	GAP	D
A-07	DeepLinkActivity: Permission enforcement on deep link handler (autoVerify only protects implicit intents)	com.google.android.apps.walletnfcrel	GAP	D
A-08	SharePasswordsActivity: Caller identity verification on AI agent credential-sharing flow (accepts arbitrary agent_name/agent_package extras)	com.google.android.gms	GAP	D
A-09	EndUserLoginActivity: Authentication enforcement on priv-app diagnostics (y() returns true unconditionally)	com.google.android.apps.diagnostics	GAP	S+D
A-10	CalendarDebug + CalendarDebugActivity: Export control on debug activities (bypass READ_CALENDAR by running in provider process)	com.android.providers.calendar	GAP	S+D
A-11	ContactsDumpActivity: Export control on debug activity (zips entire contacts data directory)	com.android.providers.contacts	GAP	D
A-12	LaunchInfoActivity: Permission enforcement on exported activity (renders calendar events without READ_CALENDAR; sequential ID enumeration)	com.google.android.calendar	GAP	D
A-13	QuickContactActivity: Permission enforcement on exported activity (renders full contact cards without READ_CONTACTS)	com.google.android.contacts	GAP	D
A-14	HostPhotoPagerActivity + EditActivity: Permission enforcement on exported activities (render photos without READ_MEDIA_IMAGES; sequential MediaStore ID enumeration)	com.google.android.apps.photos	GAP	D
A-15	DispatchActivity: Permission relay/trampoline (unpermissioned DispatchActivity forwards intents to DUMP-protected BugIntentDialogActivity using BetterBug's identity)	com.google.android.apps.betterbug	GAP	D
A-16	MediaProjectionPermissionActivity: Caller identity validation (confused deputy: when getCallingPackage()==null, reads package name from attacker-controlled extra for consent-reuse check)	com.android.systemui	GAP	S
A-17	AppDiscoveryLaunchActivity: URL validation on intent redirect (singleInstance; forwards caller-controlled URI from Play Store process context; flag-gated fix)	com.android.vending (Play Store)	GAP	S
A-18	ImRootActivity, DeepLinkEntryPointActivity, AssistedSignInActivity: Input validation on Chimera module-loading framework (crash loop from empty intents, 5-7+ restarts)	com.google.android.gms	GAP	D
A-19	GatewayActivity (DobbyPromoGatewayHandler): Caller verification consistency (Dobby handler is the ONLY GatewayActivity handler lacking GoogleSignatureVerifier check)	com.google.android.dialer	GAP	S+D

Appendix table, continued

ID	Component and boundary tested	Package or scope	Outcome	Evid.
A-20	KeyChainActivity: Input validation on Serializable extras (ClassCastException crashes system-UID process)	com.android.keychain	GAP	D
A-21	ReceiverActivity: Input validation on Parcelable extras (NPE crashes android.process.media)	com.android.mtp	GAP	D
A-22	BrightnessDialog: Dead legacy code (unconditionally throws IllegalStateException; crashes main SystemUI process)	com.android.systemui	GAP	D
A-23	KeyAttestationWarningActivity: Null check on getCallingPackage() (crash loop, 10+ cycles; cascading Play Store billing service restart)	com.google.android.gms	GAP	D
A-24	EmergencyDialerActivity + LockScreenActivity: Caller context validation (designed for lockscreen but exported to any app; exposes full name, medical info, emergency contacts)	com.google.android.apps.safetyhub	GAP	D
A-25	WfcActivationActivity: Permission enforcement (siblings correctly require MODIFY_PHONE_STATE; this activity has none)	com.google.android.wfcactivation	GAP	D
A-26	QrDownloadActivity: Caller validation on eSIM provisioning (FORCE_PROVISION=true from zero-perm app)	com.google.android.euicc	GAP	D
A-27	KidSetupActivity: Export control on parental controls (accepts DM_PRE_REMOVE_ACCOUNT, ENFORCE_LAUNCHER without manifest permission)	com.google.android.gms.supervision	GAP	S
B2: ContentProviders (29 evaluations: 21 gaps, 8 holds)				
CP-01	CalendarProvider2 (selection): Input validation on selection parameter (arbitrary subqueries in WHERE clause)	com.android.providers.calendar	GAP	D
CP-02	CalendarProvider2 (projection): Input validation on projection parameter (arbitrary SQL functions and subqueries; cross-table extraction in single queries)	com.android.providers.calendar	GAP	D
CP-03	CalendarProvider2 (write path): Input validation on UPDATE/DELETE selection (side-channel confidentiality breach via write operations)	com.android.providers.calendar	GAP	D
CP-04	CalendarProvider2 (compat gate): Platform enforcement of setStrict(true) via ENFORCE_STRICT_QUERY_BUILDER (ChangeId 143231523 globally disabled on Android 17)	com.android.providers.calendar	GAP	S+D
CP-05	ContactsProvider2 (selection): Input validation on selection parameter (CASE WHEN subqueries with cross-table references; blind boolean binary search)	com.android.providers.contacts	GAP	D
CP-06	ContactsProvider2 (projection): Input validation on projection parameter (correctly blocked by setStrict(true))	com.android.providers.contacts	HOLDS	D
CP-07	CallLogProvider (selection): Input validation on selection parameter (blind boolean; internal tables currently empty)	com.android.providers.contacts	GAP	D
CP-08	CarrierIdProvider (selection): Input validation on selection parameter (full UNION injection with 13-column matching; no permission required)	com.android.providers.telephony	GAP	D
CP-09	SortOrder injection (multi-provider): Input validation on sortOrder parameter (CASE WHEN subqueries in ORDER BY; 4 providers affected)	Contacts, Calendar, Settings, UserDictionary	GAP	D
CP-10	CellBroadcastContentProvider: Input validation on query parameters (no setStrict; projection/selection/sortOrder injection possible; gated by READ_CELL_BROADCASTS)	com.google.android.cellbroadcastreceiver	GAP	S
CP-11	CalendarProvider2 (instances): Input validation on instances query (setStrict + setStrictColumns + setStrictGrammar all enabled unconditionally)	com.android.providers.calendar	HOLDS	S+D
CP-12	MediaProvider: sortOrder validation (properly rejects "Invalid token SELECT")	(system)	HOLDS	D
CP-13	DownloadProvider: sortOrder validation (properly rejects "Invalid token SELECT")	(system)	HOLDS	D
CP-14	LegacyStorageBackendContentProvider: Permission enforcement on exported provider (query(): zero checks; openFile()/call(): gated by Phenotype flag 45793750, defaults false)	com.google.android.apps.docs (Drive)	GAP	S+D
CP-15	LegacyStorageBackendContentProvider: Same shared library vulnerability as CP-14	com.google.android.apps.docs.editors.docs (Docs)	GAP	S+D
CP-16	LegacyStorageBackendContentProvider: Same shared library vulnerability as CP-14	com.google.android.apps.docs.editors.trix (Sheets)	GAP	D
CP-17	LegacyStorageBackendContentProvider: Same shared library vulnerability as CP-14	com.google.android.apps.docs.editors.slides (Slides)	GAP	D
CP-18	StorageBackendContentProvider: Permission enforcement on SAF provider (MANAGE_DOCUMENTS, signature	com.google.android.apps.docs (Drive)	privileged)	HOLDS

Appendix table, continued

ID	Component and boundary tested	Package or scope	Outcome	Evid.
CP-19	LegacyStorageBackend openFile(): Path traversal via URI manipulation (database ID lookup, not filesystem path construction; UriMatcher blocks multi-segment traversal)	(Drive/Docs/Sheets/Slides)	HOLDS	S
CP-20	DumpFileProvider: Export control on debug provider (serves contacts database ZIP; no permission)	com.android.providers.contacts	GAP	D
CP-21	SuggestionStateProvider: Permission enforcement (no permission, no getCallingPackage() check; exposes screen lock and biometric enrollment status)	com.android.settings	GAP	D
CP-22	AvatarContentProvider: URI validation on proxy provider (inverted allowlist proxies any content:// URI through Messages' READ_CONTACTS)	com.google.android.apps.messaging	GAP	D
CP-23	MmsSmsProvider: Write permission declaration (readPermission=READ_SMS but no writePermission; zero-perm app can delete all SMS/MMS)	com.android.providers.telephony	GAP	D
CP-24	GoogleSettingsProvider: Read permission declaration (writePermission set but no readPermission; leaks 18 partner settings including privacy prefs)	com.google.settings	GAP	S+D
CP-25	ServiceStateProvider: targetSdkVersion gating on permission checks (targetSdk < 31 bypasses both READ_PRIVILEGED_PHONE_STATE and location permission)	com.android.phone	GAP	D
CP-27	GServicesProvider: Permission protection level adequacy (READ_GSERVICES is normal, auto-granted; 1404 config entries including 3 API keys, 86 URLs, 10 certs)	com.google.android.gsf	GAP	D
CP-31	BluetoothOppProvider: SQL injection defense (setStrict(true) + path-permission)	com.google.android.bluetooth	HOLDS	S
CP-32	IccProvider, SimPhonebookProvider: Permission enforcement (READ/WRITE_CONTACTS required)	com.android.phone	HOLDS	S
CP-34	ContactsProvider2 zero-perm variant: URI grant scope (ACTION_PICK contact picker grants temporary URI but SQLi from that URI accesses ALL internal tables)	com.android.providers.contacts	GAP	D
B3: BroadcastReceivers (11 evaluations, all gaps)				
BR-01	GmsExternalReceiver (FRP): Per-action caller validation (FrpUpdateIntentOperation writes FRP secret without verification)	com.google.android.gms	GAP	D
BR-02	GmsExternalReceiver (CryptAuth): Per-action caller validation (AuthZenEventHandler triggers ForceRegistration of 6 security keys)	com.google.android.gms	GAP	D
BR-03	GmsExternalReceiver (systemic): Architectural pattern: single receiver, 31 actions, no android.permission, no per-action validation	com.google.android.gms	GAP	D
BR-04	TestingToolsBroadcastReceiver: Build-time stripping of debug code (receiver exported in production; gated by runtime Phenotype flag, not build-time removal)	11 Google apps (Gmail, Dialer, Files, Keep, etc.)	GAP	S+D
BR-05	CalendarDebugReceiver: Export control on debug receiver (accepts SECRET_CODE broadcast)	com.android.providers.calendar	GAP	S+D
BR-06	TestReceiver: Export control on test receiver (no permission, no build-type guard; forwards SIMULATE_STATE to privileged StateService)	com.google.android.apps.scone	GAP	S+D
BR-07	MediaOutputDialogReceiver: Protected broadcast declaration (LAUNCH_MEDIA_OUTPUT_DIALOG NOT declared as protected; sibling LAUNCH_SYSTEM_MEDIA_OUTPUT_DIALOG IS; no validation of package_name/user_handle extras)	com.android.systemui	GAP	S+D
BR-08	CrisisNotificationBroadcastReceiver: Authorization for safety-critical notification dismissal (any app can dismiss earthquake/tsunami/active shooter alerts)	com.google.android.apps.safetyhub	GAP	S+D
BR-09	TelephonyBroadcastReceiver: Permission enforcement path (unpermissioned receiver directly calls DMIntentReceiver.onReceive() as Java method, bypassing OS broadcast permission check)	com.android.omadm.service	GAP	D
BR-10	GmsCoreDelegateReceiver + NotificationBroadcastReceiver: Authorization for notification management (CANCEL_NOTIFICATIONS suppresses fraud alerts; NotificationBroadcastReceiver proxies arbitrary PendingIntents)	com.google.android.apps.walletnfcrel	GAP	D
BR-11	RcsProvisioningBroadcastReceiver: Permission protection level (PROVISIONING_EVENT is normal, auto-granted; validates sender only for 1 of 7 actions)	com.google.android.apps.messaging	GAP	D
B4: Services (14 evaluations: 10 gaps, 4 holds)				
SV-01	MainAppMediaBrowserService: Caller validation in onGetRoot() (only blocks SystemUI; accepts all other callers; zero permissions for metadata + playback control)	com.google.android.youtube	GAP	S+D

Appendix table, continued

ID	Component and boundary tested	Package or scope	Outcome	Evid.
SV-02	BluetoothMediaBrowserService: Caller validation in onGetRoot() (accepts ALL callers with zero validation; returns BrowseTree.ROOT)	com.google.android.bluetooth	GAP	S
SV-03	126 exported services: Manifest-level permission enforcement (126 services exported with no android.permission; FRP service returns 56-byte config blob)	com.google.android.gms	GAP	S+D
SV-04	7 backup services: Permission enforcement (all exported=true, no permission; internal backup services correctly exported=false)	com.google.android.apps.photos	GAP	D
SV-05	GrpcEndpointService: Permission protection level (ACCESS_ENDPOINT_SERVICE is normal, auto-granted)	com.google.android.apps.messaging	GAP	D
SV-06	FederatedComputeManagingServiceImpl: Binder caller identity check (zero enforceCallingPermission/checkCallingOrSelf/getCallingUid in entire codebase; schedule()/cancel() accept caller-supplied identity)	com.google.android.federatedcompute	GAP	S+D
SV-07	KeyChainService (5 of ~50 methods): Per-method authorization (5 methods including CA enumeration lack caller check; 120 system CA aliases + DER certs readable)	com.android.keychain	GAP	D
SV-08	NfcService (notifyTestHceData, notifyHceDeactivated): Per-method authorization (2 methods lack NFC permission check; forged HCE APDU injection into card-emulation pipeline)	com.google.android.nfc	GAP	D
SV-09	OdpResultHandlingService: Permission enforcement (no permission; sibling correctly requires BIND_EXAMPLE_STORE_SERVICE; writes attacker-controlled tokens under spoofed ComponentName)	com.google.android.onddevicepersonalization.services	GAP	D
SV-10	LensService: Permission protection level (LENS_SERVICE is normal, auto-granted; telemetry suggests startActivity and PendingIntent creation on behalf of caller)	com.google.android.googlequicksearchbox	GAP	S
SV-11	FelicaServiceImpl: Binder caller UID check (getCallingUid() properly enforced)	com.google.android.apps.walletnfcrel	HOLDS	S
SV-12	ZeroPartyBinder (GMS Supervision): Caller validation (UID check + signature check; only GMS can bind)	com.google.android.gms.supervision	HOLDS	S
SV-13	NfcService (all other Binder methods): Permission enforcement (NFC, NFC admin, foreground checks all enforced)	com.google.android.nfc	HOLDS	S
SV-14	GMS HardwareInfoQueryService, GooglePowerService: AIDL-level authorization (SecurityException on unprivileged bind)	com.google.android.gms	HOLDS	D
B5: Custom URI schemes (18 evaluations, all gaps)				
URI-01	otpauth:// (TOTP 2FA secret keys)	com.google.android.apps.authenticator2 / MainActivity	GAP	D
URI-02	LPA://, com.android.euicc:// (eSIM activation codes, carrier OAuth codes)	com.google.android.euicc / QrDownloadActivity, RedirectUriReceiverActivity	GAP	D
URI-03	FIDO:// (FIDO2 authentication session data)	com.google.android.gms / QRBounceActivity	GAP	D
URI-04	contactkeys:// (E2E encryption key verification data)	com.google.android.contactkeys / MainActivity	GAP	D
URI-05	tez://, gpay:// (UPI payment data (payee VPA, amount))	com.google.android.apps.nbu.paisa.user	GAP	D
URI-06	genericidp://firebase.auth/ (OAuth authorization codes (Firebase Auth SDK))	5 apps (Recorder, BetterBug, DeviceUsageStudy, Scribe, GMS)	GAP	D
URI-07	gmail://label (Mail label names, search queries)	com.google.android.gm / PublicLabelDeepLinkV2	GAP	D
URI-08	com.google.android.wear.companion://, comgooglecast://, vnd.youtube.kids:// (OAuth authorization codes (5 endpoints))	3 apps (Wear Companion, Home, YouTube Kids)	GAP	D
URI-09	pix:// (Pix payment identifiers, amounts)	com.google.android.apps.walletnfcrel / DeepLinkActivity	GAP	D
URI-10	comgooglewallet:// (Card tokenization, Digital Car Key tokens)	com.google.android.gms / AddNewCardThroughBrowser-Activity, EntryPointDckActivity	GAP	D
URI-11	googlechrome:// (URLs including auth tokens, session IDs)	com.android.chrome / IntentDispatcher	GAP	D
URI-12	fitbit:// (Health/fitness deep links, auth flow params)	com.fitbit.FitbitMobile	GAP	D
URI-13	mpa:// (Purchase order IDs, promo codes)	com.google.android.apps.tips / DeepLinkGateway	GAP	D
URI-14	bbps://, bharatconnect://, upimeta:// (Biller IDs, payment amounts, VPAs)	com.google.android.apps.nbu.paisa.user	GAP	D
URI-15	searchlite://image/ (Private search-result image URLs)	com.google.android.apps.searchlite	GAP	D

Appendix table, continued

ID	Component and boundary tested	Package or scope	Outcome	Evid.
URI-16	safetyhub://medicalinfo (Medical info deep link params)	com.google.android.apps.safetyhub	GAP	D
URI-17	com.google.android.apps.youtube://oauth2redirect (OAuth authorization codes (missing state validation))	com.google.android.youtube / AccountLinkingActivity	GAP	S+D
URI-18	com.google.android.apps.gsa.gal://oauth2redirect (OAuth codes (shared library, all devices))	com.google.android.googlequicksearchbox / AccountLinkingActivity	GAP	S+D
B6: Platform security controls (10 evaluations, all gaps)				
SYS-01	ENFORCE_STRICT_QUERY_BUILDER (ChangeId 143231523) globally disabled on Android 17	All ContentProviders using this gate	GAP	S+D
SYS-02	ENFORCE_INTENTS_TO_MATCH_INTENT_FILTERS globally disabled on Android 17	All Activities with intent filters	GAP	D
SYS-03	BLOCK_NULL_ACTION_INTENTS globally disabled on Android 17	All intent receivers	GAP	D
SYS-04	ENFORCE_STRICT_SQL_CHECKS gated at targetSdk=37 (no Play Store app targets 37)	All ContentProviders	GAP	D
SYS-05	PARCEL_HARDENING gated at targetSdk=37	All Parcelable IPC	GAP	D
SYS-06	RESTRICT_DATA_URI_COLUMNS gated at targetSdk=37	ContentProvider URI grants	GAP	D
SYS-07	DETECT_IMPLICIT_URI_PERMISSION_GRANT gated at targetSdk=37	URI permission grants	GAP	D
SYS-08	Settings Tile Parcelable write/read mismatch (CVE-2023-45777 regression)	Bundle-based IPC via AccountManager, notifications, activity results	GAP	S+D
SYS-09	TLPE via InCallController createPackageContext flag 3 (CVE-2026-49881)	system_server code loading	GAP	S
SYS-10	Maps AGMM JavaScript bridge openUrl()	JS-to-native intent redirect within Maps process	GAP	S
Positive controls: well-defended components (11 evaluations, all holds)				
NEG-01	Pixel Launcher (all providers): Signature permissions, SHA-256 allowlists, test-mode gating	com.google.android.apps.nexuslauncher	HOLDS	S
NEG-02	Google Meet: No intent forwarding pattern, no addJavascriptInterface	com.google.android.apps.tachyon	HOLDS	S
NEG-03	NFC stack (entire codebase): Flag 0 only, no FLAG_MUTABLE, all Binder methods enforce permissions, user confirmation for dangerous NFC actions	com.android.nfc, com.google.android.nfc, com.google.android.pixelnfc	HOLDS	S
NEG-04	ManagedProvisioning: Code-level caller validation (verifyActionAndCaller, getCallingPackage, checkDevicePolicyPreconditions, FRP check) compensates for missing manifest permission	com.android.managedprovisioning	HOLDS	S
NEG-05	com.android.phone: All providers require permissions, dangerous RadioInfo operations gated by mSystemUser and Build.isDebuggable()	com.android.phone	HOLDS	S
NEG-06	Gmail providers (10 tested): SecurityException, empty results, or not exported	com.google.android.gm	HOLDS	D
NEG-07	GMS ContentProviders: Not exported or require permissions	com.google.android.gms	HOLDS	D
NEG-08	FileProviders (Gmail, Wallet, Messages): Confirmed not exported	Multiple	HOLDS	S+D
NEG-09	TLPE variant hunt (Maps, Camera, Search, Recorder): All resolve to Google-signed packages (Android Auto SDK, VR Core, GMS dynamite); all in app context, not system_server	Multiple	HOLDS	S
NEG-10	DynamiteModule pattern (~20+ apps): Hardcoded GMS package name	Multiple	HOLDS	S
NEG-11	17 “dead ends” in Vulnerability Reward Program testing: Components properly secured or not exploitable	Multiple (16 apps)	HOLDS	D