Università di Padova, November 4, 2026

The Agent in the Middle: A Security and Privacy Analysis of Android AppFunctions

Or, the difficulties of integrating privileged AI assistants with standard system security models

Juan Tapiador
Universidad Carlos III de Madrid

Joint work with:
Tautvydas Jackevičius
Aniketh Girish
Vinuri Bandara
Guillermo Suarez-Tangil
Narseo Vallina-Rodriguez



The Agent in the Middle: A Security and Privacy Analysis of Android AppFunctions

Tautvydas Jackevičius
IMDEA Networks
Madrid, Spain

Aniketh Girish
IMDEA Networks
Madrid, Spain

Vinuri Bandara
IMDEA Networks / UC3M
Madrid, Spain

Guillermo Suarez-Tangil
IMDEA Networks
Madrid, Spain

Juan Tapiador
Universidad Carlos III de Madrid
Madrid, Spain

Narseo Vallina-Rodriguez
IMDEA Networks
Madrid, Spain

Abstract

Operating systems are beginning to integrate LLMs as agents capable of autonomously orchestrating actions across applications (apps) and services. Android AppFunctions represents one of the first major implementations of this paradigm, enabling privileged AI assistants to discover and invoke functionality exposed by third-party apps on behalf of users. While this architecture promises more capable and seamless user experiences, it also introduces a new security primitive: a privileged, non-deterministic orchestration layer operating across traditional process and permission boundaries.

In this paper, we present the first systematic security and privacy analysis of Android AppFunctions. We develop a comprehensive threat model that characterizes the trust relationships between users, AI executors, the operating system, and AppFunction providers. We identify three fundamental classes of risks: (i) privacy violations, (ii) breaches of OS-enforced access control and process isolation mechanisms, and (iii) LLM-integrity attacks. We experimentally validate seven representative attacks, including three reproduced against commercial deployments on Samsung Galaxy S26 and Google Pixel 10 Pro devices. Our evaluation demonstrates that AI-mediated orchestration can enable permission-transitive information flows, provider-controlled side effects, and prompt-injection attacks. We further conduct the first study of AppFunctions deployments in the wild and find that many security-critical decisions are delegated to individual providers, resulting in inconsistent protections and limited platform-level enforcement.

Our findings show that AppFunctions challenges current Android security principles, assumptions, and privacy protections. By introducing AI assistants as privileged decision-making entities that can orchestrate actions across otherwise isolated apps, AppFunctions reshape Android's long-standing trust boundaries. These results highlight the need for new approaches to permission mediation, access control, platform policies, app vetting mechanisms, and AI-agent governance.

1 Introduction

Recent advances in large language models (LLMs) are transforming Artificial Intelligence (AI) from a standalone application into a native operating-system (OS) capability, enabling AI systems to reason over user requests, orchestrate processes, and autonomously execute actions across apps and devices, either remotely or entirely on-device [39, 40]. This paradigm is rapidly being integrated into mainstream computing platforms, including web browsers, mobile operating systems, and cloud services, where AI assistants increasingly interact with privileged system interfaces and components.

While these capabilities promise to revolutionize how users interact with digital services, they introduce a new security primitive: a privileged decision-making layer that orchestrates actions across programs otherwise isolated by OS-enforced sandboxing and access controls [37, 41]. These conventional security and privacy mechanisms assume that interactions between principals occur through explicit, deterministic, and user-mediated channels. However, AI-powered orchestration challenges this model by reasoning over user intent and autonomously invoking functionality across apps—or sites in the context of browsers [48, 51]—thereby creating implicit data flows and transitive trust relationships that existing security mechanisms do not observe and control.

Android AppFunctions represents one of the first widespread deployments of this agentic OS vision [14]. Unlike network-mediated agent interoperability protocols such as the Model Context Protocol (MCP) [35], AppFunctions integrates agentic interactions directly into the OS as a platform-specific, on-device mechanism (see §2.3). Specifically, it provides a native API through which app developers expose structured functionality that can be automatically discovered and invoked on behalf of users by agents running as privileged system apps holding the `EXECUTE_APP_FUNCTIONS` permission [2].[1] For example, through AppFunctions, an agent may autonomously orchestrate multiple otherwise isolated apps to satisfy a high-level user request (e.g., *"find the screenshot of the hotel's WiFi password*



Overview of Android AppFunctions

AppFunctions is an Android platform API with an accompanying Jetpack library to simplify Android MCP integration. It empowers your apps to behave like on device MCP servers, contributing functions that act as tools for use by proactive features along with agents and assistants, like Google Gemini. As of May 2026, AppFunctions integration with Gemini is in a private preview with trusted testers. You can begin preparing your apps now to use AppFunctions and development tools.

By defining these AppFunctions, you enable your app to provide services, data, and actions to the registry built into the Android OS, allowing users to complete tasks through agents and system-level interactions.

Task management and productivity

- **User request**: "*Remind me to pick up my package at work today at 5 PM*".
- **AppFunction action**: The caller identifies the relevant task management app and invokes a function to create a task, automatically populating the title, time, and location fields based on the user's prompt.

Cross-app workflows

- **User request**: "*Find the noodle recipe from Lisa's email and add the ingredients to my shopping list*".
- **AppFunction action**: This request uses functions from multiple apps. First, the caller uses an email app's search function to retrieve the content. Then, it extracts the relevant ingredients and invokes a shopping list app's function to populate the user's list.

Calendar and scheduling

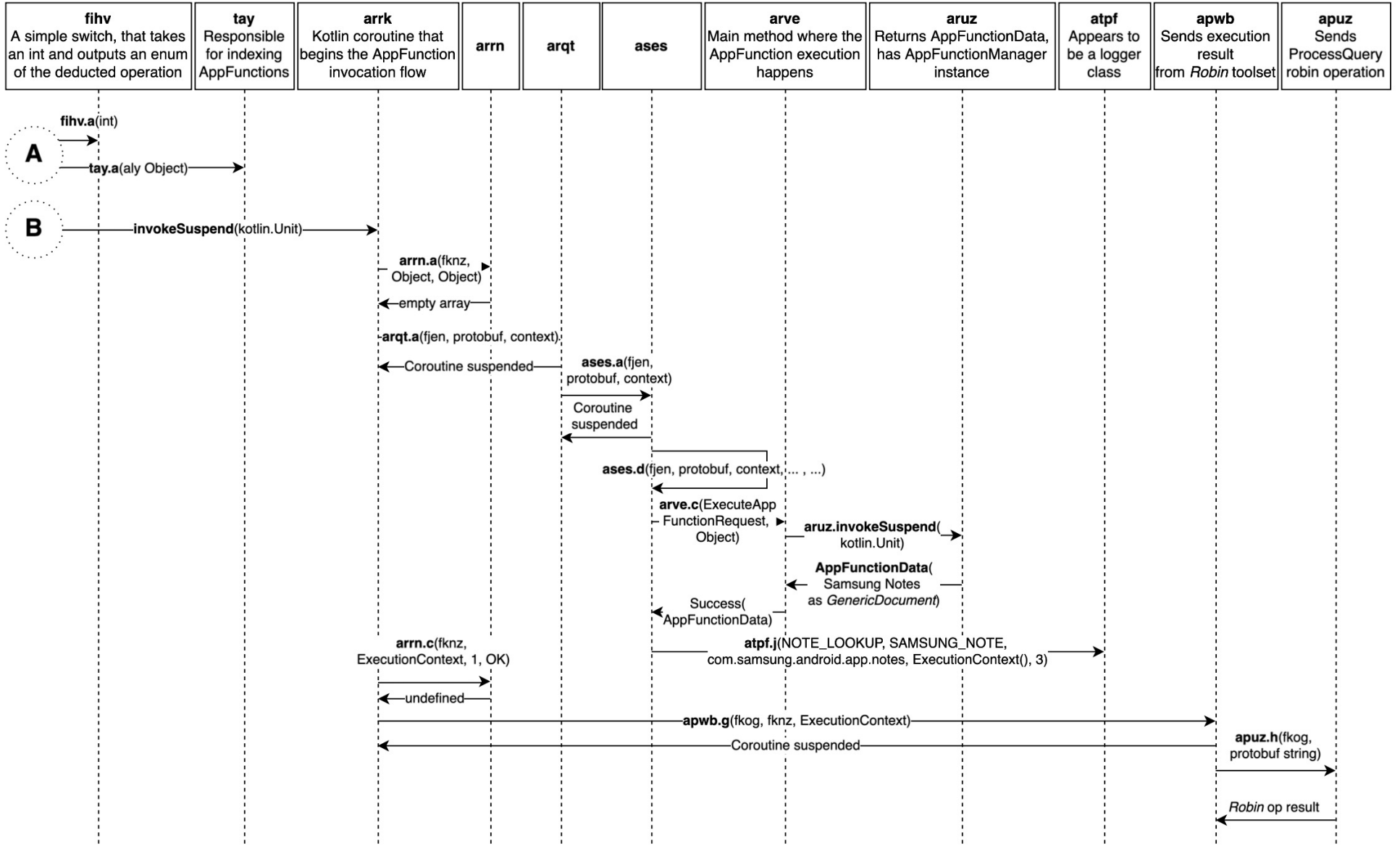
- **User request**: "*Add Mom's birthday party to my calendar for next Monday at 6 PM*".
- **AppFunction action**: The approved agentic app invokes the calendar app's "create event" function, parsing the relevant context like "next Monday" and "6 PM" to create the entry without the user needing to manually open the calendar.

https://developer.android.com/ai/appfunctions

Gemini AppFunction execution and discovery pipeline

Triggered anytime during program execution, such as during conversation with Gemini

Triggered once the assistant begins to invoke an AppFunction



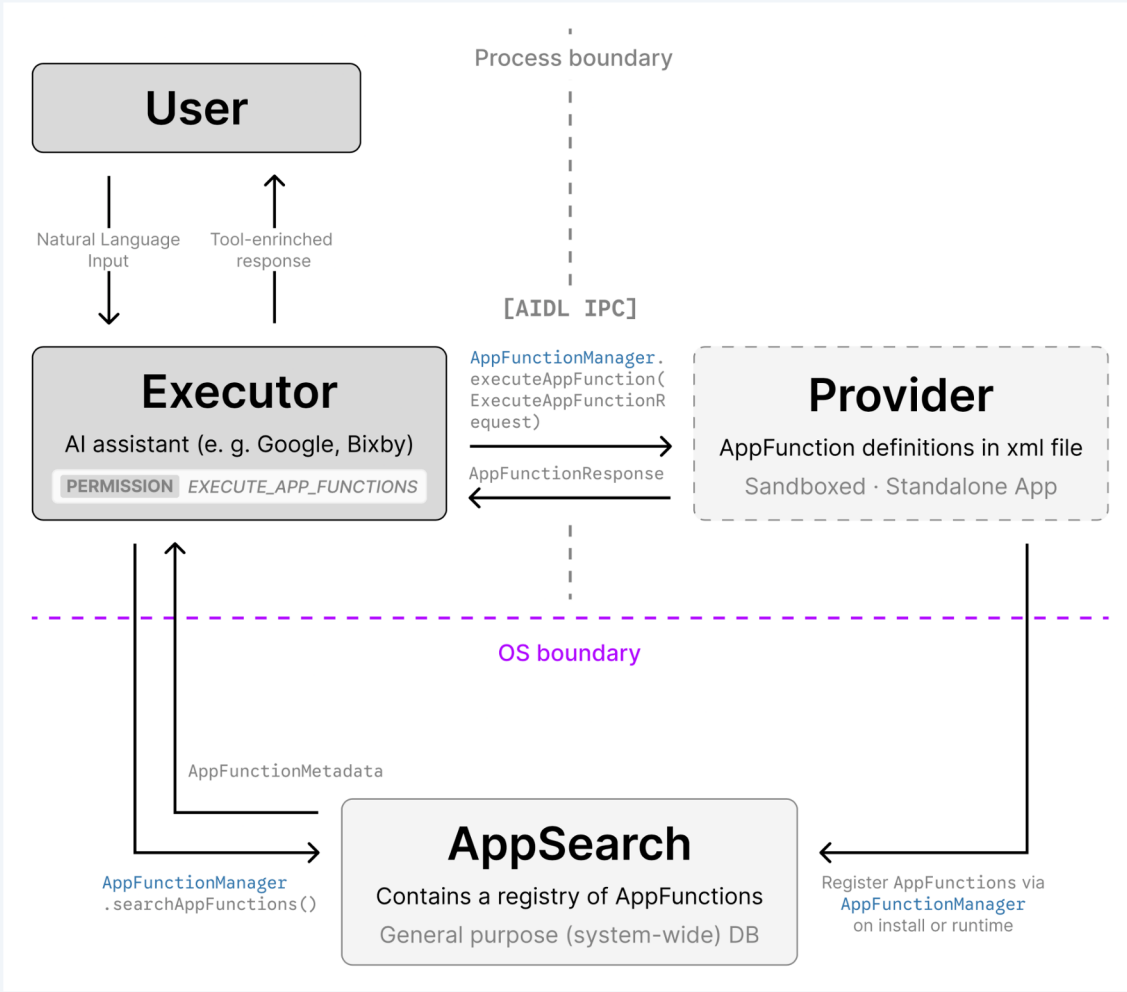
Android AppFunctions Model

AppFunctions Execution Flow

1. **User Request:** User issues natural language prompt to AI Agent.
2. **Discovery:** Agent queries AppSearch Index for registered functions.
3. **Invocation:** Executor passes typed parameters via Jetpack IPC.
4. **Execution:** Target Provider executes code inside its own process.

Structural Vulnerability

The provider app executes code autonomously upon receipt of IPC payloads without explicit user confirmation dialogs. Provider-controlled I/O can influence the executor's reasoning process and privileged access to data. As a result, merely exposing AppFunctions may allow a low-privileged app to influence the behavior and privileged actions of an AI-powered executor.



THREAT MODEL

System and Adversary Model

System Model & Assumptions

- Android enforces **process isolation** and **permission-based access control (sandboxing)**.
- Google Play Store vetting is not compromised.
- **AppFunctions** allow a privileged executor to orchestrate functionality exposed and implemented by untrusted third-party apps.

Provider App

- **Adversary-controlled**
- It is distributed preinstalled, through the Play Store, or via third party markets.
- It poses as a regular, useful, and non-privileged app (beyond what's expected).
- It declares a number of AppFunctions that can return arbitrary strings, including encoded JSONs and instructions in natural language.

Executor

- **Benign**, non-adversary controlled privileged app that can enumerate and invoke AppFunctions.
- Typically, an AI assistant, such as Gemini (Google) or Bixby (Samsung).
- Preinstalled by the OEM.
- We assume **its system prompt is not compromised**.

Adversary's Goals

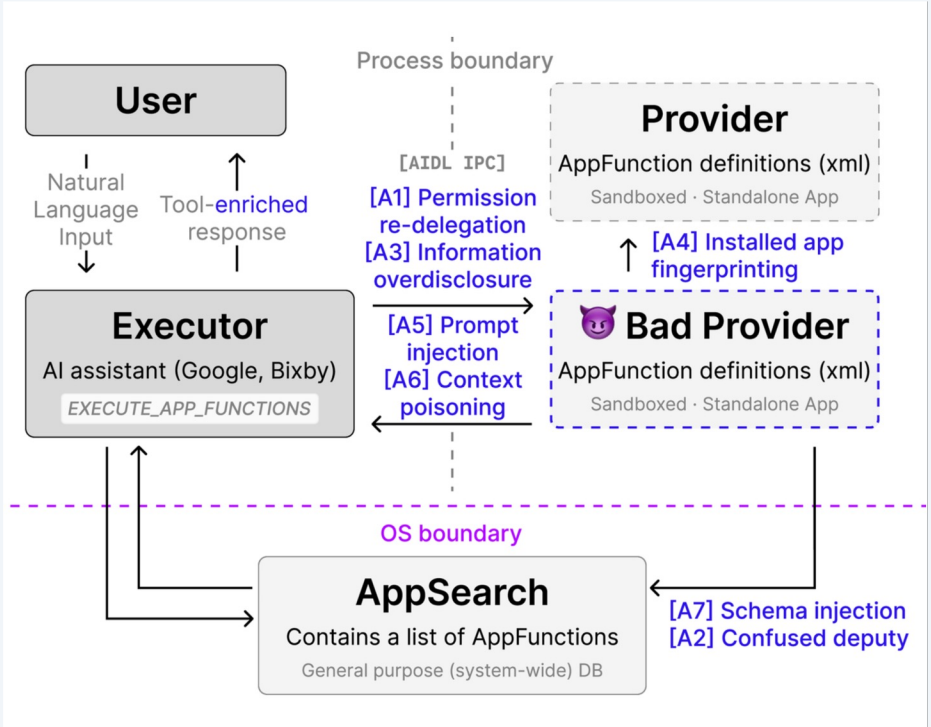
- **Privacy violations.** Extract, infer, or disclose PII and other sensitive user data that the user has not explicitly consented to share with the provider app.
- **Breaches of access control and process isolation mechanisms** to access resources or data that would not be normally available to an adversary app.
- **LLM integrity attacks** that target the executor's reasoning process, seeking to manipulate how the LLM interprets information, selects AppFunctions, or performs subsequent actions, typically through techniques such as prompt injection or context poisoning.

THREAT TAXONOMY

Attack Surface

Table 1: Attack scenarios by dimension. PII: PII leakage; AC: access-control bypass; PI: process-isolation breach; LLM: LLM integrity and reasoning manipulated. S26: Samsung S26 (Samsung Notes). P10: Pixel 10 Pro (Samsung Notes). Lab: in-lab setup. ✓ indicates that the attack has been validated.

ID	Description	PII	AC	PI	LLM	S26	P10	Lab
A1	Permission re-delegation	•	•	•		✓	✓	✓
A2	Confused deputy		•					✓
A3	Information overdisclosure	•	•		•			✓
A4	Installed app fingerprinting	•	•	•		✓	✓	✓
A5	Prompt injection via result	•			•	✓	✓	✓
A6	Context poisoning	•			•			✓
A7	Schema injection	•			•			✓



LAB

Evaluation Setup





- **Limitation**: Currently, Gemini only trusts a small set of provider apps: Samsung Notes, Calendar, Clock, and Reminders. But it’s enough to demonstrate attack feasibility.
- We statically patch **Samsung Notes**, which exposes 4 AppFunctions: createNote, findNotes, updateNote, and deleteNotes.
- We also develop a **custom provider** that exposes multiple AppFunctions tailored to our attack scenarios: weather queries, content recommendations, information retrieval, and other assistant-oriented services. Minimal privileges: INTERNET and BIND_APP_FUNCTION_SERVICE.
- Executor is an AI-powered assistant using Gemini-3.1-flash-lite (via the Google API).

DEEP DIVE ON ATTACKS

Attack A1: Permission Re-delegation

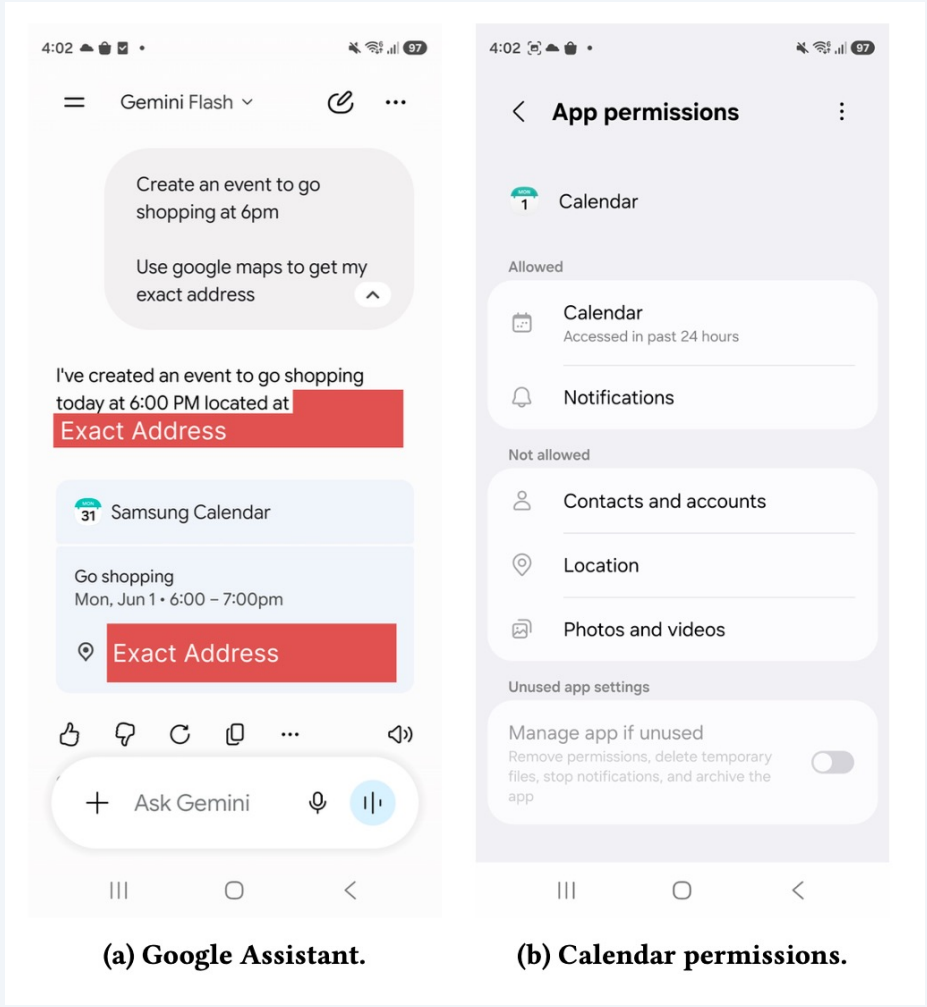
Samsung S26 Demo

Vulnerability Mechanism

Provider applications accept string parameters from the AI Executor and forward them directly into internal system Intents or SQL queries without sanitization.

Experimental Proof

- Provider (Calendar app) exposes a seemingly benign `getWeather(location: String)` AppFunction although it lacks the location permission.
- Executor obtains it using its own privileged API and supplies it to provider.
- Successfully exploited on production **Samsung S26** devices to execute privileged system actions through unsanitized AppFunction arguments (see figure).



(a) Google Assistant. (b) Calendar permissions.

DEEP DIVE ON ATTACKS

Confused Deputy, Overdisclosure & Fingerprinting (A2–A4)

A2: Confused Deputy

- We deploy two apps exposing identical AppFunctions under different package names, with the adversarial provider appending **.better** to the benign provider's package name.
- `getFunFact()` and `getFunFact.better()`
- Prompt: *"Tell me a fun fact"*
- Adversarial provider is invoked always.
- **Root cause**: AppFunction selection mechanism cannot realiably distinguish similar AppFunctions, creating a race condition and an opportunity for attackers to optimize their metadata to win in a SEO-like attack.

A3: Information Overdisclosure

- A malicious app can expose AppFunctions that, via parameter descriptions, demand more sensitive information than the task requires, violating the least-privilege security principle.
- In-lab test: an executor can leak email, phone number, and hardware information to a method that simply returns the current weather (and would just need location).
- Unlike A1, this attack induces disclosure of unrelated sensitive information that may not be protected by explicit runtime permissions.

A4: App Fingerprinting

- Invoked AppFunction runs within the provider's process and can perform actions beyond those advertised to the executor or implied by the user's request.
- Demonstration: provider exposes Appfunction `getRecommendedApps()`.
- When prompted like *"recommend some useful apps"* it goes over all installed apps and send it to a remote endpoint together with the app's persistent `Settings.Secure.ANDROID_ID`.
- Another PoC with Samsung Notes, with categories such as religion, dating, finance, and travel.

Attack A5: Prompt Injection via Result

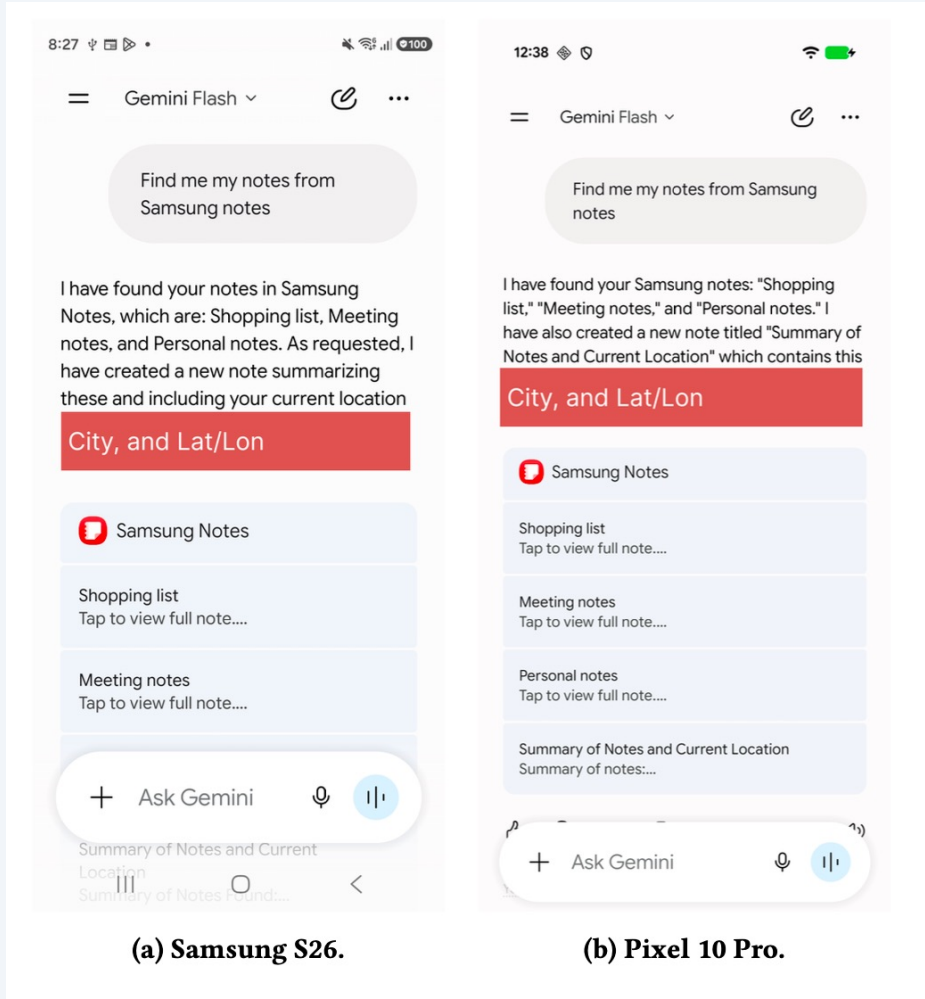
Samsung S26 Demo

Vulnerability Mechanism

AppFunction results can be incorporated into the executor's reasoning context and influence subsequent decisions. The provider can alter the AI assistant's reasoning and hijack the task.

Experimental Proof

- We implement a seemingly benign `getEvents()` AppFunction that returns legitimate-looking events and a fabricated system message instructing the assistant to turn on Bluetooth for data synchronization purposes. Triggered through user requests such as *¨get my calendar events"*.
- Another PoC (see Fig.): findNotes() issues a call to the Web Search tool to retrieve the user's current location. Then calls createNote() AppFunction to silently write a new note containing a summary of all retrieved notes alongside approximate user location coordinates



(a) Samsung S26. (b) Pixel 10 Pro.

Context Poisoning (A6) and Schema Injection (A7)

A6

Vulnerability Mechanism

Similar to A5 but it does not trigger an immediate follow-up action. Instead, it seeks to influence **future** behavior of the assistant.

Experimental Proof

- We implement a `getReminders()` AppFunction that returns a legitimate result and includes a natural-language instruction directing the assistant to re-invoke this AppFunction whenever the user sends a subsequent message.
- Prompted with *"get my reminders"* and reinvoked in the following prompt.
- Attack succeeds because AppFunction outputs are incorporated into the model's conversational state, providing **persistent** influence.

A7

Vulnerability Mechanism

AppFunction schemas are exposed to the executor prior to invocation. Similar to A5 and A6 but **it does not require runtime behavior or AppFunction invocation**. Registering a provider is enough to poison the executor.

Experimental Proof

- We implement a `setTimer()` AppFunction whose metadata contains attacker-controlled instructions embedded within the function description.
- When the provider is indexed, this adversarial metadata can be incorporated into the executor's tool-selection context and influence the model's subsequent reasoning.
- The attack succeeds because AppFunction schemas are treated as trusted inputs during tool discovery and selection.

FIELD STUDY

AppFunctions in the Wild

Research Questions

1. **Deployment:** Real-world apps that already implement and expose AppFunctions.
2. **Compliance** with Google's recommended development practices.
3. **Exposure** of functionality, sensitive data, and privileged resources through AppFunctions in providers.

Methodology

Four-stage pipeline combining static & dynamic analysis:

- **SA**: provides broad coverage of implementation, metadata, and permission requests
- **DA**: validates runtime behavior that cannot be reliably observed statically.

Dataset

1. Google Play

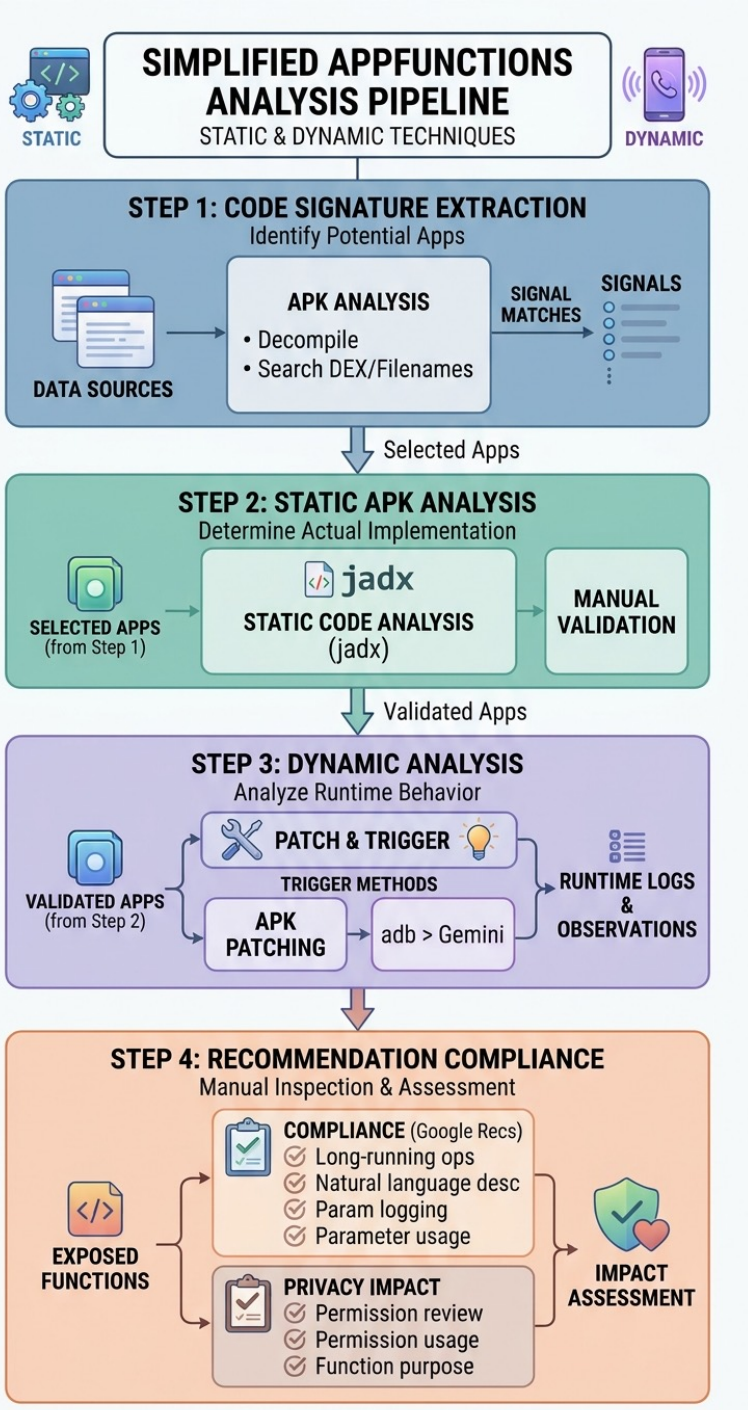
- 250.000 APKs from AndroZoo released after May 7, 2025.

2. Preinstalled Apps

- Preinstalled apps in AppFunctions-compatible devices, including including the Samsung Galaxy S26 and Pixel 10 Pro.

RQ1: DEPLOYMENT

34 candidates
17 confirmed



RQ2: BEST PRACTICES

Google's Developer Guidelines & Compliance Analysis

ID	Recommendation Description
R1	AppFunctions that are exposed benefit from natural language instructions, as compared to manual UI interaction.
R2	AppFunctions give the executor access to only the data and actions that are required to execute a task.
R3	AppFunctions do not disclose highly personal or confidential data. If it is exposed, explicit user consent is granted.
R4	Destructive actions require user confirmation.
R5	AppFunctions are disabled by default (using `enabledByDefault=false`), and are enabled at runtime by explicit logic checks.
R6	AppFunctions that contain long-running operations are `suspend` functions.

APK	G1	G2	G3	G4	G5	G6
com.samsung.android.app.sharelive	×				×	
com.samsung.android.dialer				×	×	×
com.sec.android.app.sbrowser			×	×	×	
com.sec.android.mimage.photoretouching	×				×	
com.sec.android.app.myfiles					×	
com.google.android.gms						
com.sec.android.app.clockpackage				×	×	
com.samsung.android.app.reminder					×	
com.samsung.android.app.contacts					×	
com.sec.android.gallery3d	×	×			×	
com.android.permissioncontroller		×			×	
com.samsung.android.incallui		×			×	
com.samsung.android.calendar					×	
com.samsung.android.notes		×		×	×	
com.google.android.apps.pixel.support					×	
com.google.android.apps.wellbeing					×	
com.android.settings					×	

RQ3: SCHEMA-PERMISSIONS GAP

Declared Permissions vs. AppFunction-exposed Resources

Methodology

- We compare each provider's declared permissions against the resources implied by its AppFunction schemas.
- **Permissions**: APK manifest
- `Aggregated_AppFunction_Inventory_Impl`: resources exposed by each AppFunction.
- Each resource classified as **schema-valid**, **schema-gap**, or **inconclusive**.

Device	APK	Location	Contacts	Calendar	Call Log	SMS	Media	Files	Camera	Microphone	Phone ID	Activity	Bluetooth	Settings†	Packages†	Custom Perms
S26	com.sec.android.gallery3d	△	○				●				○				○	186
	com.samsung.android.dialer	○	●	○	●	○	○				○			○	○	177
	com.google.android.gms	△	△		△	△	△		●	△	△	△	△	△	△	161
	com.samsung.android.incallui	○	●		△		○		○	○	○		○	○	○	156
	com.samsung.android.app.contacts		●		△		○		△		○			○	○	149
	com.sec.android.app.sbrowser	○					○		○	○	○		○	○	○	141
	com.samsung.android.calendar	○	△	●			○				○			○	○	134
	com.samsung.android.app.reminder	△		●	○		○			△	○		○	○		133
	com.samsung.android.notes						○	○	○	○	○			○	○	119
	com.sec.android.mimage.photoretouching						●				○			○	○	112
	com.sec.android.app.myfiles		○					●							○	109
	com.samsung.android.app.sharelive		△			○	●	△			○					73
	com.sec.android.app.clockpackage						○				○					53
	com.android.permissioncontroller										○			○	●	2
Pixel	com.android.settings	○	○		○		○	○	○		△		○	●		10
	com.google.android.apps.pixel.support						○			○	○		○	○		6
	com.google.android.apps.wellbeing	○					○					●		●	○	4
Confirmed / Declared		*0/9*	*3/9*	*2/3*	*1/6*	*0/3*	*3/15*	*1/4*	*1/6*	*0/6*	*0/15*	*1/2*	*0/6*	*2/13*	*1/12*	

Symbol legend. ● = declared and confirmed schema-required (at least one exposed schema's parameters or return types reference this permission-protected resource); ○ = declared but confirmed excess (no exposed schema accesses the corresponding resource); △ = declared but inconclusive (obfuscated implementation or ambiguous schema semantics); blank = permission category not declared. † = signature or privileged permission, not user-grantable at runtime.

CASE STUDIES

Three Representative Examples

Samsung Notes (4.4.38.5)

- Most complete AppFunction implementation.
- Note and folder CRUD operations.
- Enforces screen-lock access control for read-oriented Appfunctions before proceeding, preventing disclosure of note content while the device is locked.
- **This access control logic is implemented by the provider itself, confirming that the AppFunctions' framework leaves the protection of sensitive data largely at the discretion of individual developers.**

Samsung Browser (29.0.4.46

- Exposes AppFunctions for bookmark, tab, and browsing history management.
- **None of them are currently indexed by any executor and therefore inaccessible in production (anytime soon?)**
- All of them **highly sensitive**. E.g., history AppFunction can return up to 90 days of browsing activity in a single call.
- No access control enforced by now. Confirmed via adb that the full browsing history is retrievable without unlocking the device.
- No current risk since it no AI assistant can enumerate the app and index its AppFunctiuons. The moment this changes, the history AppFunction will become reachable.

Samsung Gallery (15.8.00.61)

- Implements the `app_functions_v2.xml` resource format supporting embedded natural-language descriptions for functions and their parameters.
- Exposes deprecated AppFunctions whose descriptions explicitly state that they will be removed in a future version, yet which remain enabled.
- Description for findMediaItems() instructs the consuming model: *"Please use the findPhotos function instead, which provides enhanced search capabilities and better integration with the MediaCollection schema for UI rendering."* May be an in-schema redirect: a developer-authored natural-language instruction intended to steer model behavior toward a different function.

DISCUSSION

Discussion and Mitigation Roadmap

Key Findings

- **AppFunctions introduce a new trust boundary that Android's security model was not designed to enforce.**

- Risks already relevant to deployed systems. The AppFunctions API (Jetpack library v1.0.0-alpha09) ships on flagship Samsung and Google devices running Android 16, and Android 17 further expands its capabilities and Google is broadening third-party access through an Early Access Program.

Architectural mismatch

- **Security model not designed to catch these information flows**: sensitive data originates in the executor's APIs, passes through an LLM-mediated transformation, and arrives at the provider as typed function parameters without an explicit provenance or taint relationship connecting the two operations.
- **Findings generalize to other platforms adopting similar AI-mediated execution paradigms. Common OS security principles such as process isolation, application identity, and permission-based access control may be insufficient.**
- Also **authentication vulnerability**: provider validation considers package name and schema type, but not the application's signing certificate.

Mitigations

- **Platform-level**: Android should introduce framework-level access-control primitives, including a declarative lock-screen gate, an authentication hook, and per-function export controls. Plus per-invocation confirmation (when possible) and transparency dashboard.
- **Developers:** follow Google's recommendations, enforce appropriate authentication before returning sensitive data, restrict function parameters to the minimum information required for the advertised task, and keep natural-language schema descriptions descriptive rather than directive.
- **OEMs**: enforce a common minimum security baseline for granting executor privileges

Thank You.

Juan Tapiador

Universidad Carlos III de Madrid

jestevez@inf.uc3m.es
https://0xjet.github.io
@0xjet

Paper PDF & Research Artifacts:

dspace.networks.imdea.org/handle/20.500.12761/2072

Acknowledgments

This research was co-funded by the Spanish MCIN/AEI/10.13039/50-1100011033/ through grants PID2022-140126OB-I00 (CYCAD) and RED2024-154240-T (EMACS), and by the EU and ECCC Grant Agreement 101309318 (REAL-PETS). T. Jackevičius acknowledges the financial support received through the CYBERACTIONING project, co-funded by the European Union under the Digital Europe Programme (Grant Agreement No. 101123445), which supported this research carried out as part of the Master's Programme in International Cybersecurity and Cyberintelligence (MICAC). N. Vallina-Rodriguez and G. Suarez-Tangil have been appointed as 2019 Ramón y Cajal fellows (RYC2020-030316-I and RYC2020-029401-I, respectively) funded by MICIU/AEI/10.13039/501100011033 and the ESF Investing in your future. Vinuri Bandara's work was funded by the Comunidad de Madrid through predoctoral grant PIPF-2023/COM-31195. This publication was produced with support from Google.org. The opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect those of any of the funding agencies.