AttestLens: A Large-Scale Measurement of Play Integrity Adoption in Android Apps

Collin MacDonald
cmacdonald01@wm.edu
William & Mary
Williamsburg, Virginia, USA

Stephen Herwig
smherwig@wm.edu
William & Mary
Williamsburg, Virginia, USA

Abstract

Google's Play Integrity framework has replaced SafetyNet as the primary mechanism for app, device, and account attestation on Android, yet its real-world adoption remains poorly understood. We present AttestLens, a large-scale measurement of attestation usage across 125,421 APKs collected from AndroZoo in April 2025. Our dataset spans six app categories—popular, banking, cryptocurrency, government, gaming, and random—and includes a longitudinal set of 814 government applications.

We develop robust static markers to detect SafetyNet and Play Integrity, even under common obfuscation techniques, and use them to quantify adoption by category and by app characteristics such as downloads, release date, and rating. Our findings are concerning: roughly 90% of apps in every category reference neither framework, and adoption remains particularly low in security-sensitive areas such as banking, cryptocurrency, and gaming. Additionally, many government apps continue to reference SafetyNet despite its deprecation. Across 6,281 Play Integrity-invoking applications, we find that most attestation originates from third-party packages rather than first-party invocation. On a positive note, most invoking apps follow Google's recommendation of pairing Play Integrity with other defense-in-depth techniques.

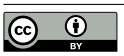
CCS Concepts

• **Security and privacy → Domain-specific security and privacy architectures**; **Mobile platform security**; • **General and reference** → *Measurement*; • **Human-centered computing** → *Ubiquitous and mobile computing*.

Keywords

Play Integrity, SafetyNet, remote attestation, Android, mobile security, measurement study, defense in depth

ACM Reference Format:
Collin MacDonald and Stephen Herwig. 2026. AttestLens: A Large-Scale Measurement of Play Integrity Adoption in Android Apps. In *Proceedings of the 19th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '26), June 30-July 03, 2026, Saarbrücken, Germany*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3765613.3811674



WiSec '26, Saarbrücken, Germany

ACM ISBN 979-8-4007-2201-1/2026/06
https://doi.org/10.1145/3765613.3811674

1 Introduction

Attestation frameworks play a central role in Android security by gating access to sensitive features such as financial transactions, account recovery, and anti-abuse controls. Google's Play Integrity API [27] has replaced the deprecated SafetyNet Attestation API [18] and is now the recommended mechanism for verifying app, device, and user integrity before processing high-value actions. In public developer communications, Google has suggested that apps adopting Play Integrity features have an 80% reduction in unauthorized usage [9], underscoring the framework's intended role in strengthening anti-abuse protections.

In principle, Play Integrity offers stronger guarantees than its predecessor. It ties app integrity to Play-distributed packages, leverages hardware-backed security signals where available, and exposes richer verdicts on device state, Google Play Protect, and potentially abusive behavior. Google has also fully shut down SafetyNet as of January 2025, making Play Integrity the only first-party option for Google Play apps. In practice, however, we do not know how widely Play Integrity is adopted or the nature of apps that choose to rely on it.

These questions matter for both security and platform design. If adoption is low in high-risk categories (e.g., banking, crypto, government, and games with strong cheating incentives), then the ecosystem may not be getting the benefits Google intended. Whereas an earlier work, SafetyNOT [33], examined how apps misused the older SafetyNet API, our focus is on ecosystem-wide adoption and usage of Play Integrity—a framework Google explicitly designed to eliminate such misuse. SafetyNet's deprecation also raises a natural migration question: when a core security service disappears, do developers update their apps, remove checks, or simply leave broken code in place?

We address these questions with AttestLens, a large-scale measurement study of attestation framework usage in Android apps. Using AndroZoo [2, 4], we construct a dataset of 125,421 unique APKs with Google Play metadata, spanning six categories: popular, banking, cryptocurrency, government, gaming, and a random sample of apps. We also collect all historical versions of 814 government apps, enabling a longitudinal view of migration behavior around the SafetyNet deprecation. We identify static markers for SafetyNet and Play Integrity that survive common obfuscation, and we validate them on hand-built sample apps and decompiled binaries.

Applying these markers, we first quantify adoption: whether an app references either attestation framework, and if so, which one—or both. We then relate adoption to app characteristics such as download count, initial release date, and average star rating. We find that roughly 90% of apps in each category include neither SafetyNet nor Play Integrity, and that Play Integrity adoption is heavily

skewed toward highly downloaded "popular" apps, with banking, crypto, gaming, and government apps lagging behind. Among government apps, SafetyNet usage persists well past Google's end of support, and only a small fraction migrate to Play Integrity.

Next, we examine usage: how Play Integrity is actually invoked in code. Decompiling 7,334 APKs that contain Play Integrity-related strings, we distinguish between first-party and third-party usage and identify common integration patterns. We find that a majority of Play Integrity references reside solely in third-party packages. Finally, we explore Play Integrity's role in defense-in-depth, measuring co-occurrence with other hardening techniques such as root detection, runtime integrity checks, and certificate pinning.

Contributions. We make the following contributions:

- **Large-scale adoption measurement.** We present the first in-depth analysis of Play Integrity and SafetyNet adoption across 125k Android APKs, covering six app categories plus a longitudinal corpus of government apps.
- **Adoption factors and migration behavior**. We quantify how Play Integrity usage correlates with downloads, release date, and rating; and we characterize real-world migration away from SafetyNet in government apps.
- **Identification of third-party usage.** We analyze how developers integrate Play Integrity in practice, and provide strong evidence that third-party libraries (e.g., Firebase, reCAPTCHA) drive Play Integrity's adoption.
- **Defense-in-depth assessment.** We examine how Play Integrity is used with other mobile hardening techniques.

2 Background

2.1 SafetyNet

Google introduced the SafetyNet Attestation framework in 2017 to strengthen Android's anti-abuse and device integrity protections. SafetyNet enabled developers to verify that an app was running on a genuine, non-rooted Android device and whether the app had been tampered with, such as through repackaging. According to Google, its goal was to help servers "distinguish traffic coming from genuine, compatible Android devices from traffic coming from less-trusted sources" [18].

SafetyNet operated through a secure process separate from the app itself: the *Google Mobile Services (GMS)* component performed integrity checks and sent the results to Google's servers, which returned a signed attestation for developers to verify on their backends. Correct implementation, however, proved difficult, as SafetyNet required secure server-side nonce generation and verification; apps that mistakenly performed these steps on the client were vulnerable to bypasses [33]. Google officially deprecated SafetyNet in June 2022 [40] and fully ceased support in January 2025 [19].

2.2 Play Integrity

Play Integrity [27], introduced in February 2022, builds on and strengthens SafetyNet by providing more comprehensive, developer-friendly, and tamper-resistant attestation. In addition to detecting rooted or modified devices, it verifies that user actions originate from an untampered app installed via Google Play[1] on a certified device, and reports broader risk signals such as missing security updates, active overlay or screen-capture apps, disabled Play Protect, and anomalous request patterns [26]. Like SafetyNet, Play Integrity gathers integrity evidence on the device and submits it to Google's servers, which return a signed and encrypted token containing verdicts about the app, device, and user. By leveraging hardware-backed security signals and server-side verification, Play Integrity reduces opportunities for client-side tampering and misconfiguration while enabling app servers to evaluate client trustworthiness.

Play Integrity offers two API modes—*Classic* and *Standard*—that differ in both usability and security tradeoffs. The Classic API [23], available on Android 4.4 and later, performs a fresh integrity assessment for each request and uses a developer-generated nonce to bind the attestation to a specific action. While this can reduce the time-of-check-to-time-of-use (TOCTOU) window, it incurs higher latency and requires developers to manage replay protections correctly. The newer Standard API [24], introduced in July 2023, is designed for low-latency, high-frequency checks. It leverages on-device caching of integrity signals, replaces the nonce with a cryptographic *requestHash* that binds the token to a specific server request, and delegates replay protections to Google Play infrastructure, reducing integration complexity.

3 Methodology

3.1 Data Source

We collect mobile applications and associated metadata for AndroTestLens from the AndroZoo dataset [2, 4] maintained by the University of Luxembourg. We focus exclusively on Google Play apps because Play Integrity's verdicts are most informative for Play-distributed binaries, and AndroZoo provides detailed Play Store metadata (e.g., star ratings, see §4.3) for these entries. Since not all APKs originate from Google Play and some Play-based records lack metadata, we first merge and normalize these sources. To capture SafetyNet's introduction and analyze migration patterns—particularly in our longitudinal study of government apps—we include APKs with upload dates as early as 2017.

The resulting dataset contains verified APK hashes and Google Play metadata for over 13 million Android apps. Each APK hash corresponds to a unique `(package_id, version)` pair. When duplicates occur—such as for multiple language builds—we retain only the most recently updated APK hash for each pair.

3.2 App Categorization

Downloading all 13 million APKs would have required over 100 TB of storage, making a full download impractical. Instead, we filtered the normalized dataset into six manageable categories: *(1)* popular, *(2)* banking, *(3)* crypto, *(4)* government, *(5)* gaming, and *(6)* a

[1] Play Integrity is not restricted to Play-Store-distributed apps: the API only requires an associated Google Cloud project [29], so apps distributed through third-party stores or direct download may invoke it. Non-Play installs receive `UNRECOGNIZED_VERSION` (app integrity) and `UNLICENSED` (licensing) verdicts rather than `PLAY_RECOGNIZED` / `LICENSED` [28]. Play Store availability is required only to link the project to the Play Console and to request daily quota increases beyond the default 10,000 requests [29].

Table 1: App categories used in AttestLens, along with associated metrics.

Category	Filter Criteria	App Version	Valid APKs	Invalid APKs
Popular	1,000,000+ Downloads	Latest	69,312	7
Banking	Package name contains `bank`	Latest	12,847	0
Crypto	Package name contains `crypto`	Latest	4,371	0
Government	Package name starts with `gov.`	Latest	814	0
Games (partial)	Published game	Latest	20,000	0
Random	N/A	Latest	20,000	3
Government (longitudinal)	Package name starts with `gov.`	All	4,271	0

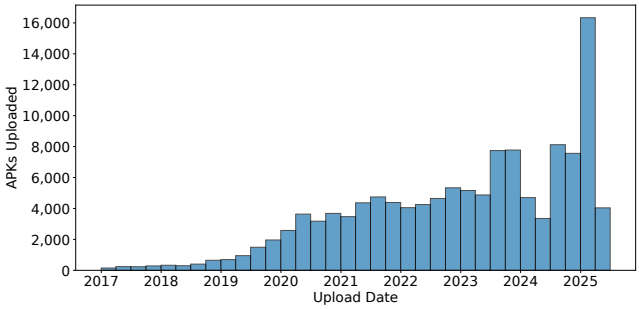


Figure 1: Histogram of APK uploads by date.

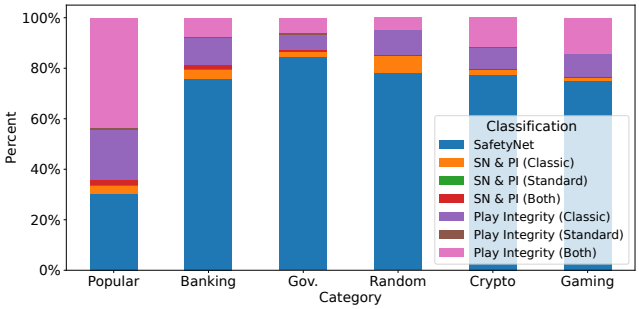


Figure 2: Percentage of attestable applications by category using an attestation framework.

random sample of apps.[2] These categories capture a broad range of applications, particularly those handling sensitive or financial data. Table 1 summarizes the selection criteria and statistics; apps may appear in multiple categories. For longitudinal analysis, we download all available versions of each government-published app.

After filtering, we downloaded 125,421 unique APKs in April 2025, with only 10 failing verification;[3] 1,923 appeared in multiple categories. For string-based categories, we validated classification accuracy using random samples sized for approximately 95% confidence and ±5% margin of error. Our manual review yielded 100% accuracy for crypto (58 sampled apps) and government (56 apps), and 96.6% precision for banking (59 apps, 2 misclassifications). The APKs span uploads from 2017 to 2025 (Figure 1).

4 Adoption

Since its release in early 2022, the Play Integrity framework has been widely discussed in developer blogs and online forums [10, 11, 30, 46]. While past studies provided limited insights into Play Integrity adoption, none focused on the details of Play Integrity's global adoption [43]. In this section, we examine the adoption of Play Integrity across over 125,000 unique Android apps, including key factors that correlate with Play Integrity usage and broader adoption trends.

[2]Google Play's native category taxonomy is not available in our metadata, and security-relevant populations such as government apps span multiple Play categories (e.g., Finance, Education, Tools).

[3]Each downloaded APK was hashed and compared to the hash provided by AndroZoo. If a downloaded APK's hash did not match the expected value, we attempted to redownload it up to 3 times before considering it `failed` and excluding it from future analysis.

4.1 Detecting Framework Adoption

To detect the use of SafetyNet and Play Integrity in downloaded APKs, we first need to understand how each framework appears in compiled apps. We thus build a set of sample Java applications that use either SafetyNet, Play Integrity with the Standard API, Play Integrity with the Classic API, or no attestation framework at all. We compile each sample in both debug and release modes, and with and without code obfuscation via the R8 compiler [22]. Using the `strings` command-line tool, we extract and examine framework-related strings from each build to identify distinctive markers (see Appendix A) that persist even under obfuscation. We then use these markers to statically identify SafetyNet and Play Integrity usage in downloaded APKs without requiring decompilation. We call an app that has such a marker an *attestable app*.

4.2 Adoption by Category

Play Integrity adoption varies widely by application type, popularity, and publisher. Overall, most apps use neither SafetyNet nor Play Integrity. Among those that do, Figure 2 shows the proportion of attestable apps in each category, with Table 2 providing the corresponding counts. The following sections describe each category in detail.

Popular apps. Popular apps—those with the highest download counts—show the highest rate of migration from SafetyNet to Play Integrity ($\chi^2(1, N = 13,883) = 2,719, p < 0.00001, V = 0.44$). In total, 13.3% of popular apps reference at least one attestation framework, similar to other categories. However, unlike other categories, most attestable popular apps adopt Play Integrity: 69.7% reference either the Classic or Standard API, the highest rate among all app

Table 2: Adoption by category. (The attestation columns are mutually exclusive.)

Category	Total Apps	No Attestation	SafetyNet	PI			SafetyNet + PI (Either)
				Classic	Standard	Both	
Popular	69,312	60,100 (86.7%)	2,788	1,800	92	4,005	527
Banking	12,847	11,521 (89.7%)	1,005	143	6	98	74
Crypto	4,371	3,957 (90.5%)	320	35	0	48	11
Government	814	710 (87.2%)	88	6	1	6	3
Games	20,000	19,003 (95.0%)	748	93	1	141	14
Random	20,000	18,170 (90.9%)	1,433	175	1	88	133

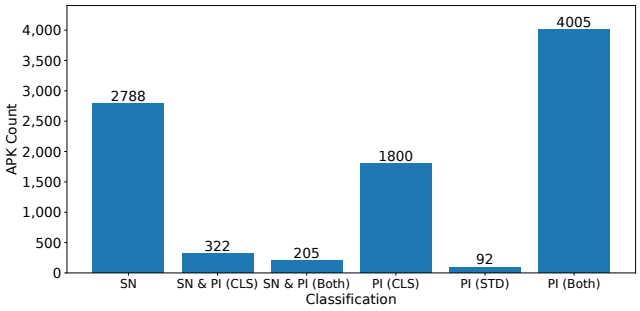


Figure 3: Play Integrity adoption among attestable popular apps.

groups. Figure 3 shows a detailed breakdown of framework usage for this category.

Banking apps. Despite being highly regulated and handling sensitive financial data, banking apps show limited use of Play Integrity: 89.7% contain no references to either SafetyNet or Play Integrity. Among the small subset that include attestation, 75.8% rely exclusively on the now-defunct SafetyNet. We infer that these apps either do not rely on SafetyNet to secure core functionality, or implement fail-open logic to prevent the shutdown of SafetyNet from affecting end users of the application.

Crypto apps. Crypto apps exhibit a nearly identical pattern: 90.5% do not use any attestation framework, and among those that do, 77.3% depend solely on SafetyNet. As with banking apps, Play Integrity adoption remains rare despite the security-sensitive nature of these services.

Government apps. Government-published apps show somewhat higher adoption: 12.8% reference at least one attestation framework. A multi-signal classifier[4] marks 666 (81.8%) of these apps as US-published and 145 (17.8%) as non-US (3 unknown); attestation rates are nearly identical across the split (12.8% vs. 12.4%), indicating the `gov.` filter's US skew does not distort the reported adoption rate. Among the government apps in our corpus, 84.6% rely only on SafetyNet, making this category the most SafetyNet-dependent of all.

[4]Signals include publisher address, domain and package-name ccTLDs, developer metadata, Unicode script, and US-specific markers (`.gov`, state names, municipal phrases).

Gaming apps. Gaming apps have the lowest adoption overall, with 95.0% containing no references to either SafetyNet or Play Integrity ($\chi^2(1, N = 127,344) = 855, p < 0.00001, V = 0.08$). This suggests that, despite Play Integrity support in Unity and Unreal Engine, most game developers are not using attestation for anti-cheating protection. Moreover, Unity- and Unreal-based apps account for only 17.7% (3,543 of 20,000) of APKs in the games category, so even universal SDK-level adoption would leave the majority of games uncovered.

Random apps. Randomly sampled apps from Google Play show similarly low adoption: 90.9% lack any reference to attestation frameworks, and among attestable apps, 78.3% use only SafetyNet. This is somewhat expected, as many apps—especially those without networked or server-backed functionality—have little need for attestation.

Adoption Insights by Category

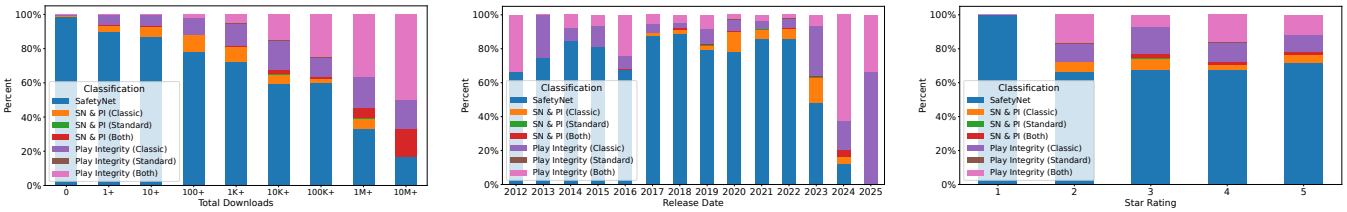
- ~90% of applications do not include SafetyNet or Play Integrity.
- Among attestable applications, popular applications adopt Play Integrity at the highest rate (69.7%).
- Less than 5.0% of games published on the Google Play Store include SafetyNet or Play Integrity.

4.3 Adoption Factors

In this section, we examine how download count, release date, and average star rating relate to an app's adoption of SafetyNet or Play Integrity. We focus on the *random* category, noting that AndroZoo metadata includes these attributes for many—but not all—of our 20,000 *random* apps. Figure 4 summarizes our findings among the 1,830 attestable apps in this category.

Total downloads. Of the three factors, total downloads is the strongest predictor of Play Integrity adoption among attestable apps. As Figure 4(a) shows, 20% of apps with even 1,000 lifetime downloads include Play Integrity, and more than half of apps with at least one million downloads include Play Integrity (see Table 3). This trend is intuitive, as apps with more users are likely to generate more revenue, and therefore place a higher value on security features.

Release date. Google introduced Play Integrity in February 2022; we examine whether apps first published before and after this point include the framework at different rates. As Figure 4(b) and Table 4 show, apps released since 2023 include Play Integrity far



(a) Adoption by Total Downloads (b) Adoption by Release Date (c) Adoption by Star Rating

Figure 4: Factors that influence adoption of Play Integrity among attestable *random* apps.

Table 3: Adoption by total downloads (*random* apps).

Total Downloads	Total Apps	No Attestation	SafetyNet	PI			SafetyNet + PI (Either)
				Classic	Standard	Both	
0	945	817 (86.5%)	126	1	0	0	1
1+	2,864	2,630 (91.8%)	210	15	0	0	9
10+	4,678	4,267 (91.2%)	358	27	0	1	25
100+	4,786	4,352 (90.9%)	338	42	0	9	45
1,000+	3,576	3,247 (90.8%)	237	44	0	17	31
10,000+	2,000	1,825 (91.3%)	104	30	1	26	14
100,000+	832	752 (90.4%)	48	9	0	20	3
1,000,000+	255	222 (87.1%)	11	6	0	12	4
10,000,000+	57	51 (89.5%)	1	1	0	3	1
100,000,000+	7	7 (100.0%)	0	0	0	0	0

more frequently than old apps ($\chi^2(1, N = 18,971) = 485, p < 0.00001, V = 0.16$). Older apps—those first published between 2012 and 2016—also show modest adoption. In contrast, apps released during SafetyNet's peak years of support (2017–2022) exhibit the lowest uptake.

Star rating. Play Integrity adoption shows little relation to an app's overall star rating (see Table 5; $\chi^2(4, N = 5,278) = 2.58, p = 0.63$). Although Figure 4(c) shows some variation between ratings, it is minimal for apps with an average rating above 1.5.[5] This result is not unexpected, as Play Integrity (and SafetyNet) are not inherently user-facing features.

Adoption Insights by Factor

- Apps with high download counts and recent release dates show the strongest adoption of Play Integrity.
- An app's star rating has little relation to its adoption of Play Integrity.

4.4 Longitudinal Study: SafetyNet Migration in Government Apps

Google's retirement of the SafetyNet framework creates an opportunity to examine how developers respond to major service deprecations in mobile apps. Developers maintaining SafetyNet-based apps faced three choices: migrate to Play Integrity, remove SafetyNet entirely, or take no action. To understand how this transition played out in practice, we conduct a longitudinal analysis of attestation use in government-published apps.

We analyze more than 4,000 APKs spanning the 814 government applications and track which frameworks appear across different app versions. As Figure 5 shows, adoption gradually shifts from SafetyNet toward Play Integrity, but overall progress remains slow. By April 2025, only 15.4% of attestable government apps reference Play Integrity—well below the 21.7% observed in our random-app category, though only modestly supported by the data ($\chi^2(1, N = 1,934) = 2.33, p = 0.13, V = 0.03$). Alarmingly, we identify 90 apps that continue to reference SafetyNet without adopting Play Integrity, 7 that remove SafetyNet without adding any replacement, and only 9 that migrate from SafetyNet to Play Integrity. A potential cause for the lackluster response is that more than half of all government apps have not been updated in over two and a half years (see Figure 6).

SafetyNet Migration Insight

Few government apps have adopted Play Integrity, despite prior SafetyNet usage.

5 Usage

In this section, we move beyond detecting the mere presence of SafetyNet or Play Integrity and examine how Play Integrity is actually used, and when it is not. We narrow our analysis from the 125,421 APKs studied in §4 to the 7,334 apps flagged as containing Play Integrity strings, which we decompile to inspect their code.

[5]Star rating buckets were defined as 1 (1.0 to 1.5), 2 (1.5 to 2.5), 3 (2.5 to 3.5), 4 (3.5 to 4.5) and 5 (4.5 to 5.0).

Table 4: Adoption by release date (*random* apps).

Release Date	Total Apps	No Attestation	SafetyNet	PI			SafetyNet + PI (Either)
				Classic	Standard	Both	
2010	6	6 (100.0%)	0	0	0	0	0
2011	34	34 (100.0%)	0	0	0	0	0
2012	57	54 (94.7%)	2	0	0	1	0
2013	125	117 (93.6%)	6	2	0	0	0
2014	189	176 (93.1%)	11	1	0	1	0
2015	340	324 (95.3%)	13	2	0	1	0
2016	574	549 (95.6%)	17	2	0	6	0
2017	1,015	940 (92.6%)	66	4	0	4	1
2018	1,438	1,348 (93.7%)	80	3	0	4	3
2019	2,747	2,584 (94.1%)	130	15	0	13	5
2020	4,059	3,859 (95.1%)	156	13	1	5	25
2021	3,766	3,393 (90.1%)	321	18	0	12	22
2022	3,086	2,564 (83.1%)	448	30	0	10	34
2023	1,370	1,130 (82.5%)	116	72	0	15	37
2024	155	131 (84.5%)	3	4	0	15	2
2025	10	7 (70.0%)	0	2	0	1	0

Table 5: Adoption by star rating.

Star Rating	Total Apps	No Attestation	SafetyNet	PI			SafetyNet + PI (Either)
				Classic	Standard	Both	
1	37	34 (91.9%)	3	0	0	0	0
2	174	156 (89.7%)	12	2	0	3	1
3	724	681 (94.1%)	29	7	0	3	4
4	2,681	2,474 (92.3%)	140	23	1	33	10
5	2,112	1,938 (91.8%)	125	18	0	20	11

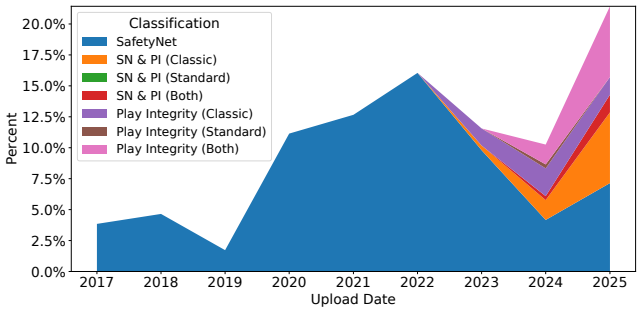


Figure 5: % of Government APKs uploaded by year with an attestation framework.

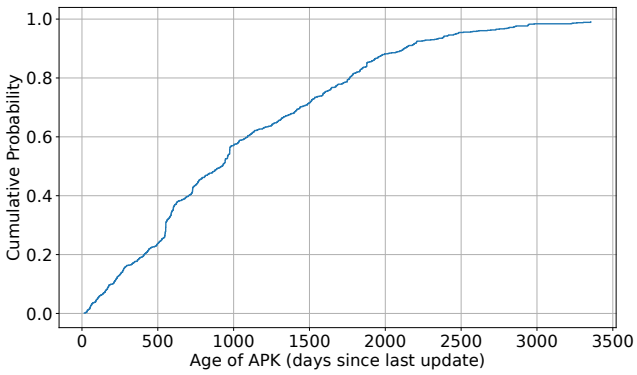


Figure 6: CDF of APK age for government apps.

5.1 Identifying Play Integrity Code

Our adoption analysis identified 7,334 apps containing Play Integrity strings. To study their usage, we decompile each APK with JADX [42], a tool that reconstructs Java source code from DEX and APK files. We decompile every app that we flagged as including Play Integrity in the first static-analysis phase (§4.1). For each decompiled APK, we scan all text-based files for an expanded set of Play Integrity markers (see Appendix B), based on the Android Developer documentation [23, 24]. Using the package names, we distinguish between *first-party* code—classes under the app's own package namespace, written by the developer—and *third-party* code—classes drawn from external libraries and SDKs bundled into the APK. We call an app that contains Play Integrity markers outside of the Play Integrity library code itself an *invoking app*. Apps which include the Play Integrity library code but do not invoke it are *non-invoking apps*.

Table 6: Where is Play Integrity code being used?

Metric	APKs
First-party	4,930
Third-party	6,982
Only First-party	341
Only Third-party	2,393
First-party & Third-party	4,589
Classic API	6,197
Standard API	1,492
Invoking Apps	6,281
Non-Invoking Apps	1,042

Table 7: Play Integrity in invoking applications.

API	First-party	Third-party	Both
Standard	22	9	53
Classic	111	1295	3383
Both	9	417	982

5.2 Code-level Results

After decompiling the 7,334 apps flagged as using Play Integrity, we successfully find Play Integrity library code in 7,323 of them—99.9% of the original set.[6] Table 6 summarizes high-level usage patterns. Only 341 apps use Play Integrity exclusively in first-party code. The majority (4,589 apps) combine first-party and third-party code, while 2,393 rely solely on third-party code. Figure 7 provides a detailed breakdown of Play Integrity code across all decompiled apps by API type (Classic vs. Standard), first-party vs. third-party code, and Play Integrity library location.

Invoking applications. We identified invocation-related Play Integrity code in 6,281 APKs. Table 7 contains key metrics on Play Integrity usage for all invoking applications for both the Classic and Standard APIs. Usage is dominated by the Classic API and third-party code. Usage of the Standard API without the Classic API is exceedingly rare among both first- and third-party codebases, and represents less than 1.4% of all invoking applications.

Non-invoking applications. Despite a variety of obfuscation and compilation techniques present in the decompiled APKs, we successfully detected code for the Play Integrity library in 7,323 APKs, of which 1,042 were non-invoking apps. In these apps, we detected the presence of the Play Integrity library without detecting corresponding invocation code—suggesting an incomplete implementation, accidental library inclusion, or other development-centric mistake. Non-invoking apps include the Play Integrity library in first-party code (199), third-party code (672), and both first- and third-party code (171).

Third-party code. Through manual inspection, we identify 22 distinct third-party packages containing Play Integrity-related code across both invoking and non-invoking apps (Table 8). Google-published libraries account for most third-party usage, followed by analytics and advertising frameworks. Of these 22 libraries, 20 use the Classic API and only 7 use the Standard API. In many cases, third-party usage primarily protects the SDK provider's backend (e.g., CAPTCHA or authentication services), benefiting the host app only when those services gate core workflows.

[6]Six apps contain none of the usage-related markers, while five apps appear to invoke Play Integrity without including the Play Integrity library code; we exclude these eleven apps from further analysis.

Table 8: Play Integrity findings in third-party code. Google-published packages appear in gray.

API Type	Package	APKs
Classic API	GMS	1,220
	reCAPTCHA	2,373
	Firebase (App Check)	434
	Firebase (Auth.)	3,880
	Firebase (misc.)	840
	Google (misc.)	889
	AdJoe	88
	AppsFlyer	510
	Daon	3
	FutureAE	6
	Games37	15
	GeoComply	5
	IdWall	5
	Protectt	13
	Radar.io	2
	RSA	2
	Shield Square	2
	Ticket Master	9
	VISA	14
	Yandex	71
Standard API	reCAPTCHA	1187
	Firebase (misc.)	2
	Google (misc.)	3
	AdJoe	52
	GG Incent	3
	Radar.io	5
	Trusting Social	1

Play Integrity Code-level Insights

- Among both first- and third-party code, usage of the Classic API is dominant.
- Over 30% of APKs include Play Integrity solely through third-party dependencies, so their inclusion may be incidental rather than a deliberate integration by the app developer.
- Google's Firebase Authentication package is present in over half of all APKs we analyzed.

6 Defense-in-Depth

Although Play Integrity is a valuable security mechanism, it is not intended to operate in isolation. As Google notes, it works best as part of a broader anti-abuse strategy [26]. In this section, we examine how often Play Integrity is used alongside three defense-in-depth mechanisms: *(1)* root detection, *(2)* runtime integrity checks, and *(3)* certificate pinning.

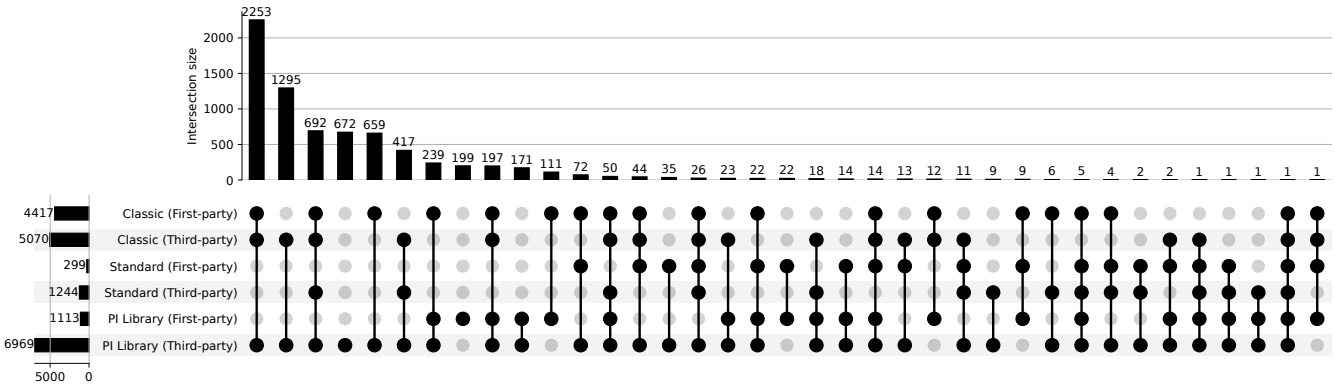


Figure 7: Play Integrity usage among invoking and non-invoking applications.

Root detection. Root detection is a widely discussed topic in Android security, along with methods to evade it [16, 44, 48]. A common approach checks for the presence of a *superuser* (su) binary. Other techniques include detecting non-release builds, inspecting system properties, and using root-detection libraries such as RootBeer [3].

Runtime integrity checks. Even if developers accept the risks of running their application in a rooted environment, they still may want to prevent users from directly debugging their application with tools such as `ADB` or `Frida` [15, 21]. By examining the system for evidence of such tools at runtime, or evidence of programs intended to hide such tools, applications can locally evaluate the system they are running on with no dependency on external servers.

Certificate pinning. For applications transmitting sensitive data over untrusted networks, TLS certificate pinning can limit the impact of a man-in-the-middle attack by explicitly stating what root certificates to trust for TLS connections. Android supports certificate pinning through a custom network security configuration [25], allowing an app to specify the hashes of trusted X.509 certificates. Because users have the ability to install custom Certificate Authorities (CAs) on their Android device, the use of certificate pinning is part of a robust defense-in-depth strategy for Android applications [20]. However, while supported at the system level, the Android Developer website explicitly advises caution when using certificate pinning due to the risk of future server changes breaking existing client-side apps [17].

6.1 Joint Adoption of Play Integrity and Defense-in-Depth

For each defense-in-depth mechanism, we develop a set of string-based markers and search for them in the decompiled APKs from the invoking apps identified in §5 (6,281 APKs). Our markers draw from the Android developer documentation and other website articles discussing detection (or evasion) strategies in Android applications [7, 14, 25, 35, 38].

Overall, 5,918 invoking apps (94.2%) combine Play Integrity with at least one hardening technique. Figure 8 displays how each technique is used across first- and third-party code, and Table 9 provides an aggregate summary.

Table 9: Hardening technique statistics.

Technique	APKs	%
Root Detection	5,682	90.5%
Runtime Checks	5,860	93.3%
Certificate Pinning	167	2.7%
None	363	5.8%
Only Root Detection	53	0.84%
Only Runtime Checks	231	3.7%
Only Certificate Pinning	2	0.032%

Unlike Play Integrity usage, which is dominated by third-party packages, usage of hardening techniques is widespread among both first- and third-party code. Root-detection and runtime checks are extremely common in APKs, and are adopted by the majority of applications we analyzed. Certificate pinning, despite going against current developer best practices, is still present in 167 applications.

Table 10 lists the most common markers for each of the three techniques. For root detection, superuser-related markers dominate, while debugger- and QEMU-related markers are most common for runtime integrity checks. In the case of certificate pinning, the vast majority of apps specify both *pin digest* and *pin-set* (as expected): 164 of the 167 specify both, and the remaining 3 specify only one of the two.

6.2 Defense-in-Depth Without Play Integrity

The preceding analysis measures hardening only in apps that invoke Play Integrity. We next ask whether apps that do not invoke Play Integrity instead rely on third-party hardening libraries as a substitute. We focus on banking apps because their heavy regulation and sensitive financial data give non-adopters the strongest incentive to compensate. Specifically, we compare defense-in-depth usage between the 321 banking apps that invoke Play Integrity and the 12,526 that do not (§3).

Invoking apps adopt every hardening technique more often: root detection (86.0% vs. 51.6%), runtime integrity checks (89.4% vs. 58.5%), and certificate pinning (10.3% vs. 2.9%), with risk ratios of 1.7, 1.5, and 3.6 (all $p < 0.00001$, χ^2). The same pattern holds

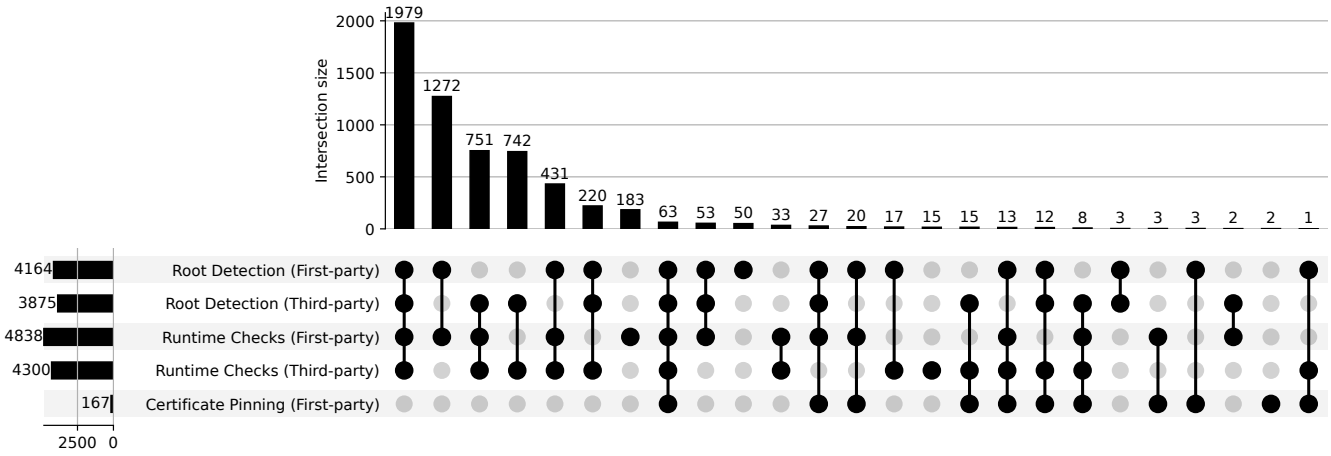


Figure 8: Hardening technique usage among invoking applications.

Table 10: Most commonly used hardening techniques among all invoking applications.

Technique	Marker	APKs	%
Root Detection	/system/xbin/su	5,579	88.8%
	/system/app/Superuser.apk	5,508	87.7%
	/system/bin/su	3,000	47.7%
	/sbin/su	2,913	46.3%
	/system/sd/xbin/su	2,891	46.0%
	com.noshufou.android.su	1,326	21.1%
	eu.chainfire.supersu	1,324	21.1%
	com.koushikdutta.superuser	1,293	20.6%
	com.thirdparty.superuser	1,290	20.5%
	com.yellowes.su	767	12.2%
	com.topjohnwu.magisk	761	12.1%
Runtime Checks	isDebuggerConnected	5,764	91.8%
	ro.kernel.qemu	2,098	33.4%
	de.robv.android.xposed	899	14.3%
	com.saurik.substrate	563	9.0%
	waitForDebugger	334	5.3%
	XposedBridge	188	3.0%
	frida-gadget	31	0.49%
Certificate Pinning	pin digest	166	2.6%
	pin-set	165	2.6%

in first- and third-party code separately. We find no evidence that defense-in-depth compensates for Play Integrity—non-adopters use hardening at lower rates than adopters.

7 Discussion

Usage of other attestation frameworks. In AttestLens, we examined how Android applications are shifting away from using SafetyNet and are beginning to adopt Play Integrity at scale. However, several device manufacturers provide their own attestation frameworks. Huawei offers the *Safety Detect SysIntegrity API* [32] for HarmonyOS and Android devices, allowing developers to detect rooted, unlocked, or vulnerable devices. Samsung provides *Knox Device Health Attestation* [41], a hardware-backed system for verifying a device's current security state. Xiaomi's *MIUI* operating system, based on Android, includes a pre-installed device certificate that enables attestation on Mi phones [47]. While this paper focuses only on Google's attestation frameworks, future work could also assess the adoption and usage of these manufacturer-specific frameworks.

Drivers of adoption gaps. The low and uneven adoption we observe reflects several compounding factors. First, the Play Integrity Standard API requires a developer-maintained backend to process attestation tokens, raising the bar for smaller developers and companies. Second, our longitudinal data suggests that many apps ship only infrequent updates. Because of this, even when Play Integrity becomes a desired feature, its adoption can take years. Third, much of the adoption we observe is indirect: applications frequently invoke Play Integrity through embedded third-party SDKs rather than first-party code (§5), suggesting that library choice—as much as deliberate security design—drives measurable adoption. Finally, our banking-category comparison (§6.2) finds no evidence that hardening compensates for Play Integrity; if anything, adopters of Play Integrity use hardening at higher rates than non-adopters.

Incidental security findings. Beyond studying Play Integrity in defense-in-depth strategies, we also incidentally identified several recurring security mistakes. A number of apps that use Play Integrity still leak sensitive information by embedding it directly in the client-side APK. In particular, we found 9 apps hardcoding non-test private keys (PEM-encoded) in their source code—primarily PKCS#8—and 12 apps including API keys (as for cloud services) labeled as `PRIVATE` or `SECRET` in their `.env` files.

Together, these apps account for more than 117 million Google Play downloads. These cases underscore that adopting a strong security framework like Play Integrity does not guarantee overall app security; careless handling of secrets elsewhere in the code can still compromise the entire app.

Study Limitations. Our static analysis—which we use for both adoption and usage measurements—relies on matching predefined

string markers in compiled and decompiled apps. While common in prior work, this approach trades precision for soundness and scalability. A small number of apps we identify as using an attestation framework may not actually do so at runtime due to obsolete code, developer mistakes, or unused library dependencies. Future work could incorporate dynamic analysis to observe whether apps request or verify attestation tokens, though doing so at scale remains challenging. Finally, our dataset consists solely of Google Play Store apps; attestation practices in non–Play Store ecosystems may differ.

8 Ethics

Throughout AttestLens, we identified several Android applications with apparent misconfigurations, implementation errors, or other security weaknesses. When these issues were discovered—whether related to SafetyNet, Play Integrity, or other mechanisms—we followed responsible disclosure practices and notified the corresponding developers. In total, we contacted the developers of more than 110 applications, primarily within government organizations, to share our findings. Fewer than 5% of the contacted developers replied, and those that did acknowledged our responsible disclosure and indicated that they had forwarded the report to the appropriate team.

9 Related Work

Attestation frameworks. Early work reverse-engineered SafetyNet [45], and SafetyNOT [33] provided the first large-scale analysis of SafetyNet usage, identifying common developer mistakes. Zhou et al. [49] conducted a similar study that also covered Huawei's Safety Detect framework. Recent survey work [37] situates Play Integrity within the wider landscape of Android security mechanisms. Steinböck et al.'s HALY [43] complements this line by statically and dynamically analyzing 2,646 popular apps shipped on both Android and iOS to measure the prevalence of eight hardening techniques recommended by the Mobile Application Security Verification Standard (MASVS) [39]. AttestLens differs from HALY in scope and focus: we analyze over 125,000 Android apps, target Play Integrity specifically rather than hardening broadly, and study not just prevalence but *how* developers integrate attestation with complementary anti-abuse techniques (root detection, runtime checks, and certificate pinning).

Large-scale measurements of mobile security. Our work builds on a broad literature that uses large-scale app analysis to evaluate security and privacy practices in the Android ecosystem. Enck et al. [12] conducted one of the earliest large-scale studies, decompiling 1,100 popular apps and revealing pervasive misuse of device identifiers and extensive reliance on embedded advertising and analytics libraries. The MalloDroid project [13] analyzed 13,400 apps and found that 8% used TLS incorrectly, leaving them vulnerable to man-in-the-middle attacks. Egele et al. [8] similarly demonstrated that 88% of apps misused cryptographic APIs in ways that undermined confidentiality and key strength. Many studies analyze the adoption and misuse of specific Android features, such as VPN services [34], native code [1], and biometric APIs [5]. Other analyses, such as MassVet [6], have focused on detecting repackaged or cloned apps commonly used to distribute malware. These studies highlight the value of ecosystem-scale measurements for understanding real-world security practices; our study extends this methodology to the domain of attestation frameworks.

Analysis infrastructure. Several systems aim to support large-scale dynamic analysis of Android apps, including platforms such as PUMA [31] and BareDroid [36], which provide automated execution environments for behavioral analysis, malware detection, or integration testing. While our study focuses on static detection of attestation usage, these dynamic-analysis platforms are complementary and could be used in future work to observe attestation-related events and how frequently apps trigger them.

10 Conclusion

In this work, we presented AttestLens, a large-scale study of Play Integrity adoption and usage across more than 125,000 Android APKs. Our findings show that adoption remains low across all application categories, including highly downloaded, highly rated, and security-sensitive apps. On a more positive note, adoption continues to increase year over year, and most developers who use Play Integrity use it as part of a broader defense-in-depth strategy.

Acknowledgments

We thank the AndroZoo team at the University of Luxembourg for providing access to the dataset used in this study. We also appreciate the helpful feedback from our anonymous reviewers and shepherd, Mathieu Cunche. This work was partially funded by the Virginia Space Grant Consortium.

References

[1] Vitor Afonso, Antonio Bianchi, Yanick Fratantonio, Adam Doupé, Mario Polino, Paulo de Geus, Christopher Kruegel, and Giovanni Vigna. 2016. Going Native: Using a Large-Scale Analysis of Android Apps to Create a Practical Native-Code Sandboxing Policy. In *Network and Distributed System Security Symposium (NDSS)*.

[2] Marco Alecci, Pedro Jesús Ruiz Jiménez, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. 2024. AndroZoo: A Retrospective with a Glimpse into the Future. In *Proceedings of the 21st International Conference on Mining Software Repositories*. 389–393.

[3] Scott Alexander-Bown. 2025. scottyab/rootbeer. https://github.com/scottyab/rootbeer original-date: 2015-06-19T09:01:55Z.

[4] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 468–471. doi:10.1145/2901739.2903508

[5] Antonio Bianchi, Yanick Fratantonio, Aravind Machiry, Christopher Kruegel, Giovanni Vigna, Simon Pak Ho Chung, and Wenke Lee. 2018. Broken Fingers: On the Usage of the Fingerprint API in Android. In *Network and Distributed System Security Symposium (NDSS)*.

[6] Kai Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Nan Zhang, Heqing Huang, Wei Zou, and Peng Liu. 2015. Finding Unknown Malice in 10 Seconds: Mass Vetting for New Threats at the Google-play Scale. In *USENIX Security Symposium*.

[7] Matthew Dolan. 2022. Android Security: SSL Pinning. https://appmattus.medium.com/android-security-ssl-pinning-1db8acb6621e

[8] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An Empirical Study of Cryptographic Misuse in Android Applications. In *ACM Conference on Computer and Communications Security (CCS)*.

[9] Dom Elliot. 2024. Making the Play Integrity API Faster, More Resilient, and More Private. https://android-developers.googleblog.com/2024/12/making-play-integrity-api-faster-resilient-private.html Accessed: 2026-3-03.

[10] ElyeProj. 2024. Play Integrity API, any potential issue of turning it ON? https://www.reddit.com/r/androiddev/comments/1fhupub/play_integrity_api_any_potential_issue_of_turning/

[11] emaayan. 2024. [HELP] is there a way to pass Strong Device Integrity by the play store? https://www.reddit.com/r/Magisk/comments/1c5pejq/help_is_there_a_way_to_pass_strong_device/
[12] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. 2011. A Study of Android Application Security. In *USENIX Security Symposium*.
[13] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory Love Android: An Analysis of Android SSL (In)security. In *ACM Conference on Computer and Communications Security (CCS)*.
[14] Ahmed Abdul Fatah. 2024. Android security for dummies: Root detection. https://medium.com/@ahmedafatah/android-security-for-dummies-root-detection-695bd4d90db8
[15] Frida. 2025. frida/frida. https://github.com/frida/frida
[16] Ioannis Gasparis, Zhiyun Qian, Chengyu Song, and Srikanth V. Krishnamurthy. 2017. Detecting Android Root Exploits by Learning from Root Providers. 1129–1144. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/gasparis
[17] Google, LLC. [n. d.]. Security with network protocols. https://developer.android.com/privacy-and-security/security-ssl
[18] Google, LLC. 2017. SafetyNet attestation, a building block for anti-abuse. https://android-developers.googleblog.com/2017/04/safetynet-attestation-building-block.html
[19] Google, LLC. 2025. About the SafetyNet Attestation API deprecation | Security. https://developer.android.com/privacy-and-security/safetynet/deprecation-timeline
[20] Google, LLC. 2025. Add & remove certificates. https://support.google.com/pixelphone/answer/2844832?hl=en
[21] Google, LLC. 2025. Android Debug Bridge (adb). https://developer.android.com/tools/adb
[22] Google, LLC. 2025. Enable app optimization. https://developer.android.com/topic/performance/app-optimization/enable-app-optimization
[23] Google, LLC. 2025. Make a classic API request. https://developer.android.com/google/play/integrity/classic
[24] Google, LLC. 2025. Make a standard API request. https://developer.android.com/google/play/integrity/standard
[25] Google, LLC. 2025. Network security configuration. https://developer.android.com/privacy-and-security/security-config
[26] Google, LLC. 2025. Overview of the Play Integrity API. https://developer.android.com/google/play/integrity/overview
[27] Google, LLC. 2025. Play Integrity API Library release notes. https://developer.android.com/google/play/integrity/reference/com/google/android/play/core/release-notes
[28] Google, LLC. 2025. Play Integrity API verdicts. https://developer.android.com/google/play/integrity/verdicts
[29] Google, LLC. 2025. Play Integrity Setup. https://developer.android.com/google/play/integrity/setup
[30] Google, LLC. 2025. Stronger threat detection, simpler integration: Protect your growth with the Play Integrity API. https://android-developers.googleblog.com/2025/10/stronger-threat-detection-simpler.html
[31] Shuai Hao, Bin Liu, Suman Nath, William G.J. Halfond, and Ramesh Govindan. 2014. PUMA: Programmable UI-automation for Large-scale Dynamic Analysis of Mobile Apps. In *ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*.
[32] Huawei. 2023. Safety Detect. https://developer.huawei.com/consumer/en/doc/Security-Guides/introduction-0000001050156325
[33] Muhammad Ibrahim, Abdullah Imran, and Antonio Bianchi. 2021. SafetyNOT: On the Usage of the SafetyNet Attestation API in Android. In *ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*.
[34] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar, and Vern Paxson. 2016. An Analysis of the Privacy and Security Risks of Android VPN Permission-enabled Apps. In *ACM Internet Measurement Conference (IMC)*.
[35] Indusface Pvt. Ltd. [n. d.]. How to Implement Root Detection in Android? https://www.indusface.com/learning/how-to-implement-root-detection-in-android-applications/
[36] Simone Mutti, Yanick Fratantonio, Antonio Bianchi, Luca Invernizzi, Jacopo Corbetta, Dhilung Kirat, Christopher Kruegel, and Giovanni Vigna. 2015. BareDroid: Large-Scale Analysis of Android Apps on Real Devices. In *Annual Computer Security Applications Conference (ACSAC)*.
[37] Arto Niemi, Vijay Nayani, Mariam Moustafa, and Jan-Erik Ekberg. 2023. Platform Attestation in Consumer Devices. In *Conference of Open Innovations Association (FRUCT)*.
[38] Artem Oppermann. [n. d.]. How to Detect Frida Toolkit Abuse in Your Mobile App. https://fingerprint.com/blog/exploring-frida-dynamic-instrumentation-tool-kit/
[39] OWASP Foundation, Inc. 2023. OWASP MASVS v2.0.0. https://github.com/OWASP/masvs/releases/tag/v2.0.0
[40] SafetyNet API Clients Team. 2022. Discontinuing the SafetyNet Attestation API. https://groups.google.com/g/safetynet-api-clients/c/ac_AmiRCn0U
[41] Samsung. 2025. Knox Device Health Attestation. https://docs.samsungknox.com/admin/fundamentals/whitepaper/samsung-knox-mobile-security/system-security/device-health-attestation/
[42] skylot. 2025. skylot/jadx. https://github.com/skylot/jadx original-date: 2013-03-18T17:08:21Z.
[43] Magdalena Steinböck, Jens Troost, Wilco Van Beijnum, Jan Seredynski, Herbert Bos, Martina Lindorfer, and Andrea Continella. 2025. SoK: Hardening Techniques in the Mobile Ecosystem — Are We There Yet?. In *2025 IEEE 10th European Symposium on Security and Privacy (EuroS&P)*. 789–806. doi:10.1109/EuroSP63326.2025.00050 ISSN: 2995-1356.
[44] San-Tsai Sun, Andrea Cuadros, and Konstantin Beznosov. 2015. Android Rooting: Methods, Detection, and Evasion. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, Denver Colorado USA, 3–14. doi:10.1145/2808117.2808126
[45] Romain Thomas. 2022. DroidGuard: A Deep Dive into SafetyNet.
[46] XDA. 2025. [GUIDE] ??? How to Pass Strong Integrity on Android (Step-by-Step Guide). https://xdaforums.com/t/guide-how-to-pass-strong-integrity-on-android-step-by-step-guide.4729435/
[47] Xiaomi. 2021. Hardware Trusted Environment. https://hasura.io/docs/miui-security-white-paper-global/2/1
[48] Hang Zhang, Dongdong She, and Zhiyun Qian. 2015. Android Root and its Providers: A Double-Edged Sword. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, Denver Colorado USA, 1093–1104. doi:10.1145/2810103.2813714
[49] Ziyi Zhou, Xuangan Xiao, Tianxiao Hou, Yikun Hu, and Dawu Gu. 2023. On the (In)Security of Manufacturer-Provided Remote Attestation Frameworks in Android. In *European Symposium on Research in Computer Security (ESORICS)*.

A Markers used in Adoption-based Analysis

To detect adoption of SafetyNet and Play Integrity at scale, we need to identify their inclusion in an un-extracted APK's binary. Table 11 shows the string we identified for each framework as being present in the APK binary on obfuscated release builds generated through Android Studio.

B Markers used in Usage-based Analysis

Post extraction, we analyze all text-based files in each APK for markers specific to Play Integrity. Table 12 shows the strings we use to identify each framework, based on the Android Developer documentation [23, 24].

Table 11: Adoption-focused API markers.

Framework	Marker
SafetyNet	`com.google.android.safetynet.ATTEST_API_KEY`
Play Integrity (Classic API)	`com.google.android.play.core.integrityservice.BIND_INTEGRITY_SERVICE`
Play Integrity (Standard API)	`com.google.android.play.core.expressintegrityservice.BIND_EXPRESS_INTEGRITY_SERVICE`

Table 12: Usage-focused API markers.

Framework	Marker
Classic API	`requestIntegrityToken`
	`decodeIntegrityToken`
	`IntegrityManager`
	`IntegrityTokenRequest`
	`IntegrityTokenResponse`
	`deviceRecognitionVerdict`
	`appRecognitionVerdict`
Standard API	`prepareIntegrityToken`
	`PrepareIntegrityTokenRequest`
	`StandardIntegrityManager`
	`StandardIntegrityTokenProvider`
	`StandardIntegrityTokenRequest`
	`StandardIntegrityToken`