Code Speaks Louder: Exploring Security and Privacy Relevant Regional Variations in Mobile Applications

Jiawei Guo
University at Buffalo, SUNY
jiaweigu@buffalo.edu

Yu Nong
University at Buffalo, SUNY
yunong@buffalo.edu

Zhiqiang Lin
The Ohio State University
zlin@cse.ohio-state.edu

Haipeng Cai✉
University at Buffalo, SUNY
haipengc@buffalo.edu

***Abstract*—Mobile apps are known to distribute different versions across geographic regions to accommodate local regulations and market preferences. While prior research has examined metadata-level differences such as permissions and privacy policies, there lacks systematic investigation into code-level geographic variations that may impact security. In this paper, we present the first comprehensive study of geo-feature differences (GFDs) in Android apps at the code implementation level. We develop FREELENS, a novel framework that overcomes key technical challenges including code obfuscation and analysis scalability to identify and characterize security-relevant variations across regions. Using FREELENS, we conducted a large-scale study of 21,120 Android apps distributed across ten countries with diverse levels of Internet freedom. Our findings reveal that GFDs are widespread, with significant variations in advertising, data handling, and authentication mechanisms. These differences frequently compromise security baselines and introduce disparities in privacy protections across regions. The study highlights a rising trend in GFD prevalence, emphasizing the urgency for harmonized privacy and security standards. Based on our empirical findings, we also provide actionable insights for developers, platform providers, and regulators to ensure equitable user protections.**

1. Introduction

It is known that mobile applications (apps) are distributed with different versions, which may exhibit different behaviors, across geographic regions, driven by factors ranging from regulatory compliance to market adaptation. For instance, investigations into popular messaging apps revealed varying levels of encryption and content filtering across countries [1], demonstrating how geographic variations can impact user privacy and information access.

Understanding these geographic feature differences (GFDs) and their implications for security and privacy is crucial for users, developers, and regulators alike. However, identifying and analyzing such differences systematically presents significant challenges. While these variations often stem from legitimate needs to comply with local regulations or adapt to market preferences, they can also mask concerning disparities in security protections and privacy guarantees.

Previous research has primarily approached this problem through analysis of app metadata and configurations. Kumar et al. [2] conducted the first large-scale study of geodifferences in mobile apps by examining application variations in permissions, privacy policies, and basic security features in Android across regions. Yang et al. [3] investigated differences between official and third-party app markets by analyzing app metadata and market policies. While these studies provide valuable insights into surface-level differences, they cannot capture the full spectrum of behavioral variations implemented at the code level.

In this paper, we fill this gap by conducting a large-scale, semantic *code-level* characterization of feature variations between country-specific versions of the same mobile app, referred to as *geo-feature differences* (**GFD**s). Examining GFDs at code level is essential because code implementations ultimately determine an app's actual behavior, potentially revealing security and privacy relevant variations that are not apparent or cannot be identified from metadata alone. For instance, two versions of an app might declare identical permissions but implement different data collection or processing logic, leading to varying privacy implications across regions. Through code-level differencing of diverse regional versions of apps with GFDs (referred to as **GFD apps**), the overarching goal of our study is to systematically demystify how mobile app vendors/developers may (stealthily) differentiate their application functionalities and what the functional variations imply in security and privacy.

However, conducting such a study at scale presents several critical challenges. First (**C1**), collecting representative app samples across regions is non-trivial due to geographic restrictions and version control mechanisms enforced by app distribution services (e.g., Google Play Store). Second (**C2**), it is well known that precise, fine-grained analysis of mobile apps (e.g., in Android) is costly hence often facing scalability challenges [4], [5], [6], [7], while surface-level analysis (e.g., of metadata [2]) only offers very limited understanding of real GFDs. Third (**C3**), for legitimate (e.g., compliance/protection) purposes, mobile apps are often obfuscated [8], which makes it difficult to identify meaningful differences between app versions. Finally (**C4**), connecting low-level code to high-level feature differences and further

✉ Haipeng Cai is the corresponding author.

to security/privacy understandings requires bridging significant semantic gaps.

To address these challenges, we developed FREELENS, an automated framework that combines strategic app collection, obfuscation-resilient code analysis, and semantic code-level analysis techniques. Our framework employs a three-phase approach, with each phase aiming to overcome each fundamental challenge. To overcome C1, Phase 3.2 conducts efficient mining of region-specific app versions using controlled Google Play Store accounts. To counter C2 and C3, Phase 3.3 balances analysis scalability and effectiveness through profiling method-level control flow in apps, while identifying meaningful code differences through an obfuscation-resilient call-path analysis. To address C4, Phase 3.4 employs interpretation of these differences using large language models to bridge the semantic gap between code changes and security implications. This approach enables us to analyze geographic variations systematically while maintaining scalability and accuracy.

Using this framework, we conducted a large-scale measurement study of 21,120 Android apps across ten countries representing different levels of internet freedom. Our investigation examines how apps implement geographic variations, what patterns these variations follow, and what security and privacy implications they carry. The study pays particular attention to highly-popular apps, where geographic variations can affect millions of users worldwide.

Our study reveals several important findings. **First**, our analysis revealed that GFDs are widespread, with over 5% of the examined apps exhibiting feature disparities between country-specific versions. These differences span various categories, including advertising strategies, user interface design, authentication mechanisms, and system management. Notably, advertising and monetization emerged as the most prevalent category, reflecting the strong influence of regional market dynamics. **Second**, we observed critical security and privacy implications arising from these differences. For instance, data security and privacy issues were the most common, often linked to regional variations in data handling practices. Similarly, disparities in advertising implementations, such as varying levels of user tracking and behavioral profiling, highlighted privacy risks specific to certain regions. These findings suggest that regional adaptations frequently compromise consistent privacy protections and security baselines. **Third**, the study highlighted trends indicating an increasing prevalence of GFDs in recent years, particularly in apps released after 2020. This rise underscores the growing complexity of regional customization in the mobile app ecosystem, raising the stakes for ensuring equitable security and privacy standards.

The implications of our findings extend to multiple stakeholders. For developers, our results highlight the need for more systematic approaches to managing geographic customization while maintaining consistent security standards. For platform providers, our findings suggest opportunities for improved tools and guidelines to help developers track and evaluate security implications of geographic variations. For users, our study reveals the importance of understanding potential security and privacy differences in apps based on their geographic location.

This paper makes the following key contributions:

- We conducted the first systematic study of code-level geographic feature differences in Android apps and their security implications.
- We developed FREELENS, a novel framework for identifying and analyzing security-relevant geographic variations in Android apps.
- We performed comprehensive characterization of security/privacy implications in geographic app variations, supported by detailed case studies.
- We provided actionable insights for improving the security consistency of apps across geographic regions.

We have released our code and datasets to facilitate relevant future research, as found at https://figshare.com/s/e68f2b0a7247192cc909.

2. Background and Motivation

In this section, we provide the relevant background for and then motivate our study.

2.1. Google Play App Release Mechanisms

The Google Play Store utilizes advanced app distribution mechanisms, allowing developers to tailor application releases for specific geographic regions [9]. Developers can target production, open testing, or closed testing tracks for country-specific app rollouts, forming the foundation of our investigation into cross-regional app functionality variations, particularly in security and privacy features.

Android apps adhere to a dual-versioning system: *version name* and *version code*. The version name, a user-facing identifier (e.g., "2.1.4"), follows semantic patterns and appears in the Google Play Store to communicate updates. The version code, an internal integer identifier, ensures proper sequencing for app updates. While invisible to users, version codes are pivotal for Android to manage app version precedence.

Google Play's model ensures users can only access the *latest version* available for their region. This constraint facilitates compliance with regional regulations but creates scenarios where users in different regions receive distinct app versions. These mechanisms enable developers to customize app functionality based on geographic requirements, laying the groundwork for geographic feature differences (GFDs), the focus of our study.

2.2. Android App Code Analysis

Analyzing Android apps requires sophisticated tools to reveal functional differences at the code level. Tools like FlowDroid [4] enable static taint analysis and precise modeling of Android's lifecycle, making it a critical asset for identifying security and privacy issues.

However, app code analysis faces challenges from obfuscation techniques like name obfuscation, control flow modification, string encryption, and dead code insertion [10], [11]. Tools such as ProGuard and DexGuard [12] introduce these measures, complicating the analysis and masking functional differences between app versions. These challenges underscore the need for robust methodologies in understanding geographic variations.

2.3. Cross-Domain App Differential Analysis

Prior work has examined cross-domain app differences from various perspectives. Yang et al. [3] explored security discrepancies of the same app between Google Play and third-party app markets, highlighting issues like excessive permissions and vulnerabilities. Dong et.al [13] investigated different behaviors of the same app on differenct devices Most relevant, Kumar et al. [2] analyzed geographic app differences, identifying regional disparities in privacy policies and permissions.

While these studies focused on *metadata-level* differences, our work advances the field by delving into code-level implementation differences, revealing deeper insights into how apps handle security and privacy across regions. For example, we analyzed the Google Authenticator [14] app distributed in the U.S. and India, as contrasted in Figure 1. Despite *similar metadata—permissions, privacy policies, and third-party libraries, only the U.S. version implemented a "Privacy Screen" feature* at the time, which obscures sensitive authentication codes when multitasking, mitigating shoulder-surfing attacks. This disparity highlights how the nuance detected by metadata-level analysis fails to reveal exact functionality feature differences that significantly impact user privacy and security.

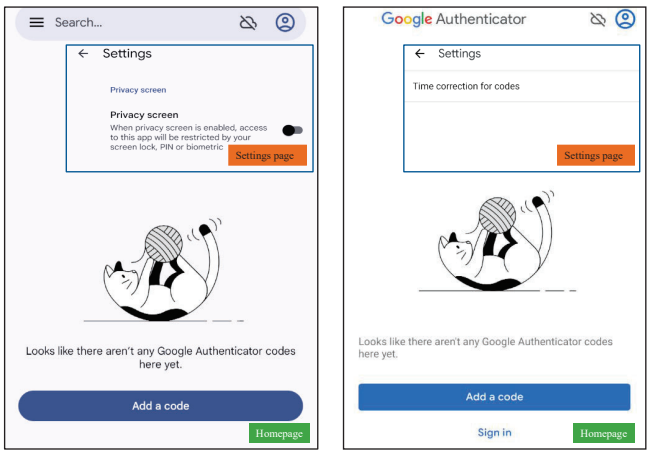

Figure 1. A motivating example of GFDs only revealed at code level.

By investigating deeper, code-level disparities, our work uniquely uncovers previously hidden implications of regional app customization, providing actionable insights for developers and platform providers to ensure consistent security/privacy standards across all regions.

3. The FREELENS Approach

In this section, we present the technical design and effectiveness evaluation of FREELENS, our automated GFD characterization approach. We start with an overview (§3.1) of FREELENS, followed by the details of its three modules/phases: *potential GFD app mining* (§3.2), *obfuscation-resilient GFD app identification* (§3.3), and *semantic GFD characterization* (§3.4). Lastly, we assess the effectiveness of FREELENS in terms of its accuracy (§3.5).

3.1. Overview

Figure 2 presents a high-level overview of FREELENS highlighting its architecture. The workflow begins with three **FREELENS Inputs** (1) app list: a curated list of Android apps (by package names) that meet our selection criteria; (2) country list: a list of target countries where the selected apps are distributed; and (3) country-specific user accounts: accounts registered in the target countries with associated payment methods. The first two inputs define the scope of our study, while (3) is needed for scraping the apps from the target countries. As we scrape the apps using Google Services, the accounts are Google accounts.

With these inputs, in **Phase 1**, FREELENS first scrapes the latest version of each chosen app across all target countries using our parallel app-scraping infrastructure. The resulting APKs then undergo a lightweight screening process, filtering out apps that unlikely have GFDs between their country-specific APKs. The goal of this phase is to garner *potential GFD apps* (i.e., those not filtered out).

Taking these potential GFD apps, FREELENS performs static code analysis in **Phase 2** to identify actual GFD apps and their GFDs. For each candidate GFD app, this phase profiles each pair of its country-specific APKs, resulting in pairwise profiles, followed by diffing these profiles. To balance analysis scalability and effectiveness, we profile apps at the granularity level of method-level control flows (i.e., *call-path* level), while examining call paths up to framework/SDK APIs (i.e., *API-bounded* profiling). The rationale is that apps cannot access security/privacy relevant capabilities except via those APIs in Android [15], [16], [17], and the APIs cannot be obfuscated. The method-level analysis is also more robust (e.g., than diffing at statement level) against app obfuscations. To further enhance the robustness, FREELENS computes call-path signatures to capture the most significant app differences without being impeded by method-level obfuscations (e.g., method renaming).

Finally, in **Phase 3**, FREELENS characterizes the GFD apps and their call-path diffs computed in **Phase 2**, generating natural-language summaries of feature variations from those diffs between each pair of country-specific APKs of each GFD app through (call-)*graph-based reasoning* (about the code diffs) on foundation LLMs. The resulting pairwise GFDs are fed to these LLMs again to summarize their associated security and privacy implications/violations. The *two-level summary mapping* (i.e., from code-level to natural-language-level feature differences and further to those impli-

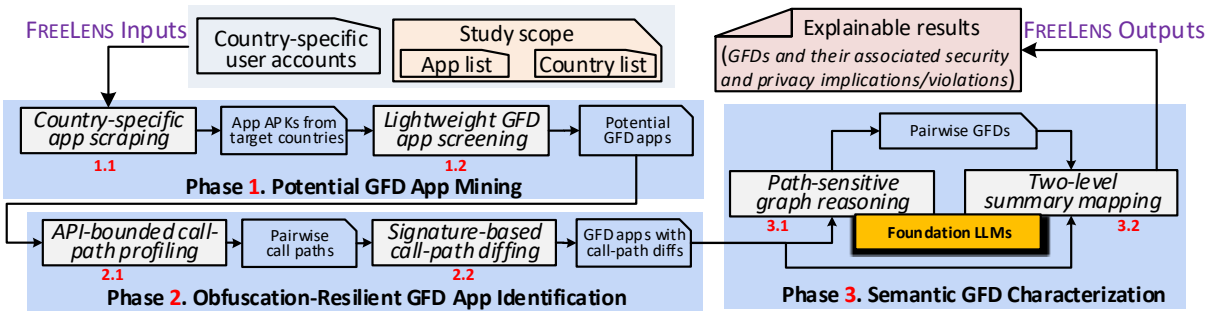

Figure 2. An overview of FREELENS, including its inputs, three working phases and per-phase steps, and outputs.

cations) enables explainable characterization results, which are the main **FREELENS Outputs** for further manual confirmation inspection in our measurement study (§4).

The key advantage of this multi-phase approach is its ability to efficiently process large numbers of apps while maintaining precision in identifying security-relevant differences. **Phase 1** quickly filters out apps unlikely to contain GFDs, allowing **Phases 2** and **3** to focus computational resources on detailed analysis of promising candidates. Additionally, our approach's resilience to code obfuscation ensures reliable results even when analyzing commercial apps that employ aggressive code protection techniques.

3.2. Potential GFD App Mining (Phase 1)

This phase serves initial data collection and filtering purposes, aiming to mine a comprehensively dataset of potential GFD apps while overcoming the associated challenge (**C1**). It works in two key steps as elaborated as follows.

3.2.1. Country-Specific App Scraping (Step 1.1). Given an app list, a country list, and an app-download account for each country, FREELENS automatically scrapes for each app the country-specific APK of a specified version from each of the countries in the target list, using the account registered for that country. To accelerate the scraping process and maintain version consistency, the FREELENS component for **Step 1.1** is deployed on multiple machines running this component in parallel, each configured with a different (Google) account representing the corresponding target country. This parallelization strategy is crucial for minimizing version discrepancies that could arise from app updates during the scraping process. While such updates may occur, our parallel approach significantly reduces the likelihood of obtaining different versions across countries. For instance, if an app updates during collection, the probability of downloading an old version for one country and an updated version for another is minimal, as the downloads for each app across all countries occur near-simultaneously.

To ensure that we did obtain the desired app version (i.e., the APK is the same as what users in the target country download from Google Play Store), we randomly picked some apps and compare their metadata which is shown on its Play Store page against the APKs. Our result shows that they are all identical. Since each app we scraped is an APK bundle [18], which consists of the base APK and the split APKs (e.g., configs, native library), we check the suffix of config APKs which is the language code. If two APK bundles of the same app have different config APKs, this indicates that we downloaded the version distributed in that country. For example, consider the app `com.mustread` distributed in India and USA. If they have the localization support, they will have different number of config APKs related to languages. As illustrated in Figure 3, the Indian version contains more language config APKs to support regional languages. For example, `bn` stands for Bengali [19] and `gu` for Gujarati [20], which are both languages spoken in India. In contrast, the US version contains fewer config APKs, especially only English language support. These differences in language support configurations provide evidence that we successfully downloaded country-specific versions of apps. The presence of regional language resources in the Indian version versus the English-only US version aligns with the expected localization patterns for these markets.

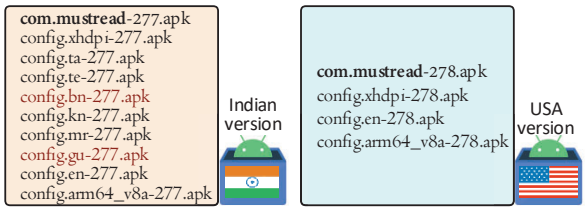

Figure 3. Split APKs for the same app distributed in India and USA.

3.2.2. Lightweight GFD App Screening (Step 1.2). For identifying potential GFD app candidates, FREELENS adopts a pruning step prior to detailed app differencing analysis for efficiency/scalability purposes. Unlike Kumar et al.'s [2] use of binary diffing, we quantify the structural characteristics of apps at three levels: number of classes, number of methods, and instruction counts. Then, if these metrics are all identical between two APKs, we rule out the possibility that they have code differences between them. For instance, intuitively, given the typically large number of instructions in an Android app, exact match in instruction count between two code-different APKs is very unlikely. We conducted pairwise comparisons of these metrics across

all the countries for all the apps in our study (§4) and confirmed that this heuristic consistently holds true. Since computing these structural metrics is quite lightweight and most of the APK pairs turn out to be identical indeed, such an initial screening step is highly cost-beneficial. This step is especially crucial given the quadratic complexity of pairwise comparisons—comparing all the APK pairs via our detailed code diffing (even just at call-graph level) would face a major scalability challenge for a large-scale study like ours.

After the screening, apps with at least one pair of country-specific APKs that have any difference in any of the three metrics are considered *potential GFD apps*, which are passed to **Phase 2**.

3.3. Obfuscation-Resilient GFD App Identification (Phase 2)

This phase aims to identify GFD apps and their concrete GFDs from the candidate GFD apps from **Phase 1** via detailed code analysis. To that end, it has to address two challenges to static analysis of Android apps: scalability barriers to such analyses (**C2**) and prevalent obfuscations in these apps (**C3**). This is achieved via two steps below.

3.3.1. API-Bounded Call-Path Profiling (Step 2.1). Given two (country-specific) APKs of a given app, we compare them to identify differences (i.e., GFDs). Prior work [2] opted for differencing apps at the level of metadata (e.g., permissions), which is overly coarse-grained and leads to missing important app differences that are only reflected at code level (§2.3). On the other hand, fine-grained analysis (e.g., of data/control flow at statement level) of real-world Android apps is known to suffer critical scalability barriers [4], [5], [21], [22], [7], which exacerbate for a large-scale study like ours. Thus, we choose to compare the two given APKs in terms of method-level control flow, diffing apps by important calling relationships in them, so as to strike a balance between the cost and effectiveness of our app differencing analysis. Given that accesses in Android apps to security/privacy capabilities must be through Android framework/SDK APIs [15], [16], [17], we profile call paths bounded by (i.e., ending at) callsites to these APIs in each APK. In this way, we focus on app differences in terms of security/privacy related accesses, which aligns well with the overall goal of our study. The fact that these APIs cannot be obfuscated further implies robustness merits of profiling at the call-path level.

Specifically, we first construct the standard (static) call graph [4] of each APK and augment it with edges representing calls to any APIs. Next, we traverse this augmented call graph, starting from each entry method until an API is encountered, resulting in *API-bounded call paths* within the APK. Considering the framework-centric structure of Android apps, we identify entry methods by locating Android framework callback overrides in the user code. Crucially, these callback methods cannot be obfuscated, providing reliable alignment points between the two APKs and minimizing false-positive differences due to renaming obfuscation. Both the start and end points of each call path are unobfuscated methods, which serve as stable anchors for the diffing step next to achieve obfuscation resiliency. Since it is based on (method-level) control flow structure, this call-path profiling approach is generally only affected by control flow obfuscation.

3.3.2. Signature-Based Call-Path Diffing (Step 2.2). After obtaining the call paths, we perform differential analysis to identify (code-level) GFDs between the two given (country-specific) APKs of each app. To maintain resilience against obfuscation, our comparison strategy focuses on three stable characteristics of each call path: (1) the entry method (Android framework callback), (2) the endpoint method (Android SDK method), and (3) the call path length. Then, we may treat all intermediate methods on a call path as symbolic placeholders, avoiding direct comparison of potentially obfuscated method signatures. While this approach ensures robustness against obfuscation, it introduces a new challenge: when the two versions (APKs) of an app have different numbers of call paths with the same start (entry method) and end points (API), determining which specific path represents the actual difference becomes non-trivial.

To address this challenge, we leverage an empirical observation about Android app obfuscation: when using the same obfuscation configuration, identical methods will be consistently renamed following the same pattern. While this does not help with direct method alignment, it provides a way to identify differing paths through aggregate characteristics. Specifically, for each intermediate method signature on each call path, we abstract it into its aggregate (total) length. Then, we use this metric together with the entry method, path length, and end-point API as a **call-path's signature**, hence comparing two call paths based on their signatures. This approach allows us to handle cases where methods have been obfuscated but maintain consistent renaming patterns. For example, consider obfuscated methods `<com.example.app.a1: void b2()>` and `<com.example.app.e3: void d4()>`. Despite different obfuscated names, these methods would be identified as equivalent based on their signature structure and position in the call path. Figure 4 shows an example.

In this example, if call-paths are matched by entry methods, end points, and path depths, there will be an extra path (3 paths versus 2 paths) in the right-side version. Since we cannot directly compare the method signatures to identify the extra path due to renaming obfuscation (e.g., 'ak.a' versus 'kk.a', 'bd.a' versus 'dd.a'), we instead leverage the key observation mentioned before: when using the same obfuscation configuration, identical methods will be consistently renamed following similar patterns. This means that while we cannot predict what 'ak.a' will be renamed to, we know that all instances of this method will be renamed to the same pattern (in this case, 'kk.a') with identical length and parameter types. By comparing aggregate signature characteristics of entire paths, we can reliably group paths that represent the same functionality despite obfuscation. This allows us to determine that the rightmost path in

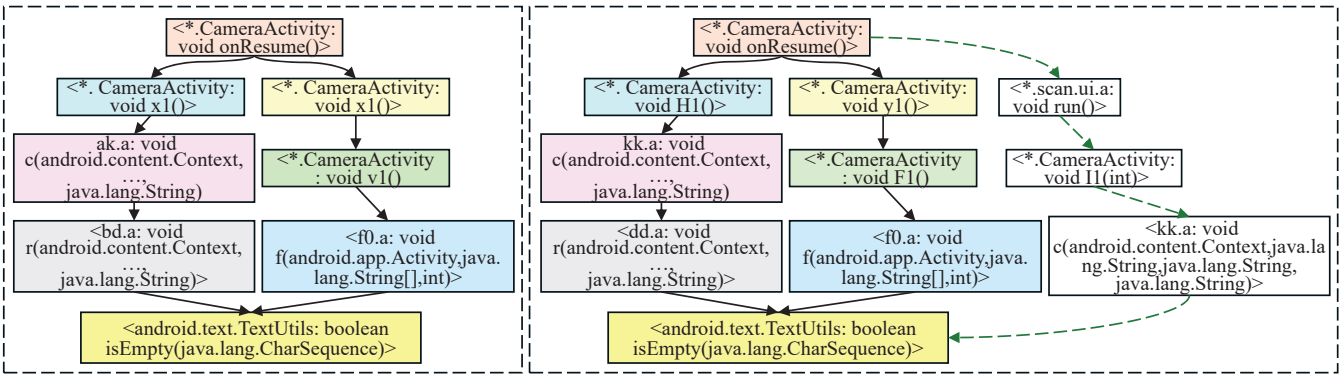

Figure 4. Different call paths of the same length, entry points, and end points.

the second version (through scan.ui.a.run()) is the actual additional functionality, as it has different signature patterns from all paths in the left version.

In this step, our signature-based call-path diffing specifically addresses obfuscation challenges by matching structural patterns rather than direct method signatures. Importantly, even when control flow is obfuscated, as long as the same obfuscation configuration is applied consistently to both APKs being compared (which is typically the case for different app versions from the same developer), our approach remains resilient. Thus, put together with **Step 2.1**, the scenario where FREELENS would face significant challenges is only when control flow obfuscation is adopted and inconsistently applied between the two compared APKs—a rare case in practice as developers typically use consistent build configurations. Nevertheless, our approach is practically accurate and robust overall (§3.5).

3.4. Semantic GFD Characterization (Phase 3)

From the call-path diffs as *code-level GFDs* of GFD apps identified in **Phase 2**, this phase aims to generate associated feature difference summaries as *natural-language-level GFDs* and their security/privacy implications. The key challenge lies in the gaps between the low-level code differences and these high-level (functionality and security) semantics of those differences. FREELENS overcomes this challenges via **Phase 3**, which works in two main steps.

3.4.1. Path-Sensitive Graph Reasoning (Step 3.1). To summarize code-level differences as feature changes, we leverage large language models (LLMs) via graph-based reasoning [23], [24] about program behaviors. We structure our analysis by representing call path differences in a format that facilitates both human comprehension and LLM analysis. We use *foundation* LLMs given their generalizability and high-accuracy in code summarization [25], [26].

First, we create a method-id mapping for all methods involved in the changed call paths, including: Entry methods (Android framework callbacks), intermediate methods in the call path, and Terminal SDK methods.

Next, we transform the call-path diffs into a tree-like representation that preserves the calling relationships between methods. This hierarchical structure, similar to the output of the Unix `tree` command, clearly shows the execution flow and method dependencies. In this tree structure, one changed call path starts with the root node, ends up with a leaf node, with all the internal nodes being intermediate nodes on the call path. For example, in Listing 1, line 1 to line 8 are the method-id mapping. Starting from line 10 are the changed paths. In this example, the two added paths are 9-8-12-11-19 and 9-8-12-14-1.

```
1  [ID to Method Mapping]: // Class: ID-Method
2  android.content.IntentFilter: 1-addAction
3  android.util.Log: 7-println
4  *.MainActivity: 8-InitBigoAds, 9-onCreate
5  sg.bigo.ads.BigoAdSdk: 11-b, 12-initialize
6  sg.bigo.ads.common.d.a: 14-a
7  sg.bigo.ads.common.s.a: 19-a
8  ......
9  [Changed Paths]:
10 Activity: *.MainActivity
11 - Entry Method: onCreate // Lifecycle Callback
12 - Added Paths:
13 9 (onCreate)
14 |_8 (InitBigoAds)
15     |_12 (initialize)
16         |-11 (b) // Obfuscated identifer
17         |    |_19 (a) // Obfuscated identifer
18         |       |_7 (println)
19         |_14 (a) // Obfuscated identifier
20             |_1 (addAction)
21 ......
```

Listing 1. An example LLM prompt for **Step 3.1** in **Phase 3**.

The method-id map and transformed call-path diffs form the *LLM prompt*. As such, the model summarizes GFDs based on (1) semantic information in the class/method-name identifiers and (2) the semantic relationships between the methods that are sensitive to specific control flow (i.e., call) paths, hence the *path-sensitive* graph reasoning.

In particular, this tree representation enables LLMs to reason about behavioral changes by analyzing (1) the entry points where changes occur (e.g., which system callback methods are affected); (2) the sequence of method calls that implement the change; and (3) the terminal APIs that

indicate the ultimate system interactions. By processing this structured representation, LLMs can infer higher-level feature changes from low-level code differences. For instance, in the example above, the LLM can identify that the change introduces advertisement initialization and tracking functionality in the app's startup sequence.

3.4.2. Two-level Summary Mapping (Step 3.2). Beyond identifying feature differences, FREELENS further summarizes their security/privacy implications. As such, we obtain three layers of information between each pair of country-specific APKs/versions of each GFD app: code (call-path) diffs (*L1*), feature difference summary (*L2*), and security/privacy summary (*L3*), as ordered from low to high semantic levels. Then, we further prompt the LLMs to perform two levels of summary mapping: mapping from L1 to L2 (*M1*) and mapping from L2 to L3 (*M2*). The rationale is to provide **explainable results**: the lower level information is used to explain the next higher level information. In this way, for both levels of summary (L2 and L3), there is *supporting evidence* (via M2 and M3 *resp.*) to justify the summary.

For example, when analyzing advertisement-related changes, the LLMs identify: *"Added initialization of Bigo advertisement SDK during app startup with custom tracking configuration."* at L2. Then, at L3, the models summarize the security/privacy implication with supporting evidence as shown in Listing 2.

```
# Security Implication: "Increased privacy risk
through additional user tracking"
# Supporting Evidence:
- Addition of advertisement SDK initialization
(path: onCreate -> InitBigoAds)
- Implementation of custom tracking callbacks
(path: initialize -> setupAds ->logEvent)
```

Listing 2. An example security/privacy implication summary with M2.

These two evidence-based mappings are essential for the further inspection and characterization in our measurement study (§4). First, it enables human analysts to verify the LLMs' conclusions by examining the specific code paths that led to each identified security concern. Second, it helps maintain traceability between high-level security implications and their concrete implementation (i.e., feature) differences. Moreover, it reduces the risk of false positives (due to LLMs' hallucinations) by requiring explicit evidence for each identified feature difference and security/privacy issue.

3.5. Implementation and Evaluation

In **Step 1.1**, we utilized Raccoon [27], an Android app downloader which supports batch download apps with CLI support, to assist with acquiring the given version of an app. For **Step 1.2**, we employed Jadx [28] to analyze the structural characteristics hence computing the three metrics (e.g., instruction count) for each APK. Finally, for creating the standard call graph of each APK, we used FlowDroid [4], a popular static analyzer of Android apps. We analyze the entire APK, including any third-party code if present in the APK. We did not explicitly recognize third-party code, nor treat them differently from user code, in our analysis. Accordingly, when differencing the two country-specific APKs of an app, the differences in third-party code, if any, are also analyzed. Similarly, for native code, our approach handles related differences in calling relationships with native functions, as these are captured in our call graph and call-path diffing. For the two-level summarization step, we provide the prompt template used in Figure 5.

```
You are an expert in analyzing Android app changes and their security implications. You will analyze call path
changes or class/methods addition/deletion in Android and provide insights
focusing on functionality changes and security/privacy implications.

# [Input Format]
Added and Removed classes and methods [if applicable]:
{Added and removed methods in the added and removed classes}
Modified methods [if applicable]:
{Method ID to signature mapping}
{Changed call paths showing added/removed call paths per class and entry method. All paths starting from entry
methods to end points are changed paths.}

# [Output Format - Strict JSON only]
{
  "changes": [
    {
      "change_summary": "summarize how functionality features changed in non-technical terms",
      "supporting_evidence": ["relevant unobfuscated package, class, method, or API calls from the paths"],
      "security_implications": ["concrete security/privacy concerns or empty array if none identified"]
    }
  ],
  "changes_summarization": {
       "summary": "Summary of overall changes"
       "security_level": "High/Medium/Low/None based on the identified security/privacy implications"
  }
}
Or an empty JSON object if no changes are found.

# [Analysis Rules (Do not include in output)]
1. Focus **only** on non-trivial changes
    - Non-obfuscated method/class/package names and their purposes
    - SDK/framework API calls that indicate user-facing features
    - Summarize "How" the changes are made, not just "What" changes are made

2. For obfuscated methods:
    - Only consider unobfuscated parts
    - Ignore fully obfuscated methods unless their context (surrounding calls, parent class) provides clear
meaning
    - Use SDK method calls as context to understand the purpose

3. Change Detection Rules: Entry methods themselves are not added or removed, but the call paths starting from
them are changed.
     - Entry methods appearing in the changes DO NOT indicate the feature itself is added/removed. These entry
methods (and their classes) exist in both versions - call paths starting from them are what changed.
     - Only summarize functionality changes if:
       a) The changed call paths show clear functionality differences
       b) The non-obfuscated methods in the paths indicate specific feature changes
     - Ignore changes whereno clear functional difference can be inferred from the paths

4. Group related changes to a single record

Remember to output ONLY the JSON result without any explanation, reasoning, or additional text.
```

Figure 5. Prompt template used for the two-level summarization.

Importantly, given that FREELENS is an automated tool, it is critical to evaluate its accuracy in identifying GFD apps and characterizing their GFDs before conducting our study using the tool. To conduct a rigorous evaluation, we randomly sampled our entire dataset of 21,120 scraped apps (§4) with 98% confidence level (CL), 5% margin of error (ME), and population proportion of 5% (given that 1,122 of the 21,120, or approximately 5%, were eventually identified as GFDs). This led to 103 apps as a statistically representative sample. Around these 103 apps, we build ground truth including (1) whether each app is a GFD app according to one of its country-specific APK/version pairs, (2) the code (call-path) differences between that pair, and (3) the summaries of feature differences and security/privacy implications for the same pair. Our ground truth creation involved a meticulous manual analysis pipeline: an in-depth app code review by comparing code-level differences between the two versions, reviewing their release notes and relevant documentation (e.g., apps' About and Data-Safety pages on Play Store, Google's official policies/regulations), and conducting extensive functionality testing of each APK by manual app exploration on physical devices. To ensure reliability and mitigate bias, three of the authors each in-

TABLE 1. EFFECTIVENESS OF FREELENS

Capability	Precision	Recall	F1 Score
GFD classification	98.97%	100%	99.48%
GFD characterization	92.09%	89.31%	90.68%

dependently created the ground truth, followed by cross-validation and a consensus process.

Using this carefully curated ground truth, we assessed the accuracy of FREELENS in (1) identifying GFD apps (i.e., *GFD classification*) and (2) generating the summary of feature differences and that of their security/privacy implications (i.e., *GFD characterization*). For classification, the evaluation was automated by exactly matching FREELENS produced labels against the ground-truth. For characterization, three of the authors manually validated the results based on the semantic equivalency of FREELENS produced GFDs and security/privacy implications against respective ground truth. First, each rater independently performed the evaluation. Then, per-rater results are cross-checked hence reaching consensus by resolving disagreement via negotiation. Table 1 lists the accuracy results. As shown, FREELENS can almost perfectly detect GFD apps—only one app was misclassified as its two given APKs have spurious call-path diffs due to their inconsistent obfuscation configurations. The results on GFD characterization show that FREELENS can effectively identify the vast majority of actual GFDs—the high recall suggests that FREELENS's approach of analyzing call paths and leveraging unobfuscated Android framework methods effectively captures most meaningful feature differences. Meanwhile, the strong precision indicates that our two-level mapping strategy successfully filters out coincidental differences and correctly identifies security-relevant variations.

4. Study Methodology

To systematically investigate GFDs and their security implications, we formulate four research questions (RQs):

RQ1 *How prevalent are GFD apps and what are the patterns of GFDs across different regions?*
RQ2 *What are the security/privacy implications of GFDs?*
RQ3 *How do the GFDs manifest in widely-used apps?*
RQ4 *Are GFDs aligned with relevant policies/regulations?*

Next, we describe the dataset collected and procedure followed for answering the RQs using FREELENS.

4.1. Dataset

Countries. The first step in studying GFDs is to determine target geographic regions. Similarly to relevant prior study [2] and others, we utilize the Freedom House's Internet Freedom (FHIF) score [29] as our primary selection criterion. This score provides a comprehensive metric that considers factors such as access restrictions, content limitations, and violations of user rights in different countries. Unlike the earlier study employing VPN services and physical devices across numerous countries [2], we opted for a more focused approach using registered Google accounts with associated payment methods, as documented in Google's policy of switching Play Store country [30]. This methodology, while covering fewer countries, ensures more reliable and consistent access to apps across regions. We carefully selected 10 countries that represent the full spectrum (i.e., 3 levels) of Internet freedom, as Table 2 shows.

TABLE 2. COUNTRIES SELECTED PER FHIF LEVELS AND SCORES

Country	FHIF Level	FHIF Score
United States (USA)	Free	76
Germany	Free	77
India	Partly Free	50
Bangladesh	Partly Free	40
Nigeria	Partly Free	59
Egypt	Not Free	28
Saudi Arabia	Not Free	25
Vietnam	Not Free	22
Turkey	Not Free	31
Pakistan	Not Free	27

Apps. For app selection, we employed a multi-pronged approach to ensure both breadth and depth in our dataset. First, we analyzed metadata from AndroZoo [31] to identify apps meeting two key criteria: (1) $\geq$1 million installations, indicating significant user impact and (2) availability across all our target countries, eliminating apps with geographic restrictions that could confound our analysis. This initial filtering yielded a substantial dataset of **20,211** apps.

To capture both current trends and historical significance in the Android ecosystem, we supplemented the above *base dataset* with two additional app sets: (1) top-10 currently trending apps from all 54 functionality categories, in total 540, on Google Play Store at the current time, and (2) top-500 most installed apps in Google Play Store history [32]. To maintain dataset independence and prevent duplicate analysis, we carefully filtered the AndroZoo-derived list to exclude apps present in either of these supplementary sets. Similarly, we ensured no overlap between the current trending apps and historical most-installed apps. After removal of 131 duplicated apps between most installed and current trending app sets, there are **909** unique apps. Therefore, the total number of apps we scraped is 21,120. This meticulous process resulted in 3 distinct, complementary datasets that collectively provide a comprehensive set of apps. As in prior work [2], we scraped *latest versions* of each app per country throughout October, 2024.

4.2. Procedure

We fed FREELENS with the 21,120 scraped apps, resulting in 1,902 unique apps with code differences. However, for 780 of these apps, the differences are only in third-party code not used by the apps yet. Ruling out these apps led to **1,122** GFD apps. Then, among the 1,122*C(10,2)=50,490 pairs of (country-specific) APKs, 20,752 have GFDs, for a total of 44,872 individual GFDs, which have 37,177 individual security/privacy implications. Then, our investigation of GFDs follows a principled procedure to address each RQ.

For **RQ1**, which examines the characteristics and patterns of GFDs, we employed an open coding approach to categorize functionality differences, first creating a codebook and then using it to label GFDs. To create the GFD codebook, we randomly sampled 333 out of all the GFDs, a sample size of 95% CL and 5% ME (with a conservative 50% population proportion). Then, three of the authors independently analyzed the sampled differences to create initial categorization schemes. For functionality differences, each author (1) examined the code-level changes identified by FREELENS, (2) assessed whether each difference fit existing categories, and (3) created new categories when needed. When establishing a new category, they defined a descriptive label, provided detailed category criteria, and included example code patterns characteristic of that category. The authors then resolved any categorization disagreements through discussion until reaching consensus on the final 15 functionality categories (F1-F15) as shown in Table 3.

After establishing the codebook, we categorized all identified GFDs. To ensure reliable categorization, we employed negotiated agreement, which is particularly valuable when the research aims to provide new insights into complex phenomena. The three authors who developed the codebooks worked together to categorize each difference, reaching consensus through discussion when initial categorizations differed. For labeling (remaining) GFDs, this process involved examining code changes to identify characteristic patterns matching our established categories.

For **RQ2**, we followed a similar coding approach. Using the same statistical parameters as for RQ1, we determined a significant sample size of 323 cases to create the security/privacy implication codebook, as shown in Table 5. During the coding process, for labeling remaining implications, we analyzed how each functional difference might impact security properties such as data protection, user privacy, and system integrity. When differences exhibited multiple characteristics, we categorized them according to their primary impact while noting secondary effects for further analysis. To maintain analytical consistency, we examined the same apps selected for RQ1, but focused specifically on security and privacy aspects.

To address **RQ3**'s focus on code-level manifestation of security/privacy implications in widely-used apps, we conducted detailed code analysis examining the call path differences identified in **Phase 2** of FREELENS. We studied multiple apps from each major category identified in RQ1, analyzing their specific code patterns and technical mechanisms for implementing geographic variations. This analysis provided insights into how popular apps concretely implement feature differences across regions.

To answer **RQ4**, we focused on apps exhibiting privacy-sensitive variations identified in our previous analysis. For each selected app, we conducted a thorough documentation review including: (1) privacy policies from both app stores and developer websites, (2) release notes describing version changes, (3) data safety declarations in Google Play Store, and (4) any region-specific documentation or user notifications. We compared these documented policies against observed code-level differences to identify potential discrepancies between stated privacy practices and actual implementations across regions. We also evaluated whether these differences comply with major privacy regulations such as GDPR and CCPA, as well as Google Play Store's privacy/security requirements. This analysis helps us understand how developers manage and communicate geographic variations in their privacy practices to users and regulators.

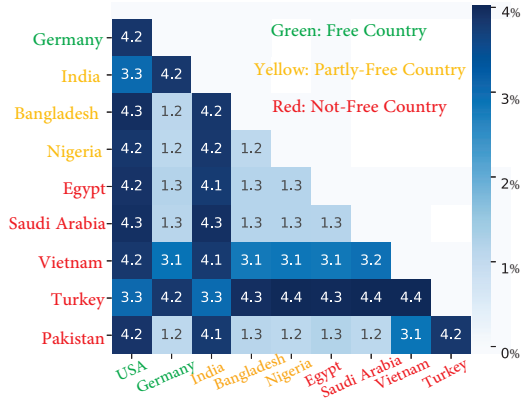

Figure 6. Distribution of the version differences between countries.

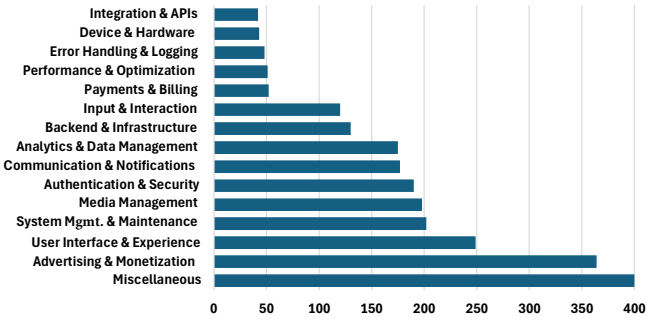

Figure 7. Distribution of functionality feature categories by focus area.

5. Characterization Results on GFDs (RQ1)

Categories of GFD. We identified 15 distinct categories of feature differences, as seen in Figure 7 and Table 3. Advertising and Monetization (F1) emerged as the most common category with 364 instances, reflecting significant regional variations in how apps implement revenue generation features. "User Interface & Experience" modifications (F2) constituted 249 cases, indicating substantial regional customization of app presentation and interaction patterns.

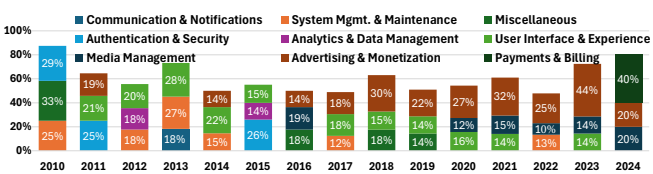

Figure 8. Top-3 GFD categories across years the apps were first released.

TABLE 3. MAIN CATEGORIES OF GFDS FOUND IN OUR STUDY

GFD Category (label)	Description (*what features are changed and how they are changed*)
Advertising & Monetization (F1)	Added/removed advertising and monetization features including ad SDK, purchase flows, and consent handling.
User Interface & Experience (F2)	Modified user interface experience through dialog prompts, navigation controls, browser interactions, etc.
System Mgmt. & Maintenance (F3)	Refactored core system components including app initialization, intent handling, runtime configurations, etc.
Media Management (F4)	Changed media handling capabilities through audio controls, image loading, video playback, etc.
Authentication & Security (F5)	Altered user privacy and security through permission controls, consent management, authentication, etc.
Communication & Notifications (F6)	Added/removed communication features through push notifications, social login flows, in-app messaging, etc.
Analytics & Data Management (F7)	Modified analytics tracking and data management through legacy integrations and state persistence.
Backend & Infrastructure (F8)	Changed backend and infrastructure for authentication flows, payment integrations, messaging systems, etc.
Input & Interaction (F9)	Altered user interaction security through improved input handling, navigation controls, chat interfaces, etc.
Payments & Billing (F10)	Modified payment and billing systems with subscription flows, authentication methods, and purchase handling.
Performance & Optimization (F11)	Optimizing app performance through background processing, lazy loading, memory management, etc.
Error Handling & Logging (F12)	Implemented comprehensive error handling updates and logging modifications.
Device & Hardware (F13)	Added/removed camera, barcode scanning, and location-related hardware functionalities.
Integration & APIs (F14)	Modified third-party integrations, authentication methods, and tracking functionalities.
Miscellaneous (F15)	Mixed feature modification including context management, storage access, encryption mechanisms, etc.

TABLE 4. NUMBER OF GFD APPS RELEASED IN EACH YEAR

Year	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024
#Apps	25	33	39	64	64	70	69	107	86	121	139	123	112	67	3

This distribution reveals that while apps maintain core functionality across regions, they frequently adapt secondary features like advertising, user interface, and system management to regional requirements or preferences. The high number of advertising and monetization differences suggests that regional market significantly influence how apps generate revenue, while the substantial variations in authentication and security implementations indicate different approaches to user protection across regions.

Distribution of the version differences between countries. Figure 6 shows the percentages of the examined apps that have different versions on each of the country pairs. For example, 4.2% of the examined apps have different versions between Pakistan and USA, while the number is 1.2% between Germany and Pakistan. We also label the freedom levels of the countries with different colors where green means a free country, yellow means a partly-free country, and red means a not-free country, as indicated in Table 2. We notice that USA, India, Vietnam, and Turkey have more apps with different versions to other countries, while Germany, Bangladesh, Nigeria, Egypt, Saudi Arabia have less. The larger percentages on USA and India are possibly caused by the large markets in the two countries [33], while the cases on Vietnam and Turkey are more likely caused by the strict regulations [34], [35]. However, we do not notice apparent patterns between the country pairs with the same and different freedom level.

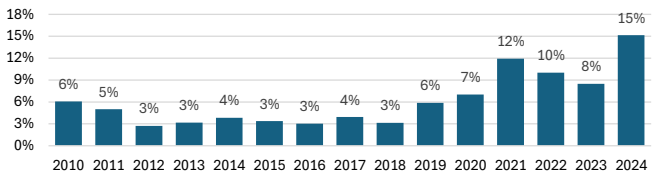

Figure 9. Percentage distribution of GFD apps by release year.

Distribution of GFD across years. Figure 9 shows the distribution of GFD app percentage over all apps examined over the years the apps were first released. We notice that the apps that were first released after 2020 had higher percentage (7%-15%) of GFD apps. This indicates that the GFDs become more frequent these years and thus relevant security issues from GFDs should be paid attention to. We further provide the number of GFD apps released in each year in Table 4.

GFD categories across years the apps were first released. Figure 8 shows the top-3 GFD categories across years the apps were first released. We notice that "Advertising & Monetization" became more frequent (20%-44%) after 2018, while "System Mgmt. & Maintenance" became less frequent after 2016. This indicates that developers were more likely to specify advertising and monetization for different countries these years, while the system management and maintenance was more unified.

6. GFDs' Security/Privacy Implications (RQ2)

Categories of GFDs' Security/Privacy Implications. Table 5 and Figure 10 show the descriptions and distribution of security implications. Among the 2,443 functionality differences identified, 2,018 (82.6%) exhibited potential security or privacy implications. Our analysis revealed 15 distinct categories of security and privacy concerns as seen in Figure, with variations ranging from data handling practices to system-level security implementations. "Data Security and Privacy" (S1) emerged as the most prevalent category with 508 instances, highlighting how geographic variations often impact user data protection measures. These differences manifested in various ways, from data collection practices to storage mechanisms and privacy controls. "Advertising Security" (S2) followed as the second most common category with 365 cases, reflecting significant regional variations in how apps implement and secure their advertising frameworks.

TABLE 5. MAIN CATEGORIES OF SECURITY/PRIVACY IMPLICATIONS OF GFDS FOUND IN OUR STUDY

Category (label)	Description (*what security & privacy implications are and how they may be realized*)
Data Security and Privacy (S1)	Handling sensitive user data biometrics and permissions compromises privacy and security.
Advertising Security (S2)	Modifying advertising integrations and tracking mechanisms affects user privacy and cause potential attack.
Permission and Consent (S3)	Modifying permission handling and consent management affects user authorization and data protection compliance.
Access Control (S4)	Altering authentication flows, permission handling, and data access exposes sensitive user information.
Authentication and Authorization (S5)	Changing authentication flows and verification mechanisms like login, payment, messages compromises security issues.
Device and System Security (S6)	Modify system settings, access device information, and manage broadcast receivers impacts device security and privacy.
Web and Browser Security (S7)	Insecure WebView modification exposes applications and users to web-based vulnerabilities and privacy risks.
Application Security (S8)	Modifying core application components like licensing and API communication impacts application security.
Payment and Financial Security (S9)	Insecure handling of financial transactions and secure validation causes unauthorized access and data leakage.
Resource Management Security (S10)	Inappropriate system resource management causes unauthorized data exposure, resource leaks, and vulnerabilities.
Input and Validation Security (S11)	Modifying input handling mechanisms and validation processes across introduces vulnerabilities if not properly secured.
Monitoring and Analytics (S12)	Modifying monitoring and logging mechanisms affects security auditing, user tracking, and advertisement analytics.
Configuration and Management (S13)	Modifying system configurations and management introduces system weaknesses through component changes.
Network Security (S14)	Altering network connections and permissions for ads, licensing, and VPN functionality causes security risks.
Other (S15)	Other core feature modification like cryptography and third-party integrations weakens system protections.

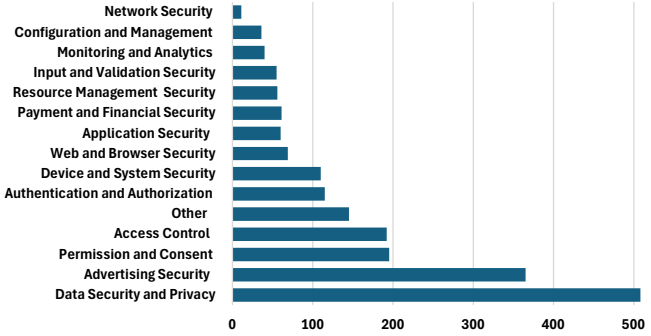

Figure 10. Distribution of security and privacy categories by focus area.

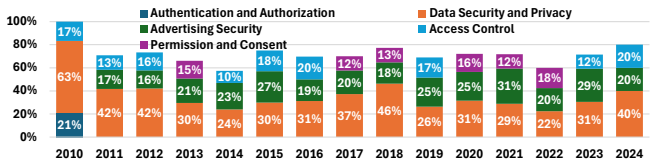

Figure 11. Top-3 security implication across (app release) years.

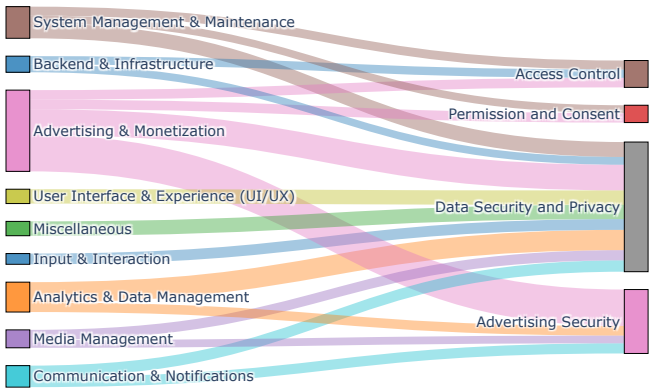

Figure 12. Top mappings from feature changes to security implications.

Security implication across years. Figure 11 shows the top-3 security implication across years the apps were first released. We notice that "Data Security and Privacy" was the dominant category from 2010 to 2024. This indicates that data security and privacy is the major concern from the geographic feature differences. Besides, advertising security became more popular in recent years (up to 31% in 2021). This indicates that advertising should be paid attention to for the possible security issues from GFDs.

Mapping from functionality feature changes to security implications. Figure 12 shows the top mappings from functionality feature changes to security implications. We notice that category "Advertising & Monetization"(F1) causes diverse security implications including "Access control"(S4), "Permission and Consent"(S3), "Data security and Privacy"(S1), and "Advertising Security"(S2). In contrast, "System Mgmt. & Maintenance"(F3) tends to cause security implications on "Access control"(S4) and "Data security and Privacy"(S1), while "Analytics & Data Management" (F7) tends to cause "Data security and Privacy"(S1).

7. Code-Level GFD Manifestations (RQ3)

To understand the concrete manifestations of these security variations, we conducted in-depth analysis of representative apps from the three top categories (excluding S15), as seen in Figure 5. These case studies revealed how seemingly minor feature differences often carry significant security implications. The following sections present detailed analyses of these representative cases, illuminating common patterns and their security implications within each category.

7.1. Advertising and Monetization

Our in-depth analysis reveals that advertising-related differences represent the most prevalent category of GFDs except for third party library internal changes. Through careful examination of the code differences identified in **Phase 2**, we observed several distinct patterns in how advertising implementations vary across regions.

Ad Complexity Variations. We found significant differences in the sophistication of advertising

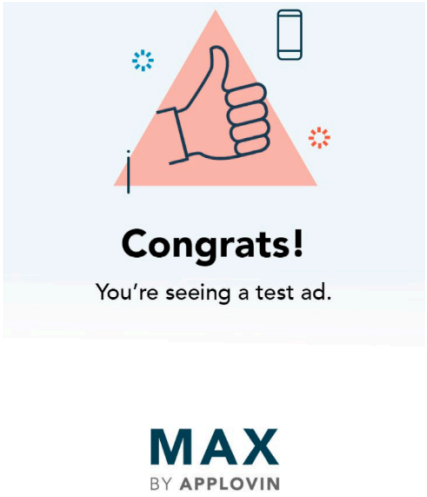

Figure 13. Test ad which does not contain real ad content.

implementations between regional versions. Some versions implement basic or test advertisements, while others deploy complex advertising systems. This is evidenced by the presence of distinctive advertising-related components in certain versions. For example, in versions with interstitial ads, we can see the presence of `com.applovin.adview.AppLovinFullscreenThemedActivity`, `com.vungle.ads.internal.ui.AdActivity`, or `com.unity3d.ads.adplayer.FullScreenWebViewDisplay`, etc. These classes indicate the implementation of advanced advertising features such as full-screen interstitial ads, rewarded video ads, and sophisticated ad debugging capabilities. The presence of methods like `dispatchTouchEvent` in these classes suggests interactive ad experiences that can capture and respond to user interactions. A recent study [36] has shown that third-party libraries with interstitial advertising capabilities may have silence installation, inappropriate ads to children, and ad fraud. The optional integration of these libraries exposes these concerns to users in certain regions. Figure 13 and Figure 14 show an example of test ad and an insterstitial ad that contains rich content, respectively.

Ad Placement Strategies. We observed significant variations in both where and when advertisements appear within apps. A common pattern involves what we perceive as "unwanted advertising," where ads are integrated into every major user interface component. The ads are typically placed at the creation or the destruction of an Activity class. In the user's view, it will be displayed when the user enters or leaves some interfaces. For example, in several apps, we found differences in back-button handling, as shown in Listing 3. This implementation demonstrates a strategy where interstitial ads are triggered by navigation events, specifically when users attempt to leave an activity using the back button. This approach maximizes ad exposure by intercepting common user interactions. As per Google Play's official policy [37], showing ads that are displayed to users in unexpected ways harms the user experience and is considered Mobile Unwanted Software.

```
onBackPressed()
|_ AdsHelper.showInterBack()
   |_ Admob.showInterAds()
      |_ onAdClosedByUser()
```

Listing 3. An example of ad placement strategy.

These findings suggest that regional differences in advertising implementations often go beyond simple presence or absence of ads, encompassing sophisticated variations in ad types, placement strategies, and conditional display logic. Such differences can have significant implications for user experience and privacy across different regions. For example, recent studies [38] [39] discovered that library-based Ad promotion can be exploited by malicious developers.

7.2. Authentication

Several apps we observed show some authentication implications. `com.pairip.licensecheck3` is a license verification module commonly used in Android apps to validate legitimate app purchases and prevent unauthorized app usage. It provides authentication services to ensure that apps are being used according to their licensing terms.

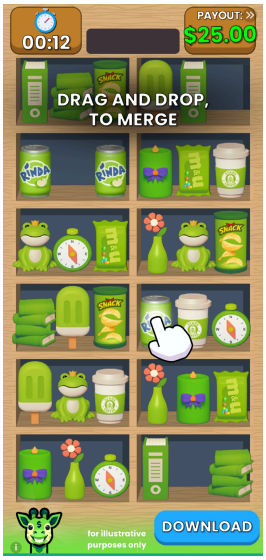

Figure 14. Interstitial ad provided by third-party libraries.

As shown in Listing 4, the code path reveals a license verification process that is present in one version but not in another. During activity creation (`onCreate`), the app initializes the license check through `LicenseClientV3`. The client then connects to a licensing service and validates the response through several steps. First, it establishes a connection with the licensing service and receives a response bundle containing license information. Then, it performs response validation using cryptographic signatures, which involves decoding a Base64-encoded public key, verifying the signature of the license response, and validating the JSON response containing license status. If any errors occur during this process, the app shows an error dialog and handles the exception appropriately.

```
onCreate(android.os.Bundle)
|_onActivityCreate()
  |_initializeLicenseCheck()>
    |_connectToLicensingService()>
       |_handleError()
          |_showErrorDialog()
       |_validateResponse()
          |_ verifySignature()
```

Listing 4. An example of authentication implication.

The absence of this license verification mechanism raises several security concerns. First, without license verification, unauthorized copies of the app could run without validation, potentially violating the developer's intellectual property rights and licensing terms. Second, the verification process includes signature validation using public key cryptography, and its removal eliminates a crucial integrity check that ensures the app hasn't been tampered with or repackaged. Third, the licensing system helps prevent revenue loss by ensuring only legitimate purchases can use the app, and its absence in some versions could lead to unauthorized distribution and usage. Furthermore, the presence of different authentication mechanisms across versions creates inconsistencies in how the app manages user access and validates installations. We noticed that this integrity check mechanism is replaced by Play Integrity API [40], potentially creating security loopholes as the version with outdated verification check mechanism could be exploited. We also observed that most of apps that did not enforce integrity check will also be blocked by Google Play Service to be installed, which is shown in Figure 15.

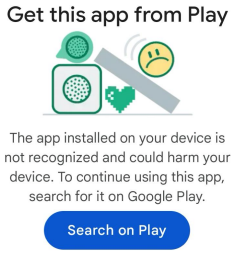

Figure 15. A example of Google Play Service blocking App without license check.

This case demonstrates how geographic variations in authentication mechanisms can create security disparities across regions. When some versions of an app implement robust license verification while others omit these checks entirely, it creates an uneven security landscape where users in certain regions may face greater risks of using compromised or unauthorized versions of the app than users in other regions. Such inconsistencies in authentication mechanisms not only affect the app's security posture but also raise questions about equitable security provisions across different geographic regions.

7.3. Access Control

The most common pattern of access control variation relates to monetization strategies across different regions. While some access control differences naturally arise from feature updates where certain users cannot access newly added functionality, the most prevalent pattern emerges in apps with consistent core features but varying monetization implementations. In these cases, one regional version typically offers users the option to remove advertisements or unlock premium features through in-app purchases, while the other version either lacks this option entirely or implements it differently. For example, in apps that support payment options to remove advertisements, we consistently observe that these versions also implement particularly aggressive advertising strategies. The implicit design appears to incentivize purchases by making the free experience less appealing through frequent ad interruptions.

The implementation of these access control differences typically manifests through in-app purchase validation mechanisms. This advertising/monetization feature variation can imply inequitable access to app functionality across geographic regions. It has been shown that local monetization strategies often demand adjustments to user-data handling and disclosure practices [41]. Our analysis reveals common patterns where apps integrate with Google Play's billing system to manage feature access. For instance, consider the implementation in Listing 5, this reveals a monetization strategy where ads are conditionally displayed based on in-app purchase status, with the billing client verification determining whether a user should see advertisements.

```
1 onCreate()
2 |_BaseBannerAdActivity.S0()
3    |_BillingClient.g()
```

Listing 5. An example of access control implication.

This differential access control creates a two-tier user experience across regions. Users in regions with premium options can potentially access an enhanced version of the app, while users in other regions are locked into a single experience level, often with unavoidable advertisements or permanently restricted features. This disparity raises questions about equitable access to app functionality across different geographic regions and highlights how monetization strategies can lead to significantly different user experiences despite apps sharing the same core functionality.

Beyond simple feature gating, we observed that these access control variations often extend to the entire user experience. Apps implementing premium options typically include additional UI elements for purchase workflows, modified navigation paths to highlight premium features, and sophisticated state management to handle the transition between free and premium states. These implementations demonstrate how deeply monetization-based access control can influence an app's architecture and user interaction patterns across different regions.

8. GFD'S Alignment With Policies/Regulations (RQ4)

Our analysis of how geographic feature differences align with relevant policies and regulations reveals concerning disparities between implemented features and their disclosure to users. We manually examined privacy policies and documentation of apps exhibiting significant privacy-relevant GFDs, particularly those categorized under Data Security and Privacy (S1) and Permission and Consent (S3), focusing on three key aspects: privacy policy disclosures, regulatory compliance, and platform policy adherence.

Data Privacy. The optional integration of third-party libraries raises concerns as they might violate data privacy [42] [43] [44]. A representative case highlighting these issues is `com.wildberries.ru` [45], which is one of the largest online retail stores based in Russia and has over 100 million downloads. It incorporates Sentry [46]

crash analysis functionality in one of its regional versions. Sentry provides detailed crash diagnostics through session replay, which captures user interactions leading to crashes, including sequential screenshots of the user interface, as shown in Listing 6. While Sentry's default configuration masks sensitive text and images [47], and we confirmed that the developer has not enabled the unmask option, this implementation still raises privacy concerns.

```
1 onResume()>
2 |_<getValue()>
3    |_<io.sentry.android.replay.ReplayIntegration$rootViewsSpy$2: io.
          sentry.android.replay.RootViewsSpy invoke()>
4       |_<io.sentry.android.replay.RootViewsSpy$Companion: io.sentry.
             android.replay.RootViewsSpy install()>
5          |_postAtFrontOfQueue(), getMainLooper()
```

Listing 6. How the app collect crash report via Sentry's integration.

Though Google Play Store's data safety page indicates that the app may collect crash logs and diagnostic data, this broad categorization inadequately conveys the extent and nature of the data collection. Session replay, even with masking enabled, can reveal sensitive interaction patterns and user behaviors. For instance, the sequence and timing of user actions, screen navigation patterns, and general usage habits are still captured and transmitted to third-party servers. This granular behavioral data could be used to profile users or reconstruct their activities within the app.

Furthermore, while Sentry's documentation explicitly states that developers should provide appropriate notices [48] to users about session replay functionality, we found no explicit mention of this feature in the app's privacy policy or user interface. This lack of transparency is particularly concerning because users in regions where this feature is enabled are unknowingly having their interactions recorded and analyzed at a level of detail beyond what would reasonably be expected for basic crash reporting. The disparity between regions - where some users are subject to this detailed behavioral monitoring while others are not - raises questions about equitable privacy protections across geographic boundaries.

This case exemplifies several critical policy compliance issues we observed across our dataset. First, despite Sentry's documentation explicitly requiring developers to notify users about session replay functionality, the app's privacy policy and user interface contain no mention of this feature. Second, the selective implementation of detailed behavioral monitoring in certain regions creates uneven privacy protections, raising concerns about equitable treatment of users across different jurisdictions. Third, the app's data collection practices go beyond what users might reasonably expect from "basic crash reporting," yet this expanded scope is not clearly communicated through any official channels.

This pattern demonstrates how geographic variations in feature implementation often lead to policy compliance issues, particularly when privacy-impacting features are deployed inconsistently across regions without appropriate disclosures. Such practices not only potentially violate platform policies requiring transparent privacy disclosures but also raise questions about compliance with regional privacy

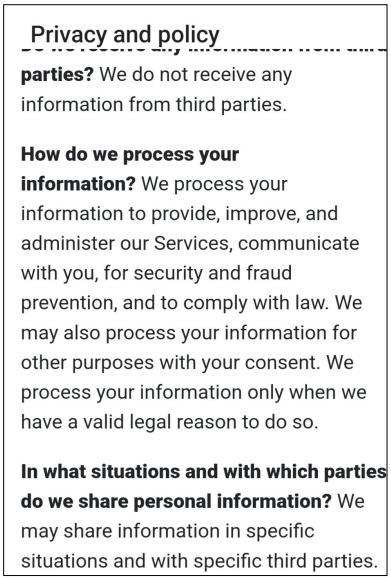

Figure 16. A privacy policy not matching with Play Store about page.

regulations that mandate clear user notification and consent for data collection activities. The collection of detailed interaction data, even if partially masked, represents a form of surveillance that users should be explicitly informed about and given the opportunity to consent to or opt out of.

Permission and User Consent. Although Android's runtime permission system provides a standardized mechanism for users to control app permissions, we observed significant variations in how apps supplement this system with additional user notifications and consent mechanisms. Some regional versions of apps implement explicit informational interfaces, such as dedicated UI pages explaining permission requirements or custom widgets displaying detailed consent information, while their counterparts in other regions rely solely on Android's default permission dialogs. This raises concerns as study [49] has shown that apps may send personal data towards data controllers without the user's explicit prior consent.

In app `com.jvstudios.gpstracker` [50] we analyzed, some region's version of the app will explicitly show customized user consent form and the link to privacy policy when the user first uses the app or changed the app's language. In contrast, versions distributed in other countries lack this transpareny - users are simply presented with system-default permission dialogs without any contextual explanation, and notably, these versions provide no in-app access to view the privacy policy.

More interestingly, while the developer claim in the Play Store About page that this app "does not collect or share user data, their privacy policy reveals discrepancies. For question "In what situations and with which parties do we share personal information?", they answer "We may share information in specific situations and with specific third parties". This makes sense as we can see a lot of third-party ad libraries are used in the app. Previous studies [51] [52]

also have shown that this discrepancy is alarmingly common. However, only users in certain regions are properly informed about this data collection through the explicit consent form. This regional disparity in privacy transparency raises ethical concerns about informed consent and suggests possible attempts to comply with stricter privacy regulations in specific markets while maintaining minimal disclosure in others. The consent form and the related privacy policy screenshot can be found in Figure 17 and Figure 16.

These regional variations in consent implementation raise several policy compliance concerns. First, they create an information asymmetry where users in different regions make privacy decisions with varying levels of context and understanding. While users in some regions receive detailed explanations about permission requirements and data usage, others must make decisions based solely on system-level permission dialogs. Second, this pattern suggests a compliance-driven approach to privacy, where stronger privacy protections are implemented only in regions with strict regulatory requirements rather than as a universal standard.

The disparity becomes particularly problematic when considering the app's actual data collection practices. Users in regions without explicit consent mechanisms may be unaware that their data could be shared with third parties, despite using an app that claims not to collect or share user data. This situation highlights how geographic variations in privacy implementations can undermine transparency and informed consent, creating situations where privacy protections become a function of geographic location rather than a fundamental user right.

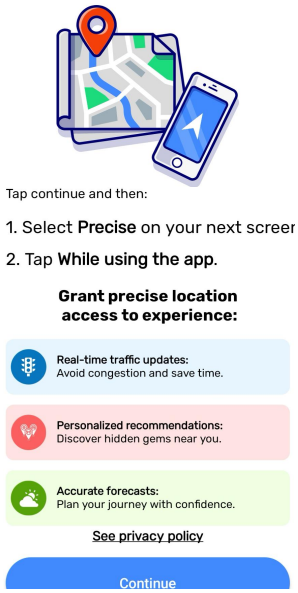

Figure 17. The consent form of the app analyzed.

This case exemplifies a broader pattern we observed where apps implement varying levels of privacy transparency across regions, potentially meeting minimum legal requirements while falling short of consistent privacy protection standards. Such practices not only raise ethical concerns about informed consent but also suggest the need for more harmonized privacy requirements across regions.

9. Discussion

In this section, we discuss the implications of our results, limitations of our technique, and disclosures of our findings.

9.1. Implication of Results

Our findings reveal several important implications for different stakeholders in the mobile app ecosystem: For App Developers, the prevalence of security and privacy variations across regions suggests a need for more systematic approaches to managing geographic customization. While regional adaptations are often necessary for business or regulatory reasons, our findings show that these modifications frequently introduce unintended security disparities. Developers should establish clear security baselines that remain consistent across all regional versions, only varying security implementations when explicitly required by local regulations. The high occurrence of advertising-related security differences (S2) particularly suggests that developers should carefully evaluate how regional monetization strategies impact their app's overall security posture.

For Platform Providers, the significant number of apps exhibiting security-relevant geographic differences (82.6% of identified variations) indicates a need for better tools and guidelines from platform providers like Google. Current app store policies primarily focus on individual app versions rather than cross-regional consistency. Platform providers should consider implementing mechanisms to help developers track and evaluate security implications of geographic variations. For instance, the app submission process could include automated comparison of security-relevant features across regional versions.

For Security Researchers, the diversity of security-relevant variations we uncovered, spanning from data handling to authentication mechanisms, suggests fertile ground for future research. Particularly noteworthy is how seemingly benign feature differences often carry subtle security implications. Our methodology of analyzing unobfuscated APIs provides a foundation for developing more sophisticated tools for cross-version security analysis.

For End Users, the variations in security and privacy implementations across regions indicate that users should be more aware of potential differences in protection levels based on their location. While users might assume that globally distributed apps maintain consistent security standards, our findings suggest this is not always true. This highlights the importance of users understanding security and privacy features available in their regional versions of apps.

These implications underscore a broader tension in the mobile app ecosystem between the need for regional customization and the importance of maintaining consistent security standards. As apps continue to evolve with regional variations, addressing these challenges will become increasingly critical for ensuring equitable security and privacy protections across geographic boundaries.

9.2. Limitations

Technical Limitations. Our technique is primarily designed to handle renaming obfuscation, which is the most common type of code obfuscation in Android apps. However, more sophisticated obfuscation techniques, such as control flow obfuscation, string encryption, or dynamic code loading, may lead to false positives in our analysis. When apps employ these advanced obfuscation methods, FREELENS might incorrectly identify or characterize feature

differences due to its inability to fully comprehend the obfuscated code patterns.

Modern commercial Android apps often exhibit high complexity in their architecture and dependencies, which can pose challenges for static analysis tools. In particular, we found that FlowDroid, which we rely on for call graph construction, sometimes fails to process these complex apps. When call graph construction fails, FREELENS cannot proceed with its analysis, limiting our coverage of the Android app ecosystem. Additionally, many modern Android apps utilize dynamic feature modules and on-demand content loading to optimize app size and performance. Since FREELENS performs static analysis, it cannot capture differences in dynamically loaded features or content that may vary by region. This limitation particularly affects large-scale commercial apps that often contain the most interesting geographic variations. For efficiency purposes, we did not consider code (whether it be third-party, native, or user code) that is not used by the app (i.e., not reachable from the app's user-code entry points as per the call graph) yet. However, the reachability analysis could be inaccurate if the two APKs inconsistently applied control-flow obfuscation. But again this is not tied with if the code is third-party or not.

Regarding substantial native code components, while FREELENS effectively captures differences in calling relationships with native functions through our call graph and call path diffing, it cannot analyze implementation differences within the native code (C/C++ and other compiled languages) itself. This limitation would lead to missing GFDs that are reflected primarily in changes to native function implementations rather than their calling patterns. Additionally, native code involved in obfuscated control flow (i.e., when the native code is invoked via reflective calls) is subject to the same reachability related limitation as mentioned above.

Threats to Validity. Our analysis provides a snapshot of geographic differences at a specific point in time, rather than a comprehensive view of how these differences evolve. This limitation stems from two factors: (1) Google Play's API does not provide access to the metadata of historical app versions. Even though we can get the historical app versions, we cannot align what versions are the latest at the specific time due to the lack of upload date. (2) Developers' app update policies and schedules vary significantly across regions. Consequently, we cannot determine whether identified differences are persistent across versions or represent temporary variations. This limits our ability to understand the long-term patterns and motivations behind geographic feature differences.

Moreover, our approach relies on matching apps across regions using package names, which means we cannot automatically detect regional variants of apps that use different package names. For example, Amex app (`com.americanexpress.android.acctsvcs.us`) [53] and its other countries' variants, e.g., Amex United Kingdom (`com.americanexpress.android.acctsvcs.uk`) [54] are essentially regional versions of the same app but are treated as separate apps in our analysis. Similarly, other major platforms may maintain region-specific variants with distinct package names to comply with local regulations or cater to regional user preferences. To study such cases, manual curation of app pairs would be necessary, requiring domain knowledge and continuous monitoring of regional app markets. This limitation means our study might miss important geographic variations implemented through separate app variants rather than within a single app package.

9.3. Ethics and Disclosures

Our study involved accessing and analyzing Android applications across different geographic regions, raising several ethical considerations that we carefully addressed throughout our research.

First, all app downloads were conducted through legitimate Google Play Store accounts with proper payment methods registered in the respective countries. We followed Google Play's terms of service and did not employ any circumvention techniques to access region-restricted content. Our app selection process focused on publicly available applications, and we did not attempt to access any private or restricted versions.

In collecting app data, we were mindful of bandwidth usage and rate limits. Our parallel downloading infrastructure was designed to respect Google Play's servers by maintaining reasonable request rates. When analyzing apps, we limited our investigation to static analysis of code and did not perform any dynamic analysis that could potentially interfere with app operations or backend services. During our security analysis, we identified several concerning variations in privacy and security implementations across regions.

To responsibly disclose our findings, we informed relevant stakeholders of security and privacy concerns identified during our study. Following standard responsible disclosure practices, we contacted both the Google Play Store security team and the developers of apps with significant privacy-relevant GFDs. Our communications detailed specific inconsistencies between declared data practices and actual implementations, explained the regional variations we observed in privacy controls, and provided actionable recommendations for addressing these issues. Moreover, to the developers of respective apps, we also provided sufficient technical details to reproduce and verify our findings while offering assistance for any follow-up investigations. This approach aims to ensure that security and privacy issues could be addressed before *public* disclosure of our research findings.

10. Related Work

Prior work has explored/measured various ramifications (e.g., security/privacy impact) of Android app evolution [55], [56]. Kumar et al. [3] examined security discrepancies between app versions from different markets. Supporting those measurements, software differencing techniques, ranging from textual to semantic analysis [57], [58], [59], [60], have also been developed. In the Android context, tools like Androguard [61] and LibRadar [62] compare

code structures and libraries, while others like WuKong [63] detect app clones. Several studies examined the evolutionary patterns of benign apps revealing their relatively rare use of refactoring or obfuscation [56], the evolution of behavioral differences between benign and malicious apps [64]. A recent study [65] explored the security relevancy of app incompatibilities, revealing that malware suffers notably lesser run-time compatibility issues than benign apps.

Android static analysis tools, such as FlowDroid [4], Amandroid [66], and DroidRista [67], employ static taint analysis to map data flows. Code summarization techniques, both traditional [68] and context-aware [69], [70], aim to generate descriptions of code functionality. Android-specific methods incorporate dynamic analysis and external resources [71], [72], [73]. These approaches, while valuable for understanding code behavior, do not directly address the problem of characterizing GFDs and their security implications across different regions.

11. Conclusion

This work presents the first large-scale, *code-level* study of geographic feature differences (GFDs) in Android apps and their security implications, analyzing over 21,000 apps across ten countries. We designed FREELENS framework which effectively identifies these security-relevant variations. We find that most of identified GFDs have security or privacy implications, revealing significant regional variations, highlighting the impact of commercial and regulatory pressures on user security.

Acknowledgments

We thank the reviewers for insightful and constructive comments. J. Guo, Y. Nong, and H. Cai were supported in part by the Open Technology Fund (OTF) under Grant B00236-1220-00, and in part by Office of Naval Research (ONR) under Grant N000142212111. Z. Lin was supported by National Science Foundation (NSF) under grant 2330264.

References

[1] "Encrypted messaging applications and political messaging: How they work and why understanding them is important for combating global disinformation - center for media engagement - center for media engagement," https://mediaengagement.org /research/encrypted-messaging-applications-and-political-messaging/, Jun. 2023.

[2] R. Kumar, A. Virkud, R. S. Raman, A. Prakash, and R. Ensafi, "A large-scale investigation into geodifferences in mobile apps," in *USENIX Security*, 2022, pp. 1203–1220.

[3] S. Yang, G. Bai, R. Lin, J. Guo, and W. Diao, "Beyond the horizon: Exploring cross-market security discrepancies in parallel Android apps," in *ISSRE*, 2024, pp. 558–569.

[4] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," in *PLDI*, 2014, p. 259–269.

[5] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard, "Information flow analysis of Android applications in DroidSafe," in *NDSS*, 2015.

[6] J. Zhang, C. Tian, and Z. Duan, "Fastdroid: efficient taint analysis for Android applications," in *ICSE-Companion*, 2019, pp. 236–237.

[7] D. He, H. Li, L. Wang, H. Meng, H. Zheng, J. Liu, S. Hu, L. Li, and J. Xue, "Performance-boosting sparsification of the IFDS algorithm with applications to taint analysis," in *ASE*, 2019, pp. 267–279.

[8] P. Wang, Q. Bao, L. Wang, S. Wang, Z. Chen, T. Wei, and D. Wu, "Software protection on the go: A large-scale empirical study on mobile app obfuscation," in *ICSE*, 2018, pp. 26–36.

[9] "Distribute app releases to specific countries - Play Console Help," https://support.google.com/googleplay/android-developer/answer/7550024?hl=en.

[10] "Shrink, obfuscate, and optimize your app | Android Studio," https://developer.android.com/build/shrink-code.

[11] A. Bacci, A. Bartoli, F. Martinelli, E. Medvet, and F. Mercaldo, "Detection of obfuscation techniques in Android applications," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 2018, pp. 1–9.

[12] "ProGuard Manual: Home | Guardsquare," https://www.guardsquare.com/manual/home.

[13] Z. Dong, Y. Zhao, T. Liu, C. Wang, G. Xu, G. Xu, L. Zhang, and H. Wang, "Same app, different behaviors: Uncovering device-specific behaviors in Android apps," in *ASE*, 2024, pp. 2099–2109.

[14] "Google Authenticator - Apps on Google Play," https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2.

[15] A. Developers, "Permissions overview," 2023. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview

[16] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An information-flow tracking system for real-time privacy monitoring on smartphones," in *OSDI*, 2010, pp. 393–407.

[17] J. Lin, J. Liu, N. Sadeh, and J. I. Hong, "Android permissions: User attention, comprehension, and behavior," in *SOUPS*, 2014, pp. 1–14.

[18] "About Android App Bundles," https://developer.android.com/guide/app-bundle.

[19] "Bengali language," *Wikipedia*, Nov. 2024.

[20] "Gujarati language," *Wikipedia*, Nov. 2024.

[21] M. Benz, E. K. Kristensen, L. Luo, N. P. B. Jr, E. Bodden, and A. Zeller, "Heaps'n leaks: how heap snapshots improve Android taint analysis," in *ICSE*, 2020, pp. 1061–1072.

[22] J. Lerch, B. Hermann, E. Bodden, and M. Mezini, "FlowTwist: efficient context-sensitive inside-out taint analysis for large codebases," in *FSE*, 2014, pp. 98–108.

[23] J. Tang, Y. Yang, W. Wei, L. Shi, L. Su, S. Cheng, D. Yin, and C. Huang, "Graphgpt: Graph instruction tuning for large language models," in *SIGIR*, 2024, pp. 491–500.

[24] H. Wang, S. Feng, T. He, Z. Tan, X. Han, and Y. Tsvetkov, "Can language models solve graph problems in natural language?" *NeuralIPS*, vol. 36, pp. 30 840–30 861, 2023.

[25] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *TOSEM*, vol. 33, no. 8, pp. 1–79, 2024.

[26] T. Ahmed, K. S. Pai, P. Devanbu, and E. Barr, "Automatic semantic augmentation of language model prompts (for code summarization)," in *ICSE*, 2024, pp. 1–13.

[27] "APK Downloader for PC," https://raccoon.onyxbits.de/.

[28] skylot, "Skylot/jadx," Nov. 2024.

[29] "Countries," https://freedomhouse.org/countries/freedom-net/scores.

[30] "How to change your Google Play country - Google Play Help," https://support.google.com/googleplay/answer/7431675?hl=en.

[31] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of Android apps for the research community," in *MSR*, 2016, pp. 468–471.

[32] "List of most-downloaded Google Play applications," *Wikipedia*, Sep. 2024.

[33] "Google play store statistics by device traffic," https://www.enterpriseappstoday.com/stats/google-play-store-statistics.html.

[34] "Internet censorship in vietnam," https://en.wikipedia.org/wiki/Internet_censorship_in_Vietnam.

[35] "Censorship in turkey," https://en.wikipedia.org/wiki/censorship_in_Turkey.

[36] F. P. Research, "AppLovin (APP) – Formers Allege Ad Fraud; Is DTC Hype Actually 'Stealing' Meta's Data; Illegal Tracking of Children & Serving Sex Ads to Kids," Feb. 2025.

[37] "Mobile Unwanted Software - Play Console Help," https://support.google.com/googleplay/android-developer/answer/9970222?sjid=5354602318567233593-NA#ProtectUserDataAndPrivacy.

[38] S. Ma, C. Chen, S. Yang, S. Hou, T. J.-J. Li, X. Xiao, T. Xie, and Y. Ye, "Careful about what app promotion ads recommend! detecting and explaining malware promotion via app promotion graph," *arXiv preprint arXiv:2410.07588*, 2024.

[39] V. Rastogi, R. Shao, Y. Chen, X. Pan, S. Zou, and R. D. Riley, "Are these ads safe: Detecting hidden attacks through the mobile app-web interfaces." in *NDSS*, 2016.

[40] "Overview of the Play Integrity API | Google Play," https://developer.android.com/google/play/integrity/overview.

[41] C. Kuner, L. A. Bygrave, C. Docksey, L. Drechsler, and L. Tosoni, "The eu general data protection regulation: A commentary/update of selected articles," *Update of Selected Articles (May 4, 2021)*, 2021.

[42] K. Zhao, X. Zhan, L. Yu, S. Zhou, H. Zhou, X. Luo, H. Wang, and Y. Liu, "Demystifying privacy policy of third-party libraries in mobile apps," in *ICSE*, 2023, pp. 1583–1595.

[43] Y. Shen, P.-A. Vervier, and G. Stringhini, "Understanding worldwide private information collection on Android," *arXiv preprint arXiv:2102.12869*, 2021.

[44] M. H. Meng, C. Yan, Y. Hao, Q. Zhang, Z. Wang, K. Wang, S. G. Teo, G. Bai, and J. S. Dong, "A large-scale privacy assessment of Android third-party sdks," *arXiv preprint arXiv:2409.10411*, 2024.

[45] "Wildberries - Apps on Google Play," https://play.google.com/store/apps/details?id=com.wildberries.ru&hl=en.

[46] "Application Performance Monitoring & Error Tracking Software," https://sentry.io/welcome/.

[47] "Set Up Session Replay | Sentry for Android," https://docs.sentry.io/platforms/android/session-replay/.

[48] "Protecting User Privacy in Session Replay," https://docs.sentry.io/security-legal-pii/scrubbing/protecting-user-privacy/.

[49] T. T. Nguyen, M. Backes, N. Marnau, and B. Stock, "Share first, ask later (or never?) studying violations of {GDPR's} explicit consent in Android apps," in *USENIX Security*, 2021, pp. 3667–3684.

[50] "GPS+ Maps, Navigation, Traffic - Apps on Google Play," https://play.google.com/store/apps/details?id=com.jvstudios.gpstracker&hl=en.

[51] I. Arkalakis, M. Diamantaris, S. Moustakas, S. Ioannidis, J. Polakis, and P. Ilia, "Abandon all hope ye who enter here: A dynamic, longitudinal investigation of Android's data safety section," in *USENIX Security*, 2024, pp. 5645–5662.

[52] L. Yu, X. Luo, X. Liu, and T. Zhang, "Can we trust the privacy policies of Android apps?" in *DSN*, 2016, pp. 538–549.

[53] "Amex - Apps on Google Play," https://play.google.com/store/apps/details?id=com.americanexpress.android.acctsvcs.us&hl=en_US.

[54] "Amex United Kingdom – Apps on Google Play," https://play.google.com/store/apps/details?id=com.americanexpress.android.acctsvcs.uk&hl=en_GB.

[55] H. Cai, "Embracing mobile app evolution via continuous ecosystem mining and characterization," in *IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2020, pp. 31–35.

[56] H. Cai and B. Ryder, "A longitudinal study of application structure and behaviors in Android," *TSE*, vol. 47, no. 12, pp. 2934–2955, 2020.

[57] B. Fluri, M. Wursch, M. PInzger, and H. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," *TSE*, vol. 33, no. 11, pp. 725–743, 2007.

[58] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, "Fine-grained and accurate source code differencing," in *ASE*, 2014, pp. 313–324.

[59] J. Guo and H. Cai, "EvoTaint: Incremental static taint analysis of evolving Android apps," in *TOSEM*, 2025.

[60] J. Guo, H. Yang, and H. Cai, "VerLog: Enhancing release note generation for Android apps using large language models," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, 2025.

[61] "Androguard/androguard," androguard, Nov. 2024.

[62] Z. Ma, H. Wang, Y. Guo, and X. Chen, "LibRadar: Fast and accurate detection of third-party libraries in Android apps," in *ICSE-Companion*, 2016, pp. 653–656.

[63] H. Wang, Y. Guo, Z. Ma, and X. Chen, "Wukong: A scalable and accurate two-phase approach to Android app clone detection," in *ISSTA*, 2015, pp. 71–82.

[64] H. Cai, X. Fu, and A. Hamou-Lhadj, "A study of run-time behavioral evolution of benign versus malicious apps in Android," *Information and Software Technology (IST)*, vol. 122, p. 106291, 2020.

[65] J. Guo, X. Fu, L. Li, T. Zhang, M. Fazzini, and H. Cai, "Characterizing installation-and run-time compatibility issues in Android benign apps and malware," *TOSEM*, 2024.

[66] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of Android apps," *TOPS*, vol. 21, no. 3, pp. 1–32, 2018.

[67] A. Alzaidi, S. Alshehri, and S. M. Buhari, "DroidRista: a highly precise static data flow analysis framework for Android applications," *International Journal of Information Security*, vol. 19, no. 5, pp. 523–536, 2020.

[68] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for Java methods," *TSE*, vol. 42, no. 2, pp. 103–119, Feb. 2016.

[69] Z. Tang, X. Shen, C. Li, J. Ge, L. Huang, Z. Zhu, and B. Luo, "Ast-trans: Code summarization with efficient tree-structured attention," in *ICSE*, 2022, pp. 150–162.

[70] Q. Chen, X. Xia, H. Hu, D. Lo, and S. Li, "Why my code summarization model does not work: Code comment improvement with category prediction," *TOSEM*, vol. 30, no. 2, pp. 1–29, 2021.

[71] A. Aghamohammadi, M. Izadi, and A. Heydarnoori, "Generating summaries for methods of event-driven programs: An Android case study," *JSS*, vol. 170, p. 110800, 2020.

[72] A. Naghshzan, L. Guerrouj, and O. Baysal, "Leveraging unsupervised learning to summarize APIs discussed in Stack Overflow," in *IEEE International Conference on Source Code Analysis and Manipulation (SCAM)*, 2021, pp. 142–152.

[73] E. Aghajani, G. Bavota, M. Linares-Vásquez, and M. Lanza, "Automated documentation of Android apps," *TSE*, vol. 47, no. 1, pp. 204–220, 2019.

Appendix A.
Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

The Android app analysis framework FreeLens can spot differences in mostly similar apps. FreeLens capabilities are demonstrated by studying regional differences in Android apps and evaluating their security and privacy impact.

A.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Identifies an Impactful Vulnerability
- Provides a Valuable Step Forward in an Established Field

A.3. Reasons for Acceptance

1) Addresses interesting challenges to understand geo-feature differences.
2) FreeLens will be openly available, enabling future research that requires app diffing.
3) Novel diffing technique used, which uses the fact that Android Studio's code shrinking preserves obfuscated function name lengths.

A.4. Noteworthy Concerns

1) Optimized towards handling the most common type of Android app obfuscation, in particular Android Studio's code shrinking. It will likely fail for some other obfuscation techniques, such as mixed boolean arithmetics and VM-based obfuscation.
2) Only captures differences for calling into native code but not the native code itself (C/C++ and other compiled code).