Comparing Apples to Androids: Discovery, Retrieval, and Matching of iOS and Android Apps for Cross-Platform Analyses

Magdalena Steinböck
TU Wien
Vienna, Austria
magdalena.steinboeck@seclab.wien

Jakob Bleier
TU Wien
Vienna, Austria
jakob.bleier@seclab.wien

Mikka Rainer
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
mikka.rainer@cispa.de

Tobias Urban
Institute for Internet Security
Gelsenkirchen, Germany
urban@internet-sicherheit.de

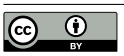
Christine Utz
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
christine.utz@cispa.de

Martina Lindorfer
TU Wien
Vienna, Austria
martina@seclab.wien

ABSTRACT

For years, researchers have been analyzing mobile Android apps to investigate diverse properties such as software engineering practices, business models, security, privacy, or usability, as well as differences between marketplaces. While similar studies on iOS have been limited, recent work has started to analyze and compare Android apps with those for iOS. To obtain the most representative analysis results across platforms, the ideal approach is to compare their characteristics and behavior for the same set of apps, e. g., to study a set of apps for iOS and their respective counterparts for Android. Previous work has only attempted to identify and evaluate such *cross-platform apps* to a limited degree, mostly comparing sets of apps independently drawn from app stores, manually matching small sets of apps, or relying on brittle matches based on app and developer names. This results in (1) comparing apps whose behavior and properties significantly differ, (2) limited scalability, and (3) the risk of matching only a small fraction of apps.

In this work, we propose a novel approach to create an extensive dataset of cross-platform apps for the iOS and Android ecosystems. We describe an analysis pipeline for discovering, retrieving, and matching apps from the Apple App Store and Google Play Store that we used to create a set of 3,322 cross-platform apps out of 10,000 popular apps for iOS and Android, respectively. We evaluate existing and new approaches for cross-platform app matching against a set of reference pairs that we obtained from Google's data migration service. We identify a combination of seven features from app store metadata and the apps themselves to match iOS and Android apps with high confidence (95.82 %). Compared to previous attempts that identified 14 % of apps as cross-platform, we are able to match 34 % of apps in our dataset. To foster future research in the cross-platform analysis of mobile apps, we make our pipeline available to the community.



MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04.
https://doi.org/10.1145/3643991.3644896

CCS CONCEPTS

• **Software and its engineering** → **Software libraries and repositories**; • **General and reference** → **Empirical studies**.

KEYWORDS

iOS, Android, mobile apps, app retrieval, app matching, app stores

ACM Reference Format:
Magdalena Steinböck, Jakob Bleier, Mikka Rainer, Tobias Urban, Christine Utz, and Martina Lindorfer. 2024. Comparing Apples to Androids: Discovery, Retrieval, and Matching of iOS and Android Apps for Cross-Platform Analyses. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3643991.3644896

1 INTRODUCTION

Since the advent of the first smartphones in the 2000s, personal computing has been increasingly moving from desktops to mobile devices such as smartphones and tablets. Consequently, research has placed great focus on studying the mobile ecosystem, including devices, their operating systems (OS), and designated applications (apps). As of 2023, the mobile ecosystem is governed by two big platforms with a combined market share of over 99 % [60]: Android, an open-source OS managed by Google, and iOS, a closed-source OS by Apple, along with their respective app stores, the Google Play Store and the Apple App Store. While alternative app stores, also referred to as marketplaces, for Android apps do exist, iOS users without jailbroken phones so far have been limited to the official store. As of March 2024, iOS users in the European Union (EU) can also install apps from alternative marketplaces [8], diversifying this ecosystem as well.

A rich body of work has already studied Android apps from the Google Play Store and alternative marketplaces from different perspectives, investigating, for example, software engineering practices concerning releases and updates [12, 16, 54, 55], business models [24, 45, 58], security and privacy [6, 22, 33, 37, 38, 43, 51], and usability [10, 15, 19, 36]. Apps for iOS have been studied to a far lesser extent [1, 14, 17, 41], and only recently have researchers started to compare the behavior and properties of apps between platforms, for example, comparing Android and iOS apps for privacy

issues [31, 32, 42, 48, 52, 53] or user reviews and ratings [4, 28, 29]. However, most of these cross-platform studies compared samples of apps individually drawn from the two official stores. As a result, this work may suffer from the "*comparing apples to oranges*" problem, as the apps investigated to compare the two ecosystems, albeit sampled using the same criteria, may widely differ, and the compared datasets may only have limited comparability. Those studies that did attempt to find an iOS app's Android counterpart (or vice versa) and analyzed their specific properties studied only a subset of apps that use frameworks for cross-platform app development [4], relied on manual app matching [16, 28, 52], or performed app matching with limited metrics, such as string comparison of app and developer names [3, 25, 32], without verifying the accuracy of such an approach. Albeit limited in scale, earlier work already noticed significant differences between the "average app" and cross-platform apps [32], confirming the need for a robust approach to identify apps across platforms for analysis at scale.

We address this research gap and study *cross-platform apps*, by which we denote apps available for different platforms that can be considered to be "the same," except for being offered for other ecosystems, such as the instant messenger Signal for iOS and Signal for Android. Reaching this verdict, let alone automating it, is a nontrivial problem. For example, it is not sufficient that two apps provide the same service (e. g., web browsers) or are the same software (e. g., Firefox and the pre-release build Firefox Nightly serve different use cases). Our definition of cross-platform apps is also broader than that of apps sharing the same code across platforms, such as those built with cross-platform app development frameworks [4] or web technologies using app building frameworks [40]. For example, if a company hires different developers to create an iOS and an Android app with the same requirements, functionality, and interface, we consider the apps the same.

In this work, we collect a large dataset of popular mobile apps for iOS and Android composed of *only* cross-platform apps as defined above. We describe how we sampled, collected metadata, downloaded, and matched around 10,000 popular apps from the Apple App Store and the Google Play Store, respectively. We investigate various existing and new approaches for app matching, using different types of app metadata and data from the apps themselves.

To address the lack of ground truth in this area, we present a novel approach to collect reference pairs from Google's data migration service for users transitioning from iOS to Android phones. We evaluated the performance of our matcher against these reference pairs and manually verified a subset of app pairs that were not included in the reference list. We find that out of 3,467 resulting app pairs that were also among the reference pairs, our matcher correctly identified 3,322 (95.82 %) pairs of cross-platform apps. Compared to previous work that collected a set of cross-platform apps to study their privacy aspects [32] and identified only 14 % of the downloaded apps as such, we can confidently identify 34 % of iOS and Android apps as cross-platform app pairs.

Our contributions can be summarized as follows:

- We discover, retrieve, and download around 10,000 popular apps for each iOS and Android and identify 3,322 pairs of cross-platform apps that can be the basis for future cross-platform studies of the iOS and Android ecosystems.
- We evaluate different types of app store metadata and app data in terms of how reliable they are for finding the same app for iOS and Android. We obtain the best results using seven matching criteria (app name, developer name, app ID, privacy policy URL, app description, app icon, deep links).
- We describe how we leveraged Google's API for iOS-to-Android data migration to obtain a set of 7,671 reference pairs, which we used to validate our matching results.

Artifacts: We provide the code of our analysis pipeline, reference pairs, resulting matches, and other supplementary information at https://github.com/SecPriv/cross-platform-matching. Due to the size of the downloaded apps (roughly 2 TB), we only make them available for research purposes upon request (also see Section 4).

2 RELATED WORK

Our work goes beyond previous research that retrieved apps and their metadata from app stores, compared iOS and Android apps, or investigated methods to match apps across platforms.

App Store Mining. Related work already developed scalable techniques for crawling the Google Play Store (and alternative Android app stores) to collect metadata and Android app datasets. AndroZoo [5] collects apps from the Google Play Store and alternative marketplaces to provide current and historical apps to the research community, but it only has been incorporating app metadata since December 2023 [63]. Viennot et al. [64] proposed a more comprehensive crawling technique that harvests both metadata and app-level information. The metadata available in Android app stores has been used to understand different aspects of the Android ecosystem, including user perceptions of apps [28, 29, 36], popularity [34], release practices [12, 66], and developer attribution issues [23]. Related work further characterized Android apps within the Google Play Store [64, 70] and across markets [34, 45, 65, 67].

Cross-Platform Studies. Relying on app store metadata and with a focus on comparing the Android and iOS ecosystems, Ali et al. [3] characterized apps based on user ratings, versions, and prices on a large scale. On a smaller scale but with a longitudinal aspect, Domínguez-Álvarez and Gorla [16] performed a study on release practices of the most popular apps on each platform. Sebastián et al. [57] identified developer accounts and other features that can be used to attribute malicious operations across app stores. Hu et al. investigated how user reviews and ratings differed both for manually matched cross-platform apps [28] and those developed with designated cross-platform frameworks [29].

From a wider perspective, the most active field of research for cross-platform comparisons of mobile ecosystems has focused on security and privacy issues. Han et al. [25] performed a cross-platform comparison of apps to understand how privacy protection methods are implemented for the same app on different platforms. Squires et al. [59] showed that Android apps were often over-privileged while iOS did not suffer from such issues; until Android 6.0 (released in October 2017), only iOS used dynamic permission control. Chen et al. [13] assessed the usage of potentially harmful software libraries across Android and iOS apps and found that almost 7 % of the analyzed Android apps and nearly 3 % of the iOS apps utilized such libraries. To find personal information leaks, Ren

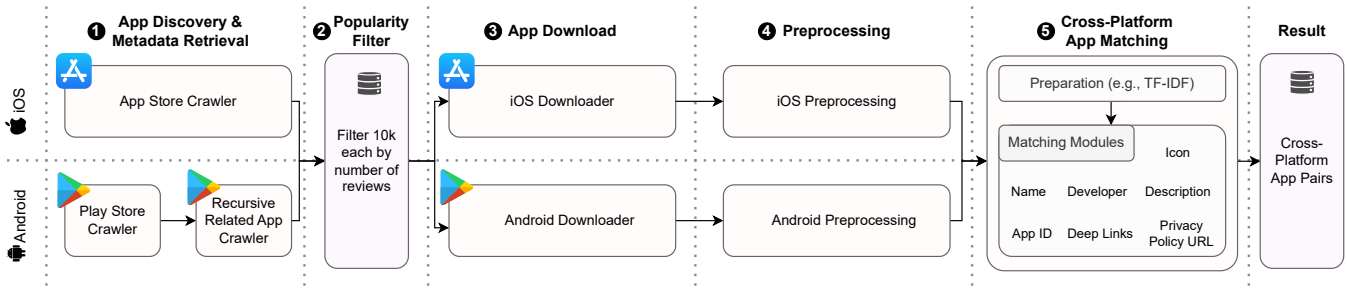


Figure 1: *Overview of our app retrieval and analysis pipeline.* After discovering apps and retrieving their metadata, we filter apps based on their popularity and download them. We then preprocess the apps' (meta)data and feed it into the matcher that tries to match apps between platforms. The final output is the list of matched cross-platform apps.

et al. [52] developed an OS-agnostic approach that analyzes network traffic and found that apps across platforms used and leaked users' personal information. Kollnig et al. [32] further developed a static and dynamic program analysis pipeline to perform a comparative study to assess whether iOS apps are more privacy-friendly than their Android counterparts. They found potential privacy violations on both platforms. Finally, Pradeep et al. [48] performed a comparative analysis on the usage of certificate pinning between iOS and Android apps and found inconsistent implementations of this security feature across platforms.

Related work has already highlighted interesting directions for cross-platform studies on iOS and Android apps. While not all of the above approaches attempted to match apps across platforms and instead more generally compared the ecosystems, we argue that for a sound cross-comparison of apps available on multiple platforms or in different app stores, it is vital to match and compare the corresponding apps from these sources.

App Matching across Platforms. Prior research has already partially applied methods to identify an Android app's iOS version (or vice versa). Work focusing on apps developed with designated cross-platform frameworks has identified these by looking for whether the decompiled APKs load framework-specific classes [4]. To identify cross-platform apps in the broader sense as we defined earlier, smaller-scale studies with datasets of 50–100 apps [16, 28, 52] relied on manual matching of popular apps. Pradeep et al. [48] built their set of app pairs based on a website of crowdsourced software recommendations, also including partial manual verification. On a larger scale, Ali et al. [3] identified over 80,000 app pairs (6.7 %) in their dataset of 2.4 million apps from the Apple App Store and Google Play Store by combining multiple attributes of an app (i. e., app name, category, and developer) to build clusters and identify matching apps in the two analyzed stores. Similarly, Han et al. [25] relied on comparing attributes (i. e., name, description, and developer) of an app. They found 12.2 % cross-platform apps within their dataset of 300,000 Android and 400,000 iOS apps. Kollnig et al. [32] utilized a statistical, frequency-based approach to compute the (cosine) similarity of two apps based on their titles and identifiers to find sets of matching apps.

However, as we show in Section 5.3, the discussed related work has limited accuracy in identifying matching apps, and a more robust app matching method such as ours can increase the soundness, comprehensiveness, and completeness of cross-platform studies.

3 METHOD

In this section, we describe our approach to create a large dataset of popular cross-platform apps from the Apple App Store and the Google Play Store. This requires us to discover apps in these two stores, to download them and their metadata, and to match apps in the two datasets. As displayed in Figure 1, the architecture of our pipeline consists of the following modules:

❶ *App Discovery and Metadata Retrieval:* We crawl both the Apple App Store and the Google Play Store to discover apps (see Section 3.1). Because the Google Play Store has no public listing of all apps, we use the popularity charts and lists of related apps it provides to seed a recursive crawl of related apps until we no longer discover any new apps.

❷ *Popularity Filter:* We select the top 10,000 free apps per store, ranked by their number of user ratings. This number is available for both stores, while (approximate) download numbers are only available on the Google Play Store (see Sections 3.1.1 and 3.1.2).

❸ *App Download:* For each app, we download the app itself and the corresponding metadata from its store listing (e. g., app IDs, price, descriptions, and rating count), which we use for matching and can also be used for future studies requiring cross-platform app pairs. Note that we use *app ID* as an umbrella term for bundle IDs (iOS) and package names (Android).

❹ *Preprocessing:* After downloading the selected apps, we perform various preprocessing steps on both apps and metadata, such as calculating image hashes of icons and static analysis, to gather characteristics that we use for matching. We store the results of this step in a MongoDB database for further processing.

❺ *Cross-Platform App Matching:* Finally, we process the collected data in our matcher to calculate a similarity score between any pair of an iOS app and an Android app and find matching app pairs (see Section 3.2.2).

To evaluate the quality of our cross-platform app matching and to calibrate score thresholds, a ground truth of correctly matched app pairs is necessary. In the absence of such a gold standard, we looked for a proxy dataset. We discovered that both Google and Apple provide their own matching mechanism for cross-platform migration during the setup of a new device. We use Google's previously undocumented API for iOS-to-Android migration to create a list of reference pairs (see Section 3.3) for our empirical evaluation of feature combinations and score thresholds (see Section 3.4).

3.1 App Discovery and Metadata Retrieval

The first step of our analysis pipeline comprises (1) the discovery of a large set of apps in both the Apple App Store and the Google Play Store and (2) the retrieval of app metadata, which we use to select a sample of popular apps that are most likely in active use. This is straightforward for the Apple App Store, as Apple provides a web page that lists all apps available in the App Store [7]. The Google Play Store does not provide such a comprehensive list of available apps but features popularity charts and lists of recommended apps that can be explored recursively. A full comparison of both stores is out of scope for this work; as we wanted to keep our dataset manageable for collecting, processing, and sharing, we evaluated our approach on a subset of apps. Thus, after app discovery, we use the number of user ratings an app has received in its store to select the top 10,000 free apps from each store for further processing. While this metric can be manipulated [49, 69] to make an app appear more popular than it is, related work has found the number of affected apps to be negligible (< 0.7 % of apps) [69]. Unlike the approximate number of app downloads, which is not available on the Apple App Store and has similar potential for manipulation [18], the number of user ratings exists on both the Apple App Store and the Google Play Store. We argue that the number of ratings is a useful metric to select two comparable samples of apps from the two stores and reflects apps that are actively used by users in practice and, thus, the most interesting and significant for further studies.

We performed all our experiments from the university network of TU Wien in Austria. Wherever possible, we used the German locale. Both stores present localized versions of both the store pages and the apps if the developers choose to support multiple languages. However, not all developers provide a localized or an English version of their app, which can have an impact on the collected data, as, e. g., the language of descriptions can differ between apps. In general, app store data collection is a trade-off between metadata consistency and availability of localized apps.

3.1.1 *Apple App Store.*

Discovering iOS Apps and Crawling Metadata. Apple conveniently provides a web page (the "iTunes preview") that lists iOS apps available in the App Store [7]. The page is organized by app categories and within each category by the first character of the app name. In theory, these listings can be scraped to obtain all available App Store IDs (not to be confused with the *bundle ID*, which uniquely identifies a downloaded app) for both iOS and iPadOS apps. However, the web interface only provides filters to find apps starting with uppercase letters, special characters, or numbers. If we only relied on these filters, we would miss apps whose name starts with, e. g., a lowercase letter or an umlaut. To circumvent this limitation, we used an existing metadata crawler for the Apple App Store [46] and extended it to first select an app category (e. g., finance) and then iterate over all alphanumeric characters (including umlauts and special characters) to obtain the app store URLs for all apps in each category via the provided API. For example, the request for finance apps starting with 'A' goes to `https://apps.apple.com/de/genre/ios-finance/id6015?letter=A`).

In the next step, we scrape the available metadata (e. g., developer name, bundle ID, or the link to the privacy policy) for each iOS app from the store's web listing and store it in our MongoDB database for further processing. Using this process, we identified 795,037 iOS apps, considerably more than we would be able to obtain for Android via the Google Play Store, which is missing a comparable listing of all apps (see Section 3.1.2).

Popularity Filter. To create a more balanced and comparable set of apps for both stores, as well as include apps that are both free and widely used, we excluded all apps from the Apple App Store that were not available for free (i. e., price > 0) and only considered apps that had received at least one user rating. After applying both filters, this left us with 168,573 (21.20 %) apps. To allow the download to be conducted in a reasonable time frame, and to keep the final dataset size manageable, we chose to use the top 10,000 apps based on the number of user ratings.

Downloading iOS Apps. We download the selected top 10,000 apps using IPATool [2]. We linked Apple (user) IDs with an iPhone 8 running iOS 16.1.2 (released in November 2022). In order to prevent inconsistencies between the downloaded app and its metadata, which could happen when an app or its metadata was updated between discovery and download, we also re-downloaded the store metadata at the same time.

For a given Apple (user) ID, Apple's billing server only allows users to purchase 300 apps per day, regardless of whether they are free or paid. To overcome this limitation, we parallelized the download using three Apple IDs, allowing us to download 900 apps per day. For the pipeline setup, we created three different user accounts on a Mac Mini M1 running macOS Monterey 12.5. This step is necessary because IPATool requires access to the macOS keychain and currently only supports one keychain item per Mac user. Hence, we started three terminals, logged into each user account, and unlocked the keychain for each user for access within the terminal session. With these account limitations in place, the initial download of 10,000 iOS apps took roughly 12 days, but it could be sped up with additional Apple IDs to parallelize with.

3.1.2 *Google Play Store.*

Discovering Android Apps and Crawling Metadata. Crawling the Google Play Store is more complex because there is no (publicly available) way to exhaustively list the most popular apps across all categories, let alone obtain a full listing of all its apps. Previous work often either followed "random" links that lead to additional apps in the store [71] or crawled the top apps in each app category [6, 43]. In this work, we extended the latter approach: We first compiled a seed list of app IDs (called *package name* on Android) from the most popular apps offered for free ("Top Charts"), which comprised 7,659 seed apps. Starting from this list, we recursively explored all apps suggested by the Google Play Store when viewing an app listing, which includes related apps, apps by the same developer, and recommended apps ("users also installed" in the web interface) until no new apps were found, which happened at a recursion depth of six. This resulted in a list of 53,232 unique package names. We chose to use the web interface to discover apps, because the undocumented API used by the official Google Play Store app (see below) imposed a rate limit on searches that would have increased the run time by several orders of magnitude.

Popularity Filter. As with the Apple App Store, we extracted and stored app metadata from the Google Play Store's app listings. Even though we started the metadata retrieval on the same day the app discovery process was finished, it failed for 55 apps (< 0.1 %), as they no longer existed in the store, leading to a set of 53,177 known Android apps. As for iOS apps, here we also selected the top 10,000 apps by the number of user ratings to download.

Downloading Android Apps. While information about apps can also be accessed with a desktop browser, downloading apps is intended to be only possible on an actual mobile device, but unlike iOS apps, Android apps are not tied to a specific device. There is an undocumented API [9] that allows the Google Play Store app to interface with the Google Play Store and not just search for apps, but also download all necessary files to install them. Its usage requires a Google Play Service Identifier (GSFID) and a corresponding device token, which is bound to a specific device profile and requires a Google account that has accepted the Google Play Store's terms of service. There have been efforts in the past to create open-source libraries that allow the use of this API, but they require regular updates to stay compatible with changes to the Google Play Store. Additionally, we found that using them for app discovery is very slow, as it has a much lower rate limit than the web interface. To download the top 10,000 apps of interest, we extended the existing library `gplaycrawler` [47] to support a more recent phone, a Google Pixel 3a running Android 13 (released in August 2022). We ran two parallelized download processes to achieve a rate of roughly 3,000 app downloads per day, resulting in a download time of three days for the 10,000 Android apps. As with the iOS apps, we re-downloaded the store metadata at the same time for consistency.

3.2 Cross-Platform App Matching

The question whether two or more apps are identical or similar can be approached from different perspectives and is, to some extent, defined by the scope of the study. For example, apps with the same or an extremely similar feature set (e. g., two calculator apps) could be considered similar in an experiment that compares apps based on their features (e. g., to identify the most privacy-friendly calculator). Another, more strict approach is the one we use in this paper: to identify cross-platform apps for iOS and Android that human expert judgment would consider to be "the same." This includes pairs of apps that are designed for the same use cases, offer the same core functionality, and can be attributed to the same provider, but are made for different platforms and offered in different stores. For example, TikTok is available for both iOS and Android, serves the same use cases, offers the same core features on both platforms, and both apps are provided by TikTok Ltd. Thus, a human tasked with finding cross-platform apps for iOS and Android would conclude that these are the iOS and Android versions of the "same" app.

The goal of our app-matching is to automate the identification of cross-platform apps. Hence, throughout the rest of the paper, we use the term *same app* to denote apps that either human judgment or our human-verified reference pairs (see Section 3.3) would consider to be equivalent on both iOS and Android. By contrast, we use the term *matching apps* to refer to apps that our matching algorithm has deemed to be equivalent on iOS and Android.

Finding the same app for both iOS and Android is the prerequisite of any sound cross-platform analysis of apps available for both ecosystems. As discussed in Section 2, previous work conducted this step by manually identifying these apps [3, 25, 28], which is not feasible for larger datasets. Other work relied on simple features for app matching without systematically verifying the accuracy of the resulting matches. For example, Kollnig et al. [32] used app names and IDs to create cross-platform app sets to compare their privacy properties. Han et al. [25] used app and developer names as well as app descriptions to match over 20,000 apps but only used 1,300 manually selected app pairs for their privacy analysis, while Ali et al. [3] used similar app names and developer names to find 80,000 app pairs, of which they manually verified 100.

Names alone can be a poor match for the same app between the App Store and the Google Play Store, as illustrated by the examples "Babbel – Language Learning" (app name, iOS) vs. "Babbel – Learn Languages" (app name, Android) and "1&1 Mail & Media GmbH" (developer name, iOS) vs. "1&1 Telecom GmbH" (developer name, Android). In both cases the names share the same prefix, but their normalized Levenshtein distances, a metric for measuring differences between strings, are 0.5 and 0.48, respectively. Hence, these two pieces of metadata are insufficient to determine an app pair with high confidence, and more sophisticated methods are needed to enable a robust, reliable, and comprehensive matching of apps between ecosystems. Another limiting factor for large-scale app matching is performance. While all types of app (meta)data can contribute to identifying related apps across platforms, there are features, e.g., ones extracted through static or dynamic code analysis, that are more time-consuming and resource-intensive to process. Scaling them to market sizes is infeasible if a one-on-one score has to be computed for each candidate pair.

3.2.1 General Matching Approach. As input for the matching module of our pipeline, we use some of the information scraped from the stores, as well as extracted from the apps themselves. The decision to include the latter is based on the assumption that downloading the apps is required anyway for further cross-platform analysis of app pairs. In the following, we use the term *feature* for a data point that describes an app and that we either extracted from the respective store listing or directly from the app. Table 1 lists all features we use for matching. We selected these based on related work and our own investigation of what types of (meta)data were available for both iOS and Android apps and could be obtained and evaluated via light-weight analysis. We scraped all but one of the used features (i. e., app name, developer name, app ID, privacy policy URL, app description, and app icon) from store listings, whereas information about deep links needed to be extracted from the apps themselves. We used the collected features to compute a *matching score* that indicates the similarity of two given apps (see Section 3.2.2). For this, we computed subscores for each feature and later combined them into a single score (see Section 3.4).

Preprocessing. Some features require preprocessing or benefit from pre-computing values, such as the TF-IDF for the description, deep link extraction from the apps, or icon hashes. As shown in Figure 1, some of these steps are performed in the preprocessing module and others in the matcher. We describe these steps along with each feature in Section 3.2.2.

Feature	Type	Source	Comparison
App Names	String	Store	max(SPL, LD)
Developer Names	String	Store	max(SPL, LD)
App IDs	String	Store	max(SPL, LD)
Privacy Policy URLs	String	Store	max(SD, SPL, LD)
App Descriptions	String	Store	TF-IDF
App Icons	Image	Store	max(ahash, phash, whash)
Deep Links	String	App	max(SD(US), SD(AL))

Table 1: *Features used in our matcher.* App ID is the umbrella term for iOS bundle IDs and Android package names. We use the shared prefix length (SPL) and Levenshtein distance (LD) for names, as well as the "same domain" for URLs (SD). For longer strings (i. e., the descriptions) we use the term frequency-inverse document frequency (TF-IDF). For deep links we use URL Schemes (US) and App Links (AL). We compare icons using the average, perceptual, and wavelet hash. We normalized all values to the $[0, 1]$ range.

String Comparison. Multiple features we investigated for app matching are composed of (rather short) strings, such as app name or developer name. To compare these, we used two different metrics for string comparison: (1) shared prefix length and (2) Levenshtein distance. For an array of strings, the *shared prefix length* (SPL) provides the common prefix shared by all strings in the array. This metric is especially useful for comparing app and developer names, because they often start with brand names followed by suffixes that can be spelled differently and thus may diverge between platforms, e. g., "Ltd." and "Limited." The *Levenshtein distance* (LD) measures the difference between two strings by computing the minimal number of operations (e. g., insertion or substitution) needed to change one string to another. In all matching approaches that require string comparison, we computed both metrics separately and used the maximum for our final score.

3.2.2 Matching by Feature. We evaluate the seven features listed in Table 1 for app matching as follows:

- *App Names.* We use string matching as described above to compare the names of apps provided by their app store listings.
- *Developer Names.* In the same vein, we compare the names of app developers provided in the app store listings.
- *App IDs.* We compare the strings that uniquely identify each app on a device, i. e., bundle IDs (iOS) and package names (Android), again using the string-matching approach.
- *Privacy Policy URLs.* We check if the URLs to the apps' privacy policies, as provided in the store listings, lead to the same policy. For this, we check if the URL contains the same full domain and apply the usual metrics for string matching, Levenshtein distance and shared prefix length. We separately check the domain and URL in case one of them is localized for a platform but not the other. We do not consider the content of the privacy policies, as the extraction and preprocessing of privacy policies from websites have shown to be complex tasks in themselves [26].
- *App Descriptions.* We compute the similarity of the description texts from the app store listings using term frequency-inverse document frequency (TF-IDF). For this, the matcher first uses the Python module `scikit-learn` [44] and its `TfidfVectorizer` to convert the raw description texts from all iOS and Android apps into a vectorization of TF-IDF features. We provided the `TfidfVectorizer` with a combined list of stop words from the Natural Language Toolkit [39] in English and German, which are removed from the descriptions and thus not included in the vectors. Then we calculate the cosine distances from the vectorized iOS descriptions to the vectorized Android descriptions. The result is an $N \times M$ matrix of similarity values in the $[0, 1]$ range, where N is the number of iOS app descriptions and M is the number of Android app descriptions. This has the benefit that the matching only requires a lookup of the similarity score for the corresponding app pair in the matrix of similarity values.
- *App Icons.* We match app icons through image hashes, which, unlike cryptographic hashes, produce similar output hashes for similar input images. Using the Python library ImageHash [11], we compute, for each app icon in our dataset, its average hash, perceptual hash, and wavelet hash. Using a comparison function that counts how many bits are different, we calculate the distance between two hashes of the same kind, which indicates whether two apps use the same or a very similar icon. We use the highest similarity of the three hashes.
- *Deep Links.* The declaration of deep links in an app allows to directly open specific URLs in this specific app, rather than in the standard browser [20, 61]. As it is in the interest of app developers not to differentiate between platforms in deciding which links should redirect users to the different versions of their app, we assume that these are unique per app but identical across platforms. There are two types of deep links: scheme URIs (e. g., `myapp://`), which are named *Custom URL Schemes* on iOS and *Scheme URLs* on Android, and HTTPS URLs (e. g., `https://myapp.com`), which are named *Universal Links* on iOS and *App Links* on Android. To match apps via their deep links, we compute for two candidate apps the similarity of all declared Custom URL Schemes (iOS) with Scheme URLs (Android) and Universal Links (iOS) with App Links (Android). For Universal Links and App Links, we first remove prefixes such as wildcards (`*`) and "`www.`" to normalize them. We compare the resulting sets of URLs via exact string matching; for each candidate pair, the matching score is the higher percentage of identical links for each of the two deep link types.

3.3 Reference Pairs

To determine the best approach to combine subscores and to evaluate the performance of our matcher, we investigated ways to obtain reference pairs of already matched apps and found a useful data source: *mechanisms for data migration between devices and app ecosystems*, which have not been discussed in the literature before. Both Android and iOS support the migration of data and (some) installed apps when a device is first set up from an existing device, even from a different platform. In this process, the target OS takes the list of installed apps on the user's previous device (connected via cable or WiFi) and tries to identify and download their counterparts for the target platform. We use the data obtained from this process as *reference pairs* to evaluate the performance of our matcher.

3.3.1 Google API for iOS-to-Android Migration. The "Data Restore Tool" [21] (package name: `com.google.android.apps.restore`) is an Android app that collects the bundle IDs of all apps installed on a connected iPhone and sends them to an API endpoint that either returns the matching Google Play Store package names or indicates that a match could not be found. As this service is used to automatically install apps equivalent to those installed on an iPhone when migrating to Android, it can be assumed that Google is confident in the results. The API (`androidbackupmigration-services-pa.googleapis.com`) requires a bearer token that is generated with the backup migration scope set via the migration tool and is valid for one hour. It is not the same as the GSFID and token pair used for downloading Android apps. To obtain this bearer token, we set up a Google Pixel 6a with Android 13 and rooted it with `magisk` [68] before starting the migration. Next, we used the Android Settings app to trigger the migration process from an iPhone 8 running iOS 16.1.2. Using `mitmproxy` [35], we intercepted the phone's traffic and received the necessary bearer token. With this token, we sent the 775,561 iOS app bundle IDs to the backup migration API and received a JSON containing matching Google Play Store package names. An example of this is shown in Listing 1. Possible responses are the indication that a free or paid match was found, along with its package name; that no app was matched; or that the app has been deny-listed. As this is an undocumented API, we can only speculate what the latter indicates.

3.3.2 Manual Verification. We validated the app matching returned by the Google migration API for the 300 app pairs with the highest number of user ratings in the Apple App Store. For this, we manually inspected the store pages for the returned app identifiers for whether they appeared to refer to the "same" apps on the two platforms. Since only a bundle ID is provided as opposed to an App Store ID, we resolved them to Apple App Store pages using URLs of the pattern `http://itunes.apple.com/lookup?`*`bundleId=test.bundle.id`*`&country=de`. Two of the authors each checked half of the 300 app pairs and annotated whether the apps matched or if the result returned by the API looked off. After discussing the questionable pairs, a third researcher acted as a tie-breaker to classify a pair of matched apps as the "same" or not. For three iOS apps, we found that their store page was no longer available, and there was one pair (bundle ID `com.yk.qrcoder`, package name `com.pastapizza.qrcode`) that we deemed to be not a real match. We also investigated the `NO_MATCH` result and found examples of false negatives: The iOS app `com.rockwellventures.woah` should have been mapped to the Android app `app.woah`, which we verified by visiting the apps' website `https://woahtheapp.com/`. This hints at Google itself using heuristics for app matching. We discuss these reference pairs in more detail in Section 6.

3.3.3 Android-to-iOS Migration. For migration from Android to iOS, the process is similar: Android sends the package names of the installed apps to the new iPhone. However, in this direction the matching is performed by the OS and not by a separate app. Intercepting this would require disabling certificate pinning at the OS level, which in turn requires a jailbroken iPhone. Therefore, we leave the exploration of the Android-to-iOS migration API and its suitability for app matching to future work.

```
1   { "bundleId": "app.organicmaps",
2     "packageName": "app.organicmaps",
3     "result": "MATCH_FREE"
4   },
5   { "bundleId": "ch.threema.iapp",
6     "packageName": "ch.threema.app",
7     "result": "MATCH_PAID"
8   }
9   { "bundleId": "com.rockwellventures.woah",
10    "result": "NO_MATCH"
11  },
```

Listing 1: *Excerpt from the app matching results (reference pairs) provided by Google's migration API.* Results indicate if a free (line 3), paid (line 7), or no app was matched (line 10).

3.4 Combination of Features and Evaluation

After we computed all subscores for the individual features, as described in Section 3.2.2, the matching module combines them into a final similarity score for each candidate pair of apps. To compute this *matching score*, we implemented three different approaches: (1) the average of all subscores, (2) their median, and (3) a clamped linear function. We also re-implemented additional approaches from related work [3, 24, 27, 32] for comparison (see Section 5.3).

We used the reference pairs (see Section 3.3) for which we had both the iOS app and Android app in our set of downloaded apps to evaluate how many of these expected pairs we could find with each approach. This list contains 3,467 app pairs on which we evaluated our three scoring approaches and found the average to perform best (see Section 5.2.1). In case the matching result based on the average score diverged from the reference, we manually inspected the pairing to understand whether the result was a false positive or if the reference pairing from Google's migration API was flawed. We further evaluated the accuracy of our approach by inspecting a sample of the matched apps that were not part of the reference. We present the results in Section 5.2.

4 RESEARCH ETHICS

For this work, we did not collect, store, or process personal data, and we did not involve any human subjects. Thus, our study was not required to be reviewed by an Institutional Review Board at any of our institutions. Nevertheless, we carefully considered the potential ethical implications of our study.

Both the Apple App Store and the Google Play Store implement protection mechanisms so that one cannot download apps at excessive rates, or easily extract the source code of these apps (see Section 3.1). Our crawler honors the stores' rate limitations for downloading apps and metadata and does not generate traffic volumes that could jeopardize the availability or functionality of the stores, e. g., by inadvertently performing a denial-of-service attack by sending too many requests in a short period of time. Still, our approach generates traffic that uses some resources of the store providers, which otherwise would not have been needed. We argue that all studies that involve the discovery and/or retrieval of iOS and Android apps and/or their metadata at scale share this issue and that our method follows commonly accepted procedures.

To foster reproducibility and to reduce the barrier of entry for other researchers in this area, we provide the downloaded apps for research purposes upon request. This also prevents the additional strain on the stores that a replication of our data collection would cause. While it could be argued that facilitating the collection and sharing of such large datasets could violate app store policies, this again is common practice for studies in the mobile app space. Furthermore, our dataset contains only free apps, such that sharing does not deprive app developers of revenue. Our pipeline is based on already publicly available tools that tech-savvy users can use to download a comparable set of apps.

Finally, our insights into the efficiency of different features for app matching could inform malicious actors who aim to create phishing or copycat apps for apps available only on iOS or Android and publish them in the respective other store. However, both the Apple App Store and the Google Play Store claim to have vetting mechanisms in place to verify developers and prevent such abuse.

5 RESULTS

In this section, we provide an overview of the collected iOS and Android app datasets, present the results of our matching approach, and evaluate the performance of our analysis pipeline for app discovery, download, and matching, as well as prior approaches.

5.1 Dataset

Table 2 provides an overview of the datasets that resulted from our app discovery and download process, which we conducted from October 15–23, 2023. Our final app corpus consists of 9,943 iOS and 9,942 Android apps (we failed to download 57 and 58 apps, respectively, as they were not available anymore).

To obtain the set of reference pairs, we requested a total of 775,561 out of the 795,037 discovered iOS apps to be matched by the Google migration API and received 271,457 (35.00 %) matches with free apps and 10,526 (1.36 %) matches with paid apps. The remaining 493,578 (63.64 %) bundle IDs returned `NO_MATCH`. This could be due to apps only being available for iOS or not covered by Google's (opaque) data migration API.

Out of the 9,943 iOS apps we downloaded, 7,671 (77.15 %) were paired with an Android app by the Google Migration API. In the remainder of this paper, we refer to this subset of the returned reference list as the *Reference Pairs*. For the remaining 2,272 (22.85 %) iOS apps we could only resort to manual validation to determine whether our matcher worked correctly. We label this subset the *No Reference Pairs*. For 3,467 (34.87 %) iOS apps our downloaded corpus contained the matching Android app from the *Reference Pairs*. This is a subset of the *Reference Pairs* and we label this intersection between the *Reference Pairs* and our corpus of downloaded iOS apps and Android apps *Verifiable Set*. For these reference pairs, both apps exist in our downloaded dataset. We use this subset as one indicator to automatically validate the quality of our matching approach described in Section 3.4. We refer to the 4,204 iOS apps present in the *Reference Pairs* but not in the *Verifiable Set* set as *Unverifiable Set*, as we assume that our matcher cannot get true positives from this set because the respective Android apps are not in our downloaded dataset.

	Discovery	Filtering	Download
iOS Apps	795,037	168,573 (21.20 %)	9,943 (5.90 %)
Android Apps	53,177	15,509 (29.16 %)	9,942 (64.10 %)

Table 2: *Number of apps in our Android and iOS app datasets* after discovery (i. e., enumerating them in the store), filtering (for popularity and free apps), and download.

	Verifiable Set	*Unverifiable Set**	*No Reference Pairs**
Matches	3,322 (99.58 %)	7 (2.47 %)	82 (61.19 %)
Mismatches	14 (0.42 %)	-	-
App Missing	-	256 (90.46 %)	50 (37.31 %)
Unknown	-	20 (7.07 %)	2 (1.49 %)
Total	3,336 (33.55 %)	283 (2.85 %)	134 (1.35 %)

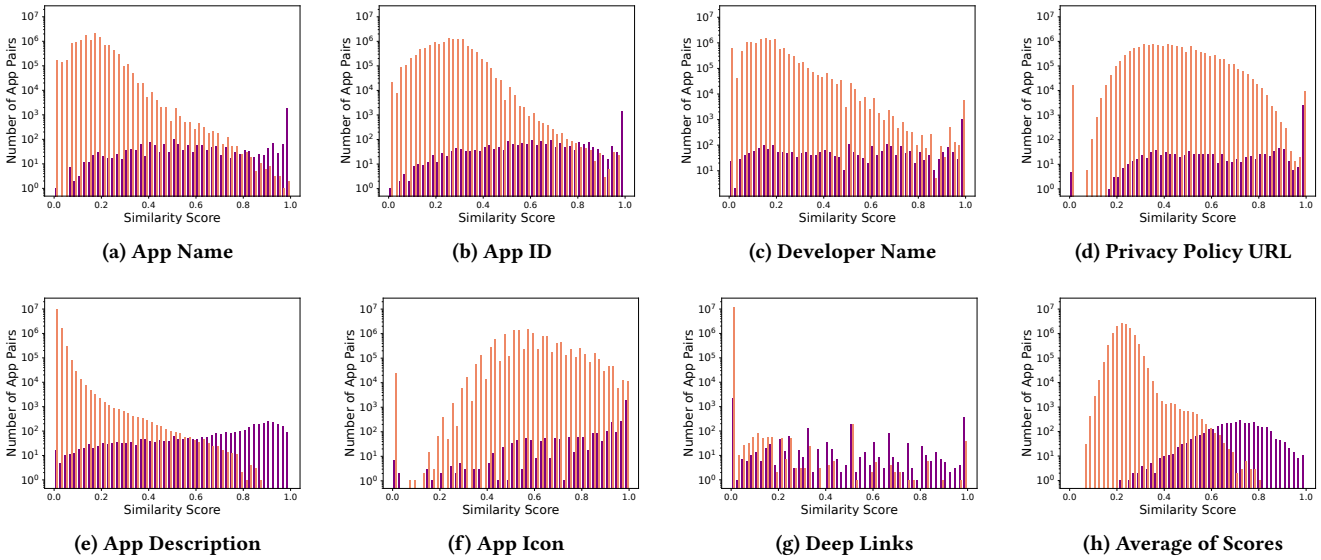
Table 3: *Quality of our matches with a matching score* ≥ 0.5 *(n = 3,753).* * indicates manually verified matches.

5.2 Cross-Platform App Matching

5.2.1 Scoring Approaches. To assess how to best combine feature subscores into a final matching score for two candidate pairs, we investigated different metrics: (1) the average of all subscores, (2) their median, and (3) a clamped linear function. We compared the performance of each scoring approach by its ability to match the same app pairs as Google's migration API for the *Verifiable Set*. The average score outperformed the other approaches by correctly identifying matches for 3,432 apps (98.99 %) in the *Verifiable Set*. The second best approach, clamped linear function, only correctly identified matches in 3,387 cases (97.69 %). The median of the feature subscores performed worst, with only 3,300 (95.18 %) correct matches for the *Verifiable Set*. Thus, for the remainder of this work, we use the feature score average for app matching. More complex approaches might yield better results; we leave these to future work (see Section 6). However, compared to related work, our approach shows a notable improvement, finding cross-platform matches for approximately 34 % of apps.

5.2.2 Matching Accuracy. We evaluated the accuracy of our matching approach against the app pairs in the *Reference Pairs*. As described above, we obtained the matching score for two candidate apps by computing the average feature rating.

We correctly matched 3,432 (98.99 %) of the app pairs in the *Verifiable Set*, using the highest score. Our matching was incorrect in 35 (1.01 %) cases, i. e., an app pair was matched that does not belong together according to the reference pairs. We manually investigated all 35 cases of mismatches with the corresponding reference pairs and found that 21 had a matching score below 0.5. If we take the average score of 0.5 as a threshold for matching accuracy, only 14 false positives remain. Even though this threshold also removes 141 (4.11 %) of true positive matches, the correct matching rate stays considerably high (95.82 %). Because some apps have no pairs, the threshold allows a matcher to return no result; without it, the matcher would pair every app, even if it does not have a cross-platform equivalent. The threshold also allows to tune the matcher: By increasing it, the number of false positive matches decreases, but the number of false negative non-matches increases.



(a) App Name (b) App ID (c) Developer Name (d) Privacy Policy URL

(e) App Description (f) App Icon (g) Deep Links (h) Average of Scores

Figure 2: *Distribution of the similarity scores for the individual features, as well as the average matching score.* Using the *Verifiable Set*, ● shows matches according to the reference pairs, while ● shows mismatches, i. e., all possible cross-combinations of apps we downloaded that both appear in the *Verifiable Set* but are not matches. Note that the number of app pairs on the y-axis is plotted on a log scale since there are far more mismatches than matches.

We manually verified all matching results with an average score above the threshold from the *Unverifiable Set* and the *No Reference Pairs* to evaluate our matcher against these datasets. For these apps our matcher found good matching candidates, but it is unclear whether the identified apps are actual matches, as they are not in the *Reference Pairs*. Table 3 shows the performance of our matcher for these datasets. In the case of the *Unverifiable Set*, our matcher could find 7 new matches that were not covered by the reference pairs. In 256 cases, the matcher found matches that were wrong according to the reference. However, these are cases in which the matcher could not have found the correct matching app (because it does not exist in the set of downloaded Android apps) but it falsely identified one; hence the dash in Table 3. Therefore, a high error rate was expected for the *Unverifiable Set*. Regarding the *No Reference Pairs*, our matcher could successfully find 82 previously unknown matching pairs, with a much lower mismatch rate of only 50 pairs. For these apps, the Google API did not find a matching app but our matcher did.

Despite our manual effort, we could not classify all matches in the sets *Unverifiable Set* and *No Reference Pairs*, as in some cases the Android reference app was either not available in the Google Play Store or we could not manually match the apps with certainty. Overall, our results indicate that, as long as the corresponding apps on both platforms exist in the dataset, our matcher identifies them with high accuracy and a low rate of mismatches. Otherwise, the threshold can be fine-tuned to trade off more matches with an increased number of false positives.

5.2.3 Effectiveness of Different Features. Figure 2 visualizes the distribution of scores of all the features we used, for both matches and mismatches. The set of matches is composed of the matches from the *Verifiable Set*. For the mismatches, we first compared all candidate pairs of iOS and Android apps in the *Verifiable Set* and then removed the reference pairs from this set. This approach gives us comparisons between apps that we know have at least one match in the *Verifiable Set*, but excluding the matches we obtained from Google's migration service.

A perfect feature would yield two distinct and non-overlapping distributions for matches and mismatches. Unfortunately, no feature by itself achieves this. While app names, app IDs, and developer names have often been used in the past, they require a high threshold that would miss a lot of genuine matches. The comparison of descriptions is a better feature, as a threshold of 0.95 seems to provide only matches but still misses expected pairs. The average of scores manages to tame the noise of the feature subscores into a less even and less overlapping distribution.

The difference between the highest score to the second highest score for the *Verifiable Set* is on average 0.28, with a standard deviation of 0.12. The smallest difference was 0.009 for the iOS app `com.avast.ios.security`, which was correctly matched with `com.avast.android.mobilesecurity`, with a score of 0.5211. The next best match was `com.avast.android.secure.browser`. In general, higher scores resulted in a larger difference to the second highest score.

None of the features show strong co-linearity. The Pearson correlation coefficient between all features is highest between the description matching and deep link matching, with a value of 0.32. All other feature pairs have a coefficient of 0.17 or lower. To measure the influence of each feature, we performed an additional ablation study by removing a single feature and calculating the scores for a fixed threshold of 0.5, presented in Table 4. To our surprise,

Feature (Sub)Set	TP (%)	FP (%)	FN (%)
All seven features	3,322 (95.82 %)	14 (0.40 %)	131 (3.78 %)
w/o Deep Links	3,391 (97.81 %)	27 (0.78 %)	49 (1.41 %)
w/o Developer Name	3,341 (96.37 %)	18 (0.52 %)	108 (3.12 %)
w/o Description	3,279 (94.58 %)	36 (1.04 %)	152 (4.38 %)
w/o App Name	3,258 (93.97 %)	35 (1.01 %)	174 (5.02 %)
w/o App ID	3,250 (93.74 %)	19 (0.55 %)	198 (5.71 %)
w/o Privacy URL	3,206 (92.47 %)	7 (0.20 %)	254 (7.33 %)
w/o App Icon	3,173 (91.52 %)	20 (0.58 %)	274 (7.90 %)

Table 4: *Impact of different subsets of features on matching rates*, with a matching threshold of 0.5, for the *Verifiable Set*. w/o indicates that the specific feature was omitted. True positives (TP) increase without some features, but at the cost of false positives (FP) and false negatives (FN).

Approach	TP (%)	FP (%)	FN (%)
Our Approach	3,322 (95.82 %)	14 (0.40 %)	131 (3.78 %)
Hu et al. [27]	2,908 (83.88 %)	303 (8.74 %)	256 (7.38 %)
Kollnig et al. [32]	1,781 (51.37 %)	403 (11.62 %)	1,283 (37.01 %)
Ali et al. [3]	588 (16.96 %)	0 (0.00 %)	2,079 (59.97 %)
Han et al. [24]	514 (14.83 %)	0 (0.00 %)	2,953 (85.17 %)

Table 5: *Comparison of our approach with previous matching strategies;* based on the true positives (TP), false positives (FP) and false negatives (FN) for the *Verifiable Set*.

removing the deep link or developer name features increased the number of true positives, but also of false positives. There appears to be a three-way trade-off between the number of true positives, false positives, and false negatives. We consider the inclusion of all features a suitable default for an overall high matching rate with low numbers of false positives and false negatives. Our implementation allows to change the features that are included in the matching score to suit the requirements of different applications.

5.3 Comparison With Existing Approaches

We evaluated our matcher's performance against four different existing approaches for cross-platform app matching [3, 24, 27, 32]. The code by Ali et al. [3] is publicly available, and Kollnig et al. [32] kindly provided us with their code. We re-implemented the other two approaches [24, 27] based on their respective papers; however, as not all details were given, we had to make assumptions at times.

Ali et al. [3] used an exact string match of both app name and developer name as a matching metric. Han et al. [24] also used exact string matching of app and developer names, but additionally included a Vector Space Model to match the descriptions of apps with a similarity threshold of 0.45. As they did not provide implementation details, we used our TF-IDF implementation as the Vector Space Model. Hu et al. [27] detected pairs of cross-platform games on Steam and consoles by calculating their similarity based on the games' names, developers and publishers. Since our approach concerns a different domain (mobile apps) and the apps in our dataset do not contain publisher information, we omitted this feature and calculated the similarity based on app names and developers only. For preprocessing we followed the paper but used a German stop word list, as we had downloaded our app dataset from German stores. Kollnig et al. [32] individually calculated the similarity of app IDs and app names using TF-IDF with a similarity threshold of 0.95. As their goal was to *create sets of cross-platform apps instead of pairwise app matches*, they included all candidate apps above the threshold into their datasets. Hence, we adapted their approach and chose only the candidate with the highest average score across both features.

We ran all four approaches on the *Verifiable Set* and always selected the candidate with the highest score to make the results comparable to our approach. Table 5 shows the results. The two approaches using exact matches yield less than 17 % of true positives, but also no false positives. Kollnig et al.'s [32] approach performed better than the exact string matching approaches and was able to find around 51 % of correct matches. The method by Hu et al. [27] performs well on our dataset but has approximately 21 times more false positives than our approach and finds 424 fewer app pairs. However, this approach was originally designed for a different domain, Steam and console games.

5.4 Performance

The full sets of 9,943 iOS and 9,942 Android apps took 12 and three days, respectively, to download for the first time. Re-downloading iOS apps (e.g., to fetch new versions) is faster if an app has been downloaded before. The downloaded apps, including all metadata, have a size of approximately 2 TB (roughly 100 MB per app).

To find the best match for an app, we needed to compare each iOS app with each Android app, which resulted in $9{,}942 \times 9{,}943 = 98{,}853{,}306$ comparisons. We parallelized the matching process to compare the apps in a reasonable time. On a 16-core machine with 32 GB RAM and 14 worker threads, the matching process of all pairs took less than 30 minutes. Comparing all iOS and Android apps (see Section 5.2) resulted in a MongoDB collection of 98,853,306 pairs of app IDs (iOS, Android) with their respective matching score and all corresponding feature scores, that required roughly 10 GB of storage capacity (6 GB if compressed).

Besides sufficient storage capacities for downloading apps and metadata, the amount of required RAM for the app matching process scales with the number of worker processes. Each worker process requires roughly 1.2 GB of RAM, as the processed data needs to be held in memory by each process. Additionally, the MongoDB database used to store the resulting app pairs will occupy up to 12 GB of cache in RAM during matching.

6 LIMITATIONS & FUTURE WORK

Our work aims to find the "same app" across platforms. However, there is no clear definition of what "same" means in this context, and it partly depends on human judgment, which is difficult to automate. In our case, the paper's authors manually verified a subset of the matches. One direction for future work is to perform a user study on how end users perceive and identify cross-platform apps. However, in this context human judgment can also be exploited by potentially unwanted apps, as evidenced by copycat or counterfeit apps that attempt to closely resemble popular apps to trick users into downloading them, which could have been automatically detected through icon similarity [50, 62].

Dataset Coverage and Filtering. We were able to enumerate all listed apps from the iTunes preview of the Apple App Store, but a similarly complete listing for the Google Play Store does not exist. Hence, we were limited to the apps we discovered using other methods, which include bestseller lists, popular apps per category, related apps, and apps from the same developer. We evaluated our pipeline for popular Android and iOS apps, as they are the most likely to have actual matches. As an indicator for popularity we used the number of user ratings, as this metric exists in both stores. We believe that ratings sufficiently reflect popularity, as crawlers and incentivized installs can easily influence download counts [18], whereas ratings, while they can be manipulated as well [49, 69], are representative of actual app usage. Future work could extend the filtering criteria or expand the dataset to include the long tail of less popular apps from both stores. However, as mentioned above, the collection and analysis of less popular Android apps are hindered by the inability to fully enumerate apps in the Google Play Store.

App Store Localization. We used the German versions of the Apple App Store and Google Play Store and accessed them from TU Wien in Austria, resulting in a set of apps that are supposed to comply with the General Data Protection Regulation (GDPR). Our pipeline can easily be set up and run from other jurisdictions to create datasets for the comparison of apps with a focus on different privacy laws. However, store localization, e. g., different local languages for the metadata, can pose issues (see Section 3.1).

Alternative Marketplaces. Beyond the Google Play Store, other stores provide apps for Android devices, such as the Huawei AppGallery, Amazon Appstore, Samsung App Store, or the F-Droid Store for open-source apps. Our work considers only the Google Play Store as a source of Android apps, as it is the largest and most popular source of apps for Android devices. Future work could investigate the cross-platform matching of apps in and between these additional app stores, including ones for iOS, since Apple has to open up the app distribution at least for users in the EU [8].

Matching Ground Truth. We used pre-matched app pairs obtained from a Google API for data migration from iOS to Android to validate our results. We also found Huawei [30] and Samsung [56] to provide similar migration apps, even with hard-coded lists of app pairs, but on a very limited scale (662 and 47 pairs, respectively). In the opposite direction, Android to iOS, such a mechanism exists on the iOS level. We leave it to future work to reverse engineer Apple's matching functionality. However, neither the Google nor the Apple migration tools provide any documentation on how these matchings are created or vetted, making the quality of the used reference set unclear. Yet, our manual analysis of 300 reference pairs showed that the Google API was only incorrect or incomplete in 1 % of the cases (see Section 3.3.2). One direction to enhance the robustness of this process might be to use additional sources to create and maintain a list of vetted cross-platform app matchings that could serve as the ground truth for future work.

Features and Feature Combinations. Our approach follows a rather straightforward, yet effective and efficient method to finding matching apps. However, as shown by Hageman et al. [23], some of the features we used in our study, such as an app name or developer name, can be (1) spoofed by a malicious actor and (2) inconsistent even between apps from the same developer within a given store. By using (1) the most popular and thus likely vetted apps and (2) a more fuzzy string matching instead of exact matches on app store metadata, we limit the potential effects of this on our study.

It is worth noting that calculating pairwise similarity needs to be efficient because finding the best matches of apps requires $O(n^2)$ comparisons. Future work can explore other approaches for cross-platform app matching by utilizing different feature sets and more sophisticated feature combination and matching methods that profit from, e. g., machine learning or probabilistic models.

7 CONCLUSIONS

In this work, we created a dataset of 3,322 cross-platform pairs of popular apps from the Apple App Store and their counterparts in the Google Play Store. We provide the analysis pipeline we developed for the app discovery, retrieval, and matching to the community to enable future studies and comparisons of apps between the iOS and Android ecosystems.

We evaluated existing as well as new approaches for cross-platform app matching based on seven features from app store metadata and the apps themselves. We evaluated the quality of our matches against reference pairs we obtained from Google's data migration service. We determined that we can correctly match 95.82 % apps using features that allow for fast large-scale comparisons.

The challenges that we identified during our study include that even within the 10,000 most popular apps for iOS and Android we found a lower number of apps to have a counterpart on the respective other platform (34 %) than expected. On the other hand, we found developers that are publishing a large number of apps, especially games and gambling apps, that look and are named similarly for the same platform, which makes it challenging even for a human to determine the correct match. Furthermore, differences in apps' localization in app stores pose challenges to automated matching using app descriptions. Finally, there is a lack of a sufficiently large ground truth, but the reference pairs from Google's data migration can act as a proxy – with the caveat that it is undocumented how Google determined them.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments and input for improving the paper. We also would like to thank the students whose theses served as the foundation of our analysis pipeline: Thomas Jirout, Leon Weiß, Anna Hartig, Marcel Pfefferle, Julian Knorr, and Mehmet Kurt. Additional thanks to David Schmidt for his help in obtaining the full list of reference pairs and Simon Hayden for engineering support.

This work is based on research supported by the Vienna Science and Technology Fund (WWTF) and the City of Vienna [Grant ID: 10.47379/ICT22060 and Grant ID: 10.47379/ICT19056], the Austrian Science Fund (FWF) [Grant ID: 10.55776/F8515-N] and SBA Research (SBA-K1), a COMET Centre within the framework of COMET – Competence Centers for Excellent Technologies Programme and funded by BMK, BMDW, and the federal state of Vienna. The COMET Programme is managed by FFG. The authors further gratefully acknowledge funding from the German Federal Ministry of Education and Research [Grant ID: 16KIS1629/UbiTrans and Grant ID: 16KISR002/HealthNet].

REFERENCES

[1] Y. Agarwal and M. Hall. "ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing". In: *Proc. of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2013. DOI: 10.1145/2462456.2464460.

[2] M. Alfhaily. *IPATool*. 2023. URL: https://github.com/majd/ipatool.

[3] M. Ali, M. E. Joorabchi, and A. Mesbah. "Same App, Different App Stores: A Comparative Study". In: *Proc. of the IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 2017. DOI: 10.1109/MOBILESoft.2017.3.

[4] M. Ali and A. Mesbah. "Mining and Characterizing Hybrid Apps". In: *Proc. of the International Workshop on App Market Analytics (WAMA)*. 2016. DOI: 10.1145/2993259.2993263.

[5] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. "AndroZoo: Collecting Millions of Android Apps for the Research Community". In: *Proc. of the International Conference on Mining Software Repositories (MSR)*. Available at: https://androzoo.uni.lu. 2016. DOI: 10.1145/2901739.2903508.

[6] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie. "PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play". In: *Proc. of the USENIX Security Symposium*. 2019.

[7] Apple Inc. *App Store Downloads on iTunes*. 2023. URL: https://apps.apple.com/us/genre/ios/id36.

[8] Apple Inc. *Update on apps distributed in the European Union: Alternative distribution on iOS in the EU*. 2024. URL: https://developer.apple.com/support/dma-and-apps-in-the-eu/#ios-app-eu.

[9] Aurora OSS. *GPlayAPI*. 2023. URL: https://gitlab.com/AuroraOSS/gplayapi.

[10] B. Bonné, S. T. Peddinti, I. Bilogrevic, and N. Taft. "Exploring Decision Making with Android's Runtime Permission Dialogs using In-context Surveys". In: *Proc. of the USENIX Symposium on Usable Privacy and Security (SOUPS)*. 2017.

[11] J. Buchner. *ImageHash*. 2023. URL: https://github.com/JohannesBuchner/imagehash.

[12] P. Calciati, K. Kuznetsov, X. Bai, and A. Gorla. "What did Really Change with the new Release of the App?" In: *Proc. of the International Conference on Mining Software Repositories (MSR)*. 2018. DOI: 10.1145/3196398.3196449.

[13] K. Chen, X. Wang, Y. Chen, P. Wang, Y. Lee, X. Wang, B. Ma, A. Wang, Y. Zhang, and W. Zou. "Following Devil's Footprints: Cross-Platform Analysis of Potentially Harmful Libraries on Android and iOS". In: *Proc. of the IEEE Symposium on Security and Privacy (S&P)*. 2016. DOI: 10.1109/SP.2016.29.

[14] L. Chen, C. Shi, Y. Ma, and Z. Shao. "Research on iOS Application of Internal Privacy Leakage Detection Technology". In: *Proc. of the International Conference on Intelligent Computation Technology and Automation (ICICTA)*. 2018. DOI: 10.1109/ICICTA.2018.00046.

[15] Q. Chen, C. Chen, S. Hassan, Z. Xing, X. Xia, and A. E. Hassan. "How Should I Improve the UI of My App? A Study of User Reviews of Popular Apps in the Google Play". In: *ACM Transactions on Software Engineering and Methodology* 30.3 (2021). DOI: 10.1145/3447808.

[16] D. Domínguez-Álvarez and A. Gorla. "Release Practices for iOS and Android Apps". In: *Proc. of the International Workshop on App Market Analytics (WAMA)*. 2019. DOI: 10.1145/3340496.3342762.

[17] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. "PiOS: Detecting Privacy Leaks in iOS Applications". In: *Proc. of the Annual Network and Distributed System Security Symposium (NDSS)*. 2011.

[18] S. Farooqi, Á. Feal, T. Lauinger, D. McCoy, Z. Shafiq, and N. Vallina-Rodriguez. "Understanding Incentivized Mobile App Installs on Google Play Store". In: *Proc. of the ACM Internet Measurement Conference (IMC)*. 2020. DOI: 10.1145/3419394.3423662.

[19] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. "Why People Hate Your App: Making Sense of User Feedback in a Mobile App Store". In: *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2013. DOI: 10.1145/2487575.2488202.

[20] A. Geitgey. *Everything you need to know about implementing iOS and Android Mobile Deep Linking*. 2016. URL: https://medium.com/@ageitgey/everything-you-need-to-know-about-implementing-ios-and-android-mobile-deep-linking-f4348b265b49.

[21] Google LLC. *Data Restore Tool*. 2023. URL: https://play.google.com/store/apps/details?id=com.google.android.apps.restore.

[22] M. Gruber, C. Höfig, M. Golla, T. Urban, and M. Große-Kampmann. "'We may share the number of diaper changes': A Privacy and Security Analysis of Mobile Child Care Applications". In: *Proc. on Privacy Enhancing Technologies (PETS)* 2022.3 (2022). DOI: 10.56553/popets-2022-0078.

[23] K. Hageman, Á. Feal, J. Gamba, A. Girish, J. Bleier, M. Lindorfer, J. Tapiador, and N. Vallina-Rodriguez. "Mixed Signals: Analyzing Software Attribution Challenges in the Android Ecosystem". In: *IEEE Transactions on Software Engineering* 49.4 (2023). DOI: 10.1109/TSE.2023.3236582.

[24] C. Han, I. Reyes, A. Elazari Bar On, J. Reardon, Á. Feal, K. A. Bamberger, S. Egelman, and N. Vallina-Rodriguez. "Do You Get What You Pay For? Comparing the Privacy Behaviors of Free vs. Paid Apps". In: *Proc. of the Workshop on Technology and Consumer Protection (ConPro)*. 2019.

[25] J. Han, Q. Yan, D. Gao, J. Zhou, and R. Deng. "Comparing Mobile Privacy Protection through Cross-Platform Applications". In: *Proc. of the Annual Network and Distributed System Security Symposium (NDSS)*. 2013.

[26] H. Hosseini, M. Degeling, C. Utz, and T. Hupperich. "Unifying Privacy Policy Detection". In: *Proc. on Privacy Enhancing Technologies (PETS)* 2021.4 (2021). DOI: 10.2478/popets-2021-0081.

[27] H. Hu, Y. Tian, S. Hassan, and D. Lin. "Analyzing Gamer Complaints in Reviews of Cross-Platform Video Games on Steam". In: *Proc. of the IEEE Conference on Games (CoG)*. 2023. DOI: 10.1109/CoG57401.2023.10333139.

[28] H. Hu, C.-P. Bezemer, and A. E. Hassan. "Studying the Consistency of Star Ratings and the Complaints in 1 & 2-Star User Reviews for Top Free Cross-Platform Android and iOS Apps". In: *Empirical Software Engineering* 23 (2018). DOI: 10.1007/s10664-018-9604-y.

[29] H. Hu, S. Wang, C.-P. Bezemer, and A. E. Hassan. "Studying the Consistency of Star Ratings and Reviews of Popular Free Hybrid Android and iOS Apps". In: *Empirical Software Engineering* 24 (2019). DOI: 10.1007/s10664-018-9617-6.

[30] Huawei Internet Services. *Phone Clone*. 2023. URL: https://play.google.com/store/apps/details?id=com.hicloud.android.clone.

[31] S. Koch, B. Altpeter, and M. Johns. "The OK Is Not Enough: A Large Scale Study of Consent Dialogs in Smartphone Applications". In: *Proc. of the USENIX Security Symposium*. 2023.

[32] K. Kollnig, A. Shuba, R. Binns, M. Van Kleek, and N. Shadbolt. "Are iPhones Really Better for Privacy? A Comparative Study of iOS and Android Apps". In: *Proc. on Privacy Enhancing Technologies (PETS)* 2022.2 (2022). DOI: 10.2478/popets-2022-0033.

[33] M. Lindorfer, S. Volanis, A. Sisto, M. Neugschwandtner, E. Athanasopoulos, F. Maggi, C. Platzer, S. Zanero, and S. Ioannidis. "AndRadar: Fast Discovery of Android Applications in Alternative Markets". In: *Proc. of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 2014. DOI: 10.1007/978-3-319-08509-8_4.

[34] W. Liu, G. Zhang, J. Chen, Y. Zou, and W. Ding. "A Measurement-based Study on Application Popularity in Android and iOS App Stores". In: *Proc. of the Workshop on Mobile Big Data (Mobidata)*. 2015. DOI: 10.1145/2757384.2757392.

[35] Mitmproxy Project. *mitmproxy – an interactive HTTPS proxy*. 2022. URL: https://mitmproxy.org/.

[36] P. Nema, P. Anthonysamy, N. Taft, and S. T. Peddinti. "Analyzing User Perspectives on Mobile App Privacy at Scale". In: *Proc. of the International Conference on Software Engineering (ICSE)*. 2022. DOI: 10.1145/3510003.3510079.

[37] 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: *Proc. of the USENIX Security Symposium*. 2021.

[38] T. T. Nguyen, M. Backes, and B. Stock. "Freely Given Consent? Studying Consent Notice of Third-Party Tracking and Its Violations of GDPR in Android Apps". In: *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2022. DOI: 10.1145/3548606.3560564.

[39] NLTK Project. *Natural Language Toolkit (NLTK)*. 2023. URL: https://github.com/nltk/nltk.

[40] M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes. "The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators". In: *Proc. of the IEEE Symposium on Security and Privacy (S&P)*. 2018. DOI: 10.1109/SP.2018.00005.

[41] D. Orikogbo, M. Büchler, and M. Egele. "CRiOS: Toward Large-Scale iOS Application Analysis". In: *Proc. of the Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*. 2016. DOI: 10.1145/2994459.2994473.

[42] F. Paci, J. Pizzoli, and N. Zannone. "A Comprehensive Study on Third-Party User Tracking in Mobile Applications". In: *Proc. of the International Conference on Availability, Reliability and Security (ARES)*. 2023. DOI: 10.1145/3600160.3605079.

[43] E. Pan, J. Ren, M. Lindorfer, C. Wilson, and D. Choffnes. "Panoptispy: Characterizing Audio and Video Exfiltration from Android Applications". In: *Proc. on Privacy Enhancing Technologies (PETS)* 2018.4 (2018). DOI: 10.1515/popets-2018-0030.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011). Available at: https://scikit-learn.org.

[45] T. Petsas, A. Papadogiannakis, M. Polychronakis, E. P. Markatos, and T. Karagiannis. "Rise of the Planet of the Apps: A Systematic Study of the Mobile App Ecosystem". In: *Proc. of the ACM Internet Measurement Conference (IMC)*. 2013. DOI: 10.1145/2504730.2504749.

[46] M. Pfefferle. *Apple App Store Crawler*. 2021. URL: https://gitlab.com/marzzzello/appstore_crawler.

[47] M. Pfefferle. *gplaycrawler*. 2021. URL: https://gitlab.com/marzzzello/gplaycrawler.

[48] A. Pradeep, M. T. Paracha, P. Bhowmick, A. Davanian, A. Razaghpanah, T. Chung, M. Lindorfer, N. Vallina-Rodriguez, D. Levin, and D. Choffnes. "A Comparative Analysis of Certificate Pinning in Android & iOS". In: *Proc. of the ACM Internet Measurement Conference (IMC)*. 2022. DOI: 10.1145/3517745.3561439.

[49] M. Rahman, N. Hernandez, R. Recabarren, S. I. Ahmed, and B. Carbunar. "The Art and Craft of Fraudulent App Promotion in Google Play". In: *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2019. DOI: 10.1145/3319535.3345658.

[50] J. Rajasegaran, N. Karunanayake, A. Gunathillake, S. Seneviratne, and G. Jourjon. "A Multi-Modal Neural Embeddings Approach for Detecting Mobile Counterfeit Apps". In: *Proc. of the World Wide Web Conference (WWW)*. 2019. DOI: 10.1145/3308558.3313427.

[51] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. Choffnes, and N. Vallina-Rodriguez. "Bug Fixes, Improvements, ... and Privacy Leaks - A Longitudinal Study of PII Leaks Across Android App Versions". In: *Proc. of the Network and Distributed System Security Symposium (NDSS)*. 2018. DOI: 10.14722/ndss.2018.23143.

[52] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. "ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic". In: *Proc. of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2016. DOI: 10.1145/2906388.2906392.

[53] D. Rodriguez, A. Jain, J. M. D. Alamo, and N. Sadeh. "Comparing Privacy Label Disclosures of Apps Published in both the App Store and Google Play Stores". In: *Proc. of the International Workshop on Privacy Engineering (IWPE)*. 2023. DOI: 10.1109/EuroSPW59978.2023.00022.

[54] P. Salza, F. Palomba, D. Di Nucci, C. D'Uva, A. De Lucia, and F. Ferrucci. "Do Developers Update Third-Party Libraries in Mobile Apps?" In: *Proc. of the IEEE/ACM International Conference on Program Comprehension (ICPC)*. 2018. DOI: 10.1145/3196321.3196341.

[55] P. Salza, F. Palomba, D. Di Nucci, A. De Lucia, and F. Ferrucci. "Third-Party Libraries in Mobile Apps: When, How, and Why Developers Update Them". In: *Empirical Software Engineering* 25.3 (2020). DOI: 10.1007/s10664-019-09754-1.

[56] Samsung Electronics Co., Ltd. *Samsung Smart Switch Mobile*. 2023. URL: https://play.google.com/store/apps/details?id=com.sec.android.easyMover.

[57] S. Sebastian and J. Caballero. "Towards Attribution in Mobile Markets: Identifying Developer Account Polymorphism". In: *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2020. DOI: 10.1145/3372297.3417281.

[58] S. Seneviratne, H. Kolamunna, and A. Seneviratne. "A Measurement Study of Tracking in Paid Mobile Applications". In: *Proc. of the ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. 2015. DOI: https://doi.org/10.1145/2766498.2766523.

[59] W. Squires and P. Centonze. "Cross-Platform Access-Rights Analysis of Mobile Applications". In: *Proc. of the International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 2016. DOI: 10.1145/2897073.2897717.

[60] StatCounter. *Mobile Operating System Market Share Worldwide, Oct 2022 – Oct 2023*. 2023. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[61] M. Steinböck. *Android vs. iOS: Security of Mobile Deep Links*. Master Thesis, TU Wien, 2022. DOI: 10.34726/hss.2022.93327.

[62] C. Tang, S. Chen, L. Fan, L. Xu, Y. Liu, Z. Tang, and L. Dou. "A Large-Scale Empirical Study on Industrial Fake Apps". In: *Proc. of the International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 2019. DOI: 10.1109/ICSE-SEIP.2019.00028.

[63] University of Luxembourg. *Androzoo: Google Play Metadata*. 2023. URL: https://androzoo.uni.lu/gp-metadata.

[64] N. Viennot, E. Garcia, and J. Nieh. "A Measurement Study of Google Play". In: *Proc. of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. 2014. DOI: 10.1145/2591971.2592003.

[65] H. Wang, H. Li, and Y. Guo. "Understanding the Evolution of Mobile App Ecosystems: A Longitudinal Measurement Study of Google Play". In: *Proc. of the World Wide Web Conference (WWW)*. 2019. DOI: 10.1145/3308558.3313611.

[66] H. Wang, Z. Liu, Y. Guo, X. Chen, M. Zhang, G. Xu, and J. Hong. "An Explorative Study of the Mobile App Ecosystem from App Developers' Perspective". In: *Proc. of the International Conference on World Wide Web (WWW)*. 2017. DOI: 10.1145/3038912.3052712.

[67] H. Wang, Z. Liu, J. Liang, N. Vallina-Rodriguez, Y. Guo, L. Li, J. Tapiador, J. Cao, and G. Xu. "Beyond Google Play: A Large-scale Comparative Study of Chinese Android App Markets". In: *Proc. of the ACM Internet Measurement Conference (IMC)*. 2018. DOI: 10.1145/3278532.3278558.

[68] J. Wu. *Magisk*. 2023. URL: https://github.com/topjohnwu/Magisk.

[69] Z. Xie, S. Zhu, Q. Li, and W. Wang. "You Can Promote, but You Can't Hide: Large-Scale Abused App Detection in Mobile App Stores". In: *Proc. of the Annual Conference on Computer Security Applications (ACSAC)*. 2016. DOI: 10.1145/2991079.2991099.

[70] N. Zhong and F. Michahelles. "Google Play Is Not A Long Tail Market: An Empirical Analysis of App Adoption on the Google Play App Market". In: *Proc. of the Annual ACM Symposium on Applied Computing (SAC)*. 2013. DOI: 10.1145/2480362.2480460.

[71] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. Sadeh, S. M. Bellovin, and J. Reidenberg. "Automated Analysis of Privacy Requirements for Mobile Apps". In: *Proc. of the Network and Distributed System Security Symposium (NDSS)*. 2017. DOI: 10.14722/ndss.2017.23034.