IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 9 February 2024, accepted 26 February 2024, date of publication 4 March 2024, date of current version 14 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3373202

APPLIED RESEARCH

On Identification of Intrusive Applications: A Step Toward Heuristics-Based Adaptive Security Policy

FADI MOHSEN[1], USMAN RAUF[2], VICTOR LAVRIC[1], ALEXANDER KOKUSHKIN[1], ZHIYUAN WEI[3], AND ADALYNN MARTINEZ[2]
[1]Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, 9712 CP Groningen, The Netherlands
[2]Department of Mathematics and Computer Science, Mercy University, Dobbs Ferry, NY 10522, USA
[3]Rocky Mountain Robotech, Broomfield, CO 80020, USA

Corresponding author: Fadi Mohsen (f.f.m.mohsen@rug.nl)

This work was supported in part by the National Centers of Academic Excellence in Cybersecurity, which is part of the National Security Agency (NSA) under Grant H98230-22-1-0256.

ABSTRACT Android is widely recognized as one of the leading mobile operating systems globally. As the popularity and usage of Android OS and third-party application stores continue to soar, the process of developing and publishing applications has become increasingly accessible. However, the absence of a robust filtering mechanism to ensure that applications only request appropriate and secure permissions poses a significant concern. While extensive research has been conducted on malware analysis, the realm of intrusive applications remains largely unexplored. The lack of defensive measures to promptly identify invasive applications tilts the balance in favor of malicious actors and developers who may embed intrusive behavior within their products. It is imperative to develop new monitoring tools and techniques that address these privacy gaps. In light of this, we propose a ***Continuous Threat Monitoring Framework*** (CTMF) designed to safeguard mobile users from intrusive apps both before and after installation. Our framework, implemented and evaluated in the Android environment, offers practical deployability without imposing excessive overhead. It fills the void by considering the changes occurring within an app while it remains on a user's device, setting it apart from existing anti-intrusiveness solutions primarily focusing on app installation.

INDEX TERMS Insider threats, intrusive applications, android applications, mobile security, threat analysis.

I. INTRODUCTION

Android stands out as one of the most prevalent and dependable mobile operating systems, with a staggering 3.3 billion global users in 2023, securing a substantial 71.8% share of the mobile operating systems market [1]. This pervasive adoption can be attributed to the platform's modularity and accessibility, with free development tools like Android Studio facilitating widespread application creation [2]. As of March 2023, the Google Play Store hosted an extensive array of 2.67 million apps, underscoring the integral role mobile applications play in our daily lives [3].

However, amidst this digital landscape, an alarming trend emerges—nearly 49% of U.S. consumers, as per McAfee's report, neglect to employ mobile security software, rendering them susceptible to sophisticated cyber threats and data breaches [4]. The Android permission system stands as the primary defense against these perils, intending to alert users to permission requests before granting them. However, its effectiveness has been questioned, especially due to users' inattention and misunderstanding of the system's prompts. This has been highlighted by prior studies [5], [6], [7].

To address these concerns, researchers have proposed supplementary solutions, such as improving the usability and effectiveness of the Android permissions system, particularly against intrusive third-party applications [8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. These endeavors primarily

The associate editor coordinating the review of this manuscript and approving it for publication was S. K. Hafizul Islam.

aim to assist users in selecting less invasive applications or replacing installed apps with more secure alternatives. However, a critical observation reveals that many of these solutions are limited in capturing the evolving permission requests across different app versions.

Our investigation into existing works revealed a common limitation—they rely on a single snapshot of the declared configurations such as permissions and system actions [8], [9], [10], [11], [12], [13], [14], necessitate changes to the underlying operating systems [15], [16], or primarily target developers rather than end users [17]. Thus, we believe that these solutions might fall short of capturing the dynamic evolution of permission requests over various app versions.

In response, our research introduces the *Continuous Threat Monitoring Framework* for Android applications, leveraging a cutting-edge scoring mechanism proposed in [10]. Unlike existing approaches, our framework adopts a live scoring approach, evaluating an app's score based on its declared permissions as well as its actual usage over its lifespan on the device. We employ an altered growth rate formula to scrutinize changes in permissions usage across different app versions [18]. Our work makes four novel contributions: (i) adopting a scoring technique demonstrated to outperform counterparts in recent literature [14], (ii) introducing the concept of live scoring, (iii) utilizing an altered growth rate formula to detect anomaly usage of permissions, and (iv) providing a detailed implementation addressing intricate technical challenges.

This paper is structured as follows: Section II delves into related works, Section III discusses the technical background of our framework, Section IV provides insights into the implementation specifics, and finally, Section VII concludes our research. We would like to emphasize that the source code and data used in this research will be made available upon the paper's acceptance.

II. RELATED WORKS

This section summarizes the related works on risk assessment techniques and server-side-independent solutions.

Before Android 6.0, numerous works in this research area existed, but we will not delve into them due to significant changes introduced in Android after that version. From Android 6.0 onward, several studies were conducted to devise risk assessment techniques. Yan Hu et al. employed the `HITS` page ranking algorithm to construct a static call graph of Android applications, aiding security analysts in identifying vulnerabilities and sensitive methods in malware [8]. While their primary focus is on malware, their approach offers valuable insights. They encountered a similar challenge to ours in extracting permissions-related information from Google Play, opting for a third-party solution instead of developing their own. Another work targeting malware was Khariwal et al., who developed IPDroid, which takes another approach to malware detection, employing a combination of permissions and intents to identify the most effective mix for detecting malware [19]. Chih-Chang et al. proposed a framework for estimating privacy risk scores of mobile apps, acknowledging the potential privacy concerns associated with their data collection practices [9]. The authors devised an automated privacy risk assessment, focusing on data access permissions and privacy policies. The work leverages the UT CID ITAP dataset, which comprises identity assets, their vulnerabilities, and associated risk values. Identity assets are collected from apps through privacy policies and Android manifest XML files. We believe relying solely on the information declared in the privacy policy and manifest files may not accurately represent the app's actual behavior. User behavior, such as denying permissions or not reaching specific functionalities, can impact the real-world privacy risks associated with an app.

The scoring technique introduced by Mohsen et al. in their paper [10] holds relevance to our work, as our framework is built upon their ideas and concepts. Particularly the notions of using user preferences, broadcast receivers, and permissions to calculate the security score of an app. In their study, user preferences are characterized as a rating score the end user assigns to a permission group. Moreover, Mohsen et al. incorporate the prevalence of broadcast receivers' actions in privacy score computations, representing a noteworthy enhancement over other approaches such as [11] and [12].

However, it's important to note that their approach relies solely on the information declared in the manifest file. As mentioned earlier, some of these declared permissions may never be activated or used. Additionally, their work lacks discussion on the technical details of obtaining this information directly from a mobile phone.

On a similar conceptual basis as our approach, Rashidi et al. developed an advisory app that consistently monitors requested permissions, offers recommendations, and ranks applications [13]. Nevertheless, unlike our application, which depends on a pre-computed seed dataset, their app targets highly qualified users.

Several server-side solutions have been introduced, such as the *Privacy-Palisade* app by Quattrone et al. [15]. This app identifies outliers, specifically apps employing uncommon permissions, using the Isolation Forest technique. The solution requires changes to the Android OS, specifically modifications to the Android Launcher. Our approach draws the intrusiveness score from an app's permissions, actions, and user ratings. Importantly, extracting the app's data mandates no modifications to the underlying operating system. João Marono et al. presented an additional solution [16], which comprises an Android app, a privacy quantification module, and a server. In this solution, the server is responsible for downloading and processing the source code of the specified app, along with its configuration file. Subsequently, the obtained results are forwarded to the quantification module for score calculation, and the calculated score is then transmitted back to the app. Notably, this approach has limitations, as it does not address paid applications, analyzes apps in isolation, without comparing them to similar apps or considering their descriptions, and overlooks the different states of an application after its installation.

Rahman et al.'s research [17] similarly focuses on analyzing the source code, utilizing static code metrics such as lines of code and bad coding practices. SonarQube was employed for metric extraction, and risk scores were generated using the Androrisk tool. The primary objective of this study is to assist developers in identifying privacy and security risks during the early stages of application development. Notably, the application of this approach to help users avoid intrusive or vulnerable applications is constrained by SonarQube's requirement for access to the application's source code. While the research on intrusiveness or insider threat detection is extensive, we cannot cover all the details within the confines of this article. Therefore, we recommend referring to a comprehensive and recently published literature review on insider/intrusiveness detection for further information and in-depth analysis [20].

In summary, our work brings forth novel contributions: (i) we adopt a cutting-edge scoring technique, demonstrated to outperform its counterparts in recent literature [14]. (ii) We introduce the concept of live scoring. This approach assesses an app's score based on declared permissions and actual usage over its lifespan on the device. (iii) We employ an altered growth rate formula to scrutinize changes in permissions usage across different app versions. (iv), we provide a detailed implementation, addressing intricate technical challenges.

III. TECHNICAL BACKGROUND, REQUIREMENTS & PROPOSED FEATURES

In this section, we overview the basic concepts and quantitative metrics for calculating the intrusiveness of an application. These concepts are fundamental to our work and define conditions/notions on which the following proof of concept has been developed. We also briefly explain a few technical terms we use throughout the forthcoming sections.

A. TERMS AND DEFINITIONS

Below are the terms and definitions that we will be referring to throughout the paper. To ensure a comprehensive understanding, each term is accompanied by an explanation of its significance and context within the framework:

- **Broadcast Receivers Actions; Receivers Actions; Actions**: These terms are used interchangeably to refer to the set of actions assigned to '<intent-filters>' of broadcast receivers. This set of actions dictates how the app responds to incoming broadcast messages, influencing its behavior and functionality.
- **Permissions**: This term refers to the complete set of permission strings requested by an Android application. Permissions dictate the level of access an app has to various system resources and user data.
- **Receiver Score; Receiver's Privacy Score**: The privacy score of an app is calculated based on the actions of its broadcast receivers. This score reflects the app's potential privacy implications in terms of its interaction with broadcast messages.
- **Permissions Score; Permissions Privacy Score**: This score represents the privacy implications of an app based on the permissions it requests. It evaluates the potential risks associated with granting the app access to certain system resources and user data.
- **Seed Data Set**: This is the reference data set of Android applications used as a basis for evaluating and scoring new apps. It serves as a benchmark for comparison and analysis.
- **Suggestion(s) Set**: This subset of the seed data set is specifically curated to improve app suggestions within a particular genre. It helps refine recommendations based on the characteristics and preferences of the target audience.
- **IRP (Individual Receiver/Permission Prevalence)**: IRP measures the prevalence of a specific broadcast receiver action / Permission within the sample set of Android applications. It provides insights into the frequency of usage of a particular action across different apps.
- **AORP (App Overall Receiver/Permission Prevalence)**: AORP aggregates the individual receiver prevalence scores to calculate the overall prevalence of broadcast receiver actions / Permissions for an app. This metric offers a comprehensive view of an app's engagement with broadcast messages and its potential privacy implications.

B. THREAT EVALUATION METRIC

To ascertain whether an application qualifies as intrusive, it undergoes evaluation based on specific mathematical criteria. One such metric, the App Overall Receiver/Permission Prevalence (*AORP*), proposed in the literature [10], serves to gauge abnormality and rank applications. *AORP* assesses abnormality by separately computing scores for permissions ($AORP_p$) and receivers ($AORP_r$), subsequently combining them to determine the final privacy score of an application. *AORP* is defined as follows:

$$\boxed{AORP_{final} = \frac{AORP_r + AORP_p}{2}}$$

Whereas, $AORP_r$ and $AORP_p$ can be defined as follows:

$$\boxed{AORP_r = \frac{-1}{([c \cdot \sum_{IRP \in App_i} \lg(IRP)] - 1)}}$$

$$\boxed{AORP_p = \frac{-1}{([c \cdot \sum_{IRP \in App_i} \lg(IRP \cdot g(p))] - 1)}}$$

Both formulas include a constant `c` and the `IRP` value. The `IRP` quantifies the relative impact or usage of permissions or broadcast receivers within the application data's sample set.

The formula for calculating *IRP* is as follows:

$$\boxed{IRP = \frac{x}{total}}$$

In this context, x denotes the count of applications of a particular genre utilizing a particular broadcast receiver action or permission, while *total* signifies the total number of applications in that genre. We intend not to alter the x and *total* counters when evaluating new applications but rather to update them solely upon changes to the seed dataset. Therefore, we slightly adjust the existing *IRP* formula to include the currently evaluated application as follows:

$$\boxed{IRP = \frac{x+1}{total+1}}$$

To dissect the results of $AORP_r$, we can examine the smaller components of the formula. The value of IRP always remains less than 1, implying that $\lg(IRP)$ is always less than or equal to 0. Consequently, the following term is consistently negative:

$$c \cdot \sum_{IRP \in App_i} \lg(IRP) \leq 0 \implies [c \cdot \sum_{IRP \in App_i} \lg(IRP) \leq 0] - 1 \leq -1 \implies AORP_p \leq 1 \quad (1)$$

Therefore, we observe that a score of 1.0 represents the highest possible score, indicating minimal intrusiveness of the application in terms of broadcast receivers or permissions use. A lower score closer to 0 signifies greater intrusiveness. A similar analysis can be applied to the permission score ($AORP_p$).

Where $g(p)$ represents the user preference for a specific permission group, such as setting the `PHONE` permission group with a security level of 2, resulting in $g(p) = \frac{1}{2}$. Analogously, we can demonstrate that the permissions' score $AORP_p \in (0, 1]$, leading to $AORP_{final} \in (0, 1]$.

C. FUNCTIONAL REQUIREMENTS

In this section, we delve into the core functional requirements essential for the successful development of our framework. These requirements serve as the foundation upon which our implementation details are built, guiding our efforts toward the realization of a robust and versatile solution.

1) RECOMMENDATIONS AND EVALUATION

The recommendation will consist of the highest-scored app from the suggestions set, ensuring it belongs to the same genre as the evaluated application. We will also provide the recommendation score and a Google Play link. This link will allow the user to access the dedicated Play Market page and consider our suggestion as a potential alternative to the installed program being evaluated.

2) INSTALLED/UPDATED APPLICATIONS EVALUATION AND REMOVED APPLICATIONS DETECTION

To accurately detect recently installed, updated, or removed applications, we plan to maintain a set of tuples formed during evaluations. These tuples will include installed apps and their respective last update times, structured as follows:

$$apps = \{(pkgName, time) \mid pkgName \in (['a', \ldots 'z'] \cup ['.'])^* \ \wedge \ time \in (0, +\infty)\}$$

where *packageName* denotes the package name of an installed application and *time* represents the last time the application was updated. Let's consider *installed* as a map that is structurally identical to *apps*, containing the currently queried application from the device. Therefore, we can define the detection procedure for all three types of applications as follows:

$$updatedApps = \{\, y \mid x \in apps,\ y \in installed, x.packageName = y.packageName, y.time > x.time \,\} \cup \{\, y \mid \ y \in installed, \forall x \in apps \implies x.packageName \not\models y.packageName)\}$$

We aim to make several evaluation modes available at the user's discretion to provide flexible and rigorous usability for our application:

- **All Applications Evaluation**: The system does not activate this mode; rather, it can only be initiated by the end user.
- **Background Evaluation**: The evaluation mode begins on login and is user-independent. It can only be terminated by force-stopping the application, and it evaluates newly installed/updated apps while removing evaluation information for deleted ones.
- **Re-evaluation**: To re-evaluate a specific app in the list, select it and request re-evaluation; only the selected app will be evaluated

We rely on the Application Ranking for all evaluation scores, but we also provide two additional practical scores that adjust the formulas outlined in [10]:

- **The Granted Permission Score (GPS)** is utilized during evaluation and focuses solely on the granted permissions (GP). It's important to note that while an app may request certain permissions, users need to grant them for the app to use them explicitly. GPS can be defined as follows:

$$\boxed{AORP_{GPS} = \frac{-1}{[c \cdot \sum_{x,\ total\ \ IRP} \lg(IRP \cdot g(p))] - 1}}$$

where *total* $\in GP$ represents the total number of granted permissions, and $IRP \in App_i$ represents the Individual Receiver Prevalence for the evaluated app.
- **Granted Final Score (GFS)** is computed as the average of the Receiver Score ($AORP_r$) and the Granted

Permission Score ($AORP_{GPS}$):

$$\boxed{AORP_{GFS} = \frac{AORP_r + AORP_{GPS}}{2}}$$

3) EVALUATION SUMMARY AND DETAILED VIEW

Our solution enables end-users to request reviews of their evaluations, which requires maintaining a comprehensive record of all app evaluations conducted throughout the device's application lifecycle. Additionally, users should have access to detailed information about a specific evaluation, including the app's details, all assigned scores, and its Anomaly Level.

4) IN THE MARKET EVALUATION

Users may wish to evaluate an app for malicious or intrusive behavior before installing it. The *Evaluate In-the-Market* feature facilitates this process by providing the app's Google Play URL or package name. This allows users to compare the privacy scores of different apps and choose the least intrusive option. It's important to note that evaluations are based solely on the permissions listed on the Google Play store pages, which may not correspond directly to specific permission groups or names. To address this limitation, we developed a routine called `PlayStoreInterpreter`, which maps Play Store permissions to their corresponding manifest permission groups. However, it's important to acknowledge that this mapping relies on certain assumptions, which may result in different approximations. In this specific case, we rely on information provided by Google LLC [21].

5) ANOMALY DETECTION

Updates play a critical role in the life cycle of an app, often introducing new features, improving existing functionality, and incorporating new broadcast receivers and/or permission declarations. These updates can significantly impact the app's privacy ranking and serve as a key indicator when evaluating its intrusiveness trend. We utilize a modified growth rate formula adapted from [18] to address this. The alteration accounts for the fact that the growth rate of intrusiveness corresponds to a lower final score - meaning, the more intrusive the application, the lower the final score. The growth rate (GR) is defined as:

$$\boxed{GR = -\frac{x_i - x_{i-1}}{x_{i-1}}, \text{ where } x_i \in \mathbb{R},\ 0 < x_i \leq 1,\ \forall i \in \mathbb{N}}$$

Here, GR represents the growth rate of an application, and x_i represents the i^{th} evaluation.

However, we need to calculate the growth rate value over a period, not just between the two most recent evaluations. Therefore, we adapt the mean growth rate (MGR) formula from [22]:

$$\boxed{MGR = \frac{\sum_{i=1}^{n-1} -(\frac{x_i - x_{i-1}}{x_{i-1}})}{n-1},\ \forall n \in \mathbb{N},\ x_i \in \mathbb{R},\ 0 < x_i \leq 1}$$

The numerator represents the sum of all consecutive growth rates, and we divide by $n - 1$ to account for one less growth rate.

If the MGR approaches 1.0 (100%), it indicates that, on average, the application's current privacy score is definitely lower than half of the previous score, as follows for all consecutive evaluations:

$$\boxed{current\_score < 0.5 \times previous\_score}$$

Therefore, our anomaly detection subroutine must alert the user in case of such misbehavior. We define three status indicators:

- $0 \leq$ *mean growth rate* ≤ 0.5: Assigned green status, indicating acceptable changes.
- $0.5 <$ *mean growth rate* < 1: Assigned orange status, indicating considerable intrusiveness rate detection.
- *mean growth rate* ≥ 1: Assigned red status, indicating an ultimate threat to end-user privacy.

6) BACKGROUND ASSESSMENT

The assessment of installed, updated, or removed applications needs to occur both in the background and foreground, with a primary focus on accurate background activity. We conduct application assessments only upon installation or update to ensure reliability and efficiency. An essential requirement is that the background process should initiate automatically upon device boot completion without the need for the user to restart the app. Furthermore, the app should remain operational even when closed, meaning it should not be force-stopped.

7) NEW DATA SET PREPARATION

To ensure the ongoing effectiveness of our solution, we need to update or replace the current seed dataset and all extracted information periodically. However, this presents challenges as we do not alter the values in the IRP calculation or the suggestion set during new evaluations. Acquiring new seed data involves resource-intensive and time-consuming tasks such as scraping, downloading, and parsing applications from Google Play. To simplify this process, we've incorporated the collection of the new seed dataset into the evaluation process. The new dataset must be processed and incorporated into future updates.

IV. PROPOSED FRAMEWORK

In this section, we will explore the building blocks of our proposed framework, encompassing the features discussed in earlier sections. The framework is designed to facilitate continuous threat monitoring for Android applications. Certain related features may be grouped, and the implementation of specific features may span multiple subsections. Our framework revolves around the *Privacy Watcher* app, and thus, we will provide a more detailed examination of its implementation compared to other components.

A. TECHNOLOGY STACK

We developed the *Privacy Watcher* app primarily using Java and utilized Firebase Real-time Database [23] for database-related operations. Additionally, Firebase Authentication was used to handle user sign-up and sign-in procedures, both on the client and server side [23]. Finally, we employed Python's Django Framework to develop the server-side API as an alternative to extracting permissions/actions.

B. APPLICATION FRAMEWORK OVERVIEW

Figure 1 provides a high-level overview of our framework. Next, we will delve into a more detailed examination of some of its key components.

1) XML EXTRACTOR

The XML Extractor is a web application, entirely written in Python, using Django, and hosted on Heroku [24]. The application decompiles the `AndroidManifest.xml` on POST request and returns a JSON object containing the application's permissions and broadcast receivers' actions. Please note that this service is intended as a backup if the alternative local option, Section V-D, stops working due to Android's changes.

2) FIREBASE

Firebase is a platform developed and maintained by Google that provides solutions for easy integration and development [23]. We utilize two services of Firebase: Authentication, which represents the email/password standard authentication scheme for our application, and Realtime Database - a tree-structured database [25], easily stored and accessible, provided that the user has a network connection.

3) PLAY STORE

We leverage the Google Play store to retrieve the genres of the installed applications because it is required to calculate the privacy scores. Additionally, we retrieve the permission list of uninstalled apps for the *Evaluate In-the-Market* feature.

4) AAPKS.COM

aapks.com is a website that shares free Android applications [26]. We use it to locate the APK file of a given application, which is needed for the *Download and Evaluate* feature.

5) PRIVACY WATCHER APP

The *Privacy Watcher* App serves as the central component of our solution. It is an Android-based application that leverages various services including the Play Store, Firebase, aapks, and XML remote service. Its primary function is to empower users to monitor their installed apps and make informed decisions about which apps to install from the market.

The entire Android application and its external library flow heavily rely on callbacks and listeners. This mandates using a *design pattern* to handle all the changes resulting from executing these callbacks and listeners. Therefore, we select to implement the *Observer Design Pattern*. We divide the whole application structure into three types of actors.

- **Only listeners**: Classes that execute an update upon a triggered event.
- **Only updaters**: Classes that only trigger an event.
- **Both listeners and updaters**: Parts of the application that have both roles in executing a triggered update and notifying other listeners about a change.

In the next section, we will discuss the implementation details of our proof of concept.

V. IMPLEMENTATION DETAILS

This section provides a comprehensive overview of the implementation details for key components of our framework, such as the database structure and login procedures, different flavors of evaluations, anomaly detection, and ensuring continuous background monitoring of applications.

A. DATABASE STRUCTURE AND AUTHORIZATION RULES

To ensure optimal performance and user experience, we organize our seed data set into two main parts:

- **Summary Set:** This encompasses data collected from the entire provided set, categorized by genre. The summary set is further divided into two categories:
  - `Counters`: This contains information about the actions and permissions present within apps from a specific genre.
  - `Sizes`: This segment stores the number of applications sorted by genre.
- **Suggestions Set:** This set is formed from a filtered subset of data, ordered and filtered by the scored AORP rank.

We calculate the permission and receiver scores for all the applications in the suggestion set. We then sort the applications in this set based on the mean value as follows:

$$\boxed{AORP_{final} = \frac{AORP_r + AORP_p}{2}}$$

We select a minimal subset of each genre's top 100 scored applications. To ensure persistence, we utilize the Firebase Real-time Database API for remotely storing all application-related data [23]. The database adopts a tree structure rather than conventional tables and relations, and we design structured, minimalistic sub-trees for each application part.

Authorization is implemented using Firebase Realtime Database's integrated functionality, restricting user access based on Authentication UID issued by the Firebase Authenticate module [23]. We define Database Rules to control read and write permissions, applying them per sub-tree [23].

B. REGISTRATION/LOGIN

We employed Firebase Authentication functionality [23] to ensure robust implementation of the Authentication service. This service safeguards user credentials and is integral to the operations of two components in our system: the `XML`

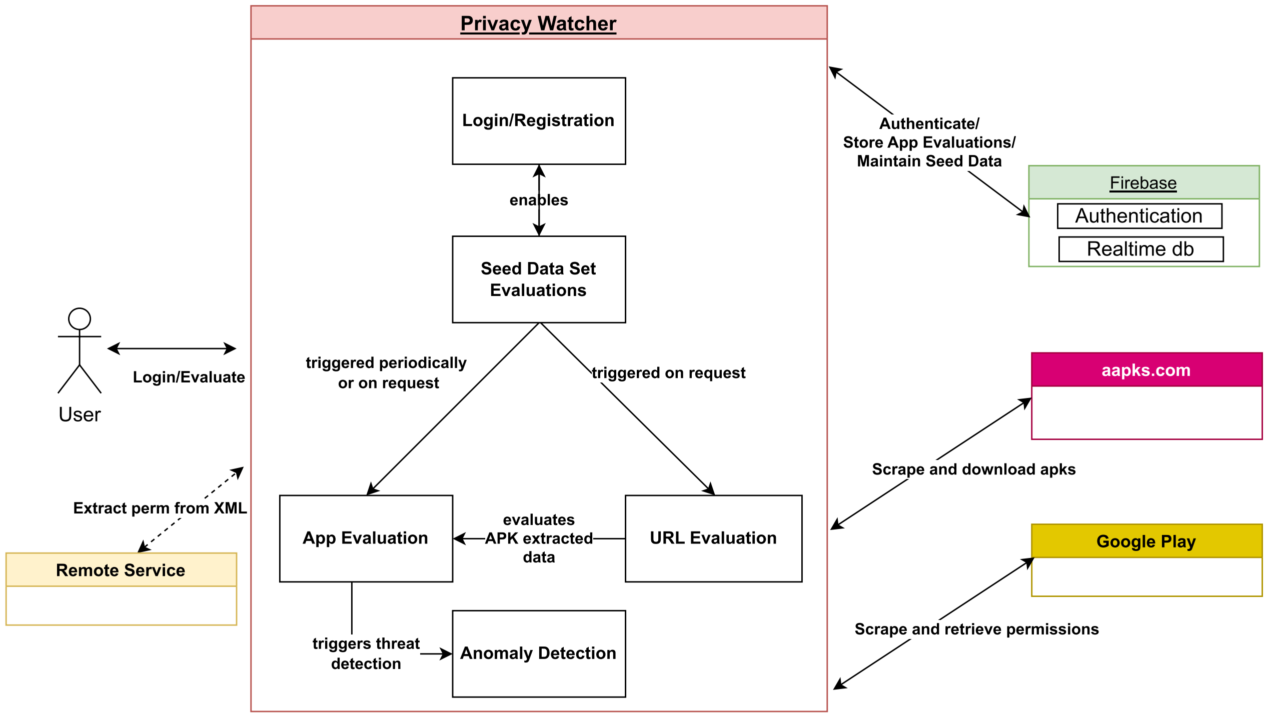


FIGURE 1. A high-level overview of the Continuous Threat Monitoring Framework (CTMF). The Remote Service is kept as a backup option if the local extraction of the permissions and system actions is impossible.

`Extractor`, which permits access to the extraction of the actions/permissions stack only to authorized users, and the `Privacy Watcher`, which relies on the Authentication UID for authorized interactions with the Firebase Realtime Database [23]. For the sign-in/up process, we seamlessly integrated the standard email/password authentication scheme using the Firebase API [23]. We establish persistent authentication by employing `Encrypted Shared Preferences` to securely store the user's email and password, mitigating the need for frequent login requests. Upon deregistration, the user's details and evaluation stack are downgraded, retaining only the necessary data for generating the new seed data set. The Authentication process is streamlined using the `Singleton Design Pattern`, facilitating convenient access from various application components. Users are prompted to specify their permission group preferences during registration, following the methodology outlined in [10]. Upon completion of this process, the permissions score and final score for the suggestion data set are computed, and the relevant data is then retrieved to the Firebase Realtime Database.

C. SUGGESTIONS AND THEIR EVALUATIONS

We follow these steps to implement recommendations: first, we traverse the permission stack and identify the app's permission group. Then, we calculate the score using modified formulas from [10] (as detailed in Section III-B). Next, we compute the mean value of the pre-computed receiver score and the permission rank. The evaluated suggestion is then displayed in a dedicated window unless the evaluated application belongs to an unknown genre or significantly outperforms our suggested programs. This display includes the application package name, its score, and a Google Play link.

D. APP EVALUATION AND REMOVAL DETECTION

We've implemented two methods to retrieve intent-filter actions and permissions since using Android *Package Manager* and the *GET_INTENT_FILTERS* flag was impossible due to their deprecation in Android 11 and 12. The first method, as detailed in Section IV-B1, involves parsing *AndroidManifest.xml* on a remote web service using *androguard*. The parsed actions and permissions are then sent to the *Privacy Watcher* App. The second method entails extracting and parsing *AndroidManifest.xml* on the device for both installed and downloaded Android packages as follows:

1) The `Privacy Watcher` app starts by searching for the package file in the public source directory (`/data/app/Androidpackage.apk`). If not found in the first directory, it assumes it is in the second directory.(`/storage/emulated/0/Download/Androidpackage.apk`). The app then stores the located APK using the following statement: `ZipFile apk = new ZipFile(filePath)`, where `filePath` is one of the directories mentioned in this step.
2) The `XMLExtractor` Java class extracts the `AndroidManifest.xml` file from the APK by

TABLE 1. Sample outputs from the first two steps we performed to decode a manifest file.

Step	Sample Result
1	`D/XMLExtractor: Encoded: AwAIAJQjAAABABwALBIAAFYAAAAAAAAAAAAAAAHQBAAAAAAAAAAAAAAA4A AAAcAAAAKAAAADQAAABM (...) ABgAAAACAAAA//////////9LAAAAAQEQABgAAAACAAAA/////yEAAA`
2	`D/XMLExtractor: Decoded & parsed: Vt4LdxVpfhfz4D0x0n0DZnthemelabel (...) android. intent.action.BOOT_COMPLETED (...) android.permission.INTERNET (...) o3qFIQQIGGIq FkJexPIII55IdIxPII66ILLI77IL2KI`

executing the following statement:
`ZipEntry m= apk.getEntry`
`(`AdroidMnfest.xml')`
All `AndroidManifest.xml` files in .apk directories have the same name.

With the manifest file extracted, the next step is to retrieve the permission/action stack using a parser to decompile the compressed manifest file on the device. The decompilation process involves the following steps:

1) To retrieve the permission/action stack from the compressed manifest file, the `Privacy Watcher` app must first extract the file's contents. This is done by calling `apk.getInputStream(manifest).`
2) Once the stream's contents have been retrieved, they are converted into an array of byte code using: `byte[] code=IOUtils.toByteArray(stream)`
3) If the array of byte code is base64 encoded,[1] it can be converted into a string using: `str=new String(Base64.encode(code,Base64. DEFAULT)`
The resulting encoded string should look similar to the sample output shown in Step 1 of Table 1.
4) Next, we base64 decode the encoded string and remove all non-alphanumeric characters using the `replaceAll()` method with a regular expression `"[^A-Za-z-0-9_.-]"` .[2] This results in a string without spaces. To identify the permission and intent actions, we add a space before every occurrence of `android.permission.` and `android.intent.action.` in the string. After applying these filters, the string contains all the necessary information from `AndroidManifest.xml` for evaluation, as shown in Step 2 of Table 1.
5) Finally, with spaces introduced to separate each permission and intent action, the `XMLExtractor` creates stacks for both permissions and intent actions and returns it to the `AppEvaluation` class, where the permission and broadcast receivers can be evaluated.

[1]A method of encoding binary data into ASCII characters to ensure safe transmission over text-based communication protocols.

[2]A sequence of characters that forms a search pattern, used for pattern matching within strings.

To ensure the accuracy of our application, it's essential to retrieve information about all installed applications. We use the `getInstalledApplications` method and the `QUERY_ALL_PACKAGES` permission. To compute the practical scores, i.e., `Granted Permission Score`, we use the `checkPermission` function and check if the return result is equal to:
`PackageManager.PERMISSION\_GRANTED`
The `Granted Final Score` is calculated by taking the mean value of the Granted Permission Score and Receiver Score [27].

E. DOWNLOAD AND EVALUATE

Initially, we planned to download the APK file from the Google Play Store directly. However, we encountered a challenge due to the lack of an official API provided by Google for this purpose. Alternatively, we turned to third-party websites like *aapks.com* [26]. The *Download and Evaluate* approach involves three main steps:

(i) Locating the download link involves sending a request to the website's search engine and parsing the returned results (`https://aapks.com/?s=<packageName>`)

(ii) Downloading the APK: Once the download link is obtained, we proceed to download the APK file. During this process, we register a broadcast receiver for the `ACTION_DOWNLOAD_COMPLETE` event and prompt the user to grant the `WRITE_EXTERNAL_STORAGE` and `READ_EXTERNAL_STORAGE` permissions.

(iii) Alerting the user: After the download is complete, we notify the user and inform them again when the evaluation is ready.

However, this approach has limitations:

- Inconsistency in application versions: There is no guarantee that the downloaded application will match the exact version intended by the user.

- Uncertainty in retrieving desired applications: Factors such as the app not being listed or requiring payment can hinder retrieving the desired application's download link.

- Safety concerns with aapks.com: While we use the relatively safe parsing tool JSoup [28] to protect users from most malicious software, aapks.com has been flagged as a phishing and malicious website by several sources.

TABLE 2. The permission matching table used to map the permissions found in Google Play to their corresponding manifest permissions.

Google Play Permission	Manifest Permissions
Messages	RECEIVE_MMS, READ_SMS, RECEIVE_WAP_PUSH, SEND_SMS
Files and docs	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE
Contacts	READ_CONTACTS, WRITE_CONTACTS
Voice or sound recordings	RECORD_AUDIO
Photos and videos	CAMERA, READ_EXTERNAL_STORAGE
Account management	GET_ACCOUNTS, ACCOUNT_MANAGER
Precise location	ACCESS_FINE_LOCATION
Approximate location	ACCESS_COARSE_LOCATION
Web browsing history	INTERNET, READ_HISTORY_BOOKMARKS
Calendar	WRITE_CALENDAR, READ_CALENDAR

The extracted APKs may contain errors or modifications, although they are not inherently dangerous until installed. Nonetheless, users should approach the URL evaluation results with caution.

F. EVALUATE IN-THE-MARKET

The `Evaluate In-the-Market` feature provides an alternative to the `Download & Evaluate` feature, eliminating the need to download the APK file. This feature aims to quickly approximate the privacy permission score by utilizing the permission data on the Google Play store. The `Privacy Watcher` app retrieves the app's genre(s) and permissions through pattern matching. It's important to note that these retrieved permissions may not correspond to those in the manifest file. We define a procedure based on certain assumptions to establish a matching between the two permission sets, which may result in different approximations. These assumptions are derived from the information provided by Google LLC [21]. We show our permission matching table in Table 2. Once the matching is established, the evaluation process is similar to that of installed applications. With the `Privacy Watcher` app, users can compare the approximate scores of two or more apps. This feature allows users to compare between apps based on their privacy scores before installing any of them; see Appendix VII-A.

G. CONTINUOUS EVALUATION AND NOTIFICATION

The primary function of the Privacy Watcher app is to evaluate installed apps for any malicious behavior continuously. This evaluation is based on the formula outlined in Section III-C5. The results are conveyed to the user through three colors and shapes: Greenheart, Orange exclamation mark, and Red hand (refer to Figure 5). Additionally, a notification system alerts users if an app exceeds the red status threshold of 1.0 to alleviate the need for manual checks.

H. ADAPTING TO IMPLICIT BROADCAST RECEIVER LIMITATIONS

In earlier Android versions, implicit broadcast receivers could monitor installation, update, and uninstall events. However, since Android 8.0, Google has restricted the use of such receivers. To address this limitation and accommodate unexpected changes, we adopted Job Scheduling, as recommended in [29], to schedule new jobs every 3-5 minutes. Although we experimented with different scheduling intervals, we found that shorter intervals were too frequent and disrupted jobs.

Our objective is to ensure that our application automatically starts after each successful device reboot without requiring manual intervention from the user. To accomplish this, we utilize the `BOOT_COMPLETED` intent, which is one of the exempted implicit broadcast receiver actions, and register it with the associated `RECEIVE_BOOT_COMPLETED` action [27].

VI. EVALUATION OF CTMF

In Sections I and II, we explained the uniqueness of our work compared to the literature. Consequently, our evaluation must consider that. As such, we evaluated our framework on multiple levels, including compatibility, performance, and accuracy.

A. COMPATIBILITY

In Section IV, we explained our implementation choices to accommodate for compatibility. Our app has undergone rigorous testing by running it on both real and emulated smartphones. While our main focus was on Android 11, we also ensured compatibility with Android 12 by testing the framework's performance on this version as well. Below is the device that we used the most during the testing:

- Samsung Galaxy Note S20 Ultra, running Android 12 on One UI 4.1 (baseband version N986BXXU4F VE7 with Android security patch level: 1 June 2022), Qualcomm Snapdragon 865 Octa-core, 12 GB RAM/128 GB of internal and 128GB of SD storage.

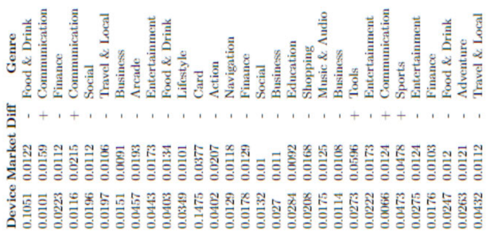
Furthermore, our inclusion of the XML Extractor, Section IV-A as a backup aims to accommodate any changes that might arise in the future.

B. PERFORMANCE

The acceptability of the *Privacy Watcher* app by the end users relies on many factors, one of which is performance. To evaluate the execution time, we conducted tests on specific features of the *Privacy Watcher* app, namely (i) Evaluate

TABLE 3. The time (in seconds) it takes to complete each listed task.

Task	#apps	Median	Mean	SD
XML extraction	169	0.004	0.0454	0.1163
Genre extraction	169	0.079	0.211	0.2783
Evaluate Installed	169	0.01	0.2387	0.3304
Download&Evaluate	5	96.911	107.5841	126.1841
Evaluate In-the-Market	5	0.529	0.5632	0.6483



Device	Market	Diff	Genre
0.1051	0.0122	-	Food & Drink
0.0101	0.0159	+	Communication
0.0223	0.0112	-	Finance
0.0116	0.0215	+	Communication
0.0196	0.0112	-	Social
0.0197	0.0106	-	Travel & Local
0.0151	0.0091	-	Business
0.0457	0.0193	-	Arcade
0.0443	0.0173	-	Entertainment
0.0403	0.0134	-	Food & Drink
0.0349	0.0101	-	Lifestyle
0.1475	0.0377	-	Card
0.0402	0.0207	-	Action
0.0129	0.0118	-	Navigation
0.0178	0.0129	-	Finance
0.0132	0.01	-	Social
0.027	0.011	-	Business
0.0284	0.0092	-	Education
0.0208	0.0168	-	Shopping
0.0175	0.0125	-	Music & Audio
0.0114	0.0108	-	Business
0.0273	0.0596	+	Tools
0.0222	0.0173	-	Entertainment
0.0066	0.0124	+	Communication
0.0473	0.0478	+	Sports
0.0275	0.0124	-	Entertainment
0.0176	0.0103	-	Finance
0.0247	0.012	-	Food & Drink
0.0263	0.0121	-	Adventure
0.0432	0.0112	-	Travel & Local

FIGURE 2. The comparison of permission scores obtained through on-device evaluation and market evaluation for 30 applications belonging to popular genres. The Diff column indicates the difference between the two scores, with a '−' indicating that the market score is smaller than the device score, and a '+' indicating the opposite.

Installed Apps, (ii) XML extraction, (iii) Genre extraction, (iv) Download & Evaluate, and (v) Evaluate In-the-Market. For operations 1-3, we used a dataset of 169 applications, while for operations 4 and 5, we considered popular apps such as Booking.com, Reddit, Telegram, Spotify, and Viber. The tests were conducted on the Samsung device, refer to Section VI-A. To make a fair comparison, we will use the maximum values obtained during each category's testing. Our evaluation criteria include CPU usage, memory usage, network consumption, and execution time. We used the `Android Profiler` tool to collect all execution data concerning CPU, Memory, and Network usage.

On average, the *Privacy Watcher* app has the following overhead upon evaluating 169 installed applications: 30.4% CPU usage, 229.7 MB memory usage, and 0.94 and 0.14 MB/s incoming and outgoing network usage, respectively. It is worth noting that this overhead occurs only when the *Privacy Watcher* app is run for the first time or when there is a change in the list of applications. Table 3 shows the results of our runtime analysis, indicating that the solution does not impose significant execution or resource overhead, making it reasonably practical to use.

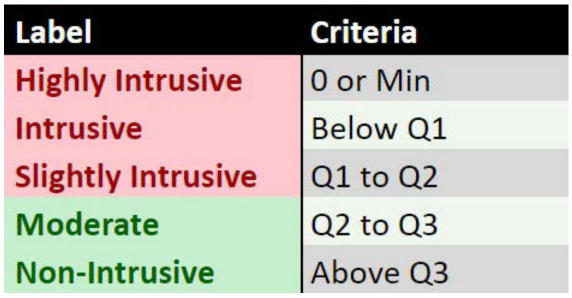
C. EFFICIENCY

We aim to evaluate the efficiency of our framework by primarily testing its scoring mechanism. While previous studies have demonstrated its superiority over comparable methods [14], we opted to further validate these findings through experimentation with real datasets. Our efficiency testing comprises two main aspects: approximation accuracy and detection accuracy.

1) APPROXIMATION ACCURACY OF CTMF

In this section, our focus is on evaluating the *Evaluate In-the-Market* approach, which relies solely on permissions retrieved from the app store and mapped to corresponding Manifest permissions (see Section III-C4). To assess its accuracy, we evaluated a diverse set of 30 applications spanning various genres such as communication, finance, social, and card games. Table 2 depicts the results of our analysis. While the permission scores obtained through the *Evaluate In-the-Market* approach generally tend to be slightly lower compared to those acquired through the *Download and Evaluate* method (which provides ground truth scores), they still offer valuable insights for users. Despite the minor differences, the Market Evaluation scores provide a reliable method for users to compare applications before installation, facilitating informed decision-making. Additionally, users who prioritize utmost accuracy can utilize the Download and Evaluate feature of the *Privacy Watcher* app.

TABLE 4. Five-Point analytics based statistical threshold.



Label	Criteria
Highly Intrusive	0 or Min
Intrusive	Below Q1
Slightly Intrusive	Q1 to Q2
Moderate	Q2 to Q3
Non-Intrusive	Above Q3

2) DETECTION ACCURACY

This evaluation aims to assess our chosen scoring mechanism's ability to identify malicious applications and compare it with existing literature accurately. We utilized two datasets containing labeled applications for this purpose. Initially, we computed intrusiveness scores for all applications in both datasets using our approach and the counterparts in recent literature [14]. Subsequently, each application was assigned an intrusiveness label based on these scores. To establish thresholds for label prediction, we employed five-point statistical analytics, dividing the data into quartiles represented by specific percentile ranges. Table 4 illustrates these ranges and their corresponding labels, with red-highlighted rows indicating intrusive apps and green ones denoting safe or non-intrusive ones. This evaluation methodology was chosen due to the literature's inability to assess intrusiveness across multiple app updates, prompting us to adopt a threshold-based labeling approach using single instances of apps to ensure consistency in evaluation criteria. It's essential to note that despite the classification criteria, the underlying metrics remain consistent.

a: EVALUATION USING LARGE DATASET

The first dataset comprised manifest and store data of 870,515 Android mobile applications, including their status, sourced from [30]. The status denotes whether an application was removed from the market during the collection period, serving as our ground truth. Of the 485,738 apps removed by the

TABLE 5. Evaluation with known intrusive applications.

		Literature	Research
Package Name	Genre	Literature Prediction	Our Prediction
de.nineergysh.quickarttwo	Photography	Moderate	Slightly Intrusive
gb.painnt.moonlightingnine	Photography	Moderate	Slightly Intrusive
gb.twentynine.redaktoridea	Photography	Moderate	Slightly Intrusive
de.photoground.twentysixshot	Photography	Moderate	Slightly Intrusive
de.xnano.photoexifeditornine	Photography	Moderate	Slightly Intrusive
de.hitopgop.sixtyeightgx	Photography	Moderate	Slightly Intrusive
de.sixtyonecollice.cameraroll	Photography	Moderate	Slightly Intrusive
de.instgang.fiftyggfife	Photography	Moderate	Slightly Intrusive
de.fiftyninecamera.rollredactor	Photography	Moderate	Slightly Intrusive
com.neonthemekeyboard.app	Personalization	Moderate	Intrusive
com.androidneonkeyboard.app	Personalization	Moderate	Intrusive
com.cachecleanereasytool.app	Tools	Moderate	Intrusive
com.fancyanimatedbattery.app	Personalization	Moderate	Intrusive
com.fastcleanercashecleaner.app	Tools	Moderate	Intrusive
com.funnycallercustomtheme.app	Personalization	Moderate	Intrusive
com.callercallwallpaper.app	Personalization	Moderate	Intrusive
com.mycallcallpersonalization.app	Personalization	Moderate	Intrusive
com.funnywallpapaerslive.app	Personalization	Moderate	Intrusive
com.newscrean4dwallpapers.app	Personalization	Moderate	Intrusive
hungerfourk.thinkmobileprostudio.newcamera	Photography	Slightly Intrusive	Slightly Intrusive
com.helphomestickers.heartcarejingchat	Personalization	Moderate	Intrusive
com.waxwell.saunders.pistaphotoeditor	Photography	Moderate	Non-Intrusive
com.marinabradley.pocolauncher	Entertainment	Moderate	Slightly Intrusive
dailyyoga.yogago.loseweight.yoga	Health & Fitness	Slightly Intrusive	Intrusive
com.cliffbradley.youtooncartooneffect	Photography	Slightly Intrusive	Slightly Intrusive

Google Store, our approach successfully labeled 378,480 as intrusive, achieving an accuracy of 77.919%. In comparison, the literature's scoring metric identified only 339,729 apps as intrusive, resulting in an accuracy of 69.941%. These findings underscore the superiority and heightened sensitivity of our scoring mechanism in detecting apps prone to future removal due to their inherent intrusiveness.

b: EVALUATION USING KNOWN INTRUSIVE APPS

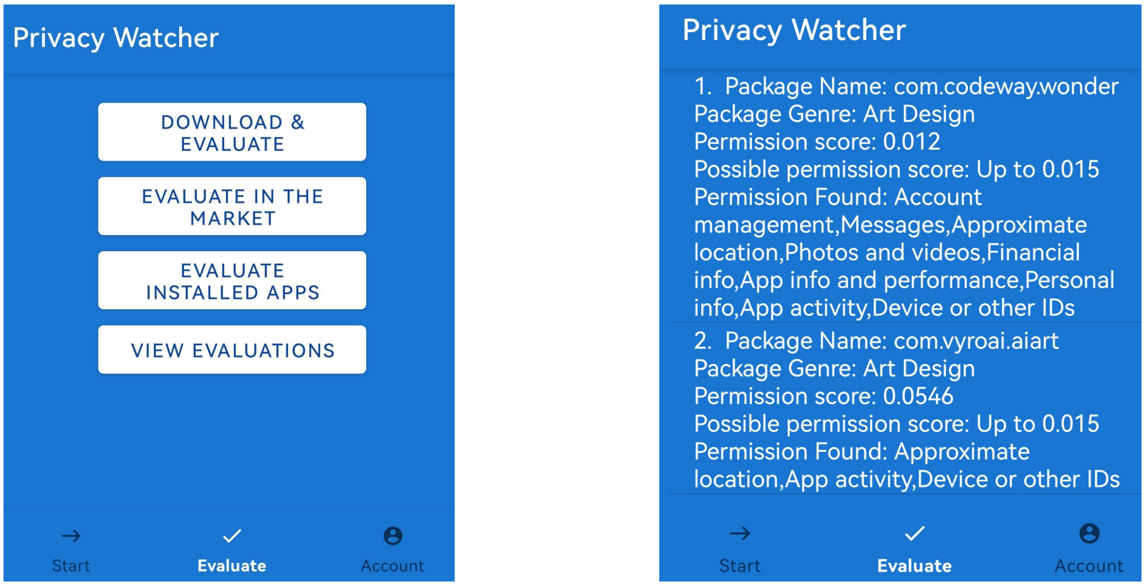
While our approach has demonstrated superiority in accurately identifying apps removed from the Google Store using a literature dataset, this phase of evaluation focuses on directly scrutinizing known intrusive applications recently reported by media and news agencies [31], [32]. To achieve this, we extracted permission and broadcast receiver data from these notorious intrusive apps (refer to Table 5) and used the aforementioned five-point criteria for threshold-based classification to predict labels of these known intrusive apps.

Table 5 reveals that our accuracy significantly outperforms the literature in this aspect of the evaluation as well. We accurately detected 24 out of 25 famously known intrusive apps, whereas, using the literature metric, only 3 out of 25 were correctly identified. These results clearly illustrate the practicality and efficacy of our approach in identifying intrusive apps.

D. ADDRESSING THREAT ACTOR UTILIZATION OF PREVALENT PERMISSIONS

Concern: Threat actors may exploit the prevalence of specific permissions in legitimate apps to evade detection by privacy assessment mechanisms.

Response: We believe that our approach is unsusceptible to this threat because it introduces user preferences ($g(p)$) associated with permissions. This ensures that a rogue app will not necessarily pass privacy assessment solely based on the prevalence of permission. The abnormality score considers both permission prevalence and user preferences, providing a more nuanced evaluation. Additionally, our method offers the option to combine scores from permissions and broadcast receivers. By averaging these scores, our approach delivers a comprehensive assessment that factors in abnormality based on receivers and the influence of user preferences on granted permissions. Finally, we acknowledge


Privacy Watcher

DOWNLOAD & EVALUATE

EVALUATE IN THE MARKET

EVALUATE INSTALLED APPS

VIEW EVALUATIONS

Start Evaluate Account

Privacy Watcher

1. Package Name: com.codeway.wonder
Package Genre: Art Design
Permission score: 0.012
Possible permission score: Up to 0.015
Permission Found: Account management,Messages,Approximate location,Photos and videos,Financial info,App info and performance,Personal info,App activity,Device or other IDs

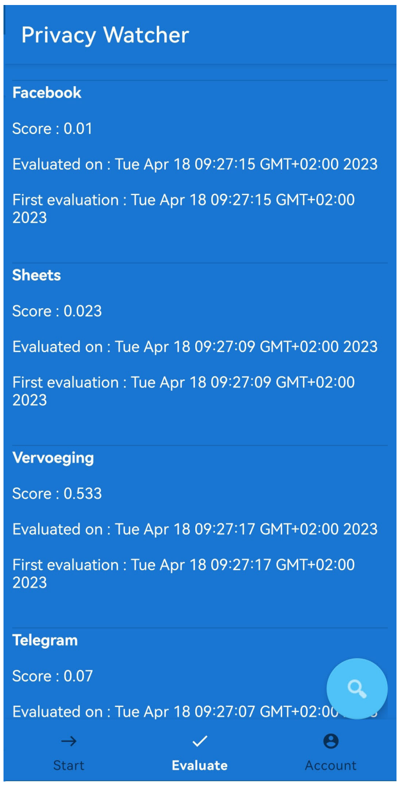
2. Package Name: com.vyroai.aiart
Package Genre: Art Design
Permission score: 0.0546
Possible permission score: Up to 0.015
Permission Found: Approximate location,App activity,Device or other IDs

Start Evaluate Account

(a) The main menu of the Privacy Watcher App.

(b) The results of comparing two AI art generator apps before installing them.

FIGURE 3. Screenshot examples of the privacy watcher app.


Privacy Watcher

Facebook

Score : 0.01

Evaluated on : Tue Apr 18 09:27:15 GMT+02:00 2023

First evaluation : Tue Apr 18 09:27:15 GMT+02:00 2023

Sheets

Score : 0.023

Evaluated on : Tue Apr 18 09:27:09 GMT+02:00 2023

First evaluation : Tue Apr 18 09:27:09 GMT+02:00 2023

Vervoeging

Score : 0.533

Evaluated on : Tue Apr 18 09:27:17 GMT+02:00 2023

First evaluation : Tue Apr 18 09:27:17 GMT+02:00 2023

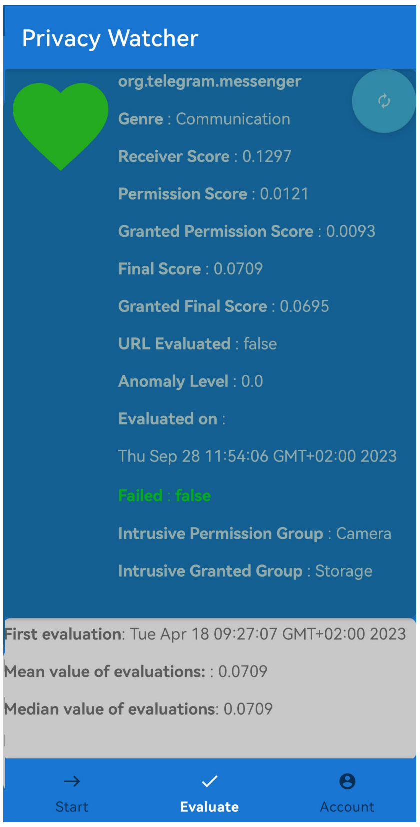
Telegram

Score : 0.07

Evaluated on : Tue Apr 18 09:27:07 GMT+02:00

Start Evaluate Account

FIGURE 4. The results page of all evaluated apps.


Privacy Watcher

org.telegram.messenger

Genre : Communication

Receiver Score : 0.1297

Permission Score : 0.0121

Granted Permission Score : 0.0093

Final Score : 0.0709

Granted Final Score : 0.0695

URL Evaluated : false

Anomaly Level : 0.0

Evaluated on :

Thu Sep 28 11:54:06 GMT+02:00 2023

Failed : false

Intrusive Permission Group : Camera

Intrusive Granted Group : Storage

First evaluation: Tue Apr 18 09:27:07 GMT+02:00 2023

Mean value of evaluations: : 0.0709

Median value of evaluations: 0.0709

Start Evaluate Account

FIGURE 5. The evaluation results of an installed app.

the significant effort required by attackers to identify popular permissions across a large number of apps within a specific category. The resource-intensive and time-consuming nature of this process acts as a deterrent, increasing the difficulty for potential attackers to exploit prevalent permissions effectively.

VII. CONCLUSION

In this study, we have developed a framework for continuously monitoring Android applications, overcoming restrictions on certain Android APIs, and enabling evaluation of newly installed or updated apps. Our framework also allows users to evaluate applications on Google Play before installation and detect anomalous apps using the growth rate metric. We emphasize the importance of developing privacy-scoring apps and urge the Android development team to enhance tools for analyzing the `AndroidManifest.xml`. Additionally, we highlight the need for a dedicated Google Play API to extract application information effectively without resorting to web scraping. Our work contributes to promoting better privacy practices and robust monitoring in the Android ecosystem, driving advancements in application security and privacy evaluation.

APPENDIX

A. SCREENSHOTS

See Figures 3–5.

ACKNOWLEDGMENT

Any findings, conclusions, or recommendations expressed in this research are those of the author(s) and do not necessarily reflect the views of the sponsor.

REFERENCES

[1] D. Ruby. (2023). *20 Android Statistics in 2023 (Market Share & Users)*. [Online]. Available: https://shorturl.at/dyBM9

[2] Google. (2024). *Android Studio*. [Online]. Available: https://developer.android.com/studio

[3] statista. (2009). *Number of Available Applications in the Google Play Store From December 2009 to March 2023*. [Online]. Available: https://www.statista.com/

[4] A. Dolezal. (2021). *As Mobile Usage Skyrockets, Nearly Half of Consumers Do Not Protect Personal Data*. [Online]. Available: https://shorturl.at/kGRTU

[5] K. Benton, L. J. Camp, and V. Garg, "Studying the effectiveness of Android application permissions requests," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, Mar. 2013, pp. 291–296.

[6] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proc. 8th Symp. Usable Privacy Secur.* New York, NY, USA: Association for Computing Machinery, Jul. 2012, pp. 1–14, doi: 10.1145/2335356.2335360.

[7] G. L. Scoccia, S. Ruberto, I. Malavolta, M. Autili, and P. Inverardi, "An investigation into Android run-time permissions from the end users' perspective," in *Proc. IEEE/ACM 5th Int. Conf. Mobile Softw. Eng. Syst. (MOBILESoft)*. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 45–55.

[8] Y. Hu, W. Kong, D. Ding, and J. Yan, "Method-level permission analysis based on static call graph of Android apps," in *Proc. 5th Int. Conf. Dependable Syst. Their Appl. (DSA)*, Sep. 2018, pp. 8–14.

[9] K. Chang, R. Nokhbeh Zaeem, and K. Barber, "A framework for estimating privacy risk scores of mobile apps," in *Proc. Int. Conf. Inf. Secur.*, 2020, pp. 217–233.

[10] F. Mohsen, H. Abdelhaq, H. Bisgin, A. Jolly, and M. Szczepanski, "Countering intrusiveness using new security-centric ranking algorithm built on top of elasticsearch," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./12th IEEE Int. Conf. Big Data Sci. Eng.*, Aug. 2018, pp. 1048–1057.

[11] V. F. Taylor and I. Martinovic, "SecuRank: Starving permission-hungry apps using contextual permission analysis," in *Proc. 6th Workshop Secur. Privacy Smartphones Mobile Devices*. New York, NY, USA: ACM, Oct. 2016, pp. 43–52, doi: 10.1145/2994459.2994474.

[12] H. Quay-de la Vallee, P. Selby, and S. Krishnamurthi, "On a (Per)mission: Building privacy into the app marketplace," in *Proc. 6th Workshop Secur. Privacy Smartphones Mobile Devices*, Oct. 2016, pp. 63–72. [Online]. Available: http://cs.brown.edu/~hannahqd/pubs/PerMissionSPSM16.pdf

[13] B. Rashidi, C. Fung, A. Nguyen, T. Vu, and E. Bertino, "Android user privacy preserving through crowdsourcing," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 3, pp. 773–787, Mar. 2018.

[14] F. Mohsen, H. Abdelhaq, and H. Bisgin, "Security-centric ranking algorithm and two privacy scores to mitigate intrusive apps," *Concurrency Comput., Pract. Exper.*, vol. 34, no. 14, p. e6571, Jun. 2022, doi: 10.1002/cpe.6571.

[15] A. Quattrone, L. Kulik, E. Tanin, K. Ramamohanarao, and T. Gu, "PrivacyPalisade: Evaluating app permissions and building privacy into smartphones," in *Proc. 10th Int. Conf. Inf., Commun. Signal Process. (ICICS)*, Dec. 2015, pp. 1–5.

[16] J. Marono, C. Silva, J. P. Barraca, V. Cunha, and P. Salvador, "Assessing mobile application privacy: A quantitative framework for privacy measurement," 2023, *arXiv:2311.00066*.

[17] A. Rahman, P. Pradhan, A. Partho, and L. Williams, "Predicting Android application security and privacy risk with static code metrics," in *Proc. IEEE/ACM 4th Int. Conf. Mobile Softw. Eng. Syst. (MOBILESoft)*, May 2017, pp. 149–153.

[18] J. Chen and M. James. (2021). *Average Annual Growth Rate (AAGR)*. [Online]. Available: https://www.investopedia.com/terms/g/growthrates.asp

[19] K. Khariwal, J. Singh, and A. Arora, "IPDroid: Android malware detection using intents and permissions," in *Proc. 4th World Conf. Smart Trends Syst., Secur. Sustainability*, Jul. 2020, pp. 197–202.

[20] U. Rauf, F. Mohsen, and Z. Wei, "A taxonomic classification of insider threats: Existing techniques, future directions & recommendations," *J. Cyber Secur. Mobility*, vol. 12, pp. 221–252, May 2023.

[21] Google. (2023). *Play Store Data Safety: Data Types*. [Online]. Available: https://shorturl.at/uxJXZ

[22] A. Hayes and C. Rhinehart. (2021). *Average Annual Growth Rate (AAGR)*. [Online]. Available: https://www.investopedia.com/terms/a/aagr.asp

[23] Google. (2024). *Firebase: Google's Mobile and Web App Development Platform*. [Online]. Available: https://firebase.google.com/

[24] Salesforce. (2023). *Heroku: Cloud Application Platform*. [Online]. Available: https://www.heroku.com/

[25] Google. (2024). *Structure Your Database*. [Online]. Available: https://firebase.google.com/docs/

[26] (2021). *AAPKS*. [Online]. Available: https://aapks.com/

[27] Google. (2024). *Android Developers*. [Online]. Available: https://developer.android.com/

[28] J. Hedley. (2009). *Jsoup: Java Html Parser*. [Online]. Available: https://jsoup.org/

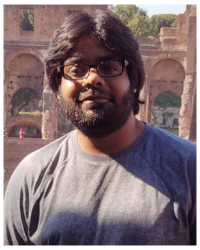
[29] Google. (2023). *Background Execution Limits*. [Online]. Available: https://shorturl.at/pESX3

[30] F. Mohsen, D. Karastoyanova, and G. Azzopardi, "Early detection of violating mobile apps: A data-driven predictive model approach," *Syst. Soft Comput.*, vol. 4, Dec. 2022, Art. no. 200045. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772941922000114

[31] A. Spadafora. (2022). *Malware Hits Millions of Android Users—Delete These Apps Right Now*. [Online]. Available: https://www.tomsguide.com/news/malware-hits-10-million-android-users-delete-these-apps-right-now

[32] M. Humphries. (2022). *36 Malicious Android Apps Found on Google Play, Did You Install Them?* [Online]. Available: https://www.pcmag.com/news/36-malicious-android-apps-found-on-google-play-did-you-install-them

FADI MOHSEN received the B.Sc. degree in computer information systems from the University of Jordan, Jordan, the M.Sc. degree in computer science from the University of Colorado at Colorado Springs, USA, as a recipient of the Fulbright Scholarship, in 2008, and the Ph.D. degree in computing and informatics from the University of North Carolina at Charlotte, USA, in 2016. Currently, he is a tenured Assistant Professor in computing science with the University of Groningen. His primary research interests include usable security, information privacy, mobile and web security, moving target defense, and security analytics.



USMAN RAUF received the B.S. degree in computational physics from the University of the Punjab, Pakistan, in 2008, the M.S. degree in computational sciences and engineering from the Research Center for Modeling and Simulation, National University of Sciences and Technology, Pakistan, in 2011, and the Ph.D. degree in computing and informatics (with a concentration in information security) from the University of North Carolina at Charlotte, USA, on a fully funded Ph.D. Scholarship. Since 2020, he has been an Assistant Professor in cybersecurity with Mercy University, NY, USA. He also holds several research and development awards by several U.S. federal agencies. He was awarded the Scholarship for Service Award for the M.S. degree.

VICTOR LAVRIC received the bachelor's degree from the University of Groningen, in 2021. Throughout his tenure at the university, he honed his expertise in computer science, also serving as a Teaching Assistant for multiple courses. Starting from 2022, he has been actively contributing as a Software Engineer.

ALEXANDER KOKUSHKIN received the bachelor's degree in computer science with a minor in entrepreneurship from the University of Groningen, in 2022. He has been a seasoned Development Engineer with Yokogawa, contributing his expertise in full-time capacity, since October 2022. Prior to this role, he gained valuable experience as a Software Developer Intern with Medusa Radiometrics Ltd., during a five-month internship.

ZHIYUAN WEI received the B.S. degree in computer science and technology from Shanghai Jian Qiao University, Shanghai, China, in 2021, and the M.S. degree in cyber security from Mercy University, NY, USA, in 2023. He is currently a Software Engineer with Broomfield, CO, USA. Since 2022, he has been with the Cyber Resilience Integration with Security & Privacy (CRiSP) Laboratory for Insider threat analytics. His research interests include threat analytics, data science, and machine learning.

ADALYNN MARTINEZ is currently pursuing the M.S. degree in cybersecurity with Mercy University, NY, USA. She is also a Research Assistant with the Cyber Resilience Integration with Security & Privacy (CRiSP) Laboratory. Prior to her graduate studies, she completed the Cybersecurity Program with Westchester Community College, NY, USA, earning a certificate and the Cybersecurity Curriculum Award, in 2023. She is also a Senior Tutor for the WCC Cybersecurity Program, where she supports and mentors fellow students.