Permission-Educator: App for Educating Users about Android Permissions

Akshay Mathur[1], Ethan Ewoldt[2], Quamar Niyaz[2], Ahmad Javaid[1], and Xiaoli Yang[2]

[1] The University of Toledo, Toledo, OH 43607, USA
{akshay.mathur, ahmad.javaid}@utoledo.edu
[2] Purdue University Northwest, Hammond IN 46323, USA
{eewoldt,qniyaz,yangx}@pnw.edu

Abstract. Cyberattacks and malware infestation are issues that surround most operating systems (OS) these days. In smartphones, Android OS is more susceptible to malware infection. Although Android has introduced several mechanisms to avoid cyberattacks, including Google Play Protect, dynamic permissions, and sign-in control notifications, cyberattacks on Android-based phones are prevalent and continuously increasing. Most malware apps use critical permissions to access resources and data to compromise smartphone security. One of the key reasons behind this is the lack of knowledge for the usage of permissions in users. In this paper, we introduce *Permission-Educator*, a cloud-based service to educate users about the permissions associated with the installed apps in an Android-based smartphone. We developed an Android app as a client that allows users to categorize the installed apps on their smartphones as system or store apps. The user can learn about permissions for a specific app and identify the app as benign or malware through the interaction of the client app with the cloud service. We integrated the service with a web server that facilitates users to upload any Android application package file, i.e. *apk*, to extract information regarding the Android app and display it to the user.

Keywords: Smartphone security · Android · Malware Detection · App permissions · Education

1 Introduction

To combat against the rise in malware in operating systems (OS), system developers fix vulnerabilities in their systems through regular updates. With each new version of Android (OS), Google also fixes the vulnerabilities discovered in the previous versions to make it more resilient against cyberattacks and ensure the security and privacy of its users. The first line of defense in Android is Google Play Protect that identifies any malware apps on Google Play Store. However, there are several third-party app stores where malware apps can still be found. Another major contributor to its security is the permission-based resources access system, which prevents apps from gaining unauthorized access to resources such as camera, microphone, and internal file storage. Even with this dual-layer defense mechanism, malware attacks are still prevalent in Android.

In the permission-based system, each permission holds rights to access a specific resource. Therefore, an application needs the user's consent to use certain resources. These permissions are categorized into four levels of protection: *i) Normal, ii) Dangerous, iii) Signature*, and *iv) Signature|Privileged* [1]. Permissions categorized as *Dangerous* are the most sensitive among all of them as they manage users' personal information and they might compromise the security and privacy of users when used with malicious intent. Hence, user approval is requested for these permissions,e.g. `CALL_PHONE` permission that enables an app to make phone calls, or `CAMERA` permission to give an app access to the device camera.

A naive user might be oblivious to whether a permission is requested by a benign or malware app during installation. Malware developers employ several techniques to hide the actual intent of a malware app and deceive users by making it appeared as benign. Malware apps gain access to smartphone's resources through over-claimed permissions – requesting more permissions than required [2], drive-by-download – a malicious download link [3], or permission escalation – using other permissions used by other apps through intents [4]. Malware such as Trojan, ransomware, spyware, and worms take advantage of an uninformed user about Android's permission system and compromise user's data. Therefore, a need arises to educate users about Android permissions.

In this paper, we present a cloud-based service called *Permission-Educator* to help users understand Android permissions and their usage in apps. Permission-Educator provides an Android app as a client using which a user can select an app and see the permissions associated with the installed app on the smartphone. A web interface is developed to allow users to upload Android apps for the same purpose from a desktop system. We have also integrated Permission-Educator with an in-house developed malware detection system, NATICUSdroid [5], which can analyze an installed app's permissions and identify the app as benign or malware. The contributions of this work are as follows:

1. A cloud-based service, Permission-Educator, is developed to educate users about various permissions associated with the installed apps, with the motive to educate naive users about the importance of permissions with respect to data privacy.
2. An Android app and a website are developed as clients of the cloud-based service. The app displays all the installed apps on a user's smartphone, and the web client allows users to upload an Android *apk* file from a browser to view permissions and their purpose and identify whether the app is malware or not.
3. We bring a recently *in-house developed* malware detection system (NATICUSdroid) in production by integrating it with Permission-Educator to identify an app as benign or malware. NATICUSdroid uses the permissions of an app as features and determines whether the app is benign or malicious.

Towards this direction, the paper is structured as follows. Section 2 provides a review of related works. Section 3 and 4 discuss the architecture of Permission-Educator and give an insight on its functionality, respectively. Section 5 concludes the paper along with the possible future works.

2 Related Work

Until Android Kitkat (Android release 4.4), apps were installed using a "take-it or leave-it" approach, where a user could either install an app with its all requested permissions or deny the installation. The app would display information on all necessary dangerous category permissions to the user before the choice is made. From Android Marshmallow (release 6.0) onward, users can either grant permission or deny it while running the app based on their judgment. One could think this choice could prevent malicious intents of apps. However, it has not made a significant change for users [6].

Another change made to facilitate this new method of granting permissions is to create permission groups. For example, `GET_ACCOUNTS`, `READ_CONTACTS`, and `WRITE_CONTACTS`, all are being covered by the app's request to access *Contacts*, i.e, granting all through one. While this provides a choice for granting permissions during run-time over the "take-it or leave-it" approach, the grouping may grant permissions to the app that users may not want, but just because they are in the requested permissions group. As a result of this grouping, earlier, an app needed to request solely `READ_PHONE_STATE` permission to find if a call is in progress or not, could now also gain access to `CALL_PHONE` permission that allows making phone calls without the user's knowledge. Such functionality makes this security module translucent for users as opposed to the prior take-it or leave-it model, which was more transparent regarding individual permissions [7].

While a simple solution to this could be to provide a list of permissions granted in the permission group, it would still not solve the existing problem, i.e., the user's lack of understanding of permissions and their purposes. While the permission models have changed vulnerabilities are still there and user's understanding of what all a permission does, attitude towards different permissions, and apps that request access have not varied considerably. Surveys conducted after Android Marshmallow demonstrate that even if a naive user is constantly 'nagged' by the app to grant a permission the user would give in one day without having a full understanding of them. Such a behavior of the user makes one even more vulnerable to malware infection [7].

Felt et al. surveyed 333 participants and reported that only 17% of individuals paid attention to permissions while installing an application, and only 6.6% (22 of 333) participants completely understood the utility of permissions in general [8]. Ramachandran et al. found in their survey conducted with 478 people that around 63% participants do not review an app's requested permissions diligently. They also found that if a user has downloaded apps from a category before, they hint at what permissions that app can ask. This is why they felt "very comfortable" in downloading and installing a new app from a similar category [9]. This could be an issue as such users are at risk of downloading potential malware.

Several studies have been published to better help non-tech savvy users handle apps more safely. Oglaza et al. and Scoccia et al. attempt to automate the process of permission granting by learning what permissions the user accepts or denies and automatically grant or deny runtime permission pop-ups while installing new apps [10][11]. However, this could become counter-productive if the user is not familiar with permissions in apps and encourages the model to grant all permissions to any app, including malware, automatically. A few researchers attempt to use more mathematical-based models to predict

an app's credibility and, in doing so, assist the user in knowing whether an app is potentially harmful or not before installation. While Hamed et al. in [12] focuses solely on permissions requested compared to the permissions already granted to other apps on a user's phone, Moussa et al. in [13] proposed the ACCUSE model that includes an app's rating and the number of downloads as a "popularity factor." ACCUSE was even found to outperform other malware detection models when tested against 868 malware apps. However, these models and apps do not assist users in understanding why an app is requesting specific permissions and what they are used for.

Although the studies mentioned above attempted to reduce malware app installation or help in handling run-time permission pop-ups, many users still do not comprehend the purpose of permissions. This lack of knowledge can inhibit the previous efforts from providing sufficient protection. Consequently, *Permission-Educator* focuses on educating users about permissions and potential malware for installed apps on user's smartphones to make better decisions in granting permissions to apps.

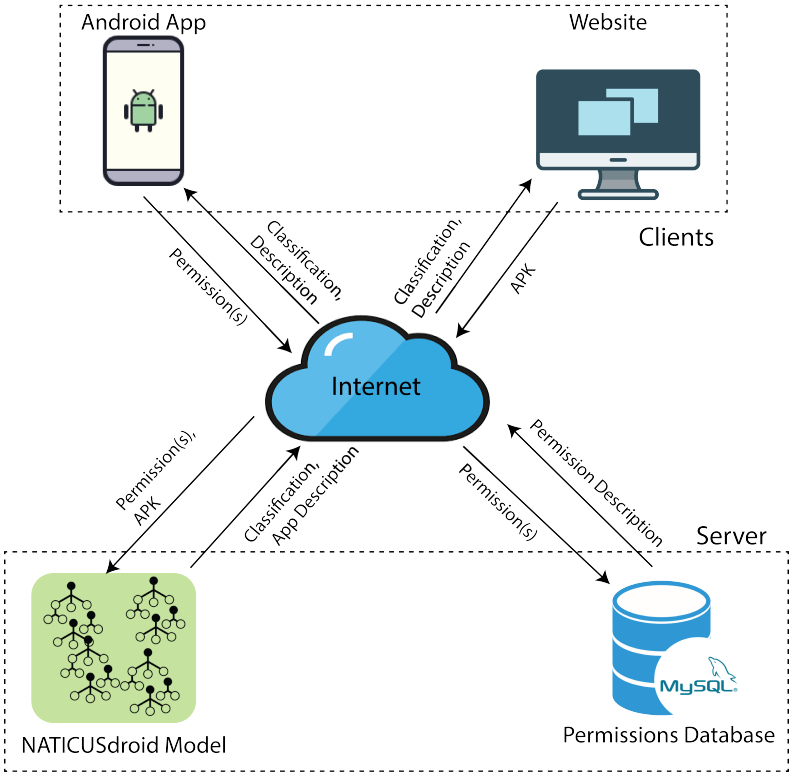


Fig. 1: Permission-Educator Architecture

3 Permission-Educator

We believe by educating naive users about Android and its permission-based security model we can contribute more towards preventing malware attacks. The purpose of permission educator is to inform and make users understand the role permissions play in safeguarding our data and privacy. This app not only gives a detailed description of what a permission intends to do, but also uses these permissions to detect any potential malware applications on the device. This is intended to give the user an idea as which permissions can play a crucial role in data privacy. The proposed web application also allows the user to check whether the apk at hand is malicious or not, and give a detailed description of the permissions used in `apk`.The source code of *Permission-Educator* is released on Github for adoption by the research community[14]. The overall architecture of Permission-Educator is shown in Fig. 1.

Algorithm 1: Pseudo code for separation of System and Store Apps

Input:
`installedApps[]`: Apps installed on phone

```
storeApps, storeAppNames, systemApps, systemAppNames = [];
for appInfo in installedApps do
    label = pm.getApplicationLabel(appInfo);
    if appInfo.FLAGS == TRUE &
      FLAG_SYSTEM == 1 & FLAG_UPDATED_SYSTEM_APP == 1
      then
        systemApps.add(appInfo);
        systemAppNames.add(label);
    else if appInfo.FLAGS == TRUE &
      FLAG_SYSTEM == 0 & FLAG_UPDATED_SYSTEM_APP == 0
      then
        storeApps.add(appInfo);
        storeAppNames.add(label);
```

3.1 Clients

There are two clients of Permission-Educator – an Android app and a website – that communicate with the cloud service.

Android App: The app is developed on Android Studio and evaluated on Google Pixel 2 Android Emulator running Android 7.1.1. This app will be compatible with the latest Android versions as well. It categorizes all the apps in a user's smartphone into System and Store apps and allows them to select an app from one of these categories. *System apps* are pre-installed and signed with the manufacturer's certificate, while *Store apps* are usually installed from different app stores. This grouping was done for two

reasons: i) It provides a simple method of sorting the apps into more organized groups by offering a simple user interface, and ii) it allows the user to see differences in the type, category, and frequency of permissions System and Store Apps. We separated the apps using `ApplicationInfo` class and its two flags – `FLAG_SYSTEM` and `FLAG_UPDATED_SYSTEM_APP`. The former is used to discern whether an app is installed in the device's system image, which, if set to true, the app was present when the user first booted up the device. The latter is then used to discern whether the application has been installed as an update to a built-in system application, which means system applications that the user can use. Algorithm 1 shows the pseudo-code for separating system and store apps.

The Website: The website was developed using HTML 5 and integrated with JavaScript for dynamic components. The purpose of the website is to provide a summary of any uploaded Android app (apk) from a Desktop system similar to VirusTotal [15]. It provides information for the permissions used in the app, the purpose and category of each permission, and the overall behavior of the app (malware or benign). We also used Jinja2 library [16] for integrating the website's front-end with the Python back-end.

3.2 Cloud Service

We have used a Flask server at the backend of cloud service. NATICUSdroid was hosted on the Flask server. The server receives all the queries using POST requests. The database to store permissions information was implemented using MySQL Server.

Flask Server: Flask server is a Python-based micro web framework. It is classified as a micro framework as it does not require particular tools or libraries [17]. We deployed a malware detection system, NATICUSdroid, on this server. The server receives permissions or apks from the clients and send the requested information in String format.

NATICUSdroid is an Android malware detection system that our research team has recently developed [5]. It is an adaptive, lightweight, fast, and robust ML-based framework developed to detect Android malware spreading from third-party Android app stores and phishing attacks. We analyzed an extensive and recent collection of benign and malicious Android apps and found that several 'dangerous' native and custom Android permissions were used frequently in both benign and malware apps. A rigorous feature selection process yielded a combination of 56 native and custom permissions, which were critical in the classification process. We used these as features for training a Random Forest Classifier that classifies apps with an accuracy of 96.9%. Since the classifier was trained using only permissions from both malware and benign apps, it provides the most appropriate insight on which apps could potentially be malicious.

MySQL server: The MySQL server hosts a database that holds two tables – *permissions* and *missed_permissions*. The *permissions* table consists of three fields – `perm_name`, `perm_about`, and `perm_category`. `perm_name` stores permission name as mentioned in `AndroidManifest.xml` file, `perm_about` stores permission information retrieved from Android Developers website [1] and several other sources to have

as much detail about the permission as possible, and `perm_category` stores permission category, i.e. *Normal, Dangerous, Signature,* and *Privileged*. The field `perm_name` is the primary key for this table. All three fields store string values. Once a permission is clicked on the client, the permission name is sent to the MySQL server, where a query retrieves the information and category of the permission and returns it to the client. *missed_permissions* table consists of two fields – `perm_name` and `perm_category`. Both the fields store string values and similar information as in the *permissions* table. This table is used for keeping a record of permissions which are used in apps, but do not exist in the *permissions* table, as there are several custom permissions whose information are not available easily.

4 Results

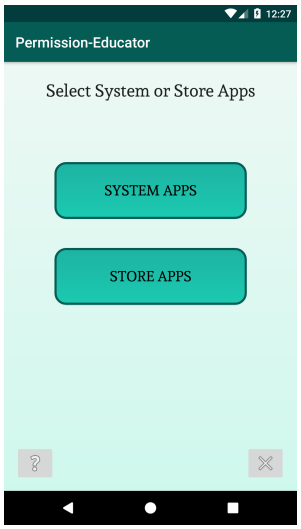
As Permission-Educator focuses on individuals who are not as fluent in Android technology and permissions, the goal is to provide a simple and informative User Interface (UI).

4.1 The App

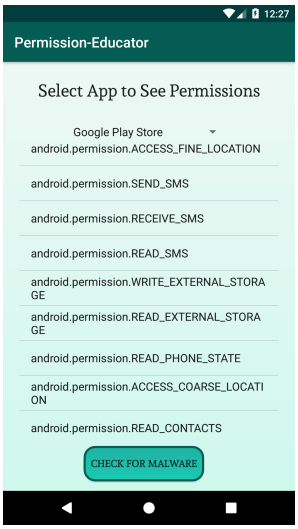
A user interface (UI) overview of the Android client app is shown in Figure 2. Figure 2a is the home screen of the app that provides the user a choice of selecting an app from two groups: one that comes pre-loaded on the phone during startup and whose files are not accessible by the user (System apps), and the other downloaded by the user from app stores or pre-downloaded apks that can be installed and uninstalled by the user (Store apps).

Figures 2b and 2d show the UI of "System Apps" and "Store Apps" activities. Once the user chooses system or store apps, a drop-down menu is provided listing all the apps in that category shown in Figure 2d. Once an app is selected, we provide a clickable list of all the permissions used by the chosen app, as shown in Figure 2b. This allows the user to see the details for each permission (either has access to or will request if it has not done so already) for the selected app. This is the first step in educating the user on app permissions, learning what an app has access to on their phone. However, that information alone is not sufficient. The users may not know the full context of every permission. For this reason, we provide the user with more information on what each permission means and what it lets the app access on the user's phone. To access this feature, the user needs to click on the permission of their choice, and they are redirected to a new screen where the chosen permission's information is retrieved from the server database and displayed as shown in Figures 2c and 2e.

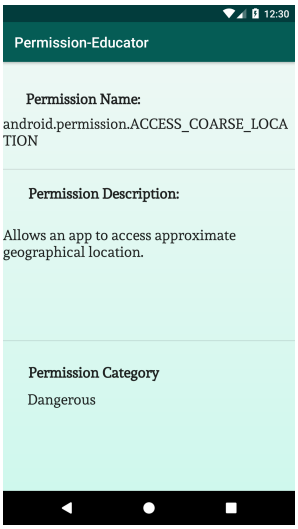
Once the permissions of a chosen app are shown, a button at the bottom *Check for Malware* is provided that sends (when pressed) all the permissions mentioned in the app's `AndroidManifest.xml` file to the server. A permission vector is created by matching the final permissions set defined for NATICUSdroid to the permissions received at the server. This vector is then passed to the model, which classifies the permission vector as malicious or benign. The predicted class is sent back to the client, which is displayed to the user in a dialog box as shown in Figure 2f.



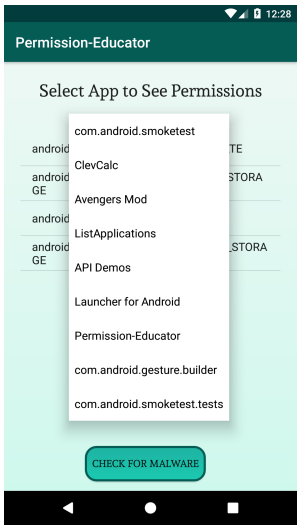
(a) Main Page of the App



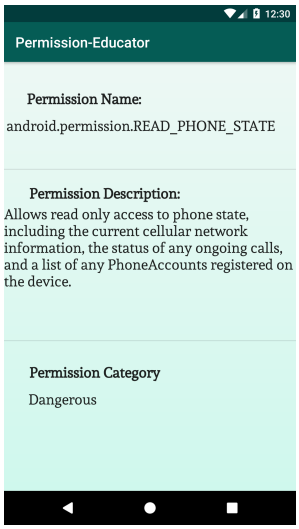
(b) System Apps (e.g. Google Play Store)



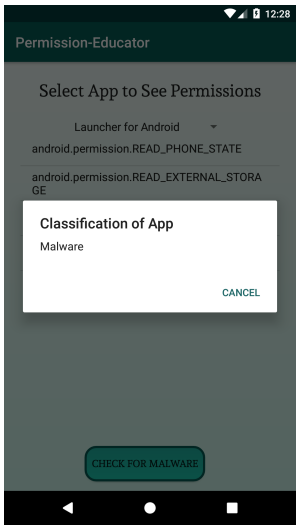
(c) Information of permission for System app



(d) Store Apps (e.g. Launcher for Android)

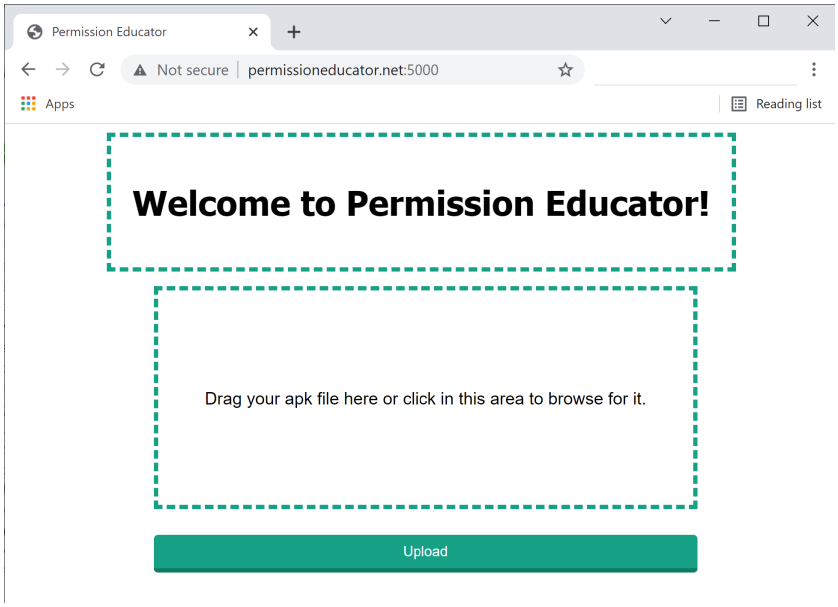


(e) Information of permission for Store app

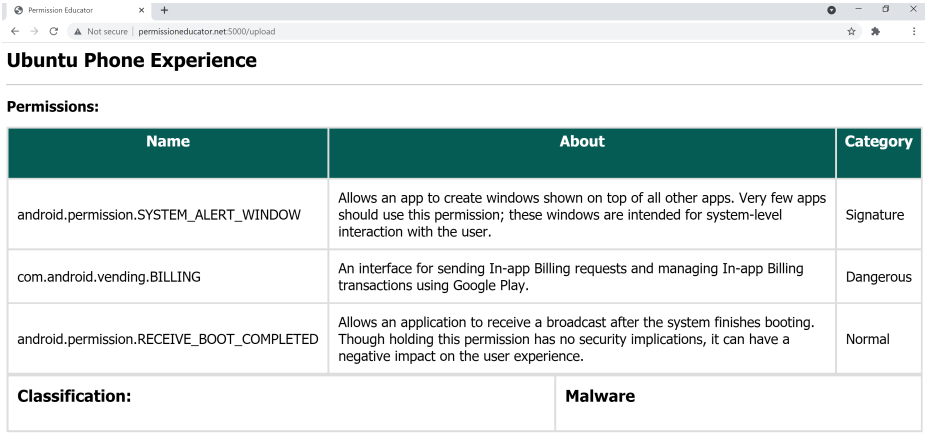


(f) Dialog box for App Classification

Fig. 2: User interfaces for Android app client of Permission-Educator



(a) Home Page of Permission Educator Website



Ubuntu Phone Experience

Permissions:

Name	About	Category
android.permission.SYSTEM_ALERT_WINDOW	Allows an app to create windows shown on top of all other apps. Very few apps should use this permission; these windows are intended for system-level interaction with the user.	Signature
com.android.vending.BILLING	An interface for sending In-app Billing requests and managing In-app Billing transactions using Google Play.	Dangerous
android.permission.RECEIVE_BOOT_COMPLETED	Allows an application to receive a broadcast after the system finishes booting. Though holding this permission has no security implications, it can have a negative impact on the user experience.	Normal
Classification:	**Malware**	

(b) Information for an uploaded App file, *Ubuntu Phone Experience*

Fig. 3: The UI of the Website

4.2 The Website

The website provides similar information as the app. Figure 3a shows the Home page of the website. The user can select or drag and drop an apk file from the system and click on "Upload" to send it to the server. The server, upon receiving the apk as a POST request, starts extracting the name of the app and the permissions mentioned in the `AndroidManifest.xml` file. These permissions are then queried in the database for corresponding information on its utility and category. If the permission name is available in the database, its information and category are retrieved and stored in a dictionary. The permissions are also used to create a vector of binary permission by matching the final permissions set defined for NATICUSdroid, which is fed to the pre-trained model. It returns a 'benign' or 'malicious' classification for the app. The permission dictionary and the classification result are returned to the web page to display this information, as seen in Figure 3b, for an app *Ubuntu Phone Experience*.

In the background `missed_permissions` table keeps track of the permissions for which there was a miss in the `permissions` table. This helped keep track of permissions and information that still need to be added to the database to have a high hit ratio. Upon collecting several such permissions, the `permissions` table is updated. An example of such permissions would be the custom permissions that are not native to the Android APIs, such as, `WRITE_USE_APP_FEATURE_SURVEY`, and so on, about which information is not readily available.

5 Conclusion and Future Work

Permission-Educator app works to bridge the knowledge gap of users about Android permissions and preventing malware infection. It helps educate the user by providing information such as the utility of the permission in the app, its defined category and helps in determining whether the app is malicious. A working, dynamic website also helps check an app's declared permissions, utility, and nature. This is done with the help of a cloud-based service, where the permission information is stored in a MySQL database and nature of the app is determined by a malware detection system NATICUSdroid, both hosted on the server. The app and the website have not been published publicly and are still in production. However, they have been evaluated on a private network and the results presented are through test cases.

We aim to bring more information to the user about apps that can help determine the use of permissions and whether they should be granted to an app. A quiz module could also be added to the app, which can regularly check users' knowledge about permissions to observe a possible change in users' knowledge or attitude towards permissions. The app can then be released on Google Play Store. This will enable us to get more feedback from the users and improve the app over time.

6 Acknowledgement

This project was partially supported by National Science Foundation Grant Awards #1903419 and #1903423.

References

1. Android for Developers. https://developer.android.com/, December 2019. Accessed: 2020-04-24.
2. Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security, pages 627–638. ACM, 2011.
3. Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware guard extension: Using sgx to conceal cache attacks. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 3–24. Springer, 2017.
4. R. Mathew. Study of privilege escalation attack on android and its countermeasures. 2012.
5. Akshay Mathur, Laxmi Mounika Podila, Keyur Kulkarni, Quamar Niyaz, and Ahmad Y Javaid. Naticusdroid: A malware detection framework for android using native and custom permissions. Journal of Information Security and Applications, 58:102696, 2021.
6. Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. A conundrum of permissions: Installing applications on an android smartphone. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7398 LNCS:68–79, 2012.
7. Efthimios Alepis and Constantinos Patsakis. Unravelling Security Issues of Runtime Permissions in Android. Journal of Hardware and Systems Security, 3(1):45–63, 2019.
8. Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. SOUPS 2012 - Proceedings of the 8th Symposium on Usable Privacy and Security, 2012.
9. Selvakumar Ramachandran, Andrea Dimitri, Maulahikmah Galinium, Muhammad Tahir, Indirajith Viji Ananth, Christian H. Schunck, and Maurizio Talamo. Understanding and granting android permissions: A user survey. Proceedings - International Carnahan Conference on Security Technology, 2017-October:1–6, 2017.
10. Gian Luca Scoccia, Ivano Malavolta, Marco Autili, Amleto Di Salle, and Paola Inverardi. User-centric android flexible permissions. Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017, (i):365–367, 2017.
11. Arnaud Oglaza, Romain Laborde, Abdelmalek Benzekri, and François Barrère. A recommender-based system for assisting non technical users in managing Android permissions. Proceedings - 2016 11th International Conference on Availability, Reliability and Security, ARES 2016, pages 1–9, 2016.
12. Asma Hamed and Hella Kaffel Ben Ayed. Privacy risk assessment and users' awareness for mobile apps permissions. Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA, 0, 2016.
13. Majda Moussa, Massimiliano Di Penta, Giuliano Antoniol, and Giovanni Beltrame. ACCUSE: Helping Users to Minimize Android App Privacy Concerns. Proceedings - 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2017, pages 144–148, 2017.
14. Akshay Mathur and Ethan Ewoldt. Permission Educator App. https://github.com/akshaymathur05/Permission_Educator.
15. VirusTotal. https://www.virustotal.com/, Accessed: 2021-07-05.
16. Jinja2. https://jinja.palletsprojects.com/en/3.0.x/, Accessed: 2021-07-05.
17. Flask (web framework), 2012. https://en.wikipedia.org/wiki/Flask_(web_framework), Accessed: 2021-07-05.