A Multi-Dimensional Analysis of IoT Companion Apps: a Look at Privacy, Security and Accessibility

Faiza Tazi, Suleiman Saka, Shradha Neupane, Ethan Myers, Sanchari Das, Lorenzo De Carli, Indrakshi Ray

***Abstract*—Internet of Things (IoT) devices provide convenience to users by simplifying household tasks. Most IoTs can be remotely controlled via mobile companion apps, which constitute the main interface between devices themselves and their users. Such apps are used to configure, update, and control the device(s) and thus constitute a critical component in the IoT ecosystem. However, they have historically been understudied which prompts us to look into them. In this paper, we report on a study where we evaluated a sample of 455 IoT companion apps and analyze their privacy, security, and accessibility aspects. Our research aim is to understand these metrics, gauge their state and evaluate whether there is a correlation between them. Our primary findings from the analysis are: (i) most apps have reasonable security and accessibility posture, but in several dimensions there exists a long tail of apps with significant problems and (ii) apps tend to over-request permissions which are not related to their main goal. Moreover, the quality of an app along one aspect is uncorrelated to the same along other aspects. We conclude with actionable recommendations for companion app developers.**

***Index Terms*—IoT devices, IoT security, IoT privacy, Mobile security, Mobile Apps, Accessibility Evaluation**

I. Introduction

The Internet of Things (IoT) refers to interconnected smart devices that communicate through wireless networks. From wearable tools to high-tech devices, IoT has become an integral part of our lives, for entertainment [1], health monitoring [2], online education [3], and other daily activities [4]–[6]. As there is no universally recognized definition for IoT device or companion app, we settle for one which is sufficiently precise to be rigorously applied: *An IoT device is one which senses/actuates the physical world and is able to exchange sensing/actuation data with another networked node* [7]. In practice, this style of definition is typically extended to exclude non-embedded computing devices such as laptops and smartphones; we follow the same approach in our work.

IoT devices are not used in a standalone fashion, but almost universally via mobile apps [8]. Thus, *a companion app is an app which interfaces with an IoT device to sense/affect the physical world by leveraging the transducer from the device; and transmits the captured information over a network.* In other words, a companion app acts as a *gateway* between the IoT device, and the user and cloud backends [8].

IoT companion apps provide visibility into the data exchanged by connected devices (e.g., camera feeds, network details) and control over potentially sensitive operations (e.g., smart locks, children's toys). Despite this, they are generally used by individuals with limited technical knowledge and often lack transparent information. Many of these apps suffer from poor design and implementation, resulting in both usability and security challenges [9]–[11]. IoT technology is becoming more relevant to vulnerable populations; for example, these devices increasingly support independent living for seniors [12] and offer interactive experiences for children through smart toys [13]–[15]. Consequently, interfaces between IoT devices and users must prioritize accessibility, privacy, and security [16]–[19].

Indeed, most works on privacy, security and accessibility of IoT devices focus on the devices themselves, ignoring their mobile companion apps. To make matters worse, online characterizations of user experience about IoT companion apps insinuate poor design in regard to key aspects [20], [21]. This affects mostly privacy and security, but also the accessibility of apps. These shortcomings can be detrimental to the privacy and security of users. Furthermore, the adoption of companion apps depends on the accessibility of such applications.

To the best of our knowledge, no rigorous assessment of the software quality of companion apps along the above dimensions exists. Therefore, closing this gap is important and will further help to identify problem areas deserving further study and industry focus. To this effect, we conducted an analysis of 455 IoT mobile companion applications collected from the Google Play Store, and examined the aforementioned key aspects, namely privacy, security, and accessibility. For each aspect, we measure relevant metrics using appropriate and carefully selected tools. We then report the distribution of values and existing correlations between multiple metrics.

In both our security and accessibility analyses, we find that most apps have an acceptable posture, although many have serious issues (e.g., apps with known CVSS vulnerabilities with severity in the "medium" range; apps with > 100 accessibility rule violations). While our privacy analysis did not identify serious systemic problems, it found that many apps over-request unnecessary permissions.

Broadening our analysis along multiple dimensions allows us to answer important empirical research questions: *Are software quality issues concentrated along a single dimension, or do they propagate along multiple dimensions?*; and *Is there evidence of trade-offs where high quality along a certain*

F. Tazi and S. Saka are with the Computer Science Department, University of Denver, Denver, CO, USA.

S. Neupane was with the Computer Science Department, Worcester Polytechnic Institute, Worcester, MA, USA at the time this work was performed.

E. Myers and I. Ray are with the Computer Science Department, Colorado State University, Fort Collins, CO, USA.

S. Das is with the Information Sciences and Technology, George Mason University, Fairfax, VA, USA.

L. De Carli is with the Electrical and Software Engineering Department, University of Calgary, Calgary, AB, Canada.

dimension correlates with low quality along another one?

This work is an extension of our previous publication [22], whose contributions were (i) a method to iscrape IoT companion apps from the Google Play store; (ii) methods to assess aspects of app privacy and security; and (iii) a large-scale measurement of app privacy and security. It also integrates results from our recent extended abstract on accessibility [23].

In particular, this article extends our previous work by: (iv) augmenting our cross-dimensional analysis, by incorporating selected metrics from our privacy, security and accessibility measurements. This enables us to analyze potential correlation between different metrics, exploring the possibility of tradeoffs not only between privacy and security, but also accessibility. (v) Compared to our conference submission, we also significantly extend this analysis by stratifying our app set in different categories by number of downloads and user review scores, and investigating the presence of correlation between metrics within each category. Finally, (vi) we present more extensive take-aways from our effort, including considerations about the viability of existing app analysis tools in the Android ecosystem. These considerations have practical implications both for further analysis effort, and mobile app development.

Overall, this article presents in a unified way the results of several years of work on measuring relevant characteristics of IoT-related mobile apps int he Android ecosystem, which our previous publications only discussed in limited and disjoint form. As such, it provides substantial new value to security practitioners and developers of IoT mobile companion apps.

II. Data Collection and Analysis

A. Data Collection

Our data collection methodology was inspired by Wang et al [24], and consists of the following steps.

- **Step 1: Manual Search**: We manually downloaded IoT apps from Google Play store. We did so by looking through apps used in the context of smart home or collecting information about the physical environment and connectivity. This formed our "Seed App Set".
- **Step 2: App Scraping**: For each app in our Seed App set, we scraped app names and descriptions through the "Similar Apps" suggestions in the Play Store using play-scraper[1]. After this, we had a set of 2000 scraped apps.
- **Step 3: Keyword-based Filtering**: We performed keyword-based filtering to remove obvious false positives, i.e., apps that did not match our definition of IoT companion apps. We empirically generated a set of keywords based on the correlation of keywords to these false-positive apps. Then we removed apps that match specific keywords (e.g. *currency, compiler, etc.*)
- **Step 4: Naïve-Bayes Classification**: Using Machine Learning we then refined the candidate set to classify IoT and non-IoT apps. After evaluating various approaches we converged on Naïve-Bayes, which led to the best accuracy although far from optimal (64.6%). In aggregate, Step 3 and Step 4 reduced our set of 2000 apps to 1596 (20.2%), by weeding out a large number of IoT-unrelated apps returned by the scraping process.
- **Step 5: Manual Filtering**: Due to the limited accuracy shown by the Naïve-Bayes classifier on preliminary experiment, we decided to further manually review all remaining apps, and only retain the ones which match our definition of companion apps, resulting in 484 (30.3%) matching our definition of IoT apps.

Our automated filtering (steps 3 and 4) uses a simpler approach than that of Wang et al., who use Affinity Propagation clustering [25]. Compared to Naïve-Bayes, their approach results in only a modest increase (10%) in filtering power based on published results. We consider our approach acceptable since automated filtering is followed by manual review.

We acknowledge that manual review can introduce experimenter bias. First, we note that whe two experimenters who performed app filtering also participated in formulating our shared definition of companion app (Section I). To further confirm accuracy, a third experimenter reviewed a sample of 10% of selected apps (45 apps) to check for inaccuracies. The review identified 41 apps (91%) controlling household devices (alarms, appliances, etc.), 3 apps (7%) for remote management of smart cars, and 1 app (2%) generically designed for remote access to microcontrollers. Thus, we conclude that our approach is effective in identifying relevant apps.

Data collection results: Once we had the list of IoT apps, we downloaded the app packages using PlaystoreDownloader[2]. Out of the 484, we were able to retain only 455 APKs because the remaining could not be downloaded.

B. Data Analysis and Reporting

For each app being examined wrt. privacy, security and accessibility, we measure relevant metrics. For each, we report the distribution of the metric values across the app dataset, and correlation (if any) between multiple metrics within the same category. In Section VI, we also report the correlation between aggregate metrics. As the distribution of many metrics strongly deviates from normal, we use non-parametric Spearman's rank correlation (r_s). According to accepted practice [26], we interpret r_s values above 0.7 as representing high correlation; values between 0.5 and 0.7 as moderate, and values between 0.3 to 0.5 as low correlation.

III. Privacy Analysis

Android provides a permission-based privacy model for third-party app developers which requires developers to declare the sensitive permissions their apps use. These control access to hardware, settings, and user data. The user is informed of such permissions during app installation [27]. Further, Google Play Store, as well as multiple privacy laws (e.g., GDPR), require app developers to provide a detailed privacy policy [28], [29]. Through our analysis, we leveraged Android's permission-based model and privacy policies to evaluate IoT companion applications' privacy. In this section, we detail the methods of data analysis and the results found.

[1]https://pypi.org/project/play-scraper

[2]https://github.com/ClaudiuGeorgiu/PlaystoreDownloader

TABLE I: Details of privacy categories and their associated permissions.

Category	Permissions
Network	INTERNET, ACCESS_NETWORK_STATE, CHANGE_NETWORK_STATE, ACCESS_WIFI_STATE, CHANGE_WIFI_STATE, CHANGE_WIFI_MULTICAST_STATE
Content	WRITE_MEDIA_STORAGE, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, MANAGE_EXTERNAL_STORAGE, MANAGE_MEDIA, MOUNT_FORMAT_FILESYSTEMS
Location	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION, ACCESS_BACKGROUND_LOCATION, ACCESS_MEDIA_LOCATION, ACCESS_LOCATION_EXTRA_COMMANDS
Device Id	READ_PHONE_STATE, READ_BASIC_PHONE_STATE, READ_PRECISE_PHONE_STATE, MODIFY_PHONE_STATE
Contact	WRITE_CONTACTS, ACCOUNT_MANAGER, READ_CONTACTS
Telephony	CALL_PHONE, PROCESS_OUTGOING_CALLS, READ_SMS, SEND_SMS
Services	READ_PHONE_NUMBERS, USE_SIP, MANAGE_ONGOING_CALLS, CALL_PRIVILEGED, ANSWER_PHONE_CALLS, WRITE_CALL_LOG, READ_CALL_LOG, RECEIVE_SMS
Calendar	WRITE_CALENDAR, READ_CALENDAR

A. Methods

We started by first extracting the mobile app permissions from the app manifest. These permissions provide access to resources that manage sensitive data such as personally identifiable information, location data, contact books and so on [30]. This classification is necessary as these categories can reveal sensitive user and device information regardless of the app type. The permission categories included: *Network*, *Content*, *Location*, *Device ID*, *Contact*, *Telephony Services*, and *Calendar* [31]. We then identified the Android permissions that allowed access to data pertaining to these categories. The details and permission requested by these categories are provided in Table I. The app manifest provides detailed information [32], [33] including measurements the app does to obtain user data, i.e. the permissions required, details, and how it is collected, thus analysis of the app manifests is critical [34], [35]. The first relevant issue is whether an app exhibits *requests for privacy-sensitive permissions*. However, the presence of sensitive permission does not imply leaks; thus we also consider *evidence of permission-related data leaks*. The app privacy policy, when available, also provides context. Thus we perform an *analysis of privacy policy*.

B. Tools and Analyses

*Manifest Permission Analysis **(Cat. 1)***. From the declaration of the app permissions, we calculate privacy scores for the categories discussed above using $S_c = p_c / \sum_i p_i$ where S_c is the score for category c, p_c is the number of permission pertaining to c, and $\sum_i p_i$ is the sum of all permissions pertaining to c, This calculation is an extension of Kang et al.'s *privacy meter* [36]. Each category can request several permission (Table I). For each, we assign 0 or 1 depending on whether the app requests the permission. Thereafter, we follow a hierarchical permission model. In the case of higher privilege access encompassing lower privileges, we only considered the higher privileges. For example, *Full Network Access* encompasses all the other network-related permissions. Thus, if an app only requests this permission, it will get a score $S_{network} = 1$. Similarly, if an app is requesting fine-grained location, it is considered as $p_{location} = 2$ since coarse-grained location is covered by fine-grained. Furthermore, permission to write storage implies permission to read. This also applies to read/write contacts, call logs, and calendar events. This results in an app privacy score between 0 and 1, with 0 indicating the highest degree of privacy preservation, and 1 the lowest.

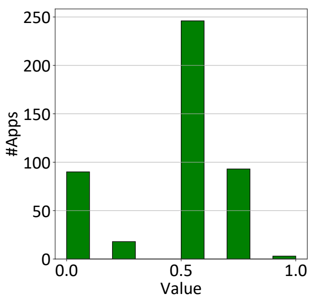
TABLE II: Mean, min and max for each privacy category

Perms	Mean	Min	Max
Network	0.75	0	1
Content	0.26	0	0.5
Location	0.44	0	1
Device Id	0.08	0	0.5
Contact	0.12	0	1
Telephony Services	0.01	0	0.75
Calendar	0.01	0	1

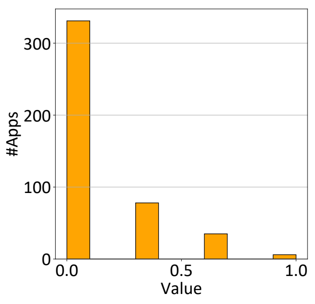
*FlowDroid **(Cat. 2)***. We use FlowDroid to analyze each app's privacy leaks and map these leaks to permissions that are necessary for the app to have access to that type of data. FlowDroid investigates data flows between *sources* and *sinks*. Source represents the location where sensitive information may be entered and sink represents the location where information could leave the app. FlowDroid parses app binaries and produces an analysis of the application call graph as the output, including paths from source to sink. These represent situations where leakage of sensitive data is possible. In this analysis, we only consider the sources, since leaking personal data is a breach of privacy, no matter the destination. We consider this approach acceptable as our goal in this work is to broadly characterize the privacy profile of an app, rather than analyzing each app's treatment of personal data in details.

*Analysis of Privacy Policy **(Cat. 3)*** We conduct analysis of app privacy policies using Polisis [37]. Polisis passes a privacy policy into a set of neural network classifiers to label the privacy practices described in the policy; the labels include 10 high-level and 122 fine-grained classes. These classifiers are trained on the OPP-115 dataset by Wilson et al. [38].

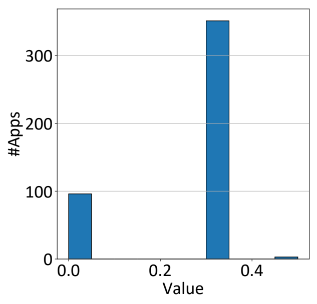
The output is a classification by category level for the segment classifier and attribute levels where applicable for each segment. In this analysis we were mainly interested by the following classification labels: *First Party Collection*, *Third Party Sharing*, *Access*, *Edit*, *Delete*. As for the attributes we were interested in: *personal-information-type*, *third-party-entity*, *access-type*, *action-first-party* and *identifiability*.



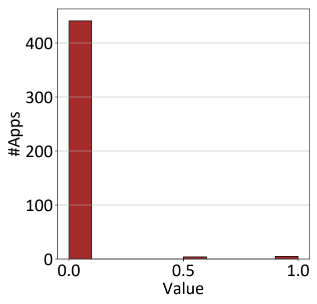
(a) Location privacy score



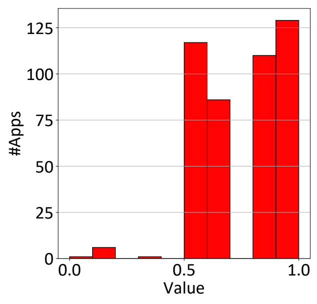
(b) Contact privacy score



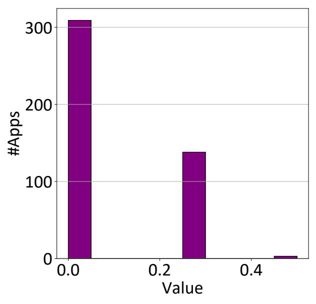
(c) Content privacy score



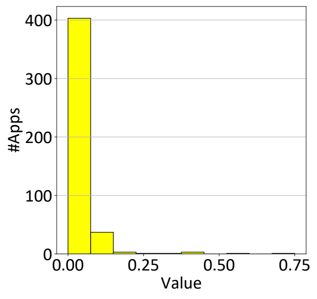
(d) Calendar privacy score



(e) Network privacy score



(f) Device Id privacy score



(g) Phone privacy score

Fig. 1: Individual distributions of privacy scores for the IoT applications

TABLE III: r_s Between privacy categories (Loc = Location; Ctc = Contacts; Con = Content; Cal = Calendar; DId = Device Id; TeS = Telephony Service.)

	Net	**Con**	**Loc**	**Did**	**Ctc**	**TeS**	**Cal**
Net	1.00	-	-	-	-	-	-
Con	0.325	1.00	-	-	-	-	-
Loc	0.388	0.296	1.00	-	-	-	-
Did	0.119	0.216	0.182	1.00	-	-	-
Ctc	0.195	0.195	0.222	0.293	1.00	-	-
TeS	0.065	0.062	0.114	0.305	0.374	1.00	-
Cal	-0.062	-0.002	0.0	0.090	0.195	0.310	1.00

C. Results

1) Manifest Permissions Analysis: 129 of the 455 apps got a network permission score of 1, which means that these apps either requested the whole set of network permissions or opted to request the highest privilege permissions. A high score is expected as companion apps need to be linked with a device through the network. The mean network permission score was 0.75. The mean location permission score was the second-highest (0.44). The most requested permissions for location are ACCESS_COARSE_LOCATION and ACCESS_MEDIA_LOCATION which allows apps to access locations saved within the user's media.

The content permission score mean was not as high (0.26), with most applications (353) requesting permission to write external storage. Only 92 apps did not request any content permission. The remaining categories had lower mean scores. It should be mentioned, however, that none of those categories is necessary to implement the core functionality of a companion app. Figures 1a-g detail the privacy score for each category.

Table III presents Spearman's rank correlation coefficient r_s values between each pair of privacy categories (for all reported values, $p < 0.01$). We use Spearman as score distributions is non-normal. The most relevant finding is that a number of categories exhibit weak (> 0.3) correlation with each other.

2) FlowDroid Permissions Analysis: FlowDroid detected leaks in 315 apps. 96 applications presented privacy leaks that were caused by custom permissions, not included in this analysis. 219 applications presented leaks caused by a set of 8 permissions: ACCESS_FINE_LOCATION, ACCESS_WIFI_STATE, READ_PRIVILEGED_PHONE_ STATE, RECORD_ AUDIO, READ_SMS, READ_ PHONE_STATE, READ_PHONE_NUMBERS, BLUETOOTH_CONNECT. Six of these were identified in privacy categories from Table I.

The permission which provided data for the most number of leaks was ACCESS_FINE_LOCATION, with 214 apps leaking information related to fine-grained geographical location. Only 20 apps declared this specific permission in their manifest. A less significant issue concerned 20 apps leaking privileged phone state data. Moreover, 4 apps leaked audio recording data. Similarly, SMS, phone numbers, and phone state data were leaked by 3 apps each. Results are shown in Figure 2.

3) Privacy Policy Analysis: Polisis: We analyzed personal information type labels (as categorized by Polisis) declared in app privacy policies. Polisis uses the following 15 personal information type labels: *Computer information (CI)*, *Contact (CON)*, *Cookies and tracking elements (C&TE)*, *Demographic (DEM)*, *Financial (FIN)*, *Generic personal information (GPI)*, *Health (HLT)*, *IP address and device IDs (IP&ID), Location (LOC)*, *Personal identifier (PI)*, *Social media data (SMD)*, *Survey data (SURV)*, *User online activities (UOA)*, *User profile (UP)* and *Other Data (OD)*. Polisis predicts attributes' labels with an average precision of 0.84, and only considered a prediction of over 0.5. The distribution of number of labels per

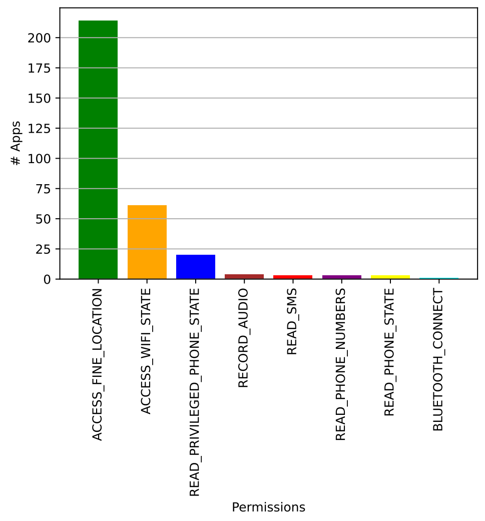


Fig. 2: Permissions responsible for the data leaks found through FlowDroid

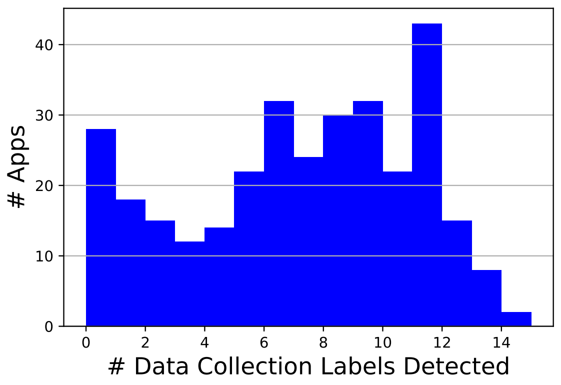


Fig. 3: Frequency of labels detected per app.

app is in Figure 3. The maximum number of labels detected in a privacy policy was 14 in two apps, and the average number of labels per privacy policy was 6.76. The *Other Data* label was the most recurrent label, predicted in 277 privacy policies. The least frequent label was *Location* which was predicted in 41 of the policies. 28 privacy policies stated that they did not collect personal information on users.

Limitations: In this section, we are limited in analyzing exclusively the permissions stated in each app manifest rather than the true permissions used by the app itself. As such, the analysis is strictly constrained to the permissions declared and not to the ones actually used. Furthermore, we only considered predefined permissions for our permission analysis. Finally, the privacy policy analysis was carried out for 402 apps (the remaining had non-English policies unsupported by tooling, or missing policy.)

4) Take-aways: Significant number of apps request permissions (e.g., *phone state*) not directly related to their typical functions; although the privacy score for such categories is low. Many apps leaked information requested through ACCESS_FINE_LOCATION, without declaring this permission. Many app providers did not describe all the types of collected data in their privacy policy.

IV. Security Analysis

Both mobile phones and IoT devices have the ability of hold and sense personal and private data. It is therefore important that companion apps are maintained according to security best practices and free of known vulnerabilities.

A. Method

Many app security analysis techniques exist (e.g., [39], [40]). For a technique to be used in our analysis, we require the following: tooling must be available; not abandoned (we used a fixed threshold of two years w/o updates); and executable on a modern machine/OS. Given the techniques meeting the criteria above, we then restrict our search to those measuring one of three different categories. We determined these categories based on discussion within the group, and our collective expertise on security issues in mobile apps. The first is the *presence of known vulnerabilities* (1), as vulnerable software is at higher risk of facilitating information leaks. However, an app may also suffer from latent, undiscovered security issues. Thus, we also measure *evidence of lack of maintenance* (2), as lack of updates may cause the presence of latent issues [41]. The third category is lack of *in-transit data protection* (3), as even an app without security vulnerabilities may leak data in transit, if this data is not appropriately encrypted.

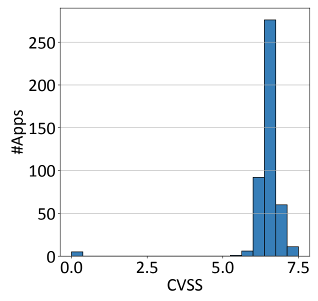
B. Tools and Analyses

Based on our three categories of analysis, we reviewed existing security analysis tools (a more in-depth discussion is presented in Section VII-B). We ultimately converged on performing the bulk of the analysis using the Mobile Security Framework (MobSF)[3] which is a mature Open-Source pen-testing tool for Android, popular in Android security research [42]–[44]. MobSF implements a vulnerability scanner, which is appropriate for Category 1 - presence of known vulnerabilities. It also identifies various misconfigurations in transport-layer security, causing data to be improperly encrypted and are thus relevant to Category 3 - lack of in-transit data protection. Note, however, that MobSF is unable to identify all possible data protection issues, and thus we complement this analysis with further manual inspection. Finally, MobSF does not perform analyses relevant to category 2 - *Evidence of lack of maintenance*. Therefore, we operationalized and measured this characteristic as software abandonment. We discuss individual analyses below.

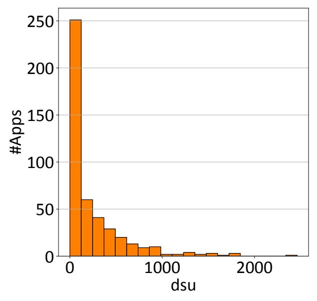
*MobSF CVSS score **(Cat. 1)**.* We use MobSF to statically fingerprint common, generic security-adverse patterns. MobSF further assigns a severity score between 0 and 10 to each issue, using the CVSS scoring system [45]. The tool also outputs the average CVSS scores across detected issues. MobSF also generates information about the high-level category associated with each problem, based on categorization [46].

*Days since update **(Cat. 2)**.* Modern apps make use of many third-party libraries. Every software needs to be updated and patched regularly as new vulnerabilities get discovered. We use Days Since Last Released Update as an analysis

[3]https://github.com/MobSF/Mobile-Security-Framework-MobSF



(a) CVSS scores



(b) Days since update

Fig. 4: Distribution of values for security metrics across apps

to investigate abandonment issues. This is consistent with previous work [47], which has used "failure to release" as a proxy symptom of abandonment for software projects.

MobSF transport misconfiguration detection ***(Cat. 3)***. HTTPS (Secure HTTP) uses the TLS standard for encrypted and secure transfer of data. We use MobSF static analyzer to identify app misconfigurations which may allow data to be transmitted in an improper encrypted form. Even if TLS is configured correctly, the use of invalid TLS certificates may create risks. Thus we also use MobSF's static analyzer to identify invalid/expired TLS certificates.

MobSF transport secrecy issue detection ***(Cat. 3)***. We complement static TLS analysis with dynamic analysis of app-generated connections. We perform four MobSF-driven tests: TLS misconfiguration, TLS Pinning/Certificate transparency, TLS Pinning/Certificate transparency Bypass, and Cleartext Traffic. The TLS misconfiguration uncovers insecure configurations allowing HTTPS connections bypassing certificate errors or SSL/TLS errors in WebViews. The TLS certificate transparency bypasses test attempts to bypass the certificate or public key pinning and certificate transparency controls. The ClearText Traffic test inspects whether an app exchanges non-TLS-encrypted traffic.

Manual user flow analysis ***(Cat. 3)***. We manually performed the following operations: user creation, password recovery, login, and single sign-on (if applicable), while recording traffic generated by the app. Subsequently, we inspected the traffic for issues. Due to the significant manual effort involved, we limited this test to 15 apps that were found to transmit unencrypted data by other tests. Inspection involved determining the nature and sensitivity of such unencrypted data.

Limitations: The MobSF static analyzer failed on 3 apps. 20% of apps we run through the *Transport Layer Secrecy Issues* analysis crashed due to incompatibility with test environment (Android Studio) and other issues.

C. Results

1) Presence of Known Vulnerabilities: Measurements of the aggregate MobSF CVSS score resulted in an average score of 6.45, with a minimum of 0 and a maximum of 7.5. The distribution is presented in Figure 4a. Scores exhibit a fairly concentrated distribution, with only 5 apps showing a score of 0 and 417 having a score of 4.0 to 6.9, which is considered medium severity according to the CVSS qualitative severity scale [45]. While the data may appear to be close to normally distributed, it fails the Shapiro-Wilk normality test ($W = 0.37$, $p = 1.68e^{-36}$). It is important to contextualize this result, as they suggest that a large number of apps may be affected by somewhat serious security issues. Upon further analysis, we determined that MobSF captures several discouraged but common programming practices; the fact that a small set of issues recurs across a large number of apps causes the consolidation of scores observed in the data. While such practices pose security risks in principle, at least some of those would require significant effort by an attacker to turn into active exploits. We elaborate below.

TABLE IV: Common CWEs found by MobSF Issue Analysis

CWE	Name	Apps
CWE-532	Insertion of Sensitive Information into Log File	445
CWE-276	Incorrect Default Permissions	411
CWE-312	Cleartext Storage of Sensitive Information	403
CWE-330	Use of Insufficiently Random Values	390
CWE-327	Use of a Broken or Risky Cryptographic Algorithm	381

Table IV lists the top-5 CWEs associated with the vulnerabilities in the apps. Several common CWEs identify behaviors that might pose relatively minor threats. CWE-532 was found in almost every application because they were creating files in the system or logging operation results. Other common CWE such as CWE-312, -330, and -327 are concerning as they indicate deviation from best data secrecy practices.

2) Evidence of Lack of Maintenance: The average number of days since an update is 237 (min: 0; max: 2462). The distribution (Figure 4b), shows that most apps are frequently updated: more than 238 were updated within the last 100 days.

3) Lack of In-transit Data Protection: **Transport misconfigurations:** Table V summarizes network-related vulnerabilities in an app's network security configuration, which centralizes network security settings in a standardized file. Enforcing directives in the configuration is up to individual network libraries, and indeed not all libraries respect the configuration [48]. The top-two issues allow unencrypted communications to all or some domains. The others violate various best practices concerning trusting TLS certificates. We also measured tinvalid/expired TLS certificates, which affects 96 out of 452 (21.2%) applications.

TABLE V: Breakdown of transport misconfiguration issues detected by MobSF across the app dataset

Description	Apps
Base cfg is insecurely configured to permit clear text traffic.	99
Domain cfg is insecure and permits clear text traffic.	71
Base cfg is configured to trust system certificates.	65
Base cfg is configured to trust user installed certificates.	16
Base cfg is configured to bypass certificate pinning.	12
Domain cfg is configured to trust system certificates.	5
Domain cfg is configured to trust user installed certificates.	2

Transport secrecy issues: For this dynamic analysis, we selected a number of apps for which other analyses had identified potentially concerning issues. We further reduced this set to 15 apps based on quantitative risk analysis by RiskInDroid [49]. Results are summarized in Table VI. This test identified

TABLE VI: Dynamic analysis results for the selected 15 apps

TLS Test	Percent passing
Cleartext traffic Test	60%
TLS Misconfiguration Test	46.7%
TLS Pinning/Cert. Transparency Bypass Test	73.3%
TLS Pinning/Cert. Transparency Test	53.3%

significant TLS deployment issues. Note that, differently from the "Transport layer misconfigurations" test—which represents issues potentially leading to security problems—the dynamic test identifies issues concerning the actual network traffic generated by apps.

Manual user flow analysis: To further investigate the results above, we manually analyzed app-generated traffic. This analysis revealed that, in practice, most apps were using TLS to send critical information and only used cleartext traffic to access public data. Two apps, however, had concerning behavior: (i) *XVRView (com.xvrview)* is an app developed by 杭州韬视 to monitor home cameras from a cell phone. The app has $>$ 100,000 downloads and receives active support from its developers, as the last update took place in February 2022. However, it failed every TLS test except "TLS Pining/Certificate Transparency Bypass". Our manual analysis found that all the user credentials are sent in clear text, which is an extremely risky practice. An on-path attacker could obtain credentials and get direct access to users' home cameras. (ii) *Vss Mobile*, by ZenoTech, has $>$ 500,000 downloads. It is a remote monitoring client that allows users to view videos over the network. At the time of testing, the application had not been updated in over a year. Manual traffic analysis revealed that the app transmits login credentials over unencrypted TCP connections. This is a critical issue that could breach data privacy and allow snooping on user data. We disclosed the issues to the app maintainers.

4) Correlation between metrics: We calculate the Spearman's correlation coefficient between the security metrics. We ran the analysis only for apps which had available values for all metrics. Spearman's analysis showed no significant pairwise correlation between any of the security metrics.

5) Take-aways: Results show that many apps are plagued by noncritical but concerning poor programming patterns (e.g., logging sensitive information in cleartext); and configuration issues affecting secrecy of in-transit data (e.g., use of expired certificates). An in-depth dive into problematic apps revealed sensitive credentials being sent in cleartext.

V. Accessibility Analysis

We report on the evaluation of the accessibility posture of various IoT apps within the Android ecosystem. Accessibility aims to improve user access to systems through the interface, hardware, and environmental characteristics of a system. Indeed, accessible websites, tools, and technologies should be designed and developed for everyone to use seamlessly, regardless of disabilities or special needs. Along these lines, an accessible technology enables all users to be able to perceive, understand, navigate, interact, and contribute to it [50].

A. Methods

To evaluate Accessibility, we used *Accessibility Insights for Androids*[4], which is a free, open-source tool that allowed us to evaluate accessibility based on the WCAG 2.1 guidelines. This tool can detect common accessibility issues such as: poor contrast, missing names and descriptions, or inadequate touch target sizes, through UI analysis. Accessibility Insights for Android bases its rules on the axe-android, which is an automated WCAG 2.0 and WCAG 2.1 Accessibility library for Android apps [50]. The tool works by taking screenshots of the interface of the app being evaluated and highlighting any instances that may relate to accessibility issues. Our accessibility analysis include the following:

1) *ActiveViewName*: Active views must have a name that is available to assistive technologies. For example, a slider should be associated with text describing what the slider is used for. Missing text results in a violation.
2) *ImageViewName*: Meaningful images must have alternate text. Images w/o associated text result in a violation.
3) *TouchSizeWcag*: Touch inputs must have a sufficient target size. The tool checks elements to have a minimum width or height of 44dp; elements smaller than 44dp result in a violation.
4) *EditTextValue*: EditText elements (used to enter text) must expose entered text to assistive technologies. Failing to expose such text results in a violation.
5) *ColorContrast*: Text elements must have sufficient contrast against background. E.g., light gray text on a white background may be a violation. This category is different from the others as it requires operator discretion.

To apply Accessibility Insights to each app, we executed it with the Android Studio emulator analyzing each app of interest. This setup allowed us to test each app's screens for violations of the five rules described above. We tested every screen, after a preliminary analysis on a sample of 15 apps suggested that a non-negligible fraction of apps only exhibit certain categories of errors on specific screens.

After running Accessibility Insights on the app, the tool returns a list of failed instances. A single page of an app can have multiple failed instances for any number of categories. When a previously used asset returns the same failed instance(s), we recorded each failed instance only once, to prevent double counting. We also identified false positives – apparent violations of rules caused by assets that are not visible. Since the developer's intention is clearly for such assets not to be visible, we filtered out these instances. After performing these adjustments, we then recorded any failed instances that the tool returned for all app pages.

All APKs of the IoT mobile companion apps required for the project were downloaded and installed on the emulators in Android studio. For every IoT app, all pages were analyzed, including the landing page and subsequent pages of the app. Some apps display certain interface screens only in the presence of a specific IoT device. We did not evaluate such screens. Thus, our approach may conservatively underestimate the number of accessibility issues for some apps.

[4]https://accessibilityinsights.io/docs/en/android/overview/

B. Results

Since many apps use anti-emulation techniques, or failed to run on the emulator due to bugs, out of the 455 IoT companion apps that we initially collected, we were only able to analyze 248 apps. We considered the trend and usage frequencies of all the companion apps to ascertain we were analyzing commonly used IoT companion apps. The emulators, Nexus (Android 9.0) and Pixel (Android 8.1), were preferred as virtual devices because they seem more stable than other tested virtual devices. After that, Accessibility Insights for Android was launched to analyze and evaluate the various pages of individual IoT apps. It took about 15 – 25 minutes to analyze each app, depending on the number of pages[5].

Analysis of the accessibility metrics reveals that *TouchSizeWcag* (95%) has the highest rate of violations and errors, whereas *EditTextValue* (3.94%) has the least number of violations and errors. Furthermore, the majority of the apps exhibit some errors. There were only two apps that did not have failed accessibility instances across the metrics, one of them has a total of six pages, while the other has 12 pages. In fact, the 248 IoT companion apps produced 5,349 violations over a total of 3964 pages during accessibility analysis. Overall error counts are as follows: *TouchSizeWcag* = 2357, *Contrast* = 1200, *ActiveViewName* = 1322, *ImageViewName* = 515 and *EditTextValue* = 53. Data shows that *TouchSizeWcag* accounted for almost half of the total errors while the errors from *EditTextValue* are almost negligible compared to other metrics.

Table VII summarizes the standard deviation, mean, and range for each of the accessibility metrics. We took into account errors on the landing page, the total number of errors within an app, and the average number of errors per page. In the following, we expand on individual results.

1) Landing Page vs Other Pages: Results show that the number of available pages to be evaluated and analyzed varied based on the app. The page variation was considered because we observed that apps with more pages tend to produce more opportunities for errors. Furthermore, for almost all companion apps, an account must be created on the landing page to access subsequent pages.

2) ActiveViewName: Different UI elements on apps should have a name accessible by assistive technologies for ease of navigation. *ActiveViewName* errors detect UI elements in apps with missing names. These errors represent about 25% (1322) of all errors. It is the second most violated feature across the accessibility metrics considered during this analysis. Furthermore, 86.67% of the apps analyzed presented at least one *ActiveViewName* error throughout its pages, and only 33 apps avoided it. The most *ActiveViewName* errors within the same app was 35 errors in 23 pages, however, this was not the highest rate, as that was 6 errors for a one-page app.

3) ImageViewName: ContextDescription attribute provides text alternatives to explain the meaning of an image to users. So, any meaningful image that does not provide such a description violates accessibility standards. Errors associated with *ImageViewName* were about 9.63% of the total errors. We found 128 companion apps without violation of *ImageViewName* and two apps with the highest number of errors (19 errors). The highest rate of errors was detected for a one-page app (7 errors per page).

[5]We will make our app dataset and detailed results accessible upon request.

TABLE VII: Standard Deviation, Mean, Max and Min for accessibility violations

Category	**STD**	**Mean**	**Max**	**Min**
NumPages	13.1	16.0	68	1
ActiveViewName Landing Page	1.92	0.90	21	0
ActiveViewName Total # of Errors	5.62	5.33	35	0
ImageViewName Landing Page	0.851	0.355	6	0
ImageViewName Total # of Errors	3.23	2.08	19	0
TouchSizeWcag Landing Page	2.67	1.84	20	0
TouchSizeWcag Total # of Errors	9.06	9.50	53	0
EditTextValue Landing Page	0.0635	0.00403	1	0
EditTextValue Total # of Errors	0.799	0.214	7	0
Contrast Landing Page	1.47	0.851	10	0
Contrast Total # of Errors	8.79	4.84	72	0
ActiveViewName Avg Err per Page	0.696	0.461	6.0	0
ImageViewName Avg Err per Page	0.532	0.171	7.0	0
TouchSizeWcag Avg Err per Page	1.22	0.848	10.0	0
EditTextValue Avg Err per Page	0.0753	0.0166	0.667	0
Contrast Avg Err per Page	1.49	0.517	18.0	0

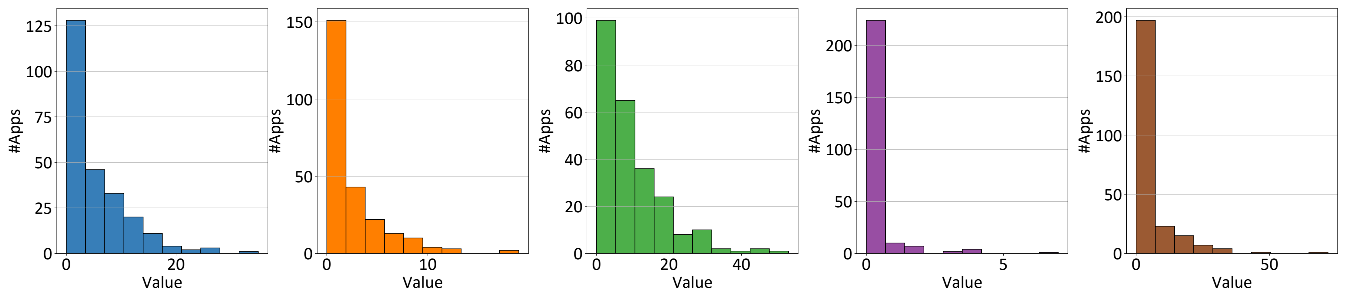
TABLE VIII: r_s Between Accessibility Metrics (AVN: ActiveViewName, IVN: ImageViewName, TSW: TouchSIzeWCAG, ETV: EditTextValue, CC: Contrast)

	AVN	**IVN**	**TSW**	**ETV**	**CC**
AVN	1.00	-	-	-	-
IVN	0.403	1.00	-	-	-
TSW	0.500	0.180	1.00	-	-
ETV	-0.169	-0.120	-.0597	1.00	-
CC	0.206	0.202	0.474	0.019	1.00

4) TouchSizeWcag: Sufficient target size of minimum width and height of 44dp is required for *touch inputs* to pass accessibility criteria determined by the WCAG. *TouchSizeWcag* violations accounted for about 25% of the total. We noticed that almost 95% of the 248 apps had *TouchSizeWcag's* errors on at least one page. At the same time, only 13 apps did not present any such error. One app presented 53 *TouchSizeWcag's* errors with an average of 2.21 per page; the highest rate is 10 errors per page for a one-page app.

5) EditTextValue: If an editable text object is not configured correctly, assistive technology may announce the type of the object rather than its text content. This, in turn, will give users who rely on assistive technology insufficient information. *EditTextValue* was only present in 24 of the apps analyzed. This was the least prominent error, making up only 3.94% of the violations. The highest number of *EditTextValue* errors within one app is 7 errors for a 21-page app; the highest rate is 0.667 errors per page for a 3-page app.

6) ColorContrast: To meet accessibility standards, developers need to consistently implement elements, including text, and images or icons, that have acceptable contrast with their background. This is the third most violated metric with about 22.44% errors of the total. It also produced the highest error within the same app (72 errors) and the highest rate of errors per page within the same app (18 within a 1-page app).



(a) ActiveViewName (b) ImageViewName (c) TouchSizeWcag (d) EditTextValue (e) ColorContrast

Fig. 5: Individual Distributions of Accessibility Errors for the IoT Apps

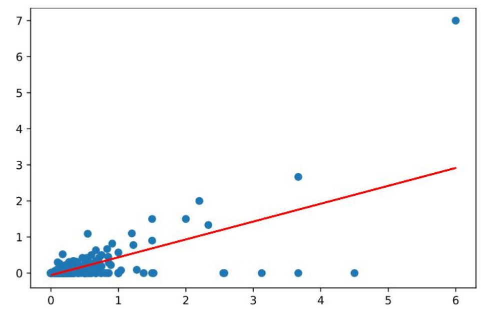


Fig. 6: Linear Regression Plot ($r_{SQ} = 0.42$) of *ActiveViewName* Average Number of Errors Across Pages vs *ImageViewName* Average Number of Errors Across Pages.

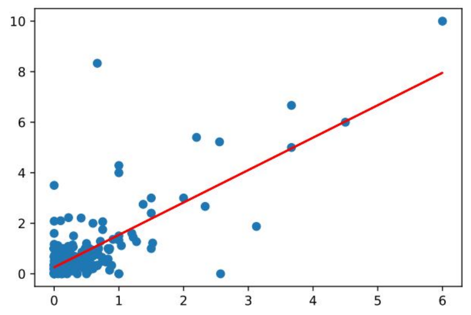


Fig. 7: Linear Regression Plot ($r_{SQ} = 0.54$) of *ActiveViewName* Average Number of Errors Across Pages vs *TouchSizeWCAG* Average Number of Errors Across Pages.

7) Correlation between metrics: A question relevant to our analysis is whether accessibility errors correlate across error categories. To answer this, we use Spearman correlation as the distribution of error per app. When computing the Spearman correlation matrix, we used the average number of errors per page rather than the total number of errors. Results are presented in Table VIII. The most relevant finding is that *ActiveViewName* violations exhibit moderate correlation with *ImageViewName* as well as *TouchSizeWCAG*. This finding is further confirmed by regression plots, which are displayed in Figures 6 and 7. This analysis also evidenced some outliers, most notably of which a one-page app produced the maximum average of errors for all the categories of errors we tested except *EditTextValue*.

8) Take-aways: While the average number of errors per category is low, there is a long tail of apps with a significant amount of issues (57 with two or more accessibility errors per page). Furthermore, some issues entailing a lack of accessible descriptions are weakly correlated across UI element types.

VI. Cross-Dimension Analysis

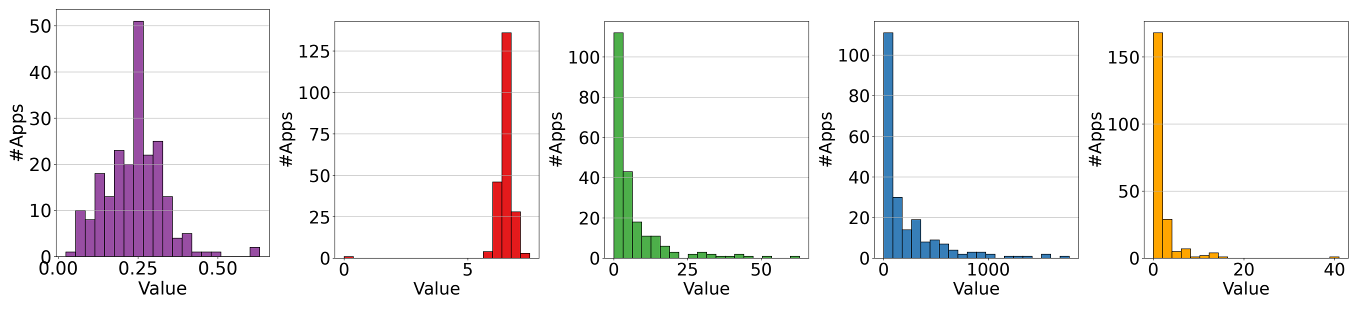
We now assess the connections between the three dimensions when analyzing app characteristics. We used data for 247 apps for which all metrics were available. For security we considered CVSS score, Flowdroid leaks and Days since last update. For privacy and accessibility, we generated an overall score for the category as the arithmetic mean of the individual metrics. The value distribution for each metric define above is shown in Figure 8. The metrics in Figure 8 may resemble a normal distribution, however none of them passes the Shapiro-Wilk test for normality (*Privacy:* $W = 0.95, p < 0.01$; *CVSS score:* $W = 0.43, p < 0.01$, *Flowdroid leaks:* $W = 0.66, p < 0.01$; *Days since last update:* $W = 0.69, p < 0.01$; *Accessibility:* $W = 0.96, p < 0.01$).

First, pairwise Spearman correlation shows that there is no significant correlation (i.e., a correlation above 0.3) between any pair of metrics. However, metric-wise relationships may exist for specific subsets of apps. We investigate this by slicing our app dataset across two taxonomies, based on number of app downloads and review scores. We chose these because they can be considered as proxy for popularity and quality of user experience, and they are uncorrelated in our dataset.

A. Cross-dimension analysis by number of downloads

Android app metadata categorically characterize number of downloads: for each app, they specify a broad range. Our taxonomy groups multiple such ranges to define five categories: *1,000-10,000 downloads*, *10,000-100,000*, *100,000-1,000,000*, *1,000,000-10,000,000*, and *above 10,000,000*. Figure 9a depicts the distribution of apps by number of downloads.

First, we conducted pairwise Mann-Whitney U tests (adjusted with Bonferroni correction) to investigate whether any metric exhibits statistically significant differences across apps in different bins. We only found a statistically significant difference between the *1,000-10,000* and *10,000-100,000* groups in regards to the *Days since last update* metric ($U = 539, Z = 3.05, p < 0.05$). Overall, based on these observations we



(a) Privacy (b) CVSS (c) Flowdroid Leaks (d) Days since last update (e) Accessibility

Fig. 8: Distributions of metrics used for cross-dimension analysis

conclude that the number of downloads does not significantly affect the value of any measured metrics.

Second, we computed Spearman correlation between all pairs of metrics, for all possible subsets of apps grouped by number of downloads. Within each pair of metrics, we again apply Bonferroni correction. We find a weak positive correlation between FlowDroid leaks and Average privacy score for the *10,000-100,000 downloads* app group ($r_s = 0.325, p < 0.05$). We also found a weak correlation between Days since last update and Average privacy score for the *1,000,000-10,000,000* app group ($r_s = -0.469, p < 0.05$). We found no other statistically significant correlation. These results suggest that no relevant pattern emerges by grouping the apps in our dataset by number of downloads.

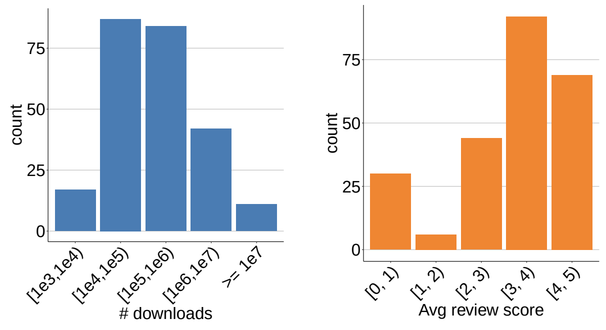
B. Cross-dimension analysis by review scores

Android app metadata also include information about the scores of the reviews by users of each app. Such scores are specified as integer between 0 and 5. We bin app review scores into five categories, each including reviews in the interval $[x, x + 1)$ for x between 0 and 4. Figure 9b depicts the distribution of apps by review score.

We conducted pairwise Mann-Whitney U tests, and we did not find any significant difference, for any metric, across apps in different review score bins. Further, we computed Spearman correlation between all pairs of metrics, for apps in each review score bin. We identify a weak negative correlation between accessibility score and CVSS issue score, for app in the $[0, 1)$ review score category ($r_s = -0.363, p < 0.05$). We interpret these results as showing that categorizing the apps by review score does not identify any significant pattern of correlation between metrics.

C. Take-aways

We now consider the research questions from Section I: *Are software quality issues concentrated along a single dimension, or do they tend to propagate along multiple dimensions?* and *Is there evidence of tradeoffs where high quality along a certain dimension correlates with low quality along another one?* Based on results, **software quality issues do not appear to propagate along multiple dimensions**: the posture of each app in one dimension is uncorrelated to other dimensions. For the same reason, **we did not find evidence of tradeoffs between app quality in different dimensions**.



(a) Distribution by #dl (b) Distribution by avg score

Fig. 9: Distribution of apps by downloads and avg score

We believe this lack of correlation is a useful result, as it suggests that issues along different dimensions do not stem from a common underlying factor, but rather from dimension-specific factors. Thus, we recommend that future work on software defect analysis, prediction and remediation within IoT companion apps, focuses on defects within individual dimensions. The same lack of correlation also suggests that it may be beneficial to involve specialized professionals, such as privacy or accessibility experts, in the design and development of IoT companion apps.

VII. Discussion

A. Limitation of Data Collection

One of the main challenges in our data collection approach was the difficulty of applying a rigorous definition of "IoT companion apps". Our approach of seeding the search with an initial set of manually curated apps, and following "related apps" suggestions by the Play Store has the advantage of quickly presenting a large set of apps to the scraper. On the other hand, it introduces significant noise, which we mitigated through automated and manual filtering. While manual filtering can in turn introduce experimenter bias, further independent review of the collected app data set provides reassurance that the large majority of samples is indeed relevant to our definition of companion app.

A possible concern is long-term repeatability, since data collection requires substantial manual effort. Since our initial collection, new text classification techniques using on transformer models have been proposed, that promise to automate manual analysis while maintaining accuracy. To evaluate whether these

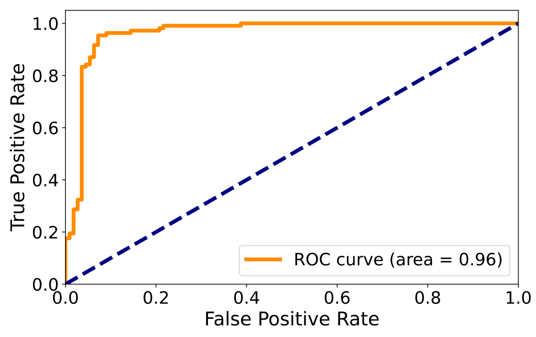


Fig. 10: ROC plot for transformer-based IoT app classifier

models can truly automate future app collection efforts, we conducted a proof-of-concept experiment. We created a dataset including 434 of our pre-vetted IoT app descriptions, and 446 non-IoT app descriptions. We collect the latter from the AndroZoo app collection [51], randomly sampling from a set of non-IoT app categories (Adventure, Lifestyle, Education, Productivity, and Food & Drink). Following best practices from literature [52], we fine-tune the Microsoft DeBERTa model [53] (chosen due to its strong text classification performance) on a subset of our dataset, evaluating classification on rest of the samples[6] via cross-validation. This approach results in a mean F1 score of 0.948 ± 0.009. Figure 10 depicts the ROC curve for a sample experiment. Overall, our results show the fine-tuned model can identify IoT apps with fairly high precision, suggesting that this approach can replace manual classification in future measurements.

A more general external validity issue stems from the fact that our measurement effort exclusively focuses on the Android ecosystem, ignoring Apple iOS. This is due to practical limitations and a dearth of tools for analysis in the iOS context, due to the fact that this ecosystem is significantly more closed than Android. We still believe that our results have a wide applicability due to (i) Android maintaining over 70% of the mobile market share worldwide [54]; and (ii) many IoT device manufacturers providing similar apps across ecosystems.

B. State of Measuring Tools

Another challenge we encountered pertains to the state of app analysis tools for the Android ecosystem. Buggy tools may cause construct validity issues and false positives/negatives; therefore we took particular care in reviewing each tool we used to ensure its viability for our measurements. Table IX summarizes the tools we evaluated for security and accessibility analyses (privacy is based on metadata, namely permissions). This table does not include the approaches we excluded because of missing or outdated code (ref Section IV-A).

a) Privacy Analysis: We first contemplated evaluating the privacy of the apps based on the permissions used by the apps, rather than the permissions declared in the manifest. However, this requires dynamic analysis which complicates obtaining an exhaustive set of permissions. Thus, we decided in favor of extracting requested permissions from app manifests. While app manifests are a direct expression of developer intentions, in practice an app may not always use all permissions listed in the manifest. Thus, we plan to extend this approach in future work.

TABLE IX: Tools evaluated for our analyses ("Accessibility Setup" refers to the Accessibility Insight+Android Studio setup used to evaluate accessibility violations)

Tool	Used?	Findings
MobSF	✓	Works on all tested apps
FlowDroid [55]	✓	Works on all tested apps
CryptoGuard [40]	✗	Perf. issues; fails to analyze some apps
CryLogger [56]	✗	64% failure rate; req. manual effort
Accessibility Setup	✓	45% failure rate; req. manual effort

b) Security Analysis: MobSF and FlowDroid are mature and well-maintained, and we found them easy to integrate into an automated analysis workflow. Both are open-source software, and backed by robust developer communities.

We also considered using CryptoGuard and CryLogger to detect issues related to incorrect use of encryption. Both are recent academic tools, but we found them to be less applicable to a broad set of mobile apps. Our CryptoGuard analysis setup run four containerized CryptoGuard instances (120GB memory, 7 Xeon 5218 cores each); the setup successfully processed 37 apps over 4 days, before all analysis threads hung; prior to that we also observed a number of crashes and out-of-memory exceptions. CryLogger also exhibited a high failure rate (64% on a test set of 50 apps), and we found it difficult to integrate it into an automated workflow due to the dynamic nature of its analysis. Thus, we decided not to use either tool into. We emphasize that these issues are to be expected in early-stage research tools. As these tools can provide high-quality data on misuse of cryptographic primitives, we believe the community would benefit significantly if both tools were to transition to a sustainable, community-supported development lifecycle.

c) Accessibility Analysis: Our accessibility setup, consisting of Accessibility Insight [57] and Android Studio failed on a significant number of apps, due to app crashes, incompatibility with emulators, apps refusing to produce meaningful output in absence of the companion device. These issues caused a significant reduction in the size of our dataset, as we dropped any app that could not undergo all three types of analyses. However, we decided to retain this approach, as we are not aware of any other technique that would enable semi-automated accessibility analysis based on rigorous guidelines.

Further, it is possible that app behavior within an emulated environment may differ from the same on an actual phone. However, 4 out of 5 of our accessibility metrics (everything except *ColorContrast*) are based on precisely measurable aspects (such as device-independent UI element size or whether content made available to assistive technologies) that are only dependent on an app's design, and not the context in which its run. Thus, we believe emulated behavior, however imperfect, to be sufficiently faithful to real-world behavior to make this approach acceptable.

[6]We fine-tune for 3 epochs, using the AdaFactor optimizer, a batch of size 8, and a learning rate of 1e-5.

C. Relevance of Analysis of IoT Companion Apps

A relevant question is whether it makes sense to measure IoT-related mobile apps separately from the rest. Indeed, with the exception of the Naïve Bayes classifier used in our app collection pipeline (ref. Section II), the analysis techniques used here are largely non-IoT specific. What makes this class of apps unique, however, is their ability to interface to IoT devices, which enable them to collect intimate information about their users (e.g. health data, private video feeds), and to directly affect quality of life. Thus, we believe analyzing this class of apps in isolation is an effective way to understand important emerging issues in the IoT ecosystem. This approach is in line with recent security analyses focusing specifically on the mobile IoT ecosystem (e.g., [52]). Overall, our results suggest that, while most apps have reasonable posture in regards to security, privacy and accessibility, there remains a long tail of apps of significant problems. These issues can directly affect users in terms of private data leaks, and reduced access to important app functionality.

D. Ethical Considerations

Most of the issues identified in this paper fall under the purview of design decisions by developers. While these have significant negative impact on users - in terms of exposing users to security risks, privacy leaks, and poor quality of experience - they are not vulnerabilities per se, and thus we chose not to report it. We consider the cleartext sensitive information disclosure in the *XVRView* and *Vss Mobile* app as meeting the bar to be considered security issues, and reported them to the app maintainers (we gave them the customary 90-day period before disclosing these issue in the paper). However, maintainers for these apps never replied to us.

VIII. Related Work

A. Metrics and Evaluation Parameters

1) Security and Privacy Metrics: Jansen [58] critiques the subjective human judgment that often skews security metrics. Krutz et al. also found a weak correlation between user ratings and actual security [59]. Our approach primarily focuses on metrics with objective properties, such as the CVSS issue scores, to circumvent these limitations. WHYPER [60] flags unexplained permissions in app descriptions but does not necessarily identify these as security risks. Kang et al. [36] suggest a privacy meter that relies on a pre-defined list of dangerous permissions. However, their method does not consider the functional necessity of certain permissions for an app. Alternatively, Biswas et al. [61] and Peng et al. [62] use ML and probabilistic generative models to analyze permissions. Based on the superiority of ML-driven methods [49], our work incorporates and extends these approaches.

2) Accessibility Metrics: Accessibility metrics have received far less attention than security and privacy metrics. Abou-Zahra et al. highlight the importance of accessibility in the Web of Things (WoT) [63]. Their work sheds light on the challenges faced by people with disabilities in the context of IoT platforms. Sohaib et al. further discuss the potential of IoT technologies to enhance e-commerce experiences for disabled individuals [64]. Our research builds on the work of De Oliveira et al. [65], who analyzed 11 specific accessibility features in six IoT apps. We also draw parallels with Ross et al. [66], albeit our analysis differs in its focus and dataset. Specifically, we employ a more recent dataset than the Rico screenshot dataset used by Ross et al. [67], enabling us to address gaps in current research.

B. IoT-Based Studies

Liu et al. [8] introduced the concept of *app-in-the-middle* IoT setups, delineating the evolving paradigm where limited-resource IoT devices are dependent on mobile applications for connectivity. Similarly, Ding and Hu [68] put forward a framework to discern security-critical hidden integrations among Samsung SmartThing apps, informed by physical-world interactions. In a policy-oriented discourse, Streiff et al. [13], [69], [70] evaluated existing legislative landscapes surrounding IoT devices. These works serve as an impetus for our exploration of safety metrics in IoT companion apps.

A more expansive empirical analysis was conducted by Jin et al. [52], over a dataset of $37,783$ IoT mobile apps from the Google Play Store. Our study diverges considerably in its scope and approach. Jin et al. focused on a broader set of apps, but confined their study to the domain of security with a specific emphasis on static analysis. Conversely, our research adopts a more holistic approach, by analyzing a smaller set of apps under the lenses of not just security, but also privacy (including permissions, data leaks, and privacy policy, which Jin et al. do not cover), and accessibility metrics. Further, our security analysis is also more comprehensive, as it embraces both static and dynamic evaluations.

Complementary to our efforts, Wang et al. [24] executed a large-scale analysis to extrapolate device and backend vulnerabilities from mobile companion apps. On a more confined scale, Junior et al. [71] carried out similar endeavors. However, both studies predominantly spotlight device-centric vulnerabilities without delving into app-specific security issues, and neither provides any insight into accessibility concerns. Fernandes et al. [72] targeted smart home apps within the Samsung Smart Things ecosystem, identifying issues rooted in over-privileging. This is conceptually distinct from our study, which emphasizes a comprehensive evaluation of mobile IoT companion apps. Mohanty and Sridhar reviewed the security features of 102 IoT companion apps [73], albeit with a constrained focus on predetermined security issues within apps developed using hybrid frameworks. Our study has a more expansive scope, both in terms of the variety of apps reviewed and the range of issues considered.

IX. Conclusions

In this work, we executed a comprehensive analysis on 455 IoT mobile companion applications. We evaluated each based on carefully-selected metrics for privacy, security, and accessibility. Our findings reveal a mixed landscape: while the general level of adherence to best practices in these domains is somewhat reassuring, we also unearthed systemic weaknesses.

Many apps exhibited known security issues, over-requested permissions, and lapses in accessibility - only two apps in our corpus conformed to the WCAG guidelines. Our study sets the groundwork for future investigations and provides actionable insights into robust quality metrics and design guidelines. Thus, it promotes the creation of IoT applications that are secure, privacy-preserving, and accessible.

X. Acknowledgment

We would first like to thank Upakar Paudel, Louis Narmour, Sai Donepudi, Harrison D. Whittaker, Amanuel Bulba, Hennick Yibra, Freddy Veloz Baez, Merzia Adamjee, and Griffin Opp for their help with the analysis. We also wish to acknowledge the InSPIRIT lab at the University of Denver.

References

[1] D. Holloway and L. Green, "The Internet of toys," *Communication Research and Practice*, vol. 2, no. 4, pp. 506–519, Oct. 2016.

[2] L. Catarinucci, D. de Donno, L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, and L. Tarricone, "An IoT-Aware Architecture for Smart Healthcare Systems," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 515–526, Dec. 2015.

[3] J. Marquez, J. Villanueva, Z. Solarte, and A. Garcia, "IoT in Education: Integration of Objects with Virtual Academic Communities," in *New Advances in Information Systems and Technologies*. Cham: Springer International Publishing, 2016, vol. 444, pp. 201–212.

[4] S. Vashi, J. Ram, J. Modi, S. Verma, and C. Prakash, "Internet of things (iot): A vision, architectural elements, and security issues," in *IEEE I-SMAC*, 2017.

[5] B. Debnath, S. Das, A. Das, and A. Das, "Smart Devices for Smart Cities in India Exploring Security Vulnerabilities of E-Waste Management," Aug. 2019.

[6] S. Das, "Eyes in Your Child's Bedroom: Exploiting Child Data Risks with Smart Toys," USENIX Enigma, 2020.

[7] S. N. Matheu, J. L. Hernández-Ramos, A. F. Skarmeta, and G. Baldini, "A Survey of Cybersecurity Certification for the Internet of Things," *ACM Computing Surveys*, vol. 53, no. 6, pp. 1–36, Feb. 2021.

[8] H. Liu, J. Li, and D. Gu, "Understanding the security of app-in-the-middle IoT," *Computers & Security*, vol. 97, p. 102000, Oct. 2020.

[9] M. Maiman, "How the User Interface on IoT Hardware Can Make or Break the User's Experience," Oct. 2019. [Online]. Available: /technologies/iot/article/21808679/how-the-user-interface-on-iot-hardware-can-make-or-break-the-users-\\experience

[10] O. Toutsop, S. Das, and K. Kornegay, "Exploring the security issues in home-based iot devices through denial of service attacks," in *IEEE SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, 2021.

[11] S. Gopavaram, J. Dev, S. Das, and L. J. Camp, "Iot marketplace: Willingness-to-pay vs. willingness-to-accept," in *WEIS*, 2021.

[12] S. Majumder, E. Aghayi, M. Noferesti, H. Memarzadeh-Tehran, T. Mondal, Z. Pang, and M. J. Deen, "Smart Homes for Elderly Healthcare—Recent Advances and Research Challenges," *Sensors*, vol. 17, no. 11, Nov. 2017.

[13] J. Streiff, S. Das, and J. Cannon, "Overpowered and underprotected toys empowering parents with tools to protect their children," in *IEEE CIC*, 2019.

[14] F. Tazi, S. Shrestha, and S. Das, "Cybersecurity, safety, & privacy concerns of student support structure for information and communication technologies in online education," *Proceedings of the ACM on Human-Computer Interaction*, vol. 7, no. CSCW2, pp. 1–40, 2023.

[15] S. Das, R. S. Gutzwiller, R. D. Roscoe, P. Rajivan, Y. Wang, L. Jean Camp, and R. Hoyle, "Humans and technology for inclusive privacy and security," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 64, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2020, pp. 461–464.

[16] S. Saka and S. Das, "Evaluating privacy measures in healthcare apps predominantly used by older adults," 2024. [Online]. Available: https://arxiv.org/abs/2410.14607

[17] U. Kishnani, N. Noah, S. Das, and R. Dewri, "Assessing security, privacy, user interaction, and accessibility features in popular e-payment applications," in *EuroUSEC*, 2023.

[18] F. Tazi, J. Dykstra, P. Rajivan, K. Chalil Madathil, J. Hughart, J. McElligott, D. Votipka, and S. Das, "Improving privacy and security of telehealth," *Communications of the ACM*, vol. 67, no. 9, pp. 27–30, 2024.

[19] S. R. Gopavaram, J. Dev, S. Das, and J. Camp, "Iotmarketplace: Informing purchase decisions with risk communication," Working Paper, 2019, ftp://svn. soic. indiana. edu/pub/techreports/TR742. pdf, Tech. Rep., 2019.

[20] finjan mobile, "Apps are Creating Mobile Security Vulnerabilities for IoT - How Bad is It?" https://www.finjanmobile.com/mobile-security-vulnerabilities-for-iot/, Feb. 2017.

[21] B. of Internet Accessibility, "Accessibility and the Internet of Things," https://www.boia.org/blog/accessibility-and-the-internet-of-things, Dec. 2018.

[22] S. Neupane, F. Tazi, U. Paudel, F. V. Baez, M. Adamjee, L. De Carli, S. Das, and I. Ray, "On the data privacy, security, and risk postures of iot mobile companion apps," in *IFIP DBSec*, 2022.

[23] F. Tazi, S. Saka, G. Opp, S. Neupane, S. Das, L. De Carli, and I. Ray, "Accessibility evaluation of iot android mobile companion apps," in *CHI Extended Abstract*, 2023.

[24] X. Wang, Y. Sun, S. Nanda, and X. Wang, "Looking from the mirror: Evaluating IoT device security through mobile companion apps," in *USENIX Security*, 2019.

[25] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[26] D. E. Hinkle, W. Wiersma, and S. G. Jurs, "Applied statistics for the behavioral sciences (vol. 663)," 2003.

[27] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *SOUPS Symposium*, 2012.

[28] M. Tahaei and K. Vaniea, ""developers are responsible": What ad networks tell developers about privacy," in *Extended Abstracts at ACM CHI*, 2021.

[29] R. Balebako, A. Marsh, J. Lin, J. I. Hong, and L. F. Cranor, "The privacy and security behaviors of smartphone app developers.(2014)," *DOI: http://dx. doi. org/10.1184*, vol. 1, p. 0, 2014.

[30] R. Baalous and R. Poet, "How dangerous permissions are described in android apps' privacy policies?" in *SIN*, 2018.

[31] S. S. Wader, "How android application permissions impact user's data privacy?" *International Journal of Research Publication and Reviews*, vol. 2, no. 3, pp. 498–502, 2021.

[32] S. Tandel and A. Jamadar, "Impact of progressive web apps on web app development," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 7, no. 9, pp. 9439–9444, 2018.

[33] A. Mylonas, M. Theoharidou, and D. Gritzalis, "Assessing privacy risks in android: A user-centric approach," in *RISK*, 2013.

[34] N. Momen, M. Hatamian, and L. Fritsch, "Did app privacy improve after the gdpr?" *IEEE Security & Privacy*, vol. 17, no. 6, pp. 10–20, 2019.

[35] A. K. Jha, S. Lee, and W. J. Lee, "Developer mistakes in writing android manifests: An empirical study of configuration errors," in *IEEE/ACM MSR*, 2017.

[36] J. Kang, H. Kim, Y. G. Cheong, and J. H. Huh, "Visualizing privacy risks of mobile applications through a privacy meter," in *ISPEC*, 2015.

[37] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, and K. Aberer, "Polisis: Automated analysis and presentation of privacy policies using deep learning," in *USENIX Security*, 2018.

[38] S. Wilson, F. Schaub, A. A. Dara, F. Liu, S. Cherivirala, P. G. Leon, M. S. Andersen, S. Zimmeck, K. M. Sathyendra, N. C. Russell *et al.*, "The creation and analysis of a website privacy policy corpus," in *ACL*, 2016.

[39] G. M. Kapitsaki and M. Ioannou, "Examining the privacy vulnerability level of android applications," in *WEBIST*, 2019.

[40] S. Rahaman, Y. Xiao, S. Afrose, F. Shaon, K. Tian, M. Frantz, D. Yao, and M. Kantarcioglu, "Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects," in *ACM CCS*, 2019.

[41] A. Mathur, N. Malkin, M. Harbach, E. Peer, and S. Egelman, "Quantifying users' beliefs about software updates," in *IS USEC*, 2018.

[42] V. P. Ranganath and J. Mitra, "Are free android app security analysis tools effective in detecting known vulnerabilities?" *Empirical Software Engineering*, vol. 25, pp. 178 – 219, 2018.

[43] G. LaMalva and S. E. Schmeelk, "Mobsf: Mobile health care android applications through the lens of open source static analysis," in *IEEE MIT URTC*, 2020.

[44] R. B. Joseph, M. F. Zibran, and F. Z. Eishita, "Choosing the weapon: A comparative study of security analyzers for android applications," in *IEEE/ACIS SERA*, 2021.
[45] CVSS, "Common vulnerability scoring system version 3.1: Specification document," https://www.first.org/cvss/specification-document, 2019.
[46] C. W. Enumeration, "Cwe list version 4.6," https://cwe.mitre.org/data/index.html, 2021.
[47] R. English and C. M. Schweik, "Identifying success and tragedy of floss commons: A preliminary classification of sourceforge.net projects," in *IEEE FLOSS ICSE Workshops*, 2007.
[48] M. Yermakov, "Understanding the android cleartexttrafficpermitted flag," https://appsec-labs.com/portal/understanding-the-android-cleartexttrafficpermitted-flag/, 2020.
[49] A. Merlo and G. C. Georgiu, "RiskInDroid: Machine Learning-Based Risk Analysis on Android," in *IFIP SEC*, 2017.
[50] Andrew Kirkpatrick, Joshue O Connor, Alastair Campbell, and Michael Cooper, "Web Content Accessibility Guidelines (WCAG) 2.1," Jun. 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/
[51] M. Alecci, P. J. Ruiz Jiménez, K. Allix, T. F. Bissyandé, and J. Klein, "Androzoo: A retrospective with a glimpse into the future," in *IEEE/ACM MSR*, 2024.
[52] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni, "Understanding iot security from a market-scale perspective," in *ACM CCS*, 2022.
[53] P. He, J. Gao, and W. Chen, "DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing," in *ICLR*, 2023.
[54] [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/
[55] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *PLDI*, vol. 49, 2014.
[56] L. Piccolboni, G. Di Guglielmo, L. Carloni, and S. Sethumadhavan, "Crylogger: Detecting crypto misuses dynamically," in *IEEE S&P*, 2021.
[57] R. Nacheva, K. Vorobyeva, and M. Bakaev, "Evaluation and promotion of m-learning accessibility for smart education development," in *EGOSE*, 2020.
[58] W. Jansen, "Research Directions in Security Metrics," NIST, Tech. Rep. 7564, 2009.
[59] D. E. Krutz, N. Munaiah, A. Meneely, and S. A. Malachowsky, "Examining the relationship between security metrics and user ratings of mobile apps: A case study," in *WAMA*, 2016.
[60] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," in *USENIX Security*, 2013.
[61] D. Biswas, I. Aad, and G. P. Perrucci, "Privacy panel: Usable and quantifiable mobile privacy," in *ARES*, 2013.
[62] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of android apps," in *ACM CCS*, 2012.
[63] S. Abou-Zahra, J. Brewer, and M. Cooper, "Web standards to enable an accessible and inclusive internet of things (iot)," in *W4A*, 2017.
[64] O. Sohaib, H. Lu, and W. Hussain, "Internet of things (iot) in e-commerce: For people with disabilities," in *IEEE ICIEA*, 2017.
[65] G. A. A. de Oliveira, R. W. de Bettio, and A. P. Freire, "Accessibility of the smart home for users with visual disabilities: An evaluation of open source mobile applications for home automation," in *IHC*, 2016.
[66] A. S. Ross, X. Zhang, J. Fogarty, and J. O. Wobbrock, "Examining image-based button labeling for accessibility in android apps through large-scale analysis," in *ACM ASSETS*, 2018.
[67] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *UIST*, 2017.
[68] W. Ding and H. Hu, "On the Safety of IoT Device Physical Interaction Control," in *ACM CCS*, 2018.
[69] J. Streiff, N. Noah, and S. Das, "A call for a new privacy & security regime for iot smart toys," in *IEEE DSC*, 2022.
[70] J. Streiff, O. Kenny, S. Das, A. Leeth, and L. J. Camp, "Who's watching your child? exploring home security risks with smart toy bears," in *IEEE/ACM IoTDI*, 2018.
[71] D. Mauro Junior, L. Melo, H. Lu, M. d'Amorim, and A. Prakash, "A Study of Vulnerability Analysis of Popular Smart Devices Through Their Companion Apps," in *IEEE SPW*, 2019.
[72] E. Fernandes, J. Jung, and A. Prakash, "Security Analysis of Emerging Smart Home Applications," in *IEEE S&P*, 2016.
[73] A. Mohanty and M. Sridhar, "HybriDiagnostics: Evaluating Security Issues in Hybrid SmartHome Companion Apps," in *IEEE SPW*, 2021.

Faiza Tazi is a PhD candidate in the Department of Computer Science in the Ritchie School of Engineering and Computer Science at University of Denver. She is a researcher affiliated with the Inclusive Security and Privacy-focused Innovative Research in Information Technology (InSPIRIT) lab, and is focused on the domains of privacy, security, usability and human-computer interaction, with a particular emphasis on healthcare

Saka Suleiman is a PhD student in Computer Science at the University of Denver, CO, USA. Previously, he obtained a Master's degree in Computer Network Security from the University of Greenwhich, London, UK. His research interests are in Usable Security, including Mobile Security and Web Security.

Shradha Neupane is a Solution Architect at Dell Technologies, MA, USA. Previously, she obtained an M.S. in Computer Science from Worcester Polytechnic Institute, MA, USA. Her research interests are in Software Security, including Mobile Security and Supply Chain Security.

Ethan Myers is an M.S. student in CS at Colorado State University, Fort Collins, CO, USA. His research interests include NLP and cybersecurity. His current research is focused on measuring privacy threats to mobile health applications to evaluate their compliance with privacy regulations.

Sanchari Das is an Assistant Professor in the Information Sciences and Technology department at the George Mason University. Her research lab (InSPIRIT) focuses on computer security, privacy, human-computer interaction, social computing, accessibility, and sustainability of new-age technologies. She received her Ph.D. from Indiana University Bloomington and her dissertation focused on understanding users' risk mental models to help in secure decision-making for authentication technologies.

Lorenzo De Carli received a M.S. (2010) and a Ph.D (2016) in Computer Science from the University of Wisconsin-Madison. He is an Associate Professor of Electrical and Computer Engineering at the University of Calgary. He is generally interested in the challenges that arise when attempting to design network and software systems which are secure and usable. His interests cover IoT and residential network security, Software security, and Usable security.

Indrakshi Ray is a Professor in the Computer Science Department at Colorado State University. She obtained her Ph.D. from George Mason University under the supervision of Sushil Jajodia and Paul Ammann. Her Master's degree in Computer Science and Engineering is from Jadavpur University, Kolkata, India. Her Bachelor of Engineering degree in Computer Science and Technology is from B.E.College, Kolkata, India. Dr. Ray's research interests include security and privacy, database systems, e-commerce and formal methods in software engineering.