On the Data Privacy, Security, and Risk Postures of IoT Mobile Companion Apps*

Shradha Neupane[1], Faiza Tazi[2], Upakar Paudel[3], Freddy Veloz Baez[1], Merzia Adamjee[1], Lorenzo De Carli[1], Sanchari Das[2], and Indrakshi Ray[3]

[1] Worcester Polytechnic Institute, Worcester MA 01609, USA
{sneupane, fevelozbaez, madamjee, ldecarli}@wpi.edu
[2] University of Denver, Denver CO 80208, USA
{Faiza.Tazi, Sanchari.Das}@du.edu
[3] Colorado State University, Fort Collins CO 80523
{Upakar.Paudel, Indrakshi.Ray}@colostate.edu

Abstract. Most Internet of Things (IoT) devices provide access through mobile companion apps to configure, update, and control the devices. In many cases, these apps handle all user data moving in and out of devices and cloud endpoints. Thus, they constitute a critical component in the IoT ecosystem from a privacy standpoint, but they have historically been understudied. In this paper, we perform a latitudinal study and analysis of a sample of 455 IoT companion apps to understand their privacy posture using various methods and evaluate whether apps follow best practices. Specifically, we focus on three aspects: data privacy, security, and risk. Our findings indicate: (i) apps may over-request permissions, particularly for tasks that are not related to their functioning; and (ii) there is widespread use of programming and configuration practices which may reduce security, with the concerning extreme of two apps transmitting credentials in unencrypted form.

Keywords: IoT security · IoT privacy · Mobile security

1 Introduction

The Internet of Things (IoT) has become an integral part of our everyday lives, for convenience, entertainment [32], health monitoring [23], online education [46], and other daily activities [62]. However, with increasing interactions of IoT devices with users through their companion mobile apps, inadequate privacy and security has caused several breaches [24, 15]. Prior research on the privacy and security of IoT devices mostly focuses on the devices themselves. Thus, it leaves out a crucial component: the mobile apps through which users configure and control IoT devices. Such IoT companion apps have visibility on the information flowing through devices (e.g., camera streams, network details) [41, 58], and retain their potentially sensitive operations (e.g., smart locks, children toys) [16,

* Shradha Neupane and Faiza Tazi contributed equally as first authors. This work was supported in part by funding from NSF under Award Number CNS 1822118, NIST, ARL, Statnett, AMI, Cyber Risk Research, NewPush, State of Colorado Cybersecurity Center, and a gift from Google.

59]. Thus, from a privacy, security, and risk standpoint they are a critical gateway; programming and design bugs, and deliberate leaking of personal data, can have a critical negative impact on users.

Our goal is to generate an understanding of privacy-related behavior and issues in IoT companion apps. We explicitly design our effort as a latitudinal study, i.e., we aim at carrying a selected number of privacy-related analyses on a large number of apps. We limit ourselves to analyzing companion apps in isolation, because analyzing each app as it interacts with its specific IoT device is not practical for a large scale study. We collect apps (N=**455**) from the Google Play Store [7] using a hybrid method that combines automatic scraping with manual review. We inspect the collected apps using a number of relevant analyses leveraging, where possible, existing industry-standard tools. The tools measure app posture in terms of three different aspects: data privacy, security (e.g. absence of privacy-sensitive vulnerability), and risk (i.e., app trustworthiness). The data privacy and security analyses focus on issues that can be specifically identified; whereas the risk analysis attempts to estimate the potential for latent issues that cannot yet be discovered.

We aim to answer the following research questions with regards to IoT apps: *RQ1: How do companion apps fare with respect to the measured security- and privacy-related aspects? Are there significant general concerns broadly affecting companion apps?* Furthermore, the use of heterogeneous analyses covering multiple app aspects also allows us to answer an additional important empirical question: *RQ2: Is there evidence of positive or negative correlation between metrics measuring different aspects of app privacy (e.g., sensitive permissions and presence of vulnerabilities)?*

Our analysis finds that many apps over-request permissions that appear to be unrelated to their core functionality. Moreover, there are many apps which deploy insecure programming or configuration practices, with the extreme of two apps transmitting user credentials in cleartext. **Our work makes the following contributions:** (i) We propose a *threat model* informed by domain expertise, and derive broad analysis categories from the model. (ii) We propose a method to identify and scrape IoT companion apps from the Google Play Store. (iii) We define methods to quantitatively and reproducibly assess aspects of companion app privacy along the dimension of data privacy, security, and risk as it correlates with the privacy and security of the IoT device itself. (iv) We perform a large-scale measurement study of the properties[4]. (v) We investigate whether privacy aspects correlate among multiple measured dimensions.

2 Threat Model

We adopt a holistic definition of IoT apps: *An IoT device is one which senses/actuates the physical world and is able to exchange sensing/actuation data with another networked node* [47, 33]. In practice, this style of definition is typically and implicitly extended to exclude non-embedded computing devices such as laptops

[4] Our raw metrics are anonymously available at `https://osf.io/gf7cs/?view_only=c701039702f648849e32ecd4c2e1fd54`.

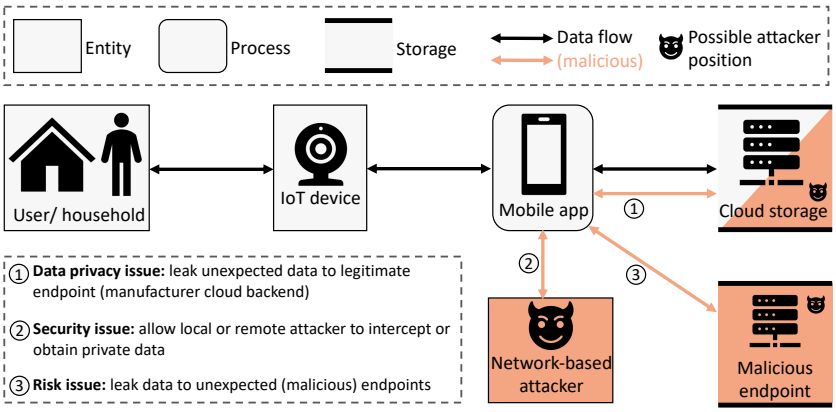


Fig. 1: IoT ecosystem and threat model of IoT companion apps

and smartphones; we follow the same approach in our work. We define a companion app as follows: *a companion app is an app which connects to an IoT device over a network, to rely actuation commands or receive sensing data.*

Our ecosystem consists of a user, an IoT device, a companion IoT app, and the cloud backend. Our model is adapted from the LINNDUN threat modeling system [66]: in Figure 1 the system is depicted as a dataflow diagram, where nodes can be *entities*, *processes*, and *data stores*. Flows of sensitive user data are represented as arrows. We also represent attacker-related entities. We make some simplifying assumptions. First, we define as "sensitive user data" any data existing on the user phone, for which the user has a reasonable expectation of privacy. Second, we assume the user is in agreement with IoT device data being transmitted/received to/from the IoT device and a cloud backend managed by the device manufacture that is needed for its core functionality. Furthermore, we define as *privacy leak* any companion app-mediated unwanted access to sensitive user data, where "unwanted" may mean: (i) the app accesses/transmits data which are not necessary to carry out its function *(data privacy issue)*; (ii) app is vulnerable/misconfigured allowing an attacker to access data *(security issue)*; (iii) app transmit data to unexpected endpoints and/or perform malicious actions (*risk issue*).

3 Data Collection and Analysis

We adapt the data collection methodology proposed by Wang et al. [64] and follow up with manual removal of all non-IoT apps. The procedure includes:

[Step 1: Manual Search] We manually downloaded IoT apps from the Google Play Store based on definition of IoT apps (ref. Section 1). This entailed searching for apps used in the context of smart home/IoT. This forms our *Seed App Set*.

[Step 2: App Scraping] We scraped more related IoT apps, starting from our seed set and following the "Similar Apps" suggestions presented by the Play Store. We used play-scraper [44] to collect app names and descriptions.

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 Services	CALL_PHONE, PROCESS_OUTGOING_CALLS, READ_SMS, SEND_SMS 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

Table 1: Details of privacy categories and their associated permissions.

[Step 3: Keyword-based Filtering] We found a high number of false positives (i.e. apps that do not match our definition of IoT companion apps) in Step 2. Therefore, we first removed apps matching specific keywords (e.g., *currency, compiler, etc.*). We generated the set of keywords empirically, by identifying keywords highly correlated with false positives.

[Step 4: Naïve-Bayes Classification] We refined the candidate set by using machine learning to classify IoT and non-IoT apps. We first attempted to apply the BERT algorithm [26] to leverage context for better classification and Logistic Regression Model in conjunction with the TF-IDF vectorizer. However, these algorithms performed poorly on our dataset. Using a Naïve-Bayes classifier lead to better, but still suboptimal accuracy (64.6%) on a small set of manually labeled data.

[Step 5: Manual Filtering] We manually reviewed all apps classified by Naïve Bayes as IoT-related and we retained only those which unambiguously match our definition of companion app (given in Section 2).

Data collection results: We downloaded the app packages (APKs) using PlaystoreDownloader [11]. Our initial set of **2000** scraped apps was reduced to **1596** by keyword-based and Naïve Bayes filtering (a **20%** reduction). Manual inspection determined that only **484** (**30%**) were matching our definition and thus relevant to our analysis. We retained only **455** APKS as the remaining apps could not be downloaded, or were in a format incompatible with our tooling.

4 Data Privacy Posture

4.1 Methods

Our privacy evaluation is an extension of Kang et al.'s work on *privacy meter* which helps visualize privacy risks [38]. The app manifest provides detailed in-

Category	Tool
1. Requests for privacy-sensitive permissions	Manifest permission analysis
2. Evidence of permission-related data leaks	FlowDroid
3. Analysis of privacy policies	Polisis

Table 2: Data privacy categories relevant to our analysis, and respective tools

formation [60, 53] 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 [52, 35]. The first relevant issue is whether an app exhibits *requests for privacy-sensitive permissions*. However, the presence of sensitive permission does not imply data leaks; thus we also consider *evidence of permission-related data leaks*. The app privacy policy, when available, also provides important context. Thus we perform an *analysis of privacy policy*.

4.2 Tools and Analyses

Mapping between analysis categories and tools is summarized in Table 2. Details of individual analyses are provided below.

*Manifest Permission Analysis **(Cat. 1)***. We identify the privacy-sensitive permissions requested by each app as classified by Google [10]. These permissions provide access to resources that manage privacy sensitive data such as personally identifiable information, location data, contact books and so on [19]. Thereafter, we identify the user data categories accessed by the mobile apps. This classification is necessary as these categories can reveal sensitive user and device information regardless of the app type. The permission categories included: *Contacts*, *Content*, *Location*, *Calendar*, *Network*, *Device ID*, and *Phone State* [63]. We then map privacy-sensitive permissions to these categories according to the type of data to which each permission grants access.

From the declaration of the app permissions, we calculate privacy scores for the categories 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. Each category can request several permission accesses as mentioned in Table 1. For each permission, we have considered a binary 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 which encompasses lower privileges, we considered the higher privileges to avoid double counting. For example, *Full Network Access* encompasses all the other network-related permissions. Thus, for our privacy score calculations, 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 data is covered by fine-grained. Furthermore, permission to write storage implies permission to read. Thus, write storage is considered as two permissions. This also applies to read/write contacts, call logs, and calendar events.

*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* (locations where sensitive information could be created) and *sinks* (locations where

Perms	Mean	Min	Max
Network	0.75	0	1
Content	0.26	0.5	0
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

Table 3: Mean, min and max for each privacy category

such information could leave the app). Sources and sinks are specified as Java function signatures. FlowDroid parses the app binaries and produces an analysis of the application call graph as the output, including the existence of 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.

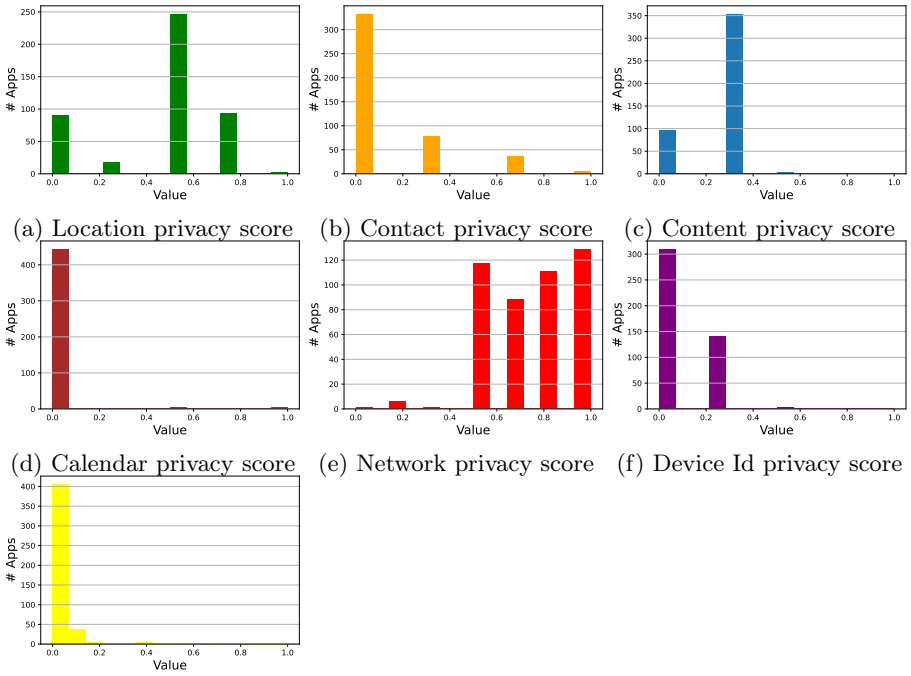
Analysis of Privacy Policy ***(Cat. 3)*** We conduct automated analysis of app privacy policies using the Polisis framework [30]. It breaks down the privacy policy into self-contained, semantically coherent segments and passes each into a set of neural network classifiers to designate labels depicting 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. [65].

Concretely, we collect policies linked by each app and pass them to Polisis. The output is a classification result 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 at the category level for segment classification: *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*.

Limitations: We considered Android-defined permissions for our permission analysis. Although custom permissions are prevalent in our corpus, we did not take into consideration these permissions—developer-defined permissions that allow them to set restrictions and share resources and capabilities with other apps—due to tool limitations. Li et al. acknowledge that the use of custom permissions is commonplace, and note that malicious applications try to exploit the flaws of custom permissions by getting dangerous system permissions without user consent and thus gaining illegal access to platform resources [43]. We were unable to extract the application manifest for **3** apps. The privacy policy analysis was carried out for **402** apps (the remaining apps had non-English policies unsupported by tooling, or did not provide a policy.)

4.3 Results

Manifest Permissions Analysis. 129 of the **448** 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



(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. 2: Individual distributions of privacy scores for the IoT applications

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 the majority of applications (**353**) requesting the permission to write to external storage. Only **92 out of 448** apps did not request any content permission. The rest of the categories had lower mean scores. It should be mentioned, however, that none of those permission categories is necessary to implement the core functionality of a companion app. Figures 2a-g detail the privacy score for each category. Table 4 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 are non-normal. The most relevant finding is that a number of categories exhibit weak (> 0.3) correlation with each other.

FlowDroid Permissions Analysis. FlowDroid detected leaks in **315 (69.23%)** 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_

	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

Table 4: r_s Between privacy categories (Loc = Location; Ctc = Contacts; Con = Content; Cal = Calendar; DId = Device Id; TeS = Telephony Service.)

PHONE_STATE, READ_PHONE_NUMBERS, BLUETOOTH_CONNECT. Six of these permissions were identified in privacy categories from Table 1.

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. Note, 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. These results are shown in Figure 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 app is presented in Figure 4a. The maximum number of labels detected in a privacy policy was **14** which was observed 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 (Figure 4b).

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.

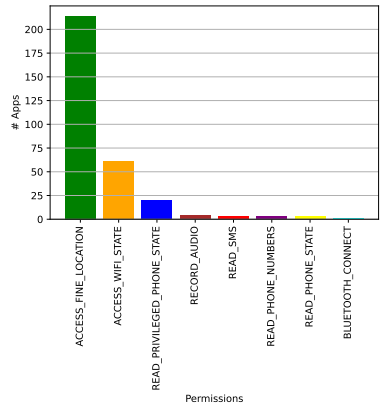
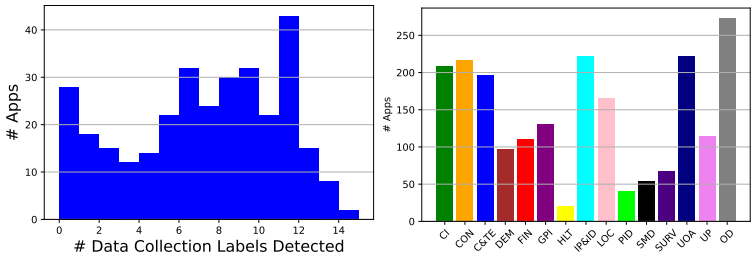


Fig. 3: Permissions responsible for the data leaks found through FlowDroid



(a) Frequency of labels detected. (b) No. of label types in each app.

Fig. 4: Personal Information Type Labels Statistics

5 Security Posture

5.1 Methods

Many app security analysis techniques exist (e.g., [39, 57, 3, 2]). For an technique to be used in our analysis, we require the following: tooling must be (i) available; (ii) not abandoned (we used a fixed threshold of two years w/o updates to determine abandonment); and (iii) executable on a modern machine/OS. We restrict our search to techniques measuring one of three different categories. The first is the *presence of known vulnerabilities*, as vulnerable software is at higher risk of facilitating accidental or attacker-driven information leaks. However, an app may also suffer from latent, undiscovered security issues that cannot be directly fingerprinted. Thus, we also measure *evidence of lack of maintenance*, as lack of updates may cause the presence of such latent issues [48]. The third security category is lack of *in-transit data protection*. Most apps encrypts in-transit data with Transport Layer Security (TLS). A misconfigured TLS stack, or lack of TLS, may result in an attacker being able to read in-transit user data.

5.2 Tools and Analyses

We decided to perform the bulk of the analysis using the Mobile Security Framework (MobSF) [3] which is a mature Open-Source pen-testing tool for Android.

Category	Tool
1. Presence of known vulnerabilities	MobSF static analysis
2. Evidence of lack of maintenance	Days since update
3. Lack of in-transit data protection	MobSF static analysis MobSF dynamic analysis Manual inspection

Table 5: Security categories relevant to our analysis

It takes an APK as input and produces a parseable vulnerability report as output. We also implemented a measurement of software abandonment (category 2 - *Evidence of lack of maintenance*). Table 5 summarizes tools used for each category. We discuss individual analyses below.

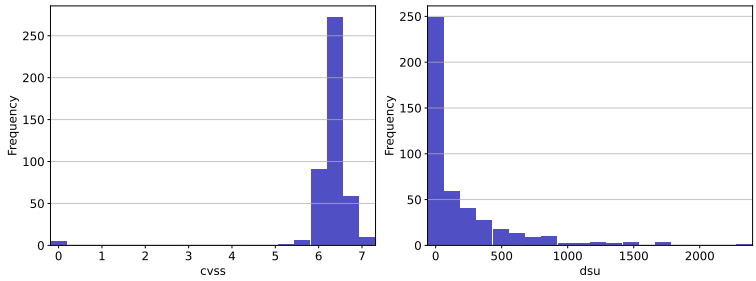
*MobSF CVSS Score **(Cat. 1)***. We use MobSF to statically fingerprint common, generic security-adverse patterns (e.g., logging sensitive information in plain text). MobSF further assigns a severity score between 0 and 10 to each issue, using the CVSS scoring system [1]. The tool also outputs the average CVSS scores across all detected issues. MobSF also generates information about the high-level category associated with each problem, according to CWE (Common Weakness Enumeration) categorization [4].

*Days Since Update **(Cat. 2)***. Modern apps tend to 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 (scraped using the `play-scraper` Python package) as an analysis to investigate abandonment issues. This is consistent with previous work [28], which has used "failure to release" as a proxy symptom of abandonment for software projects.

*Transport Layer Misconfigurations **(Cat. 3)***. HTTPS (Secure HTTP) is used for encrypted and secure transfer of data. HTTPS uses the TLS (Transport-Layer Security) standard to provide this functionality. We use MobSF static analyzer to identify app misconfigurations which may allow data to be transmitted in not properly encrypted form. Even if TLS is configured correctly, the use of invalid TLS certificates may still create risks. Thus we also use MobSF's static analyzer to identify invalid/expired TLS certificates.

*Transport Layer Secrecy Issues **(Cat. 3)***. We complement static TLS analysis with dynamic analysis of app-generated network connections. We perform four MobSF-driven tests: TLS misconfiguration, TLS Pinning/Certificate transparency, TLS Pinning/Certificate transparency Bypass, and Cleartext Traffic. The TLS misconfiguration test involved enabling an HTTPS Man-in-the-middle Proxy and removing the root CA. It 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 any non-TLS-encrypted traffic.

*Manual User Flow **(Cat. 3)***. We manually performed the following operations: user creation, password recovery, login, and single sign-on (if allowed by the manufacturer), while recording traffic generated by the app. Subsequently, we inspected the traffic for issues. We limited this test to a small set of apps



(a) Distribution of CVSS scores (b) Days since update

Fig. 5: Security Metrics

that were found to transmit unencrypted data by other tests. Inspection involved determining the nature and sensitivity of such unencrypted data if any.
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. *Manual User Flow* requires manual analysis of app network traces, which is complex and time-consuming. Due to the combination of these factors, we limited the latter two analyses to a smaller set of **15** apps.

5.3 Results

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 5a. Scores exhibits 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 [1]. While the data may appear to be close to normally distributed, it fails the Shapiro-Wilk normality test ($W = 0.37$, $p = 1.68e^{-36}$). 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.

Table 6 lists the top-10 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. Although this practice is discouraged, not all exposed information may be critical, and reading it requires physical access to the device. Other common CWE such as CWE-312, -330, and -327 are concerning as they indicate deviation from best data secrecy practices.

Evidence of Lack of Maintenance The average number of days since an update is **237** (min: **0**; max: **2462**). The distribution, reported in Figure 5b, is concentrated around low values, which implies most companion apps are frequently updated (more than **238** received an update within the last **100** days).

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
CWE-200	Information Exposure	349
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	347
CWE-649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking	208
CWE-749	Exposed Dangerous Method or Function	185
CWE-295	Improper Certificate Validation	123

Table 6: Most common CWEs found by MobSF Issue Analysis

Description	Apps
Base cfg is insecurely configured to permit clear text traffic to all domains.	99
Domain cfg is insecurely configured to permit clear text traffic to these domains.	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

Table 7: Network security configuration issues detected across the app dataset

This metric is however uncorrelated to *CVSS score* (as discussed in Section 7.1), so a high frequency of updates does not imply absence of vulnerabilities.

Lack of In-transit Data Protection. Transport-layer misconfigurations: Table 7 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 [67]. The top-two issues allow unencrypted communications to all or some domains. The others generally violate various best practices concerning the choice of TLS certificates that should be trusted. We also measured the use of invalid/expired TLS certificates, which appears to be quite widespread: it affects **96 out of 452 (21.2%)** applications.
Transport-layer secrecy issues: For this dynamic analysis, we selected **15** apps for which other analyses had identified potentially concerning issues based on high risk index values given by RiskInDroid (detailed in Section 6.2). The results of the dynamic TLS tests on the 15 apps are summarized in Table 8. While the app sample is too small to extract general conclusions, this test identified significant TLS deployment issues. Note that, differently from our static analysis results reported in the "Transport layer misconfigurations" test—which represent configuration issues potentially leading to security problems—the dynamic test identifies issues concerning the actual network traffic generated by 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%

Table 8: Dynamic analysis results for the selected 15 apps

Manual user flow analysis: In order 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 that allows users to monitor home cameras from their cell phones. 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.

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 apps with particularly concerning flags revealed isolated cases of sensitive credentials being sent in cleartext.

6 Risk Posture

6.1 Methods

By risk posture, we refer to the likelihood that an app may purposely mishandle user data. There exist a variety of risk profiling techniques [50, 55, 37, 36, 17, 61]. For selection, we used the same criterion discussed in Section 5. Most techniques have the goal of assessing the similarity of an app to given malware; therefore, we further used published results to exclude older techniques that are generally outperformed by more recent ones. *Similarity to known malware* is an indirect measure, as an app could perform malicious operations which do not resemble known patterns. Thus we also include a limited but direct measure of risk, *presence of malicious URLs*, which considers whether an app contains URLs associated with known malicious domains. Tool selection is summarized in Table 9 and discussed below.

Category	Tool
1. Similarity to known malware	RiskInDroid score
2. Presence of malicious URLs	URL analysis pipeline

Table 9: Risk categories relevant to our analysis

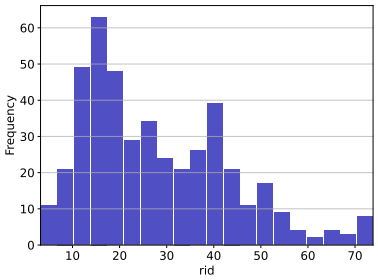


Fig. 6: RiskInDroid score

6.2 Tools and Analyses

RiskInDroid ***(Cat. 1)*** distinguishes risky applications from non-risky ones based on permissions, using machine learning-based classifiers (multiple techniques are supported, including SVM, Multinomial Naive Bayes, Grading Boosting, and Logistic Regression). The tool produces a risk index value (RIV) as output; RIV estimates the similarity of the permission set to those commonly seen in malware. Higher RIVs correspond to risky applications and possibly malware. The value of RIVs ranges from 0 to 100, where values above 75 suggests potential malware. *URL Analysis* ***(Cat. 2)*** uses MobSF static analyzer to extract URLs within an app APK. It then uses three different blocklists [6, 13, 12]) to identify risky and/or malicious URLs.

Limitations: RiskInDroid failed on **9 out of 455** apps, thus the results are for **446** apps only. The blocklists we used for URL analysis will flag an entire CDN/content provider hostname (e.g., *gw.alicdn.com*) when one of their customers performs malicious actions, thus resulting in potential false positives. We manage this issue by manually post-processing the URL analysis results.

6.3 Results

Similarity to Known Malware. The minimum RiskInDroid score in our apps was **4.9** and the maximum was **75.59**. Only **3** apps have a RIV higher than 75: *XVRView*, *blurams*, and *Verisure Cameras*. According to their descriptions, all three apps are used to remotely stream home smart cameras. A cursory app review did not find evidence of malicious behavior, so it is simply possible that the combination of permissions required by this class of apps is close to RiskInDroid's internal malware model. We remark that in our security analysis (Section 5.3) we also found that *XVRView* transmits credentials in cleartext.

Presence of Malicious URLs. Overall, we extracted and analyzed **5722** URLs from our corpus of apps. **14** hostnames appear on security blocklists,

and **3** belong to a domain which suffered a recent breach. However, upon manual inspection we determined that all hostnames either belonged to CDN platforms/backends (e.g., *ytimg.com*), appeared related to the device manufacturers (e.g., *xiaoyi.pl*), or were popular content hosting platforms (e.g., *imgur.com*). In all these cases, it is impossible, without further knowledge, to unambiguously determine that the URL is malicious. We note, however, that **345** apps (**76.3%**) were found to use backend services provided by *firebase.io*, which has been in the past the subject of large-scale data leaks [5].

Take-aways. Most apps do not exhibit characteristics commonly associated with malicious software and data misuse. This may be the result of selection bias: as the Play Store screens uploaded apps for malware, there may exist malicious apps which however do not exhibit easily detectable malware behavior. Reliable mobile malware detection is a complex task in itself, and orthogonal to our work.

7 Discussion

Our main research question (ref. Section 1) was: *How do companion apps fare in respect to the measured privacy-related aspects? Are there significant general concerns broadly affecting companion apps?* Overall, the privacy picture painted by our analysis is not critical, but certainly sub-optimal. While no app was found to carry overtly malicious behavior (Section 6), apps routinely request permissions that do not appear necessary (Section 4). Poor app programming and configuration practices, likely to increase the risk of compromise and data leaks, were also common (Section 5).

Given these considerations, we believe it would be important for users evaluating the purchase of a device and related app, to have access to understandable insights about the app privacy posture. Such information would enable users to prefer security-robust apps, which in turn may incentivize manufacturers to improve their privacy practices.

In other communities, visibility into the security of software packages via online aggregators is rapidly becoming popular. Examples from the Open Source Software community include the OpenSSF Security Metrics project [9] and the LFX insights project [8]. We believe a similar resource focused on IoT app privacy would be beneficial to the Android IoT user community.

7.1 Cross-Dimension Analysis

We now evaluate the relationships between the results of different analyses (*RQ2: Is there evidence of positive or negative correlation between metrics measuring different aspects of app privacy?* in Section 1.) We only consider measures that result in a non-boolean numerical score: Privacy Score (Section 4.3, averaged across all categories), CVSS score (Section 5.3), Days since update (Section 5.3), RiskInDroid score (Section 6.3). We additionally generate a score from the FlowDroid analysis of Section 4, by counting all paths between FlowDroid sources and sinks. As the value distribution of many measures deviates from normal, we

	Priv	FlowDroid	CVSS	DSU	RID
Priv	1.00	-	-	-	-
FlowDroid	0.147	1.00	-	-	-
CVSS	0.003	0.075	1.00	-	-
DSU	-.197	0.028	0.049	1.00	-
RID	0.149	0.027	0.104	0.007	1.00

Table 10: Spearman's rank correlation coefficients between measured characteristics ("Priv" stands for "Average Privacy Score"; "DSU" for "Days Since Update"; "RID" for "RiskInDroid Score")

used non-parametric Spearman's rank correlation (r_s). According to accepted practice [31], we interpret r_s values < 0.3 as representing no correlation.
Results: The correlation matrix is shown in Table 10; no significant correlation is evidenced between any pair of measures. Based on our results, thus **privacy issues do not appear to propagate along with multiple aspects**: the posture of each app according to one aspect is uncorrelated to the same along with other aspects. For the same reason, **we did not find evidence of tradeoffs between app quality in different aspects**.

7.2 State of Measuring Tools

FlowDroid [18], MobSF [3], and RiskInDroid [50] are mature and well-maintained tools, and we found them easy to integrate into an automated analysis workflow. MobSF and FlowDroid are backed by robust developer communities; while FlowDroid is backed by a mobile security startup. Polisis [30] works on textual privacy policies and we did not run into significant issues while using it. We considered Axplorer [21], Arcade [14] and IccTA [42] for computing app permissions without relying on the manifest. Unfortunately, none worked and we implemented our manifest analysis. We considered CryptoGuard [57] and CryLogger [56] for detecting incorrect use of crypto primitives. We found them to be less applicable to a broad set of apps. Both exhibited a high failure rate.

8 Related Work

Related Work on Metrics: Kang et al [38] calculate Android app privacy scores based on a list of 30 permissions commonly used by malware. Biswas et al. [22] developed a method to measure privacy across Location, Content, and Contacts. In our paper, we extended and combined these two methods to calculate privacy for different categories. Recently, Babun et al. [20] proposed IoTWatch, a methodology for matching user privacy preferences to app-transmitted data. Integrating IoTWatch in our work is an interesting future direction.

Jansen [34] highlighted limitations of popular security metrics, such as the reliance on human judgment and inherent subjectivity. Similarly, Krutz et al. found that user ratings weakly correlate with security [40]. Consequently, we mostly focus on security analyses rooted in objective properties of each app (e.g.,

score of CVSS issues). RiskMon [37] assesses the risk incurred by app sensitive operations relative to user-provided expectations. As we require an assessment to be automated, this approach is outside the scope of our work. WHYPER [54] flags unexplained app permissions in app descriptions. We do not include this approach as lack of explanation, while suspicious, does not represent a security issue per se. Peng et al. [55] proposed probabilistic generative models to attribute a risk score to Android apps based on the combination of permissions requested. Later work [50] has shown that ML-driven analysis of permission sets leads to superior results; therefore we prefer the latter approach.

Related Work on IoT-Based Studies: Liu et al. [45] defined the notion of *app-in-the-middle* IoTs, which illustrates the increasingly popular setup where constrained IoT devices rely on mobile apps for connectivity. They then proposed a formalization of the security of such setups. Ding and Hu [27] proposed a framework for identifying security-relevant hidden integration between IoT apps.

Wang et al. [64] and Mauro Junior et al. [49] analyzed mobile companion apps [64] to infer vulnerabilities in their devices and backends. Both works focus mainly on devices and not apps. Chatzoglou et al. [25] performed an in-depth security analysis of a small (41) number of IoT apps. We focus on large-scale lightweight automated analysis, thus maximizing external validity. Fernandes et al. [29] performed an analysis of smart home apps on the Samsung SmartThings platform, revealing numerous issues due to over-privileging. Our analysis focuses on mobile IoT apps rather than third-party smart-home apps. Mohanty and Sridhar conducted a review of the security of 102 IoT companion apps [51]. This paper focused specifically on identifying 9 pre-determined security issues in apps built using hybrid frameworks; our work has broader scope both in terms of types of apps, and issues of interest.

9 Conclusions

We analyzed the privacy posture of 455 Android mobile companion apps by evaluating three pillars of it including data privacy, security, and risk. Our study shows that overall app data privacy, security, and risk posture are reasonable. However, it also evidences some ecosystem-level shortcomings; such as over-requesting permissions; and the widespread use of programming and configuration practices negatively affecting security. We hope our work can act as the starting point for further investigations in this domain, and facilitate the design of software quality metrics and guidelines that can lead to better app design.

References

1. Common vulnerability scoring system version 3.1: Specification document. https://www.first.org/cvss/specification-document (2019)
2. GitHub - linkedin/qark: Tool to look for several security related Android application vulnerabilities. https://github.com/linkedin/qark (2019)
3. Mobile security framework. https://github.com/MobSF/Mobile-Security-Framework-MobSF (2020)

4. Cwe list version 4.6. https://cwe.mitre.org/data/index.html (2021)
5. Popular android apps with 142.5 million collective installs leak user data. https://cybernews.com/security/research-popular-android-apps-with-142-5-million-collective-downloads-are-leaking-user-data/ (2021)
6. Bulk domain blacklist checker. https://www.bulkblacklist.com (2022)
7. Google play. https://play.google.com/store (2022)
8. Lfx insights. https://insights.lfx.linuxfoundation.org/projects (2022)
9. Metrics - open source security foundation. https://metrics.openssf.org (2022)
10. Permissions on android. https://developer.android.com/guide/topics/permissions/overview (2022)
11. Play store downloader. https://github.com/ClaudiuGeorgiu/PlaystoreDownloader (2022)
12. Url/ip lookup | webroot brightcloud. https://www.brightcloud.com (2022)
13. Website reputation checker. https://www.urlvoid.com (2022)
14. Aafer, Y., Tao, G., Huang, J., Zhang, X., Li, N.: Precise android api protection mapping derivation and reasoning. In: ACM CCS (2018)
15. Alhirabi, N., Rana, O., Perera, C.: Security and privacy requirements for the internet of things: A survey. ACM Transactions on Internet of Things **2**(1), 1–37 (2021)
16. Allhoff, F., Henschke, A.: The internet of things: Foundational ethical issues. Internet of Things **1**, 55–66 (2018)
17. Alshehri, A., Marcinek, P., Alzahrani, A., Alshahrani, H., Fu, H.: PUREDroid: Permission Usage and Risk Estimation for Android Applications. In: ICISDM (2019)
18. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In: PLDI (2014)
19. Baalous, R., Poet, R.: How dangerous permissions are described in android apps' privacy policies? In: SIN (2018)
20. Babun, L., Celik, Z.B., McDaniel, P., Uluagac, A.S.: Real-time analysis of privacy-(un) aware iot applications. In: PETS (2021)
21. Backes, M., Bugiel, S., Derr, E., McDaniel, P., Octeau, D., Weisgerber, S.: On demystifying the android application framework: Re-visiting android permission specification analysis. In: USENIX Security Symposium (2016)
22. Biswas, D., Aad, I., Perrucci, G.P.: Privacy panel: Usable and quantifiable mobile privacy. In: ARES (2013)
23. Catarinucci, L., de Donno, D., Mainetti, L., Palano, L., Patrono, L., Stefanizzi, M.L., Tarricone, L.: An IoT-Aware Architecture for Smart Healthcare Systems. IEEE Internet of Things Journal **2**(6), 515–526 (Dec 2015)
24. Celik, Z.B., Fernandes, E., Pauley, E., Tan, G., McDaniel, P.: Program analysis of commodity iot applications for security and privacy: Challenges and opportunities. ACM Computing Surveys (CSUR) **52**(4), 1–30 (2019)
25. Chatzoglou, E., Kambourakis, G., Smiliotopoulos, C.: Let the cat out of the bag: Popular android iot apps under security scrutiny. Sensors **22**(2) (2022)
26. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
27. Ding, W., Hu, H.: On the Safety of IoT Device Physical Interaction Control. In: ACM CCS (2018)
28. English, R., Schweik, C.M.: Identifying success and tragedy of floss commons: A preliminary classification of sourceforge.net projects. In: FLOSS ICSE Workshops (2007)

29. Fernandes, E., Jung, J., Prakash, A.: Security Analysis of Emerging Smart Home Applications. In: IEEE S&P (2016)
30. Harkous, H., Fawaz, K., Lebret, R., Schaub, F., Shin, K.G., Aberer, K.: Polisis: Automated analysis and presentation of privacy policies using deep learning. In: USENIX Security Symposium (2018)
31. Hinkle, D.E., Wiersma, W., Jurs, S.G.: Applied statistics for the behavioral sciences, vol. 663. Houghton Mifflin College Division (2003)
32. Holloway, D., Green, L.: The Internet of toys. Communication Research and Practice **2**(4), 506–519 (Oct 2016)
33. ISO/IEC: ISO/IEC 20924:2018(en), Information technology — Internet of Things (IoT) — Vocabulary. https://www.iso.org/obp/ui/#iso:std:iso-iec:20924:ed-1:v1:en
34. Jansen, W.: Research Directions in Security Metrics. Tech. Rep. 7564, NIST (2009)
35. Jha, A.K., Lee, S., Lee, W.J.: Developer mistakes in writing android manifests: An empirical study of configuration errors. In: IEEE/ACM MSR (2017)
36. Jiang, J., Li, S., Yu, M., Chen, K., Liu, C., Huang, W., Li, G.: MRDroid: A Multi-act Classification Model for Android Malware Risk Assessment. In: IEEE MASS (2018)
37. Jing, Y., Ahn, G.J., Zhao, Z., Hu, H.: RiskMon: Continuous and automated risk assessment of mobile applications. In: CODASPY (2014)
38. Kang, J., Kim, H., Cheong, Y.G., Huh, J.H.: Visualizing privacy risks of mobile applications through a privacy meter. In: ISPEC (2015)
39. Kapitsaki, G., Ioannou, M.: Examining the Privacy Vulnerability Level of Android Applications:. In: WEBIST (2019)
40. Krutz, D.E., Munaiah, N., Meneely, A., Malachowsky, S.A.: Examining the relationship between security metrics and user ratings of mobile apps: A case study. In: WAMA (2016)
41. Kumar, D., Shen, K., Case, B., Garg, D., Alperovich, G., Kuznetsov, D., Gupta, R., Durumeric, Z.: All things considered: An analysis of iot devices on home networks. In: USENIX Security Symposium (2019)
42. Li, L., Bartel, A., Bissyandé, T.F., Klein, J., Le Traon, Y., Arzt, S., Rasthofer, S., Bodden, E., Octeau, D., McDaniel, P.: Iccta: Detecting inter-component privacy leaks in android apps. In: IEEE/ACM ICSE (2015)
43. Li, R., Diao, W., Li, Z., Du, J., Guo, S.: Android custom permissions demystified: From privilege escalation to design shortcomings. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 70–86. IEEE (2021)
44. Liu, D.: play-scraper. https://pypi.org/project/play-scraper/
45. Liu, H., Li, J., Gu, D.: Understanding the security of app-in-the-middle IoT. Computers & Security **97**, 102000 (Oct 2020)
46. Marquez, J., Villanueva, J., Solarte, Z., Garcia, A.: IoT in Education: Integration of Objects with Virtual Academic Communities. In: New Advances in Information Systems and Technologies, vol. 444, pp. 201–212. Springer International Publishing, Cham (2016)
47. Matheu, S.N., Hernández-Ramos, J.L., Skarmeta, A.F., Baldini, G.: A Survey of Cybersecurity Certification for the Internet of Things. ACM Computing Surveys **53**(6), 1–36 (Feb 2021)
48. Mathur, A., Malkin, N., Harbach, M., Peer, E., Egelman, S.: Quantifying users' beliefs about software updates. Proceedings 2018 Workshop on Usable Security (2018)

49. Mauro Junior, D., Melo, L., Lu, H., d'Amorim, M., Prakash, A.: A Study of Vulnerability Analysis of Popular Smart Devices Through Their Companion Apps. In: IEEE SPW (2019)
50. Merlo, A., Georgiu, G.C.: RiskInDroid: Machine Learning-Based Risk Analysis on Android. In: IFIP SEC (2017)
51. Mohanty, A., Sridhar, M.: HybriDiagnostics: Evaluating Security Issues in Hybrid SmartHome Companion Apps. In: IEEE SPW (2021)
52. Momen, N., Hatamian, M., Fritsch, L.: Did app privacy improve after the gdpr? IEEE Security & Privacy **17**(6), 10–20 (2019)
53. Mylonas, A., Theoharidou, M., Gritzalis, D.: Assessing privacy risks in android: A user-centric approach. In: International Workshop on Risk Assessment and Risk-driven Testing. pp. 21–37. Springer (2013)
54. Pandita, R., Xiao, X., Yang, W., Enck, W., Xie, T.: Whyper: Towards automating risk assessment of mobile applications. In: USENIX Security (2013)
55. Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Using probabilistic generative models for ranking risks of android apps. In: ACM CCS (2012)
56. Piccolboni, L., Di Guglielmo, G., Carloni, L., Sethumadhavan, S.: Crylogger: Detecting crypto misuses dynamically. In: IEEE S&P (2021)
57. Rahaman, S., Xiao, Y., Afrose, S., Shaon, F., Tian, K., Frantz, M., Danfeng, Yao, Kantarcioglu, M.: Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects. In: ACM CCS (2019)
58. Ren, J., Dubois, D.J., Choffnes, D., Mandalari, A.M., Kolcun, R., Haddadi, H.: Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In: ACM IMC (2019)
59. Rivera, D., García, A., Martín-Ruiz, M.L., Alarcos, B., Velasco, J.R., Oliva, A.G.: Secure communications and protected data for a internet of things smart toy platform. IEEE Internet of Things Journal **6**(2), 3785–3795 (2019)
60. Tandel, S., Jamadar, A.: Impact of progressive web apps on web app development. International Journal of Innovative Research in Science, Engineering and Technology **7**(9), 9439–9444 (2018)
61. Utama, R.A., Sukarno, P., Jadied, E.M.: Analysis and Classification of Danger Level in Android Applications Using Naive Bayes Algorithm. In: ICoICT (2018)
62. Vashi, S., Ram, J., Modi, J., Verma, S., Prakash, C.: Internet of Things (IoT): A vision, architectural elements, and security issues. In: I-SMAC (2017)
63. Wader, S.S.: How android application permissions impact user's data privacy? International Journal of Research Publication and Reviews **2**(3), 498–502 (2021)
64. Wang, X., Sun, Y., Nanda, S., Wang, X.: Looking from the mirror: Evaluating iot device security through mobile companion apps. In: USENIX Security (2019)
65. Wilson, S., Schaub, F., Dara, A.A., Liu, F., Cherivirala, S., Leon, P.G., Andersen, M.S., Zimmeck, S., Sathyendra, K.M., Russell, N.C., et al.: The creation and analysis of a website privacy policy corpus. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1330–1340 (2016)
66. Wuyts, K., Joosen, W.: Linddun privacy threat modeling: a tutorial. CW Reports (2015)
67. Yermakov, M.: Understanding the android cleartexttrafficpermitted flag. https://appsec-labs.com/portal/understanding-the-android-cleartexttrafficpermitted-flag/ (2020)