Understanding IoT Security from a Market-Scale Perspective

Xin Jin*
The Ohio State University
jin.967@osu.edu

Sunil Manandhar*†
IBM T.J. Watson Research Center
sunil@ibm.com

Kaushal Kafle
William & Mary
kkafle@cs.wm.edu

Zhiqiang Lin
The Ohio State University
zlin@cse.ohio-state.edu

Adwait Nadkarni
William & Mary
nadkarni@cs.wm.edu

ABSTRACT

Consumer IoT products and services are ubiquitous; yet, a proper characterization of consumer IoT security is infeasible without an understanding of what IoT products are on the market, *i.e.*, without a *market-scale* perspective. This paper seeks to close this gap by developing the IoTSpotter framework, which automatically constructs a market-scale snapshot of *mobile-IoT apps*, *i.e.*, mobile apps that are used as companions or automation providers to IoT devices. IoTSpotter also extracts artifacts that allow us to examine the security of this snapshot in the IoT context (*e.g.*, devices supported by apps, IoT-specific libraries). Using IoTSpotter, we identify 37,783 mobile-IoT apps from Google Play, the largest set of mobile-IoT apps so far, and uncover *7 key results* in the process ($\mathcal{R}_1$–$\mathcal{R}_7$). We leverage this dataset to perform three key security analyses that lead to *10 impactful security findings* ($\mathcal{F}_1$–$\mathcal{F}_{10}$) that demonstrate the current state of mobile-IoT apps. Our analysis uncovers severe cryptographic violations in 94.11% (863/917) mobile-IoT apps with >1 million installs each, 65 vulnerable IoT-specific libraries affected by 79 unique CVEs, and used by 40 popular apps, and 7,887 apps that is affected by the Janus vulnerability. Finally, a case study with 18 popular mobile-IoT apps uncovers the critical impact of the vulnerabilities in them on important IoT artifacts and functions, motivating the development of mobile security analysis contextualized to IoT.

CCS CONCEPTS

• **Security and privacy → Software and application security**; • **Computing methodologies → Machine learning approaches**.

KEYWORDS

IoT security; mobile-IoT app identification; cryptographic API misuse; third-parity library vulnerability

*These authors contributed equally to this work.
†This work was completed when the author was at William & Mary.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560640

ACM Reference Format:
Xin Jin, Sunil Manandhar, Kaushal Kafle, Zhiqiang Lin, and Adwait Nadkarni. 2022. Understanding IoT Security from a Market-Scale Perspective. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560640

1 INTRODUCTION

Regulators, researchers, and practitioners are grappling with the implications of billions of potentially vulnerable products [26] that can sense and modify private user environments. In securing IoT products at scale, a key challenge is *triaging*, *i.e.*, quickly identifying products that need the most attention due to the prevalence and impact of vulnerabilities in them. At present, triaging the vast and fragmented IoT ecosystem is infeasible due to the lack of a concrete understanding of *what constitutes IoT*, *i.e.*, what products make up the IoT ecosystem. This paper seeks to address this knowledge gap by developing an evolving *market-scale snapshot* of IoT products, *i.e.*, a concrete collection of products *available* to consumers, which will enable a diverse array of security analyses leading to *generalizable* insights that apply to all (or most) available products.

We focus on a security-critical *IoT product-class* that serves as the *controller* and *user-interface (UI)* of the IoT system: *mobile-IoT apps*, *i.e.*, mobile apps that serve as companion apps for devices, or represent third-party integration/automation services (*e.g.*, the Google Home app). Mobile-IoT apps form a *critical attack surface* of the IoT system, *e.g.*, prior work [44] compromised the TP Link Kasa app to gain remote control over a NEST security camera, demonstrating how vulnerabilities in mobile-IoT apps can enable privilege escalation to critical devices. Further, as mobile-IoT apps expose interfaces to IoT devices and cloud back-ends, their large-scale analysis could help us understand the security properties of the general IoT landscape. However, while a market-scale analysis is common in the general mobile security domain due to the availability of apps from markets (*e.g.*, Google Play), the lack of a dedicated dataset/market of mobile-IoT apps makes such analysis challenging for IoT.

This paper proposes IoTSpotter, which *identifies mobile-IoT apps* from general app markets such as Google Play, and facilitates large-scale security analysis that is *contextualized to IoT*. IoTSpotter builds upon *two key observations*: **(Ob$_1$)** As mobile-IoT apps manage IoT devices, they exhibit characteristics that distinguish them from general-purpose mobile apps, which can be leveraged for their effective identification, and, **(Ob$_2$)** The security ramifications of vulnerabilities in mobile-IoT apps go beyond the mobile space, and impact the user's physical environment via connected

IoT devices. These observations guide IoTSpotter's design for developing a *market-scale snapshot* of mobile-IoT apps and extracting IoT-related information (*e.g.*, supported devices, IoT libraries) for a multi-faceted security analysis of the snapshot. Our deployment of IoTSpotter leads to key measurement results ($\mathcal{R}_1$–$\mathcal{R}_7$), and novel security findings ($\mathcal{F}_1$–$\mathcal{F}_{10}$) that illustrate the prevalence and impact of vulnerabilities in mobile-IoT apps. We now outline the our contributions, followed by a summary of security findings:

- **A market-scale perspective of IoT–** This paper introduces a new direction for IoT security analysis by enabling a market-scale perspective over IoT products, which would allow future security research to develop analysis that *generalize* to products *available* to consumers.
- **The IoTSpotter Framework –** We design IoTSpotter, the *first* framework for generating an evolving, market-scale snapshot of mobile-IoT apps for security. With the intuition that the differences between mobile-IoT and non-IoT apps may show in app meta-data (*e.g.*, descriptions), IoTSpotter treats the problem of identifying mobile-IoT apps in an app market as a *classification* task. IoTSpotter's classifiers identify mobile-IoT apps with 94.9% precision and 92.5% recall. We deploy IoTSpotter on 2 million Google Play apps and obtain 37,783 mobile-IoT apps, the largest such dataset to date ($\mathcal{R}_1$), and manually validate a random sample of this snapshot (2,250 apps) to find that 88.4% are indeed mobile-IoT ($\mathcal{R}_2$). Finally, IoTSpotter's Named Entity Recognition (NER) model analyzes app descriptions from this snapshot to identify 65,676 unique IoT products (*i.e.*, devices, services) clustered into 962 product-types ($\mathcal{R}_4$), enabling the security analysis of apps *in the context of the IoT devices they support.*
- **Security analysis of the mobile-IoT snapshot –** We perform three analyses to explore the prevalence of vulnerabilities in the mobile-IoT snapshot, comparing with proportionate samples of non-IoT apps in each case. We first develop a novel differential analysis approach that allows IoTSpotter to identify 19,939 *IoT-specific third-party libraries* from 522,285 libraries in mobile-IoT apps ($\mathcal{R}_6$), from which we identify 11 popular library families ($\mathcal{R}_7$). To understand the effect of IoT libraries on the security of mobile-IoT apps, we analyze IoT libraries for known CVEs, and find popular apps that use them. Second, we use cryptographic API misuse detectors [46, 55] to analyze 917 popular mobile-IoT apps (>1 million installs). Finally, we analyze the entire mobile-IoT snapshot for the Janus app signing vulnerability [11].
- **Vulnerability Impact on IoT security:** As the mobile-IoT apps in our snapshot support several security/privacy-critical products (*e.g.*, cameras, security systems) ($\mathcal{R}_5$), we investigate the impact of vulnerabilities in them in the IoT context, using a systematic case study of 18 popular, vulnerable, mobile-IoT apps. We find that vulnerabilities in mobile-IoT apps pose a threat to key IoT components/functions ($\mathcal{F}_8$, $\mathcal{F}_9$), as we discuss next in our summary of security findings.

Summary of Security Findings ($\mathcal{F}_1$–$\mathcal{F}_{10}$): Our security analysis of the mobile-IoT snapshot, and the contextualization of the findings to IoT security, lead to *ten* impactful findings. We discover that of 19k IoT-specific libraries identified by IoTSpotter, several (65 libraries, 481 versions) are vulnerable (subject to 79 CVEs) ($\mathcal{F}_1$), and in use by 40 popular mobile-IoT apps (>50k installs), of which 6 have >1 million installs ($\mathcal{F}_3$). In comparison, general/non-IoT libraries (5k sample, 2.5k popular and 2.5k random) are far more vulnerable (73 libraries, 7,203 versions, 193 CVEs), with popular libraries contributing to most of the vulnerabilities ($\mathcal{F}_2$). Thus, we find a greater number of popular non-IoT apps (509 with >50k installs, 73 with >1 million installs) using vulnerable non-IoT libraries ($\mathcal{F}_4$).

Further, we find that 94.11% (863/917) top mobile-IoT apps (>1 million installs) have at least one crypto-API misuse vulnerability, which is approximately similar to our results on a proportionate set of non-IoT apps (96.29%, *i.e.*, 883/917 vulnerable) ($\mathcal{F}_5$). Our manual validation of 589 high-severity vulnerabilities detected by CryptoGuard [55] in 10 top mobile-IoT apps finds that 82.5% (*i.e.*, 486/589) are used/called from within the apps ($\mathcal{F}_6$).

We analyze the entire snapshot of 37k mobile-IoT apps and find 7,887 susceptible to the Janus vulnerability (263 with >1 million installs, 33 with >50 million installs) ($\mathcal{F}_7$). While a proportionate sample of non-IoT apps demonstrates similar results overall (7,765 vulnerable apps), a disproportionate number of moderately popular mobile-IoT apps (5k-100k installs) are more vulnerable than non-IoT, while in contrast, more unpopular (<500 installs) non-IoT apps are vulnerable than mobile-IoT ($\mathcal{F}_8$).

Finally, our case study reveals that every type of vulnerability we found impacts at least one critical IoT component, such as app/device functions, and user authentication ($\mathcal{F}_9$). All 16/18 apps in which this impact is observed support security-sensitive devices (cameras being the majority) ($\mathcal{F}_{10}$), underscoring the need for a focused, large-scale, and contextualized analysis of mobile-IoT apps.

Artifact Release and Vulnerability Disclosure: The code and data associated with this paper is available in our Github repository [40]. We have reported the confirmed vulnerabilities from the case study (Sec. 8) to 12/18 vendors, and are in the process of reporting to the remaining 6 vendors. Additional details on the vulnerability disclosure can be found in our online appendix [39].

2 MOTIVATION

Mobile-IoT apps serve as the primary UIs for controlling, configuring, and automating IoT devices, and vulnerabilities in them could provide adversaries with privileged access to IoT devices, services, and the user's physical environment [44, 48]. While there is a large body of work on mobile and IoT security, researchers have only recently begun to explore the security of mobile-IoT apps, *which lie at the intersection of mobile and IoT* [23, 43–45, 66]. For instance, our prior work [44, 45] demonstrates how vulnerabilities in mobile-IoT apps can be exploited to target IoT platforms and devices, while Wang et al. [66] use mobile-IoT apps to approximate the characteristics of the connected devices, to estimate the vulnerabilities in them. Prior work relies on small app datasets crawled from Google Play using regular expression based searches or snowball sampling [33], the largest being Wang et al.'s set of 2081 [66]. Such analysis with limited, ad-hoc, sub-samples of mobile-IoT apps may result in insufficient insights, *i.e.*, which do not apply to a representative set of apps available to consumers. Therefore, we need to develop a large collection of mobile-IoT apps from markets such as Google Play to enable a generalizable understanding of the state of IoT security.

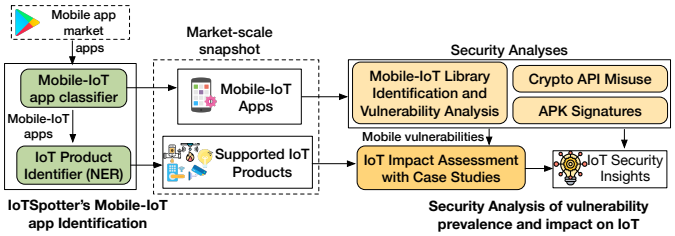


Figure 1: Overview of IoTSpotter.

This paper is motivated by the goal of developing a *market-scale snapshot* of mobile-IoT apps and demonstrating its usefulness through diverse security analyses that are impactful in the IoT context, and is guided by **two core research questions (RQs)**:

RQ1 *How to automatically develop a market-scale snapshot of mobile-IoT apps from markets containing heterogeneous apps?* This is particularly challenging, given that there is no single repository of mobile-IoT apps, and ad-hoc crawling from general app markets such as Google Play leads to insufficient results.

RQ2 *How to make this snapshot useful for IoT security analysis?* Vulnerabilities in mobile-IoT apps must not be studied in isolation, but *in the IoT context* relevant to specific apps. Thus, we need to extract useful information and artifacts from mobile-IoT apps that enable impactful IoT security analysis.

With the motivation of demonstrating IoTSpotter's utility, we seek to analyze the generated snapshot using diverse security analyses, particularly balancing between analyses that can scale to the entire snapshot (*e.g.*, testing for vulnerable IoT libraries, the Janus signature vulnerability [11]), and analyses that may not scale due to limitations in existing tools, but are yet critical for IoT security, such as the vulnerable use of cryptography. Finally, as vulnerabilities in mobile-IoT apps can be leveraged to attack the complex IoT system (*e.g.*, devices, servers) [44, 48], we seek to leverage the IoT context obtained by IoTSpotter to study the impact of the mobile vulnerabilities discovered in our analysis on IoT security.

3 THE IOTSPOTTER FRAMEWORK

We propose IoTSpotter, a semi-automated framework that systematically identifies mobile-IoT apps from popular markets such as Google Play (**RQ1**), and extracts information (*e.g.*, the IoT devices the app supports) and artifacts (*e.g.*, IoT-specific libraries) that facilitatel impactful security analysis in the IoT context (**RQ2**). Figure 1 provides an overview of IoTSpotter.

IoTSpotter frames the problem of extracting mobile-IoT apps from markets as a *classification task*, *i.e.*, of distinguishing between mobile-IoT and non-IoT apps based on salient differences that may appear in their metadata. Prior to pursuing this ML-based approach, we tried to crawl Google Play for mobile-IoT apps using regular expression-based searches applied to descriptions (similar to prior work [44]). However, we found that a regex-based approach results in significant false positives and negatives that are easily avoided using ML, primarily due to the ambiguity of several natural language terms. For example, the term "lock" can mean a phone keypad lock as well as a door lock depending on the usage. Similarly, regexes cannot capture unique terms, *e.g.*, the "Wyze Cam" would not be captured with keywords such as "camera". Thus, a regex-based search that conforms to a known set of manually-curated keywords limits data collection, both (i) in terms of identifying mobile-IoT apps, as well as (ii) analyzing their metadata further to learn about the products they support. In contrast, IoTSpotter's ML approach learns features observed in mobile-IoT metadata, and develops a general characterization that allows it to classify mobile-IoT and non-IoT apps at scale, and moreover, enables precise analysis of the identified mobile-IoT metadata to extract rich information (*e.g.*, 65k IoT products extracted using NER, Section 4.2) that regular expressions would fail to obtain from unstructured text.

To train IoTSpotter's mobile-IoT app classifier, we evaluated several metadata objects and discovered that app descriptions offer the most distinguishing source of features, as we elaborate in Section 3.1. As shown in Figure 1, IoTSpotter crawls for app metadata from Google Play, which it then classifies into mobile-IoT and non-IoT, resulting in a market-scale snapshot of mobile-IoT apps (which are also downloaded from a mirror of the market, *i.e.*, Androzoo [16]). IoTSpotter then analyzes the app descriptions in the snapshot using its NER model (Section 4.2) to identify the *supported IoT products* in each mobile-IoT app, to study the vulnerabilities in mobile-IoT apps in the context of the products they represent. To demonstrate the utility of this snapshot, we perform three types of security analyses (Section 5–Section 7). Our security analysis culminates in a systematic case study (Section 8) that assesses the impact of the discovered vulnerabilities on IoT security.

3.1 Mobile-IoT App Identification

Mobile-IoT apps are a unique class of mobile apps developed to support IoT devices/services, and may exhibit distinguishing features that can help with their classification/identification at scale. We focus on the *metadata* associated with the app, as it is intended to convey the purpose of the app to end-users, and because the actual code itself may show significant overlap between the two classes of apps given the vast amount of glue and UI code needed for mobile apps in general. However, the question is, *which of the several metadata objects (e.g., description, reviews, title) are the most suitable for distinguishing between mobile-IoT and non-IoT apps?*

To answer this question, we performed a preliminary investigation of app metadata, wherein we studied 776 apps that were available as companion apps to integrations obtained from the "Works with Google Assistant" [13] smart home platform. We found that mobile app descriptions generally use IoT keywords indicating the device types (*e.g.*, light bulb) and IoT phrases (*e.g.*, automation, remotely control), whereas, other features such as product title, permissions, reviews, or icons did not consistently reveal IoT-related characteristics. For example, titles for apps such as "Nest" [34] or "Vivint" [35] do not convey any particular information that consumers may associate with smart homes, other than the brand name itself (which a classifier may not be able to distinguish from other brands). Similarly, we observed that app permissions and user-reviews do not generally provide distinguishing information, *i.e.*, most user-reviews only discussed experience with the app, and only few apps requested wearable permissions [68]. We also analyzed the UI text from manys apps as it may carry IoT-specific information (although it is not strictly metadata), and found that UI text may contain IoT-specific phrases such as "Add Device". Thus, we decided to evaluate *app descriptions* and *UI text* as potential sources of features for IoTSpotter's classifier.

Table 1: Mobile-IoT App Classifier Performance.

	Description								UI Text				
Performance	LR	SVM	NB	RF	RNN	LSTM	BiLSTM	BERT	LR	SVM	NB	RF	CNN
Accuracy	0.927	0.916	0.914	0.926	0.947	0.946	0.952	0.957	0.625	0.637	0.608	0.620	0.543
Precision	0.932	0.897	0.909	0.929	0.925	0.957	0.962	0.949	0.675	0.67	0.639	0.688	0.545
Recall	0.896	0.908	0.889	0.896	0.954	0.915	0.925	0.951	0.530	0.586	0.553	0.483	0.707
F1-Score	0.914	0.902	0.899	0.913	0.939	0.936	0.943	0.950	0.593	0.625	0.594	0.568	0.615

3.1.1 Collecting Apps and Metadata. We begin with 5,732,376 Google Play apps from Androzoo [16], reduced to 2,182,654 unique apps (*i.e.*, after removing multiple versions). IoTSpotter automatically crawls the metadata of each app from Google Play (US) using the package names obtained from Androzoo, leading to set of 2,182,654 apps (metadata), while also downloading the corresponding APKs from Androzoo, as it mirrors Google Play and does not rate limit.

3.1.2 Creating Mobile-IoT and Non-IoT Datasets for Training and Testing. To build a classifier, we need to construct labeled datasets of both mobile-IoT and non-IoT apps. We construct the mobile-IoT app dataset through ad-hoc, keyword-based, crawling of Google Play, in a manner similar to that adopted by prior work [44, 66]. We develop a crawling heuristic to obtain an initial set of mobile-IoT apps from Google Play, which considers keywords that represent (1) 126 Device types obtained from prior work [50] (*e.g.*, smart vacuum), (2) generic IoT keywords (*e.g.*, smart room, IoT), (3) generic device keywords (*e.g.*, smart devices, appliances), (4) popular platforms (*e.g.*, Alexa, Home Assistant), (5) IoT protocols (*e.g.*, Zigbee), and (6) common regex patterns used in mobile-IoT apps (*e.g.*, phrases containing 'control' and 'remotely'). We use Snorkel [59] to programmatically build our initial training data, where we selected app descriptions that contain at least one keyword match. We find 1,758 app descriptions with at least 3 out of 6 overlapping keywords, 8,467 with 2 keywords match and 89,508 with 1 keyword match. Out of these we selected all the app descriptions with at least 3 keyword matches and an additional 800 from the remaining set. We integrate the mobile-IoT apps obtained from this crawler with a dataset of 2,081 mobile-IoT apps from prior work [66], which after removing unavailable apps and false positives leads to an eventual set of 1,574 mobile-IoT apps. Together, our heuristic-based crawler and prior work yield us 4,132 tentative mobile-IoT apps. Additionally, we sample 3,073 random non-IoT apps from Google Play, leading to a candidate list of 7,205 IoT and non-IoT app descriptions.

We manually labeled these 7,205 app descriptions to confirm their mobile-IoT/non-IoT class, in a process that involved three of the authors with prior experience in IoT research. We established a unanimously agreed-upon *definition of what constitutes a mobile-IoT app*, *i.e.*, any app that connects to or supports an IoT device, product, or service, which is in line with prior work [44, 66]. Two authors labeled the descriptions, and the third resolved conflicts, with a calculated Cohen's Kappa score [38] of 0.976, demonstrating high inter-annotator reliability score. Our labeling led to a dataset of *3,251 confirmed mobile-IoT apps*. Similarly, we obtained *3,954 confirmed non-IoT apps*, which also included the false positives from the tentative mobile-IoT set produced by our heuristic-based crawler or prior work. We split the 7,205 labeled apps into a **training dataset** consisting of 2,837 mobile-IoT and 3,403 non-IoT apps (6,240/7,205 or 86.6%) and a **testing dataset** consisting of 414 mobile-IoT and 551 non-IoT apps (965/7205 apps or 13.4%), using stratified sampling by providing equal distribution of mobile-IoT and non-IoT apps for both training and testing, as recommended in ML literature [27].

3.1.3 Building the Mobile-IoT Classifier. We evaluated the ability of several ML approaches at classifying mobile-IoT and non-IoT apps, using features from app descriptions or UI text (obtained from `.xml` resources in APKs), extracted using TF-IDF [56] and tokenizers [5, 12]. We began with algorithms such as Logistic Regression (LR), Support Vector Machines (SVM), Naive Bayes (NB), and Random Forest (RF), and then explored deep learning techniques such as Recurrent Neural Network (RNN), Long Short Term Memory (LSTM), and Bi-directional LSTM (BiLSTM), and BERT [4], which are known for their performance in text classification.

Table 1 shows the performance of classifiers trained on the training set (on top of the pre-trained model [4] for BERT), and evaluated on the testing set. Descriptions fare much better at distinguishing mobile-IoT apps from non-IoT, relative to UI text, with a difference of 30-40% in precision and recall for all classifiers. Hence, we select descriptions as IoTSpotter's sole source of features. Further, BiLSTM and BERT perform best, with a similar overall performance (F1 score of 94.3% and 95.0% respectively). As we want to prioritize precision to generate a reliable snapshot of mobile-IoT apps, we adopt the hard-voting [37] approach, *i.e.*, IoTSpotter marks an app as mobile-IoT *only if there is consensus* among both models.

3.2 Extracting IoT Product Information

As mobile-IoT app descriptions often discuss supported IoT products, we develop a Named Entity Recognition (NER) model that identifies IoT products in description-text, and enables us to understand the implications of our security analysis of mobile-IoT apps in terms of the devices affected (*e.g.*, security cameras, door locks).

3.2.1 Preparing and Annotating Descriptions. We begin by annotating mobile-IoT app descriptions that can be used to train the model. For this, we selected 600 random mobile-IoT apps, and preprocessed their descriptions (*e.g.*, discarded incomplete statements, normalized Unicode characters), which led to to 3,961 sentences for training. We used prodigy [54] to manually annotate the sentences with IoT product entities, defined as any entity that represents a device/service/*thing* that can be controlled/connected.

3.2.2 Training the NER model. We train and test the NER model by splitting the dataset of 3,961 sentences into 3,169 (80%) for training set and the remaining 972 (20%) for the hold-out test set. We use the methodology similar to prior work (*i.e.*, PolicyLint [17]), and train our model on top of the existing Spacy stock model [60]. Spacy's training process iteratively compares the predictions against the annotations to update the weights so that the predictions become more similar to the reference labels (using back-propagation [32]). Our approach results in IoTSpotter's NER model, which has 82.84% precision and 83.04% recall in identifying IoT product entities.

3.2.3 Clustering of product entities to identify common types. Upon deploying our NER model on the entire mobile-IoT app snapshot, we identified 65,676 unique IoT product entities (see Section 4.1), because IoT products are often expressed using different names e.g., outdoor camera, cctv camera, or <company_name> outdoor cameras. To understand the *types of products* supported by mobile-IoT apps, we perform short text clustering on IoT product entities using GSDMM library [36, 69]. We set the the upper bound cluster value (k) to 1000 to allow the algorithm to converge on diverse IoT product clusters considering semantically-similar entity-names.

4 APPLYING IOTSPOTTER TO 2 MILLION GOOGLE PLAY APPS

This section describes our application of IoTSpotter on the entire Google Play dataset consisting of 2,182,654 apps, and quantitative as well as qualitative insights generated during the process.

4.1 Characteristics of the Mobile-IoT snapshot

IoTSpotter's BERT-based and BiLSTM classifiers identified 58,859 and 69,270 app descriptions as mobile-IoT, respectively. Using consensus between both models to obtain high precision (as described in Section 3.1), IoTSpotter identified 37,783 mobile-IoT apps.

Result 1 ($\mathcal{R}_1$) – IoTSpotter identifies 37,783 apps out of 2,182,654 Google Play apps, the largest such set to date.

To evaluate the effectiveness of IoTSpotter in deployment, we further manually evaluated a random subset of this mobile-IoT snapshot, consisting of 2,250 apps (*i.e.*, ≈6%). The time required to label new apps was the main constraint on the size of this set.

Result 2 ($\mathcal{R}_2$) – Manual validation of 2,250 mobile-IoT apps identified by IoTSpotter shows that 88.4% (1989/2250) are indeed mobile-IoT.

Thus, we demonstrate that our mobile-IoT snapshot is reliable, and that IoTSpotter's classification approach yields an 18x increase in the obtained mobile-IoT apps in comparison with ad-hoc crawling (*e.g.*, the 2k apps obtained by [66]), at the cost of some precision. Upon further analysis, we find that most of the false positive cases contain ambiguous statements containing keywords relevant to IoT used in non-IoT contexts that are hard to disambiguate; *e.g.*, the "Pocket Cloudwatcher" app [53] mentions that an " ..*alarm* will fire if your phone stops receiving data". Other examples of false positives include apps that discuss IoT data such as blood pressure, heart and health records but do not connect to any health/wearable devices (*e.g.*, "Blood Pressure Diary" [20]).

A series of measurements on the metadata of apps in the mobile-IoT snapshot allows us to glean some of its general characteristics. As shown in Figure 2, the CDFs of the popularity/install-count distributions of the mobile-IoT snapshot and the general Google Play dataset of 2 million apps are strikingly close, indicating that mobile-IoT apps trend similarly in terms of popularity as the rest of the market; *e.g.*, the 2M Google Play app set and 37K mobile-IoT app set have 88.7% and 85.76% apps with less or equal than 10K downloads. However, we find that mobile-IoT apps are generally concentrated in a few key app categories (*e.g.*, tools, lifestyle, health); *e.g.*, several mobile-IoT apps (35.5%) fall in the tools category (relative to only

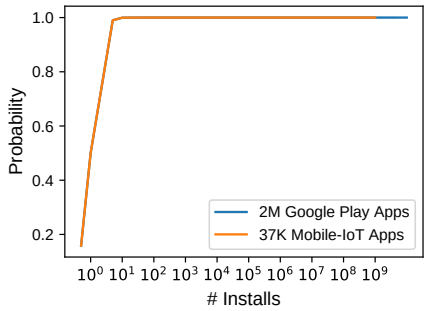


Figure 2: CDF of the popularity distribution (using install count) of mobile-IoT and general Google Play apps.

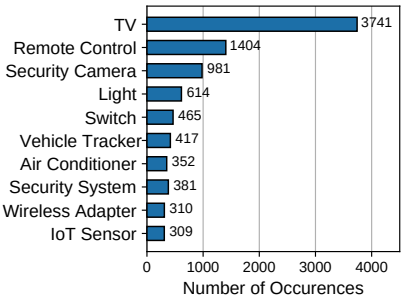


Figure 3: Top 10 IoT Product Clusters identified in IoT Apps

5.77% of Google Play apps in the same) (see the online appendix [39] for a figure showing this distribution). This statistical observation leads to the following result:

Result 3 ($\mathcal{R}_3$) – While mobile-IoT apps trend similarly in terms of popularity as the rest of the market, they are heavily skewed towards a few app categories, in contrast with the general market that is relatively evenly distributed across categories.

4.2 Products in the mobile-IoT snapshot

IoTSpotter's domain-adapted NER model analyzed the entire set of 37,783 mobile-IoT apps in the snapshot, and identified *65,676 unique IoT product entities*. The GSDMM-based approach allows us to characterize these entities, by clustering semantically-similar entities into 962 clusters representing *product types*; *e.g.*, a type "light", which includes *Neusmart lighting* and *Opple smart lighting devices* (see the online appendix [39] for additional examples).

Result 4 ($\mathcal{R}_4$) – IoTSpotter identifies 65,676 product-entities, clustered into 962 types, in the mobile-IoT snapshot.

Figure 3 shows top 10 IoT product types represented by clustering of the unique products identified by IoTSpotter's NER model. The largest such cluster represents the product type "TV", containing over 3,271 unique mentions of products of that type (*e.g.*, "Philips TV", "HiSense TV"). It is interesting to note that a prior network-based analysis of devices used in homes by Kumar et al. [47] also found TVs to be the most common of all IoT devices, which indicates a *correlation between the device types most supported by mobile-IoT apps, and those in use in the wild.*

Further, the mean and median cluster sizes, among our total 962 clusters, are 68.27 and 47, respectively, the latter of which demonstrates that significant groupings/product types exist far beyond the most populous/top clusters. Examples of clusters of average size (≈ 68 entities each) include dimming systems (which includes devices such as the "Leviton dimmer" and "Sensor3 Dimmers"), and smart glasses (which includes devices such as "Vue Glasses"). Similarly, examples of median-sized clusters (≈ 47 product entities) include smart bands and GPS trackers. Moreover, we observed that the smallest clusters containing very few products (*i.e.*, 1-5 products each) were often the result of obscure product/brand names that did not contain any information indicating its type, *e.g.*, eqb-600 [29], which is a smart watch. Finally, as we observe in Figure 3, several of the top-10 clusters indicate devices that are security or privacy sensitive, even by a conservative standard.

Result 5 ($\mathcal{R}_5$) – Security and privacy sensitive device types such as security cameras, security systems, vehicle trackers, and TVs form some of the most common devices supported by mobile-IoT apps in the snapshot.

5 MOBILE-IOT LIBRARY ANALYSIS

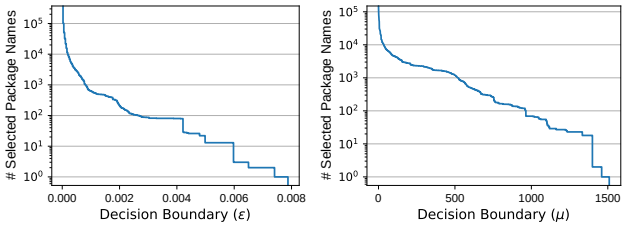
The security characteristics of IoT-specific third-party libraries and SDKs are understudied, relative to general-purpose Android libraries categorized on platforms such as Appbrain [3]. We explore a simple but fundamental question: *How do IoT-specific libraries impact the security of mobile-IoT apps?* To answer this question, we develop a method for automatically extracting IoT-specific libraries, analyze the libraries for known vulnerabilities, and identify the use of vulnerable IoT libraries in mobile-IoT apps.

5.1 Methodology

A naive approach for collecting IoT libraries used in our mobile-IoT apps would be to crawl popular repositories (*e.g.*, Maven, Github), identify IoT-specific libraries from their descriptions, and map them to the libraries in our mobile-IoT snapshot. However, we observe that online repositories do not consistently contain clear descriptions of libraries. Thus, we develop an automated approach that uses *differential analysis* to identify IoT libraries, *i.e.*, determines the relevance of a library to IoT functionality by comparing the frequency of its use in our mobile-IoT snapshot, and non-IoT apps.

1. Collecting Third-party Library Packages: We extract third-party library package names from mobile apps using LibScout [19]. For this, we first construct package trees based on the code structure. Second, we use app manifest file and the Android framework SDK (android.jar) to identify first-party libraries and framework API libraries, respectively. Finally, we identify library package names using LibScout, excluding first-party and framework API libraries. Using this approach, we identified *522,285* library packages in the mobile-IoT snapshot. Given the infeasible manual effort that would be required to identify IoT-specific libraries from this set, we develop the automated differential analysis approach discussed next.

2. Identifying IoT Library Packages: We develop a differential analysis approach that identifies library package names specific to IoT, *i.e.*, which are more commonly used by mobile-IoT apps.



(a) Decision Boundary (ϵ) **(b) Decision Boundary (μ)**

Figure 4: The number of selected package names against decision boundaries ϵ in Equation 2, and μ in Equation 3.

That is, we compare third-party libraries used in our snapshot of 37,783 mobile-IoT apps against the remaining 2,144,871 non-IoT apps. While a simple comparison can yield the difference between the two sets of library package list (*i.e.*, by excluding libraries that are common across mobile-IoT and non-IoT library set), we develop a more precise characterization, *i.e.*, to identify libraries that are (1) *popular among mobile-IoT apps*, if only used in our mobile IoT app snapshot, and (2) *significantly more popular in the mobile-IoT snapshot relative to non-IoT*, if also used in non-IoT apps.

We first define the notion of popularity for a library i from a set of libraries L, in a set of apps R containing N apps. For each library $i \in L$, f_i^R denotes the number of apps in R using the library i. We define the popularity p_i^R of the library i in the app dataset R as:

$$p_i^R = \frac{f_i^R}{N}, i \in L \tag{1}$$

We now describe our approach for defining decision boundaries, that allow us to identify libraries that are exclusive to the mobile-IoT snapshot and popular in it, as well as libraries that are common among IoT and non-IoT apps, but significantly more popular in the former. To elaborate, consider a library set L^I containing all the libraries used in the mobile-IoT snapshot R^I, and L^O from non-IoT app dataset R^O. In Equation 2 and Equation 3, p_i^I and p_i^O denote the popularity of library i in app set R^I and R^O. We define two decision boundaries: (i) ϵ, for identifying popular IoT-specific libraries (*i.e.*, those only found in mobile-IoT apps), and (ii) μ, for identifying libraries common among both sets that are significantly more popular (hence relevant) to IoT. We now define Equation 2 and Equation 3, which use these decision boundaries to identify popular IoT-specific libraries, and those common libraries that are far more relevant to IoT, respectively, as follows:

$$p_i^I > \epsilon, i \in L^I, i \notin L^O \tag{2}$$

$$\frac{p_i^I}{p_i^O} > \mu, i \in L^I, i \in L^O \tag{3}$$

To understand the distribution of common package names, we calculate the number of package names that will be selected based on a given decision boundary. As shown in Figure 4a, the common package count continuously decreases as we continue to increase the decision boundary. For our analysis, we seek to choose decision boundary parameters that identify larger set of library package names without compromising on popularity. That is, a high decision boundary will result in a high precision (*i.e.*, libraries popular among IoT apps) but low recall. We see a similar trend in Figure 4b, *i.e.*, the

package count continues to decrease as we increase the decision boundary, which means that there are very few library packages that are extremely popular in mobile-IoT set (relative to non-IoT).

Analyzing IoT Library Packages for Vulnerabilities: *To understand what vulnerabilities IoT libraries introduce in mobile-IoT apps*, we crawl the Maven Repository [14] to identify vulnerable library versions of libraries in our set. Note that we investigated alternative sources such as library websites and Github repositories, but observed that the Maven repository is a better resource, as it contains CVE badges for vulnerable library versions and has a consistent layout that facilitates automated analysis. We then check for the vulnerable library versions in apps using LibScout.

5.2 Results: IoT Library Identification, Characterization, and Security Analysis

We obtained 522,285 third-party library package names from 37K mobile-IoT apps. By choosing the decision boundary (ϵ = 0.0002), we select all the packages which are used by at least 7 apps, and identify 11,242 library package names. Similarly, we used a decision boundary (μ) of 44, *i.e.*, considered library package names that are 44 times more popular in the mobile-IoT snapshot compared to non-IoT apps, and identified 8,697 library package names. Appendix A describes the intuition behind our choice of decision boundary. Together, this resulted in 19,939 library package names that are significantly more popular among mobile-IoT apps than non-IoT apps, making them important for IoT security analysis.

Result 6 ($\mathcal{R}_6$) – IoTSpotter's differential analysis approach identified 19,939 IoT library package names.

Note that a naive approach of identifying IoT libraries by solely searching for the package name in online repositories (*e.g.*, Maven) and analyzing the description would not be effective. To elaborate, crawling Maven to search for descriptions of the 522,285 third-party library package names present in mobile-IoT apps resulted in only 4.8% (i.e., 25,321/522,285) matches. Furthermore, when we parsed the pom file of 25,321 jar files to extract the library description, we found that only 40% jar files contained descriptive text about the library, which would essentially yield descriptions of 2% of the library 520k package names found in mobile-IoT apps, analyzing which would lead to an even smaller set of eventual IoT libraries. In contrast, our approach of comparing the usage of libraries in mobile-IoT vs non-IoT apps allows us to identify over 19k library packages that are widely more popular in mobile-IoT apps than non-IoT, indicating a specific relevance to IoT, without any manual effort. Next, we characterize the most popular packages among these IoT libraries to identify *families*.

Characterizing top IoT libraries into *families*: To characterize the libraries obtained from our differential analysis, we further selected the *top 50 IoT library package names* that were used by maximum number of apps, and used them to identify library *families*. For this, we manually searched the library package names on search engines and clustered prefixes to create library families, *i.e.*, packages with the same prefix, resulting in the discovery of 11 such library families. We further searched for the common library prefixes representing each family in our mobile-IoT snapshot, which led to the discovery of the use of several packages belonging to each family in a large number of mobile-IoT apps. The top library families are described in Table 2.

As shown in Table 2, 10/11 of the library families provide functionality associated with IoT services, except `javax.jmdns`, which helps with DNS (useful for IoT, but not exclusively). This speaks to the effectiveness of our approach for identifying IoT libraries.

Result 7 ($\mathcal{R}_7$) – Our differential analysis approach is effective in isolating IoT-related libraries, as it leads to the discovery of 10/11 most popular library families (among apps in the mobile-IoT snapshot) that provide IoT-relevant functionality.

Vulnerabilities in IoT and non-IoT libraries: To understand the security characteristics of IoT libraries in terms of the CVEs associated with individual library packages, we crawled the Maven repository with all 19,939 IoT library package names identified by IoTSpotter. To perform a comparative analysis of *general mobile app libraries*, *i.e.*, all libraries barring the 19k IoT libraries, we would need to similarly crawl Maven for over 1.2 million general/non-IoT library packages found in our 2 million non-IoT apps. However, as Maven employs strict rate limits, crawling for all 1.2 million non-IoT library packages would be infeasible (*e.g.*, crawling for 19k IoT libraries took around 3 weeks). Therefore, we instead sample 5000 non-IoT library packages in two even subsets that balance impact with the need for an unbiased analysis: (i) 2,500 most used library packages (among all non-IoT apps), and (ii) 2,500 randomly sampled package names from the 1.2 million non-IoT library packages.

Upon crawling the Maven repository, we identified several IoT libraries to be vulnerable, leading to the following finding.

Security Finding 1 ($\mathcal{F}_1$) – 65 IoT libraries (481 unique versions) are vulnerable, *i.e.*, subject to 79 CVEs.

On repeating the analysis for non-IoT libraries, we found 73 vulnerable libraries (7,203 versions), *i.e.*, the subject of *193 CVEs*.

Security Finding 2 ($\mathcal{F}_2$) – The 19k IoT libraries are less vulnerable relative to 5,000 non-IoT libraries, with the top 2500 non-IoT libraries contributing significantly more vulnerabilities (193 CVEs, 63 libraries, 7,105 versions), relative to the the random 2,500 (7 CVEs, 10 libraries, 98 versions).

Table 3 lists the top 10 CVEs in IoT and non-IoT libraries.

Use of vulnerable libraries in apps: We now describe our analysis of the *use* of vulnerable IoT and general libraries in mobile-IoT and non-IoT apps, respectively. To identify the use of vulnerable IoT libraries, we used LibScout to analyze all mobile-IoT apps from our snapshot with >50k installs, *i.e.*, a total of 5,380 popular apps. For a comparable analysis in the non-IoT context, we randomly sampled 5,380 non-IoT apps, repeatedly sampling until we found a set with the same popularity distribution (as indicated by its CDF of installs) as the 5,380 mobile-IoT apps (see the online appendix [39] for a detailed description of the sampling process).

In mobile-IoT apps, we found 29 vulnerable libraries being used, of which we discarded 2/29 that are not used for IoT-specific functionality. Upon exploring the apps using LibScout, and identified

Table 2: Top IoT third-party library families.

Library Family	Functionality	# Apps	# Packages	# Classes	# Versions	Developer
com.tuya	IoT framework	1,362	4,427	71,742	1,379	Tuya, China
no.nordicsemi.android	BLE & firmware services	1,097	324	6,284	192	Nordic Semiconductor, Norway
javax.jmdns	DNS services	852	24	347	24	Hoff et. al.
com.amazonaws.mobileconnectors	IoT cloud services	751	110	1,372	211	Amazon Web Services, USA
com.connectsdk	Device control	378	69	2,041	24	Frolov et. al.
com.inuker.bluetooth	BLE services	358	106	1,488	42	dingjikerbo et. al.
com.clj.fastble	BLE services	333	21	351	11	Chen et. al.
com.hiflying	Device control	285	36	323	1	High-Flying, China
com.telink	Device control	250	95	1,631	3	Telink, China
com.hikvision	Device control	191	1,340	29,286	34	Hikvision, China
org.fourthline.cling	Device control	187	190	3,176	7	4th Line, Switzerland

Table 3: Top CVEs affecting vulnerable IoT and non-IoT/general mobile libraries. Top CVEs affecting both sets are underlined.

IoT Libraries		Non-IoT Libraries	
CVE ID	# Libraries	CVE ID	# Libraries
CVE-2020-15250	31	CVE-2017-18349	43
CVE-2022-25647	15	CVE-2020-15250	13
CVE-2022-24329	7	CVE-2021-35515	12
CVE-2020-29582	7	CVE-2021-35516	11
CVE-2021-29425	7	CVE-2021-35517	6
CVE-2021-4104	7	CVE-2021-36090	5
CVE-2020-8908	5	CVE-2020-8908	5
CVE-2020-15522	5	CVE-2020-25649	5
CVE-2022-23302	4	CVE-2020-13956	5
CVE-2022-23305	3	CVE-2020-36179	5

92 mobile-IoT apps that use the remaining 27/29 libraries. We manually validated this result and identified 52 false positives (due to LibScout's imprecision), and 40 true positives, *i.e.*, 40 mobile-IoT apps that indeed use vulnerable IoT libraries.

Security Finding 3 ($\mathcal{F}_3$) – 40 popular mobile-IoT apps are vulnerable because of vulnerable IoT library usage, out of which 6 have more than 1 million installs.

Similarly, we identified 73 vulnerable non-IoT libraries being used by 777 popular non-IoT apps. Manual validation revealed 268 false positives, and confirmed that 509 non-IoT apps indeed used vulnerable libraries, of which 73 have >1 million installs.

Security Finding 4 ($\mathcal{F}_4$) – The use of vulnerable non-IoT libraries in popular (>50k install) non-IoT apps is *significantly higher*, *i.e.*, 12.7X (509/40) apps, relative to the use of vulnerable IoT libraries in mobile-IoT apps.

There are three possible reasons for this result: First, as $\mathcal{F}_2$ shows, popular non-IoT libraries are generally more vulnerable than IoT libraries, and hence, popular but vulnerable non-IoT libraries affect several non-IoT apps; *e.g.*, all 1,441 versions of the popular com.amazonaws/aws-java-sdk-core library [18] are affected by at least 6 CVEs, compounding its impact on apps. Second, the use of top general/non-IoT libraries in non-IoT apps significantly exceeds the use of top IoT libraries in mobile-IoT apps, thus amplifying vulnerability-impact on the former, *e.g.*, the com.google.firebase library is imported by 68.5% non-IoT apps in the entire non-IoT app set (2 million apps), whereas the most popular IoT-specific library, no.nordicsemi.android/dfu, is used in 2.9% of our 37K mobile-IoT apps. Finally, mobile-IoT apps may use non-IoT libraries, but our analysis of the presence of vulnerable IoT libraries in mobile-IoT apps naturally excludes non-IoT libraries (hence reducing their impact on the security posture of mobile-IoT apps). This decision is justified as our original goal is to characterize the security *impact of vulnerable IoT libraries on mobile-IoT apps*. That said, as mobile-IoT apps are concentrated in some app categories (*e.g.*, over 35.5% in tools) relative to the more even distribution of non-IoT apps, and are on average much simpler than non-IoT apps (*e.g.*, do not use social network SDKs, or complex UIs), we believe that they may only use a few general/non-IoT libraries.

Finally, the vulnerabilities due to IoT libraries can lead to serious security implications in the IoT context, as we explore in Section 8.

6 CRYPTOGRAPHIC API MISUSE ANALYSIS

The correct use of cryptography is key data security in modern software. Particularly in the case of mobile-IoT apps, misuse of crypto-APIs can lead to severe vulnerabilities that may allow attackers to steal or modify security-critical data (*e.g.*, authentication tokens, credentials, firmware), and use it to escalate privilege to devices and the overall IoT system (see Section 8). Thus, we perform a systematic analysis to understand the prevalence and nature of crypto API misuse vulnerabilities in popular mobile-IoT apps.

6.1 Methodology

We analyze mobile-IoT apps using two static analysis security tools that specialize in detecting crypto-API misuse, CryptoGuard [55] and CogniCrypt [46], chosen as they are currently maintained and popular in practice; *e.g.*, CryptoGuard is a part of Oracle testing suite [52], while CogniCrypt is available as an Eclipse plugin.

While we would want to analyze all 37k apps from the mobile-IoT snapshot, we are limited by the runtime performance and detection ability of existing analyses. To elaborate, we observed that CryptoGuard needs 8 minutes on average to analyze one app, significantly higher than CogniCrypt's 0.67 minutes; hence, analyzing all 37k apps with CryptoGuard would be prohibitively time consuming. However, we cannot choose to abandon CryptoGuard for performance and only run CogniCrypt, given CryptoGuard's more recent development (and ruleset), which may lead it to find a larger and diverse set of vulnerabilities relative to CogniCrypt (as we indeed find in Section 6.2). Therefore, for a *tractable* and *impactful* analysis that balances the need to find vulnerabilities while also considering runtime performance, we choose to analyze a sample of *popular mobile-IoT apps* from the snapshot, *i.e.*, those with >1 million installs. Our sample consists of 917 mobile-IoT apps. We manually validated these 917 apps to confirm 91.385% (*i.e.*, 838/917) are indeed mobile-IoT, which is slightly higher than the 88.4% true positive rate for a random validated sample from our snapshot (see $\mathcal{R}_2$).

Table 4: The number of mobile-IoT (and non-IoT) apps that violate CryptoGuard's rules, *i.e.*, contain at least one vulnerability/violation of each rule-type (*sorted by # vulnerable mobile-IoT apps*). CryptoGuard assigns *severity* to rules, as we annotate in the table (high severity= [H], medium severity= [M], low=unmarked).

CryptoGuard's Rules (IDs as per [55])		# Vulnerable Apps	
ID	Rule Name	Mobile-IoT	Non-IoT
9	Insecure PRNGs (e.g., java.util.Random) [M]	842	870
16	Insecure cryptographic hash (e.g., SHA1, MD5) [H]	825	865
1	Predictable/constant cryptographic keys [H]	577	669
7	Occasional use of HTTP [H]	438	441
14,11	*64-bit block ciphers (e.g., DES, RC4), ECB mode [M]	406	376
5	Custom TrustManager to trust all certificates [H]	380	302
4	Custom Hostname verifiers to accept all hosts [H]	293	269
12	Static IVs in CBC mode symmetric ciphers [M]	239	208
6	SSLSocketFactory w/o hostname verification [H]	186	86
3	Predictable/constant passwords for KeyStore [H]	142	60
13	Fewer than 1,000 iterations for PBE	70	26
15	Insecure asymmetric cipher use	66	19
2,10	*Predictable passwords, static salts in for PBE [H/M]	63	47
8	Predictable/constant PRNG seeds [M]	50	23
-	**Number of apps that violated at least one rule**	863	883

* = CryptoGuard reports combined results for rules indicated by combined rule IDs.

To understand how mobile-IoT apps fare relative to non-IoT in terms of crypto-API vulnerabilities, we repeated the analysis on 917 popular non-IoT apps randomly sampled out of 41,041 non-IoT apps with >1 million installs, by repeatedly sampling until we found a set with the same popularity distribution as indicated by its CDF of installs (see the online appendix [39] for details).

Experiment Setup: Our analysis was carried out on an 8-core Ubuntu machine with 32 GB RAM. CryptoGuard took 8.22 minutes on average to analyze each app (*i.e.*, in both the mobile-IoT and non-IoT sets), and 251.38 hours in total, whereas CogniCrypt took 0.67 minutes on average, and 20.46 hours in total. We aggregated the results provided by both CryptoGuard and CogniCrypt and built parsers to map the identified misuse to violating method/statements in apps, as well as the justification provided by the tools.

6.2 Results: Cryptographic API Misuse

Tables 4 and 5 illustrate the number of unique apps that were found vulnerable to crypto-API misuse by CryptoGuard and CogniCrypt, respectively, omitting 17/44 of CogniCrypt's rules that no app violated.Across all rules, CryptoGuard finds vulnerabilities in 863/917 mobile-IoT apps, and 883/917 non-IoT apps, while CogniCrypt finds vulnerabilities in 599/917 mobile-IoT and 658/917 non-IoT apps. Further, CryptoGuard finds 38,486 rule violations across all 917 mobile-IoT apps, and 40,218 violations across all 917 non-IoT apps, while CogniCrypt finds 4,686 and 4,363 rule violations across mobile-IoT and non-IoT apps, respectively, as shown in Tables 7 and 8 in Appendix B.

Security Finding 5 ($\mathcal{F}_5$) – 94.11% (863/917) of the mobile-IoT apps with over 1 million installs have at least one crypto-API misuse vulnerability as detected by CryptoGuard, which is approximately equal to the fraction of non-IoT apps flagged as vulnerable, *i.e.*, 96.29% (883/917).

This finding could mean that popular mobile-IoT apps are roughly as vulnerable as a proportionate sample of non-IoT apps. Prior work

Table 5: The number of mobile-IoT (and non-IoT) apps that violate CogniCrypt's rules, *i.e.*, contain at least one vulnerability/violation of each rule-type (*sorted by # vulnerable mobile-IoT apps*).

CogniCrypt's Rules (IDs as per [46])		# Vulnerable Apps	
ID	Rule SPEC	Mobile-IoT	Non-IoT
36	MessageDigest	492	619
10	javax.net.ssl.SSLContext	186	157
2	javax.crypto.Cipher	161	119
1	javax.net.ssl.TrustManagerFactory	123	110
43	java.security.Signature	114	79
34	javax.crypto.spec.SecretKeySpec	103	68
32	javax.crypto.spec.IvParameterSpec	74	56
16	javax.crypto.SecretKeyFactory	60	38
39	javax.crypto.spec.PBEKeySpec	47	26
12	SSLSocketFactory	28	5
4	java.security.KeyPairGenerator	27	16
25	javax.crypto.Mac	27	12
22	java.security.KeyStore	24	14
24	javax.crypto.KeyGenerator	16	10
37	javax.crypto.CipherInputStream	13	10
21	javax.net.ssl.KeyManagerFactory	12	1
30	javax.crypto.CipherOutputStream	9	7
31	java.security.SecureRandom	7	5
41	javax.crypto.spec.GCMParameterSpec	6	3
15	java.security.DigestOutputStream	5	5
35	javax.crypto.spec.PBEParameterSpec	5	4
26	SSLSocket	4	1
5	java.security.cert.TrustAnchor	2	1
3	java.security.AlgorithmParameters	1	0
8	javax.net.ssl.SSLParameters	1	0
17	java.security.DigestInputStream	1	2
27	java.security.cert.PKIXBuilderParameters	1	0
-	**Number of apps that violated at least one rule**	599	658

by Kafle et al. [44] has observed a similar trend in the (narrower) context of SSL/TLS vulnerabilities, *i.e.*, that vulnerability prevalence in a small set of mobile-IoT apps is similar to that in a prior measurement of general Android apps by Fahl et al. [30]. Alternately, there is one other explanation: that we simply drew a *short straw* when sampling 917 non-IoT apps from the 41,041 non-IoT apps with >1 million downloads, *i.e.*, collected apps that were more vulnerable (whereas the 917 mobile-IoT apps are *all* mobile-IoT apps with million+ downloads). Given that we ensures a proportionate non-IoT sample in terms of the distribution of installs, the only way to avoid such a possibility would be to analyze all 41,041 non-IoT apps with >1 million installs, which is infeasible (would take roughly 234 days in compute time for CryptoGuard). However, $\mathcal{F}_5$ makes one thing clear: despite being a newer class of apps, *mobile-IoT apps are not substantially better than non-IoT apps* in terms of using cryptography correctly, which results in serious consequences in the IoT context, as we further discuss in Section 8.

High and Medium severity vulnerabilities: CryptoGuard assigns a severity to each rule based on the potential risk, in terms of how easily the attacker may exploit the associated vulnerability, and the potential impact. We leverage this additional context to further understand the nature of crypto-API misuse in mobile-IoT apps, using the high [H] and medium [M] severity rules annotated in Table 4. Particularly, we observe that 2/3 top rules with the most vulnerable apps are high severity, and overall, most apps violate at least one high severity rule (*i.e.*, rule #16, with 825 vulnerable apps). The significant prevalence of bad cryptographic hashes (*e.g.*, MD5, SHA1) (rule #16, H) can be easily exploited to violate integrity of protected objects (*e.g.*, signatures, when used in conjunction with

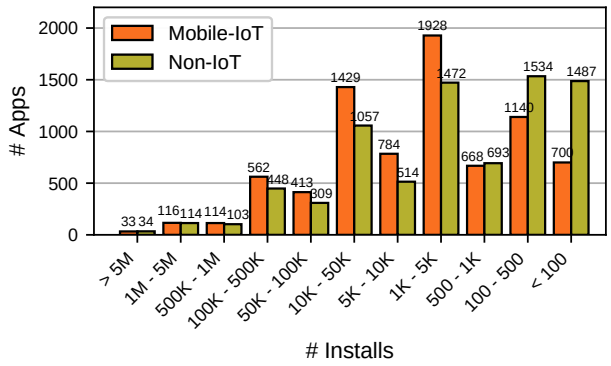


Figure 5: The number of mobile-IoT and non-IoT apps with the Janus vulnerability, distributed over the install count.

SSL/TLS). Similarly, constant cryptographic keys (rule #1, H) can be leveraged to steal confidential data in transit or at rest (*e.g.*, authentication tokens, user credentials). Similarly, we find high severity SSL/TLS vulnerabilities, *e.g.*, hostname verifiers that accept all hosts (rule#4, H), or custom trust manager's that trust all certificates (rule#5, H) prevalent in mobile-IoT apps, which is concerning as these vulnerabilities have been leveraged using Man-in-the-middle (MiTM) attacks to escalate privilege to connected IoT devices [44].

Several vulnerable mobile-IoT apps support security and privacy-critical devices. For example, *Eye4* (1 million installs), which allows remote control of video systems, has 293 violations (CryptoGuard), and violates 10 of its rules. Similarly, TP-Link's Tether app (10 million installs), which allows access to TP-Link's routers and range extenders, has 81 violations (according to CogniCrypt). Our case study explores the IoT security implications of these vulnerabilities (Section 8). Next, we leverage the severity annotations from CryptoGuard to validate a meaningful sub-sample of its results.

Manually Validating CryptoGuard's results for mobile-IoT: The crypto-API misuse violations detected in mobile-IoT apps are only significant if they are *true positives*, which must be manually validated. As it is infeasible to manually validate the thousands of violations generated in our analysis, we select a sample of the violations to validate. Particularly, CryptoGuard's severity-annotations provide us with a criteria to sample violations to validate. To balance the volume of violations as well as popularity, we selected 10 mobile-IoT apps for validation, split between 5 mobile-IoT apps with highest number of violations reported by CryptoGuard (*e.g.*, Eye4 [63], 293 violations), and 5 apps with highest install counts (e.g., Samsung Health, 1 billion+ installs). Using Jadx [28] to decompile the APKs, we manually analyzed 589 high [H] severity violations found in the aforementioned 10 apps (after removing obfuscated cases), by confirming their use from within the app, *i.e.*, that the method containing the violation is invoked, and there is some valid control flow from the method's invocation to the violation. We found that a majority of the violations were indeed true positives, leading to the following result.

Security Finding 6 ($\mathcal{F}_6$) – 82.5% (*i.e.*, 486/589) high severity violations detected by CryptoGuard in popular mobile-IoT apps are true positives, *i.e.*, used/called from within the apps.

7 APP SIGNATURE ANALYSIS

The Android package (APK) file is a special Java archive file containing all app files needed for execution, such as resource and dex files. To protect the integrity of these files, the JAR signing scheme v1 was introduced to sign all files and store signature files in META-INF folder. This folder contains three files listing file hash results: the MANIFEST.MF, CERT.SF and CERT.(RSA|DSA|EC). However, the metadata files in META-INF folder are not protected by the JAR signing scheme, which makes it vulnerable to the Janus vulnerability (CVE-2017-13156) [11] in Android versions 5.0 to 8.0. By exploiting the Janus vulnerability that exploits the v1 scheme, an attacker can inject a malicious dex file into an APK without tampering with its signatures [64]. A countermeasure for protecting the entire APK is to use the signing schemes v2, v3, and v4, which calculate signatures based on all bytes contained within the APK.

Methodology: To understand the prevalence of the Janus vulnerability mobile-IoT apps, we analyze the signatures of all 37k apps in our mobile-IoT snapshot. Moreover, we also analyze a proportionate sample of 37,783 non-IoT apps with the same popularity distribution. To automatically check for the signing scheme used by an app, we use *apksigner* from Android SDK Build Tools [2].

7.1 Results: App Signature Analysis

We find that the number of mobile-IoT apps using the signing schemes v1, v2, v3, and v4 are 36,220, 29,512, 14,529 and 0, respectively. In comparison, the number of non-IoT apps using the schemes v1, v2, v3, and v4 are 37,103, 29,891, 14,198, and 0, respectively. Surprisingly, none of the apps use the v4 scheme, which was released for Android version 11 (and onwards) on September 2020.

Figure 5 presents the distribution of mobile-IoT and non-IoT apps with the Janus vulnerability over install counts. We find 8,172 mobile-IoT apps using the vulnerable v1 scheme, 7,887 of which support at least one Android system version between 5.0 and 8.0.

Security Finding 7 ($\mathcal{F}_7$) – 7,887 (*i.e.*, 20.87%) mobile-IoT apps are susceptible to the Janus vulnerability, of which 263 have over a million installs, and 33 have over 50 million installs.

Meanwhile, 7,816 non-IoT apps use the v1 scheme, of which 7,765 apps are vulnerable as they support an Android version between 5.0 and 8.0. As we observe in Figure 5, the prevalence of the Janus vulnerability in popular mobile-IoT and non-IoT apps (*i.e.*, those with >1 million installs) is marginally different (*e.g.*, 116 vs 114 vulnerable apps with 1 - 5 million downloads). However, as the popularity decreases, especially between 5k and 100k installs, we observe that the number of vulnerable mobile-IoT apps significantly exceeds non-IoT (*e.g.*, 1429 mobile-IoT vs 1057 non-IoT apps with 10-50k downloads). This trend reverses for apps with very few installs (*i.e.*, less than 500), *i.e.*, unpopular or new non-IoT apps are far more vulnerable in comparison to mobile-IoT.

Security Finding 8 ($\mathcal{F}_8$) – Non-IoT apps are overall similarly vulnerable to Janus as mobile-IoT apps; however, moderately popular mobile-IoT apps (5-100k installs) are far more vulnerable than non-IoT, and in contrast, non-IoT apps with low popularity (<500 installs) are far more vulnerable than mobile-IoT.

8 CASE STUDY: A CONTEXTUALIZED ANALYSIS OF IOT SECURITY

Vulnerabilities in mobile-IoT apps can lead to the compromise of other components of IoT systems, such as the integrity/functionality of physical devices [44], and the authenticity of data in the cloud back-end [72]. In this section, we perform an in-depth case study to understand what the vulnerabilities discovered in Section 5–Section 7 mean in the IoT context, in terms of their impact on the security of IoT devices and functionality.

Selecting mobile-IoT apps: We sampled 18 mobile-IoT apps from those detected as vulnerable in Section 5 –Section 7, excluding apps that were obfuscated or which failed to decompile. For maximum impact, we selected apps that satisfied (and generally exceeded) the following criteria: (1) ***Vulnerability.*** The app is found susceptible to more than one type of vulnerability, (2) ***IoT security impact.*** The app supports at least one type of security/privacy sensitive IoT device, such as security cameras and smart TVs, as identified by IoTSpotter's product identification model from Section 4.2, and (3) ***User impact.*** The app is popular, *i.e.*, impacts a large number of users, measured in terms of the install counts. Our online appendix [39] provides the full list of apps.

Methodology: We use a systematic, manual, reverse engineering methodology to confirm the reachability and IoT-related impact of the vulnerabilities in our set of 18 apps (decompiled using jadx [28]). During our analysis, we take care to exclude vulnerabilities that do not impact IoT-specific functionality in spite of being reachable; *e.g.*, `Amazon Fire TV` contains a PasswordProviderFactory class with a vulnerable TrustManager that accepts all certificates. While this TrustManager is in use in the app, we exclude it from our results as it does not impact IoT-specific functionality.

Results: We were able to connect vulnerabilities to important IoT functions/components in 16/18 apps, all with >1 million installs. We summarize these vulnerabilities, the IoT functionality impacted, and the devices affected in Table 6, which lead to two key findings.

Security Finding 9 ($\mathcal{F}_9$) – Every class of vulnerabilities investigated in popular mobile-IoT apps (*i.e.*, vulnerable libraries, crypto misuse, and Janus) impacts critical IoT functions/components, including firmware integrity, app and device functionality, app integrity, and user authentication and credentials.

Security Finding 10 ($\mathcal{F}_{10}$) – The 16 vulnerable mobile-IoT apps where IoT functionality is impacted support security/privacy-critical devices, of which *security cameras* are a majority.

We now describe the discovered impact in terms of each critical IoT function/component affected.

1. Firmware: We found 3 top apps with vulnerabilities affecting device firmware updates. The `Eye4` app uses HTTP links for *downloading firmware in plaintext*, while also not verifying the checksum of the downloaded firmware, with serious implications for the integrity of the IP cameras it supports. Similarly, `Hubble Connected for Motorola`, and `JBL Music` upgrade device firmware using HTTP links, enabling attackers to inject malicious firmware into their cameras and speakers via MiTM attacks.

Table 6: Summary of the results of the study of 18 mobile-IoT apps.

IoT Impact	Vulnerabilities	Vulnerable apps	Devices Affected
Firmware (Malicious Modification)	Crypto (HTTP, no integrity checks)	Hubble Connected JBL Music Eye4	Camera JBL Speaker IP Camera
App/Device Functions (hijack, code execution)	IoT Libraries (multiple CVEs)	SURE IP Pro Vestel Smart Center Sricam LinkSys Realme Link	PTZ Camera Smart TV Vestel Smart TV IP Camera Wi-Fi Routers Watch, Bands
User Credentials, Authentication	Crypto (MD5, TrustManager, HTTP)	EagleEyes(Lite) Cetusplay Hubble Connected	IP Camera, NVR TVs, Chromecast Camera
Admin Password Leakage	Crypto (constant password, HTTP)	Eye4	IP Camera
App Integrity (Malware)	Janus	ANT+ Plugins Service Amazon Alexa	Activity trackers Echo Devices
General Data Security	Crypto (DES, MD5, ECB mode)	Remote for Samsung TV LG ThinQ Harmony	Smart TV Washer, AC, TV Lights, Blinds, TV

2. IoT App/Device Functionality: We find 6 top mobile-IoT apps using vulnerable IoT libraries, leading to critical vulnerabilities that can allow attackers to compromise app/device functionality. For example, the `IP Pro (VR Cam, EseeCloud)` app uses the `com.aliyun.alink.linksdk` library which implements the Alink protocol [1] for app-device communication, but has several vulnerabilities (CVE-2017-18349 [6] and CVE-2019-11777 [7]). CVE-2017-18349 can be exploited to gain remote code execution on IoT devices, whereas CVE-2019-11777 allows one MQTT server to impersonate another, allowing attackers to *hijack app/device interactions* and control IoT-related functions. Similarly, Realme Link uses the com.tuya.smart/tuyasmart-bizbundle-camera and com.tuya.smart/tuyasmart-bizbundle-ota libraries for IP camera control and firmware updates, which are subject to 6 CVEs that allow attackers to attack remote servers (*e.g.*, CVE-2022-25845 [10]), perform denial of service attacks (e.g., CVE-2021-36090 [9]), and execute arbitrary code (e.g., CVE-2017-18349 [6]). Finally, the `SURE - Smart Home and TV Universal Remote` app uses the com.willblaschko.android.alexa/AlexaAndroid library to integrate with the Alexa platform, which has a vulnerability (CVE-2021-29425 [8]) that enables unauthorized access to parent directories, leaking sensitive IoT device data.

3. User Credentials and Authentication: The `CetusPlay` app uses a vulnerable X509TrustManager during the device pairing process, and hence, for exchanging sensitive data with devices/servers, which is exposed to a MiTM attack. Further, the `EagleEyes(Lite)` uses a HTTP link to load its login page. Such mixed use is vulnerable to SSL stripping, wherein an attacker can potentially replace this login page with a phishing page, compromising the user's login credentials. Similarly, `Hubble Connected for Motorola` app sends and publishes commands to IoT devices in links using HTTP, potentially leaking the device IP, port number and command strings, which the attacker can not only read, but also modify in transit, thereby impacting the integrity of the internal states of both the app and the communicating device. Finally, we the `Eye4` app uses the vulnerable MD5 algorithm to create a password hash that is then sent over HTTP, which is susceptible to replay attacks at a minimum (*i.e.*, as the hash is sent over in plaintext).

4. Admin Password Leakage. Aside from authentication issues, we observed that some apps expose the password in plaintext in the app-device communication. The `Eye4` app uses HTTP links to check device status in the `CheckStatus` and `GetWifiDeviceList` activities, wherein it uses static links in which the IoT device *admin passwords are hardcoded*. Such leakage would allow an adversary to take over the IoT device, and is particularly worrisome in a shared environment (*e.g.*, smart buildings/offices) where users on the network may not have authorized access to all devices.

5. The App's Integrity. We found top IoT apps that are vulnerable to the Janus vulnerability, such as the `Amazon Alexa` app, which only uses the `v1` signing scheme with a minimum Android v6.0 supported, making it vulnerable. This vulnerability would allow attackers to inject `dex` files without violating the signature check, and hence significantly impacts the integrity of the app, and every device it controls. Since our analysis is based on apps crawled in August 2021, we also checked the latest version of `Amazon Alexa` (as of August 2022), which is signed with both v1 and v2 schemes, and hence still vulnerable to the Janus vulnerability when installed on Android 5.0-8.0 phones. Similarly, we also find many top apps (*e.g.*, `ANT+ Plugins Service` app which has 1 billion downloads) that use only the v1 scheme with low minimum Android SDK versions ($\leq$ 5.0), which leads to them being vulnerable.

6. General Data Security. We found that `Remote for Samsung TV` uses MD5 *for validating cryptographic signatures* in responses received from its server during the device discovery process. Such signature validation is meaningless due to the ease of finding collisions in MD5, and may be exploited by an attacker to forge responses to the TV. Similarly, the `LG ThinQ` app uses AES in ECB mode to encrypt passwords in its WifiNetwork class, using a hardcoded key. Both the ECB mode and the hardcoded key render the encryption meaningless. Finally, we find that the `Harmony` app uses weak symmetric encryption standards and cryptographic hashes (DES, MD5) to encrypt (and perform integrity checks on) important authentication attributes such as the `hubSecret` and `authToken`.

The real impact of vulnerabilities demonstrated in our case studies affects millions of users (and billions in the case of some apps). Our manual investigation is in no way exhaustive, and that we did not find IoT impact in 2/18 apps does not indicate its absence. With the mobile-IoT snapshot and artifacts generated by IoTSpotter, our hope is to enable regular, contextual analysis of mobile-IoT apps as they appear on the market, so as to proactively detect and mitigate such vulnerabilities.

9 LESSONS

Our findings demonstrate critical gaps in the security of mobile-IoT apps and libraries ($\mathcal{F}_1$–$\mathcal{F}_8$), and their impact on the IoT system ($\mathcal{F}_9$, $\mathcal{F}_{10}$). This section discusses three key lessons from our research at this critical intersection of mobile and IoT security.

Lesson 1: We need to focus efforts on mobile-IoT apps – While the security of mobile-IoT apps is understudied [44, 65], there is significant body of work [15, 21, 22, 31, 44, 62, 70, 73] on the security of another class of "apps" in the IoT domain, *IoT apps, i.e.*, simple trigger-action automation programs that run on popular IoT platforms (*e.g.*, SmartThings "SmartApps" [57] and NEST [51] routines). Recent work [50] demonstrates that users may not be inclined to (or need to) deploy developer-defined IoT apps, instead preferring to create routines on their own via platform UIs. Moreover, IoT apps may no longer be hosted by the platform (*e.g.*, SmartThings v3 [58]), instead being hosted entirely on third-party clouds (as black boxes), making their acquisition and analysis infeasible. In contrast, typical end-users need mobile-IoT apps, as they cannot program complex Android apps (as opposed to simple routines). Moreover, IoTSpotter makes it feasible to acquire mobile-IoT apps at scale, and our analysis shows the significant prevalence of vulnerabilities in them. Therefore, we argue that mobile-IoT apps are both *necessary* and *feasible* to analyze, and given the severe vulnerabilities uncovered in this paper, deserve greater attention from the security community, at least equal to if not more than IoT apps.

Lesson 2: We need to address the key bottleneck in large-scale mobile-IoT analysis – Despite developing a large snapshot of mobile-IoT apps, the scale of our analyses was restricted by the *runtime performance* of security and static analysis tools. For instance, it cost us more than 10 days to analyze only 917/37k mobile-IoT apps with CryptoGuard. Similarly, it took over 3 weeks in compute time to search for the use of vulnerable libraries in only 5,380/37k mobile-IoT apps (with >50k installs). Although this lack of scalability is well-known, it is interesting to observe its crippling effect on the large-scale analysis of mobile-IoT apps, even when other challenges have been met, such as a suitable large-scale snapshot of apps collected, and IoT libraries identified. From our observation, the main reason for the slowdown is its general analysis of *all areas of the app*, including those that may not be relevant to the domain impacted by the app (*i.e.*, IoT, in this case). One approach to resolve this bottleneck is to contextualize the analysis to only examine those areas of code relevant to IoT, which is nevertheless necessary to enable a more precise exploration of mobile-IoT app security, as we discuss next.

Lesson 3: We need *contextualized*, automated, security analyses for mobile-IoT – Cryptographic vulnerabilities are as prevalent in mobile-IoT apps as in general mobile apps ($\mathcal{F}_5$), which implies that developers have continued to make similar mistakes in this newer domain, but with a far worse impact on the user's physical environment ($\mathcal{F}_9$). Given this qualitative difference between the implications of vulnerabilities in mobile-IoT and non-IoT domains, we believe that it is necessary to *contextualize existing Android security analysis techniques* to IoT, *e.g.*, by adapting them to identify and prioritize areas of code relevant to IoT. Such an adapted analysis would allow us to automatically reason about the security implications of a vulnerability in the IoT context, saving manual effort, while also reducing the compute-time required to analyze an app due to the narrower focus. The recent focus on automated and scalable compliance enforcement for mobile-IoT apps (*e.g.*, by standards organizations such as ioXt [41]) would further increase the demand for such tools. We believe that the intuition from our manual reverse-engineering (Section 8), as well as the intuition behind our library-identification approach, can be automated to quickly triage apps for IoT-specific functionality. The techniques, artifacts, and insights developed by IoTSpotter enable us to pursue such contextualized mobile security analysis in the future.

10 THREATS TO VALIDITY

Our work aims to obtain a best effort estimate of the mobile-IoT app presence on markets such as Google Play, to enable insights that can only be obtained via large-scale analysis.

1. A best-effort, market-scale snapshot: As discussed in Section 4, we perform a comprehensive and systematic exploration to obtain our snapshot and demonstrate its validity by considering all possible avenues for collection, and manual validation of a subsample that demonstrates 88.4% precision. However, we note that this paper makes the first attempt to address the several challenges in creating such a snapshot, and offers a valuable understanding of the mobile-IoT presence on Google Play. Therefore, while there is imprecision in our approach, we consider the engineering effort needed to refine this snapshot, *e.g.*, of further analyzing mobile-IoT metadata to reveal increasingly distinguishing characteristics, within the scope of future work.

2. False negatives: As described in Section 3.1, the two classifiers we train and use to identify mobile-IoT apps, BERT and BiLSTM, may not always agree in their assessment for an app, and this disjoint set of apps that both disagree upon may contain several more mobile-IoT apps. That is, IoTSpotter's approach is not immune to false negatives. We believe that along with improving the features to further improve our models, future work may also explore the use of out-of-band information (*e.g.*, product titles sold on Amazon) to further extract mobile-IoT apps from this set of ambiguous apps.

3. Scale of Analysis: While certain security analyses that were performed (*e.g.*, the library analysis) at the scale of the entire mobile-IoT app snapshot, others were performed on smaller subsets for practical reasons (mainly the time required for analysis tools and manual analysis to scale). As a result, the findings resulting from these analyses may not generalize to all 37k mobile-IoT apps.

4. Exploitability: We do not evaluate the exploitability of the discovered vulnerabilities. That is, while some are certainly exploitable (*e.g.*, the Janus vulnerability), others may only be so in specific circumstances (*e.g.*, the use of a vulnerable library for a security-critical function). However, we note that the presence of several critical vulnerabilities in popular mobile-IoT apps is in itself a concern, as future code may accidentally make use of a vulnerable piece of code that is currently unused.

11 RELATED WORK

There has been significant prior work on the security of IoT apps, devices, and platforms. Our work differs from (and in many cases complements) such prior work by enabling a market-scale characterization of mobile-IoT apps, through the novel IoTSpotter framework, the evolving mobile-IoT app snapshot that it develops, its analysis of the snapshot and the resultant findings.

Past work on the analysis of mobile-IoT apps [23, 43, 66] is limited in terms of the scale and insights that IoTSpotter enables. The closest to IoTSpotter is the work by Wang et al., who develop a dataset of 2081 mobile-IoT apps. We deviate from Wang et al. in key ways: (1) Wang et al. seek to obtain *some* mobile-IoT apps for a specific analysis (identifying vulnerable components across product-types), whereas IoTSpotter is designed to obtain a large body of mobile-IoT apps (*i.e.*, a market-scale snapshot) to enable diverse analyses, as we demonstrate in Section 5–Section 7. IoTSpotter's ML-approach generalizes observations from training data, obtaining a diverse set of mobile-IoT apps 18X larger (37,783/2081) than Wang et al.'s set obtained via snowball sampling that converges on a smaller set of apps similar to the seeds. Moreover, we develop a novel approach for identifying IoT-relevant libraries that identifies 19k such packages, and our analysis yields key results characterizing mobile-IoT apps and libraries ($\mathcal{R}_1$–$\mathcal{R}_7$), and impactful security findings ($\mathcal{F}_1$–$\mathcal{F}_{10}$).

Further, prior work has analyzed IoT artifacts on a scale similar to IoTSpotter's, but from a different perspective. Particularly, the large scale network analysis by Kumar et.al. [47] answers important questions about the use of IoT devices in consumer homes. IoTSpotter complements this work by answering the related but unique questions of what IoT products (*i.e.*, mobile-IoT apps) are available on the market (rather than being used by consumers), and more so, facilitates the large scale analysis of mobile-IoT apps, which Kumar et al. do not analyze.

A large body of work has centered around identifying issues related to automation [15, 21, 31, 44, 62, 70, 73] and built tools and techniques [22, 42, 50] to mitigate risks associated with automations facilitated by IoT platforms. Similarly, researchers have also focused on analyzing the security of IoT devices to understand remote binding [25], IoT protocols [75], and memory corruption [24] in IoT devices. Both the mobile-IoT snapshot and insights generated based on our analysis of the current state of IoT market can be used to advance such research for more generalizable findings via a comprehensive characterization of all relevant products.

Finally, there has been significant work in mobile app security research focusing on fingerprinting [61, 71, 74], and library extraction [49, 67] of mobile apps. Our work builds on top of these tools and frameworks to adapt the technique for our analysis. Similarly, prior work has also separately leveraged cryptographic API misuse detectors [46, 55] to study the app market. We envision that our work is complementary and provides foundation for future work to perform similar focused explorations of IoT.

12 CONCLUSION

In this paper, we develop a systematic understanding of what products constitute the IoT mobile app market by developing a market-scale snapshot of 37k mobile-IoT apps, using the IoTSpotter framework. Our characterization of this snapshot leads to 7 key measurement results ($\mathcal{R}_1$–$\mathcal{R}_7$), and our multi-faceted security analysis of it leads to 10 security findings ($\mathcal{F}_1$–$\mathcal{F}_{10}$). Our findings demonstrate prevalence of critical vulnerabilities in mobile-IoT apps, and their serious impact on IoT security and privacy. These implications motivate an increased focus on mobile-IoT apps, and the development of mobile security analyses contextualized to IoT, which the intuition, techniques, data, and artifacts developed in this paper enable.

ACKNOWLEDGMENTS

The authors have been supported in part by the NSF-CNS-2112471, NSF-CNS-2132281, a COVA CCI Dissertation Fellowship, and a Cisco Research Grant. Any opinions, findings, and conclusions expressed herein are the authors' and do not reflect those of the sponsors.

REFERENCES

[1] "Alink protocol," https://www.alibabacloud.com/help/en/doc-detail/90459.htm, accessed: 2022-01-10.

[2] "apksigner," https://developer.android.com/studio/command-line/apksigner, accessed: 2021-01-10.

[3] "Appbrain," https://www.appbrain.com/, accessed: 2021-08-09.

[4] "Bert base model (uncased)," https://huggingface.co/bert-base-uncased, accessed: 2021-01-15.

[5] "Berttokenizer," https://huggingface.co/docs/transformers/model_doc/bert#transformers.BertTokenizer, accessed: 2021-01-15.

[6] "Cve-2017-18349 detail," https://nvd.nist.gov/vuln/detail/CVE-2017-18349, accessed: 2021-01-15.

[7] "Cve-2019-11777 detail," https://nvd.nist.gov/vuln/detail/CVE-2019-11777, accessed: 2021-01-15.

[8] "Cve-2021-29425 detail," https://nvd.nist.gov/vuln/detail/CVE-2021-29425, accessed: 2021-01-15.

[9] "Cve-2021-36090 detail," https://nvd.nist.gov/vuln/detail/CVE-2021-36090, accessed: 2021-01-15.

[10] "Cve-2022-25845 detail," https://nvd.nist.gov/vuln/detail/CVE-2022-25845, accessed: 2021-08-07.

[11] "New android vulnerability allows attackers to modify apps without affecting their signatures," https://www.prweb.com/releases/2017/12/prweb14997455.htm, accessed: 2021-01-10.

[12] "tf.keras.preprocessing.text.tokenizer," https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer, accessed: 2021-01-15.

[13] "Welcome to google nest. build your helpful home." https://store.google.com/us/category/connected_home?hl=en-US, accessed: 2022-01-24.

[14] "What's new in maven," https://mvnrepository.com/, accessed: 2021-08-29.

[15] M. Alhanahnah, C. Stevens, and H. Bagheri, "Scalable analysis of interaction threats in iot systems." New York, NY, USA: Association for Computing Machinery, 2020, pp. 272–285.

[16] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 468–471.

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

[18] A. AWS, "AWS SDK For Java Core," https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-core/, 2021.

[19] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 356–367.

[20] Blood Pressure Diary, "Blood Pressure Diary App," https://play.google.com/store/apps/details?id=com.bluefish.bloodpressure&hl=en_US&gl=US, Accessed Jan 2022.

[21] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated IoT Safety and Security Analysis," in *2018 USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 147–158.

[22] Z. B. Celik, G. Tan, and P. McDaniel, "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT," in *Proceedings of the NDSS 2019 Symposium*, 2019.

[23] E. Chatzoglou, G. Kambourakis, and C. Smiliotopoulos, "Let the cat out of the bag: Popular android iot apps under security scrutiny," *Sensors*, vol. 22, no. 2, p. 513, 2022.

[24] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. Lau, M. Sun, R. Yang, and K. Zhang, "Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing," in *NDSS*, 2018.

[25] J. Chen, C. Zuo, W. Diao, S. Dong, Q. Zhao, M. Sun, Z. Lin, Y. Zhang, and K. Zhang, "Your iots are (not) mine: On the remote binding between iot devices and users," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 222–233.

[26] Connected IoT Devices 2021, "Connected IoT Worldwide," https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/, Accessed Jan 2022.

[27] A. Dal Pozzolo, O. Caelen, and G. Bontempi, "When is undersampling effective in unbalanced classification tasks?" Springer International Publishing, 2015, pp. 200–215.

[28] J. Developers, "jadx - Dex to Java decompiler," https://github.com/skylot/jadx/, 2022.

[29] EQB-600D, "EQB-600D," https://www.casio.com/in/watches/edifice/product.EQB-600D-1A/, Accessed Jul 2022.

[30] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android ssl (in)security," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[31] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Security and Privacy (SP), 2016 IEEE Symposium on*, May 2016, pp. 636–654.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[33] L. A. Goodman, "Snowball sampling," *The annals of mathematical statistics*, pp. 148–170, 1961.

[34] Google Play, "Nest," https://play.google.com/store/apps/details?id=com.nest.android, Accessed June 2018.

[35] ——, "Vivint," https://play.google.com/store/apps/details?id=com.vivint.vivintsky&hl=en_US&gl=US, Accessed June 2018.

[36] GSDMM Clustering, "The Movie Group Process Clustering," https://github.com/rwalk/gsdmm, Accessed Jan 2022.

[37] Hard Voting Scheme, "Hard Voting," https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html, Accessed Jan 2022.

[38] L. M. Hsu and R. Field, "Interrater agreement measures: Comments on kappan, cohen's kappa, scott's π, and aickin's α," *Understanding Statistics*, vol. 2, no. 3, pp. 205–219, 2003.

[39] IoTSpotter, "IoTSpotter Online Appendix," https://github.com/Secure-Platforms-Lab-W-M/IoTSpotter/blob/main/online-appendix/online-appendix.pdf, Aug. 2022.

[40] IoTSpotter Developers, "IoTSpotter Artifact ," https://github.com/Secure-Platforms-Lab-W-M/IoTSpotter, Aug. 2022.

[41] ioXt Alliance Members, "ioxt: The global standard for iot security," https://www.ioxtalliance.org/, 2021.

[42] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, "ContexIoT: Towards providing contextual integrity to appified IoT platforms," in *Proceedings of the 2017 Network and Distributed System Security Symposium (NDSS)*, 2017.

[43] D. M. 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 *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 181–186.

[44] K. Kafle, K. Moran, S. Manandhar, A. Nadkarni, and D. Poshyvanyk, "A Study of Data Store-based Home Automation," in *Proceedings of the 9th ACM Conference on Data and Application Security and Privacy (CODASPY)*, Mar. 2019.

[45] ——, "Security in Centralized Data Store-based Home Automation Platforms: A Systematic Analysis of Nest and Hue," *ACM Transactions on Cyber-Physical Systems (TCPS)*, vol. 5, no. 1, Dec. 2020.

[46] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler *et al.*, "CogniCrypt: supporting developers in using cryptography," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 931–936.

[47] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: An analysis of iot devices on home networks," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1169–1185. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-deepak

[48] X. Li, Q. Zeng, L. Luo, and T. Luo, *T2Pair: Secure and Usable Pairing for Heterogeneous IoT Devices*, 2020, p. 309–323.

[49] Z. Ma, H. Wang, Y. Guo, and X. Chen, "Libradar: Fast and accurate detection of third-party libraries in android apps," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 653–656. [Online]. Available: https://doi.org/10.1145/2889160.2889178

[50] S. Manandhar, K. Moran, K. Kafle, R. Tang, D. Poshyvanyk, and A. Nadkarni, "Towards a Natural Perspective of Smart Homes for Practical Security and Safety Analyses," in *Proceedings of the 41st IEEE Symposium on Security and Privacy (Oakland)*, San Fransisco, CA, USA, May 2020.

[51] Nest Labs, "Meet the Nest app," https://nest.com/app/, Accessed Feb 2019.

[52] Oracle, "Oracle Application Testing Suite Documentation," https://docs.oracle.com/cd/E97999_01/index.html, 2022.

[53] Pocket Cloudwatcher, "Pocket Cloudwatcher," https://play.google.com/store/apps/details?id=es.lunatico.pocketCW&hl=en_US&gl=US, Accessed Jan 2022.

[54] Prodigy, "Prodigy," https://prodi.gy/, Accessed Jan 2022.

[55] S. Rahaman, Y. Xiao, S. Afrose, F. Shaon, K. Tian, M. Frantz, M. Kantarcioglu, and D. Yao, "Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2455–2472.

[56] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, no. 1. Citeseer, 2003, pp. 29–48.

[57] SmartThings Developer Documentation, "SmartApps," https://docs.smartthings.com/en/latest/smartapp-developers-guide/, Accessed December 2018.

[58] SmartThings Developers, "The SmartThings Ecosystem," https://smartthings.developer.samsung.com/docs/index.html, Accessed June 2019.

[59] Snorkel AI, "Snorkel," https://snorkel.ai/, Accessed Jan 2022.

[60] Spacy Model, "Spacy en_core_web_lg model," https://spacy.io/models/en#en_core_web_lg, Accessed June 2021.
[61] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, 2016, pp. 439–454.
[62] R. Trimananda, S. A. H. Aqajari, J. Chuang, B. Demsky, G. H. Xu, and S. Lu, "Understanding and automatically detecting conflicting interactions between smart home iot applications," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 1215–1227. [Online]. Available: https://doi.org/10.1145/3368089.3409682
[63] Vstarcam, "Eye4," https://play.google.com/store/apps/details?id=vstc.vscam.client/, 2021.
[64] H. Wang, H. Liu, X. Xiao, G. Meng, and Y. Guo, "Characterizing android app signing issues," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 280–292.
[65] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full sha-1," in *Annual international cryptology conference*. Springer, 2005, pp. 17–36.
[66] X. Wang, Y. Sun, S. Nanda, and X. Wang, "Looking from the mirror: Evaluating iot device security through mobile companion apps," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1151–1167.
[67] Y. Wang, H. Wu, H. Zhang, and A. Rountev, "Orlis: Obfuscation-resilient library detection for android," in *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2018, pp. 13–23.
[68] Wearable Permissions, https://developer.android.com/training/wearables/overlays/permissions, Accessed May 2022.
[69] J. Yin and J. Wang, "A dirichlet multinomial mixture model-based approach for short text clustering," *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.
[70] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1074–1088. [Online]. Available: https://doi.org/10.1145/3243734.3243820
[71] Q. Zhao, C. Zuo, B. Dolan-Gavitt, G. Pellegrino, and Z. Lin, "Automatic uncovering of hidden behaviors from input validation in mobile apps," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1106–1120.
[72] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1133–1150.
[73] ——, "Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1133–1150. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/zhou
[74] C. Zuo and Z. Lin, "Smartgen: Exposing server urls of mobile apps with selective symbolic execution," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017, p. 867–876. [Online]. Available: https://doi.org/10.1145/3038912.3052609
[75] C. Zuo, H. Wen, Z. Lin, and Y. Zhang, "Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1469–1483. [Online]. Available: https://doi.org/10.1145/3319535.3354240

APPENDIX

A INTUITION BEHIND SELECTING SPECIFIC DECISION BOUNDARY VALUES

The decision boundary (μ) of 44 was chosen with an intuition to balance identification of IoT library package names with high precision and relatively less false positive cases. For this, we randomly sampled a set of 20 library package names (starting with a decision boundary of 60). We observed that the false positive cases continue to rise as we decreased the decision boundary. We found 44 to reasonably fit our goal i.e., our sample contained less false positive cases but were 44x more popular in IoT than non-IoT apps. We used a different decision boundary (ϵ) to identify library package names that were only available in IoT apps and unavailable in non-IoT apps (as we elaborate in Fig 4(a). This allows us to identify instances similar to the ones explained in the example (i.e., 10 apps calls a given library but non of of the non-IoT apps call a given package)

Table 7: Number of violations of CryptoGuard's rules in mobile-IoT and non-IoT apps (*sorted by # violations in mobile-IoT apps*). CryptoGuard assigns *severity* to rules, as we annotate in the table (high severity= [H], medium severity= [M], low=unmarked).

CryptoGuard's Rules (IDs as per [55])		# Violations	
ID	**Rule Name**	**Mobile-IoT**	**Non-IoT**
9	Insecure PRNGs (e.g., java.util.Random) [M]	15573	16778
16	Insecure cryptographic hash (e.g., SHA1, MD5) [H]	13297	16365
7	Occasional use of HTTP	2298	1593
1	Predictable/constant cryptographic keys [H]	2271	2359
5	Custom TrustManager to trust all certificates [H]	1931	910
14,11	*64-bit block ciphers (e.g., DES, RC4), ECB mode [M]	1311	1087
12	Static IVs in CBC mode symmetric ciphers [M]	716	467
4	Custom Hostname verifiers to accept all hosts [H]	293	269
3	Predictable/constant passwords for KeyStore [H]	100	
6	SSLSocketFactory w/o hostname verification [H]	186	86
13	Fewer than 1,000 iterations for PBE	104	32
2,10	*Predictable passwords, static salts in for PBE [H/M]	85	62
15	Insecure asymmetric cipher use	71	27
8	Predictable/constant PRNG seeds [M]	67	83
-	**TOTAL Violations**	38,486	40,218

* = CryptoGuard reports combined results for certain rules.

Table 8: The number of violations of CogniCrypt's rules in mobile-IoT and non-IoT apps (*sorted by # violations in mobile-IoT apps*).

CogniCrypt's Rules (IDs as per [46])		# Violations	
ID	**Rule SPEC**	**Mobile-IoT**	**Non-IoT**
36	MessageDigest	1743	2571
10	javax.net.ssl.SSLContext	1160	661
2	javax.crypto.Cipher	485	303
1	javax.net.ssl.TrustManagerFactory	257	238
43	java.security.Signature	236	140
39	javax.crypto.spec.PBEKeySpec	140	75
34	javax.crypto.spec.SecretKeySpec	133	82
16	javax.crypto.SecretKeyFactory	132	87
32	javax.crypto.spec.IvParameterSpec	78	56
12	SSLSocketFactory	66	5
22	java.security.KeyStore	47	42
4	java.security.KeyPairGenerator	33	17
25	javax.crypto.Mac	33	25
21	javax.net.ssl.KeyManagerFactory	23	1
26	SSLSocket	23	1
24	javax.crypto.KeyGenerator	21	10
30	javax.crypto.CipherOutputStream	17	8
37	javax.crypto.CipherInputStream	17	16
35	javax.crypto.spec.PBEParameterSpec	11	5
15	java.security.DigestOutputStream	8	8
31	java.security.SecureRandom	8	6
41	javax.crypto.spec.GCMParameterSpec	6	3
8	javax.net.ssl.SSLParameters	3	0
5	java.security.cert.TrustAnchor	2	1
17	java.security.DigestInputStream	2	2
3	java.security.AlgorithmParameters	1	0
27	java.security.cert.PKIXBuilderParameters	1	0
-	**TOTAL Violations**	4,686	4,363

B CRYPTO-API MISUSE DETECTED BY CRYPTOGUARD AND COGNICRYPT

Table 7 and Table 8 show the mapping of different rules with their respective flaws and severity of violation.