THESIS

AN ANALYSIS OF INTERNET OF THINGS (IOT) ECOSYSTEM FROM THE PERSPECTIVE OF DEVICE FUNCTIONALITY, APPLICATION SECURITY AND APPLICATION ACCESSIBILITY

Submitted by

Upakar Paudel

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2022

Master's Committee:

Advisor: Dr. Indrakshi Ray

Dr. Yashwant Malaiya
Dr. Steve Simske

ABSTRACT

AN ANALYSIS OF INTERNET OF THINGS (IOT) ECOSYSTEM FROM THE PERSPECTIVE OF DEVICE FUNCTIONALITY, APPLICATION SECURITY AND APPLICATION ACCESSIBILITY

Internet of Thing (IoT) devices are being widely used in smart homes and organizations. IoT devices can have security vulnerabilities in different fronts: Device front with embedded functionalities and Application front. This work aims to analyze IoT devices security health from device functionality perspective and application security and accessibility persepective to understand holistic picture of entire IoT ecosystem's security health.

An IoT device has some intended purposes, but may also have hidden functionalities. Typically, the device is installed in a home or an organization and the network traffic associated with the device is captured and analyzed to infer high-level functionality to the extent possible. However, such analysis is dynamic in nature, and requires the installation of the device and access to network data which is often hard to get for privacy and confidentiality reasons. In this work, we propose an alternative *static* approach which can infer the functionality of a device from vendor materials using Natural Language Processing (NLP) techniques. Information about IoT device functionality can be used in various applications, one of which is ensuring security in a smart home. We can also use the device functionalities in various security applications especially access control policies. Based on the functionality of a device we can provide assurance to the consumer that these devices will be compliant to the home or organizational policy even before they have been purchased.

Most IoT devices interface with the user through mobile companion apps. Such apps are used to configure, update, and control the device(s) constituting a critical component in the IoT ecosystem, but they have historically been under-studied. In this thesis, we also perform security and

accessibility analysis of IoT application on 265 apps to understand security and accessibility vulnerabilities present in the apps and identify some mitigating strategies.

ACKNOWLEDGEMENTS

I would like to thank the CSU Graduate Student Council and the CSU Graduate School for initiating, commissioning and supporting this project. I would also like to thank Dr. Indrakshi Ray for her support and ensuring that we followed through with this project to completion. I would like to thank Dr. Suryadipta Majumdar and Dr. Lorenzo Carli for advising on various parts of this project.

This work was supported in part by funds from NIST under award number 60NANB18D204, and from NSF under award number CNS 1822118 and from NIST, Statnett, Cyber Risk Research, AMI, and ARL.

DEDICATION

I would like to dedicate this thesis to my family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Overview

The popularity of IoT devices is rapidly increasing and gaining massive traction in the current scenario of technological advancement. The increasing popularity of IoT leads to various security and privacy challenges. We are moving towards a smart and connected world, which is estimated to have 55.7 billion connected devices by 2025, out of which 75% will be IoT devices [1]. This growth in IoT acceptance comes with increased security vulnerabilities in an entire IoT ecosystem. This work aims at understanding the vulnerabilities and security flaws that might occur in IoT ecosystems from various different perspectives.

1.2 Problem Motivation

The increasing popularity of IoT is suspected of leading to more dangerous attacks and privacy implications in the cybersecurity domain. The IoT ecosystems have a large range of use cases in various diverse environments ranging from health care, smart homes, Machine-to-Machine (M2M) connected devices, industrial controls, and smart agriculture. IoT devices contain a set of transducers (sensors and actuators) for providing the smart environment, which in turn can capture sensitive and private consumer information (daily routine of smart home users). IoT devices also use IoT applications as a bridge which enable these devices to connect to the Internet as well as other devices and facilitate sharing of data and information in the network. Complete understanding of the security health of the IoT ecosystem is not possible without an understanding of device functionality. Moreover, an ambiguous understanding of device functionality will lead to partial knowledge of the strength of IoT devices and will open the IoT ecosystem to an entire foray of an attack scenario. Similarly, improperly developed or vulnerable IoT applications can act as a

gateway for attackers to get hold of the entire IoT ecosystem as well as the network in which these devices are configured.

1.2.1 Functionality Analysis

IoT devices contain many sensors and actuators and are able to perform multiple functionalities. For example, Nest Protect [2] whose primary functionality is to detect smoke, also contains a microphone, rendering it to audio capture functionality. Consumers may install Nest Protect in their bedroom for smoke detection purposes, but Nest Protect's functionality of audio capture puts their security and privacy at risk. Thus, there is a need to understand the latent functionalities of IoT devices and their impact on security and privacy.

The most widely prevalent approaches [3–11] for obtaining device functionalities fall under the category of *dynamic approaches* which profile device behaviors by installing or simulating IoT devices and observing their network behaviors, and inferring their functionalities. Often, the network traffic is encrypted; consequently, such approaches are inaccurate and report high false positive and false negative rates. Moreover, deriving high-level functionality from low-level network data is non-trivial. Such approaches require set-up effort and time for capturing device information. If such a set-up is done in real-world settings, device data may violate the security or privacy concerns of consumers. One *static approach* [12] profiles device functionalities by analyzing vendor materials of IoT devices. This approach also suffers from accuracy issues, as they mainly rely on the identification of device functionalities without considering the contextual information of words. For instance, the same word such as 'light' can have multiple meanings, and so it is hard to deduce whether the corresponding device can illuminate a room.

1.2.2 Application Analysis (Security and Accessibility)

Most IoT devices interface with the user through mobile companion apps [13]. Such apps are used to configure and control the device, and in some cases, they proxy communications between devices and the cloud. Therefore, they constitute a critical component in the IoT ecosystem, but they have historically been under-studied.

Indeed, most works on the security of IoT devices focus on the devices themselves, ignoring their mobile companion apps. To make matters worse, online characterizations of user experience pertaining to IoT companion apps insinuate poor design in regards to key aspects. This affects security/privacy [14], and accessibility [15]. To the best of our knowledge, no rigorous assessment of the software quality of companion apps along the above dimensions exists. Closing this gap is important in order to identify problem areas deserving further study and industry focus.

1.3 Contribution

1.3.1 Device Functionality

In this thesis, we propose a static analysis approach. We associate a set of keywords with generic devices based on their functionality, check for the presence of these keywords and their context in the vendor materials using NLP techniques (e.g., [16, 17]), deduce the presence of relevant transducers in the IoT device, and use this knowledge to obtain the device functionalities. The use of contextual information in the vendor material reduces the number of false positives. We evaluate our approach on IoT products from different vendors, including Arlo and Samsung, and report encouraging results.

We provide a framework that allows for extracting device functionality which can be proactively used to check the conformance of device placement even before device purchase. Our contributions include the following.

1. We provide a context-aware approach for extracting IoT device functionalities from publicly-available resources using NLP.

2. We implement our proposed approach in the context of smart homes with diverse devices (Nest Camera, Arlo Ultra Cam, Samsung Smart Cam) from various vendors (Arlo, Nest and Samsung), and evaluate its efficiency, accuracy, and scalability, to demonstrate the feasibility of our approach.

1.3.2 Application Analysis

The goal of our work is to assess the quality of IoT companion apps under the lens of security and accessibility. Security is an important concern due to the numerous high-profile issues that tend to plague the IoT ecosystems in these aspects. Accessibility are important aspects for adoption; apps with poor accessibility are less likely to be used, and to be used correctly (which can indirectly generate security risks).

In both our security and accessibility analyses, we find that most apps have an acceptable posture, although there are many apps with potentially serious issues (e.g., apps with known CVSS vulnerabilities with severity in the "medium" range; apps with > 100 accessibility rule violations).

Overall, our work makes the following contributions:

1. We define and implement a method to identify and scrape IoT companion apps from the Google play store.

2. We define methods to quantitatively and reproducibly assess aspects of app security and accessibility.

3. We perform a large-scale measurement study of the aforementioned properties.

1.4 Outline

The remainder of this thesis is organized as follows. Chapter 2 discuss related work and its limitations. Chapter 3 provides preliminaries and background knowledge about vendor materials, context-based algorithms, and security/accessibility analysis techniques. Chapter 4 discuss about a framework to enumerate device functionality from vendor materials by leveraging contextual information along with experimental results. Similarly, chapter 5 discuss approach and strategy used to successfully analyze IoT applications from security and accessibility perspective and analysis of result obtained. Finally, Chapter 6 concludes this thesis and talk about the various direction for future work.

Chapter 2

Related Work

2.1 Device Functionality Analysis

2.1.1 Dynamic Analysis

[3–9] perform fingerprinting of IoT device behavior by analyzing network traffic. [3–5] use machine learning model trained on network traffic according to their service (e.g., DNS, HTTP) and the semantic behaviors of devices (e.g., detected motion) to automatically discover and profile device behaviors. [7–9] use unsupervised learning to build models for individual devices based on captured network traffic and automatically detect device identity. Hamza et al. [18–20] use Manufacturer Usage Description (MUD) profiles to fingerprint and detect device functionalities. These works detect network level device functionality based on network traffic. Particularly, Hamza et al. [18] develop a tool to detect and verify MUD profile based on network traffic, and device classification has been performed based on observed network traffic as well as generated MUD signature. It converts MUD policies to flow rules and uses those flow rules to detect an intrusion. [20] uses security testing methodologies to augment the MUD profile and increases its expressiveness by considering additional security aspects beyond just network traffic. MUD based solutions rely on MUD profiles and on vendors to provide complete and correct MUD profiles for IoT devices. For both ML and MUD based approaches, device access is required, and the device must be connected and operational. Although these approaches can detect network-level behavior, inferring high-level operation is error-prone because of the encrypted payload.

2.1.2 Static Analysis

Our previous work [12] implements a technique of extracting IoT device functionalities based on design specifications from vendor materials. We use key term matching to detect transducers from vendor materials and map those transducers to their capabilities for identifying device func-

tionalities. However, this work ignores the usage context of the key terms and hence results in high false positive rates.

2.1.3 Context-based approaches for IoT

[21] makes an effort towards understanding the context of the data generated by IoT sensors and actuators and to provide personalized recommendations to the users based on the captured data. Similarly, [22] exploits the heterogeneous contextual information (e.g., daily activities) that are captured from IoT devices. Then, that contextual information is leveraged to personalize the care management process, especially for elderly people. Our work aims at a different objective where we intend to extract device functionalities for consumer security without using any captured data.

2.2 Application Analysis

2.2.1 Security Analysis

Jansen [23] highlighted various limitations of commonly-used security metrics, such as the reliance on human judgment and inherent subjectivity. Similarly, Krutz et al. found that user ratings weakly correlate with security [24]. Consequently, we mostly focus on security metrics rooted in the objective properties of each app, which can be measured by automated analysis (e.g., score of CVSS issues). RiskMon [25] builds a risk assessment baseline based on user expectations and then assesses the risk incurred each time an app performs a sensitive operation. As we require an assessment to be automated, this approach is outside the scope of our work. WHYPER [26] searches for explanations of app permissions in app descriptions, flagging unexplained permissions. We do not include this approach as lack of explanation, while suspicious, does not represent a security issue per se. Along these lines, Peng et al. [27] proposed the use of various types of probabilistic generative models to attribute a risk score to Android apps based on the combination of permissions requested. Later work [28] has shown that ML-driven analysis of permission sets leads to superior results compared to probabilistic models; therefore, we prefer the latter approach.

2.2.2 Accessibility Analysis

Most studies of IoT companion apps characteristics tend to focus on security; there is limited work focusing on accessibility. In regards to accessibility, Eler et al. [29] proposed MATE, a tool that automatically evaluates apps accessibility for visually impaired users. Similarly, Alshayban et al. [30] proposed a tool to automatically detect accessibility issues of Android applications. Bai et al. [31] provided an overview of different techniques and tools used for accessibility testing while focusing on testing methods that can be conducted by a software development team.

Chapter 3

Preliminaries

3.1 Device Functionality Extraction

3.1.1 Vendor Material Description

This work considers various publicly available vendor materials , including product webpages, technical specifications, and setup videos, for the purpose of extracting device functionalities. Product webpages (e.g., [32]) are the official marketing pages briefly describing a device, including its basic features. Technical specification pages (e.g., [33]) provide device specifications, including its hardware configurations. Setup videos (e.g., [34]) elaborate on how to install IoT devices , including useful information about device functionalities. During our analysis, we experience on average 32-page long specification documents and 4:48 minute long setup videos.

3.1.2 Challenges of Context-Aware IoT Device Functionality Extraction

The challenges in extracting IoT device functionalities from vendor materials are as follows.

1. *Understanding multimedia contents.* In addition to text, device vendors often use multimedia technology, such as picture, audio, and video. Thus, text processing alone becomes insufficient for extracting device functionalities. Automatically encoding these multimedia elements poses additional difficulty.

2. *Minimal mention of key terms.* In many cases, key terms (i.e., that might indicate the presence of a transducer or functionality) are mentioned very few times in vendor materials with brief information about them. Therefore, obtaining contextual information about those key terms becomes challenging.

3. *Ambiguous use of key terms.* The key terms are used in different ways by the various vendors. This makes it hard to identify the context and infer the presence of a functionality.

4. *Lack of standardized template.* Each vendor follows different conventions in arranging their materials. Furthermore, different materials of the same vendor follow different formats. Also, vendors craft expressions and styles, and include non-standard terms and phrasing unique to only their line of products. The lack of standard format for vendors to describe generic features or hardware makes device functionality extraction across different vendors challenging.

3.1.3 Intuition behind Context-Aware Device Functionality Extraction

We initially conduct a feasibility study to demonstrate the effect of context in inferring device functionality. For example, the term *temperature* is used in different contexts, such as for measuring device temperature, for referring to operational temperature, and for sensing/adjusting a room temperature. Thus, we need to understand the context before inferring device functionalities. In this feasibility study, we explore two major NLP techniques, N-gram and BERT augmented with Affinity Propagation.

3.1.4 N-gram

N-gram [16] is an NLP technique to predict the occurrence of a word based on its previous N words. [12] extracts the device specification from various vendor materials (overview page, technical specification page, and manuals). After extracting the device specification, then irrelevant contents (e.g., stop words, site navigation link, copyright information) are pruned to get the overall corpus for a specific device. We operate on this generated corpus to apply N-gram and detect local context for a specific term. Once we have a corpus for a specific device, we convert the corpus to trigram. We are interested in certain transducers (for example, *temperature sensor*) and the goal is to detect the contexts associated with the key terms of those transducers in the extracted corpus.

For example, in the Nest Thermostat [35] corpus, we have the phrase: *"nest temperature sensor room like"*. The trigrams generated from this phrase are `<nest temperature sensor>`, `<temperature sensor room>`, and `<sensor room like>`. From the Arlo Pro 3 Floodlight corpus, the phrase *"5 hrs operate temperature"* generates the following trigrams: `<5 hrs`

operate> and <hrs operate temperature>. By analyzing the preceding and following words in trigram for both Nest thermostat and Arlo Video Doorbell, we can see that the word temperature in Nest thermostat has the context of sensing temperature, whereas the word temperature in Arlo video doorbell has the context of the operational environment. We utilize this observation in our methodology in Section 4.

3.1.5 BERT

BERT [17] (Bidirectional Encoder Representation From Transformers) is a transformer-based language representation model and learns information from both sides of a word to learn the context of a specific word.

BERT is pre-trained on a large corpus of unlabelled text including the entire Wikipedia (2500 million words) and Book corpus (800 million words). Two models of BERT is introduced in [17],

- BERT Base: 12 layers (transformer blocks), 12 attention heads, and 110 million parameters
- BERT Large: 24 layers (transformer blocks), 16 attention heads and, 340 million parameters

What this means is, pre-trained BERT can be fine-tuned to create a state-of-art model for various natural language processing tasks like next sentence prediction, question answering, language inference, etc.

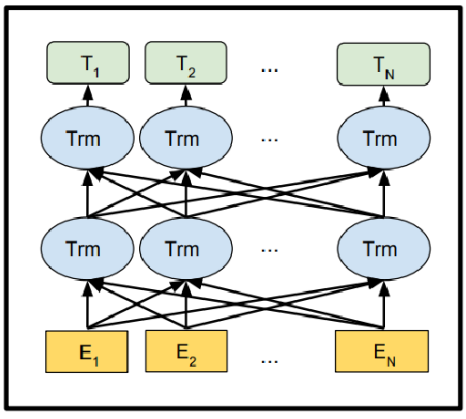


Figure 3.1: Architecture of BERT

Figure 3.1 display a basic BERT architecture. E1, E2 ... En represents a word embedding that is obtained after pre-processing so they can be fed into the BERT model. Trm represents transformer, and T1, T2 ... Tn represents the context based token generated by BERT. In the architecture, we can see the output from one layer of the transformer is fed to all the transformers in the next layer. This is how BERT achieves bi-directionality and can detect left, and right context from a sentence [36]. BERT is a deeply bidirectional algorithm that learns information from both the left and right side of a word and detects the context of a specific word based on it's learning [37].

In this work, BERT is used to detect contextual semantics associated with key terms in vendor materials. For example, in the following two sentences using the term *temperature*, BERT uses bidirectionality to understand their contextual meaning: (i) "check if the Arlo app is warning that your doorbell temperature is too high", and (ii) "the temperature they sense is warmer or cooler than homeowner's feel". In *(i)*, we factor the context of the words mentioned before *temperature* to understand whether the current context is of the type operating or notifying. In *(ii)*, to identify the context in which term *temperature* is being mentioned, we read the words following it, which clearly indicate the context of 'sensing'. In our methodology (in Section 4), we leverage this strength of BERT to accurately extract functionalities using the contextual meaning of a term.

3.1.6 Affinity Propagation

Affinity Propagation [38] is a graph-based clustering algorithm similar to K-Means that cluster the related data points together. The major drawback of K-Means clustering is that the number of clusters should be provided initially. But, the affinity propagation algorithm automatically detects an optimal number of a cluster by itself.

[39] Affinity Propagation sets exemplars (points that explain the other data points 'best'). A data point with the same exemplar is clustered together, and all data points collectively determine which data points are an exemplar for them. This process happens in the form of message passing, where all data points send messages to all other points about the willingness of the points being an

exemplar. The affinity propagation algorithm takes its course using four different sets of matrices, which are briefly described below:

- Similarity Matrix: Similarity matrix s(i,k) contains values that correspond to how similar two objects are. The similarity matrix can be calculated using a function as simple as negative Euclidean distance squared.

- Responsibility Matrix: Responsibility matrix r(i, k) reflects how well-suited point k is to be an exemplar for point i

- Availability Matrix: Availability matrix a(i,k) reflects how appropriate it would be for point it to choose point k as its exemplar.

- Criterion Matrix: Criterion matrix c(i,k) is simply the sum of r(i,k) and a(i,k). The highest criterion value of each row is then designated as an exemplar, and rows that share the same exemplar end up being in the same cluster.

3.1.7 Threat Model

This work is designed in the context of smart home, a growing IoT application. One focus of this work is to extract device functionalities using contextual information from vendor materials. The other focus is to demonstrate how the extracted functionalities are useful for security and privacy policies for consumers. The device functionalities can be used in access control scenario to verify device behavior according to required policy as well as to detect and prevent attacks that makes use of the sensors or actuators. In this work, we do not consider possible threat stemming from device misbehavior/malfunction and network attack that does not involve device transducers. This work does not rely on device access as well as network data. We enumerate device functionalities based on the vendor materials that are publicly and readily accessible. Missing information in those materials may affect the robustness of our approach.

3.2 Application Analysis

As there is no universally recognized definition for IoT companion app, we settle for one which is sufficiently precise to be rigorously applied for this thesis: *an IoT device is one that is able to interact with the physical world and is able to transmit that data to another networked node* [40, 41]. IoT devices are not used in a standalone fashion but almost universally via mobile apps [13]. Thus, *a companion app is an app that interfaces with an IoT device to sense the physical world by leveraging the transducer from the device and transmits the captured information over a network*.

3.2.1 Security Analysis

Before analyzing IoT applications from security health, we needed a coherent set of metrics that can be applied to these applications rigorously to assess their security health. In that regard, we came up with some set of security metrics that can be based on literature review and background knowledge. Some set of metrics are described here:

- *CVSS Score:* CVSS is a well known vulnerability scoring mechanism that output a value in the range of 0 to 10 where 0 corresponds to high security and 10 corresponds to less security. CVSS Score is generated by MobSF, a static analysis tool.

- *Malicious/Risky Third Party dependencies:* Android applications communicate with various hosts on the internet. It should communicate only with the nonrisky host. These metrics measure if an android application is talking to a risky or malicious host on the internet.

- *Number of ICC leaks:* talks about detecting ICC leaks, Inter Component leaks which capture the privacy leaks by one component of android application to other component or external entity.

- *Expired/Invalid certification:* Proper SSL or TLS certificates guarantee that the communication happening between client and server is encrypted on both ends, which helps protect the system from various attacks and threats. If an application communicates to an outer network

using an invalid or expired certificate, then it opens up an application to a wide range of security attacks.

- *Sensitive data to logs/third party:* Sensitive data should never be logged or sent to the third party. This metric is mentioned in OWASP Top 10 Security guidelines for mobile testing.

- *Days Since Last Released Update:* If an update for an application has been released far ahead in time, then it hints towards application being unmaintained which leads to more security vulnerability and vice versa.

3.2.2 Accessibility Analysis

Accessibility is the measure as well as technique to make web or mobile-based applications to be easily accessible and usable by a large number of people out there [42]. Application with higher accessibility scores and lower accessibility errors tend to be easy to use by a diverse group of users.

Accessibility Insights [1] is a free and open-source tool that provide an easy way for the developer to find and fix accessibility issues in web, windows, and android applications. Accessibility Insight has a specific tool called Accessibility Insight for Android [2], which captures accessibility issues in android applications by analyzing the respective screens. Accessibility Insight for Android can capture various set of accessibility errors (TouchSizeWcag, ActiveViewName, ImageViewName, EditTextValue and ColorContrast) for android applications.

[1] https://accessibilityinsights.io/

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

Chapter 4

Device Functionality Extraction

4.1 Proposed Framework

4.1.1 Overview

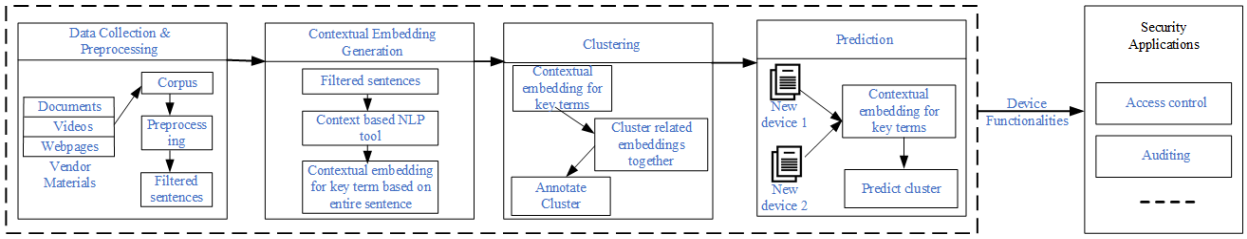


Figure 4.1: Our proposed methodology for the context-aware IoT device functionality extraction

Figure 4.1 displays a general overview of our proposed framework. Our framework can be categorized into four phases: (i) data collection and pre-processing, (ii) contextual embedding generation, (iii) clustering, and (iv) prediction. In the *data collection and pre-processing phase*, we first collect the entire corpus of texts from vendor materials (e.g., overview page, technical specification page, manuals, and setup videos) and then extract both key terms (i.e., the words that might be related to transducers) and their contextual information by leveraging NLP techniques. In the *contextual embedding generation* phase, we generate the contextual representation of key terms based on the context of an entire sentence in which the terms are present. In the *clustering* phase, we cluster sentences with similar contexts together and also annotate those clusters. Finally, in the *prediction* phase, we predict if any sentences from a new device are clustered to the annotated cluster to identify the functionalities of that device. The output (i.e., device functionalities) of our framework can be used for different security applications, such as access control, auditing. In this paper, we obtain functionalities of several smart home devices from different vendors, and utilize

those functionalities to specify and verify policies for consumers. Each phase of this framework is explained below.

4.1.2 Data Collection and Pre-processing

Data Collection: In this work, we extract all the text from various vendor materials (i.e., technical specifications, product overview pages, manuals or user guides, and setup videos). We leverage existing tools (e.g., beautifulsoup) to extract texts from the webpage (overview page and technical specification page) and manuals or user guides. We also extract texts from setup videos for a device by using Google Web Speech API [43]. Once all texts are extracted, we remove stop words and punctuation from those sentences as well as lemmatize our corpus so that we can leverage it further for understanding transducers and device functionalities.

Pre-processing: To extract the transducer information from the vendor materials, we first build an ontology, and then compare the obtained corpus and our ontology. Note that the ground truth is established based on this ontology, which is manually built from the analysis of vendor materials. To build an ontology of transducers (sensors and actuators), we perform a set matching algorithm between the key terms and words collected from the corpus. Specifically, key terms are divided into two categories: *indicative* and *related*. The indicative terms are sufficient to conclude the presence of a transducer and related functionality in a device. The related terms are also about transducers but are rather vague and insufficient in reaching a conclusion about the presence of transducers and related functionality. To match the obtained corpus and our ontology, we use one of the four algorithms as follows.

- *Device Cognisant Key Term Set Matching (dcKTSM):* In this algorithm, we select the most relevant key term set based on the presence of indicative terms. We extract matching key terms from our ontology based on the input corpus. This results in a subset of indicative terms for a candidate device. We then perform a reverse mapping from these indicative terms to the transducer they refer to.

- *Device Cognisant Full Key Term Set Matching (dcfKTSM):* This algorithm is similar to dcKTSM. The only difference is that we consider both indicative and related terms to select the most relevant key terms rather than just indicative terms as in dcKTSM.

- *Indicative Key Term Set Matching (iKTSM):* This algorithm directly considers corpus on all indicative terms from our ontology. The reverse mapping process from indicative term to transducer remains the same as above.

- *All Key Term Set Matching (aKTSM):* This algorithm is the most unguided algorithm. the iKTSM and aKTSM algorithms are very similar, except that in aKTSM, we evaluate corpus on both indicative and related terms instead of using only indicative terms in iKTSM.

However, solely applying any of the above algorithms on our corpus to elaborate device functionalities result in high false positive/negative rates (i.e., wrongly identifying the presence or absence of a functionality). This is mainly because of the naivety in our approach where we are directly mapping device to their functionalities based on the presence of indicative and related terms specific to a certain transducer in our collected corpus. For example, based on the key term set matching algorithms, the presence of sentence "*Check if the Arlo app is warning that your doorbell temperature is too high*" would indicate the presence of temperature sensor in Arlo Video Doorbell. However, in reality, Arlo Doorbell does not contain a temperature sensor. Therefore, in this work, we should also consider the context in the above sentence so that we can understand the appropriate intent (e.g., unfavorable environment being generated for a device) of the term *temperature* in that sentence. Therefore, to prune the device functionalities obtained from the key term set matching algorithms, we enlist transducers that can be represented in different contexts by vendors in their materials. For the rest of the paper, we will use temperature sensor, light sensor and lock (in eight Smart Home products: Nest Camera, Nest Protect, Nest Thermostat, Nest x Yale Lock, Arlo Video Doorbell, Arlo Pro 3 Floodlight, and Samsung Smart Cam [2, 33, 35, 44–47]) as examples to present our context-aware approach.

4.1.3 Contextual Embedding Generation

This phase generates the contextual representation for key terms. As shown in Section 3, we first build our intuition of context-aware extraction by conducting preliminary studies using both N-gram and BERT. Based on the outcome of that study, we conclude that even though N-gram (where $N = 3$) significantly improves the false positive rates over the basic key term set matching approach [12], this is an exhaustive process with low confidence. Therefore, we choose BERT to generate the contextual embedding for key terms as follows.

We extract all the sentences from vendor materials with key terms from the entire corpus on eight devices (as mentioned earlier). We then divide six devices for training purposes (where we cluster sentences with similar meaning to a key term together) and two devices for testing purposes (where we check if we could correctly predict the sentence from a new device and identify its functionalities based on the functionalities of the predicted cluster). We evaluate on all combinations ($^8C_2$) of training and testing device, which totals to 28 combinations in Section **??**. We then generate a BERT token for the key terms (e.g., *temperature* and *lock*) based on the context of the entire sentence. For example, the BERT token for the term *temperature* in the sentence "*Operating* ***temperature*** *-20 to 60 degree Celsius*" and sentence "*You adjust the* ***temperature*** *from your phone so they'll be cozy*" have different representations due to their different contexts. To generate the BERT token, we use Hianxao's BERT as a service tool [48]. We use the BERT model with 12 layers, 768 hidden units, and 12 attention heads for generating tokens.

4.1.4 Clustering

This phase clusters related tokens (that are generated by BERT) together by leveraging an affinity propagation algorithm, which is a graph-based clustering algorithm similar to K-Means that cluster the related data points together. In K-means, the value of 'K' (number of clusters) must be pre-specified, whereas the affinity propagation algorithm can automatically detect an optimal number of clusters [39]. After applying affinity propagation to BERT tokens, the contextually similar tokens are clustered together.

Table 4.1: Cluster after applying affinity propagation to BERT tokens

Cluster 1
Now turn up the temperature and get comfortable
Put a Nest Temperature Sensor in any room, like the baby's room, and you can tell Nest to make that room a priority (sold separately)
The Nest Thermostat can use sensors and your phone's location to check if you've left, then sets itself to an Eco Temperature to save energy
You adjust the temperature from your phone so they'll be cozy
You may also be into Google Nest Mini From Google Nest Protect From Google Nest Temperature Sensor
The temperature they sense is warmer or cooler than homeowners feel
In a room that's used often, so Nest can read the right temperature and the homeowner can easily reach it
If your existing chime doesn't ring when someone presses your Video Doorbell, your Video Doorbell or Power Kit might not be wired correctly, or the temperature of your Arlo Video Doorbell might be too high
Check if the Arlo app is warning that your doorbell temperature is too high

Table 4.1 and Table 4.2 display the subsets of cluster after successfully implementing affinity propagation on BERT tokens obtained from the term *temperature*. In each cluster, the term *temperature* appearing in different rows are used in very similar contexts. The sentences in Cluster 1 (in Table 4.1) have a notion of being able to sense/adjust temperature. Similarly, the sentences in Cluster 2 (in Table 4.2) provide information about suitable operating temperatures for the device. These two clusters demonstrate that BERT and Affinity propagation can successfully cluster sentences based on the contexts. After clustering the related sentences, we manually annotate the cluster, which refers to device functionalities. For example, we are interested in Table 4.1, because this cluster is about sensing/adjusting temperature, and we are interested in deriving device functionality of sensing temperature. We are not interested in Table 4.2 which is about favorable operating temperature for a device.

Table 4.2: Cluster after applying affinity propagation to BERT tokens

Cluster 2
Operating Temperature 40ºF (4 ºC) to 100ºF (38 ºC)
Operating Temperature 32°–104°F (0°–40°C)
Operating Temperature –22° to 140°F (–30° to 60°C)
Battery temperature range: 14° to 131°F (–10° to 55°C)
Operating temperature 32° to 104°F (0° to 40°C)
Operating Temperature (F) 0°C +40°C (+32°F +104°F)
Operating Temperature -20 to 60 degree Celsius
Operating Temperature -20 to 45 degree Celsius
Operating Temperature -20 to 45 degree Celsius
The operating temperature or voltage is too low

4.1.5 Prediction

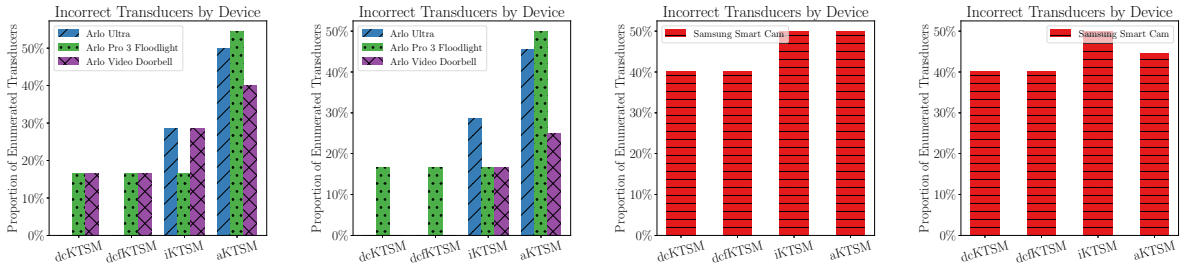
This phase predicts the functionalities of a new device. The previous phases remain the same for new devices. After context generation, we predict a suitable cluster for the sentences with key terms. If a sentence is predicted to fall under an annotated cluster, then we conclude the presence of functionality in the given device because the annotated cluster and current sentence both explain device functionality using similar semantics. Otherwise, we conclude the absence of that functionality. Our framework provides more precise information about device functionality by filtering based on the contextual meaning of transducers. An excerpt of device functionality output extracted by our framework is displayed in Table 4.3. After extracting the device functionalities, we can use them for various security applications, e.g., access control and auditing. One such use case is described below.

4.2 Result - Functionality Extraction Efficiency

Figure 4.2 portrays our results for incorrect transducer identification for Arlo and Samsung devices with the various key term set matching approaches, with and without using the N-gram approach. In Figure 4.2a and Figure 4.2b, we see that Arlo video doorbell does not show any incorrect identification of transducer on dcKTSM and dcfKTSM with N-gram approach as opposed to when only the algorithms were applied without the N-gram approach. Similarly, the false-

Table 4.3: Extraction of device functionality from our approach

Device Functionality/Device	Nest Protect	Nest Thermostat	Arlo Ultra	Samsung Smart Cam
Capture Image/Video			●	●
Detect Motion	●	●	●	●
Produce Infrared Light			●	●
Detect Light	●	●	●	●
Produce Light	●		●	
Capture Sound	●		●	●
Produce Sound	●		●	●
Detect Smoke	●			
Detect CO	●			
Measure Temperature	●	●		
Detect Contact				
Lock and Unlock door				
Control Thermostat		●		

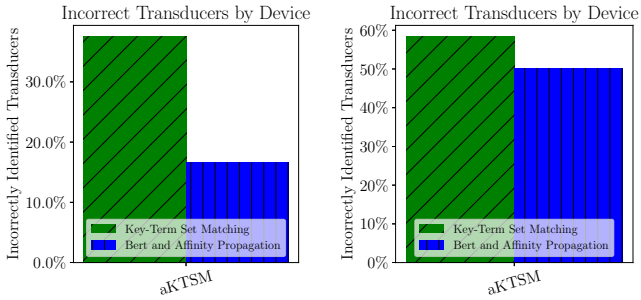


(a) Without N-gram **(b)** With N-gram **(c)** Without N-gram **(d)** With N-gram

Figure 4.2: For Arlo (fig. 4.2a and fig. 4.2b), and Samsung(fig. 4.2c and fig. 4.2d) devices, grouped by vendor and for each vendor grouped by KTSM algorithm the proportion of matching transducers that are incorrectly identified with and without N-gram

positive rate also decreases for the Arlo video doorbell with iKTSM when N-gram is applied. Also, we can see a slight improvement in the false-positive rate for Samsung Smart Cam with aKTSM approach on when N-gram is applied as opposed to when N-gram is not applied. This preliminary study concludes that detecting the true context in which the transducer is mentioned in vendor material helps to robustly enumerate the transducer and map them to their respective functionalities.

Figure 4.3 compares result initial algorithms without BERT and Affinity propagation give false positive rates of 37.5% and 58.334% respectively for Nest Cam Indoor and Samsung Smart Cam Video Surveillance, which reduced to 16.667% and 50% with the use of BERT and Affinity propagation.



(a) Nest Cam Indoor **(b)** Samsung Smart Cam

Figure 4.3: Incorrect transducer by device for Nest Cam Indoor and Samsung Smart Cam Video Surveillance with aKTSM w/o BERT and Affinity Propagation and aKTSM augmented with BERT and Affinity Propagation

We apply our framework on detecting two sets of transducers (Temperature Sensor and Lock) and map them to respective device functionalities. We test our approach on an aKTSM algorithm, because aKTSM algorithm was showing high false positive rate with key term set matching approach.

Table 4.4: Categorization of device extraction accuracy based on contextual embedding of key terms

Device Specific Accuracy	Vendor Specific Accuracy	Average Accuracy
Nest Protect: **100%**	Nest: **92.85%**	All vendors: **76.31%**
Nest Yale Lock: **100%**		
Nest Camera: **75%**		
Nest Thermostat: **100%**		
Arlo Pro 3 Floodlight: **83.33%**	Arlo: **50%**	
Arlo Ultra Cam: **60%**		
Arlo Video Doorbell: **0%**		
Samsung Smart Cam: **100%**	Samsung: **100%**	

We conduct our experiment on eight smart homes IoT devices which lead to ${}^{8}C_{2}$ i.e., 28 combinations, with a split ratio among devices of 75% (six devices) for training purposes and 25% (two) for testing purposes. We apply our framework based on BERT and Affinity Propagation to annotate the cluster that is about sensing/adjusting temperature using a training device. For each new testing device, we predict if any of the sentences with the word 'temperature' mentioned falls into our annotated cluster. If a sentence from the new device falls under our annotated cluster, we can

conclude with high confidence that the device into consideration should have a temperature sensor. While training our model on some combination of devices, it yields no significant cluster (cluster that explains device behavior) as output. Such combinations have been excluded from our result. We get a total of 19 combinations where we have a specific cluster explaining device behavior. Table 4.4 displays the accuracy of our approach categorized into vendors. We only have one Samsung device into consideration. However, we can see Nest devices performing significantly well with 92.85% of vendor accuracy, whereas Arlo vendor only have 50% accuracy. This difference in vendor accuracy stems from the clarity and conciseness with which the documents and materials pertaining to specific devices are provided by vendors. Nest provides more thorough documents for their device, which yields higher accuracy.

Chapter 5

Application Analysis

5.1 Data Collection and Analysis

5.1.1 Data Collection

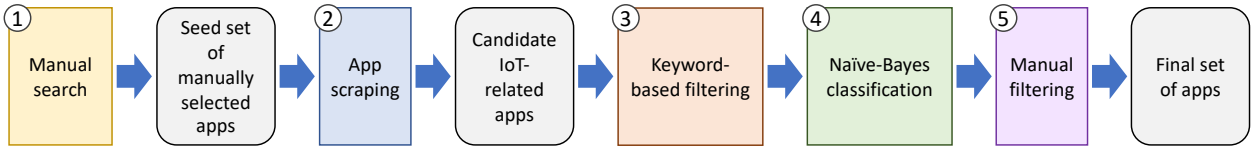


Figure 5.1: Overall process for IoT app data collection from the google play store

The data collection procedure is outlined in Figure 5.1. It includes:

[Step 1: Manual Search] We manually downloaded IoT apps from Google Play Store based on our definition of IoT apps. This forms our *Seed App Set*.

[Step 2: App Scraping] We next scraped more related IoT apps, starting from our seed set and following the "Similar Apps" suggestions presented by the Play Store for each app. We used play-scraper [49] to collect both app names and descriptions.

[Step 3: Keyword-based Filtering] While following "Similar Apps" links allows one to quickly gather a large set of apps, we found that this approach also results in a high number of false positives. Therefore, we performed additional filtering. The first of such filtering steps removes apps matching specific keywords (e.g., *currency, wallpaper, compiler, etc.*). We generated the set of keywords empirically by identifying keywords correlated with a high number of false positives.

[Step 4: Naïve-Bayes Classification] We then refined the candidate set by using machine learning to classify IoT and non-IoT apps. We initially attempted to apply the BERT algorithm [50] to leverage context for better classification and Logistic Regression Model in

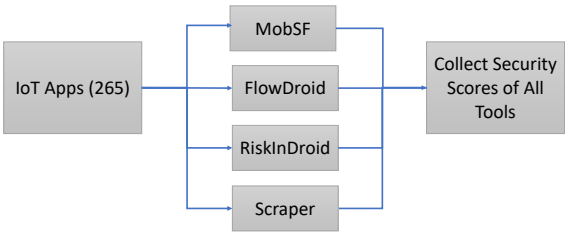


Figure 5.2: Flowchart depicting security analysis method

conjunction with the TF-IDF vectorizer. However, these algorithms did not perform well on our dataset. Using a Naïve-Bayes classifier leads to better accuracy, although far from optimal (64.6%).

[Step 5: Manual Filtering] In the last step, we manually reviewed all apps classified by Naïve Bayes as IoT-related; we retained only those which unambiguously match our definition of a companion app. At the end of this step, we were able to classify 484 apps as IoT apps (from an initial set of 2000 apps returned by the scraper).

Once we had the list of IoT apps, we used PlaystoreDownloader[3] to download the selected apps. We then performed our analysis based on each app's APK file.

5.2 Security Analysis

5.2.1 Method

The high level overview of security analysis is shown in Figure 5.2. After gathering APK's according to the process detailed in Section 5.1, we computed a number of security metrics to understand the application's security health along various dimensions. In the past few years, academia, industry, and the Open Source Software (OSS) community have proposed many strategies to measure security properties of Android apps (e.g., [25, 27, 28, 51–58]). Choosing an appropriate mix of metrics necessitates reviewing existing strategies and identifying a non-overlapping set of properties.

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

Table 5.1: Security categories relevant to our analysis

Security posture of an application. How likely is the app to be vulnerable to:	
Subcategory	**Tool**
Sensitive information leaks	MobSF
Ceding control to an attacker	MobSF
Denial of service	MobSF
Security usability of an application. How likely is the app to lead the user to:	
Subcategory	**Tool**
Pick insecure settings	RiskInDroid
Ceding control to an attacker	MobSF
Denial of service	MobSF
Software quality. How likely is the app to be affected by:	
Subcategory	**Tool**
Poor code quality	FlowDroid
Use of extraneous functionality	RiskInDroid

We required the following properties in order to be considered a metric: (i) tooling for the metric must be available; (ii) such tooling must not be abandoned (we used a fixed threshold of two years without repository updates to determine abandonment); and (iii) the tooling can be reproducibly executed on a modern machine/OS. The only exceptions are metrics that are trivially and unambiguously implementable (e.g., software abandonment, which is based on readily available metadata). As for relevance, based on our domain knowledge, we restrict our search to metrics within three different categories: security posture, security usability, and software quality of an application (the rationale for the latter is to understand and quantify the security health of an application). For each category, we further define additional subcategories. Our categories and subcategories are summarized in Table 5.1. For each subcategory, we also report specific tools matching that subcategory.

5.2.2 Tools and Metrics

In this section, we discuss various tools and metrics that we use.

RiskInDroid [28]. This technique analyzes the trustworthiness of a target application to be installed on Android devices. RiskInDroid distinguishes risky applications from non-risky ones

Table 5.2: Summary of metrics

Metric	ID	Value (v)	Better
RiskInDroid Score	rid	$0 \leq v \leq 100$	Low
MobSF CVSS Score	cvss	$0 \leq v \leq 10$	Low
User Rating	rat	$0 \leq v \leq 5$	High
Days Since Update	dsu	$0 \leq v$	Low
Use of Invalid/Exp. Certificates	ivcrt	0 or 1	0
FlowDroid Leaks	fdl	$0 \leq v$	Low

based on permissions using machine learning-based classifiers (multiple techniques are supported, including SVM, Multinomial Naive Bayes, Grading Boosting, and Logistic Regression). The tool takes an APK file as an input to produce RIV (risk index value) as output; the RIV estimates the similarity of the permission set to those commonly seen in malware. Higher RIV values correspond to risky applications and possibly malware, whereas lower RIV values correspond to the application being less risky. The value of RIV ranges from 0 to 100, where values above 65 show potential characteristics of malware.

MobSF CVSS Score [57]. The Mobile Security Framework (MobSF) is an automated penetration testing tool that generates an overall report of the security health of an android application. MobSF takes an APK as input and produces a parseable vulnerability report as output. MobSF can do both static and dynamic analyses; we only do static analysis for reasons of scalability. MobSF fingerprints known vulnerabilities in an app and aggregates their overall CVSS scores, resulting in an aggregate metric between 0 and 10 (higher scores indicate more serious vulnerabilities).

Use of Invalid/Expired Certificates. Mobile apps need to communicate and transfer data to and from outer networks and Internet. HTTPs (Secure HTTP) have been employed for encrypted and secure transfer of data. To successfully employ HTTPs by creating the secure connection between communication parties, TLS certificates need to be installed on a web server. We use MobSF to identify the use of invalid/expired TLS certificates.

User Rating: Study of app security with respect to user rating has been done previously [24], which is based on analyzing permission that the application requires for successful operation. We, therefore, use user rating as one of the dimensions and understand its correlation with the security

Table 5.3: Mean, min, and max for each security metric

Metric	Mean	Min	Max
rid	27.9	4.98	74.7
cvss	4.46	0	6.40
rat	3.57	0	4.90
dsu	163	0	1505
fdl	6.63	0	60

health of applications. We use the `play-scraper` python package [49] to scrape ratings from Google Play store. The app rating can range from 0 to 5.

Days Since Update. Modern apps make use of various third-party libraries to operate successfully. Every software needs to be updated and patched regularly as new vulnerabilities get discovered. Some previous work has looked at the issue of software abandonment [59]. We use Days Since Last Released Update (scraped using the `play-scraper` Python package) as a metric to analyze abandonment issues.

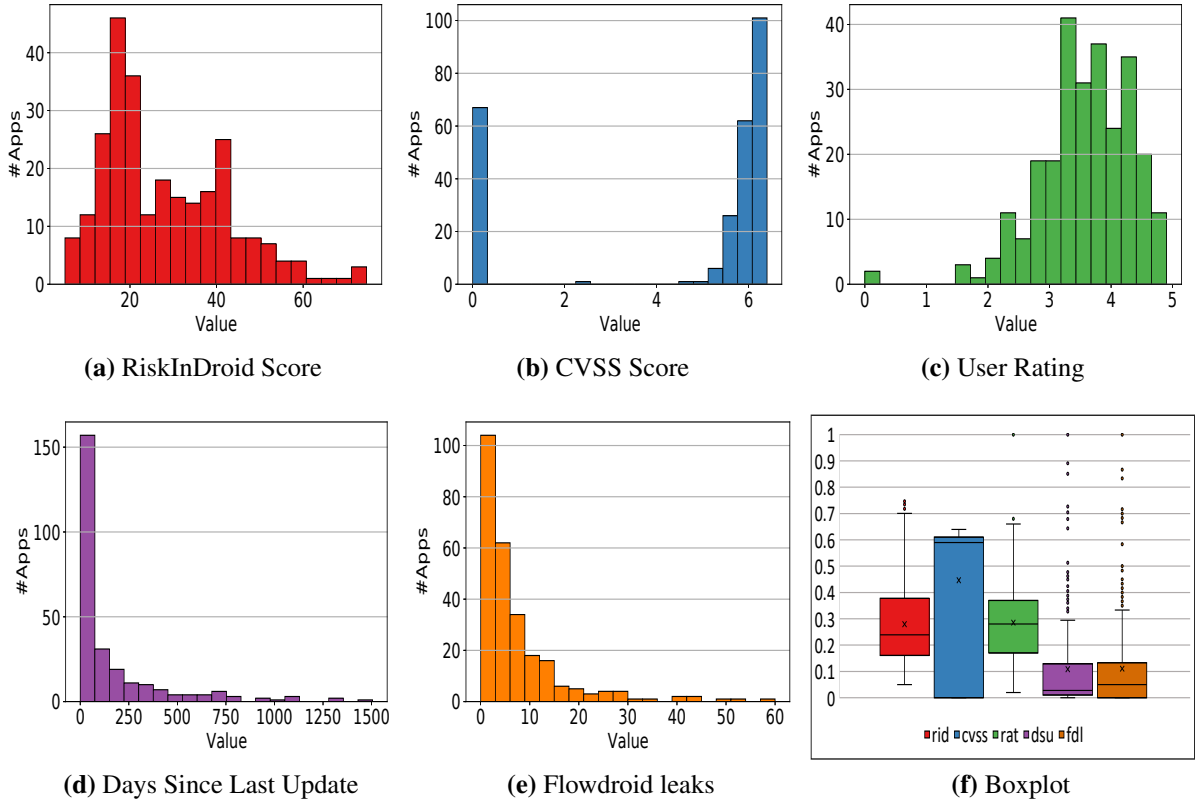
FlowDroid [55]. This tool performs intra-component analysis within Android apps. It investigates data flows between *sources* (locations where sensitive information could be created) and *sinks* (locations where such information could leave the app). FlowDroid parses the app binaries along with sources and sinks as input and produces an analysis of the application call graph as the output. After generating a call graph, it reports the existence of paths from source to sink. These paths are not vulnerabilities, but they represent situations where leakage of sensitive data is possible.

The metrics computed by each of the above techniques are summarized in Table 5.2. The use of invalid/expired certificates (ivcrt) affects 46 out of 284 (16.2%) applications. We excluded ivcrt from the rest of the evaluation due to the limited number of apps impacted, and its binary nature.

5.2.3 Results

Table 5.3 summarizes the mean and range for each of the five security metrics under discussion. Figure 5.3 displays individual distribution for 5 discussed metrics. In Figure 5.3f we also present

side-by-side boxplots of all metrics. For ease of comparison, prior to generating the boxplot each metric m was normalized, thus resulting in a normalized metric m_N, using the following process. For *RiskInDroid* and *CVSS*, $m_N = m/m_W$ (were m_W is the worst possible value, e.g., 100 for *RiskInDroid*). For *User Rating*, $m_N = 1-(m/m_W)$ (so that lower = better, for consistency with the other metrics). For *Days Since Update* and *FlowDroid Leaks*, $m_N = (m-m_{min})/(m_{max}-m_{min})$.



(a) RiskInDroid Score **(b)** CVSS Score **(c)** User Rating

(d) Days Since Last Update **(e)** Flowdroid leaks **(f)** Boxplot

Figure 5.3: Individual distributions of security scores for the IoT applications

Both RiskInDroid and FlowDroid analysis (Figures 5.3a and 5.3e) result in low to medium scores for the majority of apps (most apps received a FlowDroid score below 30, and none above 75; most apps exhibit a low number of FlowDroid leaks). Taken together, the result from *RiskInDroid* and *FlowDroid* suggests that most developers are relatively careful with the use of sensitive permissions and data. Note that the two scores are not correlated (discussed below). The distribution of *Days since the last update* is likewise concentrated around low values, which implies most

Table 5.4: Spearman's rank correlation coefficients between security metrics.

	rid	**cvss**	**rat**	**dsu**	**fdl**
rid	1.00	-	-	-	-
cvss	0.072	1.00	-	-	-
rat	0.036	0.033	1.00	-	-
dsu	-0.120	0.086	-0.180	1.00	-
fdl	0.058	0.046	0.044	-0.023	1.00

companion apps are frequently updated (more than 160 apps have received an update within the last 100 days). This metric is, however, uncorrelated to *CVSS score*, so even a high frequency of updates does not necessarily lead to the absence of vulnerabilities. *CVSS score* exhibits a bimodal distribution, with 67 apps showing a score of 0 and nearly 200 having a score of 5 to 6.5.

User Rating distribution shows that only few apps have very low (< 3) or very high (> 4.5) ratings. Most of the apps are rated between 3 to 4. Incidentally, while the data may appear to be normally or near-normally distributed, it fails the Shapiro-Wilk normality test ($W = 0.94$, $p = 1.49e^{-8}$). Furthermore, while one may hope that *User Rating* would act as an easy-to-compute proxy for other security properties, our correlation analysis suggests that this rating is unrelated to the security posture of the app.

We report the Spearman's rank correlation coefficient for each pair of metrics in Table 5.4; no significant correlation Overall, the lack of correlation indicates that our set of metrics is robust as it covers orthogonal aspects of each app. It also suggests that security-related shortcomings, when present, affect individual aspects of app security, which complicates the holistic assessment of the overall security posture.

Take-aways: While most apps fare relatively well under the most critical metrics, the same metrics exhibit a long-tail of apps with potentially problematic scores. The lack of correlation between metrics suggests that (i) our set of metrics is robust as it covers orthogonal aspects of each app; and (ii) security-related shortcomings, when present, affect individual aspects of app security, which complicates the holistic assessment of the overall security posture.

5.3 Accessibility Analysis

Accessibility aims to improve user access to systems through the interface, the hardware, and the environmental characteristics of a system. An accessible technology enables users to be able to perceive, understand, navigate, interact, and contribute to it[4].

5.3.1 Methods

To assess the accessibility of an Android app, we used the tool Accessibility Insights for Android[5]. Accessibility Insights provides a fast and detailed analysis for multiple accessibility issues of an app by evaluating individual screens of the app as seen by a user. After a review of possible approaches, we chose this tool because it automates a number of analyses (e.g., font size), while streamlining manual inspections that cannot be fully automated (e.g., contrast).

The tool works by taking screenshots of the interface of the app being evaluated and thereafter highlighting any instances that may relate to accessibility issues. "Accessibility issues" are construed as violations of the WCAG guidelines for web content accessibility[6].

1. *ActiveViewName*: Active views must have a name that is available to assistive technologies. For example, a slider should be associated with text describing what the slider is used for. The missing text results in a violation.

2. *ImageViewName*: Meaningful images must have alternate text. Images without associated text result in a violation.

3. *TouchSizeWcag*: Touch inputs must have a sufficient target size. The tool checks elements to have a minimum width or height of 44dp; elements that are smaller than 44dp result in a violation.

[4]https://www.w3.org/TR/WCAG21

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

[6]https://www.w3.org/WAI/WCAG21/Understanding/

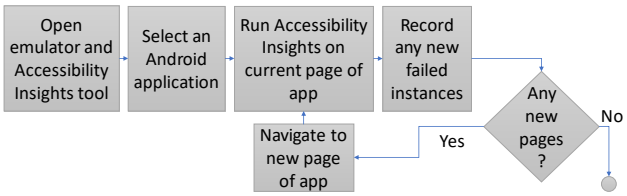


Figure 5.4: Flowchart depicting accessibility analysis method

Table 5.5: Mean, min, and max for accessibility violations

Category	Mean	Min	Max
ActiveViewName	7.93	0	99
ImageViewName	3.51	0	60
TouchSizeWcag	13.3	0	96
EditTextValue	0.291	0	17
ColorContrast	5.36	0	66

4. *EditTextValue*: EditText elements (used to enter text) must expose entered text to assistive technologies. Failing to expose such text results in a violation.

5. *ColorContrast*: Text elements must have sufficient contrast against the background. For example, a string of light gray text on a white background generally represents a violation. This category is different from the others, as it requires the operator's discretion. For example, if a button was intended to not be selectable and was grayed out intentionally, the operator should not record a failed instance.

These instances all relate to accessibility because: (i) they ensure each button is easily selectable for all users (*TouchSizeWcag*); (ii) they ensure the text is easily readable to all users (*ColorContrast*); and (iii) they ensure visually-impaired people can still use the app by having text read to them (*ActiveViewName*, *ImageViewName*, *EditTextValue*).

To apply Accessibility Insights to each app, we executed it simultaneously with the Android Studio emulator[7] as well as on Samsung mobile with Android version of 10, executing each app of interest. This setup allowed us to efficiently test each screen of each app for violations to the five rules described above. We decided to test every screen after preliminary analysis on a sample of 15

[7]Android Studio emulating a Nexus 5X API 28 device.

apps, suggesting that a non-negligible fraction of apps only exhibit certain categories of errors in specific screens. Thus, it is not possible to assess the accessibility posture of an app by examining a small predetermined number of screens.

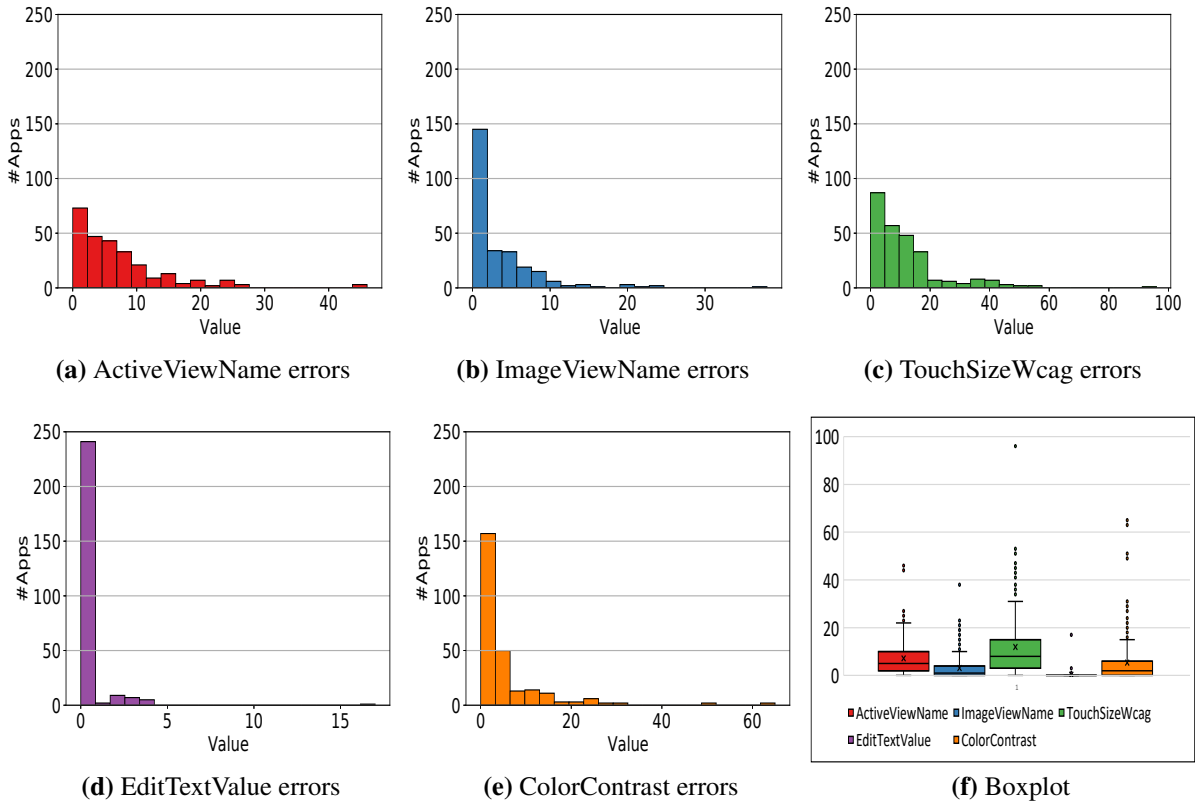
After running Accessibility Insights on the app, the tool returns a list of failed instances, if any. A single page of an app can have multiple failed instances for any number of categories. For some of these applications, each attribute returning a failed instance(s) was recorded only once, whereas for other applications, we recorded all the failed instance(s) throughout the number of screens in applications.

We also identified a number of false positives – apparent violations to rules caused by assets that are either not visible or have height and/or width set to 0. Since the intention of the developer is clearly for such assets not to be visible, we filtered out these instances from our calculation. After performing these adjustments, we then recorded any failed instances that the tool returned for all app pages. This process was repeated for every Android app. The overall process is summarized in the flowchart of Figure 5.4.

5.3.2 Results

Table 5.5 summarizes mean and range for each of the five security metrics under discussion. Value distributions for all categories of errors are depicted in Figure 5.5. Results show that, for most metrics, most apps tend to have a low number of accessibility errors. It is worth noting, however, that most apps exhibit some errors; indeed, there was only one app (`com.logitech.ueboom`) that did not have any failed accessibility instances. The most common error is *TouchSizeWcag*, affecting 93.6% of the apps. The least common error is *EditTextValue*, with an 8.3% failure rate. We remark that relative frequencies of errors are likely to be related to the frequency of the relevant controls; i.e., the most reasonable explanation for the low occurrence of *EditTextValue* is simply that text entry fields are relatively uncommon compared to other controls.

Correlation between metrics:



(a) ActiveViewName errors **(b)** ImageViewName errors **(c)** TouchSizeWcag errors

(d) EditTextValue errors **(e)** ColorContrast errors **(f)** Boxplot

Figure 5.5: Individual distributions of accessibility scores for the IoT applications

Table 5.6: r_s between accessibility metrics

	AVN	**IVN**	**TSW**	**ETV**	**CC**
AVN	1.00	-	-	-	-
IVN	0.542	1.00	-	-	-
TSW	0.444	0.290	1.00	-	-
ETV	-.257	-.044	-.017	1.00	-
CC	0.239	0.310	0.321	0.044	1.00

When considering correlation, a relevant question regards the effect of app size – in terms of the number of screens – on the value of each metric. Generally speaking, an app consisting of more screens creates more opportunities for errors; indeed, analysis reveals moderate Spearman correlation ($r_s = 0.55, p < 0.01$) between the total number of errors and the number of screens. Based on this consideration, when computing the Spearman correlation matrix, we control for app size (i.e., we compute partial correlation using the number of screens as a covariate). Results are presented in Table 5.6. The most relevant finding is that *ActiveViewName* violations exhibit

moderate correlation with *ImageViewName*. This is likely due to the fact that both violations are conceptually similar (failure to provide alternate text for UI elements).

Take-aways: Overall, the trend in accessibility is similar to that in security: while the average number of errors for each category is low, there is a long tail of apps with a significant amount of issues. Furthermore, issues entailing lack of accessible descriptions are correlated across UI element types.

Chapter 6

Conclusions and Future Works

We propose a context-based approach to detect transducers and the respective functionalities of IoT devices using vendor materials. We also show how the functionality of the devices plays an important role in an example smart home application. Specifically, we show policies where the

placement of devices is contingent on the sensitivity of the location and the functionalities of the devices, and how such policies can be specified using NIST NGAC and automatically analyzed to check for conformance and consistencies using Alloy. Our future work involves expanding our scope to include more IoT devices from various vendors and also investigating the problem in the context of industrial IoTs. We would also like to investigate more use cases involving device functionalities, including its application to security audits.

We analyzed the state of Android mobile companion apps. Our study shows that overall software quality along the dimensions of security, privacy, usability, and accessibility is reasonable. However, it also evidences some ecosystem-level shortcomings, such as the presence of known vulnerabilities, over-requesting permissions, and non-negligible accessibility issues. We hope our work can act as a starting point for further investigations in those domains; and facilitate the design of software quality metrics and guidelines that can lead to better app design.

Bibliography

[1] IoT Growth Demands Rethink of Long-Term Storage Strategies, says IDC.

[2] Nest Protect 2ng Gen - Installation and Tech Specs - Google Store.

[3] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access*, 5, 2017.

[4] T.J. OConnor, Reham Mohamed, Markus Miettinen, William Enck, Bradley Reaves, and Ahmad-Reza Sadeghi. HomeSnitch: Behavior Transparency and Control for Smart Home IoT Devices. In *WiSec*, 2019.

[5] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. HoMonit: Monitoring Smart Home Apps from Encrypted Traffic. In *CCS*, 2018.

[6] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. Behavioral Fingerprinting of IoT Devices. In *ASHES*, 2018.

[7] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In *ICDCS*, 2017.

[8] S. Marchal, M. Miettinen, T. D. Nguyen, A. Sadeghi, and N. Asokan. AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication. *IEEE J SEL AREA COMM*, 37, 2019.

[9] Jorge Ortiz, Catherine Crawford, and Franck Le. DeviceMien: Network Device Behavior Modeling for Identifying Unknown IoT Devices. In *IoTDI*, pages 106–117, 2019.

[10] V. Bhosale, L. De Carli, and I. Ray. Detection of Anomalous User Activity for Home IoT Devices. In *IoTBDS*, 2021.

[11] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Matthew Roughan, and Vijay Sivaraman. Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles. In *IoT S&P*, 2018.

[12] Andy Dolan, Indrakshi Ray, and Suryadipta Majumdar. *Proactively Extracting IoT Device Capabilities: An Application to Smart Homes*. 2020.

[13] Hui Liu, Juanru Li, and Dawu Gu. Understanding the security of app-in-the-middle IoT. *Computers & Security*, 97:102000, October 2020.

[14] Apps are Creating Mobile Security Vulnerabilities for IoT - How Bad is It?, February 2017.

[15] Accessibility and the Internet of Things, December 2018.

[16] Prachi Kumar. An Introduction to N-grams: What Are They and Why Do We Need Them?, October 2017. Section: AI.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, 2019. arXiv: 1810.04805.

[18] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Theophilus A. Benson, Matthew Roughan, and Vijay Sivaraman. Verifying and Monitoring IoTs Network Behavior using MUD Profiles. *arXiv:1902.02484 [cs]*, 2019. arXiv: 1902.02484.

[19] Combining MUD Policies with SDN for IoT Intrusion Detection. Budapest Hungary.

[20] Extending MUD Profiles Through an Automated IoT Security Testing Methodology. 7.

[21] Abayomi Otebolaku and Gyu Myoung Lee. A Framework for Exploiting Internet of Things for Context-Aware Trust-Based Personalized Services. *Mobile Information Systems*, 2018, 2018.

[22] Lina Yao, Boualem Benatallah, Xianzhi Wang, Nguyen Khoi Tran, and Qinghua Lu. Context as a Service: Realizing Internet of Things-Aware Processes for the Independent Living of the Elderly. In Quan Z. Sheng, Eleni Stroulia, Samir Tata, and Sami Bhiri, editors, *Service-Oriented Computing*. Springer International Publishing, Cham, 2016.

[23] Wayne Jansen. Research Directions in Security Metrics. Technical Report 7564, NIST, 2009.

[24] Daniel E. Krutz, Nuthan Munaiah, Andrew Meneely, and Samuel A. Malachowsky. Examining the relationship between security metrics and user ratings of mobile apps: A case study. In *WAMA*, 2016.

[25] Yiming Jing, Gail-Joon Ahn, Ziming Zhao, and Hongxin Hu. RiskMon: Continuous and automated risk assessment of mobile applications. In *CODASPY*, 2014.

[26] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. Whyper: Towards automating risk assessment of mobile applications. In *USENIX Security*, 2013.

[27] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. In *ACM CCS*, 2012.

[28] Alessio Merlo and Gabriel Claudiu Georgiu. RiskInDroid: Machine Learning-Based Risk Analysis on Android. In *3IFIP International Conference on ICT Systems Security and Privacy Protection*, 2017.

[29] Marcelo Medeiros Eler, José Miguel Rojas, Yan Ge, and Gordon Fraser. Automated accessibility testing of mobile apps. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 116–126. IEEE, 2018.

[30] Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek. Accessibility issues in android apps: state of affairs, sentiments, and ways forward. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1323–1334. IEEE, 2020.

[31] Aleksander Bai, Kristin Skeide Fuglerud, Rannveig Alette Skjerve, and Till Halbach. Categorization and comparison of accessibility testing methods for software development. 2018.

[32] Nest Cam Indoor - Home Security Camera - Google Store.

[33] Nest Cam Indoor - Installation and Tech Specs - Google Store.

[34] Arlo Smart Home. Arlo Video Doorbell: How To Install, November 2019.

[35] Nest Learning Thermostat - Installation and Tech Specs - Google Store.

[36] Yashu Seth. BERT Explained – A list of Frequently Asked Questions, June 2019.

[37] Rani Horev. BERT Explained: State of the art language model for NLP, November 2018.

[38] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[39] Ritchie Vink. Algorithm Breakdown: Affinity Propagation - Ritchie Vink.

[40] Sara N. Matheu, José L. Hernández-Ramos, Antonio F. Skarmeta, and Gianmarco Baldini. A Survey of Cybersecurity Certification for the Internet of Things. *ACM Computing Surveys*, 53(6):1–36, February 2021.

[41] ISO/IEC. ISO/IEC 20924:2018(en), Information technology — Internet of Things (IoT) — Vocabulary.

[42] What is accessibility? - Learn web development | MDN.

[43] SpeechRecognition.

[44] Nest x Yale Lock - Key-Free Smart Deadbolt - Google Store.

[45] Video Doorbell | Arlo Home Security | Arlo.

[46] Arlo Pro 3 Floodlight Camera | Arlo Wireless & AC-Powered Security Cameras.

[47] Samsung Smartcam Video Surveillance Camera: SNH-P6410BN | Samsung US.

[48] hanxiao/bert-as-service: Mapping a Variable-Length Sentence to a Fixed-Length Vector Using BERT Model.

[49] Daniel Liu. play-scraper.

[50] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[51] Georgia Kapitsaki and Modestos Ioannou. Examining the Privacy Vulnerability Level of Android Applications:. In *Proceedings of the 15th International Conference on Web Information Systems and Technologies*, 2019.

[52] Jianguo Jiang, Song Li, Min Yu, Kai Chen, Chao Liu, Weiqing Huang, and Gang Li. MR-Droid: A Multi-act Classification Model for Android Malware Risk Assessment. In *IEEE MASS*, 2018.

[53] Ali Alshehri, Pawel Marcinek, Abdulrahman Alzahrani, Hani Alshahrani, and Huirong Fu. PUREDroid: Permission Usage and Risk Estimation for Android Applications. In *ICISDM*, 2019.

[54] Ridho Alif Utama, Parman Sukarno, and Erwid Musthofa Jadied. Analysis and Classification of Danger Level in Android Applications Using Naive Bayes Algorithm. In *ICoICT*, 2018.

[55] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *PLDI*, 2014.

[56] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Danfeng, Yao, and Murat Kantarcioglu. Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects. 2019.

[57] Mobile security framework, 2020.

[58] GitHub - linkedin/qark: Tool to look for several security related Android application vulnerabilities, 2019.

[59] R. English and C. M. Schweik. Identifying success and tragedy of floss commons: A preliminary classification of sourceforge.net projects. In *FLOSS ICSE Workshops*, 2007.