Leonardo Babun*, Z. Berkay Celik, Patrick McDaniel, and A. Selcuk Uluagac

Real-time Analysis of Privacy-(un)aware IoT Applications

Abstract: Users trust IoT apps to control and automate their smart devices. These apps necessarily have access to sensitive data to implement their functionality. However, users lack visibility into how their sensitive data is used, and often blindly trust the app developers. In this paper, we present IoTWatch, a dynamic analysis tool that uncovers the privacy risks of IoT apps in real-time. We have designed and built IoTWatch through a comprehensive IoT privacy survey addressing the privacy needs of users. IoTWatch operates in four phases: (a) it provides users with an interface to specify their privacy preferences at app install time, (b) it adds extra logic to an app's source code to collect both IoT data and their recipients at runtime, (c) it uses Natural Language Processing (NLP) techniques to construct a model that classifies IoT app data into intuitive privacy labels, and (d) it informs the users when their preferences do not match the privacy labels, exposing sensitive data leaks to users. We implemented and evaluated IoTWatch on real IoT applications. Specifically, we analyzed 540 IoT apps to train the NLP model and evaluate its effectiveness. IoTWatch yields an average 94.25% accuracy in classifying IoT app data into privacy labels with only 105 ms additional latency to an app's execution.

Keywords: Internet of Things, privacy, security, NLP.

DOI 10.2478/popets-2021-0009
Received 2020-05-31; revised 2020-09-15; accepted 2020-09-16.

1 Introduction

Users install IoT apps to manage and control IoT devices such as smart thermostats, door locks, and cameras. These apps have access to sensitive data to implement their functionalities, communicate to external servers, and send notifications to users [4, 11, 35, 44, 50]. Recent studies have shown that IoT apps leak sensitive information to unauthorized parties [3, 10, 19, 41, 42, 71, 76–80], and many IoT apps transmit data to remote servers for data visualization and profiling user behaviors [1, 2, 69, 74, 75], such as energy usage of appliances. However, users have no knowledge and control over what type of sensitive data the apps access or who else see these data [15, 16, 68, 70–73].

Most attempts to date to address sensitive data leaks focus on analyzing the app's source code to extract permissions [29, 43, 53]. Others use static analysis to identify sensitive data-flows [10], inter-rule vulnerabilities in IoT deployments [54], and apply dynamic analysis techniques to isolate sensitive data within sandboxes [19]. These approaches, albeit useful, have limitations that lead to under- or over-approximating sensitive data-leaks and lack notifying the users to help them make informed decisions about the apps. More specifically, none of the earlier works (a) consider the privacy preferences of users into their implementations, and (b) address the sensitive data leaks through *strings* at taint-sink contents; for instance, we found that, out of 540 analyzed IoT apps, 64% of them leaked sensitive data through simple strings such as "the door is unlocked." While there exist privacy tools for other platforms such as mobile phones [5, 17, 25, 37–39], these techniques cannot be directly applied to IoT apps due to their unique characteristics in terms of app structures, introducing challenges in program analysis.

In this paper, we present IoTWatch, a dynamic analysis tool that uncovers the privacy risks that IoT apps pose to the privacy preferences of the users in real-time. Indeed, IoTWatch was designed based on a comprehensive privacy survey with human subjects that used various IoT devices. The study aimed at understanding the privacy concerns and expectations of the users when they installed IoT apps. At install-time, IoTWatch first provides IoT app users with an easy-to-use, intuitive interface that allows them to specify their privacy preferences (e.g., location and device states). Then, it automatically adds extra code to the apps' source code to collect the data sent out of the

***Corresponding Author: Leonardo Babun:** Florida International University, E-mail: lbabu002@fiu.edu
Z. Berkay Celik: Purdue University, E-mail: zcelik@purdue.edu
Patrick McDaniel: Pennsylvania State University, E-mail: mcdaniel@cse.psu.edu
A. Selcuk Uluagac: Florida International University, E-mail: suluagac@fiu.edu

IoT app at runtime. The collected data is used by IoTWatcH to classify the app information into user-defined (also customizable) privacy labels through Natural Language Processing (NLP) techniques. The privacy labels provide the users with a more intuitive mechanism to understand how IoT apps handle their private information. Also, IoTWatcH analyzes the privacy preferences of users to uncover sensitive data leaks. Finally, IoTWatcH notifies the users about the sensitive data-leaks when they validate the users' privacy preferences, allowing the users to make informed decisions about their privacy.

To evaluate IoTWatcH, we trained an NLP model with taint-sink data strings extracted from 380 SmartThings market apps. The model was used to classify taint-sink content (e.g., "the door is locked", and "kitchen lights are turned off") to user privacy preferences. Then, we analyzed 160 different Samsung SmartThings apps to evaluate the accuracy of IoTWatcH at runtime. IoTWatcH successfully classified 146 IoT strings to privacy preferences with an average accuracy of 94.25% and a precision of 95%. Additionally, IoTWatcH identified 35 IoT apps that leaked sensitive data to unauthorized recipients. IoTWatcH yielded minimal overhead to the IoT apps execution, introducing on average of 105 ms additional instrumentation latency.

Summary of Contributions. The contributions of this work are as follows:

- We conducted a comprehensive IoT privacy survey with 123 IoT users through a set of structured questions.
- We developed IoTWatcH, a dynamic privacy analysis tool for IoT apps. IoTWatcH provides users with a privacy interface and collects app data at runtime based on user preferences. Then, it analyzes the collected data and informs users when a data leak matches the user's privacy preferences.
- We collected data from 380 real IoT apps to train an NLP model. We used the model to evaluate IoTWatcH's performance on other 120 market and 40 malicious IoT apps. IoTWatcH classified 146 IoT taint-sink content into correct privacy labels with 94.25% accuracy while imposing an additional 105ms latency to app execution on average. Additionally, IoTWatcH identified 62 sensitive data leaks to unauthorized third parties in 35 IoT apps (29 via messaging and 33 via Internet taint-sinks).
- We made IoTWatcH freely available to the community at `https://iotwatch.appspot.com/`.

Organization. In Section 2, we articulate the privacy issues in IoT apps through a use case, and present the definitions and threat model. In Section 3, we present the results of the IoT privacy survey with 123 users. In Sections 4 and 5, we give an overview of IoTWatcH, and present IoTWatcH's architectural details. In Section 6, we provide the implementation details. Then, we evaluate IoTWatcH and show its effectiveness and performance in Section 7. Finally, we discuss the related work in Section 8, and conclude the paper in Section 9.

2 Background and Threat Model

2.1 Anatomy of an IoT App

IoT applications from different platforms may use different programming languages. For instance, Samsung SmartThings apps are written in Groovy [31], while OpenHAB [36] applications use Domain Specific Language (DSL) developed in Java. In other examples, Apple Homekit [66] apps can be written in Swift or Objective C, while Windows IoT [67] use C#, C++, or even Python. Despite this diversity, IoT applications share a similar structure [10]. In general, IoT apps have a *description block* that informs the users specifics of the app itself (e.g., the app functionality, devices it controls, etc.). We expect that developers use these description blocks to also inform the users regarding the specific types of sensitive information that the app will have access to on the IoT systems and who this information will be shared with. In that way, the users can take informed decisions regarding their privacy before installing the apps. The apps also include a *permission block* where the user adds what she desires to control with the app (e.g., smart thermostat). Also, the permission block is used to define the information used to specify device settings (e.g., min and max thermostat temperatures) and app notification recipients (e.g., phone numbers for push notifications). We study this permission block during our analysis as it contains highly private information from users and the IoT systems, including characteristics of the smart devices and their settings. Also, we correlate information defined by the user in these blocks to verify that the sensitive information is being sent to authorized recipients (e.g., user-defined phone numbers). The IoT apps also include an *event subscription block*, where developers match the specific functions that are triggered by different device states (e.g., the function f1 is executed when the presence sensor triggers a "present" state). Finally, the *event handlers* define these functions and the app logic that is executed whenever

```
1:  // smart-lock-control app: "controls the smart lock"
2:  // Permissions Block ❶
3:        Device: smart_lock sl, presence pr
4:        User-defined inputs: phone
5:  // Events Subscription ❷
6:        subscribe(pr, "present", f1)
7:        subscribe(pr, "not present", f2)
8:  //Events Handler ❸
9:        f1(){…}
10:       f2(){  sl.lock()
11:              notifyUser(sl.state, phone)
12:              leakInfo()}
13: // Sends notification to the user
14:       notifyUser(state, number){
15:          sendSMS("Your lock is: " + state, phone)
16:          POST("http://support.com", sl.getLocation())
17: // Leaks sensitive data to attacker ❹
18:       leakInfo(){
19:              sendSMS("Nobody is Home", 123-456-7890)}
20:
```

Fig. 1. An example IoT app leaking sensitive data to a hard-coded phone number and implementing insecure HTTP calls.

they are called. In our work, we study the event handlers as they define the way how sensitive information is handled within IoT apps. Also, event handlers specify the sink functions that send the sensitive data out of the apps to external parties. As a summary, we target description blocks, permission blocks, event subscription blocks, and event handlers as the most important app constructs to investigate for any sensitive data leakage from IoT apps.

2.2 Problem and Challenges

We use an example IoT app source code derived from a `smart-lock-control` app (Figure 1) to illustrate the privacy concerns in IoT apps. The expected behavior of the app is to lock the door and notify to user-defined contacts that the door is locked when the user leaves the house. At install-time, the user grants permissions to the smart lock, presence sensor, and enters a phone number for messaging notifications (❶). The app subscribes to two event handlers `f1` and `f2` to implement the app functionality. The event handlers are invoked based on the presence sensor's state (user-present and user-not-present) (❷). When the user leaves home, "not-present" event handler (i.e., `f2`) locks the door, sends a message notification, and transmits out the door lock state to a remote server (i.e., http://support.com) (❸). However, the actual behavior of the app adds a piece of code that invokes a function (i.e., `leakinfo()`) sending a *string* that contains "Nobody is Home" to a hard-coded phone number (❹). This string is highly private and informs an adversary that the house is empty.

Requirements for IoT Privacy Analysis. The diversity of information handled by IoT (e.g., location, user behavior, user preferences) makes these systems vulnerable to privacy risks, hence the need for a specific privacy analysis tool for IoT. However, IoT platforms do not provide the means for such an analysis. There exist no security mechanisms or methodologies to analyze and understand the specific types of information that an IoT application handles and discloses to support their logic and functionalities. Previous works have also noted that IoT systems do not explicitly inform the users regarding their utilization of the sensitive information, and researchers are trying to propose solutions to these issues [45].

There exist systems to identify sensitive data-flows in IoT apps. For instance, SAINT is a static tool that uses taint analysis to identify sensitive data-flows in IoT apps [10]. FlowFence, a dynamic system, uses quarantined modules to enforce data-flow policies on the use of sensitive data [19]. Albeit useful, these approaches are limited in precision and the number of privacy policies enforced as they both focus on taint data. Hence, SAINT and FlowFence are oblivious if an app leaks sensitive data through implicit programming constructs (i.e., description blocks, event handlers) via developer- or user-defined strings. For instance, the string "Nobody is home" leaked through `leakinfo()`). Meanwhile, our analysis of 540 IoT market apps showed that 64% of apps potentially leaked sensitive data through strings that did not include any tainted data, yet the string was sensitive. Indeed, static systems like SAINT, iRuler [54], and privacy tools for trigger-action platforms [27] fail to detect sensitive data leaks through dynamic method invocation [31]. Additionally, there are no approaches that allow users to closely examine their sharing and privacy preferences over individual IoT apps. For instance, a user may desire to share her energy usage data with a third party in a specific IoT app yet she wants to restrict sharing all other sensitive information with third-parties. This requires a personalized privacy setting for each app that gives control to the users over what to share. Finally, previous research [9, 10] have demonstrated that app analysis tools that may be effective for other systems like Android cannot be directly applied to IoT as they lead to substantial false positives due to important architectural and programming differences between their apps.

Challenges of an IoT Privacy Solution. In contrast to the previous approaches [9, 10, 19, 27, 54], IoTWATCH tracks runtime data-flows, and process the flow-content to determine whether a data-flow constitutes a privacy concern or not. Such a runtime analysis overcomes the limitations of static tools [10] that do not consider dynamically generated taint variables or hide them via implicit constructs. Additionally, IoTWATCH provides a user with an interface that allows them to configure their privacy settings for each app and informs the users about its findings.

We provide a detailed comparison of IoTWatcH with other privacy tools for IoT and Android apps in Section 8. However, to support its capabilities, IoTWatcH needs to overcome several technical challenges. First, it needs to capable of analyzing the source code of IoT apps and identifying privacy-relevant information beyond taint data, which is a deeper investigation than inflexible static approaches. Second, it needs to implement mechanisms to be able to collect the privacy information for further analysis. Static tools like SainT may be a good starting point for source code analysis in IoTWatcH. However, such static tools require extensive modifications to identify non-taint privacy data within apps. Similarly, SainT or other static analysis tools do not support app instrumentation, which is required for real time dynamic analysis. Also, IoTWatcH analysis requires the implementation of NLP models to classify privacy data into simpler privacy labels that can be easily understood by the users. This process requires building specific IoT corpus for training purposes. Finally, as IoTWatcH implements real-time analysis and notification systems for the user, it requires of specific APIs to support and further secure the communications between IoT apps and IoTWatcH's servers without imposing a considerable overhead.

2.3 Definitions

Taint-Sinks. IoT programming platforms define specific call-sites to transmit data [36, 51]. We focus on two main call-site types; messaging and Internet. These call-sites require two types of information: the *recipient* and the *content*. The recipients define where the content is being sent to, and the content defines the data sent in the form of strings.

Privacy Labels. We define four different privacy labels (i.e., date-time, device-info, location, and user-behavior) to classify the content of the taint-sinks. These privacy labels are selected based on users responses in our privacy survey detailed in Section 3.

Privacy Profile. The collection of privacy and notification preferences identifies the users' privacy profiles. These profiles helps IoTWatcH to selectively instruments the apps to collect data for privacy analysis.

Privacy Leaks. We consider a privacy leakage happens when data is sent to external parties, and it is neither informed to nor acknowledged by the user. For instance, if an app transmit location information and a user is not informed, then it constitutes a privacy leak.

2.4 Threat Model

We consider IoT apps that access sensitive data without user consent. Second, we consider data leaks, through Internet and messaging taint-sinks, transmitted out of an app to unauthorized recipients, and due to the carelessness of developers. Lastly, we consider app transmitting sensitive data in clear-text (through HTTP) to notify users.

We do not consider safety and security violations in IoT apps as they were already studied before [12, 13]. Additionally, we do not track data-flows via push notifications or taint-sinks that are authorized via OAuth (e.g., a user authorizes a third-party service through OAuth protocol to share the device states for data visualization) as they are considered a different external capability that supports the IoT functionality. Finally, we do not consider apps using obfuscation or encryption techniques to hide the content of the data flows; yet, we discuss potential measure for these techniques in Section 5.1.1). We note that we did not find one single app out of 540 IoT apps we analyzed that encoded the content of their messaging or Internet taint-sinks.

3 Privacy Survey

In this work, we first conducted a comprehensive survey to understand the privacy concerns of users when they use various IoT devices and apps. The entire survey was authorized by the institutional ethics review board (IRB) and occurred between April 2019 and May 2019.

Survey Goals. With this survey, we aimed to answer the following questions: (1) what are the privacy concerns of IoT users?, (2) is there a need for privacy analysis tools designed for IoT?, and (3) what are the user expectations, in terms of usability requirements, for privacy analysis tools? Specifically, we expected to understand the types of privacy sensitive information that is most relevant to the IoT users so that it can be reflected in the design of the privacy choices offered by IoTWatcH. Also, we expected to gain valuable understanding about the users' desire to have a flexible design that could be captured by the customizable privacy profile offered by IoTWatcH at install time rather than enforced fixed privacy policies for all the data. In addition, we wanted to understand the importance of prompt privacy feedback to the users so that we can incorporate real-time analysis into our approach. Finally, the survey would reveal the specific types of app communications and potential leaks that most users would want to track with our privacy tool. For all these, we created 26 dif-

ferent questions organized into three categories (sample questions in Appendix A). These categories align with three specific privacy survey goals: (1) the characterization of the participants, (2) privacy concerns of IoT users, and (3) the need for IoT privacy analysis tools and their usability requirements. We present the profiles of participants and our key findings below.

Survey Overview and Recruitment. We made the survey available to participants for four weeks. The users could access the survey and submit their responses via an online questionnaire hosted on Google Forms [22]. The questionnaire included single choice questions (e.g., yes, no), multiple-choice questions, and free-form questions. We made all the questions required except for the ones requesting an additional explanation from users in the form of free-text input. Finally, we recruited the participants via recruitment emails in our institutions. The emails included a brief description of the survey and a link to the online form. Finally, the survey was designed so the participants would spend an average of 15 minutes from start to finish.

Participant Demographics. We recruited 123 participants. In the group, 100 (81.3%) were male, and 23 (18.7%) were female. Also, 69 participants (56.1%) were in the range of 18-25 and 37 (30.1%) were in the range of 26-35 years old. The remaining 17 (13.8%) participants were 36 years or older. The majority of the participants (110 (89%)) had at least completed some bachelor-level courses, and 37 (30%) were enrolled in graduate-level courses. Finally, 100 participants reported being part of an educational institution. A total of 112 (91.05%) users shared that they currently use or are planning to use IoT devices in their homes. Finally, 19 (15.4%) participants knew how to develop their own IoT apps while 82 (66.7%) participants had previous experience installing apps from an IoT market or via using the source code of IoT apps available online.

Ethics and Analysis Approach. The IRB of our institutions approved the privacy survey. The participants had to be over 18 years old, and the survey did not collect any personal information from the participants other than an institutional email address that was requested for compensation purposes. We did not allow participants to submit multiple responses, but they had the chance to change their answers anytime before the survey closing date. No participants changed their original replies. We processed and accepted all the responses obtained from the participants. Further, we directly quantified the responses from single- and multiple-choice questions. Finally, we used two independent researchers to analyze the free-from responses remove answers flagged as potential outliers.

3.1 Discussion of Survey Results

Privacy Concerns of Users. The participants were concerned about their sensitive information being inadvertently leaked to unauthorized parties. To detail, 65 (52.8%) participants felt uncomfortable about their personal data (e.g., their password to login into edge devices), their behavior and habits (e.g., when they go to sleep), location (e.g., whether they are home or away), device's settings (e.g., heating value of a thermostat) and time configuration (e.g., when kids leave home), and device states (e.g., whether the door is locked or not) being handled by IoT systems. Also, at least 89 (72.4%) participants expressed that they are aware of IoT apps collecting their sensitive information and sending it to remote servers for data analytics such as profiling their energy usage and for advertisement purposes [2]. Finally, 103 (83.7%) participants expressed privacy concerns on the use of IoT systems, and 88 (71.5%) mentioned having heard about the privacy issues in IoT systems from the news or other media.

The Need for Real-time Privacy Tools. A total of 112 (91.1%) participants raised broad concerns about the lack of an existing tool that informs the users regarding the potential privacy risks of IoT systems in real-time. Also, 119 (96.74%) participants found the idea of using a tool to uncover privacy risks in IoT highly desired and expected. Our participants were happy to use automatic tools that instruments IoT apps to enable prompt privacy analysis and results. Out of the 123 participants, 119 (96.74%) expressed their support for this option if the tool is verified by the IoT platform.

Users' Expectations. To understand the characteristics of an easy-to-use privacy tool, we asked participants to identify their preferences on different types of sensitive information they desire to be informed. The participants noted four different privacy labels, which we later use them in our design and analysis of IoTWatch. The "Device-info" was approved by 109 (88.6%) participants to define device information (e.g., device states or device type). The "User-behavior" received 103 (83.7%) positive responses to identify information related to user options and activities (e.g., what the user does, how the user configure his/her IoT system). The "Location" was approved by 110 (89.4%) participants to identify the location of devices and users (e.g., user is at home, the kitchen switch is turned on). Lastly, the"Date-time" was

Expectations of survey's participants	% Agreement
Real-time privacy analysis	91.6%
Configurable privacy preferences	87.5%
Control over unauthorized data disclosure	86.6%
On-demand privacy controls	81.6%
Timely privacy notifications	85.3%
Inter-rater Reliability	**86.5%**

Table 1. Participant responses when asked about their expectations from a privacy analysis tool, which influenced IoTWatch's design. The percentage of agreement among the survey's participants showed a strong inter-rater reliability.

approved in 100 (81.3%) responses to identify timing-related information (e.g., the time a door is locked).

3.2 Survey Takeaways

Our findings shed light on the need for better and flexible privacy management practices between users and IoT markets, which mitigate the privacy risks of IoT apps based on user privacy preferences. Table 1 presents the needs of the users that align with their control over their privacy preferences. We obtained strong inter-rater reliability of 86.5%. Next, we summarize the design features that influenced the development of IoTWatch based on participant responses.

Real-time Privacy Analysis. The participants reflected their opinions about being aware of what information leaves an IoT system and where it is transmitted to in real time. Additionally, they reported that they expect to have minimal configurations when new devices are dynamically plugged into their IoT systems and new IoT apps are installed. Further, to support the real-time analysis in IoTWatch, we use machine learning and implement effective communication between the IoT applications and our privacy analysis server via a specific API.

Configurable Privacy Preferences. The participants noted their concerns about the lack of privacy preference controls, which limit their willingness to use IoT devices. For instance, they prefer to have intuitive categories that show the information-type leaving the IoT system such as "the door is locked" associated with a specific privacy label. For this, we offer the user a privacy interface from with which they can configure and control their privacy expectations on the IoT systems. We used the survey responses to design a friendly and functional interface that incorporates the privacy needs of users.

Unauthorized Data Disclosure. The participants liked having better control over the disclosure of any private information. For instance, they prefer to be notified when IoT systems share their data with other

```
1:  // iotwatch-enabled smart-lock-control app
2:  Description: "controls the smart lock"
3:  // Permissions Block
4:        Device: smart_lock sl, presence pr
5:        User-defined inputs: phone C1
6:  // Event Subscription
7:        subscribe(pr, "present", f1)
8:        subscribe(pr, "not present", f2)
9:  //Event Handler
10:       f1(){…}
11:       f2(){ sl.lock()
12:             notifyUser(sl.state, phone)
13:             leakInfo()}
14: // Send notification to the user
15:       notifyUser(state, number){
16:             sendSMS("Your lock is: " + state, phone) C2
17:             sendPush("App is using data of type:"
18:                      classification)} C3
19: // Leak sensitive data to attacker
20:       leakInfo(){
21:             sendSMS("Nobody is Home", 123-456-7890) C4
22:             sendPush("Privacy Risk of type:" context
23:                      "with data of type:" classification)} C5
```

Fig. 2. An example of an IoT app instrumented by IoTWatch to support the notification interface.

parties. Additionally, participants desired mechanisms that notify hard-coded messaging recipients, which mitigates the consequences of privacy violations. For this, we implemented a privacy analysis tool that uses NLP to track sensitive information leaving the IoT apps without the user authorization or knowledge. The analysis also checks if the information recipients have been previously disclosed to the user.

On-demand Privacy Analysis. The majority of the participants acknowledged the effectiveness of configurable privacy preferences over the IoT apps. They noted these configurations would help them have a better experience with a few numbers of notifications and minimal runtime delay. In IoTWatch, at install-time, users configure their privacy profiles which guides IoTWatch's real-time analysis. The profiles also define the amount of feedback and notifications the users receive from IoTWatch. Finally, to improve usability, we enable the users to update their privacy profiles anytime after the install-time.

4 IoTWatcH Illustrated

We use the `smart-lock-control` app in Figure 1 to illustrate the logical steps of IoTWatcH in Figure 2. First, the code-instrumentor adds extra code to the app's source code to implement a user interface (UI). The UI allows a user to select a set of privacy-labels (location, user-behavior, device-info, and date-time) and taint-sink types (messaging and Internet) at install time to track at run-time. These privacy settings define the user's privacy profile and are used to notify the user when her sensitive data is transmitted out of an app. Additionally, the code instrumentor adds extra logic to send app data to IoTWatcH server at run-time. The data includes user-defined input variables (i.e., phone) (C1) and the content and recipients of the functions used to send data out of the app. The `smart-lock-control` app includes two messaging taint-

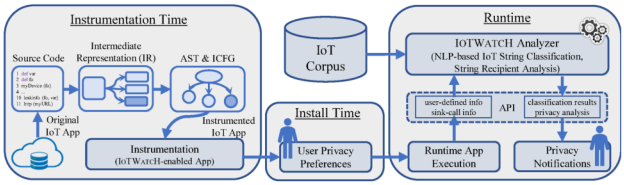


Fig. 3. Overview of IoTWatch's architecture. We instrument IoT apps to enable IoTWatch. Then, the user selects their privacy preference, and IoTWatch collects and analyzes IoT app data to uncover data leaks in real time.

sinks (C2 and C4). IoTWatch instruments data-flow of both messaging taint-sinks to collects content (i.e., "Your lock is: " + state, and "Nobody is Home") and their recipients (i.e., phone and 123-456-7890). Lastly, the collected data is processed and the instrumentor inserts extra code to notify users with IoTWatch's results (C3 and C5).

The user installs the instrumented app, which transmits its data to the IoTWatch server once a specified data-flow is flagged. This information permits IoTWatch to identify the type of sensitive information an IoT app uses, and combines this information with the app data such as its description block to uncover sensitive data leaks.

We verify whether the sensitive data is transmitted to recipients previously authorized or acknowledge by the user via matching the user-defined inputs to the recipients of the sensitive information. For instance, the data is sent to a user-defined phone number (C1) for the first potential leakage (C2); and the data is sent to a hard-coded phone number for the second possible leakage (C4), which indicates a potential privacy violation to the user. Furthermore, IoTWatch verifies that the app uses sensitive information regarding the devices (i.e., "Your lock is:") and location (i.e., "Nobody is Home") by analyzing the leaked content with NLP-based techniques.

Lastly, IoTWatch informs the user of its findings to make informed decisions. For the first potential leak, IoTWatch sends a push notification with the privacy-labels of the information included in the message (C3). For the second potential leak, IoTWatch informs the privacy content of the message and alerts the user about potential privacy violations (C5).

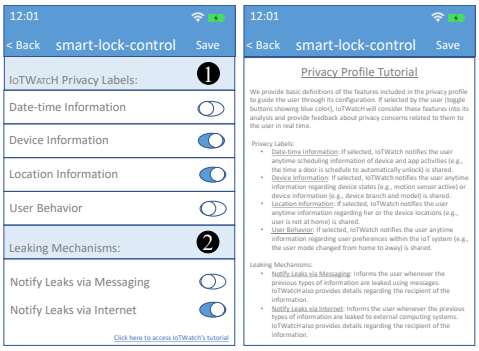
5 IoTWatcH Architecture

5.1 Source Code Analysis

Figure 3 details IoTWatch's architecture. IoTWatch's code instrumentor constructs an intermediate representation (IR) from the source code of an original IoT app. Specifically, the instrumentor uses the IR to obtain the Abstract Syntax Tree (AST) of the app. As noted in Section 2.1, we found that the apps from the major IoT platforms (e.g., Samsung Smartthings, openHAB, Apple's Home Kit) share a similar structure, as also detailed in previous works [10]. While the SmartThing apps are explicitly using the four programming blocks (i.e., description, permission, event subscription, and event handler), other platforms follow a similar approach. That is, they all follow the event-driven programming structure of the sensor-computation-actuator model. The IR builds a representation of this computational model [10, 12], and enables the design of broader solutions that can be implemented for different IoT programming platforms. This is the main reason why we built an IR of the apps on which IoTWatch operates. Further, the IR would allow the IoTWatch's analysis to be extended to other programming platforms by other researchers as it supports a generic analysis tool that converts their source code to the IR. Once the IR is obtained, IoTWatch implements custom *node visitors* algorithm to build the app's Inter-procedural Control Graph (ICFG). The ICFG is used for two purposes. First, it enables us to identify user-defined inputs in the app's permission block, and the taint-sink recipient and content. Second, it is used to add extra code to collect and transmits this data to the IoTWatch's server and to implement real-time push notifications to inform the users about IoTWatch's analysis results.

IoTWatch's instrumentor derives from Saint tool. However, we had to extensively modify this tool to enable the privacy component of IoTWatch. The instrumentor uses flow-sensitive analysis and flags nodes (1) executing functions that generate or modify sensitive information (e.g., "user-defined input: phone") and (2) send sensitive information out of the apps via sink functions (e.g., "sendSMS("The house is empty", phone)". In addition, the instrumentor insert logs to capture and send that data to IoTWatch's server and implements a novel privacy UI so the user can create her privacy profile, something that is currently missing from IoT platforms. The code instrumentor groups apps collected data into two categories: (1) app information, and (2) sink-call information. We detail each of them as follows:

App Information. The code instrumentor visits the permission block in IoT apps and extracts information defined by the user (see Figure 2). Specifically, it collects user-defined inputs that may be used as parameters in taint-sinks (e.g., a phone number for notifications). As



(a) (b)

Fig. 4. (a) IoTWatcH enables users to define their privacy preferences, and (b) sample tutorial to help users understand the meaning of the privacy profile's components.

we detail later in Section 5.2, this data is matched with the information extracted from sink-calls (i.e., recipients) to identify, for instance, whether the data sent to unauthorized recipients (i.e., not defined by the user), which may lead to potential privacy issues for the user.

Taint-Sink Information. It includes the content and recipients of messaging and Internet call sites. Turning to app source code depicted in Figure 2, IoTWatcH obtains the content "Your lock is: + *state*" from the first messaging function in Line 16, and the content "Nobody is Home" from the second messaging function in Line 21. As per the recipients, it extracts the recipient's value contained in the variable *phone* from the first messaging method (Line 16) and the hard-coded phone number "123-456-7890" from the second messaging function (Line 21). IoTWatcH uses taint-sink contents to inform a user of the sensitive data type.

Purpose of Collected Data. We implement an NLP-based model to analyze the content of taint sinks and classify them into four privacy labels. Furthermore, IoTWatcH matches the recipient information extracted from messaging and Internet taint-sinks with app information to uncover sensitive data leaks. The app information is either entered by the user, or informed by the developer via the app's description block and approved by the user. In cases where the sink-call (i.e., messaging or Internet) is executed using unauthorized recipients, a *leak* is flagged and the user is informed. Additionally, we check whether Internet taint-sinks send data in clear-text, confirming sensitive data is not accessed by potential eavesdroppers.

5.1.1 Selective Code Instrumentation

IoTWatcH performs a selective code instrumentation to implement privacy analysis on the sensitive data that is of interest to the user, which also reduces the number of real-time notifications. Also, selective instrumentation permits the analysis of encrypted IoT strings.

Privacy User Interface. The instrumentor adds additional code to implement a UI and create a privacy profile of the user. Figure 4(a) shows the privacy user interface presented to the user during install-time, which illustrates the selective code instrumentation options of the IoTWatcH-enabled app. IoTWatcH's instrumentor does not impact the UI experience of the IoT app at runtime, but offers new privacy features at install-time not available in the original app. The instrumented app offers the user the possibility to create a privacy profile and receive notifications regarding specific privacy labels that are of interest to the user (❶). Also, it allows for selecting which privacy concerns (e.g., option to notify leaks from messaging or Internet taint-sinks) must be analyzed and informed by IoTWatcH (❷). Such a design supports the expectations of the IoT app users with (1) configurable privacy preferences, (2) on-demand privacy controls, and (3) timely privacy notifications (Table 1) as were summarized in Section 3. The user interface also includes a link to a tutorial that explains the different settings of the privacy profile. The justification behind this design rationale is to to facilitate the user experience regarding the setup and use of the privacy profiles, their components and meanings, and how they can be included into IoTWatcH's analysis, as shown in Figure 4(b). The tutorial can be easily accessed from the privacy user interface anytime. Also, as we target open-source IoT platforms, the use of selective instrumentation does not change or impact the original user interface of the app which may be considered as intellectual property. For closed-source platforms, we envision developers using the features offered in IoTWatcH to evaluate and improve the protection of sensitive information and the privacy of users. Lastly, we note that users can update their privacy profiles through the settings provided in instrumented IoT app. We follow a similar principle that a user can update the app settings such as when a new device is added. The updated privacy profiles then are sent to IoTWatcH's server and the new user notifications are automatically enabled.

Evading IoTWatcH Analysis. The use of encryption and obfuscation techniques to hide the sink-call content limits the effectiveness of NLP techniques. For instance, NLP models that use plain-text data would fail to classify encrypted/obfuscated strings. However, selective instrumentation may facilitate the analysis of IoT apps that codify or encrypt the data that is sent

```
1:  //Encryption to obfuscate sink-call content
2:        def crypto = new Crypter()
3:        def plainText = "Let's leak this message"
4:        def secret = "123456789"
5:        //Selective Instrumentation before encryption
6:        IoTWatch.collect(plaintext) ❶
7:        def encryptedText = crypto.encrypt(plainText, secret) ❷
8:        leakInfo(encryptedText)
9:  // Send notification to the user
10:       notifyUser(state, number){
11:             sendSMS("Your lock is: " + state, phone)
12:             sendPush("App is using data of type:"
13:                      classification)}
14: // Leak sensitive data to attacker
15:       leakInfo(encryptedText){
16:             sendSMS(encryptedText, 123-456-7890) ❸
17:             sendPush("Privacy Risk of type:" context
18:                      "with data of type:" classification)}
```

Fig. 5. Sample IoT app that encrypts the sensitive data to be leaked in an attempt to bypass the NLP analysis of IoTWatcH. The selective instrumentation capabilities of IoTWatcH permits the analysis of the data before it is encrypted.

out to hide their intent. We note that IoT programming platforms often provide developers with a limited number of white-listed libraries [46], which do not include a class that implements encryption. For instance, we analyzed 540 open-source SmartThings IoT apps and did not find a single case of an app using encryption or obfuscation to hide leaked content. It is also important to note that IoT platforms often do not vet obfuscated apps when developers submit them for approval to markets. While IoTWatcH is designed for apps that do not use any encryption and obfuscation techniques, it inserts a code block before encryption function to collect data. However, this requires prior knowledge of encryption functions. Figure 5 shows an example IoT app that encrypts a sensitive string before it transmits it out of an app. If the function `crypto.encrypt` is known to IoTWatcH, IoTWatcH's selective instrumentation extracts the call graph of the app and detects the `crypto.encrypt` function (❷) which uses `plainText` string. The result of the encryption is stored in a new taint variable `encryptedText`, which is later leaked via messaging (❸). To solve this challenge, the instrumentor is able to track the control flow of the encrypted variable and inserts extra code to collect its content before it is encrypted (❶).

5.2 IoTWatcH Analyzer

An instrumented code sends its data to IoTWatcH Analyzer, which is composed of three analysis steps including: (1) classification of sink-call content through NLP techniques, (2) analysis of sensitive data recipients, and (3) implementation of a user privacy notification interface to inform the user about IoTWatcH's findings. We next detail each.

5.2.1 Classification of Sink-Call Content

The content of messaging and Internet taint-sinks in IoT apps may include sensitive information from taint variables or might be just string messages that includes privacy sensitive information. IoTWatcH learns a supervised model using the contents of messaging and Internet taint-sinks as input at training and assigns it to privacy labels at inference. For instance, if the sink-call contains the string message "the door is unlocked", the model is able to classify it as "Device-info". The conversion of IoT strings into privacy labels using NLP helps users understand how IoT apps use sensitive information so that they can make informed privacy decisions about the IoT apps. Below, we begin by constructing a training set from a corpus of IoT apps.

Constructing Privacy Labels. IoTWatcH classifies IoT app sink-call contents into a set of privacy labels. The use of privacy labels provides three main advantages. First, they allow users to understand what type of data the apps leak. For instance, a string that contains "The mode has been changed to Home" can be presented to users as leaking their preferences. Second, the privacy labels permit users to have control over their privacy preferences at install time. For instance, a user may desire to allow an app to transmit energy usage of a thermostat to a remote server, yet she restricts other types of data transmissions. Third, the privacy labels enable an on-demand notification system that guarantees an improved user experience with less disruptive and more intuitive notifications, and minimal runtime delay added to each app's execution.

We define four privacy labels through the analysis of the semantics of strings extracted from messaging and Internet taint-sinks (although adding more privacy labels later is a straightforward task). The privacy labels are based on user feedback that we acquired via our privacy survey in Section 3. We present the details of four privacy labels to label taint sink content:

Date-time defines the time or date information. For instance, a string "The door is unlocked at 5:00 pm" contains the time that the door would be unlocked.

Device-info identifies the text that contains the devices states (e.g., a string "energy usage" to transmit the power state of a thermostat) and device information (i.e., device type, model, manufacturer).

Location identifies the data that reveal geo-location or geographical location of users and devices. For instance, a string "kitchen switch" contains geo-location information of kitchen, and the zip-code for sunset settings contains geographical location.

User-behavior defines the text that provides information about user preferences defined in apps. For instance,

App Sink-Call Content	Assigned Privacy Labels
"Thermostat is turned on"	device-info
"The door will remain open for another 5 minutes"	device-info, date-time
"Door is closed since car was not at Home, 15 sec ago"	device-info, location, date-time
"Sleep time set for you as requested"	user-behavior

Table 2. Examples of leaked taint-sink contents from IoT apps and their privacy labels. IoTWatcH assigns multiple privacy labels to specific IoT strings to capture their complex semantics.

an app includes a string, "the user mode changed to vacation from home".

We note that taint-sink contents may require more than one privacy label assigned to guarantee accurate classification. For instance, the message string "The door will remain open for another 5 minutes" is labeled with multiple labels of types Device-info and Date-time. Similarly, "Garage door is not opening since the car was not present at Home, less than 15 sec ago", is labeled as Device-info (the door is not opening so it is still closed), Location (the car was not present at Home), and Date-time (15 sec ago). We present examples of IoT taint-sink contents assigned to the various privacy labels in Table 2. The use of multiple privacy labels to a single text includes more complex semantics string structures and in turn provide users with more accurate labels. As we explain later in this section, we use NLP techniques to achieve this goal.

IoT Application Corpus. We study sink-call contents of 540 IoT apps. These contents pose a few unique characteristics compared to those of other domains. First, the size of the texts extracted is usually three to four shingles on average, yet it contains highly private data (e.g., "the door is unlocked"). Second, their linguistic structure is minimal regarding semantics (e.g., "mode changed to away"), compared to other short documents extracted from popular general-knowledge corpora [21, 33, 56]. Third, their meanings usually are closely attached to the app's context (e.g., "if the user is not-present, turn off the light"). Based on our findings, we build an NLP model that is effective at classifying semantic-deficient, but information-rich texts. To determine whether strings include sensitive information and assign them a privacy label, we first implemented a classification model using publicly available data corpora [21, 33, 56]; however, due to the specific characteristics of IoT strings, we obtained very low classification accuracy (detailed in Appendix C). To improve this, we constructed an *IoT-specific* corpus for classification purposes that successfully considers and solves these challenges. We first collected the content of messaging and Internet calls from current IoT market apps. We pre-processed the resulting dataset by filtering out punctuation and stop words. Further, we manually labeled the IoT strings to the four privacy labels. Here, we applied multi-labeling to contents that contained information related to more than one privacy label. Overall, the resulting dataset represents the semantics of taint-sink contents found in IoT apps.

NLP Model Construction. We implement an NLP model that uses a specific IoT corpus above for training. We use the model to classify unknown sink-call contents to privacy labels. To train the model, we propose a supervised learning approach that requires labeled data as input. We expect that the supervised approach yields better results than other approaches such as keyword-search. This is because the IoT string containing privacy information often does not include sufficient information to assign the labels through simple keyword-search-based analysis. For instance, keyword-search would fail to identify the user-behavior in a messaging taint-sink that leaks a string "the kids left home". The model integrates *doc2vec* [8] to represent every n-gram IoT string (i.e., IoT document) into a multi-dimensional vector. Specifically, we choose the Paragraph Vector-Distributed Memory (PV-DM) approach as strings extracted from IoT apps have limited semantic structures, yet they include a substantial amount of information. By using doc2vec helps IoTWatcH find the leaks that are context-wise and semantically similar. Thus, we select PV-DM doc2vec over other known approaches like bag-of-words (BoW) [30] as it has proven to be effective against incomplete or semantically-limited strings. PV-DM considers the entire structure of the text and infers what is missing to perform syntax and context analysis (i.e., multi-word expression analysis) [32]. In this way, we are able to extract the privacy content from semantically-limited IoT strings. We then performed topic classification of the IoT strings using an automated machine learning approach (i.e., autoML) [6]. We note that autoML selects the best classification algorithm by trying various classification algorithms; thus, the best accuracy is always guaranteed. We provide details of the NLP model and IoT corpus in Section 6.

5.3 User Notifications

IoTWatcH implements two different privacy notification options, as shown in Figure 3(a), Appendix C. First, it allows a user to select specific privacy labels (one or multiple) for an app to receive notifications. For instance, if the user is only concerned about the location information, she may select the location label so that IoTWatcH informs if a sink-call transmits data defined

through location-label. This means that the user would not receive any other notifications of other privacy labels. The second option allows the user to determine the type of taint-sink to be notified, messaging, and Internet, or both. The selective user-specific notifications provide flexibility, reduces the IoTWatcH latency in instrumented apps, and enhances the user experience by reducing the number of notifications at runtime. Lastly, IoTWatcH also check the recipients of Internet taint-sink and see whether it uses HTTP or HTTPS to transmit sensitive data. This exposes the content of the Internet message to passive observers (i.e., eavesdroppers). In this case, IoTWatcH is able to notify users as well, as shown in Figure 3(b), Appendix C.

We note that IoTWatcH enables all privacy labels and taint-sink options by default. Tech-savvy users that fully understand the privacy risks of IoT apps may create their own privacy profiles by disabling install-time options for better user experience. Users that do not entirely understand the privacy labels in IoTWatcH may rely on the default options.

6 Implementation Details

We implemented IoTWatcH for IoT applications developed for Samsung SmartThings, which is the IoT platform that has the highest share of devices and applications in the current commodity IoT market [47, 52]. Samsung SmartThings apps are developed in Groovy, a dynamic programming language that supports static compilation. Static compilation permits for all methods and classes in the apps to be annotated at compile time, which makes this information fully available to the instrumentation portion of IoTWatcH.

Code Instrumentor. IoTWatcH traverses the Abstract Syntax Tree (AST) of the IoT app's Intermediate Representation (IR) using the Groovy's `ASTTransformation` class [23]. This class, included in the Apache Groovy project, offers several metaprogramming utility methods to build and analyze the app's AST. We used the AST to perform our analysis as we targeted open-source applications, and we could take advantage of the specific programming classes that Groovy offers for the compile-time analysis. Specifically, we use the class `ASTNode` to build an app's Intraprocedural Control Flow Graph (ICFG) [10]. From there, we used the `visit` methods to inspect the different ICFG nodes and flag methods that implement sink functions (e.g., sendSMS), handle sensitive privacy data (e.g., phone number), subscribe to events, or define event handlers. IoTWatcH involves around 1700 lines of code written in Groovy to analyze the app source code, construct the IR, generate the ICFG, and perform the code instrumentation. We implemented IoTWatcH's instrumentor as a web application using Groovy programming language (detailed in Appendix B). We made the instrumentor available online at https://iotwatch.appspot.com/.

App and IoT Data Collection Process. We collected 540 Samsung SmartThings apps to implement and evaluate IoTWatcH, which represented 100% of the open-source SmartThings apps available at the time of developing this project. These IoT apps are from six different general categories: Convenience, Smart Home Automation, Entertainment, Personal Care, Security & Safety, and Smart Transportation. Out of the 540 apps, 380 SmartThings apps were used to build an IoT corpus and train the NLP model, and 160 apps to evaluate its performance (detailed in Section 7). For comprehensiveness, we included in our analysis SmartThings market apps crawled from official Samsung repositories [48, 50] and malicious apps crawled from the IoTBench repository [28], an IoT-specific test corpus used to evaluate systems for IoT app security and privacy. The IoTBench includes flawed apps that perform various malicious activities, including sensitive data leaks via both messaging and Internet taint-sinks.

To collect the IoT strings from the crawled apps, we analyzed and instrumented the total population of apps using the IoTWatcH's code instrumenter. IoTWatcH's instrumentor adds on average 25% more lines of code (LoC) to the apps, which translates into adding 65 LoC to an IoT app that has an average size of 265 LoC. We then executed the instrumented apps in the SmartThings Simulator IDE [49], a propriety simulation tool provided by Samsung. The IDE permits the modeling of trigger-action scenarios in which an instrumented app sends messaging- and Internet-based taint sinks that contain the IoT strings to the IoTWatcH server. The IoT strings are individually labeled through app and string ID, and taint sink in the form {ID: app_ID, taint_sink, string}.

Data Split and Model Training. We first organized the strings collected from the 540 IoT apps into the six different app categories. Then, for proper data balance, we randomly selected the strings collected from 70% of the apps from each category until reaching 380 training apps. The remaining strings from the other 160 apps were used later for evaluating IoTWatcH. From the selected apps for training, we extracted a total of 2014 different IoT strings. We then labeled these strings according to the four privacy labels. Specifically, 46.8%

of the strings contained information related to the IoT devices, while 20.8% contained relevant information related to Date-time. The remaining 19.2% and 13.2% of the IoT strings shared information related to location and user-behavior, respectively. We allowed up to 75% of inter-labeling assignment to the strings, meaning, up to three different privacy labels can be assigned to a single string. In total, we applied multi-labeling to 72% of the privacy strings in the IoT corpus. Finally, we applied k-fold cross validation on the labeled data to train and validate the NLP model. We randomly selected 75% of the total corpus to train the classifier for approximately 15 hours. Then, we validated the obtained model with the remaining 25% of the data, which is a practice followed by modern autoML [7]. Initial testing results on the NLP model showed an average precision of 94.3% and recall of 89.6%.

Classification of IoT Privacy Strings. We use Automatic Machine Learning (Auto-ML) tools offered by Google App Engine [6] to perform privacy classification of IoT strings. Among its benefits, modern auto-ML approaches perform neural architecture search and transfer learning, enable hyper-parameter optimization, and utilize advanced model architectures to classify contents to privacy labels with high accuracy. More particularly, we implemented a custom multi-class multi-label model using the Natural Language API offered by Google (Google-NL) [7]. Google-NL offers a suite of ML algorithms that automatically optimize the algorithm parameters based on the specific algorithm utilized and the characteristics of the dataset to guarantee the highest accuracy. Our initial analysis of 2014 IoT strings showed remarkable semantic similarities among them; thus, the use of labeled data reduces the training time considerably. Finally, we implemented our NLP solution to classify strings written in English as only two (0.5%) IoT strings analyzed were written in Spanish.

IoTWatcH API. We implemented a REST API to enable effective and secure data exchange between the instrumented IoT app and IoTWatcH's analyzer (Figure 3). From the app to the server, the API constructed a JSON object with the user-defined recipients and the taint-sink information. From the server to the app, another JSON object was used to send the IoTWatcH's analysis results. The API also handles IoTWatcH's privacy notifications to the user. Once the privacy analysis is completed, IoTWatcH sends back to the user another JSON object containing its findings. (aditional details in Appendix C). Further, we protected the communications between the apps and the IoTWatcH's server with the *asynchttpv1* class of Samsung SmartThings [51], which allows for asynchronous and encrypted `HTTPS` calls. Also, we guaranteed integrity of the data being exchanged by digitally signing the API requests with SmartThings x.509 certificates. Specifically, the API requests fetch the public keys for signature verification from `https://key.smartthings.com + <aKeyId>`. Then, the IoTWatcH computed the validity of the signatures using the resolved key combined with the HTTP headers provided on the API callback request. Finally, we assumed the IoTWatcH servers were secure so the actual NLP analysis as well as the user notifications were not compromised by external attackers. Note that attacks to the server or server-related threats were not considered in this work.

7 Performance Evaluation

We evaluated the performance and efficacy of IoTWatcH based on the three research questions (RQ).

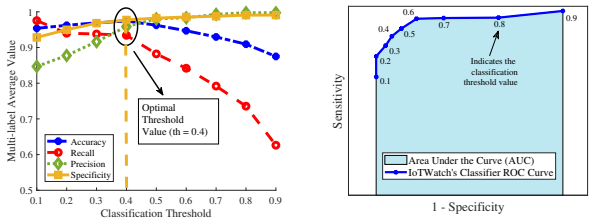
RQ1 What is the effectiveness of IoTWatcH in correctly classifying the tain-sink contents into privacy labels? (Section 7.1).

RQ2 What is the effectiveness of IoTWatcH in identifying data leaks that confirms privacy preferences of users? (Section 7.2).

RQ3 What is the runtime overhead of IoTWatcH in terms of latency and storage? (Section 7.3).

Evaluation Setup. We analyzed and extracted IoT strings from a total of 160 Samsung SmartThings apps to evaluate IoTWatcH's performance. Specifically, we included 120 market and 40 malicious apps in the evaluation. The apps were instrumented and executed in the SmartThings IDE to extract the app communication content as explained in Section 6. Finally, the apps sent their data to IoTWatcH's analyzer at execution which ran on a Python web server hosted on Google App Engine [20].

Defining User Privacy Preferences. The ideal case to define user preferences requires executing IoT apps with the real users, where each of them installs an app. However, this is time-consuming as we have hundreds of apps, and each user needs to understand the functionality of each app before they define their preferences. To address this, we assume users set their preferences when the IoT data is transmitted out of an app without their consent, which allows us to define privacy preferences of users in our experiments as a ground truth. A user is able to give consent to IoT app recipients and contents in two ways. First, a user defines the taint-sink



(a) Classification Metrics (b) ROC, AUC

Fig. 6. IoTWatch classifier performance: (a) average value of all performance metrics for the different threshold values and (b) ROC curve and AUC that illustrate the performance of IoTWatch's classifier for all privacy labels.

recipient at app install-time, for instance, a cell-phone number to receive a notification or a web page name that the content is sent. We are able to automate this process through sink-call recipient analysis. IoTWatch matches the taint-sink recipients with information defined by the user at install-time. For instance, the user defines an authorized recipient in the field *phone*. IoTWatch extracts this information and checks it against the taint-sink recipient. If they match, it means that the user defines the recipient at install-time. This is because IoTWatch recognizes that the user previously authorized this recipient, and the data transmitted to the recipient do not violate the user's privacy preference.

Defining Leaks. In cases where IoTWatch sees discrepancies between the recipient of a taint-sink and the one that was defined by the user, it identifies it as a *data leak*. Also, we considered a data leak any time an app sends information to a third party without informing the app user via the app description. As we consider that the description block of an app should explicitly specify the content transmitted out of an app and its recipients, failing to do so makes it impossible for the user to acknolewdge and confirm the app's intent at install-time. Thus, for instance, an Internet taint-sink that sends sensitive data to a server that is not informed to the user via an app description block is clearly a leak.

7.1 Taint-sink Content Classification

IoTWatch's analyzer classifies IoT app taint sink content (i.e., IoT strings) into four privacy labels. The classification results include the assigned label and the confidence scores through a threshold *th*. If the classification score is over the predefined threshold, the privacy label is assigned to the string. For instance, the string "the front door at home is unlocked" resulted in classification scores of device-info=0.94, user-behavior=0.03, location=0.86, and date-time=0.3. By design, the classification scores are independent of each other, that is,

Label	TP	TN	FP	FN	Accuracy	Recall	Precision	Specificity
Device-info	958	232	11	84	0.9260	0.9214	0.9890	0.9547
Date-time	99	1161	18	36	0.9589	0.7333	0.8918	0.9847
User-behavior	508	684	26	95	0.9079	0.8425	0.9585	0.9633
Location	221	1061	10	22	0.9756	0.9095	0.9610	0.9907

Table 3. Performance of IoTWatch in classifying IoT strings to all the different privacy labels.

if *th* value is set to 0.7, IoTWatch classifies the string into privacy information of type device-info and location. In total, IoTWatch classified 146 IoT strings extracted from 95 different IoT apps. Out of these, 112 strings were extracted from messaging taint-sinks; 54 messages from 44 market IoT apps and 58 messages from 30 malicious apps. The remaining 34 strings were extracted from Internet taint-sink; 12 Internet calls extracted from 10 market IoT apps, and 22 extracted from 11 malicious IoT apps.

Performance by Threshold Values. We study how IoTWatch's classifier performs for different threshold values *th*. The goal of this analysis is to find the value *th* that leads to the highest performance overall. Figure 6(a) summarizes the average metrics for different values of *th*. Overall, IoTWatch yields the best accuracy for threshold values between 0.1 to 0.5. We observe that $th = 0.4$ yields the best classification results for all metrics. Finally, IoTWatch classifies IoT strings to correct privacy labels with 94.25%, 85.17%, 95.01%, and 97.34% average accuracy, recall, precision, and specificity, respectively (additional details in Appendix D).

Performance by Privacy Label. We further study the sensitivity of IoTWatch's classifier to each privacy label. The goal is to determine the effectiveness of IoTWatch in classifying strings to the different privacy labels. Table 3 shows the performance of IoTWatch in classifying IoT strings to different privacy labels. IoTWatch achieves the highest accuracy for privacy labels of type location and date-time. This is because date, time, and location can be easily inferred from semantically-simple strings. Also, it is very common to find information related to these privacy labels embedded in the same string (e.g., "he arrived home 5 minutes ago"). In contrast, IoTWatch obtained the lower accuracy results for privacy labels of type user-behavior. This is because user-behavior information is harder to infer from simple strings. In spite of these results, IoTWatch achieved the lowest accuracy of 90.79% for user-behavior labels, which is comparable with the best classification results of other similar tools in the market [37]. In summary, our observation is that the classification errors of IoT strings into privacy la-

bels are due to the highly-limited structure of the IoT strings. In most cases, developers try to exfiltrate more information using short messages lacking proper semantics (e.g., "front door was unlocked 5 min ago"). In this case, we note that a straightforward classification approach does not yield successful results when extracting complete privacy information related to the device (e.g., door unlocked), date-time (e.g., 5 min ago), and location (e.g., front of the house) and, hence, would not be instrumental in uncovering all the privacy risks. Finally, we present the Receiver Operating Characteristic (ROC) curve and the corresponding Area Under the Curve (AUC) in Figure 6(b). These metrics summarize the overall performance of the IoTWatcH's classifier for all privacy labels. The AUC fully supports the classification metrics detailed in Figure 6(a) and reflects the high prediction quality of the proposed NLP classifier.

Our observation is that the classification errors are due to the highly-limited structure of the IoT strings. In most cases, developers try to exfiltrate more information using short messages lacking proper semantics (e.g., "front door was unlocked 5 min ago"). In this case, we note that a straightforward classification approach does not yield successful results when extracting complete privacy information related to the device (e.g., door unlocked), date-time (e.g., 5 min ago), and location (e.g., front of the house) and, hence, would not be instrumental in uncovering all the privacy risks. To overcome these issues and improve the classifier's performance in IoTWatch, we followed different and more fruitful design strategies, as stated in Section 5.2.1. First, we used a rich IoT corpus to train the model. Second, we represented every IoT string with a multi-dimensional vector via paragraph vector-distributed memory-based (PV-DM) doc2vec. With this approach, the IoTWatch classifier could predict the missing words from the structurally-limited strings and improve the classification results significantly. We will clarify this in the final version of the paper.

Findings on Privacy Analysis of IoT Strings. IoTWatcH classifies IoT strings to privacy labels with an average accuracy of 94.25%. Out of 160 apps, IoTWatcH identified 50 (31.25%) apps that transmit data related to device information via messaging, and 20 (12.5%) apps that do the same via Internet taint-sinks. Also, 11 (6.9%) apps handled data related to date and time in messages and only one transmit similar information using the Internet. IoTWatcH also identified 38 (23.75%) apps transmitting information related to the user behavior in their messages and nine (5.6%) including similar type of information in Internet taint-sinks. Further, 20 (12.5%) apps sent information related to location via messaging, while six (3.7%) apps did the same via Internet calls. Finally, we evaluated how the privacy analysis of IoT strings benefited from the use of NLP. IoTWatcH assigned multiple privacy labels to classify more semantically-complex IoT strings, which guaranteed completeness in the privacy analysis. Out of 146 strings analyzed, IoTWatcH applied multi-labeling to 68 (46.5%). Specifically, 54 IoT strings (36.9%) received two privacy labels and 14 (9.6%) strings received three privacy labels.

App Type	No. of Apps	Mess.	Leaks	Cl-T.	Effec.
Market	120	54	0	–	–
Malicious	40	58	29	–	100%
Total	160	112	29	–	100%

Mess. - Messaging taint-sinks Analyzed
Leaks - No. of Data Leaks Found
Cl-T. - No. of Clear-text Leaks Found
Effec. - IoTWatcH Effectiveness

Table 4. Effectiveness of IoTWatcH in detecting sensitive data leaks via messaging.

App Type	No. of Apps	Int.	Leaks	Cl-T.	Effec.
Market	120	12	11	3	100%
Malicious	40	22	22	3	100%
Total	160	34	33	6	100%

Int. - Internet taint-sinks Analyzed

Table 5. Effectiveness of IoTWatcH in detecting sensitive data leaks via Internet taint-sinks.

7.2 Data Leakage Analysis

Findings on Data Leaks via Messaging. Table 4 shows IoTWatcH's findings after analyzing messaging recipients. IoTWatcH extracted 54 recipients in messages from market apps and 58 from malicious apps. We found no data leaks via messaging in market apps, meaning, all recipients were defined (or authorized) by the user at install-time. We believe this is due the strict review process enforced by SmartThings IoT market. Further, we found 29 leaks from 14 different malicious apps [28], meaning, all these recipients were hard-coded by a developer and their intent were not defined in the app's description block. For instance, the User Event app [28] leaks privacy labels of type device-info, location, and user-behavior ("Everyone is away and hub ID is #") to a hard-coded phone number. We manually reviewed the app source codes and verified that all IoTWatcH's findings were correct. Also, we verified that all data leaks via messaging were properly flagged.

Findings on Data Leaks via Internet. Table 5 details the effectiveness of IoTWatcH's analyzer in find-

ing data leaks via Internet taint-sinks. IoTWatcH analyzed 34 recipients from Internet taint-sinks, 12 from market and 22 from malicious apps. Our tool flagged 11 Internet taint-sinks from seven market apps that leak privacy data without user consent. That is, the user is neither informed about the recipient of the data in the app description block nor enters the URL or domain name herself. For instance, the ThingSpeak Logger app [48] transmits a device ID to a remote server through an HTTP call, which is identified through a device-info privacy label. For malicious apps, IoTWatcH flagged 22 Internet taint-sinks from 14 different malicious apps as leaks. We verified that 100% of data leaks via Internet were properly flagged.

Clear-Text Data Leaks. IoTWatcH is also able to verify whether an IoT app sends a clear-text through HTTP. IoTWatcH flagged three clear-text leaks from three different market apps out of twelve Internet taint sinks. We found that one of these calls was authorized by the user at install-time through the app description block. Lastly, IoTWatcH flagged three Internet taint-sinks that uses HTTP to leak sensitive data from malicious apps.

7.3 Overhead Analysis

We show the performance of IoTWatcH in terms of runtime and storage overhead.

Runtime Overhead. Latency refers to the time elapsed from the moment that the app's data is collected to the moment that the user receives IoTWatcH's notifications. Latency overhead is calculated as the average difference in the execution time of the original and instrumented apps. On the one hand, IoTWatcH required on average 75 ms to classify the sink-call contents. On the other hand, the communication latency between the IoT app and the IoTWatcH's server was 30 ms on average. We used the Groovy class *asynchttp_v*1 to implement asynchronous HTTPS requests to reduce communication latency. We found that the total latency introduced by IoTWatcH is on average 105 ms.

Storage Overhead. We measured the storage overhead imposed by IoTWatcH. Our tool does not store app information after the analysis is completed; thus, the storage cost is determined by the total storage size of the JSON object used to exchange information between the IoT apps and IoTWatcH's analyzer (Section 6). We evaluated the storage overhead imposed by the analysis of 160 IoT apps. On average, IoTWatcH imposes a negligible 1 KB of storage overhead.

7.4 Discussion and Future Work

IoTWatcH is the first dynamic tool that performs NLP-based real-time privacy analysis in IoT apps to (1) classify IoT strings into privacy labels that are easy to understand by the user, and to (2) flag IoT apps that represent privacy concerns for the user. We implemented IoTWatcH for SmartThings IoT platform, and we plan to extend our analysis to other IoT platforms. Additionally, we analyzed 380 IoT apps and constructed a dataset to study how these apps use privacy-sensitive information. While our corpus included IoT strings extracted from SmartThings market apps, we plan to investigate other IoT platforms to construct similar datasets.

We designed and built IoTWatcH by first understanding the privacy needs of IoT users. We plan to conduct an additional study to evaluate the usability of IoTWatcH which is outside the scope of the current work. Also, IoTWatcH's analysis would benefit from mapping the app descriptions to privacy labels. However, this is challenging as the description block of an IoT app does not explicitly state the app's privacy behavior but its functionality. We plan to use more advanced NLP techniques to address this challenge. Finally, IoTWatcH's execution requires the collection and analysis of privacy-sensitive information. We use secure HTTPS communications to protect the communication between IoT apps and IoTWatcH's server. In addition, IoTWatcH does not keep record of any collected information nor share this information with any third party. As a future work, a complete privacy assessment of IoTWatcH may be conducted to guarantee that user's privacy is completely preserved.

8 Related Work

Data Flow Analysis. Previous works have focused on flow data analysis to research the security and privacy of the mobile phone [5, 14, 17, 24, 26, 57] and the IoT apps [9, 10, 18, 40]. Tools for mobile apps cannot be directly applied to the IoT as applications from both ecosystems pose different architectural challenges. For instance, FlowCog [37] establishes data flow dependencies based on Android app view context, which cannot be extracted from IoT apps due to specific architectural differences. On the other hand, previous solutions for IoT mostly consider security risks from data flows with tainted variables or via inter-rule or cross-app vulnerabilities. For instance, SainT [10] does not consider data leaks at runtime or via IoT strings, while FlowFence [19] often over-approximates the data leaks, which leads to

Tool Name	Domain	Code Analysis	Dynamic Analysis	Semantics Analysis	Privacy Analysis	NLP/ML Analysis	User Awareness	Overhead Evaluation	Freely Available
TaintDroid [17]	Android	●	●	○	●	○	●	●	●
FlowDroid [5]	Android	●	○	○	●	○	○	○	●
FlowCog [37]	Android	●	●	●	○	●	●	○	●
FlowFence [19]	IoT	●	●	○	○	○	○	●	●
ContextIoT [29]	IoT	●	●	○	○	○	●	●	○
Saint [10]	IoT	●	○	○	○	○	●	○	●
ProvThings [55]	IoT	●	●	○	○	○	○	●	○
iRuler [54]	IoT	●	○	●	○	●	○	○	○
SmartAuth [53]	IoT	●	○	●	○	●	●	○	○
IoTWatch	IoT	●	●	●	●	●	●	●	●

Table 6. Comparison between IoTWatch and other data flow analysis tools for Android and IoT apps.

failure when dependency between taint variables and leaks cannot be established. The authors in [18] proposed the use of static analysis to reveal permission-based security flaws. Also, the authors in [40] enforced data flow control by applying opacified computation. Lastly, ProvThings [55] uses static and dynamic analysis to collect data provenance and identify the root cause of attacks in IoT apps. However, this work is limited to analyze dependencies between events and data states and does not offer any built-in privacy analysis. iRuler [54] applies NLP techniques to uncover inter-rule vulnerabilities parsed not from IoT apps, but IoT services. The SmartAuth tool [53] also uses NLP, this time to generate and enforce security policies in IoT apps. Similarly, the authors in [27] analyze privacy risks in IoT online recipes. Table 6 compares IoTWatch with other the existing data flow analysis tools.

IoT Privacy. Information exposure in IoT rises concerns among users and researchers. The authors in [58] perform a qualitative analysis of the type of sensitive information IoT devices expose. Also, the authors in [59] study IoT privacy focusing on the user's perception. A novel work proposes the use of blockchain [60] or attribute-based encryption [63] to provide enhanced security and privacy to smart home systems. Meanwhile, other authors propose the use of unified systems as the solution to provide enhanced privacy in IoT [61]. Finally, some works focus on protecting the privacy of IoT communications via novel protocols [64].

Comparison to Existing Works. Different from SaINT [10] and ProvThings [55], IoTWatch uncovers privacy risks of IoT apps not only from taint data but also from simple IoT strings. Also, IoTWatch analysis focuses on privacy concerns from IoT apps, something that is missing in other analysis tool for IoT like iRuler [54], SmartAuth [53], and ContextIoT [29]. IoTWatch uses user feedback to performs qualitative analysis on the information handled by IoT applications. Such a capability is missing in all other tools included in Table 6. Specifically, IoTWatch collects app data in real time and converts the data into specific privacy labels that the user can understand and customize. Thus, IoTWatch not only studies the cause and flow of the sensitive information, but its meaning to the user and the systems. There are some other features that are unique to IoTWatch. Specifically, IoTWatch represents the first security tool that exposes the recipients of the sensitive information in real time. Finally, since the proposed privacy tool is user-centered, it guides its analysis with customizables privacy profiles that the users create at install time and change anytime after. That way, IoTWatch minimizes the numbers of notifications and focuses its analysis on privacy features that are of interest of every user individually.

9 Conclusion

IoT apps access sensitive data that, if leaked, may compromise the privacy of the users. IoT platforms do not offer real-time privacy analysis that informs users about how the IoT apps handle sensitive information. To adress these concerns, in this paper, we introduced IoTWatch, a dynamic analysis tool that uncovers the privacy risks of IoT apps in real-time. We developed IoTWatch based on a study of the privacy needs of 123 users. IoTWatch enables users to select their privacy preferences, and uses NLP techniques to classify IoT strings into user-friendly privacy labels. This allows users to make informed decisions about their privacy and reject apps. We analyzed 540 IoT apps to train the NLP model and evaluate its effectiveness. IoTWatch classifies IoT strings to correct privacy labels with an average accuracy of 94.25% and flags 35 apps that leak sensitive data. Finally, IoTWatch yields a minimal overhead to an IoT app's execution, on average 105 ms additional latency.

10 Acknowledgments

This material is based upon work partially supported by the U.S. National Science Foundation under award numbers NSF-CAREER-CNS-1453647 and NSF-1663051.

References

[1] Acar, A., Fereidooni, H., Abera, T., Sikder, A. K., Miettinen, M., Aksu, H., Conti, M., Sadeghi, A.-R., and Uluagac, A. S. Peek-a-boo: I see your smart home activities, even encrypted! *WiSec* (2020).

[2] Aksu, H., Babun, L., Conti, M., Tolomei, G., and Uluagac, A. S. Advertising in the iot era: Vision and challenges. *IEEE Communications Magazine 56*, 11 (November 2018), 138–144.

[3] Z. B. Celik and P. McDaniel and G. Tan and L. Babun and A. S. Uluagac Verifying Internet of Things Safety and Security in Physical Spaces. *IEEE Security Privacy*, 17 (September 2019), 30–37.

[4] Apple's Home Kit Security and Privacy on iOS. https://www.apple.com/business/docs/iOS_Security_Guide.pdf. [Online; accessed 9-January-2020].

[5] Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., and McDaniel, P. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. *ACM SIGPLAN Notices* (2014).

[6] AutoML. https://www.ml4aad.org/automl/. [Online; accessed 10-February-2019].

[7] AutoML Natural Language Google. https://cloud.google.com/natural-language/automl/docs/. google[Online; accessed 10-February-2019].

[8] Ayyadevara, V. K. *Word2vec*. Apress, Berkeley, CA, 2018, pp. 167–178.

[9] Babun, L., Sikder, A. K., Acar, A., and Uluagac, A. S. Iotdots: A digital forensics framework for smart environments, 2018.

[10] Celik, Z. B., Babun, L., Sikder, A. K., Aksu, H., Tan, G., McDaniel, P., and Uluagac, A. S. Sensitive Information Tracking in Commodity IoT. In *27th USENIX Security Symposium* (2018).

[11] Celik, Z. B., Fernandes, E., Pauley, E., Tan, G., and McDaniel, P. Program Analysis of Commodity IoT Applications for Security and Privacy: Challenges and Opportunities. *ACM Computing Surveys (CSUR)* (2019).

[12] Celik, Z. B., McDaniel, P., and Tan, G. Soteria: Automated IoT safety and security analysis. In *USENIX Annual Technical Conference (USENIX ATC)* (2018).

[13] Celik, Z. B., Tan, G., and McDaniel, P. IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT. In *Network and Distributed System Security Symposium (NDSS)* (2019).

[14] Clause, J., et al. Dytan: a Generic Dynamic Taint Analysis Framework. In *ACM Software Testing and Analysis* (2007).

[15] Denney, K., Babun, L. and Uluagac, A. S. USB-Watch: a Generalized Hardware-Assisted Insider Threat Detection Framework. In *in Journal of Hardware and Systems Security* (2020).

[16] Denney K., Erdin E., Babun L., Vai M., and Uluagac A. S. USB-Watch: A Dynamic Hardware-Assisted USB Threat Detection Framework. In *Security and Privacy in Communication Networks (SecureComm).* (2019).

[17] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Transaction on Computer Systems* (2014).

[18] Fernandes, E., Jung, J., and Prakash, A. Security Analysis of Emerging Smart Home Applications. In *IEEE Security and Privacy (SP)* (2016).

[19] Fernandes, E., Paupore, J., Rahmati, A., Simionato, D., Conti, M., and Prakash, A. FlowFence: Practical Data Protection for Emerging IoT Application Frameworks. In *USENIX Security* (2016).

[20] Google App Engine. https://cloud.google.com/appengine/. [Online; accessed 26-August-2020].

[21] Google Books NGrams. https://aws.amazon.com/datasets/google-books-ngrams/. [Online; accessed 10-August-2020].

[22] Google Forms. https://www.google.com/forms/about/. [Online; accessed 26-August-2020].

[23] The Apache Groovy. https://groovy-lang.org/metaprogramming.html#_abstractasttransformation. [Online; accessed 26-August-2020].

[24] Gordon, M. I., Kim, D., Perkins, J. H., Gilham, L., Nguyen, N., and Rinard, M. C. Information Flow Analysis of Android Applications in DroidSafe. In *NDSS* (2015).

[25] Gorla, A., Tavecchia, I., Gross, F., and Zeller, A. Checking App Behavior Against App Descriptions. In *Proceedings of the 36th International Conference on Software Engineering* (2014), ICSE 2014, ACM.

[26] Gu, B., Li, X., Li, G., Champion, A. C., Chen, Z., Qin, F., and Xuan, D. D2Taint: Differentiated and Dynamic Information Flow Tracking on Smartphones for Numerous Data Sources. In *INFOCOM* (2013).

[27] IFTTT (if this, then that). https://ifttt.com/, 2017. [Online; accessed 26-August-2020].

[28] IoTBench. https://github.com/IoTBench, 2017. [Online; accessed 26-August-2020].

[29] Jia, Y. J., Chen, Q. A., Wang, S., Rahmati, A., Fernandes, E., Mao, Z. M., Prakash, A., and Unviersity, S. J. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *NDSS* (2017).

[30] Le, Q., and Mikolov, T. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32* (2014), ICML'14.

[31] Metaprogramming. http://docs.groovy-lang.org/docs/next/html/documentation/core-metaprogramming.html. [Online; accessed 26-August-2020].

[32] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. Efficient estimation of word representations in vector space. *CoRR abs/1301.3781* (2013).

[33] N. Iderhoff, "nlp-datasets". https://github.com/niderhoff/nlp-datasets/blob/master/README.md. [Online; accessed 26-August-2020].

[34] OpenHAB IoT App Market (Eclipse Market Place). http://docs.openhab.org/eclipseiotmarket. [Online; accessed 26-August-2020].

[35] OpenHAB IoT App Submission Guideline. https://marketplace.eclipse.org/content/eclipse-marketplace-publishing-guidelines. [Online; accessed 26-August-2020].

[36] OpenHAB: Open Source Automation Software for

Home. https://www.openhab.org/. [Online; accessed 26-August-2020].

[37] Pan, X., Cao, Y., Du, X., He, B., Fang, G., Shao, R., and Chen, Y. FlowCog: Context-aware Semantics Extraction and Analysis of Information Flow Leaks in Android Apps. In *27th USENIX Security Symposium* (Baltimore, MD, 2018).

[38] Pandita, R., Xiao, X., Yang, W., Enck, W., and Xie, T. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *Presented as part of the 22nd USENIX Security Symposium* (Washington, D.C., 2013), USENIX.

[39] Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., and Chen, Z. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. In *Proceedings of the ACM Conference on Computer and Communications Security* (New York, NY, USA, 2014), CCS '14, ACM.

[40] Rahmati, A., Fernandes, E., and Prakash, A. Applying the Opacified Computation Model to Enforce Information Flow Policies in IoT Applications. In *IEEE Cybersecurity Development (SecDev)* (2016).

[41] Sikder, A. K., Aksu, H., and Uluagac, A. S. 6thsense: A context-aware sensor-based attack detector for smart devices. In *26th {USENIX} Security Symposium ({USENIX} Security 17)* (2017), pp. 397–414.

[42] Sikder, A. K., Aksu, H., and Uluagac, A. S. A context-aware framework for detecting sensor-based threats on smart devices. *IEEE Transactions on Mobile Computing* (2019).

[43] Sikder, A. K., Babun, L., Aksu, H., and Uluagac, A. S. Aegis: A context-aware security framework for smart home systems. *ACSAC* (2019).

[44] Sikder, A. K., Petracca, G., Aksu, H., Jaeger, T., and Uluagac, A. S. A survey on sensor-based threats to internet-of-things (iot) devices and applications. *arXiv preprint arXiv:1802.02041* (2018).

[45] Pardis E. and Yuvraj A. and Lorrie F. C. and Hanan H. Ask the Experts: What Should Be on an IoT Privacy and Security Label? *arXiv preprint arXiv:2002.04631* (2020).

[46] SmartThings Classic Documentation: Classes and JARs. https://docs.smartthings.com/en/latest/getting-started/groovy-for-smartthings.html#allowed-classes. [Online; accessed 26-August-2020].

[47] SmartThings Code Review Guidelines and Best Practices. http://docs.smartthings.com/en/latest/code-review-guidelines.html. [Online; accessed 26-August-2020].

[48] SmartThings Community Forum for Third-party Apps. https://community.smartthings.com/. [Online; accessed 26-August-2020].

[49] SmartThings Groovy IDE. https://graph.api.smartthings.com/. [Online; accessed 26-August-2020].

[50] SmartThings Official App Repository. https://github.com/SmartThingsCommunity. [Online; accessed 26-August-2020].

[51] SmartThings Official Developer Documentation. http://docs.smartthings.com. [Online; accessed 26-August-2020].

[52] SmartThings Supported IoT Products (Devices). https://www.smartthings.com/products. [Online; accessed 26-August-2020].

[53] Tian, Y., Zhang, N., Lin, Y.-H., Wang, X., Ur, B., Guo, X., and Tague, P. SmartAuth: User-Centered Authorization for the Internet of Things. In *26th USENIX Security Symposium* (Vancouver, BC, 2017).

[54] Wang, Q., Datta, P., Yang, W., Liu, S., Bates, A., and Gunter, C. A. Charting the Atack Surface of Trigger-Action IoT Platforms. In *Proceedings of 26th ACM Conference on Computer and Communications Security* (2019).

[55] Wang, Q., Hassan, W. U., Bates, A. J., and Gunter, C. Fear and logging in the internet of things. In *Network and Distributed Systems Symposium (NDSS)* (Feb 2018).

[56] Wikipedia. https://dumps.wikimedia.org/wikidatawiki/entities/. [Online; accessed 26-August-2020].

[57] Zhu, D. Y., Jung, J., Song, D., Kohno, T., and Wetherall, D. TaintEraser: Protecting Sensitive Data Leaks Using Application-level Taint Tracking. *SIGOPS Operating Systems Review* (2011).

[58] Ren, J. and Dubois, D. J. and Choffnes, D. and Mandalari, A. M. and Kolcun, R. and Haddadi, H. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. *Proc. of the Internet Measurement Conf.* (2019).

[59] A. Dorri and S. S. Kanhere and R. Jurdak and P. Gauravaram User Perceptions of Smart Home IoT Privacy. *Proc. ACM Hum.-Comput. Interact.* (2018).

[60] Zheng, S. and Apthorpe, N. and Chetty, M. and Feamster, N. Blockchain for IoT Security and Privacy: The case study of a smart home. *IEEE PerCom Workshops* (2017).

[61] R. Chow The Last Mile for IoT Privacy. *IEEE Security Privacy* (2017).

[62] Google AutoML Natural Language Google Training. https://cloud.google.com/natural-language/automl/docs/prepare. Online; accessed 26-August-2020.

[63] T. Song and R. Li and B. Mei and J. Yu and X. Xing and X. Cheng A Privacy Preserving Communication Protocol for IoT Applications in Smart Homes. *IEEE Internet of Things Journal* (2017).

[64] X. Wang and J. Zhang and E. M. Schooler and M. Ion Performance evaluation of Attribute-Based Encryption: Toward data privacy in the IoT. *2014 IEEE International Conference on Communications (ICC)* (2014).

[65] OpenHAB Community, Openhab documentation, http://docs.openhab.org/index.html (2017). [Online; accessed 26-August-2020].

[66] Apple, Apple homekit documentation, https://developer.apple.com/homekit/ (2017). [Online; accessed 26-August-2020].

[67] Microsoft, Windows IoT core documentation, https://developer.microsoft.com/en-us/windows/IoT/explore/IoTcore (2017). [Online; accessed 26-August-2020].

[68] AKM I. Newaz and A. K. Sikder and A. M. Rahman and A. S. Uluagac Healthguard: A Machine Learning-based Security Framework for Smart Healthcare Systems. *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*.

[69] AKM I. Newaz and A. K. Sikder and A. M. Rahman and A. S. Uluagac A Survey on Security and Privacy Issues in Modern Healthcare Systems: Attacks and Defenses.

arXiv preprint arXiv:2005.07359.

[70] AKM I. Newaz and A. K. Sikder and L. Babun and A. S. Uluagac HEKA: A Novel Intrusion Detection System for Attacks to Personal Medical Devices. 2020 IEEE Conference on Communications and Network Security (CNS).

[71] A. K. Sikder and L. Babun and Z. B. Celik and A. Acar and H. Aksu and P. McDaniel and E. Kirda and A. S. Uluagac Kratos: Multi-User Multi-Device-Aware Access Control System for the Smart Home. 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, 2020.

[72] L. Babun and H. Aksu and L. Ryan and K. Akkaya and E. S. Bentley and A. S. Uluagac Z-IoT: Passive Device-class Fingerprinting of ZigBee and Z-Wave IoT Devices. 2020 IEEE International Conference on Communications (ICC)

[73] J. Myers and L. Babun and E. Yao and S. Helble and P. Allen MAD-IoT: Memory Anomaly Detection for the Internet of Things. 2019 IEEE Globecom Workshops (GC Wkshps)

[74] L. Rondon and L. Babun and K. Akkaya and A. S Uluagac HDMI-Walk: Attacking HDMI Distribution Networks via Consumer Electronic Control Protocol. ACSAC 2019

[75] L. Rondon and L. Babun and K. Akkaya and A. S Uluagac HDMI-Watch: Smart Intrusion Detection System Against HDMI Attacks. IEEE Transactions on Network Science and Engineering, 2020

[76] L. Babun, H. Aksu, A. S. Uluagac, Identifying Counterfeit Smart Grid Devices: A Lightweight System Level Framework, in: 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–6 (May 2017). doi:10.1109/ICC.2017.7996877.

[77] L. Babun, H. Aksu, A. S. Uluagac, A System-level Behavioral Detection Framework for Compromised CPS Devices: Smart-Grid, in: ACM Transactions on Cyber-Physical Systems, 2019, pp. 1–28 (Nov 2019). http://doi.acm.org/10.1145/3355300.

[78] Babun, Leonardo (Miami, FL, US), Aksu, Hidayet (Miami, FL, US), Uluagac, Selcuk A. (Miami, FL, US). 2018. Detection of Counterfeit and Compromised Devices Using System and Function Call Tracing Techniques. (July 2018). https://www.osti.gov/biblio/1463864

[79] Babun, Leonardo (Miami, FL, US), Aksu, Hidayet (Miami, FL, US), Uluagac, Selcuk A. (Miami, FL, US). 2019. Method of Resource-limited Device and Device Class Identification Using System and Function Call Tracing Techniques, Performance, and Statistical Analysis. (March 2019). https://patents.google.com/patent/US10242193B1/en

[80] L. Rondon and L. Babun and A. Aris and K. Akkaya and A. S Uluagac PoisonIvy: (In)secure Practices of Enterprise IoT Systems in Smart Buildings. accepted at BuildSys '20, 2020

A Sample Survey Questions

We present a list of representative IoT privacy survey questions from all the categories due to space constraints. The entire user study is available at https://anonymous.com.

A.1 User Characterization

1. Do you use, have used, or are you planning to use any IoT device?
   ( ) Yes
   ( ) No
   ( ) Maybe

2. What is your technical experience with IoT apps?
   ( ) I can build, implement, code my own IoT app
   ( ) Installed/can install/configure an IoT app using the source code available online
   ( ) Installed/can install/configure an IoT app's marketplace (Google Play, App Store, etc.)
   ( ) I just know how to press the buttons
   ( ) I have no idea how to deal with IoT apps

A.2 Security and Privacy Concerns in Smart Apps

1. What information would you consider sensitive if used in IoT apps/devices? Please check all that apply.
   ( ) My personal data (e.g., email address, phone number, residential address, etc.) ( ) Whatever I do, my behavior (e.g., when I arrive home, when I leave home, I go to sleep, etc.)
   ( ) My (or my devices') location (e.g., my location while using the apps, etc.)
   ( ) My device settings/the way I configure the devices (e.g., Time of the day the lights turn On/Off, my thermostat temperature settings, etc.)
   ( ) My (or my devices') timing (e.g., the time passed since I left home, the time passed since I went to sleep, etc.)
   ( ) Information from my devices (e.g., device type, manufacturer, device IDs, etc.)
   ( ) Data from my devices (e.g., door state open or close, light on or off, etc.)
   ( ) Other

   If you selected "Other", please explain:

2. Have you heard or personally have privacy concerns on the use of the IoT devices and systems?
   ( ) Big concerns ( ) Some concerns

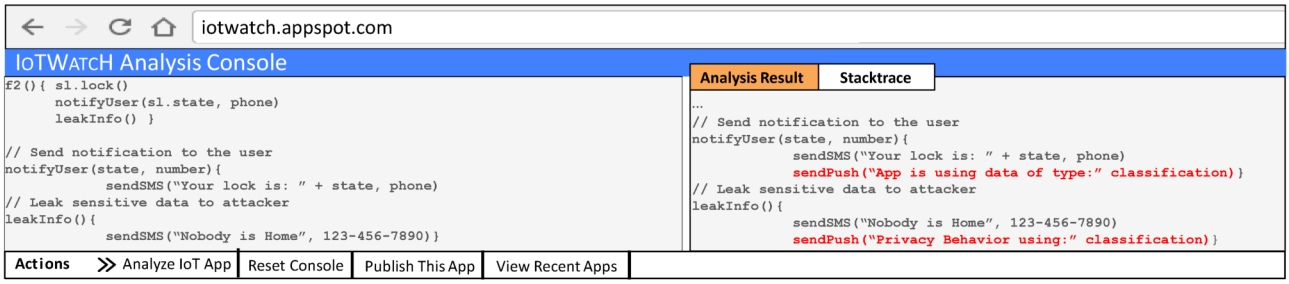


Fig. 1. The left console is the analysis area where the user inputs the original IoT app. The right console returns the output of the instrumentation process. We made IoTWatcH's instrumentor freely available to the community at `https://IoTWatcH.appspot.com/`.

( ) I do not, but I know someone that does
( ) Never thought about it, until now
( ) I do not care

A.3 Privacy Analysis Tools and Features

1. Do you think there is a need for a tool to check for security privacy risks from the smart apps?
   ( ) Yes
   ( ) No
   ( ) Maybe

2. Would you be willing to use available automatic tools that analyze and modify smart apps to enable security and privacy analysis in real-time?
   ( ) Yes
   ( ) No
   ( ) Maybe

B Online App Instrumentor

We made the instrumentor available online at: `https://IoTWatcH.appspot.com/`. Figure 1 depicts details of the online version of IoTWatcH's instrumentor. At the left console, the user inputs the IoT app source code that needs to be modified to enable IoTWatcH, and at the right console, the tool automatically returns the IoTWatcH-instrumented app. Below, we detail the implementation steps of IoTWatcH.

C IoTWatcH Analyzer

Model Construction. Our search for an adequate corpus that characterizes IoT app's data-flows faced particular challenges. First, we could not find any existing IoT corpus. Second, most of the datasets available online contain raw unlabeled data that would require a considerable amount of pre-processing time and resources. Third, the privacy labels considered by IoTWatcH could not be inferred from n-gram shingles extracted from a single corpus only. Thus, we combined different knowledge-based datasets to create a larger corpus. We combined the natural language datasets from Google Books N-grams [21] and Wikidata [56], which contain strings related to geographic (location), economic (devices, user's goods), climate (location), and encyclopedic (general knowledge) datasets. We structured, cleaned, and manually labeled the crawled data. First, we divided the corpora into single shingles (i.e., n-grams of n=1). Then, for cleaning purposes, we filtered out punctuation and stop words. We tested the first NLP model with 61 IoT strings extracted from 45 market IoT apps [48, 50]. An average value of 72% accuracy proved that the first considered model could not accurately represent information extracted from IoT environments. Based on these results, we decided to train the NLP model using specific IoT corpora only. Finally, in Figure 2 we detail the distribution of IoT strings per privacy label used in our NLP model.

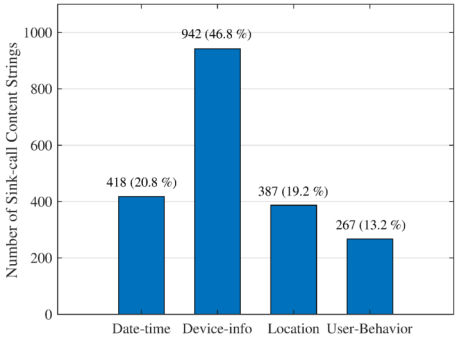


Fig. 2. Distribution of privacy labels within the IoT corpus used to train IoTWatcH's NLP model. We assigned 2014 privacy labels to strings extracted from messaging and Internet communications in 380 market IoT apps.

Listing 1. An example of a JSON object sent from an IoT app to IoTWatcH for further analysis.

```
/* An example of a JSON object sent to Daint analytics tool */

 data "{
  'exfiltration':{
    'texttype':'PLAIN_TEXT',
    'calltype':'Messaging',
    'phone':'111-111-1111',
    'content':'The door was opened for 10 min'
    'userrecipients':'123-456-7890',
  }
}" "https://iotwatchanalyticstool.com/classifytext/"
```

Listing 2. An example IoTWatcH response as a JSON object received by an IoT app.

```
/* An example of a JSON object as response from our analytics tool
     */

 data "{
  'exfiltration':{
    'texttype':'PLAIN_TEXT',
    'classification':['device-info', 'date-time']',
    'risklevel': 'data leakage'
  }
}"
```

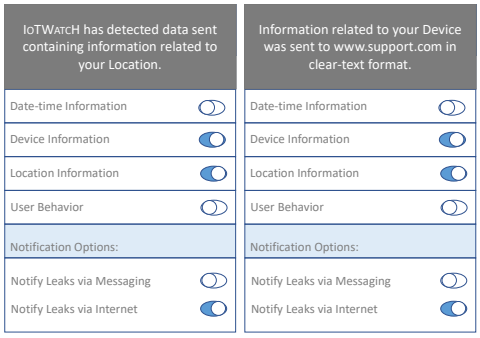
Sample API Objects Listing 1 illustrates a sample JSON object used to send data from an IoT apps to IoTWatcH analyzer. Here, information regarding a messaging content and its recipients is collected. The findings are then sent back to the user via another JSON object similar to the one depicted in Listing 2.

LC	MwoLoPC	IwoLoPC	MoLoPC	IwLoPC	Total	% Total
Device-info	47	12	43	17	119	52.3
Date-time	7	1	7	0	15	6.6
User-behavior	21	2	34	10	67	29.3
Location	8	0	14	5	27	11.8
Total	83	15	98	32	228	100

LC - Privacy Label Category
MwoLoPC - Messaging without Leaks or Privacy Concerns
IwoLoPC - Internet Calls without Leaks or Privacy Concerns
MwLoPC - Messaging with Leaks or Privacy Concerns
IwLoPC - Internet Calls with Leaks or Privacy Concerns

Table 2. Distribution of privacy labels

Privacy Labels Table 2 summarizes the distribution of the privacy labels during evaluation. One can notice that, out of 228 different labels assigned, 52.3% corresponds to device-info, followed by a 29.3% of user-behavior information. We also show in Table 2 the number of labels assigned to the different call types (i.e., messaging and Internet). For instance, 83 labels were assigned to strings extracted from messaging that do not leak information, while 98 labels were assigned to strings extracted from messaging that leaked information. On the other hand, Internet communications without privacy concerns received 15 privacy labels while 32 were assigned to Internet communications that leak



(a) (b)

Fig. 3. IoTWatcH's findings are informed to the users through push notifications. The findings include (a) the privacy labels assigned to the taint-sink content, and (b) the clear-text transmitted out of an IoT app.

data. With this distribution, one can notice that the majority of labels were assigned to leaked data. These results reflect on the fact that privacy leakage constitute a serious problem in IoT.

IoTWatcH Notification System We illustrate the notification system implemented by IoTWatcH to inform its findings to the user. In Figure 3(a), the privacy tool instruments a push notification to inform the user that an IoT string related to the user's location was sent out of the app. In Figure 3(b), the tool informs that sensitive information was transmitted in clear-text, potentially making the data available to passive eavesdroppers.

D Additional Evaluation Results

Table 1 details average metric values in classifying IoT strings to privacy labels for every different *th*. Also, we present the average metric values after combining results from all considered privacy labels and thresholds. One can verify that IoTWatcH obtains the best performance for threshold values of 0.4, which supports the results showed in Section 7, Figure 6(a). For *th* values higher than 0.5, the accuracy decreases for the privacy labels of device-info and user-behavior. This is mainly because the evaluation of semantically-limited strings related to these two privacy labels requires more sophisticated analysis. For instance, user-behavior achieved incorrect or lower classification scores as the string "IoT switched to sleep mode" cannot be easily related to user activities. We obtained similar results for recall metrics; however, recall values of date-time are lower compared to other labels. We found that date-time information could be easily missed from short strings, which makes

Evaluation Metric	Classification Thresholds									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	Average
Accuracy	0.9537	0.9623	0.9692	0.9743	0.9623	0.9469	0.9296	0.9092	0.8750	0.9425
Recall	0.9754	0.9400	0.9379	09341	0.8821	0.8419	0.7919	0.7359	0.6263	0.8517
Precision	0.8470	0.8776	0.9163	0.9587	0.9803	0.9833	0.9929	0.9975	0.9835	0.9501
Specificity	0.9283	0.9498	0.9682	0.9772	0.9823	0.9855	0.9876	0.9907	0.9907	0.9734

Table 1. Evaluation results in classifying IoT strings for all classification thresholds. The rightmost column is the average values.

Type*	App Name	Leak Type**	Recipient	Content	Behavior	Privacy Labels
	Squeeze Box Controller [48]	I	http://$ip:$port	"Open box"	●	Device-info
	StatHat Quick Start [48]	I	http://api.stathat.com/ez	"Thermostat1:thermostat"	●	Device-info
❶	ThingSpeak Logger [48]	I	http://api.thingspeak.com	"DeviceID:333:Key:123"	●	Device-info
	User Lock Manager [48]	M	defined by user	"User no longer has access to the door"	○	Device-info, User-behavior
	Smart Lock [48]	M	defined by user	"SmartLock disabled. Door unlocked indefinitely"	○	Device-info, User-behavior
	Fire Alarm [28]	I	http://stmalware/maliciousServer.php	–	●	–
	Ransomware [28]	I	http://stmalware/maliciousServer.php	–	●	–
❷	Remote Command [28]	I	http://stmalware/maliciousServer.php	–	●	–
	Spyware [28]	M	$123-456-7890$	"Doors locked after everyone departed"	●	Device-info, User-behavior
	User Event [28]	M	$123-456-7890$	"Everyone is away and Hub ID is 123"	●	Device-info, Location, User-behavior

* ❶ is for Market IoT apps and ❷ is for Malicious IoT apps (Handcrafted)
** I is for Internet and M is for Messaging. ● is for Privacy Risk and ○ is for Privacy Preference of a user.

Table 3. Examples of privacy risks and the use of sensitive information in market and malicious IoT apps.

the label specially vulnerable to false negative events. The precision and specificity improves with *th* for all the labels, which denotes a remarkable confidence of the model for those results with the highest classification scores.

Results from Real-life Apps. Table 3 presents IoTWatcH's results and findings after analyzing real market and handcrafted apps. As can be seen, IoTWatcH was able to identify leaks happening due to both messaging and Internet communications. In all the cases, it reported back to the user the type of sensitive information leaked (based on the four privacy labels) and its recipients (phone number in case of messaging and server URL in case of Internet calls). In some specific cases ("User Lock Manager" and "Smart Lock" apps), no leak was detected as IoTWatcH found the recipients were defined by the user at install time, yet IoTWatcH informed the user the type of information handled by the apps accordingly. Finally, we note that privacy leaks were flagged in all the cases as the apps did not report handling these types of information nor their recipients to the user via app's description blocks.

E Evaluation Metrics

The performance metrics used during IoTWatcH's evaluation:

- *True Positive (TP)* represents the number of times a privacy label is correctly applied to an IoT string for certain threshold *th*.
- *True negative (TN)* represents the number of times a privacy label is correctly discriminated for certain threshold *th*.
- *False Positive (FP)* is the number of times a privacy label is incorrectly assigned to certain IoT string for certain threshold *th*.
- *False Negative (FN)* is the number of times a privacy label is incorrectly discriminated for certain threshold *th*.
- *Accuracy* is the overall ability of IoTWatcH to correctly apply privacy labels to the IoT string for every different *th*.
- *Recall* is the ability of the classifier to correctly assign the privacy labels to a specific IoT string after considering both the correctly classified and the incorrectly ignored privacy labels for every value of *th*.
- *Precision* is the ability of our classifier to correctly apply the privacy labels to a specific IoT string after considering both the correct and the incorrectly applied privacy labels for every value of *th*.
- *Specificity* is the ability of our tool to discriminate the privacy labels for every different *th*.