Context-rich Privacy Leakage Analysis through Inferring Apps in Smart Home IoT

Yuan Luo, Long Cheng, *Senior Member, IEEE,* Hongxin Hu, *Senior Member, IEEE,* Guojun Peng*, Danfeng (Daphne) Yao, *Senior Member, IEEE*

***Abstract*—Emerging Internet of Things (IoT) systems leverage connected devices to enable intelligent and automated functionalities. Despite the benefits, there exist privacy risks of network traffic, which have been studied by the previous research. However, with the current privacy inference remaining at the event-level, potential privacy risks are underestimated, which, as our study shows, can be much higher than previously reported through app-level traffic analysis. A key observation of our research is that IoT event-triggered traffic is generated by apps, which often adopt an if-trigger-then-action (trigger-action) programming paradigm. We utilize this feature to develop fingerprints to differentiate running apps, and learn context-rich privacy-sensitive information from apps. In this paper, we present a privacy leakage analysis called ALTA to infer running apps in smart home IoT environments. First, ALTA identifies app fingerprints through static analysis, and extracts sensitive information from app descriptions and input prompts. Then, through dynamic traffic profiling, it learns traffic fingerprints of apps. Finally, ALTA matches the fingerprints of app and traffic, and thus is able to pinpoint which app is running from IoT traffic at runtime. To demonstrate the feasibility of our approach, we analyze 254 SmartThings applications via program and natural language processing (NLP) analysis. We also perform the app inference evaluation on 31 apps executed in a simulated smart home. The results suggest that ALTA can effectively infer running apps from IoT traffic and learn context-rich information (*e.g.*, health conditions, daily routines, and user activities) from apps with high accuracy.**

***Index Terms*—privacy risk, program analysis, traffic analysis, smart home**

I. INTRODUCTION

The rise of the Internet of Things (IoT) platforms such as Samsung SmartThings [1], Google Home [2] and Apple Homekit [3], enables fast and low-cost development of IoT solutions. With more than 450 IoT platforms in the marketplace [4], the number of IoT devices will grow from 23 billion in 2018 to 75 billion in 2025 [5]. IoT is increasingly gaining popularity in home environments, which empowers users to set up connected and automated smart homes, *e.g.*, voice-controlled lights and remote-controlled door locks.

*Corresponding author

This work was done when Yuan Luo was visiting at Virginia Tech.

Y. Luo and G.J. Peng are with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, and School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China (email: leonnewton@whu.edu.cn, guojpeng@whu.edu.cn).

L. Cheng and H.X. Hu are with the School of Computing, Clemson University, USA (email: lcheng2@clemson.edu, hongxih@clemson.edu).

D.F. Yao is with the Department of Computer Science, Virginia Tech, VA, USA (email: danfeng@vt.edu).

IoT security especially smart home security has attracted considerable attention [6]–[17]. Despite intensive efforts, security problems especially privacy risks remain unsolved [18]–[27]. Researchers have shown that encrypted network traffic from IoT devices exhibits distinguishable patterns (*e.g.*, the packet frequency and length), which provide useful clues to infer privacy-sensitive information [28]–[33].

The status change of a stand-alone smart device is typically defined as an event [34] in smart home IoT. Whenever an event happens, there is corresponding traffic between the smart hub (to report status changes) and cloud backend (to send back control commands). This cause-and-effect relationship between event and traffic has been utilized to infer sensitive information in IoT (*e.g.*, the status of switches, plugs, and simple user activities). Apthorpe *et al.* [28], [31], Yoshigoe *et al.* [29], and PINGPONG [35] utilize traffic patterns, namely traffic rate, packet size, and packet interval to identify events generated by devices. For example, contact and switch sensors generate *open/closed* and *on/off* events respectively. However, there exists ambiguity for the event-level traffic analysis to derive sensitive information. One type of event can correspond to different usages of the device. For instance, when detecting an *open* event from a contact sensor, attackers cannot figure out whether a window is opened or a garage door is opened (both are *open* events), which reflects totally different situations but exhibits same traffic patterns (see Figure 2 and Table I for details in Section III and IV). Further, as an advanced approach, based on machine learning techniques, user activities (*e.g.*, a user is walking to the bedroom) are identified through a sequence of events. Peek-a-boo [30] leverages a set of feature vectors (*e.g.*, packet size, packet interval, and statistical features of traffic time series) to deduce users' daily activities. However, attackers need to collect traffic in different locations for an extended period (*i.e.*, traffic-hungry), which increases the chances of exposure.

Another problem is that the privacy risks are usually underestimated by simply extracting events from traffic, which achieves coarse-grained and inaccurate user activity inference due to the lack of the context of device usages and user activities. To bridge this gap, instead, an app-level analysis can extract context-rich (*i.e.*, under what scenarios the devices are used) privacy-sensitive information from app descriptions and input prompts. For example, for a contact event, it can represent that a front door or refrigerator door is opened. For the former, it can be a user coming home (useful for a burglar). For the latter, it can imply that a user is reaching medicine stored in the refrigerator (useful for targeted advertising and blackmail). Context-rich information can be used to differentiate such situations that existing work cannot obtain and distinguish. For the sake of brevity, in this paper, we use apps to denote SmartApps on the Samsung SmartThings platform (details in Section II-A). Recently, HoMonit [32] monitors states of apps rather than events to conduct anomaly detection. However, it does not extract sensitive information from apps and adopts the white-box assumption (*i.e.*, knowing which app is running beforehand). The strong assumption makes the method infeasible and unpractical for launching privacy attacks.

Distinctive from existing work, our objective is to develop a *new* IoT privacy leakage analysis to advance the event-level privacy inference to the app-level, which overcomes the above limitations. The key enabler is to harvest context-rich sensitive information contained in app descriptions and input prompts. To achieve such app-level privacy inference, however, is nontrivial. In particular, we need to address three design challenges below.

- *How to differentiate different running apps, especially those with the same trigger-action pair*? We adopt a black-box assumption, which means we do not know what apps are running in a smart home environment. Moreover, multiple apps could present almost the same traffic pattern. It is because they can be triggered by the same event (*e.g.*, all apps subscribe to a movement) and generate the same command (*e.g.*, all apps operate switches). We propose fingerprints to capture subtle features (*e.g.*, *if-conditions*) to differentiate apps.
- *How to extract and match fingerprints between apps and IoT traffic*? A fingerprint is a sequence of events in a specific order, which needs to be identified from numerous events. For example, a *contact-switch* event pair can be a part of a fingerprint but also can be an irrelevant trigger-action pair. Also, events tend to have a dynamic and interfering nature. For example, *motion.active* and *inactive* event produce the same traffic fingerprint (packet size, direction, and the number of packets). We develop a matching algorithm utilizing the pair-occurrence of events and app correlation to match fingerprints.
- *How to automatically extract privacy-sensitive information from apps*? Sensitive information is contained in descriptions and input prompts of apps. The challenge is to accurately locate and extract privacy-sensitive information. We leverage NLP analysis and trigger-action nature of apps to address this problem.

In this paper, we present an app-level privacy leakage analysis named ALTA to address the above challenges. ALTA performs program analysis to extract trigger-action dependencies and *if-conditions* to construct fingerprints of apps. Then, ALTA uses NLP analysis to extract sensitive information from app descriptions and input prompts. Through dynamic traffic profiling, ALTA learns traffic fingerprints of apps. Finally, at runtime, ALTA can infer running apps via matching fingerprints extracted from the IoT traffic and app fingerprints learned from program analysis. The contributions of the paper are outlined as follows:

- *New techniques for traffic analysis.* We leverage program analysis to empower traffic analysis from learning events to context, which include techniques for automatically constructing fingerprints from the source code of IoT apps and raw traffic, and an algorithm for matching fingerprints of apps and traffic.
- *A new system for privacy leakage analysis.* We present ALTA for inferring running apps from traffic to extract context-rich privacy-sensitive information from app descriptions and input prompts, which takes a step towards exposing higher privacy risks in smart home IoT environments. Although demonstrated on Samsung SmartThings, approaches of ALTA can be potentially applied to other IoT platforms.
- To the best of our knowledge, we conduct the first systematic static analysis which investigates trigger-action dependencies, *if-conditions*, and app categories over 254 apps in the Samsung SmartThings platform. We find 61 trigger-action pairs, 113 apps containing trigger-action dependencies, 21 features of *if-conditions* and implications of app functionalities. We demonstrate the effectiveness of ALTA by experiments and case studies. We also discuss countermeasures to thwart such a privacy leakage analysis.

Roadmap. The rest of the paper is organized as follows. Section II presents the background and our threat model. Section III introduces the motivation for our work. Section IV elaborates the detailed design of ALTA to infer context-rich sensitive information. Section V presents the systematic static analysis findings and the evaluation of ALTA. Section VI discusses the limitations of this work and countermeasures. Section VII surveys and provides a comparison of related work to ours. Finally, Section VIII concludes the paper.

II. BACKGROUND & THREAT MODEL

A. Background

Samsung SmartThings. In this paper, we focus on the Samsung SmartThings, which is the largest smart home platform [32]. The SmartThings platform adopts the cloud-centric architecture, which is composed of IoT devices, the hub, SmartApps, and the cloud backend. The hub (which is usually in proximity to various IoT devices) is responsible for the communication between physical devices and the cloud backend. An IoT app is named a SmartApp (we name app for the sake of brevity in the rest of this paper) in SmartThings and developed in the Groovy language, which is executed in the cloud backend. The cloud backend 1) instantiates SmartDevices that are wrappers for physical devices interacting with apps through triggers and actions, and 2) provides a running environment for apps. It allows the remote control of IoT devices and receiving commands from mobile apps.

SmartThings adopts the *capability* model as permission management. Capabilities consist of *attributes* and *commands*. Attributes are the characteristics of devices. Value changes of attributes are *events* [34], which indicate state changes of devices. Commands are used by apps to control devices. Note that one capability can indicate different functionalities in different apps. As illustrated in Figure 1, capability *contact* has *open* and *closed* events. Contact can represent states of

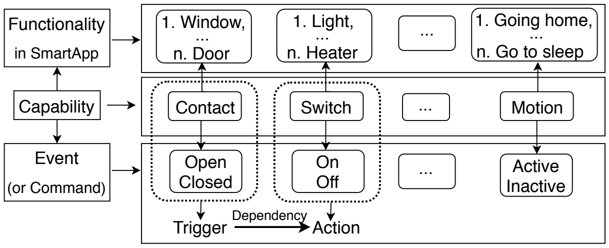


Fig. 1: Terminologies of SmartThings.

doors in an app and windows in another app, which is decided by the context where the device is used. Capability *switch* has commands *on* and *off* to control devices. Meanwhile, if event A and command B has a dependency (*i.e.*, A can cause B to happen), then we use *trigger* (or trigger event) to represent event A and *action* (or action event) to represent command B.

The trigger-action programming paradigm. We show an example of the trigger-action paradigm in Listing 1. This app detects the moment when a door is opened (trigger: *contact.open*), then it sends a command (action: *switch.on*) to turn lights on. The input prompt (lines 6 and 10) collects the input information from the user, which includes the devices being operated. The app subscribes to the trigger event *contact.open* and registers an event-handler method (line 15). Whenever the *contact.open* is triggered, the handler function executes the action command in response to the device status change. The trigger event (line 17) and the action command (line 21) generate network traffic between the hub and cloud backend to report these changes. Traffic generated by an action command normally following the traffic of a trigger event exhibits distinguishable patterns. Since this traffic is generated because of status changes of devices and apps, it potentially leaks privacy-sensitive information. However, as challenges described in Section I, our work is to detect these apps and extract heterogeneous context-rich sensitive information.

Listing 1: Code snippet of the trigger-action programming paradigm.

```
1 definition(
2    description: "Turn your lights on when an open/close
         sensor opens and the space is dark.",...
3 )
4 preferences {
5   //Input prompt of choosing a door (trigger event)
6   section("When the door opens...") {
7     input "contact1", "capability.contactSensor", title: "
         Where?"
8   }
9   //Input prompt of choosing a light (action command)
10  section("Turn on a light...") {
11    input "switch1", "capability.switch"
12  }
13 }
14 //Subscribing to the trigger event: contact.open
15 subscribe(contact1, "contact.open", contactOpenHandler)
16 //Event handler, evt denotes the trigger event that
       generates trigger traffic
17 def contactOpenHandler(evt) {
18  def lightSensorState = luminance1.currentIlluminance
19  // If-condition
20  if (lightSensorState != null && lightSensorState < 10) {
21    switch1.on() //Action command to generate action
         traffic      }
22 }
```

B. Threat Model

In this paper, we consider 1) users deploy a number of IoT devices at home. The traffic is encrypted and bidirectional between the IoT hub and the cloud backend. The hub (which is usually in proximity to various IoT devices) is responsible for the communication between physical devices and the cloud backend, and 2) adversaries have a desire to collect private information such as the daily routine of a user. The capabilities of adversaries are that they can eavesdrop on encrypted traffic communication, but cannot manipulate the traffic [28]–[30]. Specifically, we identify three possible scenarios:

- Attackers need to physically deploy eavesdropping devices. This can be achieved through wireless eavesdropping or devices to mirror traffic from cables (*e.g.*, cables of an apartment building). After deploying eavesdropping devices, attackers can obtain the traffic remotely.
- Through cyber attacks, attackers can access the traffic flow remotely. This can be achieved in various ways (*e.g.*, exploit router bugs to place backdoors [36], weak router passwords, and ARP attacks [37]).
- Malicious Internet Service Providers (ISP) may obtain private information. For example, they may share user habits to advertisers to gain profit [38].

We also assume that the adversary can access to the same types of IoT devices as the benign smart home users, and access to the source code of IoT apps. This assumption is reasonable for the SmartThings platform since apps are open-source from SmartThings Marketplace or public GitHub Repository [39], and IoT devices are readily available on the market. This enables the adversary to collect a set of labeled traffic for dynamic profiling of IoT traffic. We discuss the possible method to handle closed-source apps in Section VI. Adversaries do not know how many apps are running in a smart home environment. We assume that the platform software and hardware are trusted.

III. Motivation

The need for in-depth app-level traffic analysis. We elaborate the need for ALTA and demonstrate the limitation of existing work with examples. A detailed comparison of existing work is discussed in Section VII. As illustrated in Figure 2, *Monitor on Sense*, *Someone is Knocking Door*, and *Open Garage Door When I Arrive* are three apps that have the same trigger-action (*i.e.*, *acceleration-switch*) pair. Existing work [28]–[30], [32] focus on identifying *acceleration.active* (E1), *switch.on* (E2), *switch.off* (E3), and *contact.open* (E4) events independently, rather than which app generates the observed traffic. Thus they cannot infer context-rich information (*i.e.*, the scenarios in which devices are triggered) at the app-level. For example, visitors knocking on the door and users driving the car to leave the garage are two different yet valuable information for attackers. Specifically, there is a gap between deducing event-level and context-rich app-level privacy-sensitive information. However, the challenge is that one type of device can serve for multiple purposes (*e.g.*, a switch can be used to turn on a light or open a garage door) and how to differentiate apps in the same trigger-action pair (*e.g.*, several apps use *acceleration-switch* pairs). Our approach leverages the *trigger-action dependency* and *if-conditions* to infer different running apps. For example, a time delay (*if-condition* A) and an app correlation (*if-condition* B) can be captured by our technique (details in Section IV). Our progress is two-fold: 1) ALTA utilizes app descriptions and input prompts of apps to deduce sensitive information more accurately. Taking examples in Figure 2, with context-rich sensitive information of apps, an attacker can know whether it is someone knocking the door or a user leaving home when the *acceleration.active-switch.on* traffic is observed. And 2) ALTA provides an app-level analysis, which derives more information through NLP analysis. For example, for app *Open Garage Door When I Arrive*, existing work derives the vibration event and a device is turned on/off from *acceleration* and *switch* events. However, ALTA can deduce that the user is driving a car leaving the house, which is learned from the app description and input prompt. This information is far more valuable for burglars than vibration or device status.

App	Trigger	Action	Functionality (Privacy-sensitive information)
Monitor on Sense	acceleration	switch	When vibration is sensed Turn on switch
Someone is Knocking Door	acceleration	switch	When someone knocks on this door Turn on the light and then off
Open Garage Door When I Arrive	acceleration	switch	When the car door is opened Open the garage door

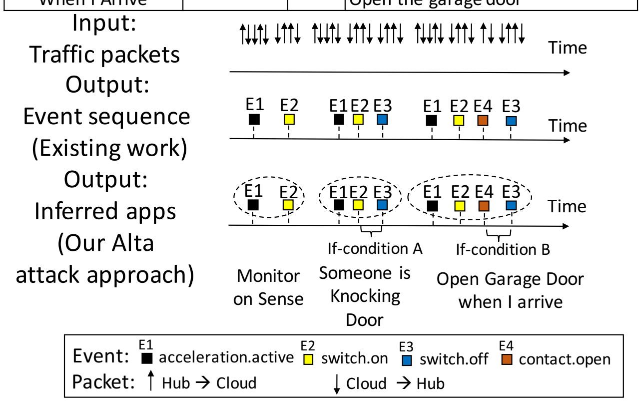


Fig. 2: Existing work [28]–[30], [32] identifies individual events (*e.g.*, E1, E2, E3, and E4). Our analysis can infer and differentiate these apps and extract context-rich sensitive information.

Privacy risks of context-rich sensitive information. Conventional event-based privacy attacks mainly extract coarse-grained user activities, which is constrained by the limited information that can be derived from events. For example, *contact* and *motion* events can only indicate the movements of a user. However, there are overlooked context-rich information contained in descriptions and input prompts of apps. Viewed through a security lens, we highlight that it is the *context* (*i.e.*, under what scenarios the devices are used) that may induce more devastating loss to users. The privacy risks are largely underestimated. Specifically, user activities, daily routine, health conditions, and hobbies information can be derived.

IV. ALTA DESIGN

In this section, we show how ALTA addresses three design challenges listed in Section I. Figure 3 illustrates the workflow of ALTA, which is composed of two stages: 1) the learning phase captures features to differentiate apps and extract sensitive information; and 2) eavesdropping phase to infer running apps from raw traffic. We elaborate the details as follows.

❶: APP ANALYSIS. This module includes *Program Analysis* and *NLP Analysis*. We use the program analysis to identify trigger-action dependencies and extract *if-conditions* from source code to generate app fingerprints. Also, app descriptions and input prompts are extracted. We perform the NLP analysis to obtain sensitive information from descriptions and input prompts (details in Section IV-B).

❷: TRAFFIC ANALYSIS. This module aims to obtain traffic fingerprints of apps from learning-phase traffic. First, through actuating devices and recording sequences of packets, we identify traffic features (*e.g.*, packet length and direction) of events (*e.g.*, *contact.open*). Then, through triggering apps with trigger-action and *if-conditions*, we learn the sequence of events of a specific trigger-action or *if-condition*. For example, a device status *if-condition* can be a *lock.locked* event before a trigger-action pair. Finally, trigger-action dependencies and *if-conditions* are extracted by pair-occurrence and app correlation features (details in Section IV-C).

❸: FINGERPRINT MATCHING. ALTA matches app fingerprints and traffic fingerprints to infer running apps by comparing trigger-action dependencies and *if-conditions*. Once an app is inferred, ALTA deduces sensitive information from the results of NLP analysis (details in Section IV-D).

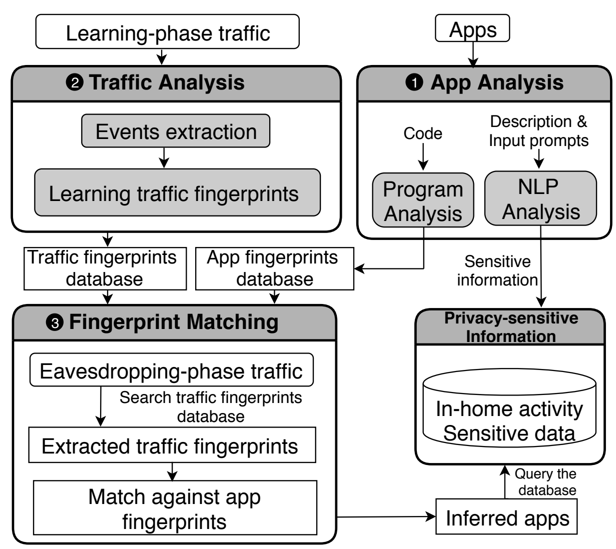


Fig. 3: ALTA attack workflow.

A. DEFINITION OF FINGERPRINTS

App fingerprints. We first present the definition of app fingerprints. We formalize *app fingerprints* as a 3-tuple $\mathbb{F}_{\mathbb{A}}$ = ($\mathcal{T}$, $\mathcal{A}$, IFs), where $\mathcal{T}$ is the trigger event, $\mathcal{A}$ is the action command, and IFs means *if-conditions* that are checked in the paths from triggers to actions. IFs consist of $\mathcal{T}$, $\mathcal{DS}$, $\mathcal{D}$, and $\mathcal{P}$. $\mathcal{DS}$ is the device status; $\mathcal{D}$ represents the elapsed time between the trigger and action; $\mathcal{P}$ denotes physical environment parameters. $\mathcal{T}$, $\mathcal{DS}$, $\mathcal{D}$, and $\mathcal{P}$ are important factors which are often checked in *if-conditions* within the trigger-action path of an app. It is possible that one app may contain multiple trigger-action paths, thus contains multiple $\mathbb{F}_{\mathbb{A}}$. We utilize app analysis to extract $\mathbb{F}_{\mathbb{A}}$ from apps.

We show examples of *if-conditions* in Listing 2. Trigger event $\mathcal{T}$ indicates that an app is triggered by a specific event (line 2). Since both triggers and actions generate traffic packets, pinpointing trigger-action pairs can differentiate apps. Some apps check the device status ($\mathcal{DS}$) before performing an action (line 6). For example, *UnlockItWhenIArrive* additionally checks the lock state before performing the unlock action. The elapsed time ($\mathcal{D}$) means apps wait for a certain time before conducting an action (line 10). For example, some apps will not turn off the lights until users finish the movements. The physical environment ($\mathcal{P}$) (line 14) indicates parameters of the living environment, *e.g.*, the illuminance.

Listing 2: Code snippet of *if-conditions*.

```
1  //Trigger event(𝒯)
2  if (evt.value == "active") {
3    switch1.on()
4  }
5  //Device status(𝒟𝒮)
6  if (currentLock == "locked") {
7    lock1.unlock()
8  }
9  //Elapsed Time(𝒟)
10 if (elapsed >= threshold) {
11   switches.off()
12 }
13 // Physical environment (𝒫)
14 if (lightSensorState < 10) {
15   switch1.on()
16 }
```

Traffic fingerprints. We define traffic fingerprints as an event sequence $\mathbb{F}_{\mathbb{T}}$ = (E_T, E_A, E_{IF}), where E_T and E_A are events that represent trigger events and action commands respectively. E_{IF} is events that represent *if-conditions*. Different apps have different traffic fingerprints since they have different trigger-action pairs and *if-conditions*. We utilize traffic analysis to identify and extract E_T, E_A, and E_{IF} from raw traffic (details in Section IV-C).

B. App Analysis

We elaborate the detailed design of two modules in our app analysis: Program analysis and NLP analysis.

Program analysis. This analysis is to harvest trigger-action dependencies and *if-conditions* as app fingerprints. Existing program analysis tools in the smart home cannot be directly applied to our analysis for two reasons: (1) data flow analysis studies (*e.g.*, SaINT [9]) focus on trace sensitive data from sources to sinks. However, our analysis requires control flow analysis to find and collect execution paths from trigger events to action commands; (2) we need to discover and gather all *if* statements contained in each trigger-action path. First, Alta filters out those apps without any trigger subscription or action command. Then, Alta builds the inter-procedural control flow graph (ICFG) of every app that contains trigger-action dependencies. Next, Alta finds all paths from an action command to its corresponding event-handler in the ICFG. If yes, there exists event-triggered traffic. Finally, Alta collects any *if-condition* appeared in that path. As illustrated in Figure 4, we demonstrate the detailed process using the example in Listing 1.

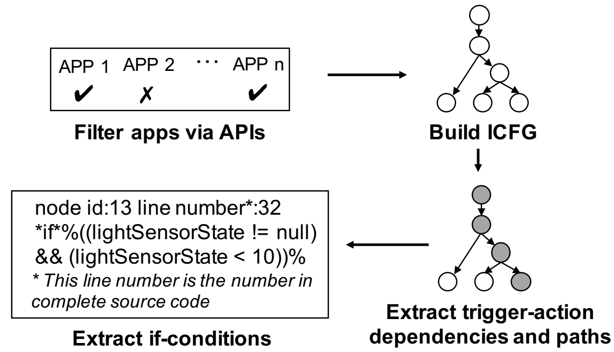


Fig. 4: The workflow of program analysis.

Step 1: Filter apps via APIs. This step is to filter out apps that do not have any trigger subscription or action command. These apps do not generate event-triggered traffic and thus are out of the scope of this paper. To this end, we search 1) *subscribe* statements that specify *event-handler* and *trigger* in apps (*e.g.*, line 15 of Listing 1); and 2) *action* commands that send commands to devices (*e.g.*, line 21 of Listing 1). We exclude apps that do not contain the above two kinds of statements.

Step 2: Extract trigger-action dependencies. We build ICFG to find possible paths from a *subscribe* statement to an *action* command. First, we record the *trigger* and *event-handler* pair in the *subscribe* statement. Next, we search any *action* command, and trace back recursively in each path to find the caller function of this action. We stop the recursive procedure until finding the *event-handler* function, *e.g.*, *contactOpenHandler* in line 17 of Listing 1. Finally, we obtain the *event-handler→action* path. In Listing 1, the path is from *contactOpenHandler* (line 17) to *switch1.on()* (line 21), and the *trigger* is *contact.open*.

Step 3: Extract if-conditions. Once we get all paths from *triggers* to *actions*, we trace back each statement starting from an action command and find if it is within a conditional block. In Listing 1, we identify an *if-condition* in line 20. This *if-condition* checks the illuminance of the home environment, which is useful to fingerprint this particular app. *If-conditions* are identified and extracted by pattern and keywords. The pattern is that a variable is compared with a certain value (*e.g.*, evt.value == "active"). Both variables and values are identified by keywords. For example, the variable name of the trigger event is always *evt.value* or *event.value*. Values can be *active*, *open*, *on*, *etc*. Also, values of elapsed time and physical environment can be numbers. *If-conditions* together with the *trigger* and *action* can be unique fingerprints of apps. Alta scans all apps in our dataset and extracts their fingerprints.

Implementation. SmartApps are written in the Groovy programming language. Alta utilizes the Abstract Syntax Tree (AST) transformation at the semantic analysis phase to analyze apps, which is a feature of Groovy. We utilize *ClassCodeVisitorSupport* to traverse each statement (via *visitStatement* API) to identify the entry point of an app, *subscribe* statements, and functions. We set a statement as a node. Further, we identify call relations, parent and predecessor relations, and metadata (*e.g.*, line number) through iterate each node to build ICFG. Then, we use a stack to discover and store the path from a trigger event to an action command via scanning the ICFG. If-statement blocks and else-statement blocks are both analyzed to decide the execution paths of descendants of if statements. Finally, since trigger-action devices are identified, we collect app descriptions and the trigger (action) input prompt.

NLP analysis. The NLP analysis takes app descriptions and input prompts as input and the goal is extracting privacy-sensitive information. When information is sufficient and clear in app descriptions, we can extract sensitive information as the method did in Smartauth [15]. However, one challenge is that key elements (*e.g.*, user behaviors) sometimes are not included in descriptions [40]. Indeed, there can be no app description. Another challenge is that descriptions only contain fuzzy device information, which does not include specific devices that users operate. For example, the *"open/close sensor opens"* phrase does not provide enough information about what device is actually opened. To overcome the above challenges, we find that input prompts can be utilized to extract additional information. As illustrated in Figure 5, we use the example in Listing 1 to address the main steps of the NLP analysis.

Step 1: Generate subject-verb-object (SVO) phrases. App descriptions can be summarized as SVO phrases, in which the subject or object part can be absent, due to their trigger-action nature. We utilize SVO phrases as the source of coarse-grained sensitive information. ALTA first conducts part-of-speech (POS) tagging to descriptions. Then, ALTA generates type dependencies of entities to mark relations between words, which are used to generate SVO phrases. Besides, ALTA supports passive voices, noun phrases, and coordinating conjunctions, which are features in the grammar. In Figure 5, three phrases are generated at this step.

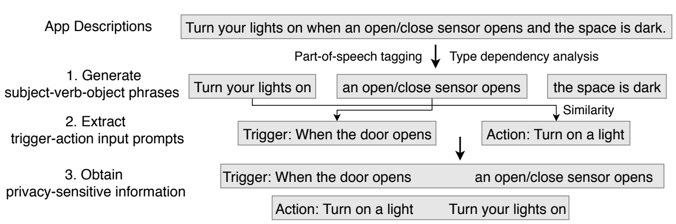


Fig. 5: An example of NLP analysis.

Step 2: Extract trigger-action input prompts. Input prompts are prompt statements in the UI of an app, which are written in the source code file of apps. There are no layout files to record input prompts. Input prompts remind users to select devices and parameters. For example, lines 6, 7, and lines 10, 11 in the Listing 1 are two different input prompts. Input prompts are identified via *section* node in the AST by the program analysis. Also, we have identified trigger-action pairs in the program analysis. Through parsing the input statements, we extract corresponding trigger-action descriptions. Input prompts are fine-grained supplementary information to SVO phrases.

Step 3: Obtain privacy-sensitive information. We calculate semantic similarities between phrases in descriptions and input prompts to correctly match the phrase pair. For instance, *"an open/close sensor opens"* in the description is closer to *"When the door opens"* (the trigger phrase describes the trigger device) than other phrases, so we classify it as a trigger phrase too. We obtain all trigger and action phrases, which denotes the functionalities of apps. So *"the space is dark"* will be discarded. Missing information in the description (*i.e.*, what object is opened) is extracted from input prompts. When descriptions are missing, input prompts will be used directly. For apps with the same trigger-action pair, functionalities can be different, which pinpoints sensitive user activities and device status. For example, for the same trigger-action pair *acceleration-switch*, *Someone is knocking door* turns lights on while *Open garage door when I arrive* opens the garage door.

Implementation. We utilize spaCy [41] to conduct the POS tagging and type dependency analysis. An example dependency tree of an app description is shown in Figure 6. The English multi-task CNN language model (we choose the en_core_web_lg) is used, which was trained on OntoNotes 5 dataset [42]. For SVO detection, we first collect all verbs. Then, for each verb, we find its corresponding subject. Note that passive (*e.g.*, nsubjpass), clausal (*e.g.*, csubj), and conjunction (*e.g.*, and) relations are also considered to find subjects. Similarly, objects of verbs are identified, which also include negative forms. Subjects and objects are identified by typical POS tags (*e.g.*, nsubj, dobj). The similarity calculation is achieved through *Token.similarity* API. The underlying technique is comparing word vectors, which are generated by the Word2vec [43] model.

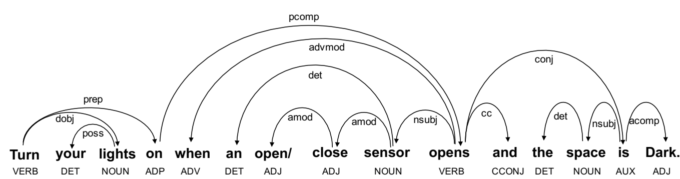


Fig. 6: An example dependency tree of an app description.

C. TRAFFIC ANALYSIS

In this section, we describe how we extract traffic fingerprints of events and then present how we extract traffic fingerprints of apps.

Learning traffic fingerprints of events. We first define the key features of a packet in traffic. The packet length and direction are essential to fingerprint an event [29], [32]. We define a packet as $p = (d, l, t)$, where d represents the direction of a packet (either from the hub to cloud or from the cloud to hub); l represents the packet length and t stands for the timestamp of a packet. The packet sequence of an event can be defined as $S(d_h, d_c) = (p_1, p_2, ..., p_z)$, where $(p_1, p_2, ..., p_z)$ denotes that z packets are generated in the time duration $(t_{p_z} - t_{p_1})$.

We identify the most frequently appeared sequence of packets to generate fingerprints of events. Specifically, we trigger each event m times and collect a sequence of packets S_i each time. For each sequence S_i, we calculate the sum of the distance between this sequence to other sequences $\sum_{j=1}^{m} dist(S_i, S_j)$. $dist()$ denotes the distance to measure the similarity between S_i and S_j. As proved effective in [32], we use the Levenshtein distance [44], where a small value

TABLE I: Examples of fingerprints of events. The number is packet length. ↑ means packets from hub to cloud. ↓ means packets from cloud to hub.

Event	Device name	Fingerprint
motion.active motion.inactive	Samsung SmartThings Motion Sensor (ZigBee)	420↑ 113↓ 420↑ 113↓
switch.on switch.off	Samsung SmartThings Outlet (ZigBee)	145↓ 113↑ 418↑ 113↓ 145↓ 113↑ 418↑ 113↓
contact.open contact.closed acceleration.active acceleration.inactive	Samsung SmartThings Multipurpose Sensor (version 2015, ZigBee)	741↑ 113↓ 113↓ 741↑ 113↓ 113↓ 756↑ 113↓ 113↓ 426↑ 113↓ 424↑ 113↓
illuminance.value	Aeotec MultiSensor 6 (Z-Wave)	118↑ 113↓

indicates a high similarity. Finally, for m sum results, we choose the smallest one and its sequence S_E as the fingerprint of this event, which means S_E has the largest similarity with the other sequences. We choose $m = 60$ in our experiment. Table I illustrates the fingerprints of nine events in packet size and direction.

Extracting traffic fingerprints. Once it learns traffic fingerprints of all events, ALTA extracts trigger-action pairs and *if-conditions* from traffic as fingerprints of apps. Given that fingerprints of events can be the same, one challenge is how to identify accurate values of trigger events (*e.g.*, *open* or *closed*). Since *if-conditions* are also events in traffic, another challenge is how to distinguish events that represent *if-conditions* from other irrelevant events. For example, how to decide a *motion.active* is checked in an *if-condition*. We use the pair-occurrence of events and app correlation features to resolve the above challenges.

Step 1: Extract all events. ALTA first splits traffic data into bursts. A *burst* is a sequence of packets generated within a certain of time (*i.e.*, the burst threshold [45]). For each burst, ALTA calculates the distance $l_{E_i} = dist(S_b, S_{E_i})$ between this burst b and the fingerprint of event E_i. If the smallest l_{E_i} is below a pre-defined threshold, we identify the event E_i from the traffic.

Step 2: Extract trigger-action dependencies. Trigger-action (E_T-E_A) pairs occur in sequential order. We observe and verify that most trigger-action pairs happen within a short period of time except the delay condition. The fingerprint of a trigger-action pair consists of the fingerprint of a trigger event and an action command. We calculate the distance between a burst and the fingerprint of a trigger-action pair. If the smallest distance is below a pre-defined threshold, we obtain a trigger-action pair in traffic. However, the challenge is that multiple events may have the same fingerprint (*e.g.*, *motion.active* and *inactive*). It is difficult to pinpoint the trigger event. We use the pair-occurrence of events to differentiate. For example, if we first identify a *motion-switch* trigger-action pair, and then another *motion* event, but both *motion* events cannot be pinpointed as *active* or *inactive*. Through the pair-occurrence feature, we can know it is *motion.active-switch* and then *motion.inactive* since the *motion* event order must be first *active* and then *inactive* and the interval between *active* and *inactive* is fixed.

Step 3: Extract if-conditions. Extracting *if-conditions* (E_{IF}) are more difficult than the trigger-action since they do not have two events in a sequential order, which is hidden in a rather long packet sequence. ALTA leverages the time interval and app correlation to address this challenge. For delay $\mathcal{D}$ in *if-conditions*, there is a time interval between trigger and action. To detect $\mathcal{D}$, we first identify triggers and then search actions in a commonly used time interval. We choose 30s, 60s, and 90s as a searching distance for the sake of efficiency (*e.g.*, turn off lights after users leave the house). To identify $\mathcal{DS}$ and $\mathcal{P}$ in *if-conditions*, we use two features of app correlation: 1) The action of app A acts as the trigger of app B; and 2) *if-conditions* $\mathcal{DS}$ and $\mathcal{P}$ in app A are the trigger or the action of app B. For 1), the trigger-action pair of A and the action of B happens in sequential order within a short time. For 2), the trigger-action of B happens before the trigger-action of A. This co-occurrence pattern will be observed whenever these two apps are triggered, which also presents a sequential order. Such features, which generate a fixed pattern in the traffic, can be used as traffic fingerprints of apps.

D. Fingerprint Matching

In this section, we illustrate how ALTA infers running apps through fingerprint matching. ALTA compares fingerprints of apps that generated through app analysis against fingerprints (*i.e.*, triggers, actions, and *if-conditions*) identified from eavesdropping traffic. The workflow consists of three tasks:

- Task 1: Identify all events from eavesdropping traffic.
- Task 2: Scan the event sequence and extract traffic fingerprints through traffic analysis.
- Task 3: Search app fingerprints to match with traffic fingerprints.

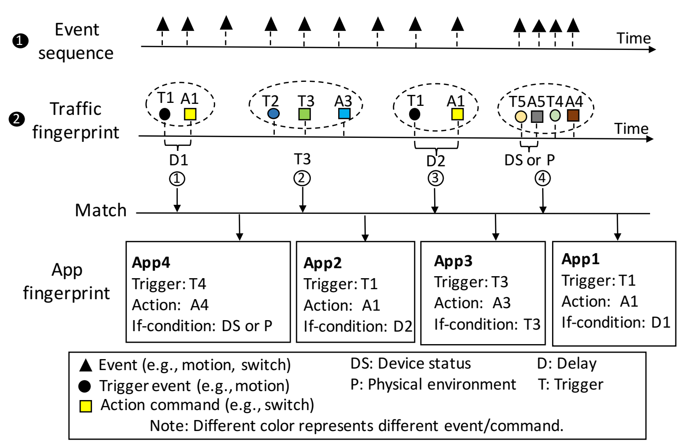


Fig. 7: An illustration of app-traffic fingerprint matching.

As illustrated in Figure 7, we use four apps that are different in trigger-action pairs and *if-conditions* to demonstrate the workflow. *Task 1*: The event sequence is extracted from raw traffic through fingerprints of events. Line ❶ shows all identified events. *Task 2*: A trigger-action pair is easily detected due to the sequential happening order and the short interval (or fixed interval in the delay scenario) as described in step 2 of extracting traffic fingerprints in Section IV-C. Line ❷ shows that all trigger-action pairs are identified (colored circles and squares, *e.g.*, T4-A4). This means we find all *possible* but *not deterministic* traffic packet sequences that can be generated by a specific app. For example, App1 can be linked to traffic packet sequences ① and ③ in dotted circles. App4 can be linked to sequence ④. *Task 3*: To narrow down and determine packet sequences generated by a specific app, through step 3 of extracting traffic fingerprints in Section IV-C, ALTA searches the packet sequence of *if-conditions* in the linked packet sequences. For example, in sequence ④, ALTA finds that T5-A5 indicates the DS or P *if-condition* in App4, which helps us to ascertain that sequence ④ is definitely generated by App4. Also, after we compare the delay *if-condition*, we find that the elapsed time of App1 is D1, so we conclude that sequence ① rather than sequence ③ is generated by App1. Irrelative events are discarded. The same process can be applied to App2 and App3. Thus, ALTA identifies running apps from traffic. Formally, we describe this process in Algorithm 1.

Note that we do not aim to pinpoint a running app just by one occurrence of a fingerprint packet sequence. By counting the occurrence of the fingerprint within a certain period, we empirically set a threshold to mitigate the influence of noise and network latency.

Algorithm 1: Algorithm for App-Traffic Fingerprint Matching

Input : AF, sets of app fingerprints
TF, sets of traffic fingerprints
Output: APPS, sets of discovered apps

/* First compare the trigger-action pair (*i.e.*, $T-A$). */
foreach $(T_i - A_i) \in TF$ **do**
  foreach $(T_j - A_j) \in AF$ **do**
    if $(T_i - A_i) == (T_j - A_j)$ **then**
      /* Then check *if-conditions* values */
      if $AF_j.T == TF_i.T \& AF_j.P == TF_i.P \& AF_j.DS == TF_i.DS \& AF_j.D == TF_i.D$ **then**
        /* Add an app to discovered apps sets */
        APPS $\leftarrow \{APP_j\}$

V. EVALUATION

We develop a prototype with the off-the-shelf hardware as listed in Table I: the SmartThings Home Monitoring Kit (which contains a hub, two multipurpose sensors, a motion sensor, and an outlet) and the Aeotec MultiSensor 6 as operating devices. We collect 254 apps, 185 from the SmartThings Public Github Repository [39] and 69 third-party apps from IoTBench repository [46]. Our experiments aim to answer the following questions:

- What are the distributions of trigger-action and *if-conditions* in apps (and thus they are potentially vulnerable to ALTA anaysis)? (Section V-A)
- How effective is ALTA in extracting fingerprints and functionalities of apps through program analysis and NLP analysis? (Section V-B)
- How effective is ALTA in learning which app is running from IoT traffic (*i.e.*, the effectiveness of launching ALTA privacy leakage analysis)? (Section V-C)

A. Potentially vulnerable apps

Numbers of apps in each trigger-action pair. We report three main findings when investigating apps in each trigger-action pair: 1) 113 out of 254 (44.5%) apps contain trigger-action dependencies; 2) there can be multiple apps with the same trigger-action; 3) ALTA identifies 61 different pairs of trigger-action and 43 of them only contain one app. Table II shows the numbers of apps in the top five trigger-action pairs. For example, there are 12 apps in *contact-switch*: 5 in *contact.open*, 1 in *contact.closed*, and 6 apps do not specify *open* or *closed*. For one trigger-action pair, if there is only one app containing this trigger-action, we can identify this app by trigger-action without using *if-conditions*. Totally, we identify 43 such apps (examples are shown in Table III).

TABLE II: Numbers of apps in the top five trigger-action pairs. Apps are counted by trigger event.

Trigger-action	Contact-switch				Motion-switch			
Trigger event	open	closed	both	total	active	inactive	both	total
#Apps	5	1	6	12	2	1	6	9
Trigger-action	Contact-lock				Temperature-switch		Humidity-switch	
Trigger event	open	closed	both	total	numerical value		numerical value	
#Apps	2	2	1	5	4		4	
Total	34							

TABLE III: Representative trigger-action pairs with only one app. The total number of such trigger-action pair is 43.

Trigger-action pair	Trigger event
carbonMonoxide-lock	clear/detected/tested
contact-alarm	open
water-switch	wet/dry
acceleration-colorControl	active
smoke-lock	clear/detected/tested
water-valve	wet
temperature-thermostat	numerical value
contact-thermostat	open/closed
smoke-thermostat	clear/detected/tested
illuminance-switch	numerical value

Categories of apps with the same trigger-action. We group apps with the same trigger-action pair into different categories by their functionalities. Different triggers or actions represent different functionalities. This helps us to understand the usage of apps and identify apps more accurately. We find that: 1) Apps with the same trigger-action can be totally different in functionalities. For example, apps in the *humidity-switch* category contain completely different privacy-sensitive information. 2) And there exists a prevalent usage for apps within the same trigger-action. For example, most apps in the *motion-switch* category first detect movements of users and then turn lights on/off. This demonstrates the need for app-level traffic analysis if an attacker wants to derive context-rich sensitive information from IoT traffic. We summarize categories of apps in the top five trigger-action by functionalities in Table IV.

If-conditions summary. We find 21 different types of *if-conditions*, *e.g.*, *lock*, *smoke*, and *acceleration*. In Table V, the

TABLE IV: Top five trigger-action categories.

Trigger-action	Category	#Apps	Example
contact-switch	Trigger: When the door opens Action: Turn on a light	10	Brighten Dark Places Hall Light: Welcome Home
	Trigger: Sensors detecting an intruder Action: Send an alarm	1	Smart Security
	Trigger: Open the window or door Action: Turn on a house fan	1	Whole House Fan
motion-switch	Trigger: When there's movement Action: Turn on/off light(s)	8	Light Follows Me Turn Off With Motion
	Trigger: When there's been movement Action: Turn on heater or air conditioner	1	Virtual Thermostat
contact-lock	Trigger: The door contact sensor is open Action: Operate the door lock	4	Smart Auto Lock / Unlock Enhanced Auto Lock Door
	Trigger: Activate the alarm system Action: Lock these locks	1	ecobeeAwayFromHome
temperature-switch	Trigger: Monitor the temperature Action: Turn on A/C or fan	4	It's Too Hot It's Too Cold
humidity-switch	Trigger: Take a shower Action: Turn on Coffee maker	1	Coffee After Shower
	Trigger: Monitor the humidity Action: Control humidifier	1	Smart Humidifier
	Trigger: Monitor the humidity Action: Control the vent fans	1	Auto Humidity Vent
	Trigger: Monitor the humidity Action: Control the switch	1	Humidity Alert!

An app may contain multiple trigger-action pairs, thus counted in different trigger-action pairs.

TABLE V: Numbers of apps with *if-conditions*. Features of *if-conditions* are close related to sensitive information, *e.g.*, *smoke*, *acceleration*, *water*.

Trigger ($\mathcal{T}$)				
Feature	value	#Apps	value	#Apps
smoke	detected	1	clear	1
contact	closed	13	open	14
switch	off	6	on	12
presence	not present	4	present	19
thermostatMode	cooling	1	heating	1
water	wet	3	dry	2
motion	active	29	inactive	3
lock	locked	4	unlocked	5
Device status ($\mathcal{DS}$)				
Feature	value	#Apps	value	#Apps
switch	on	26	off	24
thermostatMode	cool	13	heat	10
contact	open	13	closed	12
lock	locked	5	unlocked	2
alarm	off/siren/strobe	4	N/A	N/A
acceleration	active/inactive	2	N/A	N/A
Physical Environmennt ($\mathcal{P}$)				
Feature	#Apps	Feature	#Apps	
Time	45	Location.mode	12	
Temperature	22	Humidity	4	
Time(Sunset/Sunrise)	6/3	Luminance	4	
Delay ($\mathcal{D}$)				
Feature	#Apps			
elapsed	22			

top three triggers ($\mathcal{T}$) are *motion* (32), *contact* (27), and *presence* (23). Meanwhile, 96 apps depend on physical parameters ($\mathcal{P}$) such as *temperature*, *humidity*, and *luminance*, which may form connections among apps through physical channels [47]. Moreover, 111 apps check the device status ($\mathcal{DS}$) before executing commands. At last, the elapsed time ($\mathcal{D}$) appears in 22 apps, which is used for user actions or devices to run for some time. These *if-conditions* represent the context of apps (*e.g.*, *smoke*, *acceleration*, and *presence*), which means apps with these *if-conditions* may contain sensitive information. Also, a large number of *if-conditions* is checked in apps, which indicates that *if-conditions* can be a useful and feasible factor to differentiate apps. Note that an app can contain multiple *if-conditions*, so the number of *if-conditions* here exceeds the total number of apps.

B. *Effectiveness of app analysis*

Trigger-action and if-conditions extraction. We first randomly choose 100 apps with trigger-action dependencies. Then we manually analyze trigger events and corresponding action commands as the ground truth. We check the results of our program analysis against those of the manual analysis. We also record all possible execution paths from trigger events to action commands. Note that a trigger event can trigger multiple action commands and multiple trigger events may also only trigger one action command. We classify events with different capabilities as different trigger events (*e.g.*, *motion* and *contact* are different events) and statements of action in different line numbers as different action commands (*e.g.*, *switch.on* in different places in apps). In this process, we find that our method incurs no false positive and three false negatives (1.8%) in finding 163 trigger events, and no false positive but two false negatives (0.6%) in finding 318 action commands. For three false negatives, three apps use non-standard *subscribe* statements to specify trigger devices and handler functions, which is hard to extract trigger devices. For example, *subscribe (lock1, "lock", doorHandler, [filterEvents: false])* adds the *filterEvents* feature to receive non-state change events (*e.g.*, repeated *lock.locked* events). For the two false negatives in finding action commands, *SmartSecurityLight* uses the schedule feature (the non-standard *runIn* function) to run *light.off()* command [48]. In the other case, *WindowOrDoorOpen* replaces device names with *it.off()*, which is hard to find action device names in the execution path. Apps using the standard *filterEvents* feature and *runIn* function can be identified by adding new rules to parse *subscribe* statements. However, non-standard *subscribe* statements are difficult to parse, where rule-based policies may fail. For the *if-conditions* extraction, we manually check 316 paths identified from 100 apps and *if-conditions* in each path. We confirm that all *if-conditions* (100%) can be correctly extracted between trigger events and action commands.

NLP analysis. We measure the effectiveness of ALTA to summarize functionalities of apps from *descriptions* and *input prompts*. This evaluates how effective ALTA is to extract the key trigger and action information that may potentially contain user privacy. We randomly choose 50 apps from our dataset and manually extract triggers and actions to construct functionalities of apps as the ground truth. Then we manually compare functionalities generated by ALTA to the ground truth. We evaluate the functionality extraction process using app descriptions, input prompts, and the combination of both.

To achieve better performance, we use the English multi-task CNN (*en_core_web_lg*) model, which is more accurate but larger than the default *en_core_web_sm* model. ALTA achieves 86% accuracy using the combination of app descriptions and input prompts, where information in descriptions and input prompts are complementary to each other. When trigger or action information in app descriptions is missing, we use input prompts directly. This step improves the accuracy. As illustrated in Figure 8, ALTA obtains incorrect triggers or/and actions extractions in 50% apps when using only app descriptions. We identify three possible reasons that can cause

NLP analysis to fail when using only descriptions: 1) *Trigger or action missing.* Either trigger or action information is not recorded in the description sentence. For example, the description *"Control your Sonos system"* only indicates the action operation. 2) *Entity missing*. Without a device name, the description does not include any entity that represents trigger or action. For instance, *"Turn something on when you arrive"* does not specify which device to control when the user arrives at home. 3) *Complex logic.* Some apps provide rich functionalities for users to use, which include many procedures and scenarios. Such complex logic makes descriptions of apps difficult for the NLP analysis tool spaCy [41] to analyze. We also find ALTA failed in 22% apps when only using input prompts. The main reason is entity missing when developers describe app usages in input prompts.

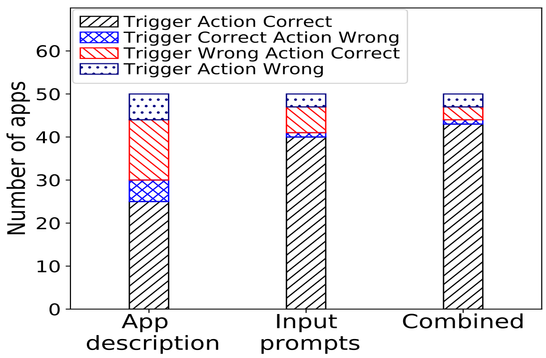


Fig. 8: Accuracy of NLP analysis.

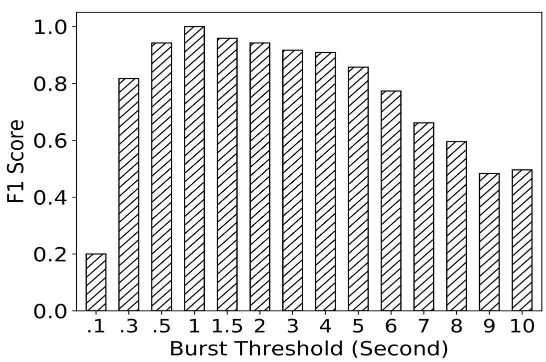


Fig. 9: F_1 scores of event-inference with different burst thresholds.

C. Effectiveness of app inference

We obtain and use all 31 apps that the current IoT devices listed in Table I can support, which contain 21 apps in the top two trigger-action pairs (*i.e.*, *motion-switch* and *contact-switch*) and 10 apps in other categories (*e.g.*, *temperature* and *humidity*). ALTA obtains 10 apps that contain fingerprints (containing both trigger-action and *if-conditions*) out of 31 apps. To measure the upper and lower bound of app inference performance in ALTA, we conduct experiments under two scenarios: 1) identifying apps with fingerprints out of a known app set (*i.e.*, a closed-world scenario). This experiment evaluates the interference between multiple apps with fingerprints, which aims to validate if ALTA can infer the correct app when there are multiple apps with fingerprints. 2) And identifying apps with fingerprints out of 254 possible apps (*i.e.*, an open-world scenario). This experiment evaluates the influence of normal apps (those without fingerprints) to the app inference, which aims to measure if ALTA can distinguish apps with fingerprints from apps without fingerprints. For each experiment, we trigger apps for 30 times during 30-40 minutes (the delay *if-condition* demands more time). We first present the experiment of how we determine the burst threshold.

Burst threshold. The burst threshold determines the sequence of packets of events. If the burst threshold is either too large or small, ALTA cannot identify trigger events, action commands, and *if-conditions* from traffic correctly. We manually operate devices in the Home Monitoring Kit and the Aeotec MultiSensor 6 each for 60 times. We set the burst threshold from 0 to 10 seconds to select the proper value. We define the precision as the correctly identified events over all identified events. And the recall as the correctly identified events over all manually operated events. The F_1 score is 2*((precision*recall)/(precision+recall)). We calculate the F_1 score to seek a balance between precision and recall. Figure 9 shows F_1 scores of event-inference. We can see that the best performance burst threshold is 1 second. The reason is that if the burst threshold is too small, it does not include the complete sequence of packets of events. If it is too large, it includes too much irrelevant packets.

Inference under a known app set. For 10 apps with fingerprints, we randomly choose X apps from 10 apps (X ranges from 1 to 9). Then, we run them at the same time and manually trigger each app 30 times according to the usage of each app. For each X setting, we repeat the random selection and detection 60 times. As illustrated in Table VI, the line Y=0 (Y denotes apps without fingerprints) indicates the result of the closed-world scenario experiment. Here a true positive (TP) is defined as a randomly selected app being predicted by ALTA; a false positive (FP) is defined as a predicted app being not selected; a false negative (FN) is defined as a selected app being not predicted. The precision is defined as TP/(TP+FP) and the recall is defined as TP/(TP+FN). From the result, we can see that the growth of apps slightly impacts the detection rate, which causes some FN cases. The main reason is that the burst and event detection is disturbed. Overall, the F_1 score is above 97%.

Inference under an open app set. In this experiment, we randomly select Y apps (Y ranges from 3 to 20) without fingerprints from 21 apps and X apps with fingerprints from 10 apps. We repeat steps in the previous section and run X+Y apps at the same time. We aim to evaluate if ALTA can detect X apps from X+Y apps. ALTA has to search fingerprints of 254 apps to detect X apps, which acts as the open-world scenario. From Table VI, we find that, for a certain X, when Y grows from 3 to 20, the FN rate increases. After examing the results, we find that *if-conditions* interfere with events of apps without fingerprints, which causes FN cases. Overall, all F_1 scores are above 91%.

TABLE VI: Results of app inference under the known app set (F_1 over 97%) and open app set (F_1 over 91%).

#Apps-without-fingerprints	#Apps-with-fingerprints				
	X=1	X=3	X=5	X=7	X=9
Y=0	1.00/1.00/1.00	1.00/0.98/0.99	1.00/0.95/0.97	1.00/0.96/0.98	1.00/0.95/0.97
Y=3	1.00/0.98/0.99	1.00/0.92/0.95	1.00/0.93/0.96	0.99/0.92/0.96	1.00/0.90/0.95
Y=5	1.00/0.94/0.97	1.00/0.90/0.94	1.00/0.87/0.93	1.00/0.86/0.92	1.00/0.88/0.93
Y=7	1.00/0.92/0.96	1.00/0.90/0.94	0.99/0.84/0.91	1.00/0.86/0.92	1.00/0.85/0.92
Y=10	1.00/0.90/0.95	1.00/0.89/0.94	1.00/0.86/0.92	0.99/0.86/0.92	1.00/0.85/0.92
Y=15	1.00/0.90/0.95	1.00/0.87/0.93	1.00/0.85/0.92	0.99/0.85/0.91	1.00/0.85/0.92
Y=20	1.00/0.88/0.94	1.00/0.86/0.92	0.99/0.85/0.91	0.99/0.84/0.91	0.99/0.84/0.91

*The results are in precision/recall/F_1 format.

1) Case studies.: We present case studies to show how we achieve the app-level traffic analysis through characterizing four factors of *if-conditions*. (1) **App correlation.** The app *OpenGarageOpenDoorwhenIarrive* opens the garage door once detecting the user opening the car door. This will trigger *Let There Be Dark* to turn off lights when it finds the garage door is opened. Such correlation patterns make two apps generate trigger-action traffic in a sequential order, which is a unique fingerprint. (2) **Device status.** *Smart Security* sends an alarm alert when the intrusion detection contact sensor is triggered. However, it also inspects the *motion* event to check the status of home residents. The *contact-switch* and *motion.active* co-occurrence pattern exhibits a unique fingerprint. (3) **Delay.** *Light Follows Me* turns on lights when there is a *motion.active* event. Users usually set a specific elapsed time since *motion.inactive*, after which the app will turn off lights. Due to different functionalities, time intervals of apps are usually different. Such time intervals between triggers and actions also make apps generate unique traffic. (4) **Physical environment.** *Smart Nightlight* turns on lights when users generate *motion.active* events. But it also checks whether the illuminance is below a threshold to save power. Meanwhile, *Light Up the Night* turns on lights when the illuminance is below a threshold. The physical environment dependency makes two apps generate traffic in a certain sequence, and leads to a unique fingerprint.

We summarize prevalent privacy information that can be extracted from apps. 1) User activities. This information includes behaviors when users are at home, *e.g.*, users' movement, the use of devices. 2) Daily routine. Users may work on a schedule. For example, users can wake up and go to work at a fixed time. 3) Health conditions. For instance, pill reminder and blood pressure monitor apps may leak health conditions. 4) Hobbies and interests. Some apps can indicate certain interests of users, *e.g.*, pet feeding app, and music playing app.

VI. Discussion

Countermeasures. To mitigate privacy risks posed by traffic analysis, we discuss possible countermeasures from two aspects: 1) IoT and cloud platform-level methods; and 2) new design and implementation of network transmission methods.

Proxies with traffic shaping. Intuitively, a straightforward solution is to use proxies to inject random noise to eliminate statistical and temporal patterns between the hub and cloud: 1) *Random padding.* The noise aims to break the size and temporal relation of packets in traffic sequences, which defeats statistical and learning-based analysis methods. 2) *Fake events.* To confuse attackers, fake events can be randomly injected into normal traffic, which will lead attackers to deduce wrong results of users' activities and sensitive information. The hub and cloud could design a protocol to recover original packets. This method would let both wireless and man-in-the-middle eavesdropping attackers fail to learn the traffic patterns. For the random padding, it is hard for the traffic analysis of ALTA to identify correct events from encrypted traffic packets. Thus, the traffic fingerprint generation process may fail. For the fake events, ALTA may extract wrong traffic fingerprints or cannot extract fingerprints at all. Thus, the fingerprint matching process cannot identify the correct app.

Move computing to edge devices. IoT platforms (*e.g.*, SmartThings) could delegate privacy-sensitive operations to edge devices, which have a medium level of computing ability. All installed apps could be sent from the cloud backend to the edge device. Then, during the runtime, devices could communicate with the edge device directly to avoid traffic leakage to man-in-the-middle attackers (*e.g.*, ISPs). Also, IoT platforms could offer a dedicated device that placed locally to deal with control logic to reduce communicating traffic. The wireless eavesdropping attackers still have the chance to obtain traffic under this protection method. Without the bidirectional traffic between the IoT hub and cloud backend, ALTA cannot obtain the needed traffic. The local communication between devices and the IoT hub may still leak events. However, it demands a strong threat model, which requires attackers to physically appear near users.

VPN or anonymity network. VPN methods hide real traffic by building a tunnel between the source and the destination side. Anonymity network (*e.g.*, Tor) methods create multiple transferring bridges and conceal the real source of traffic. Both methods make attackers hard to learn side-channel information. In the absence of side-channel information, attackers can not learn traffic patterns. The time delay is expected for two methods so they could be used in non-time-critical scenarios. VPN-based protection methods may break patterns needed to learn events from encrypted traffic. Thus ALTA cannot conduct traffic analysis. Tor-based methods cause ALTA hard to find the identity of encrypted traffic. Even apps are recognized, ALTA does not know which user is using these apps.

Generality. Although ALTA was developed on the SmartThings platform, the techniques can be applied to other smart home systems too. Google Home provides a hub which can be launched through voice commands. The cloud backend processes these requests and sends back action control signals to home devices. Apps are running on the cloud. Apple's

HomeKit offers a hub to connect devices. Although apps are running on the hub, there is traffic data between devices and the hub. The traffic can also leak events and apps can be identified. Other smart home IoT platforms adopt similar architectures. So we believe that our approach achieves a good generality.

One limitation of the generality is that ALTA mainly focuses on open-source apps. However, we find a possible method to deal with closed-source apps. Specifically, we find that the user interface (UI) of some apps (SmartApps) on the mobile app of SmartThings will ask users to set values for specific *if-conditions*. For example, some apps ask users to set the time interval between the trigger event and the action event. Thus, the time delay *if-condition* can be extracted. We find that the physical environment and device status *if-conditions* can also be set on the UI. In the future, we will explore the method to analyze the UI of apps on the mobile app of SmartThings and detect closed-source apps.

Broken app descriptions and input prompts. App descriptions can be broken or not clear. As illustrated in our NLP analysis (Section V-B), we combine input prompts to improve the performance of extracting sensitive information. But if input prompts are even broken, ALTA cannot obtain useful information. However, we argue that developers are willing to provide complete and clear descriptions since they want to attract users. Otherwise, users may not use their apps.

Traffic interference. There can be interference traffic other than IoT traffic. For example, users may watch online TV, which will generate a large volume of data. However, we can use an IP filter to obtain IoT traffic and drop other traffic. Also, for the filtered traffic, a new burst threshold can be identified. Also, a dynamic threshold could be adopted according to different volume of traffic data.

App interference. There are two particular scenarios that are hard to detect apps: (1) one trigger sensor operates multiple action devices; (2) multiple trigger sensors operate one action device. For scenario (1), if action devices are different, apps can be differentiated by action events. However, if the types of action devices are the same, these apps can be differentiated by *if-conditions*. Indeed, *if-conditions* events will form a unique event sequence together with trigger-action events. The same situation applies to scenario (2).

However, for scenario (1), if action devices are triggered at the same time, we find it hard to differentiate apps. All events will interfere with each other. For scenario (2), we argue that multiple triggers will seldom operate one action device at the same time. Because if action commands are the same, it is unnecessary to use different triggers simultaneously. If commands are different, this will generate conflict actions. If apps with delay conditions end at the same time, ALTA cannot differentiate them. Also, if multiple events happen within a burst threshold (caused by network communication latency or Hub processing latency), ALTA cannot identify correct events.

Limitations. For apps without using *if-conditions* in execution paths, ALTA is not able to extract enough factors to build fingerprints. Also, apps in the same trigger-action category may contain similar *if-conditions*, which are difficult to distinguish. Some delay time in apps does not set commonly used time intervals, which makes ALTA hard to identify delay conditions if it is the only side information. Meanwhile, static analysis has an inherent limitation in dealing with dynamic method calls. As results shown in Section V-B, we find dynamic method invocation rarely used in apps. Careless developers could write descriptions and input prompts in poor structures, and malicious attackers even could spread out misleading descriptions. However, the SmartThings team has an approval process when publishing apps [49], which makes this attack hard to conduct.

VII. RELATED WORK

In this section, we discuss related works on IoT security, privacy and safety mainly based on the SmartThings platform.

Comparison of IoT traffic analysis work. Apthorpe *et al.* [28], [31] tested seven commercially-available IoT devices and demonstrated that simple traffic features (such as MAC addresses, DNS queries, and traffic rates) can distinguish these IoT devices. Yoshigoe *et al.* [29] observed that SmartThings' SmartHub and its cloud server adopt a simple send-response communication model. They have shown that network traffic from different devices is distinguishable by packet frequency and length. Peek-a-Boo [30] is a machine-learning based multi-stage privacy attack in IoT. The attack identifies particular types of IoT devices, their actions, states, and ongoing user activities by observing the wireless traffic from smart home devices. The limitations of the above three work can be summarized as 1) the event or simple activity level of privacy leakage analysis underestimates the risks and loss to users. As illustrated in Section III, context-rich information causes more serious damage (*e.g.*, pilferage and blackmail). And 2) massive traffic is needed to deduce activities. This increases the attackers' chances of exposure. HoMonit [32] utilizes apps to conduct anomaly detection, which forms the Deterministic Finite Automaton (DFA) to characterize the running logic of apps. However, it requires prior knowledge of running apps, which impedes detecting apps that we do not know in advance from raw traffic. Thus it also cannot differentiate apps with the same trigger-action pair. Also, it is designed to conduct anomaly detection, so it does not extract privacy-sensitive information from apps. We summarize the differences between [28]–[30], [32] and ours in Table VII. Our work advances the current event-level traffic analysis attacks by deducing activities from *multiple* apps via leveraging trigger-action and *if-conditions*. There are also other traffic analysis work. PINGPONG [35] automatically generates packet-level signatures for smart home devices based on packet lengths and directions. Alshehri *et al.* [33] utilize signatures to identify smart home devices from tunneled traffic. Dong *et al.* [50] proposed an LSTM-based neural network to identify smart home devices under a complex network environment. Yang *et al.* [51] developed two designs of Onion IoT gateways to protect potentially vulnerable IoT devices by hiding them behind IoT gateways running the Tor hidden services. IOT-FUZZER [52] utilizes the official mobile app to detect memory corruptions of the corresponding IoT device, which adopts a method of runtime mutation of protocol fields.

TABLE VII: A comparison of existing IoT traffic analysis attacks.

Name	Purpose	Sensitive Information source	Black-box	Ambiguity-prone sensitive information	Traffic-hungry	Context-rich sensitive information
Apthorpe *et al.* [28], [31]	Infer privacy-sensitive information	Device	✓	Affected	Medium	✗
Yoshigoe *et al.* [29]	Mitigate privacy leakage risks	Device	✓	Affected	Medium	✗
Peek-a-boo [30]	Infer privacy-sensitive information	Device	✓	Affected	High	✗
PINGPONG [35]	Detect smart home devices	Device	✓	Affected	Medium	✗
Dong *et al.* [50]	Detect smart home devices	Device	✓	Affected	High	✗
HoMonit [32]	Anomaly detection	-	✗	-	Low	-
ALTA (Ours)	Infer privacy-sensitive information	App	✓	Not affected	Low	✓

Static Analysis based Approach. Many research efforts have been undertaken by applying static analysis techniques in identifying and improving IoT security and safety. Fernandes *et al.* [8] identified several security-critical design flaws (*e.g.*, event leakage and event spoofing) in SmartThings. SAINT [9] tracks information flows from sensitive sources to external sinks to find sensitive data flows. iRuler [53] studied the security risks of interactions between trigger and action events in IFTTT. The authors used NLP analysis to check whether one app can trigger another app. IoTMon [47] discovers possible physical interaction chains across applications and assesses the safety risk of each discovered cross-app interaction in the IoT environment. Both SOTERIA [10] IotSan [11] apply model checking to verify user-defined safety, security, and functional properties. Different from SOTERIA, IotSan focuses on revealing flaws in the interactions between sensors, apps, and actuators, *e.g.*, verifying conflicting and repeated commands from different apps.

Runtime Enforcement based Approaches. This includes enforcing access control and monitoring IoT app behaviors at runtime. He *et al.* [12] conducted a 425-participant user study about access control and authentication for the home IoT. The authors found that the fine-grained access control (*e.g.*, per-action based, rather than on a per-device granularity) is needed. ContexIoT [13] is a context-aware permission scheme, which checks context information when performing a security-sensitive action. It is achieved by inserting the constraint code in the app source code. To address the potential permission abuse and data leakage issues, FlowFence [14] enforces developer-specified data flow patterns, while blocking all other undeclared flows. SmartAuth [15] ensures the app's runtime behavior is consistent with its description in code, which mitigates the security risks of overprivileged IoT apps. Wang *et al.* [16] proposed the ProvThings, a platform-centric monitoring framework for provenance-based tracing in IoT. HomeSnitch [54] monitors semantic behaviors between clients and servers via inspecting network traffic of devices, which is used to enforce transparency and control in a smart home environment. SCADMAN [55] enforces the correctness of controllers to conduct intrusion detection and sensor fault detection by maintaining the state estimation of the system. Song *et al.* [56] propose a communication protocol to encrypt the transmitted data and utilize MAC to ensure the integrity and authenticity of the data.

VIII. CONCLUSION

In this work, we presented ALTA, a new IoT privacy leakage analysis that provides a finer-granularity to deduce context-rich privacy-sensitive information (*e.g.*, health conditions, daily routines, and user activities) in IoT smart home environment. We pointed out the limitations of existing IoT traffic analysis techniques: the inability of inferring running apps, which are useful to derive context-rich sensitive information. Also, we performed the NLP analysis to automatically gain accurate privacy-sensitive information from app descriptions and input prompts. To understand how critical the threat is, we provided a systematic static analysis on 254 apps. Our evaluation results on real-world apps have demonstrated that ALTA could effectively learn running apps and sensitive information. We also discussed countermeasures to thwart such a privacy leakage analysis.

IX. ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their insightful comments. The WHU authors are supported by NSFC U1636107, 61972297. Yuan Luo was also supported by China Scholarship Council (201806270163).

REFERENCES

[1] "Samsung SmartThings," https://www.smartthings.com, 2018.

[2] "Google Home," 2019. [Online]. Available: https://store.google.com/product/google_home

[3] "Apple HomeKit," http://www.apple.com/ios/home, 2018.

[4] "IoT Platform Comparison: How the 450 providers stack up," https://iot-analytics.com/iot-platform-comparison-how-providers-stack-up/, 2018.

[5] "IoT: number of connected devices worldwide 2012-2025," 2019. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[6] L. Cheng, K. Tian, D. Yao, L. Sha, and R. A. Beyah, "Checking is believing: Event-aware program anomaly detection in cyber-physical systems," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[7] D. D. Yao, X. Shu, L. Cheng, and S. J. Stolfo, "Anomaly detection as a service: Challenges, advances, and opportunities," *Synthesis Lectures on Information Security, Privacy, and Trust*, vol. 9, no. 3, pp. 1–173, 2017.

[8] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE Symposium on Security and Privacy (IEEE S&P'16)*, pp. 636–654.

[9] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, "Sensitive information tracking in commodity iot," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1687–1704.

[10] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 147–158.

[11] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. M. Colbert, and P. McDaniel, "Iotsan: Fortifying the safety of iot systems," in *The 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT'18)*, 2018, pp. 191–203.

[12] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur, "Rethinking access control and authentication for the home internet of things (iot)," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 255–272.

[13] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms," in *Proceedings of the 24th Network and Distributed System Security Symposium (NDSS'17)*, San Diego, CA, 2017.

[14] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 531–548.

[15] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 361–378.

[16] Q. Wang, W. U. Hassan, A. M. Bates, and C. A. Gunter, "Fear and logging in the internet of things," in *25th Annual Network and Distributed System Security Symposium (NDSS'18)*, 2018.

[17] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app threats in smart homes: Categorization, detection and handling," *arXiv preprint arXiv:1808.02125*, 2018.

[18] H. Zhang, D. D. Yao, N. Ramakrishnan, and Z. Zhang, "Causality reasoning about network events for detecting stealthy malware activities," *computers & security*, vol. 58, pp. 180–198, 2016.

[19] H. Zhang, D. D. Yao, and N. Ramakrishnan, "Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 39–50.

[20] W. Dai, P. Wan, W. Qiang, L. T. Yang, D. Zou, H. Jin, S. Xu, and Z. Huang, "Tnguard: Securing iot oriented tenant networks based on sdn," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1411–1423, 2018.

[21] M. Zhou, Q. Wang, J. Yang, Q. Li, F. Xiao, Z. Wang, and X. Chen, "Patternlistener: Cracking android pattern lock using acoustic signals," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1775–1787.

[22] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A survey of physics-based attack detection in cyber-physical systems," *ACM Computing Surveys (CSUR'18)*, vol. 51, no. 4, p. 76, 2018.

[23] P. Kasinathan and J. Cuellar, "Securing emergent iot applications," in *International Summer School on Engineering Trustworthy Software Systems*. Springer, 2018, pp. 99–147.

[24] M. Agrawal, J. Zhou, and D. Chang, "A survey on lightweight authenticated encryption and challenges for securing industrial iot," in *Security and Privacy Trends in the Industrial Internet of Things*. Springer, 2019, pp. 71–94.

[25] A. Burg, A. Chattopadhyay, and K.-Y. Lam, "Wireless communication and security issues for cyber–physical systems and the internet-of-things," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 38–60, 2017.

[26] F. Kammüller, O. O. Ogunyanwo, and C. W. Probst, "Designing data protection for gdpr compliance into iot healthcare systems," *arXiv preprint arXiv:1901.02426*, 2019.

[27] X. Wu, R. Steinfeld, J. Liu, and C. Rudolph, "An implementation of access-control protocol for iot home scenario," in *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*. IEEE, 2017, pp. 31–37.

[28] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic," *CoRR*, vol. abs/1708.05044, 2017. [Online]. Available: http://arxiv.org/abs/1708.05044

[29] K. Yoshigoe, W. Dai, M. Abramson, and A. Jacobs, "Overcoming invasion of privacy in smart home environment with synthetic packet injection," in *2015 TRON Symposium (TRONSHOW)*, 2014, pp. 1–7.

[30] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 207–218. [Online]. Available: https://doi.org/10.1145/3395351.3399421

[31] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, "Keeping the smart home private with smart (er) iot traffic shaping," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 128–148, 2019.

[32] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, 2018, pp. 1074–1088.

[33] A. Alshehri, J. Granley, and C. Yue, "Attacking and protecting tunneled traffic of smart home devices," in *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 259–270. [Online]. Available: https://doi.org/10.1145/3374664.3375723

[34] "Event — SmartThings Classic Developer Documentation," 2019. [Online]. Available: https://docs.smartthings.com/en/latest/ref-docs/event-ref.html

[35] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-level signatures for smart home devices," *Signature*, vol. 10, no. 13, p. 54, 2020.

[36] "Router Bugs Flaws Hacks and Vulnerabilities." [Online]. Available: https://routersecurity.org/bugs.php

[37] "ARP spoofing," May 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ARP_spoofing&oldid=954838082

[38] "Your ISP Is Tracking Every Website You Visit: Here's What We Know." [Online]. Available: https://www.privacypolicies.com/blog/isp-tracking-you/

[39] "SmartThings open-source DeviceTypeHandlers and SmartApps code: SmartThingsCommunity/SmartThingsPublic," Jan. 2019. [Online]. Available: https://github.com/SmartThingsCommunity/SmartThingsPublic

[40] L. Babun, Z. B. Celik, P. McDaniel, and A. S. Uluagac, "Real-time analysis of privacy-(un) aware iot applications," *arXiv preprint arXiv:1911.10461*, 2019.

[41] "spaCy · Industrial-strength Natural Language Processing in Python," 2019. [Online]. Available: https://spacy.io/

[42] "OntoNotes Release 5.0 - Linguistic Data Consortium." [Online]. Available: https://catalog.ldc.upenn.edu/LDC2013T19

[43] "Word2vec," Apr. 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Word2vec&oldid=951422159

[44] "Levenshtein distance," Jan. 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Levenshtein_distance&oldid=878599785

[45] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Security and Privacy (EuroS&P'16), 2016 IEEE European Symposium on*, 2016, pp. 439–454.

[46] I. Apps, "A micro-benchmark suite to assess the effectiveness of tools designed for IoT apps: IoTBench/IoTBench-test-suite," Dec. 2018. [Online]. Available: https://github.com/IoTBench/IoTBench-test-suite

[47] W. Ding and H. Hu, "On the safety of iot device physical interaction control," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, 2018, pp. 832–846.

[48] "Scheduling — SmartThings Classic Developer Documentation," 2019. [Online]. Available: https://docs.smartthings.com/en/latest/smartapp-developers-guide/scheduling.html

[49] "Publishing Code — SmartThings Classic Developer Documentation," 2019. [Online]. Available: https://docs.smartthings.com/en/latest/publishing/

[50] S. Dong, Z. Li, D. Tang, J. Chen, M. Sun, and K. Zhang, "Your smart home can't keep a secret: Towards automated fingerprinting of iot traffic with neural networks," *arXiv preprint arXiv:1909.00104*, 2019.

[51] L. Yang, C. Seasholtz, B. Luo, and F. Li, "Hide your hackable smart home from remote attacks: The multipath onion iot gateways," in *European Symposium on Research in Computer Security (ESORICS'18)*. Springer, 2018, pp. 575–594.

[52] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, 2018.

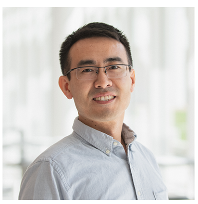
[53] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the attack surface of trigger-action iot platforms," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1439–1453.

[54] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "Homesnitch: behavior transparency and control for smart home iot devices," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2019, pp. 128–138.

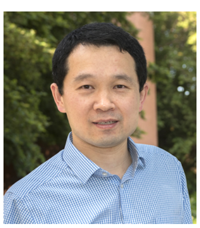
[55] S. Adepu, F. Brasser, L. Garcia, M. Rodler, L. Davi, A.-R. Sadeghi, and S. Zonouz, "Control behavior integrity for distributed cyber-physical systems," *arXiv preprint arXiv:1812.08310*, 2018.

[56] T. Song, R. Li, B. Mei, J. Yu, X. Xing, and X. Cheng, "A privacy preserving communication protocol for iot applications in smart homes," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1844–1852, 2017.

Yuan Luo is currently pursuing a Ph.D. degree with the School of Cyber Science and Engineering, Wuhan University, where he also received his bachelor's degree. His research interests include cyber-physical systems (CPS) security and mobile security.



Long Cheng is an assistant professor of computer science at Clemson University. He received his second Ph.D. in Computer Science from Virginia Tech USA in 2018, and the first Ph.D. from Beijing University of Posts and Telecommunications China in 2012. He was a Research Scientist at the Institute for Infocomm Research (I2R), Singapore from 2014 to 2015, and a Research Fellow at Singapore University of Technology and Design from 2012 to 2014. During the period from 2009 to 2011, he was a research assistant at the Hong Kong Polytechnic University, and the University of Texas at Arlington, USA. His research interests include system and network security, cyber forensics, cyber-physical systems (CPS) security, Internet of Things, mobile computing, and wireless networks. He received the Best Paper Award from IEEE Wireless Communications and Networking Conference (WCNC) in 2013, the Erasmus Mundus Scholar Award in 2014, and the Pratt Fellowship at Virginia Tech in 2017.



Hongxin Hu received the Ph.D. degree in computer science from Arizona State University, Tempe, AZ, USA, in 2012. He is currently an Associate Professor with the Division of Computer Science, School of Computing, Clemson University. He has published over 100 refereed technical papers, many of which appeared in top conferences and journals. His current research interests include security in emerging networking technologies, security in the Internet of Things (IoT), security and privacy in social networks, and security in cloud and mobile computing. He received the NSF CAREER Award in 2019. He was a recipient of the Best Paper Award from ACM SIGCSE 2018 and ACM CODASPY 2014 and the Best Paper Award Honorable Mention from ACM SACMAT 2016, IEEE ICNP 2015, and ACM SACMAT 2011.

Guojun Peng is currently a professor at the School of Cyber Science and Engineering, Wuhan University. He received the Ph.D. degree from Wuhan University in 2008. His research interests are system security and trust computing. He is a member of CCF and CSAC.

Danfeng (Daphne) Yao is a professor of computer science at Virginia Tech. Her expertise includes software and system security. In the past decade, she has been working on designing and developing data-driven anomaly detection techniques for securing networked systems against stealthy exploits and attacks. Dr. Yao received her Ph.D. in Computer Science from Brown University. Dr. Yao is an Elizabeth and James E. Turner Jr. '56 Faculty Fellow and L-3 Faculty Fellow. She has been named an ACM Distinguished Scientist for her contributions to cybersecurity. She received the NSF CAREER Award in 2010 for her work on human-behavior driven malware detection, and the ARO Young Investigator Award for her semantic reasoning for mission-oriented security work in 2014. She has several Best Paper Awards (e.g., ICNP '12, CollaborateCom '09, and ICICS '06) and Best Poster Awards (e.g., ACM CODASPY '15). She was given the Award for Technological Innovation from Brown University in 2006. Dr. Yao is an associate editor of IEEE Transactions on Dependable and Secure Computing (TDSC). She serves as PC members in numerous computer security conferences, including ACM CCS and IEEE S&P. She has over 85 peer-reviewed publications in major security and privacy conferences and journals.