HoMonit: Monitoring Smart Home Apps from Encrypted Traffic

Wei Zhang*
Shanghai Jiao Tong University
zhang-wei@sjtu.edu.cn

Yan Meng*
Shanghai Jiao Tong University
yan_meng@sjtu.edu.cn

Yugeng Liu
Shanghai Jiao Tong University
liuyugeng@sjtu.edu.cn

Xiaokuan Zhang
The Ohio State University
zhang.5840@osu.edu

Yinqian Zhang
The Ohio State University
yinqian@cse.ohiostate.edu

Haojin Zhu†
Shanghai Jiao Tong University
zhu-hj@cs.sjtu.edu.cn

ABSTRACT

Smart home is an emerging technology for intelligently connecting a large variety of smart sensors and devices to facilitate automation of home appliances, lighting, heating and cooling systems, and security and safety systems. Our research revolves around Samsung SmartThings, a smart home platform with the largest number of apps among currently available smart home platforms. The previous research has revealed several security flaws in the design of SmartThings, which allow malicious smart home apps (or SmartApps) to possess more privileges than they were designed and to eavesdrop or spoof events in the SmartThings platform. To address these problems, this paper leverages side-channel inference capabilities to design and develop a system, dubbed HoMonit, to monitor SmartApps from encrypted wireless traffic. To detect anomaly, HoMonit compares the SmartApps activities inferred from the encrypted traffic with their expected behaviors dictated in their source code or UI interfaces. To evaluate the effectiveness of HoMonit, we analyzed 181 official SmartApps and performed evaluation on 60 malicious SmartApps, which either performed over-privileged accesses to smart devices or conducted event-spoofing attacks. The evaluation results suggest that HoMonit can effectively validate the working logic of SmartApps and achieve a high accuracy in the detection of SmartApp misbehaviors.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; **Mobile platform security**; **Mobile and wireless security**;

KEYWORDS

IoT security; smart home; app misbehavior detection

ACM Reference Format:
Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. 2018. HoMonit: Monitoring Smart Home Apps from Encrypted Traffic. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3243734.3243820

*Co-first authors.
†Haojin Zhu is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '18, October 15–19, 2018, Toronto, ON, Canada
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243820

1 INTRODUCTION

Smart home, or sometimes called home automation, is a concept of adopting a large variety of Internet of Things (IoT) to aid control and automation of home appliances (*e.g.*, refrigerators, ovens and washer/dryers), lighting, heating and cooling systems (air conditioning, heaters), and various home security (*e.g.*, entry sensors, alarms) and safety (*e.g.*, water, freeze, smoke detectors) systems. In the recent years, the consumer market of smart home has experienced a rapid growth. According to IHS Markit [42], a global market size for smart home devices is forecast to be worth $3.3 billion by the end of 2017, reaching $9.4 billion in 2021. A survey suggests that the revenue from the US Smart Home market amounts to $18,877 million in 2018 with an annual growth rate of 14.8%. Besides, the household penetration is at 32.0% in 2018 and is expected to hit 53.1% by 2022 [63].

However, today, the smart home market is still in its infancy. Although numerous vendors have produced and marketed hundreds or thousands types of smart home devices, such as smart lights, smart switches and smart outlets, many of them can only interact with products from the same manufacturers. To foster inter-vendor compatibility and encourage community-based software development ecosystems (*e.g.*, iOS App Store), some major players in the market have developed a few smart home platforms to encourage manufacturers to produce compatible devices and software developers to develop applications with a uniform abstraction of smart devices. Prominent examples of these platforms include Samsung's SmartThings [52], Apple's HomeKit [3], Google's Brilo and Weave [23], Vera Control's Vera3 [13], and AllSeen Alliance's AllJoyn [1].

These platforms enable devices from different vendors to communicate through a local gateway (*e.g.*, a hub or a base station) or cloud backend servers. Software applications can be developed by third-party developers to enable smart control of the devices. For example, an application can monitor the status of one device (*e.g.*, motion sensor), and trigger some actions of another device (*e.g.*, turn on the light) upon receiving certain event notification (*e.g.*, human activity). The extensibility of the framework greatly stimulates a large number of device manufacturers and application developers to participate in the ecosystem. One good example of such platform is Samsung's SmartThings. At the end of 2015, SmartThings was the largest platform and had 521 SmartApps [19].

Along with the popularity of smart home platforms is the increasing concern on security and privacy breaches in smart home. Particularly in the case of Samsung's SmartThings, a recent research paper has revealed several flaws in the design of SmartThings, which allow malicious SmartApps—applications that run in the cloud backend and control smart devices—to not only possess more privileges than what they have been granted by the user on the smart devices, but also eavesdrop or even spoof events that should only be generated by smart devices [19].

Existing solutions to SmartThings security, especially on the aspect of misbehaving SmartApp detection and prevention, mainly fall into three categories: first, applying information flow control to confine sensitive data by modifying the smart home platform [20]; second, designing a context-based permission system for fine-grained access control [34]; third, enforcing context-aware authorization of SmartApps by analyzing the source code, annotation, and description [66]. However, these existing solutions either require modification of the platform itself [20, 66], or need to patch the SmartApps [34]. It is desirable to have a novel approach that allows a third-party defender—other than the smart home platform vendors, smart device manufacturers, and app developers—to monitor the smart home apps without making any change to the existing platform. However, without accessing to the smart sensors, gateway devices, and cloud back-end servers, the only avenue that permits the third-party monitoring is through the communication traffic, which, however, is encrypted using industry standards. Whether one could monitor the behavior of the smart home apps from encrypted traffic remains an open research question.

In this paper, we present HoMonit, a system for monitoring smart home apps from encrypted wireless traffic. We particularly demonstrate the concept by implementing HoMonit to work with Samsung's SmartThings framework and detect the misbehaving SmartApps. At the core of HoMonit is a Deterministic Finite Automaton (DFA) matching algorithm. Our intuition is that every smart home app's behavior follows a certain DFA model, in which each state represents the status of the app and the corresponding smart devices, and the transitions between states indicate interactions between the app and the devices.

To do so, HoMonit first extracts DFAs from the source code of the apps or the text information embedded in their descriptions and user interfaces (UI) of the SmartThings Android app. HoMonit then leverages wireless side-channel analysis to monitor the encrypted wireless traffic to infer the state transition of the DFA. Our *key insight* is that the smart home traffic is particularly vulnerable to side-channel analysis; the encrypted content can be inferred by observing the size and interval of encrypted wireless packets. Then HoMonit applies the DFA matching algorithm to compare the inferred transition with the expected DFA transitions of all installed SmartApps. If the DFA matching fails, with high probability the behavior of the SmartApp has deviated from the expectation—a misbehaving SmartApp is detected. Our evaluation suggests that HoMonit can effectively detect several types of SmartApp attacks, including over-privileged accesses and event spoofing.

We implemented HoMonit and evaluated its effectiveness in detecting misbehaving SmartApps. Totally 60 misbehaving SmartApps were developed by altering the code of open-source SmartApps to perform the evaluation. The results suggest that HoMonit detects misbehaving SmartApps effectively: it achieves an average recall of 0.98 for SmartApps with over-privileged accesses and an average recall of 0.99 for SmartApps conducting event spoofing attacks; the false positive rate is low when monitoring 30 benign SmartApps: an average F1 score of 0.98 for ZigBee and 0.96 for Z-Wave can be achieved.

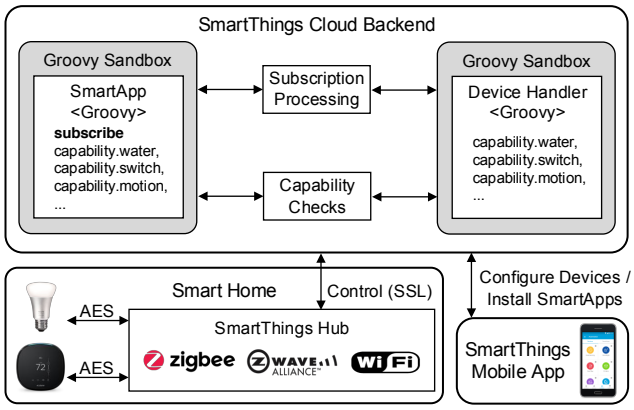


Figure 1: System architecture of SmartThings platform.

Contributions: The contributions of the paper include:

- *Novel techniques.* We developed techniques for extracting DFAs from the source code of the SmartApps or the UI of SmartThings' mobile app, and methods for inferring SmartApp activities using wireless side-channel analysis.
- *New systems.* We designed, implemented and evaluated HoMonit for detecting misbehaving SmartApps in the SmartThings platform, which operates without cooperation from the platforms, device vendors or SmartApp developers.
- *Open-source dataset.* A dataset of 60 misbehaving SmartApps will become publicly available, which can be used by researchers, vendors and developers to evaluate their security measures.

To the best of our knowledge, our work is the first to leverage wireless fingerprints to detect misbehaviors in a resource-constraint IoT environment (*e.g.*, smart home or industrial control systems). The difficulty underlying the research problem includes the challenges of upgrading legacy equipment to eliminate vulnerabilities, of deploying a monitoring system without modifying the existing infrastructure, of inferring the applications' behavior only from the encrypted traffic. Our work may shed light on how to design a practical misbehavior detection system in an IoT environment.

Roadmap. Sec. 2 introduces background of Samsung SmartThings. Sec. 3 discusses the motivation of our paper and key insights. Sec. 4 highlights the overall design of HoMonit. Then, Sec. 5 discusses DFA extraction from both open-source and closed-source SmartApps. Sec. 6 elaborates the details of traffic analysis, event inference, and DFA matching. Sec. 7 shows the evaluation result of HoMonit. Sec. 8 discusses the potential limitation and future work. Sec. 9 summarizes the related work and Sec. 10 concludes the paper.

2 BACKGROUND

2.1 Samsung SmartThings

As one of the most popular smart home platforms, Samsung SmartThings provides an attractive feature favored by many device manufacturers and software developers, which is the separation of intelligence from devices. In particular, it offers an abstraction of a variety of lower-layer smart devices to software developers so that the development of software programs (*i.e.*, SmartApps) is decoupled with the manufacture of the smart devices. In this way, the SmartThings platform fosters a vibrant software market, which encourages third-party software developers to enrich the diversity of home automation functionalities. So far, SmartThings supports 133 device types and 181 SmartApps[1] in the official GitHub repository [59]. It is conservatively estimated that at least 50 thousand families use SmartThings in 2017 [12].

The architecture of the SmartThings platform is shown in Fig. 1. Smart devices are the key building blocks of the entire SmartThings infrastructure. They are connected to the hub with a variety of communication protocols, including ZigBee, Z-Wave, and Wi-Fi.

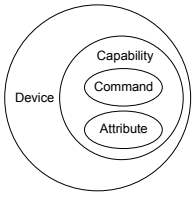


Figure 2: Relationship between device and capability.

In SmartThings, the hub mediates the communication with all connected devices, and serves as the gateway to the SmartThings cloud, where *device handlers* and SmartApps are hosted. Device handlers are virtual representations of physical devices, which abstract away the implementation details of the protocols for communicating to the devices. As shown in Fig. 2, device handlers specify the *capabilities* of these devices.

Capabilities can have *commands* and *attributes*. Commands are methods for SmartApps to control the devices; attributes reflect properties or characteristics of the devices. For example, smart device *Samsung SmartThings Outlet* has 9 capabilities, among which *Switch* and *Power Meter* are the most commonly used: *Switch* enables the control of the switch and it has two commands: *on()* and *off()*, while *Power Meter* has one attribute *power* for reporting the device's power consumption.

2.2 Communication Protocols

The SmartThings supports a variety of communication protocols. In particular, ZigBee and Z-Wave are typically characterized as protocols with low power consumption, low data communication rate and close proximity. As shown in Table 1, among the 133 smart devices we have surveyed, ZigBee and Z-Wave are two dominant wireless protocols in SmartThings device market, which together contribute to about 79.7% of market share.

[1]Partially due to the vulnerability disclosure, the popularity of SmartThings has decreased since then. As of May 2017, there are only 181 SmartApps available on the SmartThings official list [59]; over 300 SmartApps have been removed since 2016 either because of the discovered security risks or simply the lack of interests from the users [53][61].

Table 1: Protocols supported by SmartThings devices.

Protocols	Supported Devices
ZigBee	$48/133 \approx 36.1\%$
Z-Wave	$58/133 \approx 43.6\%$
Others	$27/133 \approx 20.3\%$

- *ZigBee* [2]. ZigBee is a wireless communication specification following the IEEE 802.15.4 standard. ZigBee devices in SmartThings are on the 2.4GHz band with 250 kbps data rate. ZigBee supports encrypted and authenticated communications. Encryption is enabled by default at the network layer with 128-bit AES-CCM* encryption mode. Application Support Sublayer (APS) also allows optional encryption with a link key, which is a shared key between two devices in the same Personal Area Network (PAN).
- *Z-Wave* [32]. Z-Wave is another popular low-power consumption communication protocol. Z-Wave is implemented by following the ITU-T G.9959 recommendation. It has different working frequencies in different regions. In the United States, the Z-Wave devices are specified to work on the frequency of 908.4MHz with 40 kbps data rate and 916MHz with 100 kbps data rate. Z-Wave supports strong encryption and authentication via Z-Wave S2 security solution, which implements 128-bit AES encryption.

2.3 Misbehaving SmartApps

Fernandes *et al.* [19] presented several security-critical design flaws in the SmartThings' capability model and event subsystem. These design flaws may enable the following SmartApp misbehaviors that can lead to security compromises:

- *Over-privileged accesses.* The capability model of SmartThings grants coarse-grained capabilities to SmartApps: (1) a SmartApp that only needs a certain command or attribute of a capability will always be granted the capability as a whole [58]; and (2) a SmartApp that is granted permission to some capabilities of a device can gain access to all capabilities of the devices. Hence, a malicious SmartApp could be significantly over-privileged and take control of a whole device even if it only asks for the permissions to access partial information. For example, an *auto-lock* SmartApp which only requires the *lock()* command of *Lock* capability can also get access to the *unlock()* command. Moreover, even for benign SmartApps, it is still possible for the attackers to launch command injection attacks to trick them to perform unintended actions. For example, in WebService SmartApps [60], if the developers implement the HTTP endpoints using the dynamic method invocation feature of Groovy, the SmartApp will be vulnerable to command injection attacks [19].
- *Event spoofing.* The SmartThings framework does not protect the integrity of the events, which allows event spoofing attacks. An event object is a data object created and processed in the SmartThings cloud, which contains information associated with the event, such as the (128 bit) identifiers of the hub and the device, as well as the state information. A malicious SmartApp with the knowledge of the hub and device identifiers, which are easy to learn, can spoof an arbitrary event. The event will be

deemed as legitimate by the SmartThings cloud and propagated to all SmartApps that subscribe the corresponding capability related to the event [19]. For example, an *alarm panel* SmartApp can raise a siren alarm when the CO detector is triggered. However, an attack SmartApp can spoof a fake event for the CO detector, causing the *alarm panel* SmartApp to activate the siren alarm mistakenly.

3 MOTIVATIONS AND INSIGHTS

Given the aforementioned threats from misbehaving SmartApps [19], systems for mitigating such threats are of practical importance. Previous studies have proposed context-based permission systems [34], user-centered authorization and enforcement mechanisms [66], and systems to enforce information flow control [20]. However, these solutions require either modification of the platform itself [20, 66], or changes in the SmartApps [34]. A security mechanism that works on existing platforms will be more practical as a business solution.

In this study, we propose a detection system, dubbed HoMonit, for detecting misbehaving SmartApps in a non-intrusive manner, by leveraging techniques commonly used in wireless side-channel inference. HoMonit is inspired by two observations. First, most communication protocols used in smart home environments are designed for a low transmission rate and reduced data redundancy for low power consumption. Second, the wireless communications between the hub and smart devices usually show unique and fixed patterns determined by the corresponding smart devices and SmartApps. Therefore, after extracting the working logic of SmartApps as Deterministic Finite Automatons (DFAs)—with the help of code analysis (for open-source SmartApps) or natural language processing techniques (for closed-source SmartApps)—an external observer can determine which SmartApp is operating and which state this SmartApp is currently in by monitoring only the meta data (*e.g.*, the packet size or inter-packet timing) of the encrypted wireless traffic. If a SmartApp deviates from its usual behavior, the pattern of wireless traffic will also change, which can be utilized to detect misbehaving SmartApps.

The capability of monitoring misbehaving SmartApps from encrypted traffic enables a third-party defender—other than the smart home platform vendors, smart device manufacturers and SmartApp developers—to develop a smart home anomaly detection system to detect misbehaving SmartApps at runtime. A major advantage of a third-party defense mechanism is that no modification of the protected platform is needed. HoMonit is designed to work without the need of changing the current SmartThings infrastructure, or changing the system software on the hub or smart devices, or modifying the SmartApps. HoMonit can work directly with the existing SmartThings platform, and is easily extensible to other platforms with similar infrastructures.

We illustrate this idea using a concrete example: *Brighten My Path* is a SmartApp for automatically turning on the outlet after a motion has been detected by the motion sensor. We show the observed packet sizes of the communications between the sensors and the hub in Fig. 3, in which the y-axis shows the packet sizes and the x-axis shows the timestamps of the packets when they arrive. The SmartApp subscribes to two capabilities, which include an attribute *motion* for capability *Motion Sensor* and a command *on()* for capability *Switch*. In Fig. 3, *motion.active* corresponds to packet sequence (54 ↑) and *switch.on* corresponds to packet sequence (50 ↓, 47 ↓, 47 ↓, 52 ↓), where ↓ means the packet is hub-to-device and ↑ means device-to-hub. The DFA consists of three states, which are connected by two transitions (as shown in Fig. 3). If the events corresponding to *motion.active* and *switch.on* are detected in a sequence, the DFA will transition from the *start* state to the *accept* state. Thus, the behavior of the SmartApps can be inferred from the DFA transitions: normal behavior sequences are always accepted by the DFA, whereas abnormal ones are not.

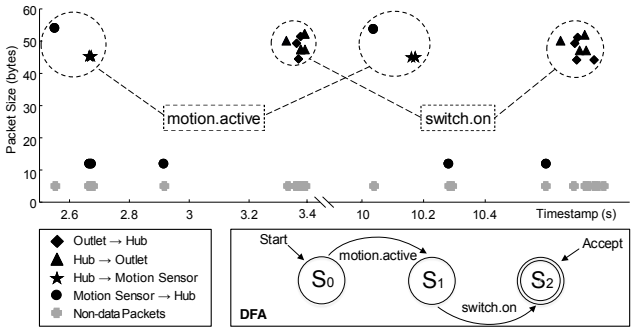


Figure 3: A motivating SmartApp: *Brighten My Path*.

4 DESIGN OVERVIEW

4.1 Threat Model

In this paper, we consider attackers who are capable of exploiting the vulnerabilities of a benign SmartApp to perform tasks that deviate from the original design goals (*i.e.*, benign DFA). Similar to the attack model in [19], the attacker-controlled SmartApp could misbehave by (1) performing over-privileged accesses to gain control of the smart devices in ways that are not prescribed by its intended functionality; or (2) performing event spoofing to alter the behaviors of other SmartApps that are installed by the same user. As shown in [19], the proof-of-concept attacks include door lock pin code snooping attack and fake alarm attack. The SmartApps from SmartThings Marketplace and SmartThings Public GitHub Repository [59] are assumed to be trusted and can be used to extract the benign DFA. It is important to point out that vulnerabilities detection of a SmartApp by analyzing its source code is an important research topic [19, 66], which deserves the separate researches and thus is out of the scope of this work. In this study, we do not consider the attacks towards smart home hardware or system software (*e.g.*, smart devices or hubs).

4.2 Design Challenges

To design and implement a smart home anomaly detection system using wireless side-channel inference techniques, we must address the following research challenges, including:

- How to automatically extract the control logic (*i.e.*, the DFA) of both open-source and closed-source SmartApps?
- How to automatically capture wireless traffic (*e.g.*, ZigBee and Z-Wave) in smart home environments and conduct side-channel inference to detect SmartApp misbehaviors?

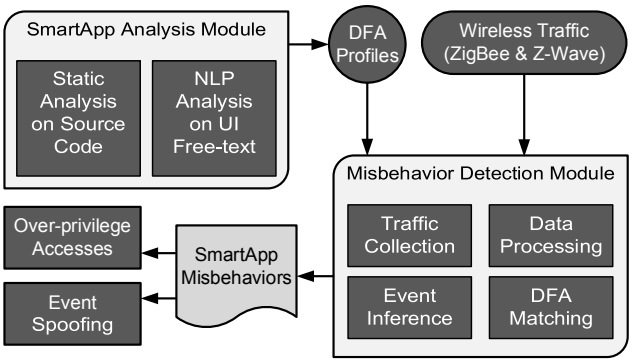


Figure 4: Workflow of HoMonit.

- How to evaluate detection accuracy in a real-world smart home setting?

4.3 System Overview

To address these design challenges, we present HoMonit, a system to monitor behaviors of smart devices using side-channel information from the encrypted wireless traffic. The smart home network is centralized by the hub, which connects to all of the smart devices via wireless communications (*e.g.*, ZigBee or Z-Wave), and to the cloud backend via Internet. HoMonit is equipped with multiple wireless interfaces to collect the wireless packets including both ZigBee and Z-Wave packets. In practice, the eavesdropping devices should be put near the hub to ensure that the wireless packets can be correctly collected.

As illustrated in Fig. 4, HoMonit system is comprised of two major components. The *SmartApp Analysis Module* (detailed in Sec. 5) extracts the expected DFA logic of the installed SmartApps from the their source code (for open-source SmartApps) or their text description (for closed-source SmartApps). The *Misbehavior Detection Module* (detailed in Sec. 6) identifies the misbehavior of the SmartApps by conducting side-channel inference on the sniffed wireless traffic and comparing the inferred behavior with the expected DFA models.

5 DFA BUILDING VIA SMARTAPP ANALYSIS

The SmartApp Analysis Module aims to extract the expected behaviors of the SmartApps. We utilize the Deterministic Finite Automaton (DFA) to characterize the logic of SmartApps. Considering the stateful nature of web applications (especially for smart home apps) [10], we choose DFA to represent a SmartApp for two reasons: (1) a SmartApp supervises a finite number of devices, and (2) devices are driven into a deterministic status by the SmartApp when a specific condition is satisfied. More specifically, we formalize the SmartApp DFA as a 5-tuple $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states of the SmartApp; Σ is a finite set of symbols, which correspond to *attributes* or *commands* of their capabilities; δ is the transition function in $Q \times \Sigma \rightarrow Q$, where $Q \times \Sigma$ is the set of 2-tuple (q, a) with $q \in Q$ and $a \in \Sigma$; q_0 is the start state, and F is a set of accept states.

Some SmartApps are open-source while others are closed-source. For open-source SmartApps, HoMonit performs static analysis on their source code and automatically translates them into DFAs. For closed-source SmartApps, HoMonit leverages both the descriptions of the SmartApps and texts embedded in the user interface (UI) of the SmartThings Android app to extract the logic of the Smart Apps and build the DFA. We analyzed 181 open-source SmartApps to built their DFAs from SmartThings Public GitHub Repository [59]. All of them are official open-source SmartApps. Moreover, 36 out of the 52 SmartApps (69.2%) in SmartThings Marketplace as of Jan. 2018 were included in the dataset.

```
1  definition(
2    name: "Smart Light", namespace: "com.example",
3    author: "example", category: "Convenience",
4    description: "Turn light on when motion detected."
5  )
6
7  preferences {
8    section("When there is movement...") {
9      input "themotion", "capability.motionSensor",
10           title: "Select a motion sensor"
11   }
12   section("Turn on a light...") {
13     input "theswitch", "capability.switch",
14           title: "Select a light"
15   }
16 }
17
18 def installed() {
19   subscribe(themotion, "motion.active", motionHandler)
20 }
21
22 def updated() {
23   unsubscribe()
24   subscribe(themotion, "motion.active", motionHandler)
25 }
26
27 def motionHandler(evt) {
28   theswitch.on()
29 }
```

Figure 5: A SmartApp code example: *Smart Light*.

5.1 DFA Building for Open-source Apps

Since the open-source SmartApps are written in Groovy, to extract their logic, we conducted a static analysis on the source code using AstBuilder [15]. Fig. 5 shows an example of the source code of a SmartApp. HoMonit converts the source code of the SmartApp into an Abstract Syntax Tree (AST) during the Groovy compilation phase.

The translation from an AST to a DFA is completed in two steps. First, to obtain the set of symbols (*i.e.*, Σ of the DFA), HoMonit extracts the capabilities requested by the DFA from the *preferences* block statement (Fig. 5, line 7). Specifically, all available capabilities are first obtained from the SmartThings Developer Documentation [58], and then the *input* method calls (Fig. 5, line 9 and 13) of the *preferences* block statement are scanned to extract the capabilities requested by the SmartApp. SmartApps use *subscribe* methods to request notification when the device's status has been changed; these notification will trigger the *handler* methods to react to these status changes. To further determine the specific commands or attributes (*i.e.*, symbols of the DFA), HoMonit scans the *subscribe* methods and their corresponding commands or attributes (*e.g.*, *motion.active* in the *subscribe* method).

The second step is to extract the state transitions (*i.e.*, δ) from *subscribe* and *handler* methods. HoMonit starts from the *subscribe*

method call in *installed* and *updated* block. Each *subscribe* method in these blocks indicates one transition from the start state to an intermediate state; by inspecting the corresponding *handler* method, how the intermediate state will transition to other states can be determined: in the example shown in Fig. 5, one transition (with *switch.on* as its symbol) moves the DFA to an accept state. Complex *handler* methods may involve multiple states and transitions before the DFA reaches the accept states. The set of states Q, start state q_0, and accept states F in the DFA are automatically constructed according to the transition function.

The DFAs for 150 out of 181 SmartApps were successfully constructed (82.9%). The DFA construction failed on some SmartApps because they request capabilities that are not associated with any device. The success rate is already very high considering that some SmartApps are much more complex than the one listed in Fig. 5. There were complex apps with over 28 states and 40 transitions in their DFAs in the dataset, and these DFAs could all be successfully extracted and further used in detection. Most of the popular SmartApps can be successfully constructed. More specifically, since SmartThings has not provided official data about app downloading statistics, the 52 SmartApps in SmartThings Marketplace can be regarded as the most popular apps. Among these 52 SmartApps, 36 are open-source and the remaining 16 are closed-source. Among the 36 open-source apps, we have successfully constructed DFAs for 32 apps. The reason of these 4 failure cases is that the working logic of these SmartApps cannot be modeled as DFAs. Take a SmartApp *Severe Weather Alert* as an example, it only acquires weather information from Internet and sends weather alert to user smartphone; a meaningful DFA cannot be constructed.

5.2 DFA Building for Closed-source Apps

The static code analysis approach can only be applied to open-source SmartApps. For the closed-source SmartApps, HoMonit builds the DFA by analyzing the text information of SmartApps. Compared to Tian *et al.* [66] who extracted the security policies of a SmartApp from its text description using NLP techniques by leveraging knowledge of source code, extracting DFA without source code, as is the case in our scenario, is more challenging, since the descriptions of the SmartApps are usually quite simple (*e.g.*, only one sentence). As such, we instead leverage the text embedded in the user interface of the SmartThings mobile app to extract the logic of the SmartApp to build its DFA. In particular, our NLP-based analysis is comprised of the following three steps: (1) text extraction, (2) symbol inference, and (3) DFA building.

5.2.1 Text Extraction. Fig. 6 shows the UI of an example SmartApp in the SmartThings Android app, which prompts the user to grant the SmartApp permission to access the devices. This interface is closely related to the *preferences* block statement, which is unknown to us. To extract the same information as we did from the source code, HoMonit first converts the UI hierarchy into an XML file by using *uiautomator* (an *adb* command on Android platform). HoMonit then traverses all XML nodes and extracts the string values of the *text* attributes in related nodes. For example, a sentence *"When water is sensed"* can be extracted from the XML node `<node index="0" text="When water is sensed" resource-id="com.smartthings.android:id/title">`.

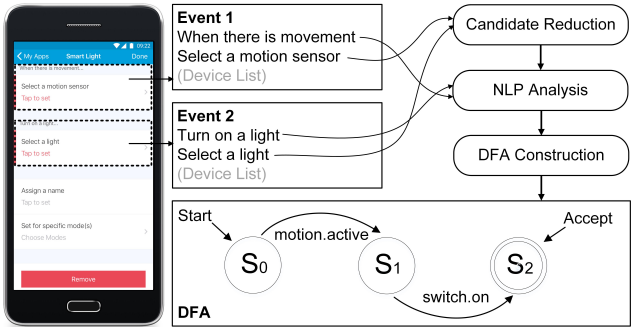


Figure 6: DFA building through UI analysis.

5.2.2 Symbol Inference. To extract the set of symbols in the DFA (*i.e.*, Σ), we need to infer associated commands or attributes of capabilities from the extracted text. The SmartThings Developer Documentation [58] defines 72 capabilities and the task of symbol inference is to select the correct one from the 72 candidates. However, directly applying NLP techniques is difficult, because text descriptions for SmartApps require a unique language model. For example, *Contact Sensor* capability implies physical touching of subjects, rather than inter-personal communication. Therefore, making inference based on generic language models is not accurate. Moreover, training a new model from the 346 sentences we extracted from the 181 SmartApps did not lead to acceptable inference results. As such, we perform an additional candidate set reduction step before applying NLP techniques.

Particularly, when the UI interface shown in Fig. 6 prompts the user to select devices that have the corresponding capability requested by the SmartApp, we can leverage the provided device list to narrow down our candidates, because all the devices in the list must contain that capability. Suppose there are n capabilities, denoted as C_j ($1 \le j \le n$), and m devices, denoted as D_i ($1 \le i \le m$). A device D_i can further be abstracted as a set of capabilities, $D_i = \{C_{i1}, C_{i2}, \ldots, C_{ik}\}$. For a given unknown capability c, we can obtain a list of devices L_c through the UI of the SmartApp, which can be utilized to obtain a reduced candidate set $\mathcal{R}$: $\mathcal{R} = \{C_u | C_u \in D_v \text{ for all } D_v \in L_c\}$.

As the target capability c must be in $\mathcal{R}$, if $|\mathcal{R}|$ is small, we are able to reduce our candidates significantly before applying the NLP technique. To show the extent to which the candidate capabilities can be reduced, we created 221 virtual devices by enumerating all device handlers obtained from the SmartThings Public GitHub Repository. There were 52 capabilities in total[2]. For each capability, we used the method mentioned above to create a candidate set $\mathcal{R}$. We found that $|\mathcal{R}|$ is only 1 for 28 out of the 52 capabilities, which means that 53.8% of the capabilities can be correctly inferred even without using NLP techniques. The average value of $|\mathcal{R}|$ for all 52 capabilities is only 2.27, which means that given the list of devices that support the requested capability, the candidate set is quite small.

We then use NLP analysis to perform symbol inference. HoMonit uses NLTK (Natural Language Toolkit) [7] to extract the noun

[2]The rest capabilities are only documented but not used by the device handlers in the repository.

phrases and verb phrases in a sentence. Capabilities are often related to the noun phrases in the text. For example, word *water* may be related to *Water Sensor* capability. Then HoMonit compares the noun phrases with capability candidate set and uses text similarity (estimated by Word2Vec [45]) to determine the requested capability. As noun phrases in SmartThings usually have special meanings, we combine Word2Vec with a model trained with the API documentation of SmartThings.

After learning the requested capabilities, the next step is to find out the exact attributes or commands of the capability. We first obtain the possible attributes and commands of each capability from the official documentation [56], and analyze the verb phrases to conduct the inference. In particular, we use Word2Vec to analyze the context of the verb phrases. For example, for *Switch* capability, there are two commands: *on()* and *off()*, and the verb phrase *turn on* is more likely to be linked to the *on()* command. However, some verb phrases do not clearly indicate which status of command it stands for. In this situation, we use NLTK to understand the context of the verb phrases and determine if it links to the common or uncommon device status. For example, in a sentence *"when water is sensed"*, it is difficult to determine whether it refers to the *wet* value or the *dry* value of *water* attribute. But since the *wet* status is uncommon, *"when water is sensed"* is more likely related to the *wet* value of *water* attribute. We conducted an empirical evaluation to show the effectiveness of our approach. In the experiment, the inference result with the similarity score lower than a threshold (0.9) is deemed as *undecided*, which need to be further analyzed using our NLP-based approach. In our dataset, 71 *undecided* cases were found out of the total 1207 samples. Using our approach, in 63 of these 71 *undecided* cases (88.7%), the device status with unclear verb phrases can be correctly inferred.

5.2.3 DFA Building. Since the set of symbols has been inferred, the remaining tasks of building the DFA is primarily constructing δ. This can be done by chaining the symbols according to the text, which describes the logical relationship of these symbols in the UI. For example, in Fig. 6, the text descriptions of the two symbols constitute a successive logical relationship: *"When there is movement, turn on a light"*. Thus we chain these two symbols following the order in the UI. When such text information is not available in the UI, we analyzed the description of the SmartApp as what they did in [66]. The remaining steps of DFA construction, *i.e.*, constructing Q, q_0, and F, are similar to those in Sec. 5.1.

Because we have no ground truth of the working logic of closed-source SmartApps, to evaluate the effectiveness of our method, we treated the 150 open-source SmartApps that we can successfully extract DFAs from their source code in Sec. 5.1 as closed-source SmartApps and conducted the evaluation. We have successfully built the DFAs of 122 out of 150 (81.3%) SmartApps using our method. Among the 28 SmartApps whose DFAs we failed to build, we could correctly infer the symbols of 15 SmartApps, but the order of the transitions were misplaced. We could not construct the symbols for the remaining 13 SmartApps.

5.2.4 Impact of UI design on DFA generation. Although the DFA generation procedure described above heavily depends on the UI design of the SmartThings mobile apps, it should be pointed out that the UI layout is constrained by the rules defined by SmartThings platform [57]. Therefore, any complying UI layout can be used for DFA generation. In our dataset, a UI always conforms to the app control logic. For example, the user interfaces of SmartApps *Turn It On When It Opens* and *Undead Early Warning* are quite similar, differing only slightly in the prompting sentences. As a result, our NLP-based approach will generate the same DFA for both SmartApps, which correctly reflects their actual control logic.

6 DETECTING APP MISBEHAVIORS BASED ON WIRELESS TRAFFIC FINGERPRINT

6.1 Traffic Collection

HoMonit collects both ZigBee and Z-Wave traffic between the hub and the smart devices. Our study suggests that these two standards are used by about 79.7% smart home devices on the market (Table 1).

6.1.1 ZigBee Traffic Collection. To sniff the ZigBee traffic, HoMonit employs a commercial off-the-shelf ZigBee sniffer (*i.e.*, Texas Instruments CC2531 USB Dongle [30]) and an open-source software tool (*i.e.*, 802.15.4 monitor [46]) to passively collect the ZigBee traffic. ZigBee breaks the 2.4GHz band into 16 channels, where SmartThings hub and its devices are on a fixed channel (`0xe` in our case). We customized the 802.15.4 monitor to achieve the real-time packet capturing. The captured packets were dumped to a log file once per second.

6.1.2 Z-Wave Traffic Collection. HoMonit adopts Universal Software Radio Peripheral (USRP) hardwares to collect Z-Wave packets and modifies an open-source software, Scapy-Radio [14], to automatically record them. It is worth noting that some Z-Wave devices may communicate with the hub at different channel frequencies in different modes (*e.g.*, *sleep* or *active* mode). Take the alarm sensor *Aeotec Siren (Gen 5)* as an example. The device communicates with the hub in *sleep* mode at the frequency of 908.4MHz with a transmission rate of 40kbps, but communicates at the frequency of 916MHz with a transmission rate of 100kbps rate in the *active* mode. As such, to monitor two channels simultaneously, we exploited two USRPs working at the above two frequencies to capture all Z-Wave traffic.

6.2 Event Inference

6.2.1 Filtering Noise Traffic. The collected wireless traffic contains packets that are considered noise for our event inference, which must be filtered out.

- *Beacon packets.* Beacon packets are mainly used for acknowledging data transmission and maintaining established connection, which carry less side-channel information. HoMonit discards ZigBee beacon packets, and drops Z-Wave packets with no payload.
- *Retransmission packets.* Retransmission packets will be sent in cases of transmission failure. In ZigBee, they can be identified by checking if two subsequent packets share the same sequence number. In Z-Wave, retransmission packets can be identified if two consecutive packets sent by the sending device are observed without having a response packet in between.
- *Unrelated traffic.* Traffic from devices using other wireless standards (*e.g.*, WiFi and Bluetooth) or from other networks are

Table 2: Fingerprints for event types that supported by 7 ZigBee devices and 4 Z-Wave devices.

Event Name	Device Name	Protocol	Fingerprint
water.wet[A]	Samsung SmartThings Water Leak Sensor	ZigBee	54 ↑ 45 ↑
water.dry			54 ↑ 45 ↑
temperature			53 ↑ 45 ↑
motion.active	Samsung SmartThings Motion Sensor	ZigBee	54 ↑
motion.inactive			54 ↑
temperature[B]			53 ↑ 45 ↑
switch.on[C]	Samsung SmartThings Outlet	ZigBee	50 ↓ 47 ↓ 47 ↓ 52 ↓
switch.off			50 ↓ 47 ↓ 47 ↓ 52 ↓
contact.open[D]	Samsung SmartThings Multipurpose Sensor (2016)	ZigBee	54 ↑
contact.closed			54 ↑
acceleration.active			69 ↑ 65 ↑ 65 ↑ 65 ↑ ···
acceleration.inactive			Occur after event *acceleration.active* finishes
temperature			53 ↑
contact.open	Samsung SmartThings Multipurpose Sensor (2015)	ZigBee	54 ↑ 45 ↑
contact.closed			54 ↑ 45 ↑
acceleration.active[E]			69 ↑ 65 ↑ 65 ↑ 65 ↑ ···
acceleration.inactive			Occur after event *acceleration.active* finishes
temperature			53 ↑ 45 ↑
beep[F]	Samsung SmartThings Arrival Sensor	ZigBee	50 ↓ 45 ↓
rssi[G]			52 ↑
presence.present[H]			57 ↑ 48 ↑ 45 ↑ 45 ↑ 49 ↑ 45 ↑ 50 ↑ 45 ↑ 50 ↑ 45 ↑ 50 ↑ 45 ↑
presence.not present			Occur after there is no periodic event *rssi*
switch.on[I]	Osram Lightify CLA 60 RGBW	ZigBee	50 ↓ 47 ↓
switch.off			50 ↓ 47 ↓
illuminance[J]			53 ↓ 47 ↓
setColorTemperature[K]			54 ↓ 47 ↓ 52 ↓ 47 ↓
setColor[L]			50 ↓ 47 ↓ 54 ↓ 47 ↓ 52 ↓ 47 ↓ 52 ↓ 47 ↓
switch.on	Power Monitor Switch (TD1200Z1)	Z-Wave	13 ↓ 12 ↓ 10 ↓
switch.off			13 ↓ 12 ↓ 10 ↓
motion.active	Aeotec MultiSensor 6	Z-Wave	14 ↑ 21 ↑
motion.inactive			14 ↑ 21 ↑
contact.open	Aeotec Door/Window Sensor 6	Z-Wave	17 ↑ 17 ↑
contact.closed			17 ↑ 17 ↑
alarm.siren	Aeotec Siren (Gen 5)	Z-Wave	13 ↓ 34 ↓ 11 ↓ 33 ↓ 11 ↓ 21 ↓ 11 ↓

treated as unrelated traffic. To identify traffic from targeted networks, ZigBee uses a unique identifier called Personal Area Network Identifier (or PANID for short) while Z-Wave uses Home ID, which denotes the ID that the Primary Controller assigns the node during the inclusion process [28]. HoMonit filters out collected traffic that has different PANIDs or Home IDs from the specified ones.

6.2.2 Fingerprinting Events. We formally denote an event as E_t^{ϕ}, which indicates that the event is of type ϕ and is generated at time t. An event type ϕ is a 2-tuple (d, e), where d is the device and e is a command sent to d or an attribute of d. We denote the set of all event types as Φ.

Each event will trigger a sequence of wireless packets. We denote a wireless packet as a quadruple $f = (t, l, d_i, d_j)$, where f refers to the packet of length l sent from device d_i to device d_j at time t. Here, d_i and d_j are represented using the MAC addresses in ZigBee or node IDs in Z-Wave. Once an event is triggered, a sequence of n packets sent between device d_i and d_j at a specific time t can be monitored during a short interval, which can be represented as $S_t^{d_i \leftrightarrow d_j} = (f_1, f_2, \ldots, f_n)$. Note that either d_i or d_j is the hub because the SmartThings framework dictates all the devices communicate with the hub. If packets for multi-hop communications are captured, HoMonit merges these consecutive packets from multiple hops into a single one. Therefore, there is typically a one-to-one mapping between an event E_t^{ϕ} and a sequence of packets, $S_t^{d_i \leftrightarrow d_j}$.

For each event type $\phi \in \Phi$, we manually trigger the event and collect m samples ($m = 50$), denoted as $\mathbb{S}^{\phi} = \{S_1^{\phi}, S_2^{\phi}, \ldots, S_m^{\phi}\}$, where S_i^{ϕ} is a sequence of packets collected in one experiment when the event is triggered. The fingerprint $\mathcal{F}^{\phi}$ of event type ϕ is defined as

$$\mathcal{F}^{\phi} = \arg\min_{S_i^{\phi} \in \mathbb{S}^{\phi}} \frac{1}{\|\mathbb{S}^{\phi}\|} \sum_{\forall S_j^{\phi} \in \mathbb{S}^{\phi}} dist(S_i^{\phi}, S_j^{\phi}),$$

where $dist(S_i^{\phi}, S_j^{\phi})$ adopts *Levenshtein Distance* [8] to measure the sequence similarity between S_i^{ϕ} and S_j^{ϕ}, *i.e.*, a small $dist(S_i^{\phi}, S_j^{\phi})$ means a high similarity between S_i^{ϕ} and S_j^{ϕ}. In Table 2, we illustrate the fingerprints of 34 different types of events of all devices we possess (including seven ZigBee devices and four Z-Wave devices, as listed in Table 3). Numeric values are size of the packets, and arrows indicate their directions.

To validate the uniqueness of the event fingerprints, we calculate the pairwise Levenshtein Distance between the fingerprints of different event types and their 50 event samples collected from all devices we possess. In Fig. 7, we illustrate the distance measure for 12 types of ZigBee events available on these seven devices, and

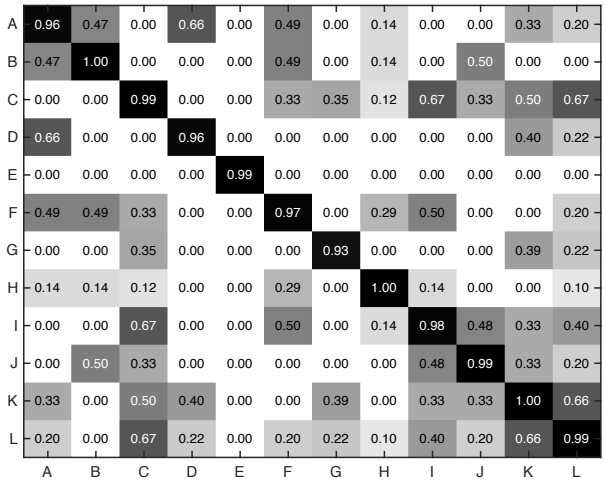
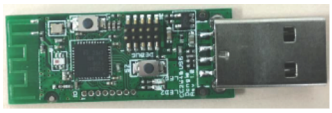


Figure 7: Levenshtein ratio for 12 types of ZigBee events available on 7 devices.



(a) ZigBee sniffer: Texas Instruments CC2531 USB Dongle.

(b) Four tested Z-Wave devices.

(c) Z-Wave sniffer: two USRPs at 908.4MHz and 916MHz.

(d) Seven tested ZigBee devices.

Figure 8: Wireless sniffers and smart devices.

the event names are marked in Table 2. More specifically, each cell of the table shows the *Levenshtein Ratio* [48], which is the result after normalizing average $dist(S_i^{\phi}, \mathcal{F}_j^{\phi})$. A high Levenshtein Ratio means a high sequence similarity. It is observed that the fingerprints of the same event are quite stable and consistent: the average Levenshtein Ratio is 0.98 for ZigBee and 0.98 for Z-Wave; and the fingerprints from different events are clearly distinguishable: the average Levenshtein Ratio is 0.17 for ZigBee and 0.25 for Z-Wave. The uniqueness of event fingerprints suggests that fingerprinting SmartThings events is feasible.

6.2.3 Inferring Events. To infer the events based on the captured wireless packets, HoMonit first partitions the traffic flow into a set of *bursts*. A burst is a group of network packets in which the interval between any two consecutive packets is less than a predetermined *burst threshold* [65]. The packets in each burst are then ordered according to the timestamps and the burst is represented as $S_t^{d_i \leftrightarrow d_j}$. HoMonit matches the burst with the fingerprints of each of the known events by calculating their Levenshtein Distance, $dist(S_t^{d_i \leftrightarrow d_j}, \mathcal{F}^{\phi})$. The event type with the smallest Levenshtein Distance from the packet sequence is considered as the inferred event.

As shown in Table 2, there are more than one events with exactly the same patterns (*e.g.*, packet size and direction). To correctly identify the event, we classify them into two categories:

- *Events of the same event type.* One example is *switch.on* and *switch.off* of Samsung SmartThings Outlet. The reason is that they are essentially the same event message with different data fields. As these events typically exist in pairs, such as *on* and *off*, *active* and *inactive*, *wet* and *dry*, we use one bit to trace the current state of each device to differentiate these events.
- *Events of different event types.* One example is *water.wet* of Samsung SmartThings Water Leak Sensor and *contact.open* of Samsung SmartThings Multipurpose Sensor (2015). We first use other unique events to identify the device, then determine the event type. For example, if we captured *acceleration.active*, then we know that this device is Samsung SmartThings Multipurpose Sensor (2015). Therefore, the event type must be *contact.open* instead of *water.wet*.

6.3 SmartApp Misbehavior Detection

To detect misbehaving SmartApps, HoMonit aims to propose the DFA matching algorithm. In particular, the SmartApp misbehavior detection module serves as an intrusion detection system (IDS) that monitors the smart home wireless network for misbehaving SmartApps. This module passively sniffs the wireless traffic between SmartThings hub and devices and tries to match it with current SmartApps working logic. An alarm will be raised once the verification fails.

Formally, the input of the algorithm is a sequence of events $\mathbb{E} = \{E_{t_1}^{\phi_1}, E_{t_2}^{\phi_2}, \ldots, E_{t_n}^{\phi_n}\}$ that is inferred from the encrypted wireless traffic, and the DFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ of the target SmartApp. The algorithm transitions the DFA from state S_i to S_{i+1} by consuming each of the events $E_{t_i}^{\phi_i}$ in the order they appear in the sequence, if $E_{t_i}^{\phi_i} \in \Sigma \wedge \delta(S_i, E_{t_i}^{\phi_i}) = S_{i+1} \in Q$. Initially, $S_0 = q_0$. If the sequence of events finally transitions the DFA into one of the accept states, that is, $S_n \in F$, the behavior of the SmartApp is accepted; otherwise a misbehaving SmartApp is detected.

In this work, we focus on detecting over-privilege and spoofing misbehavior of SmartApps. Since we assume the original SmartApp is benign, we aim to detect the misbehavior with DFA matching method. The over-privilege problem occurs due to coarse-grained permission control, which will raise extra device event compared with the original SmartApps. This misbehavior can be detected when there are new transitions detected in the DFA. The spoofing problem occurs due to being notified of fake events generated by other malicious SmartApps. Thus misbehavior can be detected when there is partial DFA matching (skipped states are resulted from spoofed events in the cloud). We will give detection details in following evaluation section.

Table 3: SmartApps used in the evaluation.

SmartApp	Protocol	Devices	Pre	Rec	F1
Lights Off When Closed	ZigBee	Samsung SmartThings Multipurpose Sensor (2015), Osram Lightify CLA 60 RGBW	1.00	0.90	0.95
Turn It On When It Opens	ZigBee	Samsung SmartThings Multipurpose Sensor (2016), Samsung SmartThings Outlet	1.00	0.95	0.97
Darken Behind Me	ZigBee	Samsung SmartThings Motion Sensor, Osram Lightify CLA 60 RGBW	1.00	0.95	0.97
Let There Be Light	ZigBee	Samsung SmartThings Multipurpose Sensor (2015), Osram Lightify CLA 60 RGBW	1.00	0.95	0.97
Monitor On Sense	ZigBee	Samsung SmartThings Multipurpose Sensor (2016), Samsung SmartThings Outlet	1.00	0.80	0.89
Big Turn On	ZigBee	Samsung SmartThings Outlet	1.00	1.00	1.00
Big Turn Off	ZigBee	Samsung SmartThings Outlet	1.00	1.00	1.00
Presence Change Push	ZigBee	Samsung SmartThings Arrival Sensor	1.00	1.00	1.00
Door Knocker	ZigBee	Samsung SmartThings Multipurpose Sensor (2016)	1.00	0.95	0.97
Let There Be Dark	ZigBee	Samsung SmartThings Multipurpose Sensor (2015), Osram Lightify CLA 60 RGBW	1.00	0.80	0.89
Flood Alert	ZigBee	Samsung SmartThings Water Leak Sensor	1.00	1.00	1.00
Turn It On When I'm here	ZigBee	Samsung SmartThings Arrival Sensor, Samsung SmartThings Outlet	1.00	1.00	1.00
The Gun Case Moved	ZigBee	Samsung SmartThings Multipurpose Sensor (2015)	1.00	1.00	1.00
It Moved	ZigBee	Samsung SmartThings Multipurpose Sensor (2016)	1.00	1.00	1.00
Light Follows Me	ZigBee	Samsung SmartThings Motion Sensor, Osram Lightify CLA 60 RGBW	1.00	0.95	0.97
Undead Early Warning	ZigBee	Samsung SmartThings Multipurpose Sensor (2016), Osram Lightify CLA 60 RGBW	1.00	0.90	0.95
Cameras On When I'm Away	ZigBee	Samsung SmartThings Arrival Sensor, Samsung SmartThings Outlet	1.00	0.95	0.97
Brighten My Path	ZigBee	Samsung SmartThings Motion Sensor, Osram Lightify CLA 60 RGBW	1.00	1.00	1.00
Dry The Wetspot	ZigBee	Samsung SmartThings Water Leak Sensor, Samsung SmartThings Multipurpose Sensor (2016)	1.00	0.95	0.97
Curling Iron	ZigBee	Samsung SmartThings Motion Sensor, Samsung SmartThings Arrival Sensor, Samsung SmartThings Outlet	1.00	1.00	1.00
Big Turn On	Z-Wave	Power Monitor Switch (TD1200Z1)	1.00	0.90	0.95
Brighten My Path	Z-Wave	Aeotec MultiSensor 6, Power Monitor Switch (TD1200Z1)	1.00	0.85	0.92
Darken Behind Me	Z-Wave	Aeotec MultiSensor 6, Power Monitor Switch (TD1200Z1)	1.00	0.85	9.92
Forgiving Security	Z-Wave	Aeotec MultiSensor 6, Aeotec Siren (Gen 5), Power Monitor Switch (TD1200Z1)	1.00	0.90	0.95
Let There Be Dark	Z-Wave	Aeotec Door/Window Sensor 6, Power Monitor Switch (TD1200Z1)	1.00	0.95	0.97
Let There Be Light	Z-Wave	Aeotec Door/Window Sensor 6, Power Monitor Switch (TD1200Z1)	1.00	0.95	0.97
Light Follows Me	Z-Wave	Aeotec MultiSensor 6, Power Monitor Switch (TD1200Z1)	1.00	0.90	0.95
Lights Off When Closed	Z-Wave	Aeotec Door/Window Sensor 6, Power Monitor Switch (TD1200Z1)	1.00	0.95	0.97
Smart Security	Z-Wave	Aeotec MultiSensor 6, Aeotec Siren (Gen 5), Aeotec Door/Window Sensor 6	1.00	1.00	1.00
Turn It On When It Opens	Z-Wave	Aeotec Door/Window Sensor 6, Power Monitor Switch (TD1200Z1)	1.00	0.95	0.97

7 EVALUATION

To evaluate the effectiveness and efficiency of HoMonit, we built a prototype system with the off-the-shelf hardware: a laptop equipped with wireless sniffer interfaces, including a Texas Instruments CC2531 USB Dongle for ZigBee and two USRPs for Z-Wave, as shown in Fig. 8(a) and Fig. 8(c). The distance between the SmartThings hub and HoMonit was about 7 feet. As listed in Table 3, we chose 30 SmartApps from the SmartThings Public GitHub Repository [59], which interact with, in total, 7 ZigBee devices and 4 Z-Wave devices, as shown in Fig. 8(b) and Fig. 8(d). The devices were located in less than 33 feet away from the hub within a room of 200 square feet.

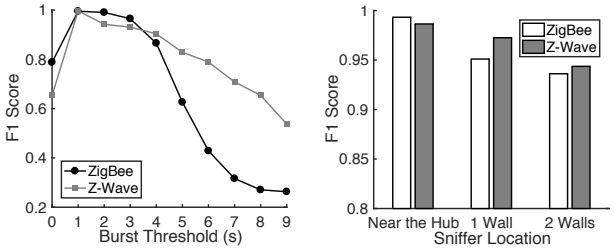
7.1 Micro-benchmark: Inference of Events and SmartApps

In this section, we evaluate the accuracy of inferring the SmartApps installed in the smart home environment from the sniffed wireless traffic. Although it is not the design goal of HoMonit (because we already assume the knowledge of the installed SmartApps), the accuracy of SmartApps inference measures the basic capability of DFA construction and DFA matching. Therefore, we use a set of SmartApp inference tests as micro-benchmarks. We also discuss the impact of a few key parameters of HoMonit, including the *burst threshold* and *sniffer distance and wireless obstacles*, on the accuracy of SmartApp inference.

7.1.1 Determining the Burst Threshold. The *burst threshold* is a parameter used to cluster captured wireless packets for the same event, which directly impacts the effectiveness of SmartApp inference. We performed the following experiments: we randomly selected 4 ZigBee devices and another 4 Z-Wave devices. We manually triggered each event type for 50 times on each of the 8 devices. The time intervals between two consecutive events were 3 to 10 seconds. We measured the accuracy of SmartApp inference when the burst threshold was selected as integer values from 0 to 10 seconds. The precision of the inference is defined as the fraction of correctly inferred events over all inferred events; the recall of the inference is defined as the fraction of successfully inferred events over all events that have been triggered. The F1 score is simply the harmonic mean of precision and recall.

As shown in Fig. 9(a), the F1 score of event inference achieves the maximum when the burst threshold is 1s. This is because a smaller burst threshold separates the packets belonging to the same events, which may cause more events being inferred than what we actually triggered; and a larger burst threshold groups unrelated packets, which may cause some events being missed by the detector. Therefore, in the remainder of our evaluation, the burst threshold was chosen as 1s.

7.1.2 SmartApp Inference Accuracy. We chose 20 SmartApps that work with ZigBee devices and 10 SmartApps connecting Z-Wave devices (listed in Table 3). Each SmartApp is invoked by manually-triggered events 20 times. During the experiment, the sniffer was placed 6 feet away from the hub and the burst threshold was set



(a) Evaluation of burst threshold. The y-axis shows the F1 score of event inference.

(b) Impact of sniffer distance and wireless obstacles. The y-axis shows the F1 score of SmartApp inference.

Figure 9: Micro-benchmark: accuracy of event and SmartApp inference.

to 1s. Table 3 contains the evaluation results for each individual SmartApp. Similar to event inference, here the precision is defined as the number of correctly inferred SmartApp invocation over the total number of inferences made; the recall is defined as the number of successfully inferred SmartApp invocation over the 20 invocation of each SmartApp. It is worth noting that the precisions for all SmartApps are 1.00, while the recalls are sometimes lower than 1.00. An average F1 score of 0.98 for ZigBee SmartApps and 0.96 for Z-Wave SmartApps were achieved. Among all 30 tested SmartApps, the F1 scores of 26 are at least 0.95 (see Table 3). This shows that HoMonit can accurately capture the working logic of SmartApps through DFA matching. It is also important to point out that the major factors contributing to the false inference come from the packet loss, unrelated wireless traffic, or traffic jam splitting a burst.

7.1.3 Impact of Distance and Wireless Obstacles. In practice, the effectiveness of SmartApp inference may be affected by the environmental conditions, such as the distance between the sniffer and the devices and various wireless obstacles (*e.g.*, walls) that block the wireless signals. Thus, we evaluated the effectiveness of SmartApp inference with different distances and wireless obstacles: (1) 6 feet without walls; (2) 16 feet with 1 wall; and (3) 33 feet with 2 walls. As shown in Fig. 9(b), although the recalls of the inference drop with longer distance and more wireless obstacles, in all three cases the recalls are above 0.88; the precisions are all 1.00; and the F1 scores are all above 0.94.

7.2 Detection of Over-privileged Accesses

We first developed over-privileged versions of the original benign SmartApps by adding malicious code to cause unintended operations. We used modified SmartApps to evaluate because there is no existing public malware dataset for SmartThings platform. We developed the misbehaving SmartApps following [19]. HoMonit will detect any malware that has over-privilege and spoofing problems. For example, a SmartApp named *Brighten My Path*, which is used to turn the light on when motion is detected, only requires *on()* command of *switch* capability according to its description. We developed an over-privileged version of this SmartApp in either of the two ways: (1) illegally obtaining accesses to *off()* command; (2) illegally gaining accesses to all the capabilities of the devices for which the user grants the SmartApp only *switch* capability. We selected 20 SmartApps that work with different categories of ZigBee devices and 10 SmartApps for Z-Wave devices (listed in Table 3). By following the above example, in total 30 over-privileged SmartApps were developed as the misbehaving SmartApps for evaluation. Therefore, the dataset used in our evaluation contains 30 benign SmartApps and 30 misbehaving SmartApps. We recruited 3 volunteers to simulate residents of the home. For each SmartApp (including both of the benign and over-privileged versions), the related devices were manually triggered for 20 times by the volunteers during 20 minutes. During this period, HoMonit continuously monitors the wireless channel and detects the misbehaviors in real-time.

The detection results are shown in Fig. 10. Here a true positive (TP) is defined as a correctly labeled misbehaving SmartApp; a true negative (TN) is defined as a correctly labeled benign SmartApp; a false positive (FP) is defined as an incorrectly labeled benign SmartApp; and a false negative (FN) is defined as an incorrectly labeled misbehaving SmartApp. Therefore, the true positive rate (TPR) is defined as TP/(TP+FN); the true negative rate (TNR) is defined as TN/(FP+TN). As shown in Table 4, the average TPR (over 40 ZigBee SmartApps) is 0.98 in detecting over-privileged accesses, with a standard deviation of 0.03; the average TNR of ZigBee SmartApps is 0.95, with a standard deviation of 0.07. The detection of Z-Wave SmartApps achieves similar TPR and TNR, which are 0.98 and 0.92, respectively.

The major reason for failed test cases is packet loss and a few unexpected wireless traffic which influence the event inference. Besides, accidental signal reception delay will break the consistency of frames for a device event, which may result in a false alarm for normal SmartApps.

7.3 Detection of Event Spoofing

We first developed event-spoofing versions of the benign SmartApps by adding malicious code to cause unintended operations. Specifically, the attackers exploit the insufficient event protection of SmartThings to spoof a physical device event and trigger the SmartApps which subscribe to this event. For example, *Flood Alert* is a SmartApp which triggers a siren alarm when the water sensor detects the wet state. In SmartThings, each connected device is assigned with a 128-bit device identifier when it is paired with a hub. Once a SmartApp acquires the identifier for a device, it can spoof all the events of that device without possessing any of the capabilities that device supports. By imitating the attacker, we raised a fake water sensor event in the cloud with a malicious SmartApp, causing the flood alert SmartApp to react and raise an alert.

In this experiment, we developed 30 misbehaving SmartApps which spoofed the device events by modifying the same set of 30 benign SmartApps (see Table 3). We performed an experimental evaluation on detecting the event spoofing attacks by following similar procedures as the previous section. The detection results are shown in Fig. 10. As shown in Table 4, the average TPR (over 40 ZigBee SmartApps) is 0.99 in detecting event spoofing, with a standard deviation of 0.02; the average TNR of ZigBee SmartApps is 0.95, with a standard deviation of 0.06. The detection of Z-Wave SmartApps achieves a similar TPR and a slightly lower TNR, which are 0.99 and 0.92, respectively.

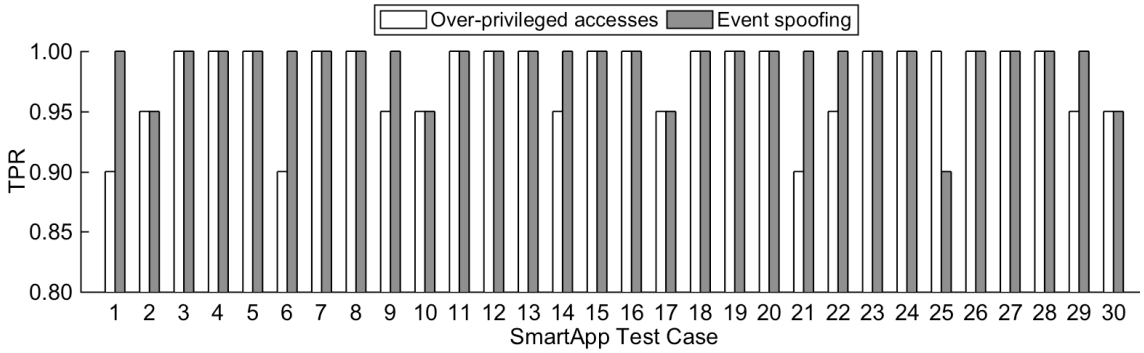


Figure 10: Detection result of 30 misbehaving SmartApps. The y-axis shows the TPR for each SmartApp.

Table 4: Detection result of misbehavior occurrence in SmartApps.

	ZigBee (20 misbehaving + 20 benign)		Z-Wave (10 misbehaving + 10 benign)	
	Over-privileged accesses	**Event spoofing**	**Over-privileged accesses**	**Event spoofing**
TPR	0.98 (0.03)	0.99 (0.02)	0.98 (0.04)	0.99 (0.04)
TNR	0.95 (0.07)	0.95 (0.06)	0.92 (0.05)	0.92 (0.05)

8 DISCUSSIONS

Generality and applicability. Though this work mainly investigates the SmartThings platform, the presented approach can be potentially applied to other IoT systems. This is because, in IoT environment, most of the devices are power-constraint and the employed wireless protocols are light-weight. These light-weight protocols are typically designed for low transmission rate and reduced data redundancy for low power consumption, which inherently suffers from a low-entropy issue. Therefore, this feature gives HoMonit the opportunity to extract wireless fingerprints for smart devices by analyzing the packet size and timing information.

We also investigate IFTTT [29], an open-source platform compatible with SmartThings, to further demonstrate the applicability and generality of HoMonit from both of the aspects of the wireless fingerprints capturing and the DFA building. To capture the wireless fingerprints in IFTTT, we develop an Applet (a SmartApp in IFTTT) that automatically turns on/off the Samsung SmartThings Outlet via ZigBee protocols, as shown in Fig. 8(d). It is shown that the events in IFTTT present the same wireless fingerprints as in SmartThings (e.g., $50\downarrow, 47\downarrow, 47\downarrow, 52\downarrow$ for *switch.on* or *switch.off*). The reason is that IFTTT employs the same lower-layer protocols as SmartThings and the wireless traffic patterns are not affected by their upper-layer platforms.

In addition to using common communication protocols, the IFTTT Applets also share a similar control logic (*i.e.*, if-this-then-that), and adopt the standard user interfaces and app descriptions. With these features, the proposed approach on DFA extraction from closed-source SmartApps is expected to be also applicable to the IFTTT platform. We will leave the study about other platforms (*e.g.*, Amazon Echo, Google Home) as our future work.

Potentially some device manufacturers might change the wireless traffic patterns (*e.g.*, normalizing the packet size) to prevent privacy leakage from traffic analysis. Such a practice will also affect the fidelity of HoMonit. In IoT settings, the previous literature have demonstrated the applicability of utilizing some other side-channel information (*e.g.*, power consumption [39], voltage output [11]) for misbehavior detection. In other research domains (*e.g.*, Botnet detection), there also exist a large body of advanced spatial-temporal traffic statistical analysis tools [25]. HoMonit needs to integrate with these methods to further improve its detection accuracy.

Privacy consideration. HoMonit leverages side-channel analysis to monitor SmartApps from encryption traffic. However, side-channel information leakage is a double-edged sword. It not only enables our detection of misbehaving SmartApps, but may also allows an attacker who can place a wireless sniffer in the proximity of the smart devices to launch inference attacks to learn private information of the residents.

- *Daily routines:* For example, *Good Night* is a SmartApp that changes its mode when there is no human activity in the home after some time at night time. The attacker can spy on the victim's daily activities and learn his sleeping patterns by monitoring this SmartApp's behavior.
- *Home occupations:* For example, a SmartApp named *Vacation Lighting Director* deceptively turns on/off lights while the residents are away, which, however, can be detected by inferring the existence of such a SmartApp. We found more than 15 SmartApps in our dataset leaking home occupation information.
- *Health conditions:* For example, *Elder Care: Slip & Fall* monitors the behavior of the aged people. Detecting such a SmartApp may leak the ages and the health condition of the residents.

These privacy concerns, however, may not be sufficient to motivate drastic changes in the wireless protocol design. Although enforcing packet length padding will make most wireless packets indistinguishable, the changes of the hardware design, increases in power consumption, and issues of backward compatibility are major obstacles in the practical adoption of a side-channel-resistant protocol. Moreover, the strong requirements of physical proximity in such attacks also make the threat less concerning. As such, one of the key enabling factors of HoMonit—side-channel leakage in the encrypted wireless traffic—is likely to remain unchanged in the foreseeable future.

Generating DFAs from benign SmartApps. The core idea of HoMonit is to compare the SmartApp activities inferred from the encrypted traffic with the expected behaviors dictated by their souce code or UI. Therefore, acquiring the DFA of the benign version of the SmartApp (or the groundtruth DFA) is critical for the successful deployment of HoMonit. The simplest way to obtain such groundtruth DFAs is to download it from the official app market, if assuming the market operator has performed a good job in vetting all published SmartApps. Otherwise, a trustworthy third-party must step in to vouch for benign apps, which will help bootstrap HoMonit.

Double-sending attacks. Because some device events may have the same wireless fingerprints, such as *switch.on* and *switch.off* of Samsung SmartThings Outlet (see Table 2), HoMonit has to keep track of the current state of the device, which can be used in turn to infer the content of event. However, a potential attack strategy is that a SmartApp may intentionally send the same commands twice to mislead HoMonit. We call this type of attack a *double-sending attack*. For example, a misbehaving SmartApp may send the command *switch.off* twice, hoping that they will be confused with a sequence of *switch.off* and *switch.on*. However, in reality, this double-sending attack does not work as the communication protocol of SmartThings devices is designed to deal with duplicated messages. We performed the following experiments: (1) two events [*switch.on*, *switch.off*] were sent by the SmartApp, (2) two events [*switch.on*, *switch.on*] were sent by the SmartApp. In both experiments, the initial state of the Outlet was set as *off*. The first experiment represents a normal case and the second represents a double-sending attack. As shown in Table 2, the collected traffic patterns are different: The packets in the first cases are $(50\downarrow, 47\downarrow, 47\downarrow, 52\downarrow)$, followed by $(50\downarrow, 47\downarrow, 47\downarrow, 52\downarrow)$. In comparison, those in the second are $(50\downarrow, 47\downarrow, 47\downarrow, 52\downarrow)$, followed by $(50\downarrow, 47\downarrow)$. We speculate this is because that, under double-sending attacks, when the hub receives the second *switch.on* command from the SmartApp, as the hub knows the outlet's status (*on*), it regards this command as duplicated, and alters the message sent to the outlet.

Interleaving SmartApp events. It is possible that multiple SmartApps may run simultaneously, rendering some events interleaved with one another. HoMonit can correctly identify each of the events even when they are interleaved. Moreover, in most cases, HoMonit can leverage these identified events to correctly infer the DFA transitions. One exception is that when more than one SmartApp operates the same device simultaneously. HoMonit may have difficulty in differentiating the two SmartApps. We admit this is a limitation of HoMonit. In our future work, we will investigate other means to address this problem.

Table 5: Wireless Fingerprints in normal and double-sending attack scenarios.

Case	Event	Fingerprint
Normal	1st: *switch.on*	50 ↓ 47 ↓ 47 ↓ 52 ↓
	2nd: *switch.off*	50 ↓ 47 ↓ 47 ↓ 52 ↓
Attack	1st: *switch.on*	50 ↓ 47 ↓ 47 ↓ 52 ↓
	2nd: *switch.on*	50 ↓ 47 ↓

Alerting users after detection. When detecting any misbehavior of a specific SmartApp, HoMonit can alert the users simply through text message, or work together with existing home safety monitoring tools (*e.g.*, Smart Home Monitor [62] in SmartThings) to take the immediate actions. For example, HoMonit can generate different alerts based on the detected scenarios such as home unoccupied, occupied or disarmed. In addition, HoMonit can serve as a building block for enforcing user-centric [66] or context-based [34] security policies and integrate with these previously proposed systems to interact with users.

Other issues. Incorrectly constructed DFAs from the closed-source apps will impact the detection accuracy. In practice, one potential strategy to avoid such errors is to perform additional validation before deploying HoMonit to end users. For instance, the service provider could manually trigger sensors to interact with the SmartApps and check if the observed state transition conforms to the extracted DFAs.

9 RELATED WORK

Security of smart home platforms. Closest to our work is a study by Fernandes *et al.* [19], which performed an empirical security evaluation on SmartThings framework and identified several design vulnerabilities. Follow-up studies have proposed systems to enforce information flow control [20], context-based permission systems [34], and user-centered authorization and enforcement mechanisms [66]. Demetriou *et al.* [17] employed a monitor app on smartphones and a security patch on the router to enforce fine-grained access control towards IoT devices in home area network. By comparison, HoMonit leverages side-channel inference techniques to monitor misbehaving SmartApps from encrypted traffic, without any modification of the existing infrastructure.

Security of smart home protocols. Zillner *et al.* [74] and Lomas *et al.* [40] highlighted several security risks in ZigBee implementations. Fouladi *et al.* [22] analyzed the Z-Wave protocol stack layers and discovered a vulnerability in the AES encryption in a smart lock. Weak authentication mechanisms in Bluetooth have been studied in previous studies [4, 5, 27].

Security of smart home hubs. Lemos *et al.* [36] analyzed the security of three smart home hubs: SmartThings, Vera Control [13], and Wink [70]. Their study reveals several vulnerabilities in these hubs. Similarly, several currently available IoT hubs were investigated in [24] and [69], and numerous security flaws were identified. Simpson *et al.* [54] proposed a central security manager that is built

on top of the smart home hub and positioned to intercept all traffic to mitigate security risks.

Security of IoT devices. Researchers have identified many security flaws in off-the-shelf devices for smart home. Some devices mistrust devices on the same LAN, leaving them vulnerable to a local adversary [47, 50]. Moreover, Ur *et al.* [67] studied access control systems for commercial smart devices and found that all devices lack the mechanisms of access monitoring, therefore users cannot identify who has accessed their devices. End-to-end authentication [41] and access control [35] are effective in solving this LAN mistrust issue. Besides, devices tend to suffer from emerging unauthenticated control signals. For example, an adversary can control a smart TV with a speaker playing synthetic voice commands [44]. Ho *et al.* [27] described how a Bluetooth smart lock can unlock mistakenly due to improper trust. Various defense approaches focus on device attestation [6, 27, 68]. Lastly, implementation flaws cause severe attacks against smart home devices. For example, Oluwafemi *et al.* [49] demonstrated that non-networked devices such as compact fluorescent lamps might be connected to networked devices and hence can be attacked by remote adversaries. Most of the access control solutions [16, 33, 47, 55, 72] can partially solve these implementation flaws. Different from these previous works, our method infers the working logic of the smart home based on a combination of wireless side-channel information and the characteristics of smart home platform (*i.e.*, the semantics of device handlers and SmartApps).

Wireless traffic analysis. Side-channel leaks due to network packet timing, sizes, sequences and so on have been discussed extensively in prior studies [18, 21, 26, 31, 37, 38, 43, 51, 64, 73]. Particularly, Chen *et al.* [10] found that an eavesdropper can exploit such type of side channel information to infer surprisingly detailed sensitive information from web applications despite encryption protection. Brumley *et al.* [9] presented a timing attack based on packet timing and sizes against OpenSSL that extracts RSA secret keys. Wright *et al.* [71] showed that the lengths of encrypted VoIP packets can be used to identify the phrases spoken within a call. Formby *et al.* [21] proposed two timing-based device type fingerprinting methods in industrial control system environments. Different from these works, we concentrate on the new emerging smart home platform — SmartThings.

10 CONCLUSION

In this paper, we present HoMonit, an anomaly detection system for smart home platforms to detect misbehaving SmartApps. HoMonit leverages the side-channel information leakage in the wireless communication channel—packet size and inter-packet timing—to infer the type of communicated events between the smart devices and the hub, and then compares the inferred event sequences with the expected program logic of the SmartApps to identify misbehaviors. Key to HoMonit includes techniques to extract the program logic from SmartApps' source code or the user interfaces of SmartThings' mobile app, and automated DFA construction and matching algorithms that formalize the anomaly detection problem.

ACKNOWLEDGMENT

We are grateful to our shepherd Sven Bugiel and anonymous reviewers for their constructive feedback. This work is supported by National Natural Science Foundation of China (No. 61672350, U1401253).

REFERENCES

[1] AllSeen Alliance. 2018. AllJoyn. https://openconnectivity.org/.
[2] ZigBee Alliance. 2012. ZigBee Specification. http://www.zigbee.org/wp-content/uploads/2014/11/docs-05-3474-20-0csg-zigbee-specification.pdf.
[3] Apple. 2018. HomeKit. https://www.apple.com/ios/home/.
[4] Liviu Arsene. 2014. Wearable Plain-Text Communication Exposed Through Brute-Force, Bitdefender Finds. https://www.hotforsecurity.com/blog/wearable-plain-text-communication-exposed-through-brute-force-bitdefender-finds-10973.html.
[5] AV-TEST. 2015. Test: Fitness Wristbands Reveal Data. https://www.av-test.org/en/news/news-single-view/test-fitness-wristbands-reveal-data/.
[6] Ibrahim Ethem Bagci, Utz Roedig, Ivan Martinovic, Matthias Schulz, and Matthias Hollick. 2015. Using Channel State Information for Tamper Detection in the Internet of Things. In *ACM Annual Computer Security Applications Conference (ACSAC)*.
[7] Steven Bird. 2018. Natural Language Toolkit. http://www.nltk.org.
[8] Paul Black. 2008. Levenshtein Distance. In *Dictionary of Algorithms and Data Structures*.
[9] David Brumley and Dan Boneh. 2005. Remote timing attacks are practical. In *Computer Networks*.
[10] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. 2010. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *IEEE Symposium on Security and Privacy (S&P)*.
[11] Kyong-Tak Cho and Kang G. Shin. 2017. Viden: Attacker Identification on In-Vehicle Networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 1109–1123. https://doi.org/10.1145/3133956.3134001
[12] SmartThings Community. 2015. How many SmartThings users? https://community.smartthings.com/t/how-many-st-users/10928/3.
[13] Vera Control. 2018. Vera3 Advanced Smart Home Controller. http://getvera.com/controllers/vera3/.
[14] Cybertools. 2018. Scapy Radio. https://bitbucket.org/cybertools/scapy-radio/src.
[15] Hamlet D'Arcy. 2018. AstBuilder. http://docs.groovy-lang.org/next/html/gapi/org/codehaus/groovy/ast/builder/AstBuilder.html.
[16] Nigel Davies, Nina Taft, Mahadev Satyanarayanan, Sarah Clinch, and Brandon Amos. 2016. Privacy Mediators: Helping IoT Cross the Chasm. In *ACM International Workshop on Mobile Computing Systems and Applications (HotMobile)*.
[17] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, Xiaofeng Wang, Carl Gunter, Xiaoyong Zhou, and Michael Grace. 2017. HanGuard: SDN-driven protection of smart home WiFi devices from malicious mobile apps. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*.
[18] Wenrui Diao, Xiangyu Liu, Zhou Li, and Kehuan Zhang. 2016. No Pardon for the Interruption: New Inference Attacks on Android Through Interrupt Timing Analysis. In *IEEE Symposium on Security and Privacy (S&P)*.
[19] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security analysis of emerging smart home applications. In *IEEE Symposium on Security and Privacy (S&P)*.
[20] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. 2016. Flowfence: Practical data protection for emerging iot application frameworks. In *USENIX Security Symposium (USENIX Security)*.
[21] David Formby, Preethi Srinivasan, Andrew Leonard, Jonathan Rogers, and Raheem A Beyah. 2016. Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems.. In *The Network and Distributed System Security Symposium (NDSS)*.
[22] Behrang Fouladi and Sahand Ghanoun. 2013. Security evaluation of the Z-Wave wireless protocol. In *Black Hat USA*.
[23] Google. 2018. Google Weave. https://developers.google.com/weave/.
[24] NCC Group. 2015. Internet of Things Security. https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/ncc_group_intern_whitepaper_2014_internet_of_things_research.pdf.
[25] Guofei Gu, Junjie Zhang, and Wenke Lee. 2008. BotSniffer: Detecting botnet command and control channels in network traffic. (2008).
[26] Dina Hadziosmanovic, Robin Sommer, Emmanuele Zambon, and Pieter Hartel. 2014. Through the Eye of the PLC: Semantic Security Monitoring for Industrial Processes. In *ACM Annual Computer Security Applications Conference (ACSAC)*.
[27] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. 2016. Smart Locks: Lessons for Securing Commodity Internet of Things Devices. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*.

[28] Honeywell. 2013. http://library.ademconet.com/MWT/fs2/L5210/Introductory-Guide-to-Z-Wave-Technology.pdf.

[29] IFTTT Inc. 2018. IFTTT. https://ifttt.com/.

[30] Texas Instrument. 2018. CC2531: System-on-Chip Solution for IEEE 802.15.4 and ZigBee Applications. http://www.ti.com/product/CC2531.

[31] Trent Jaeger, Xinyang Ge, Divya Muthukumaran, Sandra Rueda, Joshua Schiffman, and Hayawardh Vijayakumar. 2015. *Designing for Attack Surfaces: Keep Your Friends Close, but Your Enemies Closer.* Springer International Publishing.

[32] ABR JFR and NOBRIOT. 2016. Z-Wave Command Class Specification. http://z-wave.sigmadesigns.com/wp-content/uploads/2016/08/SDS12657-12-Z-Wave-Command-Class-Specification-A-M.pdf.

[33] Eun Kim Ji, G Boulos, J Yackovich, T Barth, C Beckel, and D Mosse. 2012. Seamless Integration of Heterogeneous Devices and Access Control in Smart Homes. In *International Conference on Intelligent Environments (IE)*.

[34] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z Morley Mao, and Atul Prakash. 2017. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *The Network and Distributed System Security Symposium (NDSS)*.

[35] Hyun Jin Kim, Lujo Bauer, James Newsome, Adrian Perrig, and Jesse Walker. 2010. Challenges in access right assignment for secure home networks. In *USENIX conference on Hot topics in security (HotSec)*.

[36] Robert Lemos. 2015. Hubs Driving Smart Homes Are Vulnerable, Security Firm Finds. In *Eweek*.

[37] Huaxin Li, Zheyu Xu, Haojin Zhu, Di Ma, Shuai Li, and Kai Xing. 2016. Demographics inference through Wi-Fi network traffic analysis. In *IEEE International Conference on Computer Communications (INFOCOM)*.

[38] Xiangyu Liu, Zhe Zhou, Wenrui Diao, Zhou Li, and Kehuan Zhang. 2015. When Good Becomes Evil: Keystroke Inference with Smartwatch. In *ACM Conference on Computer and Communications Security (CCS)*.

[39] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu. 2016. On Code Execution Tracking via Power Side-Channel. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 1019–1031. https://doi.org/10.1145/2976749.2978299

[40] Natasha Lomas. 2015. Critical Flaw IDed In ZigBee Smart Home Devices. https://techcrunch.com/2015/08/07/critical-flaw-ided-in-zigbee-smart-home-devices/.

[41] Parikshit N. Mahalle, Bayu Anggorojati, Neeli Rashmi Prasad, and Ramjee Prasad. 2012. Identity establishment and capability based access control (IECAC) scheme for Internet of Things. In *International Symposium on Wireless Personal Multimedia Communications (WPMC)*.

[42] IHS Markit. 2017. Global Smart Home Market to Exceed 14 Billion in 2017. https://technology.ihs.com/594650/global-smart-home-market-to-exceed-14-billion-in-2017.

[43] Yan Meng, Zichang Wang, Wei Zhang, Peilin Wu, Haojin Zhu, Xiaohui Liang, and Yao Liu. 2018. WiVo: Enhancing the Security of Voice Control System via Wireless Signal in IoT Environment. In *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc '18)*. ACM, New York, NY, USA, 81–90. https://doi.org/10.1145/3209582.3209591

[44] Benjamin Michele and Andrew Karpow. 2014. Demo: Using malicious media files to compromise the security and privacy of smart TVs. In *IEEE Consumer Communications and Networking Conference (CCNC)*.

[45] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems (NIPS)*.

[46] mitshell. 2018. 802.15.4 monitor. https://github.com/mitshell/CC2531.

[47] Sukhvir Notra, Muhammad Siddiqi, Hassan Habibi Gharakheili, Vijay Sivaraman, and Roksana Boreli. 2014. An Experimental Study of Security and Privacy Risks with Emerging Household Appliances. In *IEEE Conference on Communications and Network Security (CNS)*.

[48] Mikko Ohtamaa. 2018. Levenshtein Python C Extension Module. https://github.com/miohtama/python-Levenshtein.

[49] Temitope Oluwafemi, Tadayoshi Kohno, Sidhant Gupta, and Shwetak Patel. 2013. Experimental Security Analyses of Non-Networked Compact Fluorescent Lamps: A Case Study of Home Automation Security. In *LASER Workshop*.

[50] Paul. 2013. Breaking And Entering: Hackers Say "Smart" Homes Are Easy Targets. https://securityledger.com/2013/07/breaking-and-entering-hackers-say-smart-homes-are-easy-targets/.

[51] Giuseppe Petracca, Yuqiong Sun, Trent Jaeger, and Ahmad Atamli. 2015. AuDroid: Preventing Attacks on Audio Channels in Mobile Devices. In *Annual Computer Security Applications Conference (ACSAC)*.

[52] Samsung. 2018. SmartThings. https://www.smartthings.com.

[53] Kris Schaller. 2015. List of all Officially Published Apps from the MORE category of Smart Setup in the Mobile App. https://community.smartthings.com/t/list-of-all-officially-published-apps-from-the-more-category-of-smart-setup-in-the-mobile-app-deprecated/13673.

[54] Anna Kornfeld Simpson, Franziska Roesner, and Tadayoshi Kohno. 2017. Securing vulnerable home IoT devices with an in-hub security manager. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom)*.

[55] Vijay Sivaraman, Hassan Habibi Gharakheili, Arun Vishwanath, and Roksana Boreli. 2015. Network-level security and privacy control for smart-home IoT devices. In *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*.

[56] SmartThings. 2018. Capabilities Reference. http://docs.smartthings.com/en/latest/capabilities-reference.html.

[57] SmartThings. 2018. Preferences and Settings. https://docs.smartthings.com/en/latest/smartapp-developers-guide/preferences-and-settings.html.

[58] SmartThings. 2018. SmartThings Architecture. http://docs.smartthings.com/en/latest/architecture/index.html.

[59] SmartThings. 2018. SmartThings Public GitHub Repo. https://github.com/SmartThingsCommunity/SmartThingsPublic.

[60] SmartThings. 2018. Web Services SmartApps. http://docs.smartthings.com/en/latest/smartapp-web-services-developers-guide/overview.html.

[61] SmartThings. 2018. What SmartApps are being retired from the Marketplace? https://support.smartthings.com/hc/en-us/articles/115003072406-What-SmartApps-are-being-retired-from-the-Marketplace.

[62] Samsung SmartThings. 2018. Smart Home Monitor. https://support.smartthings.com/hc/en-us/articles/205380154.

[63] Statista. 2017. Smart Home Market Statistics in United States. https://www.statista.com/outlook/279/smart-home.

[64] Jingchao Sun, Rui Zhang, Jinxue Zhang, and Yanchao Zhang. 2016. VISIBLE: Video-Assisted Keystroke Inference from Tablet Backside Motion. In *The Network and Distributed System Security Symposium (NDSS)*.

[65] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2016. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.

[66] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, Xiaofeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. 2017. SmartAuth: User-Centered Authorization for the Internet of Things. In *USENIX Security Symposium (USENIX Security)*.

[67] Blase Ur, Jaeyeon Jung, and Stuart Schechter. 2013. The current state of access control for smart devices in homes. In *Workshop on Home Usable Privacy and Security (HUPS)*.

[68] Junia Valente. 2015. Using Visual Challenges to Verify the Integrity of Security Cameras. In *Annual Computer Security Applications Conference (ACSAC)*.

[69] Veracode. 2015. The Internet of Things: Security Research Study. https://www.veracode.com/veracode-study-reveals-internet-things-poses-cybersecurity-risk.

[70] Wink. 2018. Wink: A Simpler, Smarter Home. https://www.wink.com/.

[71] Charles V Wright, Lucas Ballard, Scott E Coull, Fabian Monrose, and Gerald M Masson. 2008. Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations. In *IEEE Symposium on Security and Privacy (S&P)*.

[72] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. 2015. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-Things. In *ACM Workshop on Hot Topics in Networks (HotNets)*.

[73] Zhe Zhou, Wenrui Diao, Xiangyu Liu, and Kehuan Zhang. 2014. Acoustic Fingerprinting Revisited: Generate Stable Device ID Stealthily with Inaudible Sound. In *ACM Conference on Computer and Communications Security (CCS)*.

[74] Tobias Zillner. 2015. ZigBee exploited: The good the bad and the ugly. In *Black Hat USA*.