PFIREWALL: Semantics-Aware Customizable Data Flow Control for Smart Home Privacy Protection

Haotian Chi
Temple University
htchi@temple.edu

Qiang Zeng
University of South Carolina
Zeng1@cse.sc.edu

Xiaojiang Du
Temple University
dux@temple.edu

Lannan Luo
University of South Carolina
lluo@cse.sc.edu

***Abstract*—Internet of Things (IoT) platforms enable users to deploy home automation applications. Meanwhile, privacy issues arise as large amounts of sensitive device data flow out to IoT platforms. Most of the data flowing to a platform actually do not trigger automation actions, while homeowners currently have no control once devices are bound to the platform. We present PFIREWALL, a customizable data-flow control system to enhance the privacy of IoT platform users. PFIREWALL automatically generates data-minimization policies, which only disclose minimum amount of data to fulfill automation. In addition, PFIREWALL provides interfaces for homeowners to customize individual privacy preferences by defining user-specified policies. To enforce these policies, PFIREWALL transparently intervenes and mediates the communication between IoT devices and the platform, without modifying the platform, IoT devices, or hub. Evaluation results on four real-world testbeds show that PFIREWALL reduces IoT data sent to the platform by 97% without impairing home automation, and effectively mitigates user-activity inference/tracking attacks and other privacy risks.**

I. INTRODUCTION

With the prosperity of Internet of Things (IoT), smart systems (e.g., smart homes, factories, and hospitals) have become realistic and are expanding with an ever-increasing speed [1]. IoT Platforms, such as Samsung SmartThings [2], Amazon Alexa [3], openHAB [4], allow smart home users to connect heterogeneous IoT devices (e.g., sensors, actuators, appliances) and install applications on the platform to create automatic interactions among devices, i.e., home automation.

As IoT device data flow outside, protecting user privacy becomes critical [5], [6]. Existing work protects user privacy from malicious application developers [7], [8], [9], [10], [11], [12], [13] or eavesdroppers [14], [15], [16], [17]. However, it is surprising that, while a platform receives huge amounts of privacy-sensitive data from bound IoT devices, few works regard the platform as untrustworthy and provide privacy protection solutions. In fact, it is baseless to assume the platform is trustworthy and its data access protection is flawless, and thus we should consider that the rich data may be exposed to attackers [18], [19]. Furthermore, many IoT platforms share user data with partners (e.g., advertisers) for the expansion of businesses [20], [21], [22]; any improper handling may disclose privacy-sensitive data to third parties.

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24464
www.ndss-symposium.org

To support home automation, many IoT devices continuously stream data, such as sensor events, to an IoT platform, although most of the data actually do not trigger automation actions. This deviates the principle of "*data minimisation*" in European General Data Protection Regulation (GDPR) [23]. We also find that no capabilities are provided for users to control the leakage of device data to the platform, failing to realize user-centric authorization. Therefore, we seek a privacy-enhancing system that can be used as an "add-on" into existing systems by privacy-conscious users. This system aims to 1) minimize the data sent to the platform and 2) allow users to define customizable data-protection policies for individual privacy preferences. Multiple challenges arise for attaining these goals.

Challenge 1. Data minimization should not affect home automation. We observe that the semantics of home automation apps can be represented as rules in a *trigger-condition-action* programming paradigm (code analysis [24], [25], [10] and natural language processing [10], [26], [27] have proven effective in extracting rules from apps). Our insight is that, according to the rule semantics, we could reduce data leakage without impairing automation. For example, given a rule "*when a motion is detected (***trigger***), if the indoor temperature is higher than 79°F (***condition***), turn on the A/C (***action***)*," we could derive data-minimization policies, such that 1) if the temperature is not higher than $79°F$, no motion or temperature data are sent to the platform; 2) if the A/C is already on (that is, the rule execution will not change anything), no data need to be sent, and 3) otherwise, the temperature and the motion data must be sent, but the temperature value can be obfuscated to a random value larger than $79°F$.

Challenge 2. Many platforms are closed systems that do not allow platform-level modifications, and it is probably unrealistic to expect a platform to cooperate to enforce data minimization. Thus, how to enforce data-protection policies in closed systems is a challenge. One may propose to *avoid* this challenge by building a new purely-local platform, such that no data have to flow out of a home, or one can simply cut the network cable of a local gateway [28] and enforce the home automation locally. However, most of the leading platforms (IFTTT, SmartThings, Google Home, Amazon Alexa) employ a cloud-based architecture due to its advantages in many aspects such as storage, integrated services, management. Thus, existing users may not be willing to give up their choice of platforms.

Inspired by the concept of firewalls [29], [30], we design

PFIREWALL as a mediator, which sits between IoT devices and the hub (or the platform backend cloud) to transparently filter data based on privacy-protection policies. However, our PFIREWALL is significantly different from the traditional firewalls. First, the "trigger-condition-action" paradigm of IoT automation rules is different from that of traditional firewall rules. Second, PFirewall utilizes application-layer data and performs semantics-aware data filtering, while traditional firewalls do not use application-layer payload data. There is another challenge for implementing PFIREWALL: the original communication between IoT devices and the hub (or the platform backend cloud) is encrypted, which prevents PFIREWALL from understanding it and then filtering data. We overcome this challenge with a *virtualization* approach: On one hand, the data mediator acts as a hub to pair with all the IoT devices. On the other hand, PFIREWALL creates a virtual device (for each real IoT device), which connects with the hub (or platform backend cloud). This way, neither the IoT devices nor the platform needs to be modified.

We demonstrate the ideas by implementing PFIREWALL on three representative IoT platforms: *SmartThings classic*, *the new SmartThings*[1] and *openHAB*, where the former two are *cloud-based* IoT platforms and the third a *gateway-based* platform. We evaluate PFIREWALL in four real-world testbeds. The results show that PFIREWALL reduces the amount of data sent to the platform by 97%, without affecting home automation. Our performance evaluation also shows that the data reduction severely impairs the attacker's ability to infer and track privacy-sensitive behaviors, such as bathroom usage, home occupancy, etc. An earlier version of this paper was posted on *arXiv* in October 2019 [31].

The contributions of this work are summarized as follows.

- We design an effective data flow control system to protect user privacy in home automation. Data-minimization policies are automatically generated based on automation apps, reporting only the needed data for app execution (minimizing data out). Moreover, we provide users an easy-to-use tool to define and customize their own privacy policies (e.g., *"during the sleep mode, no data should be sent out from devices in bedrooms"*).
- We overcome the challenges that most IoT platforms and IoT devices are closed-systems and cannot be modified to support the proposed data filtering, and implement our solution on the standard wireless communication protocols including ZigBee, Z-Wave and WiFi.
- We implement a prototype to work with various IoT devices and three popular platforms: SmartThings classic, the new SmartThings and openHAB. Through the evaluation in four real-world testbeds, we demonstrate that our system significantly reduces the privacy risks due to data leakage and it causes very small latency to home automation.

II. BACKGROUND: SMART HOME PLATFORMS

Smart home platforms can be categorized into cloud-based platforms (CBPs) and gateway/hub-based platforms (GBPs),

[1]SmartThings supported the two major versions at the time of research. Although the SmartThings classic mobile app was discontinued in October 2020, most features concerned in this paper such as SmartApps are now included in the new system.

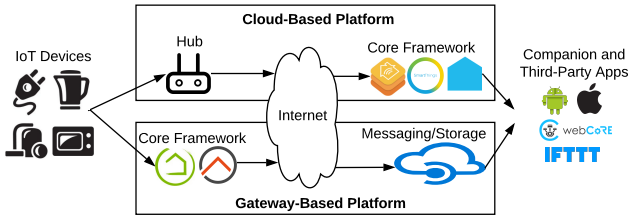


Fig. 1: Smart home platform architecture.

according to whether the core framework of a platform is hosted in a remote cloud or a gateway/hub device located at home (as shown in Fig. 1). Note that the gateway running a core framework at home does not resolve the privacy leakage threats completely, as the gateway connects to the Internet and is under the full control of the platform cloud administrator. We choose a CBP—*SmartThings*, one of the most popular and full-fledged platforms, as an example to describe the key components in a smart home system.

- **Hub.** A CBP hub connects IoT devices through distinct short/medium-range wireless radios (ZigBee, Z-Wave, etc.). The hub plays a key role to ensure the interconnectivity and interoperability of heterogeneous IoT devices. A GBP also has a hub-like device[2] which not only connects IoT devices but also hosts the core framework (described below). Note that the hub or gateway device, though physically located at home, is conceptually regarded as a part of the platform in terms of data privacy protection in that it is under the full control of the platform cloud administrator.
- **Cloud.** The backend cloud of a CBP hosts the core framework and provides cloud messaging, storage and other necessary services. The cloud in a GBP is typically responsible for messaging and storage. The cloud messaging service facilitates some critical functionalities, such as notification, third-party application integration, remote monitoring and control. Many Internet-based services depend on the cloud.
- **Core Framework.** The core framework runs major functionalities of a platform, including home automation. SmartThings classic provides a sandboxed runtime environment for running *device handlers* and *SmartApps*. Device handlers are software wrappers that abstract the physical devices as a set of *capabilities* and they handle the underlying protocol-specific communications between the core framework and the physical devices. The core framework represents each physical device as a *device instance* by instantiating the corresponding device handler. Automation rules can be defined by installing SmartApps or configuring the rule-creating interfaces on the companion mobile app.
- **Companion and Third-Party Apps.** To provide a convenient user interface (UI) for users to manage their hubs, IoT devices and apps, a platform usually provides a smartphone *companion* app. For instance, in the SmartThings companion app, users can install and configure a SmartApp.

III. MOTIVATION AND THREAT MODEL

In this section, we first discuss two privacy concerns that motivate this work, and then present the threat model.

[2]We use *gateway* for distinguishment.

A. Privacy Concerns about Platforms

1) Trust By Default: In smart home systems, the platforms are typically fully trusted. After being installed, a platform gains the access privilege to all connected home IoT devices technically by design and legally, by claiming a *terms and conditions* or a *privacy policy*. After that, IoT platforms receive rich data from connected devices, no matter whether the data are required or not for providing services. To demonstrate this fact, we conduct an experiment on an exemplar platform– SmartThings.

Are IoT data continuously flowing out of homes? To answer this question, we connected four ZigBee devices (a multipurpose sensor, a motion sensor, an arrival sensor and a smart outlet) and a Z-Wave sensor (Aeotec Multisensor 6) to a SmartThings hub and observe the data received by the SmartThings cloud on its logging interface [32]. We did *not* install any automation apps and did *not* operate any SmartThings-provided interfaces. We only interacted with the IoT devices physically. We found that the platform cloud kept receiving device data (e.g., motion, switch, temperature, etc.) from devices, indicating that device data flow out via the hub *even when they are not needed by any automation*. Besides our experiment, traffic analysis researches [33], [34] also show that certain traffic patterns for transmitting device events from the SmartThings hub to the backend cloud can be observed when the corresponding events are generated by devices, which is consistent with our result.

Reasons for this fact include: (1) to enhance interoperability, IoT devices are designed to simply send out all data to the paired hub or cloud for further processing, but unaware of how the data will be consumed; (2) most platforms support more than one service (e.g., home automation, dashboards for viewing and changing device states), and therefore do not deploy a data filtering component on the hub devices (manufactured by them) or at the front-end to filter out data that are not needed for specific services. However, we argue that the system could be improved or patched to enhance user privacy in different services. In this paper, we focus on enhancing user privacy in the home automation service, which is context-aware and more difficult to address than other services. We also discuss a possible solution for the dashboard service in Section VIII.

2) Limited User Capabilities: Currently, users have few (or no) capabilities to control device data sent to a platform [35]. They only have a binary choice: either connect a device to the platform or not; once connected, the device continuously reports data to the platform.

B. Threat Model

We consider that a smart home platform may be exploited by attackers for accessing user private data and inferring user privacy-sensitive behaviors. For example, an attacker may find a way to gain access to the data; also the platform administrators may be curious to mine user privacy. We do not assume that the platform is attacker-controlled, but aim to handle the concern that the rich data may be exposed to attackers and exploited for malicious purposes. The platform is assumed to be honest, i.e., it faithfully operates all services and does not perform active attacks on users. Attacks that exploit vulnerabilities of home IoT devices [36], [37], side channels [14], [17], [38], or home local networks [39], [40] to steal private data are not within the scope of this work.

Malicious IoT apps may request more device data than what is needed by the claimed functionality in the app descriptions. Detecting and preventing such malicious apps have been well studied, e.g., [41], [10]. In this work, we assume that this kind of malicious apps have been inspected by an existing solution such as [10]. Moreover, many platforms (e.g., SmartThings, Amazon Alexa) allow users to create rules on mobile app interfaces, largely eliminating such attacks.

PFIREWALL itself contains a firewall that only allows traffic from whitelisted sources (attacks targeting PFIREWALL are further discussed in Section VIII).

IV. SYSTEM OVERVIEW

To protect data privacy, IoT data are processed before they leave a smart home. In this work, we propose a privacy-preserving data minimization approach that can guarantee the correctness and completeness of the desired home automation, satisfy personal privacy preferences, and significantly reduce the amount of data sent to the platform. In order to achieve these goals, three challenges need to be addressed.

First, data protection must not accidentally affect the correctness and/or completeness of home automation (Section V-A1). Second, it is challenging to enforce the data protection without modifying the IoT devices, hub, or platform framework which are usually closed systems (Section V-B). Third, it is non-trivial to provide user-friendly interfaces for non-expert users to define their own privacy-protection policies (Section V-A2).

Most IoT devices can be categorized as the following two cases (Section V-B1 presents our survey of commercial IoT devices): (1) IoT devices use open-source wireless technologies (e.g., ZigBee, Z-Wave, and Bluetooth) to connect with an in-home hub, and (2) IoT devices use WiFi to connect, in an end-to-end encryption manner, with their *vendor cloud*, which then bridges IoT devices to IoT platforms through encrypted channels. In case (1), PFIREWALL can act as the hub to directly connect with IoT devices, and hence can see the packet contents from the devices. In case (2), PFIREWALL cannot act as a hub to directly talk with these devices but instead could access the devices via the REST (REpresentational State Transfer [42]) APIs provided by the vendor cloud. With the above observations, we propose to place a *mediator* between IoT devices and the platform to intervene in the communication between them. The mediator makes it possible to process the *raw* IoT data before forwarding them to the platform. With this insight, we design PFIREWALL, a system that can enforce strong privacy protection and user privacy preference, while not affecting the home automation at all.

As shown in Fig. 2, PFIREWALL comprises these modules:

- **Device Connector** talks with IoT devices directly via device-dependent protocols such as ZigBee, Z-Wave, or indirectly via vendor cloud APIs (see Section V-B1). It also forwards the raw device data to the *policy-based data filter* for processing.

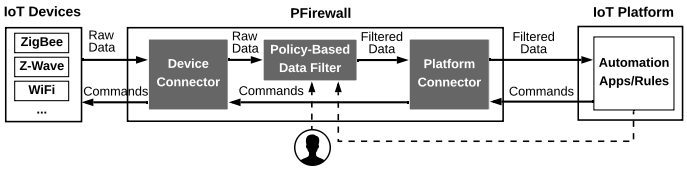


Fig. 2: Architecture of PFIREWALL. Solid arrows: device data/command flows; dotted arrows: inputs for generating policies. PFIREWALL can automatically generate policies from apps, so user-specified policies are ***optional***.

- **Platform Connector** interacts with the platform through specific connectivity technology supported by the platform, on behalf of the physical devices connected to the *device connector* (Section V-B2). The *device connector* and *platform connector* collaboratively mediate data between IoT devices and the platform.
- **Policy-Based Data Filter** sits between the device connector and platform connector to filter the sensitive data (i.e., events) from IoT devices based on policies. It has three main components: *policy generator*, *conflict detector* and *policy engine*. The *policy generator* generates data-filtering policies in two ways: (1) it takes automation rules as input to generate data-minimization policies (Section V-A1) and (2) it transforms user-specified policies (from the user interfaces) into executable-formatted policies (Section V-A2). *Conflict detector* inspects if a user-specified policy conflicts with existing data-minimization policies and reports conflicts to users (Section V-A3). *Policy engine* interprets and executes all policies.

Note that while PFIREWALL filters device events before reporting them to the platform, it does not filter commands, i.e., all commands received from the platform by the platform connector are forwarded to the physical devices via the device connector.

V. DESIGN AND IMPLEMENTATION

In this section, we present the design and implementation details. We first describe the policy-based data filter for contextually controlling IoT data flows (Section V-A). Then, we present the mediator for enforcing policies in existing IoT systems (Section V-B). To demonstrate the applicability of PFIREWALL, we show how PFIREWALL integrates with three popular platforms: SmartThings classic, the new SmartThings and openHAB (Section V-B2).

A. Data Filtering Policies

PFIREWALL filters data based on two types of policies: automation-dependent data-minimization policies (APs) and user-specified policies (UPs). To achieve data minimization, i.e., only reporting the minimum amount of data that are necessary for home automation, automation rules are analyzed to generate the corresponding APs. UPs are generated from a user interface provided by PFIREWALL and work with APs simultaneously, which is an important supplement to customize privacy preferences that cannot be learned from home automation.

1) Automation-Dependent Data-Minimization Policies: Home automation systems run user-customized *trigger-condition-action* rules, each of which can be expressed as "*when [trigger], if [condition], then [action]*" [43], [44]. Each rule runs reactively, i.e., it reacts when its *trigger* is satisfied by a new event (termed as *trigger event*) and then executes *action* if the smart home is under the prescribed *condition*. Note that the trigger and condition are different: a trigger describes a constraint that checks against the new events, while a condition includes a set of constraints that check on the static states (e.g., current device states, time, etc.). An example rule $\mathbf{R}_1$ is "*when a presence sensor ps_1 becomes present (trigger), if the reading of a temperature sensor ts_1 is higher than 86°F (condition), then turn on the fan f_1 (action)*." Some automation rules follow *trigger-action*, i.e., "*when [trigger], then [action]*", which is a special case of the *trigger-condition-action* paradigm where *condition* is always true.[3] For example, a trigger-action rule $\mathbf{R}_2$ is defined as "*when the reading of a temperature sensor ts_1 is higher than 86°F, then turn on the fan f_1*". Without loss of generality, we will use the rule $\mathbf{R}_1$ as a running example to present how PFIREWALL filters data.

Automation Rule Extraction. Extracting automation rules from rule-creating interfaces is the first step for AP generation. Automation rules are generated by installing IoT apps or using rule templates on web/mobile app interfaces. The rule extraction regarding both methods has been widely studied by state-of-the-art literature. Code analysis has been shown to be an effective way to extract rule semantics from IoT apps. For example, by utilizing Abstract Syntax Tree (AST) analysis on SmartApps, [45] identifies requested and used capabilities in SmartApps, [10], [27] break down SmartApps and extract rule information, [46], [26], [11] build Deterministic Finite Automatons (DFAs) from SmartApps. Symbolic execution is a powerful technique to analyze rule semantics from apps [24], [25]. Text data crawling and natural language processing have been used for rule extraction from mobile apps and web pages [26], [47]. Rather than develop new tools, in this paper, we adapt the solution provided in [24] to extract rules from SmartThings classic and manually encode rules defined in the new SmartThings and openHAB for which we are unable to find an open-source implementation. We envision that more rule extraction tools will be developed and made publicly available, which can eliminate the manual efforts for encoding rules.

Policy Generation. Consider the example rule $\mathbf{R}_1$. By default, the event streams from devices (presence sensor, temperature sensor, fan) are reported to the platform for executing $\mathbf{R}_1$. However, we observe that these data are not all required for executing $\mathbf{R}_1$ in cases:

(1) The presence sensor ps_1 does not send any event;
(2) ps_1 sends a "not present" event;
(3) The temperature measured by ts_1 is lower than 86°F;
(4) The fan f_1 is "ON";
(5) ps_1 sends a "present" event and the last reported temperature by ts_1 is higher than 86°F.

In cases (1)-(4), there is no need to report any data from ps_1 and ts_1, as well as data from other devices if they are not used in other rules. In case (5), it is unnecessary to report the temperature data since the temperature value

[3]To avoid confusion, we explicitly use *when* and *if* to distinguish trigger and condition, respectively.

```
TRIGGER:{                                                    1
   match  (:type).(:subject).(:attribute)                    2
   satisfy  (:operator)->(:value)                            3
   [fetch*]  (:type).(:subject).(:attribute*)                4
   [branch]  (:operator1)->(:value)                          5
   run  (:method)(:paras)(:delay)                            6
   [else]  (:method1)(:paras1)(:delay)  }                    7
CHECK: [{                                                    8
    fetch  (:type).(:subject).(:attribute)                   9
   satisfy  (:operator)->(:value)                            10
   [fetch*]  (:type).(:subject).(:attribute*)                11
   [branch]  (:operator)->(:value)                           12
   [run]  (:method)(:paras)                                  13
   [else]  (:method1)(:paras1)  }, ...]                      14
```

Listing 1: Context-aware policy format

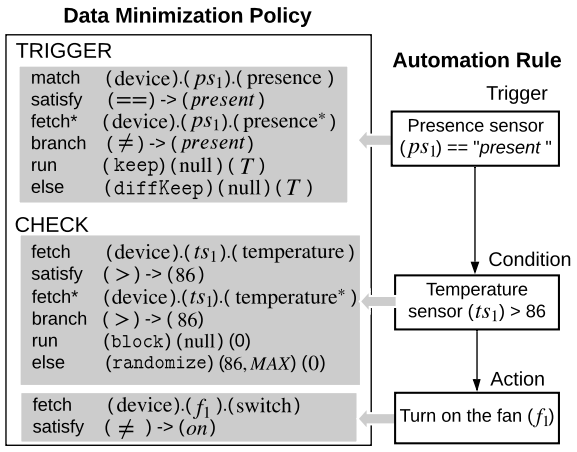


Fig. 3: Policy derivation from an automation rule.

stored in the platform database (i.e., the last reported value) satisfies the rule condition checking. In no cases, the ON/OFF state of f_1 is useful for executing $\mathbf{R}_1$. In addition, we can report a random temperature value that is higher than $86°F$ instead of the real one when reporting a temperature value is necessary for rule execution. From this example, we can see that only a small portion of the device data are required for home automation, which motivates us to run data-minimization policies that can significantly reduce the amount of data going out while not affecting any automation rules. Given the unique trigger-condition-action paradigm, the existing access control policy parsers cannot be directly used. Therefore, we develop a customized context-aware policy format and a corresponding policy engine for the trigger-condition-action paradigm.

Formally, we define a data flow policy as $\mathbf{P}$=(T, C), where T and C denote the TRIGGER and CHECK section in a policy, respectively, as shown in Listing 1. Fig. 3 illustrates an policy instance that is derived from the running rule example $\mathbf{R}_1$. TRIGGER defines the incoming event that awakens the execution of $\mathbf{P}$ and CHECK encapsulates a list of items, each of which contains a constraint that must be satisfied for the policy to indeed perform actions. TRIGGER is derived from the Trigger of an automation rule (e.g., $\mathbf{R}_1$) and CHECK is derived from the Condition and Action of the same rule.

type indicates that a datum is related to a device or to time; subject is used to identify a specific IoT device (i.e., device ID); attribute specifies the attribute of a device (a device may have multiple attributes) or a time-related feature (e.g., time of day, date, timer). Fields type, subject and attribute are used to uniquely identify an event type (e.g., an event of a device ps_1's presence) or a state type (e.g., the reading of a device ts_1's temperature). Hence, these fields in TRIGGER indicate that only data of the specific event type (e.g., ps_1's presence) trigger the execution of the policy $\mathbf{P}$; similarly, these fields in CHECK indicate what states (e.g., ts_1's temperature and f_1's switch) $\mathbf{P}$ should query for constraint satisfaction checking. Fields operator and value define a constraint to the event (ps_1's presence is "present") or state (ts_1's temperature is higher than 86) in the same block. The policy $\mathbf{P}$ will take actions only when all the constraints defined in TRIGGER and CHECK are satisfied; if any constraint is not satisfied, the policy execution aborts and will not perform actions. Recall the aforementioned cases (1)-(5). The policy in Fig. 3 only needs to report data for ensuring the correct execution of $\mathbf{R}_1$ when a new "present" event is observed from ps_1 and at the same time ts_1's current temperature is higher than 86 and f_1 is off.

TABLE I: Summary of methods used in data flow policies

Method	Description
keep()	Report the original value
block()	Do not report
diffKeep(v)	Report a different value ($\neq$v) and then value v
randomize(v1,v2)	Report a random value $\in$ (v1,v2)
startTimer(id)	Create or reset a timer with identity id
stopTimer(id)	Stop and reset a timer with identity id
fireTimer(id)	Fire a timer id and execute actions in its callbacks
addCallback(id,act)	Add an action act to the callbacks of timer id

Policy actions defined in run or else indicate how to report data to the platform. method and parameters define how to process the corresponding raw data and delay controls the timing for reporting the processed data to the platform. Table I shows a summary of all the supported methods. In the default setting, binary-value sensors such as the presence sensor reports binary values alternatively; thus, the platform only fires an event when observing a value change. Our data filtering may affect the alternation of "present" and "not present" values in the data stream of ps_1. When the platform receives "present" but finds the last received value is also "present," it will not fire a new "present" event in its framework and thus $\mathbf{R}_1$ cannot be triggered. The policy uses diffKeep() to address this issue; diffKeep() reports "not present" followed by "present" with a time delay T, which ensures a "present" event is fired on the platform[4]. However, if the last reported value is "not present", the policy only needs to use the method keep() which reports the received value "present". To handle the above situations in TRIGGER, fetch* is introduced for the policy to query the last reported value.[5] The policy takes the action defined in run if the constraint defined by branch is satisfied; otherwise, the action in else is performed. Fields marked with "[ ]" are optional in the policy format.

fetch* and else are also introduced in CHECK items to handle similar situations. If the last reported temperature value to the platform is higher than 86, the policy does not need to report the newly queried temperature value to the platform (using block()) since $\mathbf{R}_1$'s condition-checking

[4]We fine-tuned the value of T and found a sweet spot as small as 300 millisecond without causing failure in the three platforms.

[5]Different from attribute which represents the current value of a device attribute, attribute* denotes the latest reported value of this attribute.

yields the correct result based on the last reported value. If the last reported temperature is equal to or lower than 86, the policy must report a temperature event higher than $86°F$ before reporting a "present" event to make sure the rule $\mathbf{R}_1$ is correctly executed. Instead of reporting the real temperature value, the policy uses `randomize(86, MAX)` to report a random value. In $\mathbf{R}_1$, the temperature compares with a threshold ($86°F$), so a random value between $86°F$ and the upper limit of a temperature `MAX` is sufficient for the condition checking. `MAX`/`MIN` denotes the upper and lower boundary values of a specific attribute.[6] We obtain such information from SmartThings Capabilities Reference [48].

With the above design, the policy derived from $\mathbf{R}_1$ (see Figure 3) expresses multi-faceted information for PFIREWALL to process data:

1) Context: when and only when an incoming event of ps_1 is "present", the current reading of ts_1 is higher than $86°F$, and the state of f_1 is not "ON," *some* data will be reported. Otherwise, the policy will be skipped and no data will be reported at all.
2) `TRIGGER` data reporting: if the latest reported value of ps_1 is "present," use the `diffKeep()` method to process the current value for reporting; otherwise, use `keep()`.
3) `CHECK` data reporting: if the latest reported value of ts_1 is higher than $86°F$, use the `block()` method to process the current value of ts_1; otherwise, use `randomize(86, MAX)`.

We implement a policy engine to parse and execute the policies. The policy engine listens to all the incoming raw data from the IoT devices and time-related information if registered. When receiving a new data datum `D`, the engine uses `D` to evaluate the maintained policies one by one. Algorithm 1 summarizes the general workflow of how the engine evaluates and executes a policy $\mathbf{P}$. Specifically, it first checks if `D` matches the `type`, `subject`, and `attribute` in `TRIGGER`, and then examines if the value of `D` satisfies the constraint specified by `operator` and `value`. If true, $\mathbf{P}$ is triggered and proceeds to execution. Then the engine evaluates all items specified in `CHECK`. Since the data required for evaluating the `CHECK` items are not new events but the current smart home status (e.g., the device working status), the policy engine fetches the information indexed by `type`, `subject` and `attribute` from a database `DB`, which stores the current states of all connected devices and updates them when devices report any change. Only when constraints defined in all `CHECK` items are satisfied, the actions defined in all `run` or `else` fields will be performed. During the above process, a policy terminates if there is any event mismatch or constraint violation. Besides, the policy engine also maintains another database $\texttt{DB}^*$ to keep record of the last reported data for each device attribute.

PFIREWALL also deals with time-related automation. For instance, if a rule is defined as "when the door is opened if time is after 18:00, turn on TV", the derived policy needs to fetch system time for condition checking. When a rule has a timer, e.g., "when motion sensor becomes inactive for 5 minutes, turn off the light", multiple policies are bundled to operate by calling the methods for starting, stopping and firing a timer.

[6]For instance, temperature (-460∼10000 °F), humidity (0∼100%), luminance (0∼100000 lux).

Algorithm 1: The algorithm for executing a policy

```
Input : D ← new datum, P ← A privacy policy
        DB ← Newest Device Status Database
        DB* ← Newest Reported Data Database
Output: Reported Data Set DS
if match(D.source, P.TRIGGER.(type, subject, attribute)) and
   satisfy(D.value, P.TRIGGER.(operator, value)) then
    foreach checkitem ∈ P.CHECK do
        val ← fetch(DB, checkitem.(type, subject, attribute))
        if !satisfy(val, checkitem.(operator, value)) then
            return
        if !checkitem.hasField([run]) then
            continue
        if checkitem.hasField([branch]) then
            val* ←fetch(DB*,
              checkitem.(type, subject, attribute))
            if satisfy(val*, checkitem.(operator, value)) then
                DS ← run checkitem.(method, paras)
            else
                DS ← run checkitem.(method1, paras1)
        else
            DS ← run checkitem.(method, paras)
    if P.TRIGGER.hasField([branch]) then
        val* ←fetch(DB*, P.TRIGGER.(type, subject, attribute))
        if satisfy(val*, P.TRIGGER.(operator1, value)) then
            DS ← run P.TRIGGER.(method, paras)
        else
            DS ← run P.TRIGGER.(method1, paras1)
    else
        DS ← run P.TRIGGER.(method, paras)
```

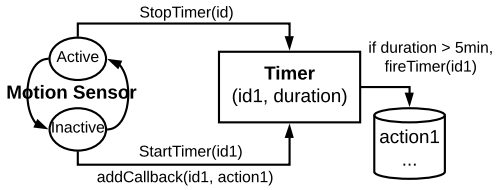
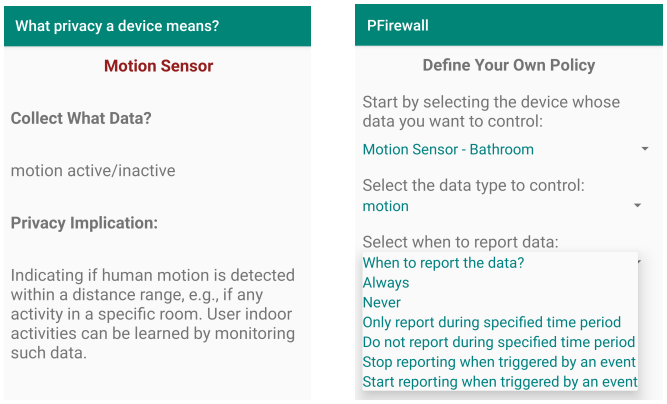


Fig. 4: An example showing how PFIREWALL handles a rule whose condition has a timer. The methods are shown in Table I. `action1` is defined to report "inactive" to the platform with method `keep` or `diffKeep`. Each timer maintains a list of actions which will be called when the timer fires. To accommodate, the timer in the rule is removed to avoid doubling the timer.

Fig. 4 illustrates the workflow of how PFIREWALL handles this example.

The policy design supports all the rule features (e.g., state-checking, timers, delayed actions) employed by most popular platforms such as HomeKit, SmartThings, Alexa, Google Home, IFTTT, etc. However, the policy design currently does not support some uncommon automation rule semantics that are rarely seen. For instance, if a rule's trigger or condition checks historical values of a device (e.g., "*if the precipitation in the past seven days is less than 100ml (*trigger*), turn on the sprinkler system (*action*)*"), PFIREWALL will simply let all the values pass to the platform to ensure the correctness of rule executions. The policy design of PFIREWALL is extendable to support such rules when needed.

2) User-Specified Policies: We propose an interactive approach for users to specify data-protection policies. This is motivated by three reasons: 1) users have individual privacy preferences that cannot be derived from automation rules; for example, users might prioritize privacy rather than automation functionality for some device types during a time period or under certain situations; 2) the platform may integrate a third-party service but there is no rule extractor available to extract semantics from it; 3) users have rights to control the use of



(a) (b)

Fig. 5: Screenshots of PFIREWALL mobile app. The app provides an information tab showing users what data every device type generates and the corresponding privacy implications, and a policy tab allows users to define context-aware data control policies.

their data. In principle, UPs have higher priority than APs in controlling data.

We develop a mobile app for end-users to specify policies. As shown in Fig. 5(a), information is displayed to help users understand what privacy issues each device and its data may imply. With the templates in Fig. 5(b), users are able to configure whitelist, blacklist and conditional-style policies during a specified time period or under certain contexts. Finally, UPs are encoded into the policy format in Listing 1 for execution.

3) Policy Conflicts: It is possible that a user's UPs conflict with existing APs and hinder the automation since UPs are designed to override APs. Nevertheless, users need a warning that shows them what conflicts are caused and which automation rules are affected. Therefore, an automated policy conflict detection is needed. Two policies $\mathbf{P}_1$ and $\mathbf{P}_2$ *conflict* with each other if all the following requirements are satisfied: (1) $\mathbf{P}_1$ and $\mathbf{P}_2$ are triggered simultaneously, i.e., an event makes both constraints c_T^1 and c_T^2 (defined in `TRIGGER` fields of $\mathbf{P}_1$ and $\mathbf{P}_2$, respectively) hold; (2) both policies are finally executed, i.e., all the constraints c_i^1 and c_i^2 in the `CHECK` fields of both policies are evaluated true; (3) two policies define different actions (i.e., data processing methods, parameters, or delays) for the same data. Formally, let $\mathbf{S}(C)$ denote the set of all possible contexts that satisfy the set of constraints C, and $\mathbf{O}(a)$, $\mathbf{E}(a)$ denote the object (i.e., the controlled data) and effects of a certain action a (defined in both `TRIGGER` and `CHECK` fields). A conflict occurs when the formulas hold.

$$\begin{cases} \mathbf{S}(c_T^1) \cap \mathbf{S}(c_T^2) \neq \emptyset, \\ \mathbf{S}(c_1^1, c_2^1, \cdots) \cap \mathbf{S}(c_1^2, c_2^2, \cdots) \neq \emptyset, \\ \exists i, j, \mathbf{O}(a_i^1) = \mathbf{O}(a_j^2), \mathbf{E}(a_i^1) \neq \mathbf{E}(a_j^2). \end{cases}$$

PFIREWALL detects policy conflict for each newly submitted/updated UP against every AP. When an AP is added/updated due to automation rule adding/updating, PFIREWALL also detects it against all existing UPs. To verify the first two formulas, we encode each constraint in a policy into a quantifier-free first-order formula:

$$\underbrace{(\texttt{type}[.\texttt{subject}[.\texttt{attribute}]])}_{\text{data source and type}}(\texttt{operator})(\texttt{value}).$$

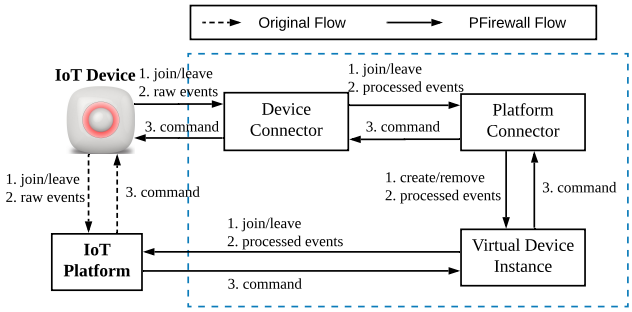


Fig. 6: Workflow of the mediator.

To verify the last formulas, we check whether the two policies perform contradictory actions (by looking at the methods and parameters in `run` and `else` fields) on the same data.

Thus, the conflict detection is transformed into a constraint satisfaction problem that can be solved by a constraint programming solver. In our implementation, we use a JavaScript linear solver `javascript-lp-solver` [49]. If the constraint set derived from two policies is not solvable, it means the two policies have no conflict. Otherwise, two policies have a conflict. PFIREWALL presents detected conflicts to users for decision making.

The solving complexity is determined by the constraints derived from policies, which only involve simple numerical relationships and the search space (e.g., the range of device values, the number of devices in a policy) is very small. In our evaluation, it takes less than 400 milliseconds to check a pair of policies and we did not encounter a failure due to timeout.

B. Data Flow Mediation

To enforce data flow policies in a closed-source IoT system, we introduce a *mediator* for relaying communications between IoT devices and the IoT platform, as shown in Fig. 6. The mediator needs to act as both a device connector to interact with IoT devices and a platform connector, which generates a virtual device on behalf of each real device, to interact with the target platform. An advantage of our design is that the mediator PFIREWALL does not modify the IoT devices, hubs, or the platforms.

1) Connecting IoT Devices: We investigate the communication technologies used by IoT devices in the market, i.e., the top 1000 IoT devices with the most total reviews on five popular online stores (Amazon, BestBuy, Costco, Walmart, and Home Depot). Our investigation focuses on two important questions: (1) the portion of IoT devices that use each of the popular wireless technologies and (2) among the devices using end-to-end encryption technologies (such as WiFi) to connect with endpoint/vendor clouds, how many of them provide cloud APIs for accessing devices. The result is presented in Fig. 7. Currently, PFIREWALL cannot handle IoT devices that use proprietary or end-to-end encrypted connections but do not provide accessing APIs. However, we envision that more IoT cloud endpoints will provide open APIs in the future to increase interoperability and facilitate service integration. To demonstrate the feasibility, we choose two open-source

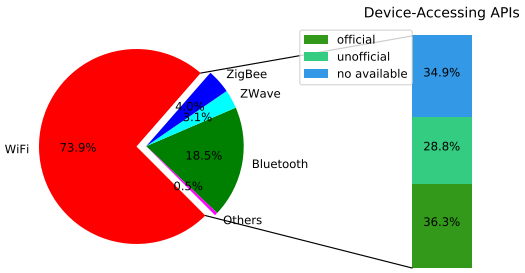


Fig. 7: Statistics of popular communication technologies employed by commercial IoT devices and statistics of availability of device accessing APIs of WiFi devices. "Others" include devices that use other technologies including Clear Connect RF [50], HAP [51], GSM [52]. PFIREWALL could handle over 74% of the investigated devices (including WiFi-connected devices with unofficial APIs).

standard protocols (ZigBee, Z-Wave) and the most common end-to-end encrypted protocols (WiFi connection with the vendor clouds which provide REST cloud APIs).

To play the role of a device connector, the mediator needs to handle 3 major tasks associated with IoT devices: 1) adding or removing devices to PFIREWALL; 2) receiving events from devices; and 3) sending commands to devices. The device connector alike functions have been implemented and are available in many open-source platforms, e.g., openHAB [4] and Mozilla IoT [28], which allow developers to add add-ons for integrating various IoT devices using different communication techniques. At the time of research, openHAB supports 275 bindings that have been tested to work with hundreds of commercial IoT devices and Mozilla IoT also has tested more than 100 mainstream devices. In our implementation, we utilize the ZigBee and Z-Wave add-ons of Mozilla IoT to realize connecting with ZigBee and Z-Wave devices. Specifically, the mediator is built on a Raspberry Pi with a Digi XStick USB dongle (ZB mesh version) and an Aeotec Z-Stick (Gen5) to extend ZigBee and Z-Wave capabilities, respectively. For WiFi devices, we use the REST APIs provided by three vendor clouds (Philips Hue, LIFX, and Honeywell) to access devices connected to them.

2) Connecting Various Platforms: When a physical device is connected to PFIREWALL, the platform connector creates an instance of *virtual device* for it. The virtual device interacts with a platform on behalf of the physical device, i.e., it (1) is discovered by the platform as a new *device instance*, (2) reports the post-filtering events to the platform, and (3) forwards commands from the platform to the physical device (via the device connector). Popular platforms support various connectivity protocols; for example, SmartThings classic supports LAN- and cloud-based device integration [53], the new SmartThings supports cloud-connected devices [54], and openHAB supports the Message Queuing Telemetry Transport (MQTT) protocol [55], Mozilla IoT provides REST-based Web Things framework and APIs [56], and Wink allows creating REST API devices [57]. These features allow a virtual device instance created by the platform connector to talk with the platform as if it is a real device. We implement the mediator to work with three representative systems with different interfacing techniques: SmartThings classic, new SmartThings, and openHAB, without modifying the software of these platforms or the hubs manufactured by them.

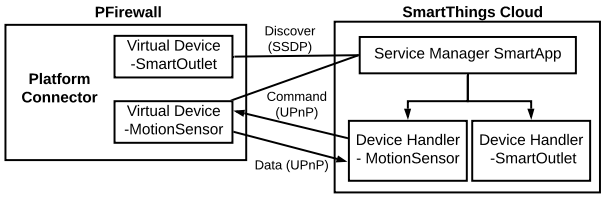


Fig. 8: Overview of interfacing with SmartThings classic. Note that the communication between PFIREWALL and the SmartThings cloud is bridged by a SmartThings hub, which is omitted here.

Interfacing with SmartThings Classic System

We choose LAN as the protocol for communicating with the SmartThings classic system. SmartThings classic uses a `device handler` (DH) for abstracting each supported device type. Accordingly, we build a virtual device (VD) type for each DH, as illustrated in Fig. 8. We develop a `service manager` SmartApp on SmartThings, which uses SSDP (Simple Service Discovery Protocol) to discover VD instances on the same LAN as the SmartThings hub. SmartThings classic uses a combination of IPs and ports to uniquely identify devices. To be discovered as different devices, each VD instance uses a unique port. After discovering a VD instance, the service manager app adds it as a `child device`. When a `child device` is added, SmartThings classic automatically selects a DH and abstracts it according to the `model` property of the `child device`. Hence, we make the `model` of a VD instance the same as the `name` of the target DH. After the initial connection, a VD instance on the mediator side interacts with a DH instance on the SmartThings via the UPnP (Universal Plug and Play) protocol. In additional, we adapt all DHs for ZigBee/Z-Wave devices available in the SmartThings IDE. For each DH, we add a `subscribe()` function that perform the SUBSCRIBE operation in UPnP. When a DH is instantiated (which means a VD instance is created and a child device is added), it uses the IP and port to send a SUBSCRIBE SOAP (Simple Object Access Protocol) message to the VD instance. Moreover, we replace the original code with new code for receiving and sending SOAP messages. The original code is in `parse` and command functions of DH for receiving ZigBee/Z-Wave data and sending ZigBee/Z-Wave commands. At this point, the VD and DH instances become addressable to each other and a subscribe/publish based UPnP communication is implemented to report data and send commands.

Interfacing with the New SmartThings System

We achieve interfacing with the new SmartThings system by developing a SmartThings schema connector [58] for PFIREWALL platform connector, which requires a cloud-to-cloud connection between PFIREWALL and SmartThings. To this end, we use *fast reverse proxy* [59] to expose PFIREWALL (placed in a home network) to the SmartThings cloud, register PFIREWALL as an application in SmartThings Developer Workspace [60], and make the platform connector (for the new SmartThings) run as an HTTPs server. The PFIREWALL schema connector implements handlers for the interaction types [61] pre-defined by SmartThings, including `Reciprocal Access Token` (mutual authentication), `Discovery` (SmartThings discovers devices connected to PFIREWALL), `State Refresh` (SmartThings queries device states from PFIREWALL), `Command` (SmartThings

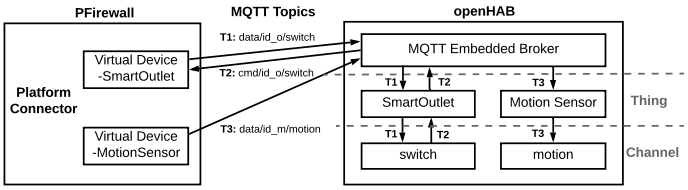


Fig. 9: Overview of interfacing with openHAB.

sends commands to PFIREWALL), `Device State Callback` (PFIREWALL reports events to SmartThings), `Discovery Callback` (PFIREWALL reports newly added devices to SmartThings). For privacy reasons, when the new SmartThings system sends a `State Refresh`) request, PFIREWALL reports the previously-reported states of the requested devices (instead of reporting the current device states).

Interfacing with openHAB

PFIREWALL uses MQTT to interface with openHAB because it is a general connectivity protocol and allows for virtualizing any device types with flexibility. Fig. 9 shows the high-level architecture of the integration. openHAB provides an embedded MQTT broker. When a new device is added to PFIREWALL, a VD instance is created. For each supported attribute of this devices, if it is a read-only attribute (e.g., motion, temperature), the VD subscribes to a topic `data/{deviceId}/{attribute}` for publishing events; if it is a writable attribute (e.g., switch), the VD subscribes to two topics `data/{deviceId}/{attribute}` and `cmd/{deviceId}/{attribute}` for publishing events and receiving commands, respectively. Meanwhile, by creating a new entry in openHAB thing configuration file [62], PFIREWALL adds a corresponding MQTT thing instance with channels to openHAB. Each thing channel in openHAB corresponds to an attribute in VD (e.g., motion, switch) and shares the same topic(s). Thus, the corresponding VD in PFIREWALL and thing channels in openHAB could exchange events and commands via the shared MQTT topic(s).

VI. EVALUATION

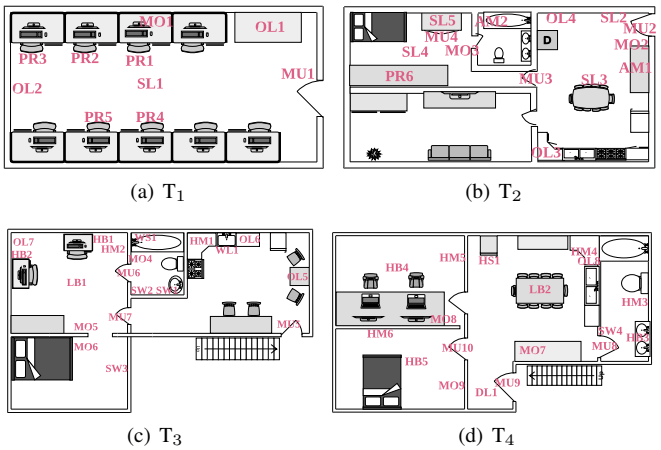
A. Evaluation Setup

We set up four real-world testbeds for evaluating the performance of PFIREWALL: an office with 5 members (T_1), a two-bedroom apartment with 1 member (T_2), and two one-bedroom apartments each of which has 2 members (T_3 and T_4).[7] The floor plan and device placement are shown in Fig. 10. The deployed platform and device types are given in Table II. The configured automation rules of the testbeds are listed in Table III. We ask the members of the testbeds to write down their desired automation rules in English for their own office or home, and then the authors implement the specified rules by installing official apps (T_1, T_2), developing custom apps (T_1, T_2, T_4), and/or using the mobile companion app interface (T_3). The rules are tested and adjusted for one week to make sure they meet the needs of the testbed members.

B. Performance of Data Mediating

To test the correctness of PFIREWALL, we disable data filtering in each testbed, i.e., PFIREWALL simply forwards

[7]We have received the IRB approval (See Appendix B).



(a) T_1 (b) T_2

(c) T_3 (d) T_4

Fig. 10: The layout and device placement in four testbeds. Each presence sensor `PR1`$\sim$`6` is carried with a member.

device events to the target platform without executing any policies. We use the built-in device-level event logs provided by SmartThings Groovy IDE [32] to capture the data received by SmartThings classic and the new SmartThings systems. openHAB does not provide a similar web interface for looking up events, so we build an event logger by utilizing the openHAB REST API [63]. We observe duplicate events in the event logs of SmartThings, and we remove duplicate events before data analysis.

We run experiments using the above setting in all four testbeds for one week and compare the event sequence received by PFIREWALL and the platform for each testbed. The experimental results are given in Table IV, which demonstrate that our mediator works correctly in relaying IoT device events to the platforms.

C. Performance of Data Filtering

To evaluate the performance of our policy-based data filter, we run the four testbeds for another week with data filtering, i.e., we enable the execution of data-minimization policies. Also, two user-specified policies: UP1 (DO NOT report `MO1` motion data between 5pm to 8am) and UP2 (DO NOT report `MU1` contact data between 10am to 6pm) are defined in T_1.

1) Correctness and Reliability: Comparing the received event sequences becomes meaningless since data are filtered in this setting. Instead, we evaluate the correctness of executions of automation rules.

Methodology. PFIREWALL records all commands sent by the platform in each testbed. We ask the testbed members to keep a record of their manual operations on mobile companion apps and manually remove the commands caused by these operations; thus, the remaining commands are supposed to be issued by executions of rules; We use *p-commands* to denote the commands that are issued by the platform when PFIREWALL is used. We check if *p-commands* (i.e., the execution result from filtered events) are **sound** (i.e., policies do not cause false positive device actuation) and **complete** (i.e., policies do not lead to missed device actuation) against *ground truth* commands, denoted as *gt-commands*, which are supposed to be issued by the platform using the same automation rules over

TABLE II: Deployed platforms and devices in the four testbeds. PFIREWALL uses ZigBee (MU, MO, OL, SL, WS, SW), Z-Wave (AM), LAN (PR) or cloud APIs (WL, HM, HB, LB, HS) to connect these devices.

Testbed	Platform	Device (Abbr.)	Protocol	Attribute	Num
T_1	SmartThings Classic	SmartThings hub v2	ZigBee, Z-Wave, WiFi, Ethernet	–	1
		Multipurpose sensor (MU)	ZigBee	contact, temp.	1
		Motion sensor (MO)	ZigBee	motion, temp.	1
		Smart outlet (OL)	ZigBee	switch, power	2
		Smart light (SL)	ZigBee	switch	1
		Presence sensor (PR)	ZigBee	presence	5
T_2	SmartThings Classic	SmartThings hub v2	ZigBee, Z-Wave, WiFi, Ethernet	–	1
		Multipurpose sensor (MU)	ZigBee	contact, temp.	3
		Motion sensor (MO)	ZigBee	motion, temp.	2
		Smart outlet (OL)	ZigBee	switch, power	2
		Smart bulb (SL)	ZigBee	switch	4
		Aeotec MultiSensor (AM)	Z-Wave	motion, humidity, luminance	2
		Presence sensor (PR)	ZigBee	presence	1
T_3	SmartThings New System	Philips Hue Bridge	ZigBee, Ethernet	–	1
		Multipurpose sensor (MU)	ZigBee	contact, temp.	3
		Motion sensor (MO)	ZigBee	motion, temp.	3
		Smart outlet (OL)	ZigBee	switch, power	3
		Smart switch (SW)	ZigBee	switch	3
		Water sensor (WS)	ZigBee	water, temp.	1
		Honeywell water leakage (WL)	WiFi	water, temp.	1
		Hue motion sensor (HM)	ZigBee	motion, temp., luminance	2
		Hue bulb (HB)	ZigBee	switch	2
		LIFX bulb (LB)	WiFi	switch	1
T_4	openHAB	Philips Hue Bridge	ZigBee, Ethernet	–	1
		Multipurpose sensor (MU)	ZigBee	contact, temp.	3
		Motion sensor (MO)	ZigBee	motion, temp.	3
		Smart outlet (OL)	ZigBee	switch, power	1
		Smart switch (SW)	ZigBee	switch	1
		Hue motion sensor (HM)	ZigBee	motion, temp., luminance	4
		Hue bulb (HB)	ZigBee	switch	3
		Hue switch (HS)	ZigBee	button	1
		LIFX bulb (LB)	WiFi	switch	1
		Dimmable light (DL3)	ZigBee	level	1

the raw (unfiltered) events. To generate the gt-commands, we disconnect PFIREWALL from physical devices and replay the one-week raw events to the platform in each testbed, i.e., the raw events are sent to the platform over another week based on their timestamps. Note that all rules in Table III are date-insensitive so that the dates of the timestamps of events does not affect the rules. For convenient comparison, we adjust the start time of gt-commands and make it the same as that of p-commands, i.e., make the two sets of commands start at the same time such that we can compare them.

To verify the *soundness* of each command in p-commands, we check if it has a counterpart within 3 seconds in the gt-commands. If so, the command is sound. Recall that PFIREWALL removes redundant rule executions (i.e., commands) while platforms not; for example, PFIREWALL will not send data for a rule to turn on a fan if the fan is already "ON" (see Section V-A1). Thus, the p-commands should not be considered *incomplete* if it has no counterpart of a redundant command in the gt-commands. Therefore, we traverse the raw events and gt-commands in parallel and remove such redundant commands before verifying completeness. Then, we check if every command in the gt-commands has a counterpart within 3 seconds in the p-commands.

Results. We compute the ratio $\mathbf{R_S}$ and $\mathbf{R_C}$ of commands that are sound and complete, respectively, as shown in Table V. We find that 15 command records are not sound and 1 command (i.e., SL4 OFF) in the gt-commands cannot find a match in the p-commands (marked in bold). Recall that we ask the testbed members to annotate their manual operations on mobile companion apps because these manual operations also generate commands which cannot pass the soundness verification. We ask the testbed members to recall and confirm if they miss annotating manual operations at the timestamp of the unsound commands. After rectifying the annotations, we repeat the soundness verification process. The updated results in the parentheses in Table V show that $\mathbf{R_S}$ increases after improving the user annotations. The remaining 4 unsound command records are very likely to be generated by forgotten manual operations. On the other hand, we look up the SmartThings live logging [32] and confirm that rule 13 (see Table III) indeed issues an OFF command to device SL4 at the timestamp of the unmatched command (SL4 OFF). Therefore, the command may be lost during transmission from SmartThings cloud to PFIREWALL. The evaluation results prove that PFIREWALL does not impede the home automation.

Evaluation of a naive pull-based approach. Considering some platforms use both *push* (i.e., devices report data to platforms via callbacks) and *pull* (i.e., platforms request data from devices) models to obtain data, a *naive* privacy-preserving approach is that PFIREWALL does not report data to the platform unless the platform proactively requests, on the assumption that the platform pulls data on demand. The SmartThings new system uses both push and pull. We use one of the SmartThings testbed T_3 to evaluate this baseline (pull-based) approach. We keep the configuration of devices and rules the same and modify the behavior of PFIREWALL, i.e., PFIREWALL responds to `State Refresh` (pull) with unfiltered real data but does not push any data. We run this new setting for a week and use the aforementioned correctness verification methodology to check if the baseline approach guarantees the correctness of automation.

The results in Table VI show that the baseline approach

TABLE III: Installed rules in all testbeds. Source: official app (O), custom app (C), templates on mobile/web interface (T).

Testbed	ID	Source	Description and Device Bindings
T_1	1	O	When front door (MU1) is opened, turn on light (SL1).
	2	O	When no motion (MO1) or presence (PR1∼5) is detected for 5 minutes, turn off light (SL1).
	3	C	When presence (PR1) becomes present, if time is before 12am, turn on coffee machine (OL1).
	4	C	When motion (MO1) detected, if temperature (MU1) is below $70^\circ F$, turn on heater (OL2).
	5	C	When motion (MO1) is active for longer than 60 minutes, send a message to alert.
	6	C	When door (MU1) is open, if no presence (PR2∼5), send a message.
T_2	7	O	When door is opened (MU2), turn on light (SL2).
	8	O	When motion (MO2) active if luminance (AM1) is below 15 LUX, turn on light (SL3).
	9	O	When presence (PR6) becomes present if between 5-8pm, turn on oven (OL3).
	10	O	When motion (MO2) active if not presence (PR6), send a notification.
	11	O	When wardrobe door (MU4) open, turn on light (SL5); when door (MU4) close, turn off light (SL5).
	12	O	When motion (MO2), if temp. (MU3) below $68^\circ F$, turn on heater (OL4); when motion (MO2) inactive for 20 minutes, turn off heater (OL4).
	13	C	When door (MU3) opened, turn on light (SL4); when door (MU3) closed if motion (MO3) inactive for 5 minutes, turn off light (SL4).
	14	C	When humidity (AM2) exceeds 85% if motion (AM2) active but motion (MO3) keeps inactive for 30 minutes, send a notification.
T_3	15	T	When bedroom motion (MO6) active between 10am and 12am, turn on light (SW3).
	16	T	When kitchen motion (HM1) active between 7am and 12am, turn on microwave outlet (OL6).
	17	T	When sink water leakage (WL1) detected and all motion (MO4∼6, HM1∼2) inactive, send a text message.
	18	T	When front door (MU5) open and luminance (HM2) is below 30 LUX, turn on lights (LB1, OL7).
	19	T	When bathroom motion (MO4) active and water sensor (WS1) wet, turn on fan (SW1).
	20	T	When 7am, turn on coffee outlet (OL5); when 6pm, turn off coffee outlet (OL5).
	21	T	When bathroom motion (MO4) active between 10am and 1am, turn on light (SW2).
	22	T	When bathroom motion (MO4) inactive and bathroom door (MU6) closed, turn off light switch (SW2) after 10 minutes.
	23	T	When study room motion (MO5) active between 9am and 12am and luminance (HM2) is below 30 LUX, turn on lights (LB1, OL7, HB1∼2).
	24	T	When study desk motion (HM2) active between 9am and 12am and luminance (HM2) is below 30 LUX, turn on light (LB1, OL7, HB1∼2).
T_4	25	C	When stovetop motion (HM4) active, if Hue switch (HS1) activates button 1 (day mode) or 3 (nap mode), turn on lamp outlet (OL8).
	26	C	When stovetop motion (HM4) inactive for 5 minutes, turn off the lamp outlet (OL8).
	27	C	When living room motion (MO7) active, if Hue switch (HS1) activates button 1 (day mode) or 3 (nap mode), turn on living room light (LB2).
	28	C	When living room motion (MO7) inactive for 20 minutes, turn off living room light (LB2).
	29	C	When bathroom motion (HM3) active, if Hue switch (HS1) activates button 1 (day mode) or 3 (nap mode), turn on lights (SW4, HB3); if Hue switch (HS1) activates button 2 (night mode), turn on light switch (SW4).
	30	C	When bathroom motion (HM3) inactive and door (MU8) open for 15 minutes, turn off lights (SW4, HB3).
	31	C	When bedroom motion (HM6) active, if Hue switch (HS1) activates button 1 (day mode), turn on light (HB5).
	32	C	When bedroom motion (HM6) inactive for 10 minutes, turn off light (HB5).
	33	C	When study room motion (HM5) active, if Hue switch (HS1) activates button 1 (day mode) or 3 (nap mode), turn on light (HB4).
	34	C	When study room motion (HM5) inactive for 30 minutes, turn off light (HB4).
	35	C	When front door (MU9) open, set the dimmer (DL1) brightness level to 100%; after 5 minutes, set the dimmer to 0%.

TABLE IV: Comparison of events received by PFIREWALL mediator and the target platform in all testbeds. **Total**: the total data volume received by PFIREWALL mediator and the platform, respectively. Due to space limits, we list partial representative devices and attributes, covering different communication protocols and attribute types (binary/numeric-value) in every platform.

Testbed	Device	Attribute	Total	Inconsistency
T_1	MU1	contact	1372, 1372	0
	MU1	temperature	6174, 6174	0
	MO1	motion	1598, 1598	0
	OL1	switch	24, 24	0
	SL1	switch	18, 18	0
	PR1	presence	46, 46	0
T_2	AM1	motion	268, 268	0
	AM1	humidity	459, 459	0
	AM1	luminance	648, 648	0
T_3	WS1	water	12, 12	0
	MO6	temperature	6919, 6919	0
	WL1	water	4, 4	0
	HM1	luminance	1547, 1547	0
	HM2	motion	4046, 4046	0
	HB1	switch	26,26	0
	LB1	switch	18,18	0
T_4	MO7	motion	2116, 2116	0
	HB4	switch	19, 19	0
	LB2	switch	26, 26	0
	HS1	button	15, 15	0
	DL1	level	32 , 32	0

leads to failures (non-executions) of most automation rules (i.e., rules in line 3-11) except rule 20 (line 1 & 2; device: OL5), which only uses a specific time to trigger the rule. According to our observation, SmartThings only pulls the state of a specific device when we operate on the SmartThings mobile app to refresh its state; otherwise, SmartThings relies on the *push* model to update devices' states and trigger rules. Thus, when the naive approach is employed, most automation rules cannot be triggered because the platform does not know the events that trigger the rules.

2) Latency: Compared to the original systems, PFIREWALL adds one more "hop" to the original system. Therefore, the additional latency L_{HA} introduced by PFIREWALL to home automation consists of the computation latency L_1 for executing policies and the additional transmission delay. The additional transmission delay includes the event transmission delay L_2 from PFIREWALL to the local hub (SmartThings classic) or to the platform (the new SmartThings, openHAB) and the command transmission delay in the reverse direction. Thus, L_{HA} is approximately equal to the sum of computation latency L_1 and a round-trip transmission delay $2 * L_2$, i.e., $L_{\mathrm{HA}} = L_1 + 2 * L_2$. We obtain L_1 by computing the elapsed time from when PFIREWALL receives an event to when PFIREWALL reports the event (if the event is reported), and we obtain L_2 by computing the elapsed time from when PFIREWALL reports an event to when the hub/platform receives the same event. We compute the latency for all events and show the averaged results in Table VII. The extra latency (about 0.6 second) is a tradeoff for mitigating privacy leakage. User experience suffers less than indicated by the latency since many rules (e.g., rule 2, 3, 4, 5 in Table III) are not time-critical. Hence, the latency (up to 0.7 seconds) of most automation rules is acceptable.

3) Reduction of Data Leakage: To show the effectiveness of PFIREWALL in filtering events, we compare the data volume collected by PFIREWALL with that reported to the platforms and compute the relative Reduction Rate (RR). As shown in Table VIII, PFIREWALL blocks more than 99% of numeric-

TABLE V: Results of correctness verification of all device commands that are issued by automation rules. N_R: the number of device commands received by PFIREWALL after removing ones caused by the annotated manual operations; N_S: the number of commands that are proved *sound*; N_C: the number of ground truth commands generated based on raw event replaying. $\mathbf{R_S}$ and $\mathbf{R_C}$ are the ratio of commands that are sound and complete, respectively.

Testbed	Device	Comm.	N_R	Soundness		Completeness	
				N_S	$\mathbf{R_S}$	N_C	$\mathbf{R_C}$
T_1	OL1	ON	**6(5)**	5	**0.83(1.00)**	5	1.00
	OL2	ON	7	7	1.00	7	1.00
	SL1	ON	9	9	1.00	9	1.00
	SL1	OFF	9	9	1.00	9	1.00
T_2	OL3	ON	**4(3)**	3	**0.75(1.00)**	3	1.00
	OL4	ON	**21(21)**	20	**0.95(0.95)**	20	1.00
	OL4	OFF	**21(19)**	17	**0.81(0.89)**	17	1.00
	SL2	ON	10	10	1.00	10	1.00
	SL3	ON	31	31	1.00	31	1.00
	SL4	ON	20	20	1.00	20	1.00
	SL4	OFF	19	19	1.00	**20(19)**	**0.95(1.00)**
	SL5	ON	13	13	1.00	13	1.00
	SL5	OFF	13	13	1.00	13	1.00
T_3	OL5	ON	7	7	1.00	7	1.00
	OL5	OFF	7	7	1.00	7	1.00
	OL6	ON	2	2	1.00	2	1.00
	OL7	ON	9	9	1.00	9	1.00
	SW1	ON	5	5	1.00	5	1.00
	SW2	ON	23	23	1.00	23	1.00
	SW2	OFF	**19(18)**	17	**0.89(0.94)**	17	1.00
	SW3	ON	7	7	1.00	7	1.00
	HB1	ON	7	7	1.00	7	1.00
	HB2	ON	7	7	1.00	7	1.00
	LB1	ON	9	9	1.00	7	1.00
T_4	OL8	ON	27	27	1.00	27	1.00
	OL8	OFF	23	23	1.00	23	1.00
	SW4	ON	67	67	1.00	67	1.00
	SW4	OFF	55	55	1.00	55	1.00
	HB3	ON	38	38	1.00	38	1.00
	HB3	OFF	38	38	1.00	38	1.00
	HB4	ON	34	34	1.00	34	1.00
	HB4	OFF	34	34	1.00	34	1.00
	HB5	ON	66	66	1.00	66	1.00
	HB5	OFF	**67(61)**	61	**0.91(1.00)**	61	1.00
	LB2	ON	37	37	1.00	37	1.00
	LB2	OFF	37	37	1.00	37	1.00
	DL1	Set 100	20	20	1.00	20	1.00
	DL1	Set 0	20	20	1.00	20	1.00

TABLE VI: Results of the naive pull-based approach. See Table V for the notations.

Testbed	Device	Comm.	N_R	Soundness		Completeness	
				N_S	$\mathbf{R_S}$	N_C	$\mathbf{R_C}$
T_3	OL5	ON	7	7	1.00	7	1.00
	OL5	OFF	7	7	1.00	7	1.00
	OL6	ON	0	0	–	3	0.00
	OL7	ON	0	0	–	8	0.00
	SW1	ON	0	0	–	5	0.00
	SW2	ON	0	0	–	27	0.00
	SW2	OFF	0	0	–	23	0.00
	SW3	ON	0	0	–	8	0.00
	HB1	ON	0	0	–	7	0.00
	HB2	ON	0	0	–	7	0.00
	LB1	ON	0	0	–	7	0.00

value sensor readings, and 96.79% of binary-value sensor readings and device states (i.e., motion, presence, switch, etc.) which reveals sensitive information. By reducing the disclosed data, PFIREWALL can effectively prevent smart home platforms and potential attackers from inferring private information of smart homes and homeowners based on large amount of IoT device data in the original system. In general, the RR of binary-value attributes are smaller than numeric-value ones, since binary attributes are more often used as rule triggers and hence cannot be completely blocked.

TABLE VII: Latency introduced by PFIREWALL (in seconds).

Testbed	Computation Latency L_1	One-Way Transmission Latency L_2	Automation Latency L_{HA}
T_1	0.136	0.233	0.602
T_2	0.087	0.203	0.493
T_3	0.143	0.274	0.691
T_4	0.083	0.293	0.669

TABLE VIII: Results of the reduction rate RR and correct tracking ratio CTR/CATR per device. VOL: the volume of raw event and the volume of event reported to the platform after data filtering, respectively. The last column is CATR if not specified and otherwise CTR. We present the results for partial representative devices due to page limits.

Testbed	Device	Attribute	VOL	RR	CATR (CTR)
T_1	MU1	contact	1244, 9	0.99	0.01
	MO1	motion	1574, 26	0.98	0.75
	OL1	switch	22, 4	0.82	0.32
	SL1	switch	18, 18	0.00	1.00
	PR1	presence	42, 30	0.29	0.91
	PR2	presence	42, 1	0.99	0.32
T_2	MU2	contact	30, 10	0.67	0.01
	MO2	motion	264, 45	0.83	0.46
	PR6	presence	22, 4	0.82	0.48
	OL3	switch	36, 4	0.89	0.03
	AM1	illumance	658, 1	0.99	0.00 (CTR)
	AM1	humidity	487, 0	1.00	0.00 (CTR)
T_3	WS1	water	14, 5	0.64	0.03
	MO4	motion	918, 45	0.95	0.15
	MO6	motion	923, 7	0.99	0.38
	HM1	motion	1090, 2	0.99	0.05
	HM2	motion	5046, 2	0.99	0.31
	HB1	switch	24, 12	0.50	0.22
	WL1	water	4, 0	1.00	0.00
T_4	HM3	motion	1354, 122	0.91	0.27
	HM4	motion	1178, 50	0.96	0.55
	HM5	motion	5778, 68	0.99	0.48
	HM6	motion	2815, 127	0.95	0.22
	MO7	motion	1898, 74	0.96	0.51
	HB3	switch	338, 122	0.64	0.79
	HS1	button	16, 14	0.13	0.54

4) Privacy Gain of PFIREWALL*:* We further investigate how much privacy PFIREWALL enhances by studying the degree that PFIREWALL prevents attackers from: (1) tracking device states, and (2) inferring sensitive user activities.

Device Status Tracking. An attacker can precisely track the state of a device by leveraging its complete event sequence. When PFIREWALL is deployed, it filters out a very large portion of device events such that it is not easy for an attacker to infer the correct device state with partial information. We define two metrics for this study: Correct Tracking Ratio (CTR) and Correct Active-State Tracking Ratio (CATR):

$$\mathrm{CTR} = \frac{\text{Duration of correctly guessing a sensor's measurements}}{\text{Total duration}} \quad (1)$$

$$\mathrm{CATR} = \frac{\text{Duration of correctly guessing a device's active state}}{\text{Duration of guessing a device's active state}} \quad (2)$$

CTR measures the correctness of tracking numeric-value IoT attributes (e.g., temperature, humidity) and CATR measures the correctness of tracking binary-value attributes (e.g., motion and switch). CTR is not suitable for measuring binary-value attributes; for instance, a CTR is equal to 80% when simply guessing a motion sensor (which is "inactive" for 80% of a day) is "inactive" all day. However, this does not indicate a good inference since an active state (e.g., motion "active",

switch "on") matters more than the other (e.g., motion "inactive", switch "off") from the perspective of activity inference. Instead, CATR only takes the active state of a binary-value attribute into calculation and thus eliminates the impact of unbalanced distributions of binary states.

We calculate the CTR or CATR of every attribute for all devices in the four testbeds over one week. The result is shown in Table VIII. The CTR of all numeric-value attributes are equal or close to zero since most of them are not used by automation rules and hence are not reported to the platform. Although some numeric-value attributes are used in rule conditions (e.g., `AM1`-luminance used by rule 8), more than 99% of their values are not reported and therefore attackers cannot correctly track them. Overall, most binary-value attributes have a low CATR, showing that data filtering prevents attackers from correctly tracking the device status. The CATR of a few binary-value attributes (e.g., `PR1`-presence) is higher because these attributes are heavily needed by rules and thus most of their values cannot be filtered out.

Activity Inference. We consider a strong attacker who has sufficient background and domain knowledge to perform sophisticated attacks to infer user activities. Smart home users usually give a meaningful name (e.g., bathroom motion sensor, bedroom light, microwave outlet) and assign a room to each device. Hence, we assume that the attacker knows the rooms of a home and the assigned room of every device. We also assume that the attacker has expertise to associate activities with device events and states.[8] Based on the above assumptions of a strong attacker, we implement methods (see Table IX) of attackers to infer users' activities. We run the methods on raw events (before data filtering) and the processed events (after data filtering) and compare the results.

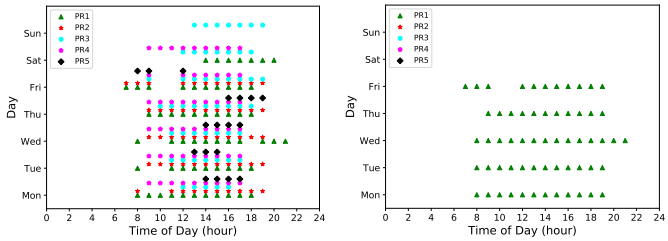
As shown in Fig 11, the working hours of every member who carries a presence sensor can be easily inferred by observing the raw presence sensor data. Note that `PR1`$\sim$`5` are the 5 presence sensors in testbed T_1. Furthermore, the device label of a presence sensor may carry personally identifiable information (e.g., name) and thus the learned working hours could be matched to individuals. When PFIREWALL is deployed, most presence data are filtered out and thus attackers cannot infer working hours correctly from `PR2`$\sim$`5`. Attackers can still achieve an accuracy of since the data of `PR1` cannot be completely filtered since both the states of `PR1` (i.e., "present" and "not present") are needed for executing automation rules 2 and 3 (see Table III).

Figure 12 shows the confusion matrices of activity inference attacks before and after data filtering on testbed T_3 and T_4. Without PFIREWALL, the attacker correctly recognizes most activities. The *recall* (a.k.a., true positive rate) is 1 for most activities except that it is 0.83 for activities 1, 2, 6 in T_3. Data filtering brings the recall of most activities down to less than 0.34 and some (activity 1, 5, 7 in T_3 and 5 in T_4) even to 0, making the attacker miss recognizing most activities. Overall, the total recall of all activities drops from 0.96 to 0.19 in T_3 and from 1 to 0.13 in T_4. Hence, by filtering data, PFIREWALL significantly degrades the performance of the activity inference attacks.

[8]Such knowledge can be easily learned from a lot of existing literature such as [64], [65], [66], [67].

TABLE IX: Methodology for inferring user activities in different testbeds. Device, room and time information is exploited.

Activity	ID	Methodology	Testbed
Working	0	presence sensor on	T_1
Leaving home	1	front door open/closed→all motion sensors become inactive for at least 10 minutes	T_3, T_4
Arriving home	2	front door open/closed → at least a motion sensor becomes active within 3 minutes	T_3, T_4
Toileting	3	1 minute < bathroom motion active < 10 minutes	T_3, T_4
Showering	4	bathroom motion active → water sensor wet	T_3
		15 minutes<bathroom motion active<60 minutes	T_4
Sleeping	5	bedroom motion active→no motion active in other rooms→bedroom motion inactive>10 minutes	T_3, T_4
Cooking	6	kitchen motion > 10 minutes	T_3, T_4
		microwave outlet power > 1000W	T_3
Preparing coffee	7	coffee machine outlet power > 1000W	T_3



(a) Without data filtering (b) With data filtering

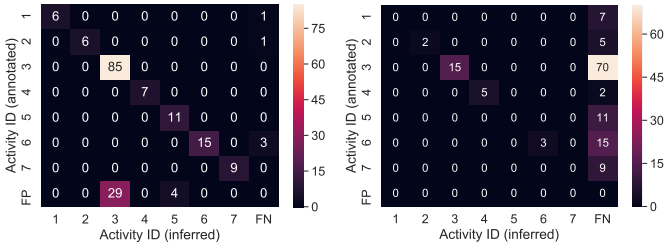
Fig. 11: Visualization of inferred working hours based on the presence sensor data in testbed T_1 without and with data filtering, respectively. For simplicity of illustration, we round all working hours to the nearest hours. `PR1`$\sim$`5`: presence sensors in T_1.

VII. Related Work

A. Smart Home Security

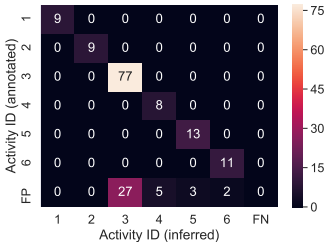
Recent research have explored smart home security and safety in various aspects. Fernandes et al. [41] revealed design vulnerabilities in the permission system of SmartThings. Follow-up works [46], [10], [68], [69], [70] have designed new mechanisms to improve access control in IoT systems. Security researches also studied the application security. For instance, Jia et al. [46] extensively surveyed app-level attacks on IoTs, and presented a context-based permission system. Other than single-app attacks, cross-app interaction (CAI) threats attracted the attention of recent works [71], [24], [25], [12], [11]. HomeGuard [71], [24] is the first work that systematically studies this problem and also the first one that proposes to use theorem proving to detect CAI threats. iRuler [72] applies the same approach to studying CAI threats on the IFTTT platform.

In addition to automation applications, resource-constrained and diverse IoT devices raise unique challenges. For example, how to conduct IoT authentication [73] and pairing [74] for IoT devices that only have a button or knob is an intriguing question. As another example, researchers frequently employ data-driven techniques to detect malfunctions in smart homes but the detection is hardly precise and explainable. HAWatcher [75] proposes a novel semantics-aware approach that utilizes semantic information, such as automation rules and device relations, to mine correlations from event logs for precise and explainable

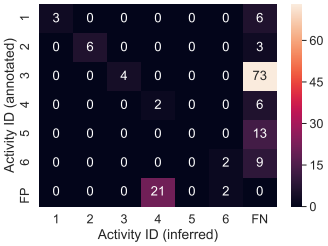


(a) T_3 without data filtering

(b) T_3 with data filtering



(c) T_4 without data filtering



(d) T_4 with data filtering

Fig. 12: Results of activity inference attacks (see Table IX) in testbed T_3 and T_4. FN and FP denote false negative and false positive inference, respectively.

anomaly and attack detection. Security issues related to voice assistants draw great attention [76], [77], [78]. For example, MVP-Ears [78] presents a highly accurate (accuracy $> 99.8\%$) audio adversarial example (AE) detection method inspired by multiversion programming, and it can proactively handle transferable audio AEs. This work is concerned with the rich privacy-sensitive data generated by IoT devices.

B. Smart Home Privacy

Privacy is an important research topic in smart home ecosystems. Zheng et al. [5] studied smart home owners' perceptions of privacy risks and actions taken to protect their privacy. The study found that users are unaware of privacy risks from inference algorithms operating on data from their IoT devices, and they expect device manufacturers to protect their privacy though it is not the case. Celik et al. [7] tracked the sensitive data flows in programming frameworks and identified that 138 out of 230 apps in SmartThings transmit at least one kind of sensitive data over platform-provided APIs, which means malicious apps have the capability to steal user data collected by the platform. The literature [41], [46], [79], [80] also demonstrated that IoT apps/rules can be exploited to breach user privacy. FlowFence [9] enforced a data flow control mechanism for sensitive data protection. However, unlike our work, FlowFence protects sensitive data from unauthorized apps rather than the platform, which is trusted in their threat model. Plus, FlowFence requires the cooperation from the platform and app developers to operate. Xu et al. [81] filter and obfuscate data sent from IoT platforms to IFTTT to enhance smart home privacy; however, the user data still flow to IoT platforms. How to protect smart home privacy from such IoT platforms as SmartThings and Amazon has not been studied prior to our work.

C. In-hub Security and Privacy Enforcement

Many in-hub schemes are proposed to enforce security and privacy schemes in the IoT domain. Simpson et al. designed an in-hub security manager built atop the smart home hub to patch vulnerable IoT devices and strengthen authentication [82]. The security manager is deployed in an open-source system HomeOS. FACT [69] and HanGuard [70] enforced access controls in the middle by implementing controllers on an open-source hub and a programmable WiFi router, respectively. By comparison, these schemes rely on a programmable hub (gateway, router) that can indeed intercept and control the communication between the home area network and the Internet. However, in cloud-based smart home platforms like SmartThings, communications between the commercial hub and the backend cloud are encrypted [83] and hence the router can neither decrypt nor modify the packets on demand. PFIREWALL provides a novel approach to working with closed systems, without modifying the IoT devices, hubs, or clouds.

D. Firewall-Based Solutions

Network firewalls have been extensively studied in different communication layers, such as data-link [84], [85], network [86], [87], transport [88], and application [89], [90], [91], and in different aspects, such as policy compactness [92], [93], verification [94], languages [95], conflicts [96], [97], and so on. PFIREWALL is essentially an application layer filtering system. Existing application firewalls, such as intrusion prevention systems [89] and web application firewalls [90], [91], check on non-payload information (e.g., IP address, port, protocol, DNS, HTTP URL, etc.). On the other hand, PFIREWALL is a novel contextualized solution to the privacy protection of smart homes by inspecting application-layer payloads (i.e., events) and performing semantics-aware data filtering against the "trigger-condition-action" automation rules.

VIII. Discussion and Limitations

Can PFIREWALL perform home automation and thus get rid of the cloud? Theoretically, PFIREWALL is capable of executing the extracted rules locally and disclosing no data to the platform. However, we did not employ this design due to practical considerations. (1) The kick-cloud-out strategy may cause ethical or legal concerns which our research team cannot tackle. While PFIREWALL may provide home automation, all other cloud-based services (messaging, storage, and remote management) will be lost. (2) Huge engineering efforts are needed to implement an equivalent rule engine that supports the same programming framework and APIs as well as maintaining them in a long run. Therefore, we strategically separate the data-filtering policy engine and the rule engine; and PFIREWALL only deals with data filtering.

Pull Strategies. There are two pull-based models for home automation: *batch pull* and *lazy pull*. A representative platform employing *batch pull* is IFTTT, which polls the most recent N (50 by default) events about once every hour [98]. This model still does not take data minimization into consideration. *Lazy pull* is a naive privacy-preserving framework where platforms only pull the current state of devices when needed instead of expecting devices to continuously push data to them. *Lazy pull* is probably impractical for home automation, as the platforms cannot predict the occurrence of device events. To our best knowledge, no smart home platforms use *lazy pull*.

User efforts. To deploy PFIREWALL, users only need to pair their IoT devices with PFIREWALL mediator and operate on the mobile companion app to connect PFIREWALL with the target platform. All these operations are similar to using their original platforms and users do not need any expertise.

Compatibility. Besides home automation, a dashboard function for viewing and controlling device states is usually provided on a companion mobile app by the IoT platform. With PFIREWALL, the dashboard may not always display the correct device states. However, this does not undermine the value of PFIREWALL, which can be deployed as a privacy enhancing *add-on* for privacy-conscious users who mainly use the platform for automation. It is worth noting that occasional remote access from the companion app does not justify that IoT data should continuously flow to the platform. With a small amount of engineering effort, PFIREWALL can provide an equivalent dashboard on its mobile app, which can be used by users for viewing and controlling device states. *End-to-end encryption* can be used to protect data transmissions between the local PFIREWALL mediator and the mobile app, such that data will not be disclosed to the platform or third parties (e.g., cloud-based relay servers).

Generalizability. Although our prototype implementations are on SmartThings and openHAB, the presented approach can be adapted to other IoT systems. As discussed in Section V-B, it is practical to implement such a mediator in most systems. On one hand, the mediator could be extended to connect a large portion of IoT devices (see Section V-B1). On the other hand, the mediator could interface with many platforms via a communication technology supported by the platform. Finally, a spectrum of techniques such as code analysis, UI parsing, and NLP have been employed for extracting automation rules from IoT apps or web/mobile interfaces. Therefore, PFIREWALL can be extended to support more devices and platforms with some engineering efforts. The impact of unsupported devices (which have no APIs) depends on how many such devices are deployed in a particular home. A large-scale investigation on real households that have IoT devices will be helpful to answer this question. This will be our future work.

Attack surface. Like many existing IoT security systems (ContexIoT [46], IoTGuard [11], FACT [69]), our prototype adds an additional component to a smart home, which might become a potential target of attackers. We argue that PFIREWALL can be considered as an IoT hub (with privacy protection functionalities), so conceptually it does not make a smart home more exploitable. PFIREWALL employs existing communication technologies used by IoT devices and platforms to connect them and does not introduce new protocols. Furthermore, PFIREWALL knows who it should talk with, so it can maintain a whitelist, using an embedded firewall, to discard any incoming traffic initiated by unknown sources. Although it is possible that the platform could detect PFIREWALL and behave adversely, it cannot obtain more sensitive data by doing so because PFIREWALL by design only passes data that enables the execution of automation rules that are specified by users.

IX. CONCLUSION

We presented PFIREWALL, a semantics-aware customizable data flow control system for protecting privacy of smart home owners. PFIREWALL filters data by enforcing data-minimization policies automatically derived from automation applications and, optionally, user-specified policies. We overcame multiple challenges and designed an elegant virtualization-based mediating system, which enforces the policies without modifying IoT platforms or devices. We implemented a prototype of PFIREWALL and evaluated it in four real-world testbeds with various IoT devices of different communication protocols (ZigBee, Z-Wave, and WiFi). The evaluation results demonstrated that PFIREWALL can significantly reduce sensitive data leakage without affecting home automation. It severely impairs the attacker's ability to monitor and infer a homeowner's privacy-sensitive behaviors. Besides smart homes, PFIREWALL can also significantly enhance privacy for many other smart environments, such as smart offices, hospitals and factories.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their constructive suggestions. This work was supported in part by the US National Science Foundation (NSF) under grants CNS-1828363, CNS-1564128, CNS-1824440, CNS-2016589, CNS-1856380, CNS-2016415, CNS-1850278, CNS-1815144, and CNS-1953073.

REFERENCES

[1] "IoT platforms company list 2017 update," https://www.statista.com/statistics/805061/global-smart-systems-services-iot-platform-market-size-by-region/, 2018.

[2] "SmartThings," https://www.smartthings.com/, 2020.

[3] "Amazon Alexa," https://developer.amazon.com/en-US/alexa/devices/smart-home-devices, 2020.

[4] "openHAB – an open-source platform for empowering home automation," https://www.openhab.org/, 2020.

[5] S. Zheng, N. Apthorpe, M. Chetty, and N. Feamster, "User perceptions of smart home iot privacy," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, p. 200, 2018.

[6] E. Zeng, S. Mare, and F. Roesner, "End user security & privacy concerns with smart homes," in *Symposium on Usable Privacy and Security (SOUPS)*, 2017.

[7] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, "Sensitive information tracking in commodity iot," in *USENIX Security Symposium*, 2018.

[8] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what?: Controlling flows in iot apps," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 1102–1119.

[9] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks." in *USENIX Security Symposium*, 2016.

[10] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," in *USENIX Security Symposium*, 2017.

[11] Z. B. Celik, G. Tan, and P. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity iot," 2019.

[12] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel, "Iotsan: fortifying the safety of iot systems," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. ACM, 2018, pp. 191–203.

[13] K.-H. Hsu, Y.-H. Chiang, and H.-C. Hsiao, "Safechain: Securing trigger-action programming from attack chains," *IEEE Transactions on Information Forensics and Security*, 2019.

[14] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and A. S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" *arXiv preprint arXiv:1808.02741*, 2018.

[15] T. Datta, N. Apthorpe, and N. Feamster, "A developer-friendly library for smart home iot privacy-preserving traffic obfuscation," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*. ACM, 2018, pp. 43–48.

[16] N. Apthorpe, D. Reisman, and N. Feamster, "Closing the blinds: Four strategies for protecting smart home privacy from network observers," *arXiv preprint arXiv:1705.06809*, 2017.

[17] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic," *arXiv preprint arXiv:1708.05044*, 2017.

[18] "Bupa suffers major data breach as disgruntled employee steals data," https://www.cbronline.com/cybersecurity/breaches/bupa-major-data-breach-disgruntled-employee-steals-data/, 2017.

[19] "The 17 biggest data breaches of the 21st century," https://www.csoonline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html, 2018.

[20] "Wink terms of use and privacy policy," https://www.wink.com/legal/, 2018.

[21] "SmartThings privacy policy," https://www.smartthings.com/privacy, 2018.

[22] "Vera terms and conditions," http://getvera.com/legal/, 2018.

[23] "European general data protection regulation," https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679, 2016.

[24] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 411–423.

[25] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *USENIX Security Symposium*, 2018.

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

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

[28] "Mozilla IoT – a smart home protect by Mozilla," https://iot.mozilla.org/, 2019.

[29] C. D. Coley and R. E. Wesinger Jr, "Firewall system for protecting network elements connected to a public network," Oct. 20 1998, uS Patent 5,826,014.

[30] R. E. Wesinger Jr and C. D. Coley, "Firewall providing enhanced network security and user transparency," Apr. 27 1999, uS Patent 5,898,830.

[31] H. Chi, Q. Zeng, X. Du, and L. Luo, "Pfirewall: Semantics-aware customizable data flow control for home automation systems," *arXiv preprint arXiv:1910.07987*, 2019.

[32] "SmartThings Groovy IDE," https://graph.api.smartthings.com/, 2020.

[33] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-level signatures for smart home devices," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2020.

[34] T. OConnor, W. Enck, and B. Reaves, "Blinded and confused: uncovering systemic flaws in device telemetry for smart-home internet of things," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2019.

[35] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P). https://doi. org/10.1109/SP*, 2019.

[36] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin, "Security analysis on consumer and industrial iot devices," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 519–524.

[37] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, "Web-based attacks to discover and control local iot devices," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, 2018, pp. 29–35.

[38] E. Ronen and A. Shamir, "Extended functionality attacks on iot devices: The case of smart lights," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 3–12.

[39] T. Chomsiri, "Sniffing packets on lan without arp spoofing," in *2008 Third International Conference on Convergence and Hybrid Information Technology*, vol. 2. IEEE, 2008, pp. 472–477.

[40] W. A. Arbaugh, N. Shankar, Y. J. Wan, and K. Zhang, "Your 80211 wireless network has no clothes," *IEEE Wireless Communications*, vol. 9, no. 6, pp. 44–51, 2002.

[41] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE Symposium on Security and Privacy 2016*.

[42] "REST API," https://restfulapi.net/, 2020.

[43] J. Bae, H. Bae, S.-H. Kang, and Y. Kim, "Automatic control of workflow processes using ECA rules," *IEEE transactions on knowledge and data engineering*, vol. 16, no. 8, pp. 1010–1023, 2004.

[44] W. Brackenbury, A. Deora, J. Ritchey, J. Vallee, W. He, G. Wang, M. L. Littman, and B. Ur, "How users interpret bugs in trigger-action programming," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019, p. 552.

[45] E. Fernandes, A. Rahmati, K. Eykholt, and A. Prakash, "Internet of things security research: A rehash of old ideas or new intellectual challenges?" *IEEE Security & Privacy*, vol. 15, no. 4, pp. 79–84, 2017.

[46] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "Contexiot: Towards providing contextual integrity to appified iot platforms," in *Proceedings of The Network and Distributed System Security Symposium*, 2017.

[47] I. Hwang, M. Kim, and H. J. Ahn, "Data pipeline for generation and recommendation of the iot rules based on open text data," in *IEEE WAINA*, 2016.

[48] "SmartThings capabilities," https://smartthings.developer.samsung.com/docs/api-ref/capabilities.html, 2020.

[49] "JavaScript linear programming solver," https://www.npmjs.com/package/javascript-lp-solver, 2016.

[50] "Clear connect rf," https://www.lutron.com/en-US/CaseStudyPDF/Clear%20Connect%20Technology%20whitepaper.pdf, 2019.

[51] "Homekit accessory protocol," https://developer.apple.com/support/homekit-accessory-protocol/, 2020.

[52] "GSM," https://www.gsma.com/, 2020.

[53] "LAN-connected devices," https://docs.smartthings.com/en/latest/cloud-and-lan-connected-device-types-developers-guide/index.html, 2018.

[54] "Types of device integrations on SmartThings," https://smartthings.developer.samsung.com/docs/devices/device-basics.html, 2020.

[55] "MQTT Binding," https://www.openhab.org/addons/bindings/mqtt/, 2019.

[56] "Web of Things," https://webofthings.org, 2019.

[57] "Wink APIs: A RESTful Service," https://winkapiv2.docs.apiary.io/#reference/a-restful-service, 2019.

[58] "SmartThings schema connector," https://smartthings.developer.samsung.com/docs/devices/smartthings-schema/schema-basics.html, 2020.

[59] "frp," https://github.com/fatedier/frp, 2020.

[60] "SmartThings developer workspace," https://smartthings.developer.samsung.com/workspace, 2020.

[61] "Interaction types," https://smartthings.developer.samsung.com/docs/devices/smartthings-schema/smartthings-schema-reference.html, 2020.

[62] "openHAB configuration guide," https://www.openhab.org/docs/configuration/things.html, 2020.

[63] "openHAB REST API," https://www.openhab.org/docs/configuration/restdocs.html, 2020.

[64] P. Rashidi, D. J. Cook, L. B. Holder, and M. Schmitter-Edgecombe, "Discovering activities to recognize and track in a smart environment," *IEEE transactions on knowledge and data engineering*, vol. 23, no. 4, pp. 527–539, 2010.

[65] R. Hammid, S. Maddi, A. Johnson, A. Bobick, I. Essa, and C. L. Isbell, "Unsupervised activity discovery and characterization from event-streams," *arXiv preprint arXiv:1207.1381*, 2012.

[66] T. Gu, Z. Wu, X. Tao, H. K. Pung, and J. Lu, "epsicar: An emerging patterns based approach to sequential, interleaved and concurrent activity recognition," in *2009 IEEE International Conference on Pervasive Computing and Communications*. IEEE, 2009, pp. 1–9.

[67] K. Viard, M. P. Fanti, G. Faraut, and J.-J. Lesage, "An event-based approach for discovering activities of daily living by hidden markov models," in *2016 15th International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS)*. IEEE, 2016, pp. 85–92.

[68] A. Rahmati, E. Fernandes, K. Eykholt, and A. Prakash, "Tyche: Risk-based permissions for smart home platforms," *arXiv preprint arXiv:1801.04609*, 2018.

[69] S. Lee, J. Choi, J. Kim, B. Cho, S. Lee, H. Kim, and J. Kim, "Fact: Functionality-centric access control system for iot programming frameworks," in *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*. ACM, 2017, pp. 43–54.

[70] S. Demetriou, N. Zhang, Y. Lee, X. Wang, C. A. Gunter, X. Zhou, and M. Grace, "Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2017, pp. 122–133.

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

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

[73] X. Li, F. Yan, F. Zuo, Q. Zeng, and L. Luo, "Touch well before use: Intuitive and secure authentication for iot devices," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–17.

[74] X. Li, Q. Zeng, L. Luo, and T. Luo, "T2pair: Secure and usable pairing for heterogeneous iot devices," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 309–323.

[75] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart homes," in *USENIX Security Symposium*, 2021.

[76] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, "Hidden voice commands," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 513–530.

[77] N. Carlini and D. Wagner, "Audio adversarial examples: Targeted attacks on speech-to-text," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 1–7.

[78] Q. Zeng, J. Su, C. Fu, G. Kayas, L. Luo, X. Du, C. C. Tan, and J. Wu, "A multiversion programming inspired approach to detecting audio adversarial examples," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 39–51.

[79] M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia, "Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 1501–1510.

[80] S. Manandhar, K. Moran, K. Kafle, R. Tang, D. Poshyvanyk, and A. Nadkarni, "Towards a natural perspective of smart homes for practical security and safety analyses," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 482–499.

[81] R. Xu, Q. Zeng, L. Zhu, H. Chi, X. Du, and M. Guizani, "Privacy leakage in smart homes and its mitigation: Ifttt as a case study," *IEEE Access*, vol. 7, pp. 63 457–63 471, 2019.

[82] A. K. Simpson, F. Roesner, and T. Kohno, "Securing vulnerable home iot devices with an in-hub security manager," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2017, pp. 551–556.

[83] "Veracode white paper - iots security research study," https://www.veracode.com/sites/default/files/Resources/Whitepapers/internet-of-things-whitepaper.pdf, 2015.

[84] A. Khan, N. Al-Darwish, M. Guizani, M. Benten, and H. Youssef, "Design and implementation of a software bridge with packet filtering and statistics collection functions," *International Journal of Network Management*, vol. 7, no. 5, pp. 251–263, 1997.

[85] J. Liu and Y. Ma, "Packet filtering in bridge," in *1999 Internet Workshop. IWS99.(Cat. No. 99EX385)*. IEEE, 1999, pp. 94–98.

[86] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets," *ACM SIGCOMM computer communication review*, vol. 31, no. 4, pp. 15–26, 2001.

[87] K. Ingham and S. Forrest, "A history and survey of network firewalls," *University of New Mexico, Tech. Rep*, 2002.

[88] B. Cheswick, "The design of a secure internet gateway," in *USENIX Summer Conference Proceedings*. Citeseer, 1990.

[89] E. Viegas, A. O. Santin, A. Franca, R. Jasinski, V. A. Pedroni, and L. S. Oliveira, "Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 163–177, 2016.

[90] T. Krueger, C. Gehl, K. Rieck, and P. Laskov, "Tokdoc: A self-healing web application firewall," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, 2010, pp. 1846–1853.

[91] S. Prandl, M. Lazarescu, and D.-S. Pham, "A study of web application firewall solutions," in *International Conference on Information Systems Security*. Springer, 2015, pp. 501–510.

[92] M. Yoon, S. Chen, and Z. Zhang, "Minimizing the maximum firewall rule set in a network with multiple firewalls," *IEEE Transactions on Computers*, vol. 59, no. 2, pp. 218–230, 2009.

[93] C. Bodei, P. Degano, L. Galletta, R. Focardi, M. Tempesta, and L. Veronese, "Language-independent synthesis of firewall policies," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 92–106.

[94] N. B. Youssef, A. Bouhoula, and F. Jacquemard, "Automatic verification of conformance of firewall configurations to security policies," in *2009 IEEE Symposium on Computers and Communications*. IEEE, 2009, pp. 526–531.

[95] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, and C. Pitcher, "Specifications of a high-level conflict-free firewall policy language for multi-domain networks," in *Proceedings of the 12th ACM symposium on Access control models and technologies*, 2007, pp. 185–194.

[96] E. S. Al-Shaer and H. H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *International Symposium on Integrated Network Management*. Springer, 2003, pp. 17–30.

[97] F. A. Maldonado-Lopez, E. Calle, and Y. Donoso, "Detection and prevention of firewall-rule conflicts on software-defined networking," in *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*. IEEE, 2015, pp. 259–265.

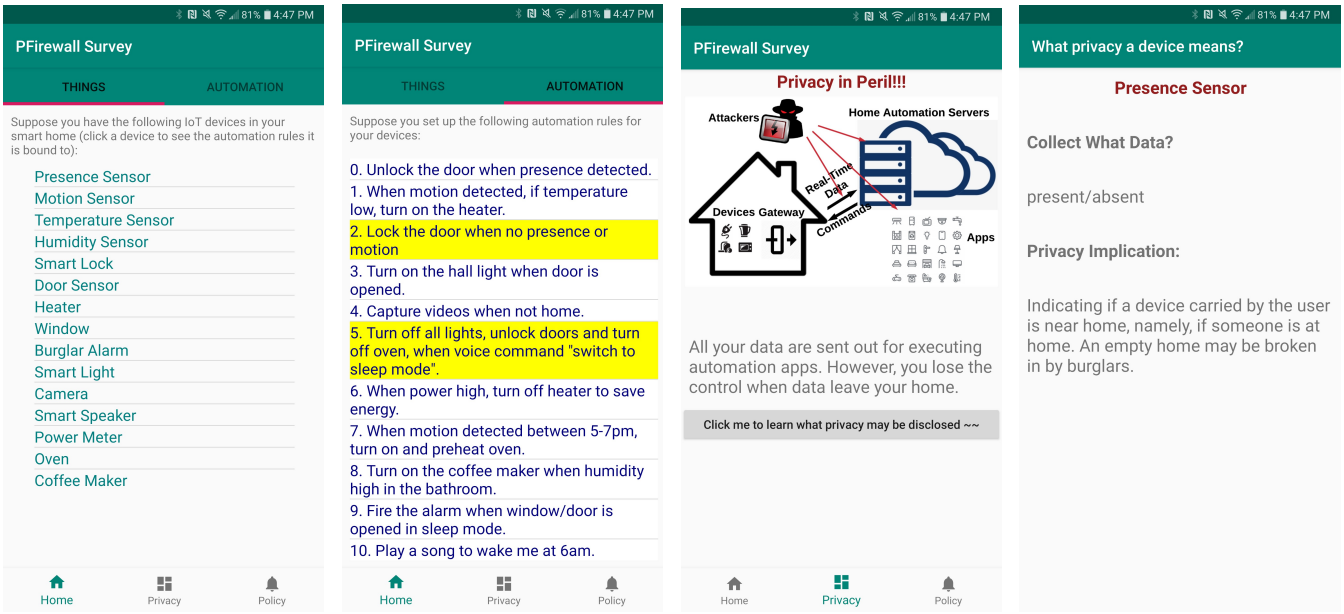
[98] "General API requirements," https://platform.ifttt.com/docs/api_reference, 2020.

APPENDIX

A. User Study

1) Setup: We conduct a user survey to study users' attitude and abilities towards defining customized data-protection policies with our policy templates (Section V-A2). We recruit 20 adult participants who are knowledgeable about the concepts "home automation", "smart home" or "IoT" from our institutions. Participants completed the trial tasks of our "PFirewall Survey" app in our lab using smartphones we provided and after that answered several questions.

We asked the participants to get familiar with a smart home setting where 10 automation rules (Fig. 13(b)) are configured to work with 15 devices (Fig. 13(a)). The app provides a page (Fig. 13(c)) to illustrate the architecture of the system and the



(a) (b) (c) (d)

Fig. 13: The PFirewall Survey mobile app used in the user survey.

potential risks of data leakage; we did not explain the content and ask questions about this page to avoid influencing the understanding of end-users by factors other than the interface itself. Besides, the app also provides an interface showing the list of 15 devices; when a device is selected, the app switches to a device detail page (e.g., Fig. 13(d)) showing what data the device generates and what privacy risks are imposed if the data are leaked. In addition, policy templates (as shown in Fig. 5(b)) were provided for participants to define their own policies. After a 30-minute trial, participants were asked to answer questions.

2) Results: All 20 participants cared about their data privacy and thought it useful to define their own data flow policies for protecting privacy. However, 2 participants thought they would not spend time in defining policies even if an app is available. We collect the number of participants who had privacy concerns on each listed device. Cameras and smart speakers were the top two devices whose data are considered sensitive by the participants (19 and 16, respectively); half or more participants had concerns on the status data of smart locks, doors and windows (11, 13, 10, respectively); Each of humidity sensors, heaters, lights, powers and coffee makers is concerned by less than 3 participants.

Besides the listed devices, the participants also cared about the data privacy of smart TVs, smart window blinds, smart outlets. Regarding the usability of our policy templates, 8 participants thought the templates are "very easy" to use and 12 participants thought them "easy" to use. Three participants found that they could not specify policies to control data by specifying multiple conditions with the templates, for example, the combination of a device state and a specified time period. According to the feedback, we address this issue by allowing users to choose if they would like to add a new one recursively after they complete a condition.

Overall, participants concern data privacy and hold a positive attitude in defining own policies with our templates. The result also shows that participants may overlook the privacy risks of some devices like humidity sensor and powers, which we have discussed in Section VI-C4. Hence, data-minimization policies and user-specified policies could work together to achieve better privacy protection.

B. IRB Approval

Our testbeds need to collect data from the testbed providers, including the 5 office members and 5 apartment members. Also, our user study involves 20 participants. We have received the approval from the IRB in the institution where all the above investigations are performed. The testbed providers and survey participants (undergraduate and graduate students) were recruited through emails and flyers. $500 was paid to the participants of each testbed and $50 was paid to each survey participant.

We value the participants' privacy during our investigation processes. The data collected from all testbeds do not contain personally identifiable information and location data. The collected data will be transmitted to and stored in the password protected computer of one of the authors. Computers that store data have password-protected accounts and will be in a locked office that has limited access. Only the researchers identified on this protocol will have access to the data. Survey participants are asked to submit their questionnaire anonymously without revealing any personally identifiable information. The questionnaire will be stored in the locked office after analyses.