Lumos: Improving Smart Home IoT Visibility and Interoperability Through Analyzing Mobile Apps

ABSTRACT

The era of Smart Homes and Internet of Things (IoT) call for integrating diverse "smart" devices, including sensors, actuators, and home appliances. However, enabling interoperation across heterogeneous IoT devices is a challenging task because vendors use their own control and communication protocols. Prior approaches have attempted to solve this problem by asking for vendor support, or even fundamentally redesigning the architecture of IoT devices. These approaches face limitations as they require disruptive changes.

This paper explores a new approach in improving IoT interoperability without requiring architectural changes or vendor participation. Focusing on smart-home environments, we propose Lumos that improves interoperability by leveraging Android apps that control IoT devices. Lumos uses this information learned from IoT apps to enable "best-effort" interoperation across heterogeneous devices. Our evaluation with fifteen commercial IoT devices from three major IoT platforms and in-depth user studies conducted with 24 participants demonstrate the promising efficacy of Lumos on implementing diverse interoperation scenarios.

1. INTRODUCTION

The smart home market was valued at \$76.62 B in 2018 and is expected to reach \$151.38 B by 2024 [54]. Several major players (Apple, Amazon, Google, Samsung, and Honeywell) are consistently introducing new smart appliances and promoting their own platforms for their market share. Along with their popularity, the demand for their *interoperability* has increased [48, 68]. However, it is not trivial to implement a well-integrated system [3, 16, 19, 39, 46, 62].

The difficulties stem from three limitations: 1) Smart home devices often have their own means of control, such as mobile apps, that are proprietary [19, 39]. 2) No single "open" IoT platform is able to cover all IoT devices—SmartThings and Wink, the two largest "open" platforms, respectively support 219 and 115 devices (May 2018), while few devices are supported by both [60, 70]. 3) Interoperation across different IoT platforms is not a design priority.

Existing approaches to IoT interoperation in the smarthome context are based on either voluntary vendor participation [1, 26, 33, 42, 45] or architecture generalization [2, 3, 6, 16, 34, 62, 74]. These approaches commonly propose standardized interfaces or architectures that heterogeneous IoT devices must conform to. Unfortunately, they are not widely deployed because they require extra engineering effort to the vendors, significant changes to the current IoT architecture, and/or the agreement of all stakeholders.

We posit that interoperability will remain a long-lasting problem in the foreseeable future and seek for an alternative solution driven by users. Our goal is to unilaterally enable interoperation in a *best effort* manner from the user side. To this end, we design a new system called Lumos that empowers users with an automated framework that supports interoperability. We leverage only the information already available, enabling interoperation of IoT devices without explicit vendor support or architectural changes. Our contribution is to show that a pure user-driven approach is viable in bridging the gap until the market fully resolves the problem.

Lumos leverages the key insight that almost all IoT devices for smart home are controlled by Android apps [28, 29, 55]. The apps are readily available and already know how to control IoT devices and query device information.

Leveraging information obtained through app analysis, Lumos improves the interoperability with minimal user configuration. To infer the semantics and control/status messages generated by an app, Lumos combines UI, program and traffic analyses performed on the app binary. It then integrates the analysis results with finer-grained semantic information from the user who best knows the context of the action she triggers when using the app. Based on the information, Lumos creates an interoperation gateway that understands the semantics of messages between the app and its IoT device and actively sends control messages and status queries to the IoT device. Finally, Lumos helps users easily create interoperation scenarios of their own by automatically generating scripts.

We evaluate Lumos using 15 commercial off-the-shelf smart-home devices. We show Lumos can learn the semantics of IoT operation from network traffic for all device features (29 out of 29), and is able to generate status and control messages for most features (26 out of 29, §6.2). Accordingly, we demonstrate this enables Lumos to support diverse interoperation scenarios. Finally, our user study with 24 participants shows that Lumos offers a usable programming framework that enables interoperation of IoT devices, offers interoperation features that are not available on commodity IoT platforms, and requires reasonable configuration effort compared with three popular IoT platform-native apps.

In summary, this paper makes two key contributions:

- **Novel approach to interoperability:** We present an au-

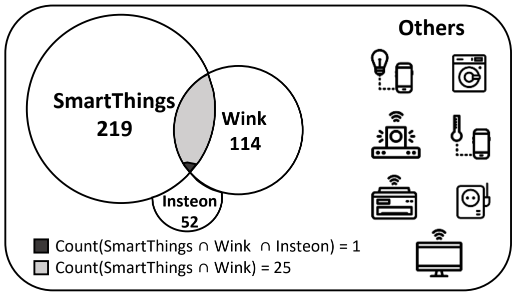

Others
SmartThings
219
Wink
114
Insteon
52
■ Count(SmartThings ∩ Wink ∩ Insteon) = 1
□ Count(SmartThings ∩ Wink) = 25

Figure 1: Fragmented smart home ecosystems

tomated framework that combines static program analysis and dynamic learning to understand and re-construct control messages with user-given semantics.

- **System prototype and evaluation:** Our in-depth evaluation shows Lumos enables interoperation between IoT devices in a unilateral fashion, further enabling new value-added home IoT services. To the best of our knowledge, our evaluation covers the *largest* set of commercial IoT devices among existing work. Our user study with 24 participants demonstrates the promising efficacy of Lumos and quantifies the user effort. We open-source our prototype in an anonymized repository: https://bit.ly/2RyGdu8.

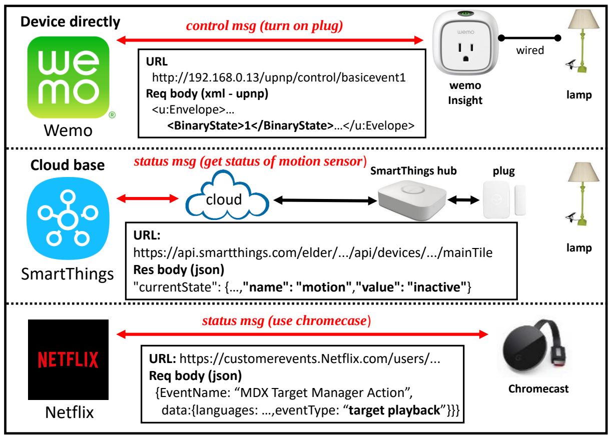
2. MOTIVATION

2.1 The Status Quo of IoT Interoperation

Market reports [31] indicate 450 IoT platforms exist worldwide as of 2017, marking a 25% increase since 2016, showing how the IoT ecosystem is becoming increasingly fragmented. *Interoperability* is an ability to create a coherent service by interacting with multiple IoT devices. Many studies confirm that such fragmentation presents a significant barrier that impedes interoperability and a wider adoption of home IoT services [17, 30, 32, 46, 71]. We often come across users experiencing frustrations and ordeals trying to make different IoT devices interoperable [53, 56].

Fig. 1 shows three major IoT platforms and the number of devices they support. We make following observations:

- Only the devices on the same platform are interoperable. Cross-platform inter-operation is generally not supported. Out of the three, only SmartThings exposes external APIs for controlling and monitoring devices [61]. Although this allows external parties to control or monitor SmartThings devices, SmartThings platform cannot do the same to devices on other platforms.
- If devices belong to a platform, they are locked-in to that specific platform. They do not support multiple platforms, and they do not change their platform either. We suspect that this limitation originates from implementation costs and business partnerships [59, 63].
- Many IoT devices are still stand-alone, and do not interoperate with other devices. For example, Chromecast [21] does not belong to any of the three major platforms, and cannot interact with any devices on these platforms.



```
URL
 http://192.168.0.13/upnp/control/basicevent1
Req body (xml - upnp)
 <u:Envelope>...
   <BinaryState>1</BinaryState>...</u:Evelope>
```

```
URL:
https://api.smartthings.com/elder/.../api/devices/.../mainTile
Res body (json)
"currentState": {...,"name": "motion","value": "inactive"}
```

```
URL: https://customerevents.Netflix.com/users/...
Req body (json)
 {EventName: “MDX Target Manager Action”,
  data:{languages: ...,eventType: “target playback”}}}
```

Figure 2: Connection types of devices and network traffic

Industry efforts: Notable approaches to this problem from the industry include OpenT2T [45], IFTTT [26], and Home Assistant [25]. OpenT2T defines common IoT schemas, which consist of properties for similar devices. For example, a common schema for IoT light bulbs can define an “on-off” property with two possible values, “on” and “off”. These schemas are then translated into vendor specific implementations. A common schema then provides a consistent user experience when operating similar devices, even when they are from different manufacturers or support different protocols. While the idea of abstraction is noble, it requires vendor participation to support common schemas. No tangible incentive for this participation has contributed to rendering OpenT2T inactive for more than three years at the time of writing [44].

IFTTT [26] allows users to write Applets that connect popular web services, apps, and IoT devices. A user can set up triggers that specify when an Applet should run, filters that express a desired condition, and actions that are executed when filter conditions are met. Combined with IoT devices, it can be used to customize IoT services. However, IFTTT leverages existing open Web APIs to access device state and issue commands, which requires vendor supports. Even when vendors expose open APIs, we find that they expose only a small set of features but not the full functionality.

Home Assistant [25] is an open source home automation platform running on a Raspberry Pi hub. It tracks and control devices at home and automate controls. Similar to OpenT2T, it requires vendor participation for supporting a new device [24]. Lumos also assumes the presence of a trusted home gateway, but does not rely on vendor participation.

2.2 Challenges and Key Insight

Approaches that demand vendor participation are still far from wide-deployment, despite increasing interoperability needs from users. However, it is non-trivial to devise a solution that does not require vendor participation because only vendors have comprehensive understandings of their devices.

The core challenges in addressing interoperability are to 1) capture device state for context monitoring (*visibility*) and

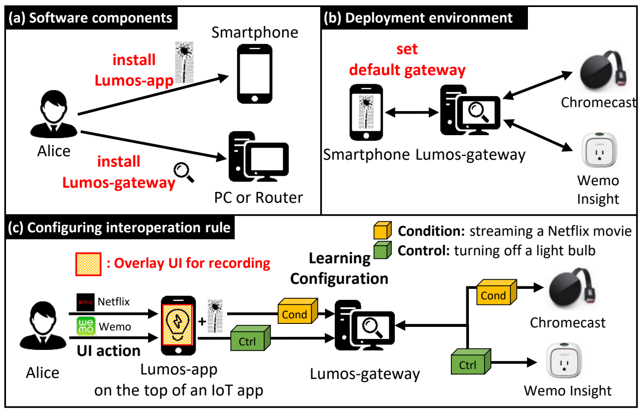


Figure 3: Usage Model - System components, Deployment environment, and Configuring interoperation rule

2) issue a desired command (*controllability*). We take the following usecase as a concrete running example to illustrate the problem. Alice has an IoT light bulb and a streaming dongle (e.g., Chromecast [21]). Each device has a companion mobile app that allows Alice to control the device. Now, Alice wants to configure them together so that the light bulb can automatically turn itself off (or dim) when she streams a movie to her TV. A smart-home system should be aware of whether the streaming device is currently playing a video and be able to issue the control command to turn off the light bulb. Once the two tasks are addressed, implementing an IoT service with multiple devices amounts to writing a composition rule (or a "recipe" in IFTTT).

Key insight: Our goal is to enable the two tasks in a user-driven, best effort manner with automation support. Our key insight is that mobile apps play a key role in communicating with IoT devices and contain valuable information for interoperability. 1) They already have the ability to control and monitor IoT devices. 2) Often vendors themselves provide the apps and maintain them up-to-date. 3) The graphical user interface (GUI) of the apps provides semantic information (e.g., this button turns off the light).

Fig. 2 shows an example where Alice uses a Wemo Insight Plug to control room lighting. When Alice watches a Netflix movie using Chromecast, the Netflix app sends an HTTP request message to the server. This request contains a message ("eventType":"target playback") denoting that the movie is to be played on the TV connected with Chromecast. When the request is detected, Lumos triggers a request to the Wemo Insight Plug to power off. This is feasible because Lumos learns from IoT apps to *recognize* the condition and *generate* the control message.

3. Lumos USAGE MODEL

In order to use Lumos, a user installs two software components (Fig. 3 (a)). One component is Lumos-app, a mobile app that allows users to configure interoperation. The other component is Lumos-gateway, a middlebox that can be installed on either a desktop or a router that can run a custom OS (e.g., a Netgear router). Lumos-gateway is a trusted party that monitors all traffic that IoT apps generate (Fig. 3 (b)). To monitor the traffic, users need to configure their devices to use Lumos-gateway as the default gateway. Note, an IoT app may directly connect to an IoT device (e.g., Wemo Insight Plug) or go through an IoT hub that talks to IoT devices (Fig. 2). Lumos-gateway supports both cases.

Configuring interoperation: Our interoperation rule consists of a condition and a control action. For example, in our running example with Alice, the condition is streaming a Netflix movie and the control action is turning off the light. When Lumos detects the condition, it performs the control action. To configure a rule, users "teach" Lumos by performing UI actions that correspond to the condition and the control. For example, Alice teaches Lumos her condition (streaming a Netflix movie) by opening her Netflix app and playing a movie. She also teaches her control (turning off a light bulb) by opening her Wemo app and turning the light bulb off.

We automate this process by capturing the user interaction with Lumos-app that runs in the background and displays an additional UI overlaid on top of an IoT app UI. The additional UI displayed by Lumos-app guides the user through the process of configuring a condition action and a control action. While a user configures a condition action and a control action, the Lumos-app monitors the UI actions performed by the user and communicates with Lumos-gateway to capture the requests and responses caused by those UI actions at the network level, as illustrated in Fig. 3 (c). Then, Lumos-gateway analyzes the requests and responses to detect the condition and trigger control actions. Our implementation of the Lumos-app extends an existing UI record-and-replay tool called SUGILITE [38], which relies on Android's accessibility API to monitor, intercept, and inject UI actions.

Lumos approach: Lumos is a *post-hoc* approach that offers a viable alternative for commonplace IoT applications, which we demonstrate in §6. We embrace the status quo (lack of interoperability due to fragmentation) and focus on developing an automated framework for pure user-driven interoperation that reduces manual human involvement in the process. We envision a community-driven approach in which users, who currently have limited or no options, can share information and make their own choice regarding interoperability of their own IoT devices. In addition to providing interoperability, we believe the approach will act as a market pressure to enable "interoperation-by-design".

Advantage over an alternative design: Given the capability of the Android accessibility API, it may seem reasonable to consider a pure UI-level approach as an alternative, where we develop a "master" app that uses the accessibility API to achieve interoperability. 1) It would record UI actions performed on an IoT app as a condition or a control using the accessibility API. 2) After recording a condition or a control, it would detect the condition while monitoring UI actions and issue the control by replaying the recorded UI actions.

However, this would result in a much more intrusive system

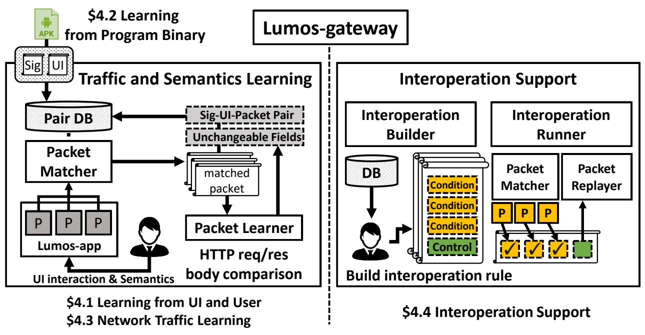


Figure 4: Lumos system overview

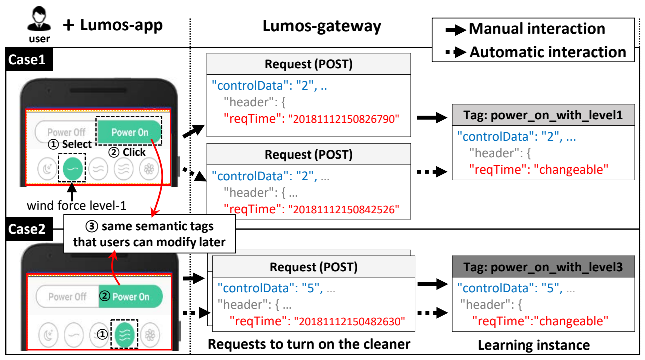


Figure 5: dynamic learning examples of WINIX app

because the accessibility API requires an app to be in the foreground. For example, suppose a condition is "if light bulb *A* is turned off." There are two ways of turning off an IoT light bulb: turn it off on the companion IoT app, or physically turn it off. The master app can easily monitor the former, since it is done through UI actions by a user. To monitor the latter, the master app must periodically monitor the light bulb status, which would be extremely intrusive to a user since it would repeatedly bring up the IoT app. In contrast, Lumos needs an IoT app only when a user configures an interoperation rule.

4. DESIGN

Achieving our goal requires satisfying three requirements: 1) Lumos-app must provide a way for users to leverage IoT app UIs to "teach" our system of their intended conditions and control actions. 2) Lumos-gateway must be able to recognize pre-configured conditions from the network messages that IoT apps generate and issue control messages to trigger desired actions. 3) It must offer programmability using the "learned" information to configure interoperation rules. Fig. 4 presents a system overview that delivers the goal. We detail each component of our design as follows.

4.1 Learning from UIs and Users

Lumos starts by learning the semantics of IoT operation through the UI and user actions on an app. The objective of this process is to identify the actions of interest that generate control/status messages and label them with a specific semantic tag. This tag denotes an IoT device operation controlled via a mobile app UI component, such as turning on/off a bulb or playing music. Lumos takes a user-assisted semantic labeling approach to reduce human involvement.

In the 'teaching' phase, when a user clicks a UI component (e.g., a button), Lumos assigns the resource ID of the UI component as the semantic tag because a resource ID is a human-readable string that usually has semantic information (e.g. brightness_slider). This is done by Lumos-app that monitoring user interactions through the Android accessibility API (e.g., android.accessibilityservice.getWindows()).

However, the tag might be insufficient. Some resource IDs do not contain any semantic information (e.g., button1), or a single button may trigger different actions depending on the context (e.g., a single switch button for power on and off).

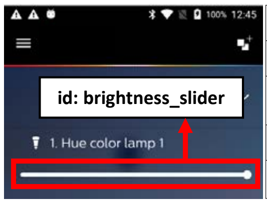
To manage this, Lumos allows a user to edit semantic tags displayed during the 'teaching' phase to make the meaning more specific and personalized to the user. For example, when Alice wants to teach how to turn on a WINIX air cleaner using its app, she demonstrates it by clicking the power-on button while Lumos-app is running. Lumos-app then records all of the interactions. When the power-on button is clicked, Lumos-app shows a dialog with the default semantic tag ("power on"), which the user can then edit for customization.

Example: Fig. 5 (left) illustrates the teaching phase in which a user teaches Lumos-app how to turn on a WINIX air cleaner with a specific wind force level. ① She sets wind force options. ② She then turns on the cleaner. ③ Lumos-app automatically assigns the resource ID ("power on") to the power-on button as the semantic tag. Lumos-app records these interactions for a later step of automatically replaying these operations (see §4.3) and allows its user to customize the tag with a concrete meaning (e.g. power on with wind force level 1).

4.2 Learning from IoT Apps

By design, Lumos-gateway should trigger a programmed action upon observing the network messages that represent an IoT operation. This architecture necessitates Lumos-gateway to identify HTTP(S) requests that an IoT mobile app generates when clickinga particular UI component. Unfortunately, identifying such requests is non-trivial without the understanding of the target app logic because mobile apps often generate other unrelated requests in the background.

To illustrate this, we quantify the amount of traffic generated while performing specific actions on IoT apps listed in Table 1. We capture traffic from each app from the time the target app is started, perform UI interactions as quickly as possible, and stop capturing traffic immediately. We report the average of 10 runs. We perform nine actions using the apps: 1) lock an August door lock; 2) stream to a Chromecast; 3) turn on a HUE bulb; 4) power on a Insteon plug 5) get status from Nest Protect; 6) power on a SmartThings plug; 7) power on a Wemo Insight; 8) play Wink chime; and 9) power on a Winix air cleaner. On average, each app generates 27.4 HTTP transactions. The August door lock app, for example, generates about forty transactions until the door lock is locked, while only one of them is for locking. The main rea-



Action: turn on
URI signature & traffic
http://(.*)/api/(.*)/lights/([0-9]+)/state
Req body signature & traffic
{on: (.*), bri: ([0-9]+)}

Figure 6: Examples of Network signature, UI control, and string ID for HUE app

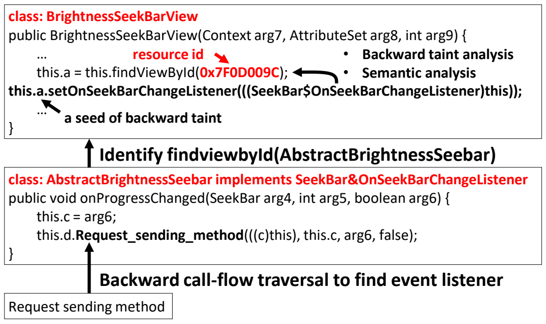
son is that it frequently sends HTTP requests for device status synchronization. According to our manual traffic analysis, all apps in our dataset continuously perform synchronization as long as the app is in the foreground. We suspect this is due to the need for minimizing synchronization delays to enrich user experience.

To isolate transactions a UI component generates, Lumos uses static program analysis. It takes an Android app binary (APK) as an input and then pairs a UI component and the regex signatures of control/status messages that the UI component generates. In addition, it tracks dependencies between messages to dynamically learn fields that come from previous messages (e.g., authorization tokens). Note, the analysis is done in offline.

Building network signatures: To identify the exact control and status message that a UI component generates or displays after receiving it, we start from an existing tool, Extractocol [35], that conducts a static taint analysis to extract network message signatures that an Android app generates or receives. It automatically identifies all app-defined methods that send network messages, extracts message signatures, and outputs them in regular expressions. Fig. 6 shows an example regular expression. Extractocol also provides a call graph of an app as well as its control-flow and interprocedural dataflow graphs. We leverage the information in the next phase of our analysis.

UI control identification: Given an app's call graph and its network signatures, we associate each message signature with a UI component that generates a network message matching the signature. The goal is to precisely identify the network messages generated by the UI actions of a user, when the user configures conditions and control actions. The Android accessibility API allows monitoring which UI components a user interacts with, by providing their resource IDs. If we can associate a resource ID to a network message signature, we can isolate the traffic that the UI component generates from Lumos-gateway that observes traffic.

Lumos takes the following two steps to accomplish the goal. First, it identifies all developer-implemented event listeners that eventually generate network messages. Since every interactable UI component has an event listener, identifying an event listener is equivalent to identifying a UI component. Lumos does a backward call graph analysis for this; it starts from each app-defined method that generates a network message (given by Extractocol) until it either finds an event listener such as `OnClickListener()` and `OnSeekBarChangeListener()`, or reaches the top of the call chain.



```
class: BrightnessSeekBarView
public BrightnessSeekBarView(Context arg7, AttributeSet arg8, int arg9) {
    ...
    this.a = this.findViewById(0x7F0D009C);
this.a.setOnSeekBarChangeListener(((SeekBar$OnSeekBarChangeListener)this));
    ...
}
```

```
class: AbstractBrightnessSeebar implements SeekBar&OnSeekBarChangeListener
public void onProgressChanged(SeekBar arg4, int arg5, boolean arg6) {
    this.c = arg6;
    this.d.Request_sending_method(((c)this), this.c, arg6, false);
}
```

resource id
• Backward taint analysis
• Semantic analysis
a seed of backward taint
Identify findviewbyId(AbstractBrightnessSeebar)
Backward call-flow traversal to find event listener
Request sending method

Figure 7: HUE app UI finding example

Second, Lumos identifies the resource ID to which an event listener is attached. Android allows a developer to register an event listener of a UI component either dynamically (in the app code) or statically (in the app's XML manifest). Statically-registered event listeners are not difficult to identify since it simply requires parsing of the XML manifest. However, dynamically-registered event listeners are not straightforward to identify. Thus, Lumos performs further analysis to identify dynamically-registered event listeners. Lumos starts this analysis by looking up the call site of every event listener registration method (e.g., `setOnClickListener()`). From there, it finds all objects that invoke the event listener registration methods. These objects are user-interactable UI objects. Subsequently, Lumos performs backward taint analysis for each user-interactable UI object until it finds a method that provides the resource ID for the object, such as `findByViewID()`. This resource ID is a hexadecimal, and we can find the corresponding string ID in another XML file (`public.xml`).

Example: Fig. 7 shows how Lumos associates a UI component and the control message it generates using the Philips HUE app. Extractocol outputs a request sending method which Lumos identifies to be (eventually) by `OnSeekBarChangeListener`. Therefore, Lumos looks for a `setOnSeekBarChangeListener` call site that registers the event handler. From the call site, it computes a backward slice that affects the object (`this.a`) to which the event handler was attached. It finally reaches the UI's resource ID ("0x7F0D009C"), which is labeled as "brightness_slider" in `public.xml`.

4.3 Learning from Network Traffic

The UI-signature pairs we extract from an app allows us to distinguish the traffic triggered by the app's UI from others. However, statically-extracted message signatures do not provide run-time values, such as URIs, query strings, and headers. Yet, Lumos should be able to construct a network request for monitoring and controlling the status of a device. This means Lumos must know how to fill in actual values.

To address this, Lumos integrates a run-time packet learning module with information learned from the static analysis. For this, Lumos-app replays all of the interactions recorded in the learning phase of §4.1 to generate network traffic. Lumos-gateway then detects and captures the network messages that match the network signatures extracted from the phase in

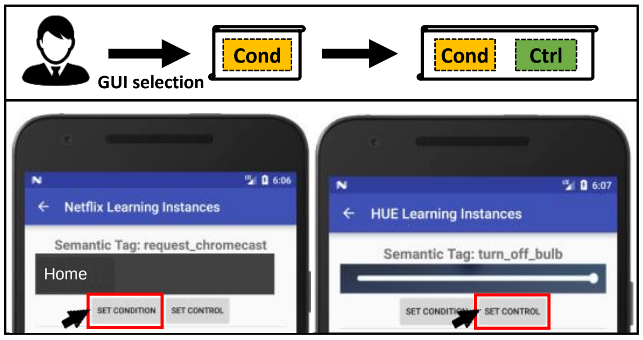
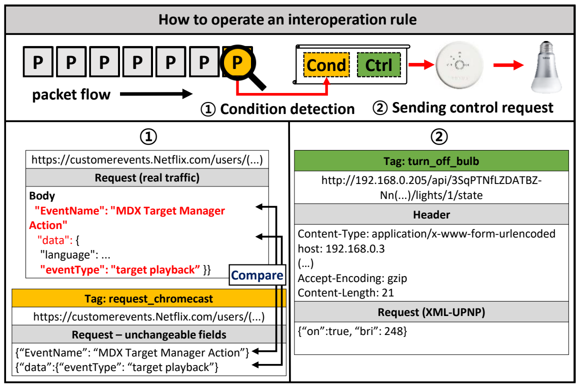


Figure 8: Rule specification on Lumos-app


How to operate an interoperation rule
P
packet flow
Cond
Ctrl
① Condition detection
② Sending control request
①
②
Compare

https://customerevents.Netflix.com/users/(...)

Request (real traffic)

Body
"EventName": "MDX Target Manager Action"
"data": {
"language": ...
"eventType": "target playback" }}

Tag: request_chromecast

https://customerevents.Netflix.com/users/(...)

Request – unchangeable fields

{"EventName": "MDX Target Manager Action"}
{"data":{"eventType": "target playback"}

Tag: turn_off_bulb

http://192.168.0.205/api/3SqPTNfLZDATBZ-Nn(...)/lights/1/state

Header

Content-Type: application/x-www-form-urlencoded
host: 192.168.0.3
(...)
Accept-Encoding: gzip
Content-Length: 21

Request (XML-UPNP)

{"on":true, "bri": 248}

Figure 9: Operation of interoperation run-time

§4.2. Lumos-app automatically performs this step multiple times to obtain multiple message instances for each network signature, as shown in Fig. 5, and uses the values obtained from the observed network messages.

Among the collected network messages, we observed that several attribute values changed over time. This implies that Lumos-gateway should identify attributes that change across message instances and should not perform exact matching on those attribute values when recognizing a pre-configured condition. On the other hand, to generate valid control messages, Lumos must accurately reconstruct them on the fly.

To this end, for each message attribute that changes dynamically, Lumos computes the program slice that are responsible for generating this attribute from a target IoT app via leveraging the program analysis module. Lumos computes this slice in the following steps: 1) The analysis module identifies a seed statement that puts the attribute in the message; 2) It performs a backward taint analysis of computing statements that have transitive data dependencies to the seed statement. We observed that a computed program slice usually consists of a small number of statements, which requires a simple computation (e.g., retrieving and formatting current time). This trend simplifies the computation of a program slice for each attribute as well as enables the execution of this slice independently.

Leverage ideas from mobile code offloading [12, 14, 22, 36, 65], we then execute the program slice on Lumos-gateway. For this, Lumos compiles the program slice to bytecode and Lumos-gateway stores the bytecode. Lumos-gateway, then, executes the bytecode on JVM and provides data that resolves dependencies to generate the desired attribute's value. Finally, Lumos-gateway uses the value to re-construct corresponding network messages. One example is "reqTime" in Fig. 5. When a user turns on the cleaner, Winix app generates the attribute by using `<init>()` and `format()` methods in `java.text.SimpleDateFormat`. Lumos executes the program slice that includes those methods and uses the regenerated "reqTime".

Automated re-learning: The communication between IoT devices and apps often uses REST APIs and, in many cases, an authorization token is used. Because the communication happens (mostly) over HTTPS and the device assumes that a token holder is a trusted party, tokens are often reused (§6.4). However, an authorization token may be updated and old ones may become invalidated, in which case Lumos-gateway has to learn a new token.

Lumos uses two methods to automate this. Lumos-gateway learns a new authorization token by monitoring network traffic and leveraging the dependency relationship between network messages analyzed by the static analysis module (as described in §4.2). In particular, the analysis module builds a dependency graph between messages at the field granularity. For example, when a user operates a door lock by using the August app, the app sends a request including the authorization token from the previous response's header ("x-august-access-token"). If the request is valid, the server sends a corresponding response with a new token, which gets reused in subsequent requests. Using the field-level dependency information, Lumos-gateway automatically tracks the refresh between the August app and server.

In the second method, Lumos actively generates messages that include a new token by leveraging the app. In this method, Lumos-gateway sends a push message to Lumos-app for token re-creation. When a user confirms the push message, Lumos-app replays a UI interaction script recorded from previous learning phase which creates a new token. Eventually, the generated packet from the script involves a new token and Lumos-gateway acquires the token from the packet. This method is used when Lumos-gateway did not observe any token exchange from the network traffic or when an active monitoring or action fails due to token expiration.

4.4 Interoperation Support

Our rule builder allows users to configure desired interoperation rules that consist of multiple conditions and control actions, enabling complex interoperation scenarios across heterogeneous IoT devices. The interoperation run-time monitors the conditions. When a condition is satisfied, the it performs the corresponding control action.

Rule builder: Lumos allows users to compose interoperation rules with conditions and control actions. We support two types of conditions: 1) a "passive" condition is detected by monitoring network traffic without Lumos-gateway sending any traffic, and 2) an "active" condition involves Lumos-gateway asking for a device status directly by sending the learned network message (e.g., motion sensor is active or not). When an active condition is set, Lumos-gateway periodically checks the device status without a user's direct action (e.g., turning on a bulb by using app). To mitigate battery drain due to the use of active conditions, we allow users to configure the interval of such queries.

Note, these queries often do not go to devices that operate on battery, but are handled through wall-powered hubs. For example, SmartThings hub caches the latest status of connected devices to remove unnecessary queries toward the devices [57]. In the case of Wink door lock sensor, it wakes up on regular intervals to communicate with a hub that maintains a mailbox [69]. Of the 15 devices in our dataset, eight devices operate on battery; six of them (75%) leverage their hubs, which support responding to active condition queries.

Fig. 8 shows how Alice configures to turn off her bulb in the living room when playing a Netflix movie on the TV connected to Chromecast. She starts Lumos-app and selects the configuration menu. The app, then, shows a list including UI components with semantic tags (e.g., "turn_off_bulb"). Alice chooses a desired semantic tag ("request_chromecast") in the list, and sets the tag as a passive condition as shown in Fig. 8 (a). As Fig. 8 (b) shows, she then selects the control to turn the bulb off. This concludes Alice's configuration steps. Lumos-app then registers the rule to Lumos-gateway. Our demo video presents this interoperation scenario with Lumos-app: https://youtu.be/enwQwZV0fJE.

Interoperation run-time: Once a rule is registered, Lumos-gateway monitors network traffic to check for the condition of the rule and performs the desired action when the condition is satisfied. For the condition detection, Lumos-gateway uses exact matching for the unchangeable fields in the condition's HTTP request body. Fig. 9 illustrates the process, assuming Alice made an interoperation rule between Chromecast and a HUE bulb. Once the rule is registered, the run-time checks whether a network message matches the condition of the rule. When it matches, the run-time generates a HTTP request that triggers the corresponding action (e.g., turning off a HUE bulb) with an up-to-date authorization token.

4.5 Automation Support and Practical Issues

This section discusses how Lumos enables users to share their setup configurations and how Lumos solves practical issues that arise when interacting with diverse IoT devices.

Information reuse: The network signature-UI pairing is a one-time setup procedure. To facilitate the adoption, Lumos supports import and export of network signature-UI pairs. We envision tech-savvy early-adopters to export this setup information to a file, and then share this file, which contributes to other users avoiding a further program analysis as long as the binary is identical. Note this setup file does not contain any credentials (e.g., authorization token, ID, and password) nor any private data because it only contains network signature-UI pairs extracted from IoT apps, which are public information that anyone can extract. After importing this shared setup file, users only need to perform UI interactions for interoperation rule configuration. We implemented this feature and measured the time to analyze the apps and load the information. For the nine IoT apps we use in our evaluation, Lumos took a total of 36.5 minutes for analysis. However, loading the result only takes 0.95 second for the nine apps. This demonstrates the effort of initial learning can be amortized across users.

Support for SSL: To support Android apps using SSL for encrypted communications with their server-side services, Lumos-gateway deploys a local certificate authority (CA) certificate, which is used prevalently in the industry [18, 50, 51, 64]. Specifically, a user installs a Lumos issued certificate into an IoT apps, which allows Lumos-gateway to inspect traffics between the app and its server-side service. Lumos-gateway then re-encrypts the inspected traffics using a session key derived from the original certificate, thus preventing man-in-the-middle entities other than Lumos-gateway from inspecting network traffic.

We applied this interim solution to eight IoT apps that we evaluated in §6 due to their SSL usage. The solution worked seamlessly without tempering the original functionalities of these apps. A longer term solution would be to embrace solutions for middlebox-based secure monitoring of encrypted traffic [23, 50, 51, 52]. Note, recent studies [43, 72, 73] commonly assume or call for a trusted party that inspects IoT traffic, which matches our usage model.

Learning message field-level semantics: Our framework can be extended to learn the semantics of individual fields and adapt individual fields because our static analysis module identifies how each message field is derived. For example, for Winix air cleaner (Fig. 5), our static analysis module finds out one can observe that the wind force level is encoded in the "controlData" field (e.g., 2 means wind level 1, and 5 means level 3). For HUE light bulbs, it finds out "bri" field indicates the brightness level from the UI control in Fig. 6.

Using this information, we believe one can identify the meaning of each field, and a grass-root type crowd-sourcing approach could bring benefits. As such, we implement a scripting feature in the Lumos-gateway, which allows a tech-savvy user to directly identify and provide network message field values. This is useful in scenarios where it is difficult or undesirable for a user to demonstrate a condition or a control action using an IoT app.

5. IMPLEMENTATION

Our implementation consists of three components, app analysis module, Lumos-app, and Lumos-gateway, totaling 10.5K lines of newly written code. We open-source the code in an anonymized repository: https://bit.ly/2RyGdu8.

The app analysis module extends Extractocol [35] that extracts network signatures and infers data dependency between

Devices			Supported Platform			Lumos
Application	Device	Type	Insteon(3/15)	SmartThings(5/15)	Wink(6/15)	(15/15)
August	August smart lock pro	Door lock	n/a	✓	n/a	✓
Netflix	Chromecast	Streaming-dongle	n/a	n/a	n/a	✓
Philips HUE	HUE	Bulb	n/a	✓	✓	✓
Insteon	Insteon door sensor	Door sensor	✓	n/a	n/a	✓
	Insteon plug	Smart plug	✓	n/a	n/a	✓
	Insteon water leak sensor	Water leak sensor	✓	n/a	n/a	✓
Nest	Nest Protect	CO&smoke detector	n/a	n/a	✓	✓
SmartThings	SmartThings plug	Smart plug	n/a	✓	n/a	✓
	SmartThings motion sensor	Motion sensor	n/a	✓	✓	✓
	SmartThings door sensor	Door sensor	n/a	✓	n/a	✓
Wemo	Wemo Insight Plug	Smart plug	n/a	n/a	n/a	✓
Wink	Wink chime	Siren&chime	n/a	n/a	✓	✓
	Wink door sensor	Door sensor	n/a	n/a	✓	✓
	Wink motion sensor	Motion sensor	n/a	n/a	✓	✓
Winix	Winix air cleaner	Air cleaner	n/a	n/a	n/a	✓

Table 1: IoT devices, platforms and interoperability (Insteon, SmartThings, Wink and Lumos).

network messages. To identify Signature-UI relationships, we modify the signauture building module with 4K lines of code (LoC). Lumos-app is built on top of SUGILITE [38] that supports automating arbitrary tasks for Android apps using Android Accessibility API. We add three modules–UI learning module, interoperation manager, and active message generator–that consist of 5K LoC. Finally, we develop Lumos-gateway by writing 1.5K LoC on top of mitmproxy [49], an open source HTTPS proxy. We add our network message learning module, the interoperation manager, and REST API that allows Lumos-app to control our gateway.

6. EVALUATION

We evaluate Lumos by answering four key questions:

- *Does Lumos enable interoperation across diverse IoT devices and platforms?* **(RQ1)**
- *Is Lumos-gateway capable of emulating key functionalities of the IoT apps?* **(RQ2)**
- *How easy is it for a user to configure interoperation rules using Lumos?* **(RQ3)**
- *How effective is the automated re-learning of authorization tokens?* **(RQ4)**

We evaluate Lumos using 15 commercial IoT devices and nine apps that control them, as listed in Table 1. To demonstrate the feasibility of Lumos in configuring interoperations across different platforms and devices, we use standalone IoT devices and devices from three popular home IoT platforms. Five devices are standalone: an August smart lock, Chromecast, a HUE bulb, Nest Protect, and a Wemo Insight plug. The rest belongs to one of the three popular platforms: SmartThings, Wink, and Insteon. We emphasize that our benchmarks cover the *largest* evaluation set. Existing works [2, 3, 4, 6, 7, 10, 16, 20, 39, 47] either present no use-cases involving commercial IoT devices or test their systems with at most three IoT devices [39].

6.1 Supporting Diverse IoT Platforms (RQ1)

We first evaluate whether Lumos can emulate the key functionality of IoT apps to monitor and control the devices. Table 1 summarizes the coverage of Lumos compared to the three popular IoT platforms. The "device" column family lists 15 IoT devices and corresponding Android apps. *Platform-native* apps (shaded) provides programmable features among various devices within the platform. In contrast, *stand-alone* apps do not provide any interoperation features. The "supported platform " column family shows how many devices the three popular IoT platforms support. In our evaluation benchmarks, SmartThings and Wink support only five and six devices, respectively. Insteon platform devices cannot interoperate with any of the other two platforms, clearly demonstrating their the mutually exclusive nature. SmartThings is the only platform that provides cross-platform interoperation by providing an external API [61]. However, understanding this and programming the desired interoperations demands domain expertise and heavy engineering efforts. In contrast, Lumos provides a much wider coverage than existing platforms (all 15 devices supported). Moreover, it is designed for non-experts who can teach Lumos of their desired interoperations via direct UI interactions.

6.2 Emulating IoT App Functionalities (RQ2)

Our next question is whether or not Lumos is able to emulate key functionalities of the IoT devices for each feature they support. Table 2 shows the coverage results compared to OpenT2T. The "OpenT2T" column shows whether the feature is supported by OpenT2T, and the "Lumos" column indicates whether Lumos is able to perform the same operation.

#	Device	App function	OpenT2T	Lumos	#	Device	App function	OpenT2T	Lumos
1	August smart lock pro	lock/unlock	No schema	PC, C	9	SmartThings motion sensor	active or not	✓	PC, AC
		get status history		PC, AC			get status history	Not supported	PC, AC
2	Chromecast (Netflix)	request Chromecast	No schema	PC	10	SmartThings door sensor	open or not	No schema	PC, AC
							get status history		PC, AC
3	HUE	turn on/off	✓	PC, C	11	Wemo Insight Plug	power on/off	✓	PC, C
		change brightness	✓	PC, C					
		change color	✓	PC, C			get current voltage	Not supported	PC, AC
		get status	Not supported	PC, AC					
4	Insteon door sensor	open/close status	No schema	PC, AC	12	Wink chime	play bell	No schema	PC, C
5	Insteon plug	power on/off	✓	PC, C	13	Wink door sensor	open or not	No schema	PC, AC
							get status history		PC, AC
6	Insteon water leak sensor	get leak status	✓	PC, AC	14	Wink motion sensor	active or not	✓	PC, AC
							get status history	Not supported	PC, AC
7	Nest Protect	get CO status	✓	PC, AC	15	Winix air cleaner	turn on/off	No schema	PC,C
		get smoke status	Not supported	PC, AC			change wind force		PC, C
		get battery health	Not supported	PC, AC			get current status		PC, AC
8	SmartThings plug	power on/off	✓	PC, C			–		
		get status history	Not supported	PC, AC					

Table 2: Functionality comparison (App/Lumos/OpenT2T schemas). 'C' stands for control, 'PC' for passive condition, and 'AC' for active condition. Features in grey represent status query functions. The rest are control functions.

Using real-world IoT devices, we examine whether each individual control and status messages are recognized and Lumos-gateway's control action triggers the a correct device behavior. App features that represent device status queries (shaded in the table) can be used as a passive condition ('PC') and/or an active condition ('AC') in Lumos. (§4.4). Control functions that trigger an action on a device can be used as passive conditions ('PC') and/or control actions ('C'). Note, control actions and active conditions require Lumos-gateway to generate actual messages and succeed in achieving the desired outcome. To support passive conditions ('PC'), Lumos-gateway must be able to recognize the network messages.

Lumos supports all 18 status query features as either passive and active conditions and all 11 control features as either passive conditions and control actions. In fact, with one exception (Chromecast), it supports both. In contrast, seven out of 15 devices (e.g., door locks, streaming dongles, etc.) do not have an OpenT2T schema. To enable interoperability using OpenT2T, vendors need to have a pre-defined schema for a new device type. Out of the 29 features that 15 devices support, only 10 are supported by OpenT2T. Even the devices that are supported by OpenT2T do not expose all their features (e.g., Nest Protect). This is because either the schema only defines minimal features for each device type or the vendor partially implements the translation from the common schema to its own features. The large difference between Lumos and OpenT2T (29 versus 10 features supported) demonstrates the benefit of our approach, which is not tied to any common interface that must be agreed upon, and does not require vendor participation.

We observe Lumos does not support control action for the Netflix-Chromecast case. When the Netflix app goes through a server to talk to a Chromecast device, and their messages do not carry context information (e.g., what movie is being played). This is the only case we have observed where the API violates statelessness. Thus, although, Lumos-gateway re-generates the message corresponding to the UI of Fig. 8 (a), Lumos does not trigger any action on Chromecast.

Mission	# devices	Clicks$_{Teach}$	Clicks$_{Conf}$	Condition
Tutorial$_{\{1,1\}}$	2	5	5	-
Insteon$_{\{1,1\}}$	2	6	6	-
Stand-alone$_{\{1,2\}}$	3	6	6	-
SmartThings$_{\{2,2\}}$	4	9	10	AND
Wink$_{\{2,2\}}$	3	10	9	OR

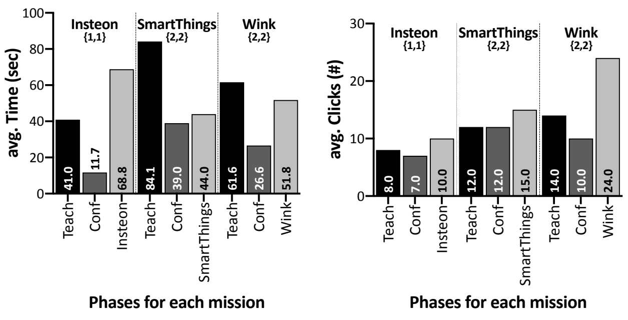
Table 3: Mission overview: "Clicks$_{Teach}$" stands for the minimum number of clicks required for teaching, "Clicks$_{Conf}$" stands for the minimum number of clicks required for configuration. "Condition" stands for the relationship among multiple conditions.

6.3 User study (RQ3)

To answer our usability question, we conducted a user study where we asked our participants to configure Lumos with new interoperations for five given missions.

Participants and environment: Twenty-four participants were recruited, who were all university students. The participants were required to be active smartphone users of age 18 or older. We asked them to mark their own IoT experience on a three-point scale: 1) "no experience"; 2) "having experience with at least one IoT device"; and 3) "having programming experience on at least one IoT platform". We then grouped the participants by the scale. Each group consists of nine, eight, and seven people.

The experiment was conducted in a test room with 15 IoT devices and three IoT platforms, as listed in Table 1. We installed Lumos-gateway which runs on a Ubuntu 16.04 PC with Intel core i5 quadcore x 2.81 GHz CPU and 16 G RAM. We provided the participants a smartphone (Xiaomi Note 5) that includes nine IoT apps and Lumos-app.



(a) The avg. completion time (b) The avg. completion click

Figure 10: The comparison of three platform-native apps and lumos-app

Missions: We assigned each user five missions. These missions are based on common interoperation scenarios from our motivation. To directly compare Lumos-app with apps provided by the existing IoT platforms (SmartThings, Wink, and Insteon) that provide programmable interfaces for automation, we designed three missions to program an interoperation rule using the existing platforms as well as Lumos.

The first mission was used as a tutorial, where the experimenter showed a participant how to teach Lumos and how to configure interoperations with our system. The other missions were given to the participants in a random order. After the tutorial, we gave users some time to get themselves familiarized with the apps (August, Netflix, Insteon, SmartThings, Wemo, HUE, and Wink). During each mission, we did not answer questions on how to use Lumos-app, but we responded to the requests for clarification on the mission specifications. The missions are as follows.

Tutorial$_{\{1,1\}}$*:* The mission assumes that a user wants to turn off a bulb connected to Wemo when streaming a Netflix movie through Chromecast. In this mission, we demonstrate the procedure of configuring interoperation using Lumos-app with two standalone devices (Chromecast and Wemo Insight). The participants need to teach Lumos-app using Netflix and Wemo apps and then configure interoperation that is made up of *one condition* and *one control* (hence $\{1, 1\}$). The standard procedure for this mission has five clicks for teaching ("Clicks$_{Teach}$") and five clicks for configuration ("Clicks$_{Conf}$").

Insteon$_{\{1,1\}}$*:* This mission requires participants to configure interoperation to turn off an Insteon plug when an Insteon water leak sensor is triggered. This mission is relatively easy because participants use Insteon app only to control those devices. This interoperation consists of *one condition* and *one control* with six "Clicks$_{Teach}$" and six "Clicks$_{Conf}$".

Stand-alone$_{\{1,2\}}$*:* This mission requires to turn off a Winix air cleaner and a Wemo Insight when an August door lock is locking, which consists of *one condition* and *two controls*. Participants need to teach Lumos-app using three IoT apps, as listed in Table 3. The standard procedure for this mission has eight "Clicks$_{Teach}$" and six "Clicks$_{Conf}$". Note only Lumos supports this, and no existing IoT platform is able to support.

SmartThings$_{\{2,2\}}$*:* This mission assumes that a user turns off a SmartThings plug and a HUE bulb connected to a SmartThings hub if a SmartThings motion sensor does not detect a motion and a door sensor is closed. Therefore, the interoperation for this mission consists of *two conditions connected by "AND"* and *two controls*. To configure the interoperation, participants teach Lumos-app by using SmartThings app to control four devices (motion sensor, door sensor, plug, and bulb). The standard procedure has nine "Clicks$_{Teach}$" and eight "Clicks$_{Conf}$". Besides, participants need to select the condition ("AND").

Wink$_{\{2,2\}}$*:* This mission requires participants to configure interoperation to play a Wink siren when a Wink door sensor is opened, or a Wink motion sensor detects a motion. Therefore, the interoperation consists of *two conditions by connecting "OR"* and *one control*, which requires 12 "Clicks$_{Teach}$" and nine "Clicks$_{Conf}$". Like SmartThings$_{\{2,2\}}$, participants need to choose an appropriate condition ("OR").

Procedure: The experiment took about one hour per participant. After signing the IRB-approved consent form, each participant performed a full rehearsal of the tutorial with the experimenter and was given some time to use nine IoT apps to control IoT devices. They also learned how to configure automation with platform-native apps (SmartThings, Wink, and Insteon app) during that time. The time for this step took about 10 minutes. Following the tutorial, the experimenter gave four missions in a random order. For a given mission, a participant first performed it using Lumos-app. For the three missions that has existing IoT-platform support (Insteon$_{\{1,1\}}$,SmartThings$_{\{2,2\}}$, and Wink$_{\{2,2\}}$), the participants performed them again by using the platform-native apps. We note that there are some nuanced differences in usability across different platforms, which we will discuss later in the section. As the final step, all participants filled out a usability and usefulness questionnaire. Participants were compensated for about \$13 for their time.

For each mission, we measured the number of clicks and time required for mission completion for the teaching step and the configuration step separately. In the teaching step, Lumos-app automatically records the count of click operations from "start recording" click to "end recording" click. Likewise, in the configuration step, Lumos-app records the count from "interoperation manager" click to "finish configuration" click.

Mission results: Overall, 23 out of the 24 (95.8%) participants succeeded in all four missions and one succeeded in three missions. The participant who failed a mission did not complete SmartThings$_{\{2,2\}}$ because the participant prematurely clicked "configuration finish" button from confirmation dialog before completing configuration; the participant was supposed to select the next condition and the 'AND' operator to combine the two conditions. Fig. 10 compares Lumos-app and three platform-native apps, in terms of mission completion times and clicks. We also measured the same metrics for completing Insteon$_{\{1,1\}}$,SmartThings$_{\{2,2\}}$, and Wink$_{\{2,2\}}$ on platform-native apps for comparison.

How does Lumos-app compare to platform-native apps in terms of time and effort to configure interoperation? To demonstrate that Lumos-app has reasonable entry barriers, we compared platform-native apps and Lumos-app for Insteon$_{\{1,1\}}$, SmartThings$_{\{2,2\}}$, and Wink$_{\{2,2\}}$ (Fig. 10). In general, using Lumos-app for all missions except Insteon$_{\{1,1\}}$ took more time, and equal or more number of clicks compared to existing platform-native apps. This is because unlike platform-native apps, Lumos involves teaching which takes most of the user time. When looking at time and clicks for configuration alone excluding teaching, Lumos-app is considerably faster. We note that teaching results are reusable. If Lumos-app has already finished learning a specific IoT device's action (e.g., turning on a bulb), users just need to decide whether it is a condition or a control in the configuration phase. The teaching phase can even be omitted using information sharing and automated re-learning functionalities that Lumos supports (See §4.5), reducing the time and effort considerably for enabling interoperation.

How does Lumos-app compare to platform-native apps in terms of usability? We found nuanced differences in the programmability of the three IoT platforms. Unlike Lumos, the platform-native apps do not easily support relationships to connect multiple conditions, which are required by our missions, Wink$_{\{2,2\}}$ and SmartThings$_{\{2,2\}}$. SmartThings supports this only through its script-based programming API to complement their usability [58]. Wink and Insteon do not provide ways to combine multiple conditions. To work around this, we prepared a SmartThings app script (about 80 lines) in advance for SmartThings$_{\{2,2\}}$ and uploaded it to the SmartThings marketplace from which the participants downloaded and chose a proper device required by the script. For Wink$_{\{2,2\}}$ that uses two conditions connected by "OR" condition, participants had to resort to breaking down the rule into two separate rules. This is the reason Wink$_{\{2,2\}}$ required more clicks than other missions. Finally, the Insteon$_{\{1,1\}}$ mission required the most time when using the Insteon platform app (Fig. 10(a)). This is because Insteon takes some time (about 30s) to apply automation rules to the desired devices after automation rule setup is done [27]. For this, six participants left negative feedback that the process was boring and inefficient. Unlike Insteon, Lumos instantly configures the rules without any noticeable inefficiency. Although we acknowledge that the usability of platform-native apps can improve with feature updates, our comparison to the current status quo shows that Lumos offers a more efficient programming environment and comparable levels of effort and difficulty as platform-native apps.

Learning curve for users. We look at whether the time to complete missions decrease as users gained more experience. Recall that we assigned the missions to each participant in a random order, making the order of missions differ from one participant to another. Fig. 11 shows the average time to complete each mission by two groups: participants who

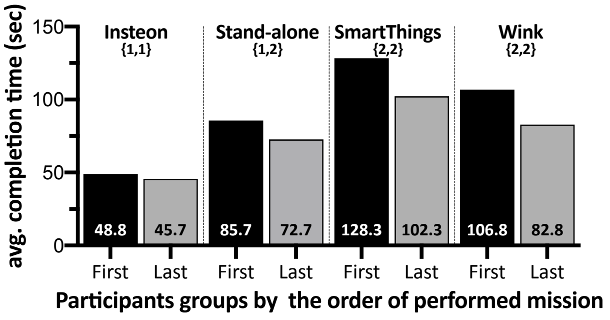


Figure 11: The average completion time (sec) grouped by the order of start mission. "First" stands for a group who conducted a specific mission first. "Last" stands for a group who performed a specific mission last.

did the mission as the first mission and participants who did the mission as the last mission. $First_i$ denotes the average time of those who started with mission i, and $Last_i$ indicates the average time of those who performed mission i as their last mission. Interestingly, for each mission i, the average time of $Last_i$ is smaller than $First_i$, which means that participants tend to get a better understanding with the last mission. Notably, the difference of average time between $First_i$ and $Last_i$ is more noticeable for more complex missions (Wink$_{\{2,2\}}$ and SmartThings$_{\{2,2\}}$) that involve more conditions than other missions. This shows participants quickly gained familiarity with Lumos.

6.4 Authorization Token Management (RQ4)

IoT devices often use authorization tokens to accept commands only from valid endpoints. Lumos maintains up-to-date authorization tokens even when they change over time using passive and active re-learning.

To evaluate this, we consider all 14 devices (controlled by eight apps) whose features Lumos is able to use as active conditions or control actions. Note, monitoring for passive conditions does not require any packet generation. Out of the eight apps, six employ authorization tokens (Table 4).

To verify passive re-learning, after the initial learning, we manually trigger a token change condition (factory reset), which causes the device to issue a new token to the app. To test whether the Lumos-gateway learns the new token, we generate a control message from Lumos-gateway and observe whether the control messages contain the latest authorization token and trigger the correct behavior. We have successfully validated that for all cases the authorization token is renewed and that the messages trigger the correct actions, demonstrating the effectiveness of the approach. Out of the six apps, August employs the most aggressive token update strategy—every request triggers the August gateway to respond with a new authorization token. Even in this case, we have validated that Lumos-gateway successfully re-learned a new authorization token by identifying the dependency from the previous request/response messages from the August gateway. Although passive re-learning is effective in learning a new

Device	Token lifetime (2 weeks)	Expire condition	Re-learning time
August	not expired	factory reset device	3.1s
Philips HUE	not expired	factory reset hub	4.2s
Insteon door sensor		factory reset hub	
Insteon plug	not expired	factory reset device	7.3s
Insteon water leak sensor			
Nest Protect	not expired	factory reset device	7.9s
SmartThings plug		factory reset hub	
SmartThings motion sensor	not expired	restart hub	6.3s
SmartThings door sensor		factory reset device	
Wemo Insight Plug	no token used	-	-
Wink chime		factory reset hub	
Wink door sensor	not expired	factory reset device	3.3s
Wink motion sensor			
Winix Air Cleaner	no token used	-	-

Table 4: Lifetime of authorization tokens, conditions for token expiration, and automated active re-learning.

token from network traffic, when there has been no use for a long time, the token might expire and Lumos-gateway might not be able to detect a new token. To handle this, Lumos automatically performs active re-learning when Lumos-gateway finds the current token is invalid. To evaluate this, we manually invalidate the authorization token and verify whether Lumos-gateway sends a push message to Lumos-app to trigger automatic relearning. We have successfully observed that Lumos-gateway triggers Lumos-app to perform the scripted interaction. The last column of Table 4 reports the time it takes to re-learn the token (averaged over 10 runs) including the time to run the UI automation script that re-generates the token. Lumos takes 5.6 seconds on average for re-learning.

Finally, to understand how frequently Lumos-gateway needs to update the authorization tokens, we measure how long a token is valid for and identify what conditions render existing tokens invalid. We tested each functionality for two weeks. To identify token expiry conditions, we performed 1) resetting a target IoT device, 2) restarting the hub device connected with the IoT device, 3) repairing the hub device and the IoT device, and 4) resuming/restarting the app. We then checked whether the old authorization token has been invalidated for each functionality after performing each operation.

Table 4 summarizes our findings. It shows the IoT mobile apps and whether the corresponding devices use any authorization token. The second and third columns show the validity of authorization tokens during the two weeks and the expiration conditions, respectively. Interestingly, although August updates its authorization tokens frequently, old tokens do not expire for at least two weeks unless the doorlock or the hub is factory-reset. Upon further inspection, we have discovered all tokens (as many as 500 we generated) are valid during the two weeks of our study. Wemo and Winix apps do not use any authorization tokens. The HUE app uses a constant token which is assigned when pairing the app with the HUE hub. Thus, Lumos does not have to re-learn the token unless the user pairs the app again with the HUE hub.

7. RELATED WORK

IoT architectures for device interoperability: Many studies [2, 3, 4, 6, 7, 10, 16, 20, 47] present middleware-based approaches to improve interoperation among heterogeneous IoT devices. Some of the approaches [2, 3] leverage a smartphone as a mobile gateway and build middleware on top of the gateway. Other approaches design middleware based on an interoperable IoT hub which manages sensors and actuators [6, 47]. They offer REST APIs which write and load data from heterogeneous IoT devices. However, all approaches in this category necessarily require additional engineering effort from vendors. In contrast, Lumos focuses on supporting interoperation with no vendor support.

Program analysis: We build on the prior work that uses static analysis for network protocols [5, 8, 9, 13, 15, 35, 66, 67]. Reformat [67], Extractocol [35], and ProDecoder [66] focus on extracting network message formats by inspecting application binaries. The main difference is that our system does not simply analyze network protocols but identifies the relationships between UI components and network behavior.

Programming by Demonstration (PBD): Programming by Demonstration (PBD) is a useful technique to allow end users to automate their UI actions without requiring any programming [11, 37, 38, 39, 40, 41]. SUGILITE [38] supports automating arbitrary tasks for Android apps using Android's Accessibility API. EPIDOSITE [39] focuses on mitigating interoperability problem of IoT devices by using Android's accessibility API. The evaluation of EPIDOSITE is rather preliminary and uses only three devices. Moreover, the limitation of using the accessibility API is that an IoT app must be in the foreground whenever EPIDOSITE needs to detect a condition or issue a control action. In contrast, Lumos avoids the limitation as we focus on monitoring and emulating network level interactions between IoT apps and devices.

8. CONCLUSION

Interoperability is crucial in enabling new IoT services and creating a rich IoT ecosystem. Despite the development of large IoT platforms and industry efforts for interoperability, we observe that the ecosystem is still largely fragmented. We posit that efforts that rely on vendor support face a fundamental challenge in deployment. To address the problem, Lumos takes a best effort approach that leverages information embedded in IoT apps and combines it with semantic information from users. Lumos automates many aspects of interoperability and learning to minimize user effort. Our evaluation across 15 IoT devices from popular vendors demonstrates the feasibility of the approach. In particular, we show devices that are not interoperable and are not designed to be so can work together. Lumos provides a wider feature and device coverage than existing open source efforts, such as Open Translator to Things (OpenT2T). In particular, our user study with 24 participants demonstrates that Lumos-app offers a usable programming framework that enables interoperation of IoT devices, offers interoperation features that are not available on commodity IoT platforms, and requires reasonable effort compared to three popular IoT platform-native apps.

References

[1] AllJoyn. an open source software framework that makes it easy for devices and apps to discover and communicate with each other. `https://openconnectivity.org/developer/reference-implementation/alljoyn`, 2019.

[2] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio. A mobile multi-technology gateway to enable IoT interoperability. In *IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 259–264. IEEE, 2016.

[3] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio. Enabling IoT interoperability through opportunistic smartphone-based mobile gateways. *Journal of Network and Computer Applications*, 81:74–84, 2017.

[4] A. Badii, J. Khan, M. Crouch, and S. Zickau. Hydra: Networked embedded system middleware for heterogeneous physical devices in a distributed architecture. In *Final External Developers Workshops Teaching Materials*, page 4, 2010.

[5] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafò. Towards automatic protocol field inference. *Computer Communications*, 84:40–51, 2016.

[6] M. Blackstock and R. Lea. IoT Interoperability: A Hub-based Approach. In *International Conference on the Internet of Things (IoT)*, pages 79–84. IEEE, 2014.

[7] A. Bröring, S. Schmid, C.-K. Schindhelm, A. Khelil, S. Kabisch, D. Kramer, D. Le Phuoc, J. Mitic, D. Anicic, and E. Teniente López. Enabling IoT Ecosystems through Platform Interoperability. *IEEE software*, 34(1):54–61, 2017.

[8] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling Active Botnet Infiltration usingAutomatic Protocol Reverse-Engineering. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 621–634. ACM, 2009.

[9] J. Caballero and D. Song. Automatic Protocol Reverse-Engineering: Message Format Extraction and Field Semantics Inference. *Computer Networks*, 57(2):451–474, 2013.

[10] M. Caporuscio, P.-G. Raverdy, and V. Issarny. ubiSOAP: A Service-Oriented Middleware for Ubiquitous Networking. *IEEE Transactions on Services Computing*, 5(1):86–98, 2012.

[11] J.-H. Chen and D. S. Weld. Recovering from Errors during Programming by Demonstration. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 159–168. ACM, 2008.

[12] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[13] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol Specification Extraction. In *Security and Privacy, IEEE Symposium on*, pages 110–125. IEEE, 2009.

[14] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 49–62. ACM, 2010.

[15] W. Cui, V. Paxson, N. Weaver, and R. H. Katz. Protocol-Independent Adaptive Replay of Application Dialog. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2006.

[16] P. Desai, A. Sheth, and P. Anantharam. Semantic Gateway as a Service Architecture for IoT Interoperability. In *IEEE International Conference on Mobile Services*, pages 313–319. IEEE, 2015.

[17] M. Díaz, C. Martín, and B. Rubio. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer applications*, 67:99–117, 2016.

[18] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson. The Security Impact of HTTPS Interception. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2017.

[19] M. Elkhodr, S. Shahrestani, and H. Cheung. The Internet of Things: New Interoperability, Management and Security Challenges. *arXiv preprint arXiv:1604.04824*, 2016.

[20] K. Gama, L. Touseau, and D. Donsez. Combining heterogeneous service technologies for building an Internet of Things middleware. *Computer Communications*, 35(4):405–417, 2012.

[21] Google. Chromecast is a line of digital media players developed by Google. `https://store.google.com/product/chromecast`, 2019.

[22] M. S. Gordon, D. K. Hong, P. M. Chen, J. Flinn, S. Mahlke, and Z. M. Mao. Accelerating Mobile Applications through Flip-flop Replication. In *Proceedings of the 13th Annual International Conference on Mobile*

Systems, Applications, and Services, pages 137–150. ACM, 2015.

[23] J. Han, S. Kim, J. Ha, and D. Han. SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module. In *Proceedings of the First Asia-Pacific Workshop on Networking*, pages 99–105. ACM, 2017.

[24] Home Assistant. Developer document. `https://developers.home-assistant.io/en/`, 2019.

[25] Home Assistant. Open source home automation. `https://www.home-assistant.io/`, 2019.

[26] IFTTT. The free way to get all your apps and devices talking to each other. `https://ifttt.com/`, 2019.

[27] Insteon. Creating a Scene with the Insteon Hub (Android). `https://www.insteon.com/support-knowledgebase/2015/7/7/creating-a-scene-with-the-insteon-hub-android`, 2019.

[28] M. Intelligence. Smart Homes Market - Growth, Trends, and Forecast (2019 - 2024). `https://www.mordorintelligence.com/industry-reports/global-smart-homes-market-industry`, 2018.

[29] IoT Agenda. A smart home app is an application used to remotely control and manage connected non-computing devices in the home, typically from a smartphone or tablet. `https://internetofthingsagenda.techtarget.com/definition/smart-home-app-home-automation-app`, 2015.

[30] IoT Agenda. Why interoperability holds the keys to the smart home. `https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/Why-interoperability-holds-the-keys-to-the-smart-home`, 2016.

[31] IoT analytics. Iot platforms company list 2017. `https://iot-analytics.com/iot-platforms-company-list-2017-update/`, 2017.

[32] IoT for all. Smart Home Interoperability: The Key Hurdles. `https://www.iotforall.com/smart-home-interoperability-key-hurdles/`, 2019.

[33] IoTivity. An open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the Internet of Things. `https://iotivity.org/`, 2019.

[34] J. Kiljander, A. D'elia, F. Morandi, P. Hyttinen, J. Takalo-Mattila, A. Ylisaukko-Oja, J.-P. Soininen, and T. S. Cinotti. Semantic interoperability architecture for pervasive computing and internet of things. *IEEE access*, 2:856–873, 2014.

[35] J. Kim, H. Choi, H. Namkung, W. Choi, B. Choi, H. Hong, Y. Kim, J. Lee, and D. Han. Enabling Automatic Protocol Behavior Analysis for Android Applications. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 281–295. ACM, 2016.

[36] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *2012 Proceedings IEEE Conference on Computer Communications (INFOCOM)*, pages 945–953. IEEE, 2012.

[37] T. A. Lau and D. S. Weld. Programming by Demonstration:An Inductive Learning Formulation. In *Proceedings of the 4th international conference on Intelligent user interfaces*, pages 145–152. ACM, 1998.

[38] T. J.-J. Li, A. Azaria, and B. A. Myers. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 6038–6049. ACM, 2017.

[39] T. J.-J. Li, Y. Li, F. Chen, and B. A. Myers. Programming IoT Devices by Demonstration Using Mobile Apps. In *International Symposium on End User Development*, pages 3–17. Springer, 2017.

[40] T. J.-J. Li and O. Riva. Kite: Building Conversational Bots from Mobile Apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 96–109. ACM, 2018.

[41] H. Lieberman. *Your Wish is My Command: Programming By Example*. Morgan Kaufmann, 2001.

[42] Lifewire. What Is Google Brillo and Weave? `https://www.lifewire.com/what-is-brillo-weave-1616282`, 2019.

[43] V. Martin, Q. Cao, and T. Benson. Fending off IoT-Hunting Attacks at Home Networks. In *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*, pages 67–72. ACM, 2017.

[44] Microsoft. OpenT2T github. `https://github.com/openT2T/translators`, 2015.

[45] Microsoft. OpenT2T - An Open Source project to translate common IoT schemas to specific hardware devices. `https://www.opentranslatorstothings.org/`, 2019.

[46] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma. A gap analysis of Internet-of-Things platforms. *Computer Communications*, 89:5–16, 2016.

[47] J. Mineraud and S. Tarkoma. Toward interoperability for the Internet of Things with meta-hubs. *arXiv preprint arXiv:1511.08063*, 2015.

[48] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad hoc networks*, 10(7):1497–1516, 2012.

[49] mitmproxy. A free and open source interactive HTTP proxy. https://mitmproxy.org/, 2019.

[50] D. Naylor, R. Li, C. Gkantsidis, T. Karagiannis, and P. Steenkiste. And Then There Were More: Secure Communication for More Than Two Parties. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 88–100. ACM, 2017.

[51] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 199–212. ACM, 2015.

[52] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy. SafeBricks: Shielding Network Functions in the Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Renton, WA*, 2018.

[53] Reddit. Homeautomation forum. https://www.reddit.com/r/HomeAutomation, 2019.

[54] ResearchAndMarkets.com. Smart Home Market by Product, Software & Services, and Region - Global Forecast to 2024. https://www.researchandmarkets.com/research/r4xlbj/smart-home-market, 2019.

[55] Sam Solutions. A Mobile App for a Smart Home Solution: Takeaways and Guidelines. https://www.sam-solutions.com/blog/why-is-a-mobile-app-important-for-an-efficient-smart-home-solution/, 2019.

[56] Samsung. SmartThings devices-integrations forum. https://community.smartthings.com/c/devices-integrations, 2019.

[57] SmartThings. SmartThings hub stores the latest value locally. https://community.smartthings.com/t/does-device-latestvalue-query-the-device-or-just-the-hub-battery-usage-concerns/61228, 2016.

[58] SmartThings. Developer Documentation. = https://smartthings.developer.samsung.com/docs/index.html, 2019.

[59] SmartThings. Publish your devices to SmartThings. https://smartthings.developer.samsung.com/distribution, 2019.

[60] SmartThings. Support Products. https://www.smartthings.com/products, 2019.

[61] SmartThings. Web Service. https://docs.smartthings.com/en/latest/smartapp-web-services-developers-guide/overview.html, 2019.

[62] Z. Song, A. A. Cárdenas, and R. Masuoka. Semantic middleware for the internet of things. In *Internet of Things (IoT)*, pages 1–8. IEEE, 2010.

[63] TechTarget. The IoT journey for manufacturers: concept, production and beyond. https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/The-IoT-journey-for-manufacturers-Concept-production-and-beyond, 2017.

[64] TLSeminar. TLS Interception and SSL Inspection. https://tlseminar.github.io/tls-interception/, 2017.

[65] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt. Cloudlets: Bringing the Cloud to the Mobile User. In *Proceedings of the third ACM workshop on Mobile cloud computing and services*, pages 29–36. ACM, 2012.

[66] Y. Wang, X. Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, and L. Guo. A Semantics Aware Approach toAutomated Reverse Engineering Unknown Protocols. In *IEEE International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, 2012.

[67] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace. ReFormat: Automatic Reverse Engineering ofEncrypted Messages. In *European Symposium on Research in Computer Security*, pages 200–215. Springer, 2009.

[68] A. Whitmore, A. Agarwal, and L. Da Xu. The Internet of Things - A survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, 2015.

[69] Wink Labs. Wink Door/Window sensor manual. http://manuals-backend.z-wave.info/make.php?lang=en&sku=Wink%20D/W%20Sensor&cert=ZC10-17075685, 2019.

[70] Wink Labs. Wink support products. https://www.wink.com/help/products, 2019.

[71] C. Yang, B. Yuan, Y. Tian, Z. Feng, and W. Mao. A Smart Home Architecture Based on Resource Name Service. In *2014 IEEE 17th International Conference on Computational Science and Engineering*, pages 1915–1920. IEEE, 2014.

[72] T. Yu, S. K. Fayaz, M. P. Collins, V. Sekar, and S. Seshan. PSI: Precise Security Instrumentation for Enterprise Networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2017.

[73] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, page 5. ACM, 2015.

[74] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta. The Internet of Things Has a Gateway Problem. In *Proceedings of the 16th international workshop on mobile computing systems and applications*, pages 27–32. ACM, 2015.