A Study of Data Store-based Home Automation

Kaushal Kafle, Kevin Moran, Sunil Manandhar, Adwait Nadkarni, Denys Poshyvanyk
William & Mary, Williamsburg, VA, USA
{kkafle,kpmoran,smanandhar,nadkarni,denys}@cs.wm.edu

ABSTRACT

Home automation platforms provide a new level of convenience by enabling consumers to automate various aspects of physical objects in their homes. While the convenience is beneficial, security flaws in the platforms or integrated third-party products can have serious consequences for the integrity of a user's physical environment. In this paper we perform a systematic security evaluation of two popular smart home platforms, Google's Nest platform and Philips Hue, that implement home automation "routines" (*i.e.,* trigger-action programs involving apps and devices) via manipulation of state variables in a *centralized data store*. Our semi-automated analysis examines, among other things, platform access control enforcement, the rigor of non-system enforcement procedures, and the potential for misuse of routines. This analysis results in *ten* key findings with serious security implications. For instance, we demonstrate the potential for the misuse of smart home routines in the Nest platform to perform a lateral privilege escalation, illustrate how Nest's product review system is ineffective at preventing multiple stages of this attack that it examines, and demonstrate how emerging platforms may fail to provide even bare-minimum security by allowing apps to arbitrarily add/remove other apps from the user's smart home. Our findings draw attention to the unique security challenges of platforms that execute routines via centralized data stores, and highlight the importance of enforcing security by design in emerging home automation platforms.

KEYWORDS

Smart Home, Routines, Privilege Escalation, Overprivilege

ACM Reference Format:
Kaushal Kafle, Kevin Moran, Sunil Manandhar, Adwait Nadkarni, Denys Poshyvanyk. 2019. A Study of Data Store-based Home Automation. In *Proceedings of ACM Conference on Data and Application Security and Privacy (CODASPY 2019).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Internet-connected, embedded computing objects known as *smart home products* have become extremely popular with consumers. The utility and practicality afforded by these devices has spurred tremendous market interest, with over 20 billion smart home products projected to be in use by 2020 [13]. The diversity of these products is staggering, ranging from small physical devices with embedded computers such as smart locks and light bulbs, to full fledged appliances such as refrigerators and HVAC systems. In the modern computing landscape, smart home devices are unique as they provide an often imperceptible bridge between the digital and physical worlds by connecting physical objects to digital services via the Internet, allowing the user to conveniently automate their home. However, because many of these products are tied to the user's security or privacy (*e.g.,* door locks, cameras), it is important to understand the attack surface of such devices and platforms, in order to build practical defenses without sacrificing utility.

As the market for smart home devices has continued to mature, a new software paradigm has emerged to facilitate smart home automation via the interactions between smart home devices and the apps that control them. These interactions may be expressed as *routines*, which are sequences of app and device actions that are executed upon one or more triggers, *i.e.,* an instance of the trigger-action paradigm in the smart home. Routines are becoming the foundational unit of home automation [8, 42, 50, 51], and as a result, it is natural to characterize existing platforms based on how routines are implemented.

If we categorize available platforms based on how routines are facilitated, we observe two broad categories: (1) API-based Smart Home Managers such as Yeti [55], Yonomi [56], IFTTT [18], and Stringify [48] that allow users to chain together a diverse set of devices using third-party APIs exposed by device vendors, and (2) smart home platforms such as Google's Works with Nest [34], Samsung SmartThings [45], and Philips Hue [40] that leverage *centralized data stores* to monitor and maintain the states of IoT devices. We term these platforms as Data Store-Based (DSB) Smart Home Platforms. In DSB platforms, complex routines are executed via reads/writes to state variables in a central data store.

This paper is motivated by a key observation that while routines are supported via centralized data stores in all DSB platforms, there are differences in the manner in which routines are created, observed, and managed by the user. That is, SmartThings encourages users to take full control of creating and managing routines involving third-party apps and devices via the SmartThings app. On the contrary, in Nest, users do not have a centralized perspective of routines at all, and instead, manage routines using third-party apps/devices. This key difference may imply unique security challenges for Nest. Similarly, being a much simpler platform within this category of DSB platforms, Hue represents another unique and interesting instance of the DSB platform paradigm. While prior work has explored the security of routines enabled by a smart home manager (*i.e.,* specifically, IFTTT recipes [49]), the permission enforcement and application security in the SmartThings platform [12], and the side-effects of SmartThings SmartApps [4], there is a notable gap in current research. Namely, prior studies do not evaluate the potential for *adversarial* misuse of routines, which are the essence of DSB platforms, and by extension, home automation.

Contributions: This paper performs a systematic security analysis of some of the less studied, but widely popular, data store-based smart home platforms, *i.e.,* Nest and Hue, helping to close the existing gap in prior research. In particular, we evaluate (1) the access control enforcement in the platforms themselves, (2) the robustness

CODASPY 2019, March 2019, Dallas, TX, USA
2019. ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

of other non-system enforcement (*e.g.,* product reviews in Nest), (3) the use and more importantly the *misuse* of routines via manipulation of the data store by low-integrity devices, [1] and finally, (4) the security of applications that integrate into these platforms. To our knowledge, this paper is the first to analyze this relatively new class of smart home platforms, in particular the Nest and Hue platforms, and to provide a holistic analysis of routines, their use, and potential for their misuse in DSB platforms. Moreover, this paper is the first to analyze the accuracy of app-defined permission prompts, which form one of the few sources of access control information for the user. Our novel findings ($\mathcal{F}_1 \rightarrow \mathcal{F}_{10}$), summarized as follows, demonstrate the unique security challenges faced by DSB platforms at the cost of seamless automation:

- **Misuse of routines** – The permission model in Nest is fine-grained and enforced according to specifications ($\mathcal{F}_1$), giving low-integrity third-party apps/devices (*e.g.,* a switch) little room for directly modifying the data store variables of high-integrity devices (*e.g.,* security cameras). However, the routines supported by Nest allow low-integrity devices/apps to indirectly modify the state of high-integrity devices, by modifying the shared variables that both high/low integrity devices rely on ($\mathcal{F}_4$).
- **Lack of systematic defenses** – Nest does not employ transitive access control enforcement to prevent indirect modification of security-sensitive data store variables; instead, it relies on a product review of application artifacts before allowing API access. We discover that the product review process is insufficient and may not prevent malicious exploitation of routines; *i.e.,* the review mandates that apps prompt the user before modifying certain variables, but does not validate *what* the prompt contain, allowing apps to deceive users into providing consent ($\mathcal{F}_5$). Moreover, permission descriptions provided by apps during authorization are also often incorrect or misleading ($\mathcal{F}_6$, $\mathcal{F}_9$), which demonstrates that malicious apps may easily find ways to gain more privilege than necessary ($\mathcal{F}_7$), circumventing both users and the Nest product review ($\mathcal{F}_8$).
- **Lateral privilege escalation** – We find that smart home apps, particularly those that connect to Nest and have permissions to access security-sensitive data store variables, have a significantly high rate of SSL vulnerabilities ($\mathcal{F}_{10}$). We combine these SSL flaws with the findings discussed previously (specifically $\mathcal{F}_4 \rightarrow \mathcal{F}_9$) and demonstrate a novel form of a *lateral* privilege escalation attack. That is, we compromise a low-integrity app that has access to the user's Nest smart home (*e.g.,* a TP Link Kasa switch), use the compromised app to change the state of the data store to trigger a security-sensitive routine, and indirectly change the state of a high-integrity Nest device (*e.g.,* the Nest security camera). This attack can be used to deceive the Nest Cam into determining that the user is home when they are actually away, and prevent it from monitoring the home in the user's absence.
- **Lack of bare minimum protections** – Unlike Nest, the access control enforcement of Hue is woefully inadequate. Third-party apps that have been added to a user's Hue platform may arbitrarily add other third-party apps without user consent, despite an existing policy that the user must consent by physically pressing a button ($\mathcal{F}_2$). Making matters worse, an app may *remove* other apps integrated with the platform by exploiting unprotected data store variables in Hue ($\mathcal{F}_3$). These vulnerabilities may allow an app with seemingly useful functionality (*i.e.,* a Trojan [21]) to install malicious add-ons in a manner invisible to the user, and replace the user's integrated apps with its malicious substitutes.

[1] In the context of our study, we define a device as high-integrity if it is advertised as security-critical by the device vendor (*e.g.,* Nest Cam) while those that are not security-critical are referred to as low-integrity (*e.g.,* Philips Hue lamp).

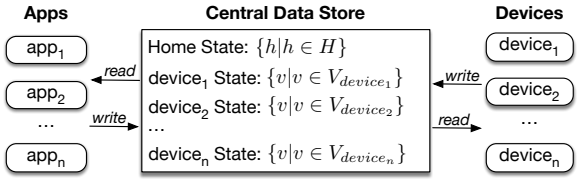

Figure 1: The general architecture of home automation platforms that leverage centralized data stores. Note that H is the universe of all home state variables, and V_{device_i} is the universe of all state variables specific to *device*$_i$

The rest of the paper is structured as follows: Section 2 describes the key attributes of DSB platforms, and provides background. Section 3 provides an overview of our analysis, and Sections 4, 5 and 6 describe our analysis of permission enforcement in Nest and Hue, security ramifications of routines, and security of smart home apps, respectively. Section 7 provides an end-to-end attack, and Section 8 describes the lessons learned. Section 9 describes the vendors' response to our findings. Section 10 describes the threats to validity. Section 11 describes the related work, and Section 12 concludes.

2 HOME AUTOMATION VIA CENTRALIZED DATA STORES

This section describes the general characteristics of data store-based platforms, *i.e.,* smart home platforms that use a *centralized data store* to facilitate routines. Following this general description, we provide background on two such platforms, namely (1) Google's "Works with Nest" [36] platform (henceforth called "Nest") and (2) the Philips Hue lighting system [39] (henceforth called "Hue"), which serve as the targets of our security analysis. While there are no official statistics on the market adoption of either Nest or Hue, the Android apps for both of the systems have over a million downloads on Google Play [16, 17], indicating significant adoption, and far-reaching security impact of our analysis.

2.1 General Characteristics

Figure 1 describes the general architecture of DSB platforms, consisting of three main components: *apps*, *devices*, and the *centralized data store*. These components generally communicate over the Internet. Additionally, a physical hub that facilitates local communication via protocols such as Zigbee or Z-wave may or may not be included in this setup (*e.g.,* the Hue Bridge); *i.e.,* in a general sense, routines are agnostic of the presence of the hub. Hence, we exclude the hub in Figure 1. Similarly, the apps may either be Web services hosted on the cloud, or mobile apps communicating via Web services. At this juncture, we generalize apps as third-party software interacting with the data store, and provide the specifics for individual platforms in later sections.

The centralized data store facilitates communication among apps and devices via state variables. The data store exposes two types of state variables: (1) *Home* state variables that reflect the general

state of the entire smart home (*e.g.,* if the user is at *home/away*, the *devices attached* to the home, the *postal code*), and (2) *Device-specific* state variables that reflect the attributes specific to particular devices (*e.g.,* if the Camera is *streaming*, the *target temperature* of the thermostat, the *battery health* of the smoke alarm).

Apps and devices communicate by reading from or writing to the state variables in the centralized data store. This model allows expressive communication, from simple state updates to indirect trigger-action routines. Consider this simple state update: the user may change the temperature of the thermostat from an app, which in turn *writes* the change to the *target temperature* variable in the data store. The thermostat device receives an update from the data store (*i.e., reads* the *target temperature* state variable), and changes its target temperature accordingly. Further, as stated previously, expressive routines may also be implemented using the data store. For instance, the thermostat may change to its "economy" mode when the home's state changes to *away*. That is, the thermostat's app may detect that the user has left the smart home (*e.g.,* using Geofencing), and *write* to the home state variable *away*. The thermostat may then *read* this change, and switch to its economy mode.

A salient characteristic of DSB platforms is that they lean towards seamless home automation, by automatically interacting with devices and executing complex routines via the centralized data store. However, even within platforms that follow this model (*e.g.,* Samsung SmartThings, Nest, and Hue), our preliminary investigation led to the following *key observations* that motivate a targeted analysis of the Nest and Hue platforms and their apps:

Key Observations: We observe that both Nest and SmartThings execute routines; however, there is a key difference in how routines are managed. SmartThings allows users to create and manage routines from the SmartThings app itself, thereby providing users with a general view of all the routines executing in the home [46]. In contrast, Nest routines are generally implemented as *decentralized* third-party integrations. Third-party products that facilitate routines provide the user with the ability to view and manage them. As a result, the Nest platform does not provide the user with a *centralized view* of the routines that are in place. Due to this lack of user control, Nest smart homes may face unique security risks and challenges, which motivates this security analysis. Similarly, we observe that the Philips Hue platform may be another interesting variant of DSB platforms. That is, Hue integrates *homogeneous* devices related to lighting such as lamps and bulbs, unlike Nest and SmartThings that integrate heterogeneous devices, and represents a drastically simpler (and hence unique) variant of home automation platforms that use centralized data stores. As a result, the analysis of Hue's attack surface has potential to draw attention to other similar, homogeneous platforms, which is especially important considering the fragmentation in the smart home product ecosystem [6]. To our knowledge, this paper is the first to analyze this relatively new class of smart home platforms, and specifically, Nest and Hue.

2.2 Nest Background

The *Works with Nest* platform integrates a heterogenous set of devices, including devices from Nest (*e.g.,* Nest thermostat, Nest Cam, Nest Protect) as well as from other brands (*e.g.,* Wemo and Kasa switches, Google Home, MyQ Chamberlain garage door opener) [36].

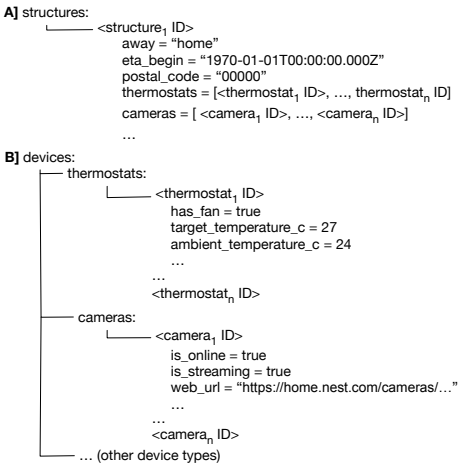

Figure 2: A simplified view of the centralized data store in Nest.

This section describes the key characteristics of Nest, *i.e.,* its data store, its access control model, and routines.

Data store composition: Figure 2 shows a simplified, conceptual view of the centralized data store in Nest. Note that the figure shows a small fraction of the true data store, *i.e.,* only enough to facilitate understanding. Nest implements the data store as a JSON-format document divided into two main top-level sections: *structures* and *devices*. A *structure* represents an entire smart home environment such as a user's home or office, and is defined by various state variables that are global across the smart home (*e.g., Away* to indicate the presence or absence of the user in the structure and the *postal_code* to indicate the home's physical location). The devices are subdivided into device types (*e.g.,* thermostats, cameras, smoke detectors), and there can be many devices of a certain type, as shown in Figure 2. Each device stores its state in variables that are relevant to its type; *e.g.,* a thermostat has state variables for *humidity*, and *target_temperature_c*, whereas a camera has the variables *is_online* and *is_streaming*. Aside from these type-specific variables, devices also have certain variables in common; *e.g.,* the alphanumeric *device ID*, the *structure ID* of the structure in which the device is installed, the device's user-assigned *name*, and *battery_health*.

Access Control in Nest.: Nest treats third-party apps, Web services, and devices that want to integrate with a Nest-based smart home as "products". Each Nest user account has a specific data store assigned to it and any product that requests access to the user's data store needs to be first authorized by the user using OAuth 2.0. Nest defines read or read/write permissions for each of the variables in the data store. Additionally, some variables *e.g.,* the list of all thermostats in the structure are always *read-only*. A product that wants to register with Nest must first declare the permissions that it needs (*e.g.,thermostat read*, *thermostat read/write*) in the Nest developer console. When connecting a product to Nest, during the OAuth authorization phase, the user is shown the permissions requested by the product. Once the user grants the permissions, a revocable access token is generated specific to the product, the set of permissions requested, and the particular smart home to which the product is connected. This token is used for subsequent interactions with the data store.

Accessing the Nest data store.: Devices and applications that are connected to a particular smart home (*i.e.,* the user's Nest account) can update data store variables to which they have access, and also subscribe to the changes to the state of the data store. Nest uses the REST approach for these update communications, as well as for apps/devices to modify the data store. The REST endpoints can be accessed through HTTPS by any registered Nest products.

Routines in Nest: The ability of connected devices to observe and write to state variables in the centralized data store facilitates trigger-action routines. However, in Nest, the user cannot create or view routines in a centralized interface (*i.e.,* unlike SmartThings). Instead, apps may provide routines as opt-in features. For example, the Nest smoke alarm's *smoke_alarm_state* variable has three possible values, "ok", "warning", and "emergency". When this variable is changed to "warning", other smart home products (e.g., Somfy Protect [28]) can be configured to trigger and warn the user. Note that in the *Home/Away assist* section of the Nest app settings, users can view a summary of how certain variables (i.e., home or away) affect their Nest-manufactured devices; however, there is no way for users to observe the triggers/apps that change the state of the *away* variable *simultaneously* with the resultant actions, preventing them from fully understanding how routines execute in their home.

2.3 Hue Background

Unlike Nest, which is a platform for heterogeneous devices, Philips Hue deals exclusively with lighting devices such as lamps and bulbs. As a result, the centralized data store of Philips Hue supports much simpler routines. Hue implements its data store as a JSON document with sections related to (1) physical lighting devices, (2) semantic groups of these devices, and (3) global config variables (such as *whitelisted apps* and the *linkbutton*). To connect a third-party management app to a user's existing Hue system, the app identifies a Hue bridge connected to the local network, and requires the user to press a physical button on the bridge. Once this action is completed by the user, the app receives a *username* token that is stored in the *whitelisted* section of the Hue data store. Whitelisted apps can then read and modify data store variables as dictated by Hue's access control policy, which grants all authorized apps the same access regardless of their purported functionality. Our online appendix provides additional details regarding the Hue platform [1].

3 ANALYSIS OVERVIEW

This paper analyzes the security of home automation platforms that rely on centralized data stores (*i.e.,* DSB platforms). Third-party apps are the security principals on such platforms, as they are assigned specific permissions to interact with the integrated devices. That is, as described in Section 2, DSB platforms consist of (1) *third-party apps* that interact with the smart home (*i.e.,* centralized data store and devices) by acquiring (2) *platform permissions*, and execute a complex set of such interactions as (3) *trigger-action routines*. Our analysis methodology takes these three aspects into consideration, starting with platform permissions, as follows:

A. Analysis of Platform Permissions (Section 4): We analyze the *enforcement* of platform permissions/access control to discover inconsistencies. For this analysis, we automatically build permission maps, and semi-automatically analyze them.

B. Analysis of Routines (Section 5): While analyzing permission enforcement gives us an idea of what individual devices can accomplish with a certain set of permissions, we perform an experimental analysis with real devices to identify the interdependencies among devices and apps through the shared data model, and the ramifications of such interdependencies on the user's security and privacy. Additionally, we notice that Nest does not enforce transitive access control policies to prevent dangerous side-effects of routines, but instead employs a product review process as a defense mechanism. We analyze the effectiveness of this review process using the permission prompts used by existing apps as evidence.

C. Analysis of Third-party Apps (Section 6): We analyze the permission descriptions presented by mobile apps compatible with Nest to identify over-privileged apps, or apps whose permission descriptions are inconsistent with the permission requested. We then analyze the apps for signs of SSL misuse, in order to exploit applications that possess critical permissions, which can be leveraged to indirectly exploit security critical devices in the smart home.

We combine the findings from these three analyses to demonstrate an instance of a *lateral privilege escalation* attack in a smart home (Section 7). That is, we demonstrate how an attacker can compromise a low-integrity device/app integrated into a smart home (*e.g.,* a light bulb), and use routines to perform protected operations on a high-integrity product (e.g., a security camera).

4 EVALUATING PERMISSION ENFORCEMENT

The centralized data store described in Section 2 may contain variables whose secrecy or integrity is crucial; *e.g.,* unprotected write access to the *web_url* field of the camera may allow a malicious app to launch a phishing attack, by replacing the URL in the field with an attacker-controlled one. To understand if appropriate barriers are in place to protect such sensitive variables, we perform an analysis of the permission enforcement in Nest and Hue.

Our approach is to generate and analyze the *permission map* for each platform, *i.e.,* the variables that can be accessed with each permission, and inversely, the permissions needed to access each variable of the data store. Note that while this information should ideally be available in the platform documentation, prior analysis of similar systems has demonstrated that the documentation may not always be complete or correct in this regard [10, 12].

4.1 Generating Permission Maps

We generate the permission map using automated testing as in prior work on Android [10]. We use two separate approaches for Nest and Hue, owing to their disparate access control models.

Approach for Nest: We first created a simulated home environment using the Nest Home Simulator [35], and linked our Nest user account to this simulated smart home. We then created our test Android app, and connected our test app to the simulated home (*i.e.,* our Nest user account) as described in Section 2.2. Note that the simulated smart home is virtually identical to an end-user's setup, such that real devices may be added to it. Using the simulator allows us to investigate the data store information of Nest devices (*e.g.,* the Smoke/CO detector) that we may not have installed.

In order to generate a complete view of the data store, we granted our test app all of the 15 permissions in Nest (*e.g.,Away read/write*, *Thermostat read*), and read all accompanying information. To build

the permission map for Nest's 15 permissions, we created 15 apps, such that each app requested a single unique permission, and registered these apps to our developer account in the Nest developer console. Note that we do not test the effect of permission combinations, as our goal is to test the enforcement of individual permissions, and Nest's simple authorization logic simply provides an app with a union of the privileges of the individual permissions.

We then connected each of the 15 apps to our Nest user account using the procedure described in Section 2.2. We programmed each app to attempt to read and write each variable of the data store (*i.e.,* the previously derived *complete view*). We recorded the outcome of each access, *i.e.,* if it was successful, or an access control denial. In the cases where we experienced non-security errors writing to data store variables (*e.g.,* writing data with an incorrect type), we revised our apps and repeated the test. The outcome of this process was a permission map, *i.e.,* the mapping of each permission to the data store variables that it can read and/or write.

Approach for Hue: We followed the procedure for Hue described in Section 2.3 to get a unique token that registers our single test app with the data store of our Hue bridge. In Hue, all the variables of the data store are "readable" (*i.e.,* we verified that all the variables described in the developer documentation [40] can be read by third-party apps). Therefore, to build the permission map, we first extracted the contents of the entire data store. Then, for each subsection within the data store, our app made repeated write requests, *i.e.,* PUT calls with the payload consisting of a dummy value based on the variable type (*i.e.,* String, Boolean and Integer). All the variables that were successfully written to using this method were assigned as "writable" variables. Similarly, our app made repeated DELETE calls to the API and the variables that were successfully deleted were assigned as "writable" variables. This generated permission map applies to all third-party apps connected to Hue, since the platform provides equal privilege to all third-party apps.

4.2 Analyzing Permission Maps

The objective behind obtaining the permission map is to understand the potential for application overprivilege, by analyzing the granularity as well as the correctness of the enforcement. We analyze the permission map to identify instances of (1) *coarse-grained permissions*, *i.e.,* permissions that give the third-party app access to a set of security-sensitive resources that must ideally be protected under separate permissions, and (2) *incorrect enforcement*, *i.e.,* when an app has access to more resources (*i.e.,* state variables) than it should have given its permission set, as per the documentation; *e.g.,* apps on SmartThings may lock/unlock the door lock without the explicit permission required to do so [12].

To perform this analysis, we first identified data store variables that may be security or privacy-sensitive. This identification was performed using an open-coding methodology by one author, and separately verified by another author, for each platform. We then performed further analysis by separately considering each such variable, and the permission(s) that allow access to it. A major consideration in our analysis is the security impact of an adversary being allowed read or read/write access to a particular resource. Moreover, our evaluation of the impact of the access control enforcement was contextualized to the platform under inspection. That is, when evaluating Nest, we took into consideration the semantic meaning and purpose of certain permissions in terms of the data store variables, as described in the documentation (*e.g.,* that the *Away read/write* permission should be required to write to the *away* variable [30]). For Hue, we only considered the security-impact of an adversary accessing data store variables. Our rationale is that the Hue platform defines the same static policy (*i.e.,* same permissions) for all third-party apps, and hence, its permission map can be simply said to consist of just one permission that provides access to a fixed set of data store variables. As a result, we judge application over-privilege in Hue by considering the impact of an adversarial third-party app reading from or writing to each of the security-sensitive variables identified in Hue's permission map.

The creation of the permission maps for both Nest and Hue requires the application of well-studied automated testing techniques, and as such, can be replicated for similar platforms, with minor changes to input data (*e.g.,* the permissions to test for). We will release our code and data to developers and platform vendors.

4.3 Permission Enforcement Findings ($\mathcal{F}_1 \rightarrow \mathcal{F}_3$)

Finding 1: The permission enforcement in Nest is fine-grained and correctly enforced, *i.e.,* as per the specification ($\mathcal{F}_1$). We observe that the Nest permission map is significantly more fine-grained, and permissions are correctly enforced, relative to the observations of prior research in similar platforms (*e.g.,* the analysis of SmartThings [12]). Some highly sensitive variables are always read-only (*e.g.,* the *web_url* where the camera feed is posted), and there are separate read and read/write permissions to access sensitive variables. Variables that control the state of the entire smart home are protected by dedicated permissions that control write privilege; *e.g.,* the *away* variable can only be written to using the *Away read/write* permission, the *ETA* variable has separate permissions for apps to read and write to it (*i.e., ETA read* and *ETA write*), and the Nest Cam can only be turned on/off via the *is_streaming* variable, using the *Camera + Images read/write* permission that controls write access to it. Moreover, since many apps need to respond to the *away* variable (*i.e.,* react when the user is home/away), device-specific read permissions (*e.g., Thermostat read*, *Smoke + CO read*) also allow apps to *read* the *away* variable, eliminating the need for apps to ask for higher-privileged *Away read* permission. The separate read and read/write permissions are correctly enforced, *i.e.,our generated permission map provides the same access as is defined in the Nest permission documentation [30].* This is in contrast with findings of similar analyses of permission models in the past (*e.g.,* the Android permission model [10], SmartThings [12]), and demonstrates that the Nest platform has incorporated lessons from prior work in permission enforcement.

Finding 2: In Hue, the access control policy allows apps to bypass the user's explicit consent ($\mathcal{F}_2$). We discovered two data store variables that were not write-protected, and which have a significant part to play in controlling access to the data store and the user's smart home. First, any third-party app can write to the *linkbutton* flag. Recall from Section 2.3 that the user has to press the physical button on the Hue bridge device to authorize an app's addition to the bridge. The physical button press changes the *linkbutton* value to "true", and allows the app to be added to the *whitelist* of

allowed third-party apps. However, we discovered that once installed, an app can toggle the *linkbutton* variable at will, *enabling third-party apps to add other third-party apps to the smart home without the user's consent*. This exploitable access control vulnerability can allow an app with seemingly useful functionality to install malicious add-ons by bypassing the user altogether. In our tests, we verified this attack with apps that were connected to the local network. This condition is feasible as a malicious app that needs to be added without the user's consent may not even have to pretend to work with Hue; all it needs is to be connected to the local network (*i.e.,* a game on the mobile device from one of the people present in the smart home). Note that it is also possible to remotely perform this attack, which we discuss in Section 10.

Finding 3. In Hue, third-party apps can directly modify the list of added apps, adding and revoking access without user consent ($\mathcal{F}_3$). Hue stores the authorization tokens of apps connected to the particular smart home in a *whitelist* on the Hue Bridge device. While analyzing the permission map, we discovered that not only could our third-party test app read from this list, it could also directly delete tokens from it. We experimentally confirmed this finding again, by removing *Alexa* and *Google Home* from the smart home, without the user's consent. An adversary could easily combine this vulnerability with ($\mathcal{F}_2$), to remove legitimate apps added by the user, add adversary-controlled apps (*i.e.,* by keeping the *linkbutton* "true"), all without the user's consent. More importantly, users do not get alerts when such changes are made (*i.e.,* since it is assumed that the enforcement will correctly acquire user consent). Hence, unless the user actually checks the list of integrated apps using the Hue Web app, the user would not notice these changes.

While the Nest permission model is robust in its mapping of data store variables and permissions required to access them, Section 5 demonstrates how fields disallowed by permissions may be indirectly modified via strategic misuse of routines, and describes Nest's product review guidelines to prevent the same [32]. Section 6 describes how badly written and overprivileged apps escape these review guidelines, and motivate a technical solution.

5 EVALUATING SMART HOME ROUTINES

Prior work has demonstrated that in platforms that favor application interoperability but lack transitive access control enforcement, problems such as confused deputy and application collusion may persist [5, 11, 23, 24]. Smart homes that facilitate routines are no different, but the exploitability and impact of routines on smart homes is unknown, which motivates this aspect of our study.

Recall that routines are trigger-action programs that are either triggered by a change in some variable of the data store, or whose action modifies certain variables of the data store. While both Nest and Hue share this characteristic, routines in Hue are fairly limited in scope, and their exploitation is bound to only affect the lighting of the smart home. As a result, while we provide confirmed examples of Hue routines in Section 2.3, the security evaluation described in this section is focused on the heterogeneous Nest platform that facilitates more diverse and expressive routines.

5.1 Methodology for the Analysis of Routines

While using the simulator as described in Section 4 allows us to understand what routines are *possible* on the platform, *i.e.,* what variables might be manipulated, and what Nest devices (e.g., the Nest Cam, Nest Thermostat) are affected as a result, we performed additional experiments with real apps and devices to study existing routines in the wild. For this experiment, we extended the smart home setup previously discussed in Section 4 with real devices.

We started by collecting a list of devices that integrate with Nest from the *Works with Nest* website [36]. Using this initial list and information from the website, we purchased a set of 7 devices that possessed a set of characteristics relevant to this study, *i.e.,* devices that (1) take part in routines (*i.e.,* as advertised on the website), (2) are important for the user's security or privacy, and (3) are widely-known/popular with a large user base (*i.e.,* determined by the number of installs of the mobile client on Google Play). We obtained a final list of devices (7 real and 2 simulated) to our Nest smart home, namely, the Nest Cam (*i.e.,* a security camera), Hue light bulb, Belkin Wemo switch, the MyQ Chamberlain garage door opener, TP Link Kasa Smart Plug, Google Home, Alexa, Nest Thermostat (simulated), and the Nest Protect Smoke & CO Alarm (simulated). Some devices that may be important for security did not participate in routines at the time of the study, and hence were excluded from our final device list.

We connected these devices to our Nest smart home using the Android apps provided by device vendors, and connected a small set of smart home managers (*e.g.,* Yeti [55] and Yonomi [56]) to our Nest smart home as well. For each device, we set up and executed each individual routine as described on the Works with Nest as well as the device vendor's website, and observed the effects on the rest of the smart home (especially, security-sensitive devices). Also, we manipulated data store variables from our test app, and observed the effects on previously configured routines and devices.

5.2 Smart Home Routine Findings ($\mathcal{F}_4 \rightarrow \mathcal{F}_5$)

Finding 4. Third-party apps that do not have the permission to turn on/off the Nest Cam directly, can do so by modifying the *away* variable ($\mathcal{F}_4$). The Nest Cam is a home monitoring device, and important for the users' security. The *is_streaming* variable of the Nest Cam controls whether the camera is on (*i.e.,* streaming) or off, and can only be written to by an app with the permission *Camera r/w*. The Nest Cam provides a routine as a feature, which allows the camera to be automatically switched on when the user leaves the home (*i.e.,* when the *away* variable of the smart home is set to "away"), and switched off when the user returns (*i.e.,* when *away* is set to "home"). Leveraging this routine, third-party apps such as the Belkin Wemo switch can manipulate the *away* field, and indirectly affect the Nest Cam, without having explicit permission to do so. We tested this ability with our test app (see Section 4) as well, which could indirectly switch the camera *on* and *off* at will. This problem has serious consequences; *e.g.,* a malicious test app with the *away r/w* permission may set the variable to "home" when the user is away to prevent the camera from recording a burglary. The key problem here is that a *low-integrity device/app can trigger a change in a high-integrity device indirectly, i.e.,* by modifying a variable it relies on, which is an instance of the well-known information flow *integrity* problem. Moreover, this is not the only instance of a high-integrity routine that relies on *away*; *e.g.,* the Nest x Yale Lock can lock automatically when the home changes to

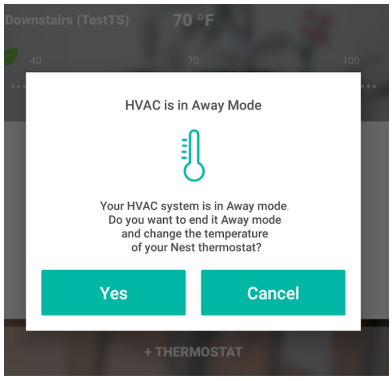

Figure 3: The *Keen Home* app asks the user to modify the thermostat's mode, but in reality, this action leads to the *entire* smart home being set to "home" mode, which affects a number of other devices.

away mode [27], leakSMART reads the *away* state of the home and can notify the user's emergency contact when a leak occurs [26].

Nest has a basic defense to prevent such issues: application design policies that apply to apps with more than 50 users [32]. App developers are required to submit their app for a product review to the Nest team once the app reaches 50 users, and a violation of the rather strict and detailed review guidelines can result in the app being rejected from using the Nest API. One of the review policies (*i.e.,* specifically policy 5.8) states that *"Products that modify Home-/Away state automatically without user confirmation or direct user action will be rejected." [32].* Nest users may be vulnerable in spite of this defense, for two reasons. First, as attacking a smart home is an attack on a user's personal space, it is feasible to assume that most attacks that exploit routines will be targeted (*e.g.,* to perform burglaries). Assuming that the adversary can use social engineering to get the user to connect a malicious app to their Nest setup, *a targeted attack on a specific user will succeed in spite of the policy*, as the app would be developed solely for the targeted user and hence will have <50 users, and be exempt from the Nest product review. Second, it is unclear how apps are checked against this policy; our next finding demonstrates a significant omission in Nest's review.

Finding 5. Nest's product review policies dictate that the apps must prompt users before modifying *away* , but there is no official constraint on *what* the prompt may display ($\mathcal{F}_5$). Consider the example in Figure 3, which shows one such prompt by the *Keen Home* app [25] *when the user tries to change the temperature of the thermostat.* That is, when the user tries to change the temperature of the thermostat while the *away* variable is set to "away", the app requires us to change it to "home" before the thermostat temperature can be changed. This condition is entirely unnecessary to change the temperature. More importantly, it presents the prompt to the user in a way that states that the home/away modes are specific to the HVAC alone. This is in contrast to the actual functionality of these modes, in which a change to the *away* variable affects the *entire* smart home; *i.e.,* we confirmed that the Nest Cam gets turned off as well once we agree to the prompt. It is important to note that the *Keen Home* app has passed the Nest product review, as it has well over 50 users (1K+ downloads on Google Play [15]). Therefore, this case demonstrates that the Nest product review does not consider the contents of the prompt, and a malicious app may easily misinform the user and make them trigger the *away* variable to the app's advantage. Finally, in Section 6.1 we demonstrate that this problem of misinforming the user is not just limited to *runtime in-app prompts* described in this section, but extends to application-defined *install-time permission descriptions* ($\mathcal{F}_6 \rightarrow \mathcal{F}_9$).

6 SECURITY ANALYSIS OF NEST APPS

In this Section, we investigate the privileges of apps developed to be integrated with Nest. Unlike prior work [12], we not only report the permissions requested by apps, but also analyze the information prompts displayed to the user when requesting the permission. Additionally, we analyze the rate of SSL misuse by both general smart home management apps as well as apps integrated with Nest. For this section, we do not consider the Hue platform as it has a limited ecosystem of apps as compared to Nest. We derived two datasets to perform the analyses that we describe in this section, the $\text{Apps}_{general}$ dataset, which contains 650 smart home management apps extracted from Google Play, and the Apps_{nest} dataset, which includes 39 apps that integrate into the Nest platform. Our online appendix [1] details our dataset collection methodologies.

6.1 Application Permission Descriptions

On Nest, developers provide permission descriptions that explain how an app uses a permission while registering their apps in the Nest developer console. These developer-provided descriptions are the *only* direct source of information available to the user to understand why an app requires a particular permission, *i.e.,* Nest itself only provides a short and generic permission "title" phrase that is displayed to the user along with the developer-defined description (*e.g.,* for *Thermostat read*, the Nest phrase is "See the temperature and settings on your thermostat(s)"). Owing to their significant role in the user's understanding of the permission requirements, we analyze the *correctness* of such developer-defined descriptions relative to the permissions requested.

6.1.1 Analysis Methodology. As described in Section 2, upon registering permissions at the developer console, developers are granted an OAuth URL that they can direct the user to for obtaining an access token. As a result, permissions are not encoded in the client mobile app or Web app (*i.e.,* unlike Android), which makes the task of extracting permissions difficult. However, we observe that the permissions that an app asks for are *always* displayed to the user for approval (*i.e.,* when first connecting an app to their Nest smart home using OAuth). We leverage this observation to obtain permissions dynamically, *i.e.,* by executing apps to the point of integrating them with our Nest smart home, and recording the permission prompt displayed for the user's approval. The procedure is the same for mobile as well as Web apps.

6.1.2 Nest App Findings ($\mathcal{F}_6 \rightarrow \mathcal{F}_9$). The two permissions that dominate the permission count are *Away read/write* and *Thermostat read/write*, requested by 20 and 24 apps respectively, from the Apps_{nest} dataset. Our online appendix [1] provides the permission count for all other permissions. Our findings are as follows:

Finding 6. A significant number of apps provide incorrect permission descriptions, which may misinform users ($\mathcal{F}_6$). As shown in Table 1, we found a total of 15 permission description violations in 13/39 apps from the Apps_{nest} dataset. We classify these

Table 1: Permission description violations discovered in Works with Nest apps

Application	Incorrect Permission Description
VC1: Requesting Read/Write instead of Read	
1. Home alerts	"**thermostat read/write:** Allows Home alerts to notify you when the Nest temperature exceeds your threshold(s)"
2. Home alerts	"**away read/write:** Allows Home Alerts to notify you when someone is in your home while in away-mode"
3. MyQ Chamberlain	"**thermostat read/write:** Allows Chamberlain to display your Nest Thermostat temperature in the MyQ app"
4. leakSMART	"**thermostat read/write:** Allows leakSMART to show Nest Thermostat room temperature and humidity. New HVAC sensor mode will notify you to shut off your thermostat if a leak is detected in your HVAC system."
5. Simplehuman Mirror	"**Camera+Images read/write:** Allow your simplehuman sensor mirror pro to capture and recreate the light your Nest Cam sees"
6. Iris by Lowe's	"**structure read/write:** View your Nest Structure names so Iris can help you pair your Nest Structures to the correct Iris Places"
7. Heatworks model 1	"**away read/write:** Allows the Heatworks MODEL 1 to be placed into vacation mode to save on power consumption while you're away"
8. Feather Controller	"**Camera+Images read/write:** Allows Feather to show you your camera and activity images. Additionally, Feather will allow you to request a snapshot."
9. Heatworks model 1	"**thermostat r/w:** Allows your Heatworks MODEL 1 water heater to go into vacation mode when your home is set to away"
VC2: Describing *Away* as a property of the thermostat alone, rather than something that affects the entire smart home	
10. Gideon	"**away read/write:** Allows Gideon to read and update the Away state of your thermostat"
11. Muzzley	"**away read/write:** Allows Muzzley to read and update the Away state of your thermostat"
12. Keen home smart vent	"**away read/write:** Allows Smart vent to read the state of your Thermostat and change the state from Away to Home"
VC3: Both VC1 and VC2	
13. WeMo	"**away read/write:** Allows your WeMo products to turn off when your Nest Thermostat is set to Away and on when set to Home."
14. IFTTT thermostat service	"**thermostat read/write:** Now you can turn on Nest Thermostat Applets that monitor when you're home, away and when the temperature changes."
VC4: Descriptions that do not relate to the permission	
15. IFTTT thermostat service	"**away read/write**: Now you can set your temperature or turn on the fan with Nest Thermostat Applets on IFTTT"
16. Life360	"**away read/write:** We need this permission to automatically turn on/off your nest system"

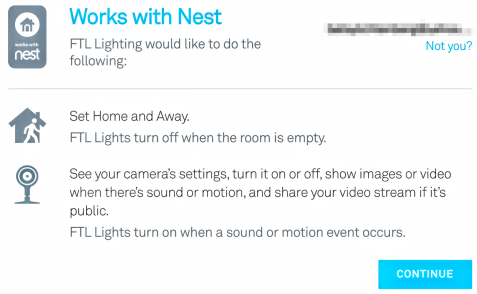

Figure 4: An example from the Nest documentation on OAuth authorization [31] that displays a permission description violation (specifically, VC1) for the *Away r/w* and *Camera + images r/w* permissions. The developer's permission description indicates that the FTL Lights only need to read data store variables, in both cases.

incorrect descriptions into 5 violation categories (*i.e.,* VC1 → VC4), based on the specific manner in which they misinform the user, such as requesting more privileges than required for the described need (*e.g.,* read/write permissions when only reading is required), or misrepresenting the effect of the use of the permission (*e.g.,* stating *Away* as affecting only the thermostat). That is, *over 33.33% of the apps we could integrate have violating permission descriptions.*

Finding 7. In most cases of violations, apps request read/write permissions instead of read ($\mathcal{F}_7$). In 9 cases, apps request the more privileged *read/write* version of the permission, when they should have clearly requested the *read* version, as per their permission description (*i.e.,* VC1 in Table 1). For example, consider the "MyQ Chamberlain" app (Table 1, entry 3), which asks for the *thermostat read/write* permission, but whose description only suggests the need for the *thermostat read* permission, *i.e.,* "Allows Chamberlain to display your Nest Thermostat temperature in the MyQ app". More importantly, a majority of the violations of this kind occur for the *Away read/write* and *Camera+Images read/write* permissions, which may have serious consequences if these over-privileged apps are compromised, *i.e.,* as *Away read/write* regulates control over indicating whether a user is at home or out of the house, and *Camera+Images read/write* may allow apps to turn off the Nest cam via the *is_streaming* variable. These violations exist in spite of Nest guidelines that mention the following as a *Key Point*: *"Choose 'read' permissions when your product needs to check status. Choose 'read/write' permissions to get status checks and to write data values."* [30]. Finally, we found that the *Nest documentation may itself have incorrect instructions, e.g.,* the Nest's documentation on OAuth 2.0 authentication [31] shows an example permission prompt that incorrectly requests the *Away read/write* permission while only needing read access, *i.e.,* with the description "FTL Lights turn off when the room is empty", as shown in the Figure 4.

Finding 8. The Nest product review is insufficient when it comes to reviewing the correctness of permission descriptions and requests by apps ($\mathcal{F}_8$). The Nest product review suggests the following two rules, violating which may cause apps to be rejected: (1) *"3.3. Products with names, descriptions, or permissions not relevant to the functionality of the product"*, and (2) *"3.5. Products that have permissions that don't match the functionality offered by the products"* [32]. Our findings demonstrate that the 16 violations discovered violate either one or both of these rules (*e.g.,* by requesting read/write permissions, when the app only requires read). The fact that the apps are still available suggests that the Nest product review may not be rigorously enforced, and as a result, may be insufficient in protecting the attacks discovered in Section 5.

Finding 9. Apps often incorrectly describe the *Away* field as a local field of the Nest thermostat, which is misleading ($\mathcal{F}_9$). One example of this kind (VC2 in Table 1) is the *Keen Home* app described in Section 5 (Table 1. entry 12), which states that it needs *Away read/write* in order to *"Allow Smart vent to read the state of your Thermostat and change the state from Away to Home"*. As a result, *Keen Home* misrepresents the effect and significance of writing to the *Away* field, by making it seem like *Away* is a variable of the thermostat, instead of a field that affects numerous devices in the entire smart home. Gideon and Muzzley (entries 10 and 11 in Table 1) present a similar anomaly. Our hypothesis is that

such violations occur because Nest originally started as a smart thermostat that gradually evolved into a smart home platform. Finally, in addition to misleading descriptions classified as VC1 and VC2, we discovered apps whose permission descriptions did not relate to the permissions requested at all (VC4), and apps whose descriptions satisfied both VC1 and VC2 (*i.e.,* VC3 in Table 1).

The accuracy of permission descriptions is important, as the user has no other source of information upon which to base their decision to trust an app. Nest recognizes this, and hence, makes permissions and descriptions a part of its product review. The discovery of inaccurate descriptions not only demonstrates that apps may be overprivileged, but also that Nest's design review process is incomplete, as it puts all its importance on getting the user's consent via permission prompts (*e.g.,* in Findings 5→9), but not on what information is actually shown.

6.2 Application SSL Use

The previous section demonstrated that smart home apps may be overprivileged in spite of a dedicated product review. An adversary may be able to compromise the smart home by exploiting vulnerabilities in such overprivileged apps. As a result, we decided to empirically derive an estimate of how vulnerable smart home apps are, in terms of their use of SSL APIs, which form an important portion of the apps' attack surface.

We used two datasets for this experiment, *i.e.,* the $\text{Apps}_{general}$ dataset consisting of 650 generic smart home (Android) apps crawled from Google Play, and an extended version of the Apps_{nest} dataset, *i.e.,* the $\text{Apps}_{nestExt}$ dataset, which consists of 111 Android apps built for Works with Nest devices (*i.e.,* including the ones for which we do not possess devices). We analyzed each app from both the datasets using MalloDroid [9], to discover common SSL flaws.

Finding 10. A significant percentage of general smart home management apps, as well as apps that connect to Nest have serious SSL vulnerabilities ($\mathcal{F}_{10}$). 20.61% (*i.e.,* 134/650) of the smart home apps from the $\text{Apps}_{general}$ dataset, and 19.82% (*i.e.,* 22/111) apps from the $\text{Apps}_{nestExt}$ dataset, have at least one SSL violation as flagged by MalloDroid. Specifically, in the $\text{Apps}_{nestExt}$ dataset, the most common cause of an SSL vulnerability is a broken *TrustManager* that accepts *all certificates* (*i.e.,* 20 violations), followed by a broken *HostNameVerifier* that does not verify the hostname of a valid certificate (*i.e.,* 11 violations). What is particularly worrisome is that apps such as *MyQ Chamberlain* and *Wemo* have multiple SSL vulnerabilities as well as the *Away read/write* permission, which makes their compromise especially dangerous. Prior work has demonstrated that such vulnerabilities can be dynamically exploited (*e.g.,* via a Man-in-the-Middle proxy) [9, 43], and we use similar approaches to demonstrate an end-to-end attack on the Nest security camera, using one of the SSL vulnerabilities discovered from this analysis.

7 LATERAL PRIVILEGE ESCALATION

While our findings from the previous sections are individually significant, we demonstrate that they can be combined to form an instance of a lateral privilege escalation attack [41], in the context of smart homes. That is, we demonstrate how *an adversary can compromise one product (device/app) integrated into a smart home, and escalate privileges to perform protected operations on another product, leveraging routines configured via the centralized data store.*

This attack is interesting in the context of smart homes, because of two core assumptions that it relies on (1) low-integrity (or non-security) smart home products may be easier to directly compromise than high-integrity devices such as the Nest Cam (*i.e.,* none of the SSL vulnerabilities in $\mathcal{F}_{10}$ were in security-sensitive apps), and (2) while low-integrity devices may not be able to directly modify the state of high-integrity devices ($\mathcal{F}_1$), they may be able to indirectly do so via *automated routines* triggered by global smart home variables ($\mathcal{F}_4$). (3) Moreover, since the low-integrity device is not being intentionally malicious, but is compromised, the product review process would not be useful, even if it was effective (which it is not, as demonstrated by $\mathcal{F}_5$→$\mathcal{F}_9$). This last point distinguishes a lateral privilege escalation from actions of malicious apps that trigger routines (*e.g.,* the "fake alarm attack" discussed in prior work [12]). These conditions make lateral privilege escalation particularly interesting in the context of smart home platforms, and especially, DSB platforms such as SmartThings and Nest.

Attack Scenario and Threat Model: We consider a common man-in-the-middle (MiTM) scenario, similar to the SSL-exploitation scenarios that motivate prior work [9, 43]. Consider Alice, a smart home user who has configured a security camera to record when she is away (*i.e.,* using the *away* variable in the centralized data store). Bob is an acquaintance (*e.g.,* a disgruntled employee or an ex-boyfriend) whose motive is to steal a valuable from Alice's house without being recorded by the camera. We assume that Bob also knows that Alice uses a smart switch in her home, and controls it via its app, which is integrated with Alice's smart home. Bob follows Alice, and connects to the same public network as her (*e.g.,* a coffee shop, common workplace), sniffs the access token sent by the switch's app to its server using a known SSL vulnerability in the app, and then uses the token to directly control the *away* variable. Setting the *away* to "home" confuses the security camera into thinking that Alice is at home, and it stops recording. Bob can now burglarize the house without being recorded.

The Attack: The example scenario described previously can be executed on a Nest smart home, using the Nest Cam and the TP Link Kasa switch (and the accompanying Kasa app). We compromise the SSL connection of Kasa app, which was found to contain a broken SSL TrustManager in our analysis described in Section 6. We choose Kasa app as it requests the sensitive *Away read/write* permission, and has a sizable user base (1M+ downloads on Google Play [14]). It is interesting to note that the Kasa app has also passed the Nest product review process and is advertised on the Works with Nest website [33], but can still be leveraged to perform an attack. We use *bettercap* [2] as a MiTM proxy to intercept and modify unencrypted data. Additionally, as described in the attack scenario, we assume that (1) the victim's Nest smart home has the Nest Cam and the Kasa switch installed, (2) the popular routine which triggers the Nest Cam to stop recording when the user is home is enabled, and (3) the user connects her smartphone to a network to which the attacker has access (e.g., coffee shop, office), which is a common assumption when exploiting SSL-misuse [9, 43].

The attack proceeds as follows: (1) The user utilizes the Kasa app to control the switch, while the user's mobile device is connected

Listing 1: The Kasa app's unencrypted GET request.

```
{"data":{"uri":"com.tplinkra.iot.authentication.impl.
     RetrieveAccountSettingRequest"},
   "iotContext":
       {"userContext":{"accountToken":"<anonymized
            alphanumeric token>",
       "app":{"appType":"Kasa_Android"},
       "email":"<anonymized>",
       "terminalId":"<anonymized>"}}, ...
```

to public network. (2) The attacker uses a MiTM proxy to intercept Kasa app's attempt to contact its own server, and supplies the attacker's certificate to the app during the SSL handshake, which is accepted by the Kasa app due to the faulty TrustManager. (3) The Kasa app then sends an authorization token (see Listing 1) to the MiTM proxy (*i.e.,* assuming it is the authenticated server), which is stolen by the attacker. This token authorizes a particular client app to send commands to the TP Link server. (4) Using the stolen token, the attacker instructs the TP Link server to set the smart home's *away* variable to the value "home", while the user is actually "away". This action is possible as the TP Link server (*i.e.,* Web app) has the *-Away read/write* permission for the user's Nest smart home. (5) This triggers the routine in the Nest Cam, which stops recording.

In sum, the attacker compromises a security-insensitive (i.e., low-integrity) product in the system, and uses it along with a routine to escalate privileges, *i.e.,* to modify the state of a security-sensitive (i.e., high-integrity) product. It should be noted that while this is one verified instance of a lateral privilege escalation attack on DSB smart home platforms, given the broad attack surface indicated by our findings, it is likely that similar undiscovered attacks exist.

8 LESSONS

Our findings ($\mathcal{F}_1$)→($\mathcal{F}_{10}$) demonstrate numerous gaps in the security of smart home platforms that implement routines using centralized data stores. Moreover, while many of the findings may apply to platforms such as SmartThings as well, their implications are more serious on Nest, as the user does not have a centralized perspective of the routines programmed into the smart home. We now distill the core lessons from our findings, which motivate significant changes in modern platforms such as Nest.

Lesson 1 : *Seamless automation must be accompanied by strong integrity guarantees.* It is important to note that the attack described in Section 7 may not be addressed by fixing the problem of over-privilege or via product reviews, since none of the components of the attack are overprivileged (*i.e.,* including TP Link Kasa), and our findings demonstrate that the Nest product review is insufficient ($\mathcal{F}_5$→$\mathcal{F}_9$). The attack was enabled due to the integrity-agnostic execution of routines in Nest ($\mathcal{F}_4$). To mitigate such attacks, platforms such as Nest need information flow control (IFC) enforcement that ensures strong integrity guarantees [3], and future work may explore the complex challenges of (1) specifying integrity labels for a diverse set of user devices and (2) enforcing integrity constraints without sacrificing automation. Moreover, as third-party devices are integrated into the data store, future work may also explore the use of *decentralized* information flow control (DIFC) to allow devices to manage the integrity of their own objects [20, 22, 57]. The introduction of *tiered-trust domains* in Nest (*i.e.,* via Weave) offers an encouraging start to the incorporation of integrity guarantees into smart home platforms [29].

Lesson 2: *Nest Product Reviews would benefit from at least light-weight static analysis.* Our findings demonstrate numerous violations of the Nest design policies that should have been discovered during the product review. Moreover, the review guidelines also state that products that do not securely transmit tokens will be rejected [32], but our simple static analysis using MalloDroid discovered numerous SSL vulnerabilities in Nest apps ($\mathcal{F}_{10}$), of which one can be exploited (Section 7). We recommend the integration of light-weight tools such as MalloDroid in the review process.

Lesson 3: *The security of the smart home indirectly depends on the smart phone (apps).* Smartphone apps have been known to be susceptible to SSL misuse [9], among other security issues (*e.g.,* unprotected interfaces [5]). Thus, unprotected smartphone clients for smart home devices may enable the attacker to gain access to the smart home, and launch further attacks, as demonstrated in Section 7. Ensuring the security of smart phone apps is a hard problem, but future work may triage smartphone apps for security analyses based on the volume of smart home devices/platforms they integrate with, thereby, improving the apps that offer the widest possible attack surface to the adversary.

Lesson 4: *Popular but simpler platforms need urgent attention.* The startling gaps in the access control of Hue demonstrate that the access control of other simple (*i.e.,* homogeneous) platforms may benefit from a similar holistic security analysis ($\mathcal{F}_2$, $\mathcal{F}_3$).

9 VULNERABILITY REPORTING

We have reported the discovered vulnerabilities to Philips ($\mathcal{F}_2$, $\mathcal{F}_3$), Google ($\mathcal{F}_1$, $\mathcal{F}_4$→$\mathcal{F}_9$), and TP Link ($\mathcal{F}_{10}$), and have received confirmations from all the vendors. TP Link has since fixed the SSL flaw in the latest version of the app. Philips Hue is currently analyzing third party apps for the specific behavior discussed in this paper, and will eventually roll out a fix to their access control policy. We have also provided recommendations to Google on improving the safety of routines, which is a design challenge that may be hard to immediately address.

10 THREATS TO VALIDITY

1. SSL MiTM for different Android versions: Our attack described in Section 7 has been tested and is fully functional on a Nexus 7 device running Android version 4.4.2. However, we have recently observed that the MiTM proxy is blocked when intercepting connections from a Pixel 2 device running the latest version of Android (*i.e.,* 8.1.0). Our hypothesis is that the TP Link Kasa app changes its SSL API use based on the Android API version, and we are currently working on locating at what Android version (*i.e.,* between 4.4.2 and 8.1.0) the SSL component of our described attack no longer functions. However, this caveat does not change the fact that our attack is feasible under certain settings, or that third-party Android apps may often have exploitable SSL verification vulnerabilities [9, 38, 43]. It is important to note that the SSL compromise is a well-studied engineering challenge, and is not the focal point of the lateral privilege escalation exploit we describe, which occurs primarily because of routines implemented using shared global variables in Nest ($\mathcal{F}_4$→$\mathcal{F}_6$).

2. Number of devices and apps: For the analysis in Section 6, our set of 9 devices (*i.e.,* 7 real and 2 virtual) allowed us to integrate a set of 39 apps into our Nest platform (*i.e.,* the Apps_{nest} dataset), out of the around 130 "Works with" Nest apps we found. Therefore, while we cannot say that our findings ($\mathcal{F}_6 \rightarrow \mathcal{F}_9$) generalize to all the apps compatible with Nest, we can certainly say that they are valid for a significant minority (*i.e.,* over 27%).

3. Local and Remote exploits of Hue: Our exploits for the Philips Hue platform demonstrated in Section 4 ($\mathcal{F}_2$ and $\mathcal{F}_3$) can be executed from an app operating on the same local network as the Hue bridge. This is feasible, as the attacker-controlled app simply needs to be on the same network (*i.e.,* not even on the victim's device). The vulnerabilities we describe may also be remotely exploited, as access control enforcement remains the same for remote access.

11 RELATED WORK

Smart home platforms are an extension of the new modern OS paradigm, the security problems in smart home platforms are similar to prior modern OSes (*e.g.,* application over-privilege, incorrect platform enforcement). As a result, some of the same techniques may be applied in detecting such problems. For instance, in a manner similar to Felt et al.'s seminal evaluation of Android permission enforcement [10], our work uses automated testing to derive permission maps and compares the maps to the platform documentation. We also leverage lessons from prior work on SSL misuse [9, 38, 43, 47] to perform the SSL Analysis (Section 6.2) and the MiTM exploit (Section 7). The lack of transitivity in access control that we observe is similar to prior observations on Android [5, 7, 11, 23, 24]. However, the implications of intransitive enforcement are different in the smart home space, and, to our knowledge, some of the key analyses performed in this paper is novel across modern OS research (*e.g.,* exploitation of home automation routines and the ineffectiveness of Nest's product review). The novelty of this paper is rooted in using lessons learned from prior research in modern OS and application security to identify problems in popular but under-evaluated platforms such as Nest and Hue, and moreover, in demonstrating the potential misuse of home automation *routines* for performing lateral privilege escalation.

In the area of smart home security, the investigation by Fernandez et al. [12] into the SmartThings platforms and its apps is highly related to the study presented in this paper. However, our work exhibits key differences. For instance, the platforms explored in this paper (*i.e.,* Nest and Hue) are popular, and have key differences relative to SmartThings (Section 2). Moreover, while Fernandez et al. focus on application overprivilege, this work studies the utility and security of routines, and leverages routines to demonstrate the first instance of lateral privilege escalation on smart home platforms. Our analysis of permission text artifacts, product review-based defense in Nest, and SSL-misuse in apps leads to novel findings that facilitate the end-to-end attack. Finally, we demonstrate that simpler platforms (*i.e.,* Hue) fail to provide bare-minimum protections.

Aside from this closely-related work, prior work has demonstrated direct attacks on smart home platforms and applications. For instance, Sukhvir et al. attack the communication and authentication protocols in Hue and Wemo [37], Sivaraman et al. attack the home's firewall using a malicious device on the network [44], and a Veracode study demonstrated issues in a range of products such as the MyQ Garage System and Wink Relay [53]. Our work performs a holistic security evaluation of the access control enforcement in DSB platforms (*i.e.,* Nest and Hue) and their applications, and is complementary to such per-device security analysis.

Prior work has also analyzed the security of trigger-action programs. Surbatovich et al. [49] analyzed the security and privacy risks associated with IFTTT recipes, which are trigger-action programs similar to routines. The key difference is that Surbatovich et al. examines the safety of individual recipes, while our work explores routines that may be safe on their own (*e.g.,* when *home*, turn off the Nest Cam), but which may be used as gadgets by attackers to attack a high-integrity device from a low-integrity device.

In a similar vein, Celik et al. [4] presented Soteria, a static analysis system that detects side-effects of concurrent execution of Samsung's smart apps. The problem explored in our paper is broadly similar to Celik et al.'s work, *i.e.,* both papers explore problems that arise due to the lack of transitive access control in smart homes. While the techniques that underlie Soteria have advanced the state of the art for analyzing smart home products, our paper exhibits two key differences that demonstrate the novelty of our analysis. First, Soteria does not aim to address the adversarial use of routines as mechanisms to perform a lateral privilege escalation. As a result, it would not detect the attack discussed in Section 7, since the precondition for the attack is not a routine (*i.e.,* it is the exploitation of SSL vulnerability in the Kasa app, which allows us to steal the authorization token and misuse the *away* permission allocated to Kasa). Second, this paper is novel in its analysis of runtime prompts and permission descriptions on home automation platforms, and uncovers problems in how users are informed of specific sensitive automation actions ($\mathcal{F}_8 \rightarrow \mathcal{F}_9$), and how the permissions that enable such actions ($\mathcal{F}_5$) are described.

Finally, prior work has proposed novel access control enhancements, which may alleviate some of the concerns raised in this paper. ProvThings [54] provides provenance information that may allow the user to piece together evidence of some of the attacks described in this paper, but does not prevent the attacks themselves. On the contrary, ContextIoT [19] provides users with runtime prompts describing the context of sensitive data accesses, which may alert users to unintended execution of routines ($\mathcal{F}_4$), *at the cost of reducing automation.* Further, SmartAuth [52] analyzes the consistency of application descriptions with code, and may benefit the Nest product review in determining the correctness of permission descriptions.

12 CONCLUSION

Smart home platforms and devices operate in the users' physical space, hence, evaluating their security is critical. This paper evaluates the security of two such platforms, Nest and Hue, that implement home automation *routines* via centralized data stores. We systematically analyze the limitations of the access control enforced by Nest and Hue, the exploitability of routines in Nest, the robustness of Nest's product review, and the security of third-party apps that integrate with Nest. Our analysis demonstrates ten impactful findings, which we leverage to perform an end-to-end lateral privilege escalation attack in the context of the smart home. Our findings motivate more systematic and design-level defenses against attacks on the integrity of the users' smart home.

REFERENCES

[1] 2018. Online Appendix. https://sites.google.com/view/smart-home-routines-analysis-2/home. Accessed August 27, 2018.

[2] BetterCAP. Accessed June 2018. BetterCAP stable documentation. https://www.bettercap.org/legacy///.

[3] K. J. Biba. 1977. *Integrity Considerations for Secure Computer Systems.* Technical Report MTR-3153. MITRE.

[4] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. 2018. Soteria: Automated IoT Safety and Security Analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC)*. 147–158.

[5] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. 2011. Analyzing Inter-Application Communication in Android. In *Proceedings of the 9th Annual International Conference on Mobile Systems, Applications, and Services*.

[6] Ry Crist. Accessed September 2018. A smart home divided: Can it stand? https://www.cnet.com/news/a-smart-home-divided-can-it-stand/.

[7] Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and Dan S. Wallach. 2011. Quire: Lightweight Provenance for Smart Phone Operating Systems. In *Proceedings of the USENIX Security Symposium*.

[8] Engadget. Accessed June 2018. SmartThings shows off the ridiculous possibilities of its connected home system. https://www.engadget.com/2014/01/11/smartthings-labs/.

[9] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*.

[10] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. 2011. Android Permissions Demystified. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[11] Adrienne Porter Felt, Helen J. Wang, Alexander Moshchuk, Steven Hanna, and Erika Chin. 2011. Permission Re-Delegation: Attacks and Defenses. In *Proceedings of the USENIX Security Symposium*.

[12] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security analysis of emerging smart home applications. In *Security and Privacy (SP), 2016 IEEE Symposium on*. 636–654.

[13] Gartner. Accessed June 2018. Gartner Says 8.4 Billion Connected Things Will Be in Use in 2017, Up 31 Percent From 2016. https://www.gartner.com/newsroom/id/3598917.

[14] Google Play. Accessed June 2018. Kasa for Mobile. https://play.google.com/store/apps/details?id=com.tplink.kasa_android.

[15] Google Play. Accessed June 2018. Keen Home. https://play.google.com/store/apps/details?id=com.hipo.keen//.

[16] Google Play. Accessed June 2018. Nest. https://play.google.com/store/apps/details?id=com.nest.android.

[17] Google Play. Accessed June 2018. Philips Hue. https://play.google.com/store/apps/details?id=com.philips.lighting.hue2.

[18] IFTTT. Accessed June 2018. IFTTT helps your apps and devices work together. https://ifttt.com/.

[19] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z Morley Mao, Atul Prakash, and Shanghai JiaoTong Unviersity. 2017. ContexIoT: Towards providing contextual integrity to appified IoT platforms. In *Proceedings of the 2017 Network and Distributed System Security Symposium (NDSS)*.

[20] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. 2007. Information Flow Control for Standard OS Abstractions. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*. 321–334.

[21] Carl E. Landwehr, Alan R. Bull, John P. McDermott, and William S. Choi. 1994. A Taxonomy of Computer Program Security Flaws. *ACM Computing Surveys (CSUR)* 26, 3 (Sept. 1994).

[22] Andrew C. Myers and Barbara Liskov. 1997. A Decentralized Model for Information Flow Control. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*.

[23] Adwait Nadkarni, Benjamin Andow, William Enck, and Somesh Jha. 2016. Practical DIFC Enforcement on Android. In *Proceedings of the 25th USENIX Security Symposium*.

[24] Adwait Nadkarni and William Enck. 2013. Preventing Accidental Data Disclosure in Modern Operating Systems. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[25] Nest. Accessed June 2018. Keen Home - Works with Nest Store. https://workswith.nest.com/company/leaksmart/leaksmart//.

[26] Nest. Accessed June 2018. LeakSmart - Works with Nest Store. https://workswith.nest.com/company/leaksmart/leaksmart//.

[27] Nest. Accessed June 2018. Nest x Yale Lock - Works with Nest Store. https://workswith.nest.com/company/yale/nest-x-yale-lock//.

[28] Nest. Accessed June 2018. Somfy Protect - Works with Nest Store. https://workswith.nest.com/company/somfy-protect-by-myfox-sas/works-with-somfy-protect//.

[29] Nest. Accessed June 2018. Weave. https://nest.com/weave/.

[30] Nest Developers. Accessed June 2018. How to Choose Permissions and Write Descriptions. https://developers.nest.com/documentation/cloud/permissions-overview.

[31] Nest Developers. Accessed June 2018. OAuth 2.0 Authentication and Authorization. https://developers.nest.com/documentation/cloud/how-to-auth.

[32] Nest Developers. Accessed June 2018. Product Review Guidelines. https://developers.nest.com/documentation/cloud/product-review-guidelines.

[33] Nest Labs. Accessed June 2018. Kasa - Works with Nest Store. https://workswith.nest.com/company/tp-link-research-america-corp/kasa.

[34] Nest Labs. Accessed June 2018. Nest Developers. https://developers.nest.com///.

[35] Nest Labs. Accessed June 2018. Nest Simulator. https://developers.nest.com/documentation/cloud/home-simulator.

[36] Nest Labs. Accessed June 2018. Works with Nest. https://nest.com/works-with-nest//.

[37] Sukhvir Notra, Muhammad Siddiqi, Hassan Habibi Gharakheili, Vijay Sivaraman, and Roksana Boreli. 2014. An experimental study of security and privacy risks with emerging household appliances. In *Proceedings of the 2014 IEEE Conference on Communications and Network Security (CNS)*. 79–84.

[38] Lucky Onwuzurike and Emiliano De Cristofaro. 2015. Danger is my middle name: experimenting with SSL vulnerabilities in Android apps. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 15.

[39] Philips. Accessed June 2018. Philips Hue: Your Personal Wireless Lighting System. https://www2.meethue.com/en-us/about-hue.

[40] Philips Hue Developers. Accessed June 2018. Philips hue API. https://developers.meethue.com/philips-hue-api.

[41] Dave Piscitello. 2016. What is Privilege Escalation. https://www.icann.org/news/blog/what-is-privilege-escalation.

[42] Popular Science. Accessed June 2018. Stop shouting at your smart home so much and set up multi-step routines. https://www.popsci.com/smart-home-routines-apple-google-amazon/.

[43] Bradley Reaves, Nolen Scaife, Adam Bates, Patrick Traynor, and Kevin R.B. Butler. 2015. Mo(bile) Money, Mo(bile) Problems: Analysis of Branchless Banking Applications in the Developing World. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. 17–32.

[44] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. 2016. Smartphones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 195–200.

[45] Smartthings Developers. Accessed June 2018. Documentation. http://developer.smartthings.com/.

[46] SmartThings Support. Accessed June 2018. Routines in the SmartThings Classic app. https://support.smartthings.com/hc/en-us/articles/205380034-Routines-in-the-SmartThings-Classic-app.

[47] David Sounthiraraj, Justin Sahs, Zhiqiang Lin, Latifur Khan, and Garrett Greenwood. 2014. SMV-Hunter: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps. In *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*.

[48] Stringify. Accessed June 2018. Stringify | Change Your Life by Connecting Everything. https://www.stringify.com//.

[49] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. 2017. 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*. 1501–1510.

[50] TechCrunch. Accessed June 2018. Google Assistant is adding Routines and location-based reminders. https://techcrunch.com/2018/02/23/google-assistant-is-adding-routines-and-location-based-reminders/.

[51] The Verge. Accessed June 2018. You can soon activate multi-step routines in Alexa with a single command. https://www.theverge.com/2017/9/27/16375050/alexa-routines-echo-amazon-2017/.

[52] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, XianZheng Guo, and Patrick Tague. 2017. SmartAuth: User-Centered Authorization for the Internet of Things. In *Proceedings of the 26th USENIX Security Symposium*.

[53] Veracode. 2016. The Internet of Things Poses Cybersecurity Risk. https://info.veracode.com/whitepaper-the-internet-of-things-poses-cybersecurity-risk.html.

[54] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. 2018. Fear and Logging in the Internet of Things. In *Network and Distributed Systems Symposium*.

[55] Yeti. Accessed June 2018. Yeti - Simplify the control of your smart home. https://getyeti.co/.

[56] Yonomi. Accessed June 2018. Yonomi app – Yonomi. https://www.yonomi.co.

[57] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. 2006. Making Information Flow Explicit in HiStar. In *Proceedings of the 7th symposium on Operating Systems Design and Implementation (OSDI)*. 263–278.