Practical Integrity Validation in the Smart Home with HomeEndorser

Anonymous Author(s)

ABSTRACT

Modern smart home platforms facilitate home automation using trigger-action *routines*. While routines enable flexible automation, they may also cause serious threats to system integrity: untrusted third-parties may use platform APIs to modify the abstract home objects (AHOs) that high-integrity devices (*e.g.*, security camera) rely on (*i.e.*, as triggers). As most accesses to AHOs are legitimate, removing the permissions or applying naive information flow controls would not only fail to prevent these problems, but also break useful functionality. Therefore, this paper proposes the alternate approach of *home abstraction endorsement*, which *endorses* a proposed change to an AHO by correlating it with expected environmental changes. We present the HomeEndorser framework, which provides a policy model to express specific changes in device states as endorsement policy templates that are automatically instantiated in a given configuration (based on device availability/placement), and a platform-based reference monitor to mediate all API requests to change AHOs. We implement HomeEndorser as an enhancement to the HomeAssistant platform, and demonstrate less than 10% performance overhead and no false alarms under realistic usage scenarios, as well as derive policy templates for 6 key AHOs.

ACM Reference Format:

Anonymous Author(s). 2023. Practical Integrity Validation in the Smart Home with HomeEndorser. In *Proceedings of 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec) (WiSec 2024).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The popularity of smart home devices [55] can be attributed in part to the convenience of home automation, wherein smart home devices automatically react to changes in the user's physical environment. For example, the user may configure a security camera to begin recording when they leave home, but turn OFF when they return to preserve their privacy [35]. Such automation is expressed using trigger-action programs known as *routines*, that execute an action (turning camera OFF) in response to a trigger ("away" to "home").

Routines are often enabled via third-party integrations that automate device-actions by leveraging platform APIs to modify two distinct types of objects, *device states* (*e.g.*, the ON/OFF state of a light bulb), and *abstract home objects* (AHOs) that are not device-specific (*e.g.*, home/away, hereby referred to as the `home` AHO).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WiSec 2024, May 27–30, 2024, Seoul, Korea

© 2023 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

https://doi.org/10.1145/nnnnnnn.nnnnnnn

A particularly dangerous attack vector that emerges from this setting is where adversaries gain privileged access to devices *indirectly*, by falsifying an AHO that a high-integrity device depends on via a routine. For instance, consider a situation wherein an adversary may want to disable the security camera to perform a burglary unnoticed, but may not have direct API access to it. An adversary with API access to modify an AHO that the security camera depends upon to deactivate, such as home/away being set to "home," may disable the security camera without direct access [29, 30].

A naive approach to address such false AHO-changes would be to prevent third-parties from accessing AHOs altogether, or to severely restrict permissions based on static [34, 44, 58] or runtime context [27]. However, in practice, such solutions may result in infeasible usability penalties, as AHOs are often computed via third-party services *of the user's choice* that infer AHOs by querying a combination of device states [22], use other (proprietary) approaches [51], or enable the user to set them [31]. Thus, we instead recognize that at its core, this is *an integrity problem* analogous to those seen in operating systems: a high-integrity process (here, the security camera) relies on the value of an object (*i.e.*, the `home` AHO) that can be modified by untrusted parties. Hence, we must directly address the *lack of integrity validation of AHO changes in the smart home*.

Information flow control (IFC) has often been proposed to ensure the integrity of information consumed by sensitive processes [3, 16, 17, 33, 37, 66], through dominance checks that regulate flows based on subject and object labels [3]. A simple IFC solution in this case would be to mark AHOs such as `home` as high integrity, while marking third-party services as low integrity, which effectively results in preventing third-parties from modifying AHOs. However, such restrictions may prevent valid changes to AHOs from services chosen by the user, resulting in false denials from the user's perspective; *e.g.*, 19/33 NEST integrations from a prior dataset [29] would be blocked due to such restrictive labeling.

IFC systems rely on *endorsement* [5, 33, 65, 66] to overcome this limitation, allowing trusted programs to change labels of objects to permit flows that would normally violate IFC. However, determining the conditions where an endorsement is allowable in IoT systems is a challenge; indeed, prior work has often avoided addressing this directly, and instead facilitates endorsement by assigning the authority to certain *trusted* high-integrity processes, thereby delegating the task of *how to endorse correctly* to the programmer or administrator [9, 32, 33, 48]. However, in our case, smart home users may lack information about dependencies among devices and AHOs to do this correctly. So, we ask instead: Is there something else we can rely on to provide endorsement for *practical* integrity validation?

Yes – the cyber-physical nature of the smart home provides us with a unique opportunity for practical endorsement, in the form of ground truth observations from devices (*i.e.*, device state changes) that can validate proposed changes to AHOs. For instance, we can endorse the change to the `home` AHO (from "away" to "home") if the door lock was legitimately unlocked (*i.e.*, with the correct keycode) recently, as it represents the home owner's intent and attempt to enter the home. In fact, rather than only depending on one device, we can leverage all the devices that may observe state changes that may correlate with each sensitive AHO change, such as motion sensors, microphones, etc. that may be available to detect changes that support the `home` AHO change. Thus, we state the following claim that forms the foundation of this work:

> Abstract home objects (AHOs) shared among third-party services and devices for the purpose of home automation are inherently tied to a home's physical state. Thus, any state change or modification to an AHO via an API call can be *endorsed* using the *local context* of the home, consisting of changes in a combination of device states.

Contributions: We introduce the paradigm of *home abstraction endorsement* to validate changes to AHOs initiated by untrusted API calls, and propose the HomeEndorser framework to enable it. HomeEndorser does not continuously monitor AHOs, but focuses on API-induced *changes* to AHOs, and performs a sanity check using *policies* that rely on recent physical state changes in smart home devices. If the check fails, the state change is denied, and the user is informed. HomeEndorser's preemptive action prevents future automation based on maliciously changed AHOs. We make the following contributions in exploring this novel design space:

1. Home Abstraction Endorsement: We introduce *home abstraction endorsement*, which leverages local device state changes to endorse proposed AHO-changes, thereby making IFC endorsement practical by exploiting the cyber-physical nature of the smart home.

2. The HomeEndorser Framework: We design the HomeEndorser framework consisting of (1) a *policy model* that allows a unified expression of location-specific device instances within a single policy (*e.g.*, endorsing `home` via multiple physical entry points), (2) a platform-based *reference monitor* that mediates sensitive state changes using these policies, and finally, (3) a mechanism to enable experts to generate endorsement policy templates (defined once for all homes), which HomeEndorser then automatically instantiates for each home, enforcing the *most restrictive but feasible policy*. We will release our source code upon publication.

3. Evaluation: We implement HomeEndorser on HomeAssistant, a popular open-source platform, and evaluate it with extensive experimental and empirical analyses. (1) We demonstrate that the home abstraction endorsement is feasible, even when using a limited set of correlating devices, by generating policies to endorse changes to the `home` AHO. (2) We demonstrate the generality of our policy model by identifying several attributes that may be used to endorse five additional AHOs. (3) We show that HomeEndorser is not susceptible to false denials, and in fact, may prevent accidental unsafe situations, by systematically testing it using 10 home usage scenarios drawn from prior work [28], and 400 realistic event sequences [35], in a smart home (apartment) testbed. (4) We demonstrate the effectiveness of HomeEndorser's integrity validation using specific attack scenarios. (5) We measure HomeEndorser's practical performance overhead with micro/macro benchmarks (9.7-12.2% on average). (6) Finally, we demonstrate the modest effort required to generate policy templates, configure HomeEndorser in user homes, and integrate HomeEndorser in popular platforms.

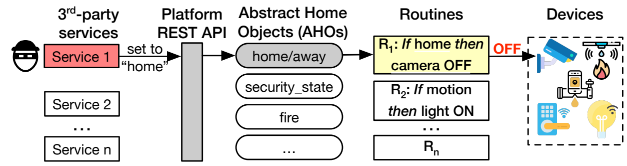

Figure 1: An attack on the security camera through the manipulation of two shared state objects by adversary-controlled integrations.

2 MOTIVATION

Instead of cloud-hosted "apps", current smart home platforms (e.g., SmartThings, Nest) provide API access to (1) *device states* of individual devices (*e.g.*, the ON/OFF state, battery), and (2) *abstract home objects (AHOs)* that are not associated with any specific device (*e.g.*, the `home` AHO to indicate whether the user is home or away). This allows for seamless integration with third-party services (*e.g.*, IFTTT [24], Yonomi [64]).

Need for Integrity Validation of AHOs: AHOs are a key component of routines, as they form the *conditions* that need to be met for a routine to execute, *e.g.*, turn the security camera on *when* user is `home`. In fact, our empirical analysis of 184 SmartThings marketplace apps uncovered 33 unique flows to security-sensitive devices through AHOs (full list in online appendix [2]). Since AHOs can be designated by the user to be set in several ways, such as via a direct command [31], a third-party service computing `home` AHO based on phone's location [57], or a proprietary/undisclosed method/device [51], this exposes a dangerous attack vector where adversaries can gain privileged access to security-sensitive devices (*e.g.*, door lock) indirectly by falsifying an AHO's state. For instance, an adversary Bob who cannot compromise or modify a high-integrity device directly can modify AHOs *directly through an API* to trigger targeted routines and *transitively* attack the device.

Therefore, this paper recognizes this problem as a smart home-specific instance of the classical *OS integrity problem* (*e.g.*, Biba [3]), wherein a high-integrity process (*i.e.*, the camera) relies on an object (*i.e.*, the `home` AHO) which can also be modified by low-integrity process (*e.g.*, the Kasa integration). Similar to OSes, platforms must provide systematic integrity protections for high-integrity objects (*i.e.*, AHOs) since untrusted principles can access them in many ways (*e.g.*, numerous API calls or routines) to manipulate high-integrity subjects (*e.g.*, security sensitive devices).

Consider the following motivating example involving an attack on a high-integrity device via the `home` AHO (inspired by a *demonstrated attack* from prior work by Kafle *et al.* [29, 30]):

Motivating Example: Alice has configured two routines in her home (advertised by Simplisafe [50] and NEST [40]): **(R1)** the camera turns ON when Alice leaves `home` for monitoring, and **(R2)** the camera turns OFF when Alice returns `home`. Bob seeks to burglarize Alice's home *without being monitored by the camera*, but does not have direct API access to the camera. However, Bob controls one (or more) third-party services connected to Alice's home, either because Alice installed Bob's service, or Bob compromised a vulnerable service (*e.g.*, the TP-link Kasa integration via MiTM attack, as demonstrated in prior work [29, 30]). Thus, Bob changes `home` to the value "home", falsely suggesting that Alice is home and triggering **(R1)**, thereby disabling the camera, as shown in Figure 1.

This problem is not just limited to the `home` AHO. Consider the `security_state` AHO, also used in routines (*e.g.*, from Ring [45], from TotalConnect [1]) to control security devices, which are "armed" when `security_state` is set to "deter", and "disarmed" when it is set to "ok". If Bob controls a service with access to `security_state`, he can set it to "ok" and disable the camera.

Threat Model: In line with the motivating example, we consider an adversary who controls/compromises any third-party service connected to the target's home, with the objective to *indirectly* modify high-integrity devices. Such third-party services can use the platform APIs to create and trigger automations via AHOs. Similar to prior work dealing with API misuse [6, 7, 14, 36], we assume the platform to be trusted and devices to be tamper-proof, as an attacker with direct access to either can simply set the device to their desired state without having to use the API.

3 LIMITATION OF PRIOR APPROACHES

Based on the threat model, we now discuss the three main limitations from prior approaches in addressing this problem:

1. Breaking existing functionalities:: As high-integrity devices rely on AHOs such as `home`, traditional wisdom dictates that low-integrity (or third-party) integrations should simply be disallowed from writing to these objects. However, API-based platforms are designed such that integration/service of user's choice can control the platform, *e.g.*, IFTTT creating routines for Nest devices [23]. Thus, disallowing *the user's choice* of third-party integration from writing to AHOs breaks useful services that the user relies on (*e.g.*, IFTTT, Kasa), which is a prohibitive cost in terms of user experience that platforms may find undesirable. In fact, in 2019, Google had to backtrack [10, 20] after ending its "Works with NEST" program in favor of a more restricted "Works with Google Assistant" program that would only be open to vetted partners. Following opposition from both users and third-party integrations [12, 13, 25, 59], it eventually offered a more flexible program that allowed a broader set of developers access to the internal home states (including AHOs) [20].

2. Focus on App analysis: Most of prior work has attempted to address this issue as an "application" security problem, which assumes a different threat model that is no longer applicable in current, API-based platforms. That is, prior work [6, 7, 27, 42, 44, 58] analyzes or instruments developer-defined automation programs (*i.e.*,*IoT apps*) to limit privilege (or API access) based on whether apps require it. However, this does not solve the core issue. First, *IoT apps are now black boxes*. That is, platforms do not host apps anymore [52], and connected third-party services trigger automations through the platform API instead. Hence, prior solutions that rely on analysis or instrumentation of source code (*e.g.*, HAWatcher [18], IoTGuard [7]), as well as those deployed outside of the platform (*e.g.*, PFirewall [8], Maverick [36]) both fail as third-party services are closed-source and communicate with the platform directly (*i.e.*, cloud-to-cloud). Second, the lack of AHO integrity *cannot be addressed by limiting privilege*. Even if we limit API access exclusively to services that require it [47], an adversary may still compromise those services and exploit the privilege (see the Motivating example in Section 2).

3. Lack of focus on AHOs: Similar to the focus on app analysis, the policy enforcement in prior work [6–8, 42, 60, 62, 63] is designed for a different threat model which does not address the core issue of AHO manipulation. The policy enforcement chiefly focuses on (1) preventing unsafe states reached via "app interactions" or "chaining" of multiple IoT apps (*e.g.*, IoTGuard [7]), or (2) preventing unsafe states in individual, sensitive devices such as a door lock (*e.g.*, Expat [63]). However, unlike this work, none focus on the modifications to AHOs, allowing an attacker to control routines, and *bypass policy enforcement in prior work altogether*. For instance, consider this policy from Expat [63]: *frontDoorLock: Front door should be locked when the user is away*. Expat enforces this policy by checking the value of presence_state (analogous to the `home` AHO). Hence, any third-party service that can modify presence_state (*e.g.*, demonstrated attack in prior work [29][30]) can trivially bypass this policy, and lead the home to an unsafe state. Similarly, consider policy *P8* from PatrIoT [62], *Deny surveillance camera to get turned off except user is at home*. This is equivalent to the routine Alice deploys in the motivating example, which allows Bob to trivially put the home in an unsafe state by manipulating the `home` AHO.

Therefore, there is a need for a solution that is (1) practical, *i.e.*, does not break functionality by preventing third-parties from accessing AHOs, and (2) *effective*, enabling integrity validation of AHO changes. This paper proposes the moderate route *i.e.*, ***runtime validation of proposed changes to AHOs***, to enable proactive integrity checking that is compatible with platform design and user choices.

4 DESIGN GOALS

This paper introduces the novel paradigm of ***home abstraction endorsement*** that provides the following *integrity guarantee* for AHOs: In the event that an untrusted service uses the platform API to modify a critical AHO, *e.g.*, `home` or `security_state`, the modification will be allowed iff it is consistent with the *local state of the home*, composed of the physical device states. Our approach builds upon the concept of trusted "guards" in the Biba integrity model [3], wherein a high integrity subject cannot receive input from a low integrity subject unless it is endorsed by a trusted guard. Similarly, we envision endorsement policies that apply trusted device states and hence serve as the trusted guards, ensuring the validity of API requests to change the AHOs that high-integrity devices rely on. Our design is guided by the following goals:

- **G1** *Expressive and Practical Endorsement Policies.* The endorsement policy structure must be designed in a way that allows it to *express common deployment factors* in smart homes that may affect the endorsement, such as *device availability* and *locality*.
- **G2** *Complete Mediation.* Given that third-party services (or apps) are black boxes [19, 54], the reference monitor should be *app-agnostic*, *i.e.*, should not depend on the analysis/instrumentation of apps/services, but should provide complete mediation for all API calls that modify AHOs, *irrespective of their origin*.
- **G3** *Tamperproofness.* Although our endorsement approach relies on device states, several device states may be modifiable by untrusted services via API. Thus, our reference monitor must *only rely on trustworthy states* modifiable only by devices.
- **G4** *Freshness.* Endorsing an AHO change may require the reference monitor to examine *recent changes* in the states of physical devices, rather than simply reading the current state (*e.g.*, as sensor states may reset after apprising the platform of an event).

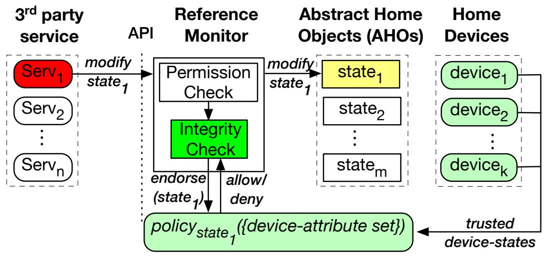

Figure 2: A conceptual overview of home abstraction endorsement.

G5 *Minimal Performance/Management Overhead.* The framework should minimize any delay *perceivable by the user*, as well as deployment and management effort.

5 THE HOMEENDORSER FRAMEWORK

We propose HomeEndorser, a framework that enables home abstraction endorsement as shown in Figure 2. When a third party service attempts to modify an AHO using the platform-API, HomeEndorser enforces an *integrity check* in addition to the platform's permission check for endorsing the proposed change. To endorse the proposed AHO-change, HomeEndorser checks the corresponding *endorsement policy* that uses recent changes in device-attributes/states (*e.g.*, motion detected, door lock unlocked). For example, to endorse a proposed change from "away" to "home" in the `home` AHO, *one* possible policy would be: *if door-lock has been unlocked recently (i.e., using the correct keycode)*, then ALLOW the change, else DENY.

Key observation: HomeEndorser's endorsement policy design accounts for *device locality* or placement in the home, due to a key observation: while AHOs such as `home`, `fire`, or `security_state` are global values that apply to the entire home, a valid change in them can be sufficiently reflected in *one or more localized events*. For example, a proposed change from "away" to "home" in the `home` AHO would be valid if the door lock *at the front door* was unlocked successfully, *or* if the one in the back was unlocked successfully. Similarly, a state change to `fire` is valid if *any* of the smoke detectors, in *any one location* in the home, detects smoke.

Location-specific policy model: This observation motivates our location-specific policy model (see Sec. 5.1), in which policy templates are composed of mutually exclusive, location-specific predicates, with each predicate representing device states at a particular location in the house, only one of which has to be satisfied for endorsement. The benefit of such a model is that the user does not need to have the devices available at *all* possible locations, but *any* one location, making it more practical (**G1**). However, a tradeoff is that our model does not currently support AHOs that do not exhibit this property (*i.e.*, require devices state from several locations together for endorsement), although we have not encountered such an example in the 5 other AHOs studied (see Sec. 7.2).

Flexible policy templates and automatic instantiation: HomeEndorser's flexible policy model allows *general expert-defined policy templates* (see Sec. 5.3) that it *automatically instantiates* in the context of a user's home (**G5**), using information regarding device availability and placement that is readily available in most smart home platforms (see Sec. 5.2). More specifically, HomeEndorser instantiates the ***most restrictive but feasible policy*** for each AHO-change to be endorsed, *i.e.*, location-specific predicates containing the largest aggregate of device-attributes that can be satisfied with devices available at each corresponding location. We define a *policy-template generation methodology* that allows experts to use open coding to define endorsement policy templates in a systematic, ground-up manner (see Sec. 5.3), using automatically-generated *endorsement attributes*, *i.e.*, *trusted* device-attributes that are either read-only or highly restricted by platforms, ensuring tamperproofness (**G3**).

Reference Monitor: HomeEndorser's *reference monitor* is integrated into the user's smart home platform in the form of an endorsement check in the platform's subsystem responsible for executing all API calls, ensuring complete mediation (**G2**, Sec. 5.2). Note that HomeEndorser's reference monitor considers the most recent change in device-attributes (**G4**), rather than the current state of the device-attribute, as the two may be different, since most sensors reset after a change, and because the most recent changes provide the context for endorsing the proposed AHO change. This decision is instrumental in eliminating unnecessary false denials (see Sec. 7.3).

5.1 Policy Model

A key challenge for HomeEndorser is designing a policy model that can alleviate two practical constraints. First, endorsement policies may consist of more than one device-attribute that must be checked together. Second, as described previously, AHO-changes can be endorsed via mutually-exclusive, localized state changes; *e.g.*, the front door lock or the back door lock can *either* endorse a change to `home`. We account for these constraints with a policy template expressed as a Disjunctive Normal Form (DNF) boolean formula:

Definition 1 (Endorsement Policy). The policy for endorsing a change in AHO x to value y, $P_x(y)$, is a DNF formula composed of one or more location-specific predicates (L_i), *i.e.*, $P_x(y) = L_1 \vee L_2 \vee ... \vee L_n$, where a location-specific predicate is defined as follows:

Definition 2 (Location-specific Predicate). A location-specific policy predicate L_i for location i (*e.g.*, entryway), *i.e.*, $L_i = d_j \wedge d_k \wedge ...d_m$, is a conjunction of one or more device-attribute checks d_j, defined as follows:

Definition 3 (Device-attribute Check). A device-attribute check d_j is a condition $d_j == s$, where s is a physical state that the particular device-attribute must have exhibited in the recent past, for the device-attribute check to return *true*.

To illustrate, let us express the policy from the motivating example for endorsing the `home` AHO's change to "home". We express the policy using a door lock and a motion sensor at the entry way, as well as the same devices at the rear entrance:

$$P_{\text{home}}(\text{home}) = (\text{door-lock\_lock} == \texttt{UNLOCKED} \wedge \text{motion\_sensor} == \texttt{ACTIVE})_{front-door} \vee (\text{door-lock\_lock} == \texttt{UNLOCKED} \wedge \text{motion\_sensor} == \texttt{ACTIVE})_{back-door}$$

The above policy considers both the door lock being unlocked, and motion being sensed, to prevent false negatives. That is, for both the conditions above to be true, a user would have to unlock the door *and then enter*, *i.e.*, confirming that they are home. On the contrary, if the user unlocks but leaves without entering, this policy condition would correctly result in a denial (as shown in Sec. 7.3). Similarly, the disjunction among location-specific predicates enables their independent evaluation, thereby allowing the AHO-change as

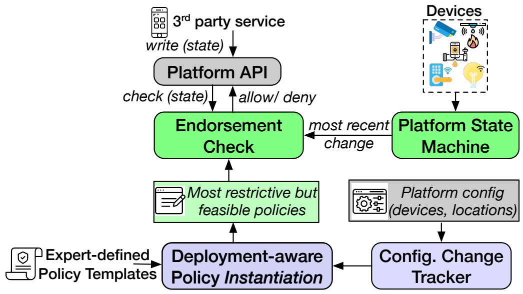

Figure 3: Design components of HomeEndorser's enforcement

long as *any one* evaluates to a *true* result. Finally, we define two policy *actions*: ALLOW and DENY, corresponding to the *true* or *false* values that the DNF formula results in, respectively.

5.2 Secure and Practical Enforcement

In a manner similar to prior solutions in the broader modern OS security space [21, 38, 39], we design HomeEndorser's enforcement as an enhancement to the OS (*i.e.*, the platform) itself. This decision is influenced by how third-party services are currently integrated in smart home platforms (*e.g.*, NEST and SmartThings v3), *i.e.*, as cloud endpoints that use RESTful APIs to interact with the platform, but execute on their own proprietary servers, without a way for the platform to inspect them. Therefore, our decision ensures that HomeEndorser will mediate all API commands from third-parties before they are executed (**G2**), in a manner agnostic to how third party integrations are implemented/deployed (*e.g.*, as black box network endpoints). We describe HomeEndorser's enforcement in terms of its three design components, as shown in Figure 3.

1. Deployment-aware Policy Instantiation: When HomeEndorser is first set up in a home, it leverages the platform's internal bookkeeping systems to extract all devices and device-locations. Then, for each AHO the user decides to endorse, it uses the policy templates (generated by experts using the policy model, as described later in Sec. 5.3) to instantiate the *most restrictive but feasible policy*, *i.e.*, a policy consisting of the largest applicable location-specific predicate(s), given the available devices and their locations. Such dynamic and adaptive instantiation is necessary to apply the policy templates to *any* home, given that a typical user's setup may only have a small subset of all the devices that can endorse a particular change, and only at a few (or single) locations. For instance, for a user who has a door lock and a motion sensor *only* at their front door, the policy template in Section 5.1 is instantiated as only one location predicate representing the said devices at the front door:

$$P_{\text{home}}(\text{home}) = (\text{door-lock\_lock} == \texttt{UNLOCKED} \land \text{motion\_sensor} == \texttt{ACTIVE})_{front-door}$$

Thus, HomeEndorser enforces the most restrictive, feasible policy for devices at each individual location in the home, and also reinstantiates the policy upon a configuration change, *i.e.*, addition, removal, or relocation of a new device (see Sec. 6 for implementation).

2. The Endorsement Check: HomeEndorser mediates all API requests, but only invokes the endorsement check if an AHO selected by the user for endorsement is about to be modified, in a manner similar to performance-preserving *hook activations* previously proposed for Android [21] (**G5**). HomeEndorser retrieves the instantiated policy for the AHO-change being endorsed and collects the most recent state changes of all the device-attributes in it. If the state of *all* the device-attributes in *any* predicate matches with the current policy, the decision is ALLOW, else DENY (and the user is notified).

Additionally, HomeEndorser considers the most granular value of a device-attribute for the enforcement check. For instance, consider that when Alice leaves, she sets `home` AHO to "away". To circumvent HomeEndorser, Bob could attempt to modify `home` (*i.e.*, back from "away" to "home") at the time of Alice's departure. This is possible because Alice leaving or coming home both involve (1) unlocking the door, and (2) triggering the door-way motion sensor. A naive endorsement approach would allow the AHO change by considering these state changes, even when triggered in the opposite order, because it matches $P_{home}(\text{home})$. However, smart home devices provide *unique* device attribute values even for similar actions, *i.e.*, the state value for unlocking the door using the keypad is different relative to simply unlocking it from the inside (*e.g.*, "owner" in the former case, and "manual" in the latter). HomeEndorser considers this available granularity, preventing such an attack.

3. Retrieving the *most recent* changes using the Platform State Machine: A naive approach of executing an endorsement check would be to query each device's *current state* at the time of check. However, such a check would most certainly fail and lead to false denials because most sensors detect and report a change, and then *reset to a predefined neutral state*. For example, recall the endorsement policy predicate to endorse `home` consisting of the door lock and the motion detector (assuming single location for simplicity):

$$\text{door-lock\_lock} == \texttt{UNLOCKED} \land \text{motion\_sensor} == \texttt{ACTIVE}$$

Unless the check happens exactly at the moment the user enters, the motion sensor will reset to INACTIVE immediately after detecting motion, causing a false denial. Thus, for correct endorsement, we check the *most recent but fresh change* in the device states (**G4**), *i.e.*, the last state change before the state automatically reset, within a configurable time threshold to ensure freshness (*e.g.*, one minute).

As described in Section 6, HomeEndorser obtains all recent device state changes and their timestamps from the platform state machine. The timestamp helps HomeEndorser discard states that are older than the preconfigured threshold, thereby preventing historical old states from causing *false allows*.

5.3 Data-driven Policy Template Generation

HomeEndorser defines a data-driven methodology to enable *experts* (*e.g.*, security researchers, platform vendors) to enumerate general endorsement policy *templates* that HomeEndorser automatically instantiates in the context of end-user homes (as previously described in Sec. 5.2). Our approach automatically creates a *device-attribute map*, *i.e.*, a comprehensive mapping between device types (*e.g.*, cameras, door locks) and the attributes they possess, and defines trusted *endorsement attributes* to be used for tamperproof endorsement. We then use open coding for identifying the *observations* and *inferences* that can be made from the endorsement attributes to generate templates using our policy model (see Sec. 5.1).

1. Generating the Device-Attribute Map: We automatically construct a comprehensive device-attribute map from several information sources selected based on platform popularity, and the potential of obtaining realistic mappings: (1) a device-resource specification from the Open Connectivity Foundation (OCF) [43], used by the platform IoTivity [26], (2) the NEST data store [41], (3) the SmartThings capability map [53], and (4) SmartThings device handlers [46]. As each of these sources exhibits a unique representation of devices and attributes, we develop customized, automated methods for extracting device-attributes from each source (details in online appendix [2]).

2. Endorsement Attributes for Tamperproofness: For tamperproof endorsement, HomeEndorser must be able to trust the information received from the participating endorsers *i.e.*, device-attribute pairs (**G3**). We achieve this goal by defining a trusted subset of device-attributes to be used for our checks, *i.e.*, *endorsement attributes*. We propose two categories of endorsement attributes: (1) *read-only attributes*, *i.e.*, which are only writable by devices, and not via API calls, rendering them read-only from the third-party API caller's perspective (*e.g.*, motion sensor reading), and (2) *designated attributes*, which are writeable in theory, but are considered high-integrity by platforms and prior security research [11, 56] alike (*e.g.*, locking the door lock), and hence, heavily restricted. For example, NEST only allows its platform app to unlock locks, but not third-party services. Both *read-only* and *designated* device attributes would have a *higher integrity level* than an AHO such as `home` as they are not modifiable by a third-party service, and hence, would be trusted to endorse it.

3. Generating Policy Templates from Inferences: We now address the question of *how* the endorsement attributes are used to endorse a specific AHO, by designing a holistic *inference-based* template generation process that is a one-time, expert-driven effort. We begin by identifying 5 additional AHOs from prior work [14, 29] and AHOs which we encountered when building our device-attribute map (*e.g.*, the `security_state` from NEST). Then, we consider each device-attribute, and identify the type of information sensed or observed by that device-attribute, which we then translate to an inference that could be used for endorsing an integrity-sensitive change in one or more of the AHOs. For example, the device-attribute pair <security-panel, disarmed> indicates that the security panel/keypad was recently disarmed, which may provide an inference to endorse the `home` AHO's proposed change to "home". We combine inferences identified for each AHO to construct policy templates using the structure defined in Section 5.1. That is, each inference becomes a part of the AHO's endorsement policy, which is then instantiated in a user's home. Sections 7.4 provides examples of instantiated policies under different scenarios (complete scenarios table in the online appendix [2]).

HomeEndorser's policy instantiation approach is also resistant to conflicts, as it instantiates only a single policy for a specific deployment context using a *quantitative* criteria. The instantiation criteria for the most restrictive but feasible policy is governed by the number of device-attributes used in predicates (larger number indicating more restrictions) that are feasible given the devices at specific locations, and not the actual values/states of the device-attributes in the predicates. Since only a single policy is instantiated in this way, the issue of policy conflict does not arise.

Finally, HomeEndorser's expert-driven template generation approach has several advantages over automatically-generated correlations in systems that learn from IoT app source code, event logs, and user activities [18]. First, as we do not trust apps, our policy templates are not susceptible to the problem of *false learning*, unlike correlations influenced by untrusted app code. Second, learning from app code is becoming infeasible (see Sec 2). Third, our approach is not privacy invasive as it does not involve large-scale collection of real user data/behavior. Fourth, HomeEndorser's endorsement outcomes are independent of the number of users in a home, in contrast with systems that learn correlations specific to users available during training, which may change at enforcement.

6 IMPLEMENTATION

This section describes our policy template generation study, as well as the reference monitor implemented in HomeAssistant. We plan to release all the code and data upon publication.

1. Policy Template Generation Study: We automatically generated a combined device-attribute map from all the data sources consisting of 100 device-types and 510 device-attribute pairs, of which were 41 endorsement attributes, *i.e.*, read-only or designated device-attributes. Two authors independently identified the inferences that could be drawn from these endorsement attributes to endorse changes to one or more of our 6 AHOs. The coders disagreed on 12/510 device-attribute pairs (2.4% disagreement rate), which were resolved via discussion (see details on disagreements in online appendix [2]). The inferences led to 10 endorsement attributes for `home` alone, which can feasibly instantiate several policies (see Sec. 7.1).

2. Implementation on HomeAssistant: We implemented HomeEndorser in HomeAssistant, a popular open-source platform. We set the default time threshold as 1 minute, which we found to be sufficient in our trials for a user to enter home, platform state machine to be updated, and user's service to set `home` AHO (details in online appendix [2]). HomeEndorser uses HomeAssistant's state machine to track most recent state changes and their timestamps, and also to intercept the incoming state change requests to mediate all API accesses. Furthermore, HomeEndorser uses callbacks in HomeAssistant's *Event Bus* to track the addition/removal of devices for re-instantiating policies as the home evolves. Finally, HomeEndorser keeps track of device-connectivity using the state machine, and falls back to the next most restrictive policy in case a device becomes unavailable at runtime. We provide log screenshots from the deployed HomeEndorser in our online appendix [2].

3. Policy Instantiation Using Platform Metadata: HomeEndorser extracts device-metadata from HomeAssistant, including device types (*e.g.*, door lock), and locations within the home (*e.g.*, front door). To instantiate the most restrictive but feasible policy for an AHO, for each location-specific predicate in the policy, HomeEndorser attempts to find the constituent devices in the same location, and selects the largest predicate that matches entirely for each location, *i.e.*, where all required devices are present.

7 EVALUATION

We evaluate HomeEndorser along 7 research questions:

• **RQ$_1$:** (*Feasibility of policy model*) Is it feasible to generate endorsement policies using a small subset of *endorsement attributes*?

• **RQ$_2$:** (*Generalizability of policy model*) Do policies exist for endorsing AHOs other than `home`?

• **RQ$_3$:** (*False Denials*) What is the rate of false denials in typical benign usage, i.e., when users intentionally cause AHO changes, and over a period of home automation usage?

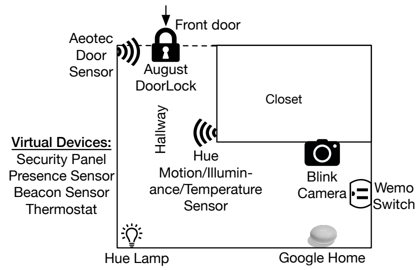

Figure 4: Layout of the physical device placement

Table 1: Sample policies for endorsing `home` **("away"→"home")**

	Policy
P_1	<security-panel, disarmed> ∧ <motion-sensor, active>
P_2	<Doorlock, unlocked> ∧ <presence-sensor, active> ∧ <beacon, active>
P_3	<Garage-doorlock,unlocked> ∧ <beacon,active>

• **RQ$_4$:** (*Security*) Does HomeEndorser prevent an attacker from escalating privilege to a high-integrity device (e.g., a camera) using one or more AHOs?
• **RQ$_5$:** (*Runtime Performance*) What is the performance overhead introduced by HomeEndorser?
• **RQ$_6$:** (*Cost*) How much effort is required to integrate and deploy HomeEndorser?

Experimental Setup: We installed HomeEndorser (HomeAssistant v0.112.0) on a Macbook Pro with 16GB RAM, connected 7 real and 4 *virtual* devices (full list in appendix A.1) in a room as shown in Figure 4, guided by deployment from prior work [28].

7.1 Feasibility of the Policy Model (RQ$_1$)

We identified 10 endorsement attributes for endorsing the `home` AHO-change from `away` to `home` using the approach in Section 5.3. However, HomeEndorser's policy instantiation automatically adapts to cases where any subset of the 10 attributes are present and enforces the most restrictive policy for that subset, allowing flexible device combinations in the user's home. For instance, a total of 1023 combination of devices at a single location are possible to enable HomeEndorser to endorse `home` AHO (full list in online appendix [2]). In fact, having one device-attribute at any location (*e.g.*, front door, garage) is enough to enable endorsement, as HomeEndorser instantiates devices in different location as mutually-exclusive, thus increasing the number of combinations.

For instance, as Table 1 shows, a user with a door lock and motion sensor, or another with a security panel, or another with a garage doorlock and a presence sensor, would all be able to endorse `home`. For *stronger* validation, the user may consider combination of device-attributes, up to all 10 device-attributes. This demonstrates that our approach is feasible, i.e., we can define a large number of diverse policies for an AHO (*i.e.*, `home`), using a limited set of endorsement attributes, and hence, increase the possibility of finding *a* policy that contains the limited set of devices a particular user possess (**RQ$_1$**).

7.2 Generalizability of the Policy Model (RQ$_2$)

To demonstrate the generalizability of our policy model, we consider 5 additional AHOs (`security_state`, `fire`, `water leak`, `illuminance`, `safety_state`). For each AHO, we identified endorsement attributes from the device-attribute map and generated inferences using the process from Section 5.3. Our process resulted in 41 inferences (cumulatively) useful for endorsement, with each AHO being endorsed using *at least* 3 device-attributes (examples in appendix Table 5). This demonstrates the generalizability of our approach, *i.e.*, similar policies are feasible for 5 other AHOs (**RQ$_2$**).

7.3 HomeEndorser Operation under Realistic Home Automation Usage (RQ$_3$)

To test whether HomeEndorser reliably enforces endorsement policies in expected cases, we perform 2 analysis: 1) Evaluating HomeEndorser's operation automatically under realistic event sequences generated by Helion [35], and 2) Executing realistic user behavior scenarios derived from prior work [28] with HomeEndorser enabled.

1. Evaluation with Event Sequences: To test that HomeEndorser performs endorsement according to its policies in regular usage, we used Helion [35] to generate event sequences that are likely to occur next in the home given an initial home event. We provided Helion with 400 randomly chosen starting events and generated 8191 events, consisting of 64 unique devices. We created 51 virtual devices in addition to the 13 in our setup, and automatically tested HomeEndorser's endorsement accuracy by running the event sequences with HomeEndorser enabled, comparing HomeEndorser's decision with the expected behavior based on the device states, and restarting the system in between the execution of successive event sequences.

During the experiment, the effective policy consisted of 6 devices at location 'front door': 4 sensors (motion, presence, beacon, door), and 2 devices (door lock, security panel). To assess accuracy, we assume that the user is `away` and save snapshots of device states at the time of check to automatically compare with the effective policy.

Result: HomeEndorser was invoked in 605 `home` AHO state change requests, correctly allowing in 562 cases and denying in 43. Without HomeEndorser's interception, all 43 cases would have incorrectly allowed the AHO change from `away` to `home`, leading the home to an unsafe state. One question is how many sensors were needed to deny the 43 unsafe AHO change requests correctly. We found that in each denial there were no cases where two or more sensors were triggered at the time of AHO change request, while in 9/43 cases, at least one sensor (either motion or presence) was triggered. The door lock stayed 'locked' in all cases. Hence, either having the door lock or at least 2 sensors in the user's setup can be a viable strategy to enable correct endorsement using HomeEndorser.

2. Evaluating Accuracy with Intentional AHO Changes: To further evaluate HomeEndorser's performance in specific benign cases of intentional AHO changes made by the user (*e.g.*, manually setting `home` using platform's mobile app, automatically setting `home` using a third-party service after the user gets home), we derive a set of 10 *realistic user behavior scenarios* from prior work [28], and enact those scenarios in our apartment testbed. Due to space constraints, we summarize 3/10 exemplary scenarios to discuss HomeEndorser's decisions in response to benign user behavior, with the rest in the online appendix [2]).

Scenario 1 – Unlocking the house, and then leaving: Alice returns home and opens the front door after unlocking the door lock. However, she gets a call from her office and leaves immediately without entering, accidentally also leaving the door open in the process. Regardless, Alice's home/away service accidentally requests the `home`

AHO to change to "home" (*i.e.*, even when Alice has actually left). In response to the request, HomeEndorser gathers the recent states of the devices to check against the policy $P_{\text{home}_2}(\text{home})$.

$$P_{\text{home}_2}(\text{home}) = (\text{door-lock\_lock} == \texttt{UNLOCKED} \land \text{motion\_sensor} == \texttt{ACTIVE} \land \text{door\_sensor} == \texttt{ACTIVE})_{front-door}$$

The policy constraints are *partially* satisfied, as the door lock was unlocked and the door sensor was opened recently. However, as Alice did not enter, the motion sensor did not detect any motion, and the policy results in a *correct denial*, preventing an unsafe situation in which the camera is turned off while the home is vulnerable (i.e., the door is unlocked). Thus, HomeEndorser's composite policy design comprising of multiple devices provides *stronger* endorsement, preventing accidental but unsafe changes.

Scenario 2 – Disarming the security panel and entering: Alice returns home and disarms the home by entering the key-code in the security panel near the door. She then enters the home triggering the motion sensor. At the same time, a home security service requests change to the `security_state` AHO on Alice's behalf, from "deter" to "ok", which if allowed, would disable the security camera, as well as any other security devices (*e.g.*, alarms).

HomeEndorser gathers the recent states of the devices to check against the policy $P_{\texttt{security\_state}_1}(\text{ok})$ (provided previously in Section 7.4). Since the security panel was manually disarmed and the motion sensor was recently active, the policy is satisfied and the state change is correctly allowed.

Scenario 3 – Direct state change request: Alice manually changes the `home` AHO to "home" using the HomeAssistant UI. HomeEndorser identifies that the request was not made through the REST API, and allows it without checking the policy.

To summarize, our evaluation demonstrates that HomeEndorser correctly endorses AHO-changes, and does not cause false denials under benign behavior. In scenario 1, its denials prevents an *accidental* and harmful state change by users (**RQ**$_3$). In some cases, at the time of endorsement check (*i.e.*, time of API call), some devices had reverted to their default states (*e.g.*, motion sensor to "inactive" state). Thus, HomeEndorser's approach of checking the *most recent* state changes rather than only the states at the time of endorsement prevents such potential false denials.

7.4 Preventing Privilege Escalation (RQ$_4$)

An attacker (*e.g.*, Bob)'s goal during privilege escalation is to modify a high-integrity device (*e.g.*, a security camera) that they cannot directly access or compromise by maliciously introducing changes to any AHO that the device depends on. As Bob already has access to modify the AHOs (*e.g.*, via a service he controls, see motivating example in Sec. 2), the access control model without HomeEndorser is unable to prevent Bob from changing the AHO value arbitrarily. However, with HomeEndorser enabled, Bob needs endorsement from the devices associated with the endorsement attributes (see Sec. 5.2), which he is unable to gain, and the attack is prevented.

To demonstrate, we assume the threat described in the motivation (see Sec. 2), where the camera depends on both the `home` and `security_state` AHOs, and experimentally validate HomeEndorser's effectiveness with two attack scenarios.

Malicious Scenario 1 – Bob modifies `home`: We deploy a malicious third-party service controlled by the attacker, Bob. We assume that Alice has granted to the service the permission (*i.e.*, a REST API token) to write to the `home` AHO. When Alice is out of the home, Bob writes to the value "home" to `home`, to disable the camera. Without HomeEndorser, the `home` AHO will change, allowing Bob to remotely disable the security camera; however, we consider that Alice uses HomeEndorser with the policy $P_{\text{home}_1}(\text{home})$:

$$P_{\text{home}_1}(\text{home}) = (\text{door-lock\_lock} == \texttt{UNLOCKED} \land \text{motion\_sensor} == \texttt{ACTIVE})_{front-door}$$

Thus, when Bob writes to `home`, the policy $P_{\text{home}_1}(\text{home})$ is checked as follows: HomeEndorser queries the state machine, and obtains the most recent change to the door lock and the motion detector at the front door. Since the door lock was not unlocked, and the motion detector has not been active recently, the policy returns a DENY decision, preventing the attack. It is also important to note that Bob could attempt to circumvent HomeEndorser's policy by satisfying one of the two conditions in it, *e.g.*, by sliding a thin object (e.g., a card) through the door to trigger the motion sensor; however, the conjunction among device-attributes prevents this variant.

Malicious Scenario 2 – Bob modifies `security_state`: We deploy a malicious third-party service controlled by Bob, to which Alice has granted the permission to write to the `security_state` AHO. Bob will attempt to set the `security_state` to "ok" (as opposed to "deter"), which will trigger a routine that turns off the camera. Just like the prior scenario, without HomeEndorser Bob will succeed; however, Alice uses HomeEndorser with the policy $P_{\texttt{security\_state}_1}(\text{ok})$:

$$P_{\texttt{security\_state}_1}(\text{ok}) = (\text{security-panel} == \texttt{DISARMED} \land \text{motion\_sensor} == \texttt{ACTIVE})_{front-door}$$

When Bob writes to `security_state`, $P_{\texttt{security\_state}_1}(\text{ok})$ is checked. Since the security panel was not disarmed and the motion sensor was not active recently, the policy returns a DENY decision.

Thus, HomeEndorser successfully prevents the AHO modification, which would be allowed by default access control, because the AHO update cannot be endorsed by the states of the correlating devices, as required by the rules above.

7.5 Runtime Performance (RQ$_5$)

We compute microbenchmarks to capture each aspect of the platform that HomeEndorser affects, in particular, the time taken for (1) *policy instantiation* (*i.e.*, delay at boot time), (2) *policy update* during runtime (3) the *endorsement hook invocation* overhead of an API call to a state *not being endorsed*), and, (4) the *endorsement check* overhead of an API call to a state being endorsed. Further, we perform 2 macrobenchmarks to assess HomeEndorser's end-to-end impact on the execution times of remote IoT services that execute an automation using the REST API (5) involving an AHO being endorsed, and (6) involving an AHO not being endorsed. We perform each experiment 50 times, using the largest (worst-case) policy (P_{109}, online appendix [2]), and use vanilla HomeAssistant as a baseline.

Results: Table 2 shows the mean results with 95% confidence intervals. As seen in the table, HomeEndorser has negligible performance overhead for operations that do not involve the AHO being endorsed (*i.e.*, #3 and #6). For endorsed AHOs, HomeEndorser adds only 0.916ms (9.69% overhead) to an AHO-change invoked via an API call (microbenchmark), and adds 2.016ms (12.16% overhead) to the overall execution time of an automation execution that changes

Table 2: Performance overhead of HomeEndorser (in comparison with the unmodified HomeAssistant baseline)

No.	Operation	HomeAssistant Baseline (ms)	HomeEndorser (ms)	Overhead(ms)	Overhead(%)
1.	Policy Instantiation (*Boot up time*)	23.851 ± 1.738	33.669 ± 5.042	9.818	41.16
2.	Policy update during runtime	-	4.350 ± 0.515	-	-
3.	Changing non-endorsed AHO (*Hook invocation cost*)	9.854 ± 0.723	9.916 ± 0.814	0.062	0.63
4.	Changing endorsed AHO (*Endorsement check cost*)	9.451 ± 0.605	10.367 ± 0.482	0.916	9.69
5.	Automation execution with endorsed AHO	16.582 ± 2.388	18.598 ± 0.669	2.016	12.16
6.	Automation execution with non-endorsed AHO	14.609 ± 1.026	14.311 ± 0.477	-0.298	-2.04

Table 3: The (minimal) cost of Integrating HomeEndorser with respect to the properties identified in Section 7.6

	H.Assistant	IoTivity	OpenHAB	SmartThings	NEST	GoogleHome
$Prop_1$	✓	✓	✓	✓*	✓*	✓*
$Prop_2$	✓	✓	✓	✓	✓*	✓*
$Prop_3$	✓	✗	✓	✓	✓*	✓*
$Prop_4$	✓	✓	✓	✓*	✓*	✓*

✓ = Directly portable, ✓* = Directly portable, but needs confirmation from source code, ✗= design-level constraint/extension

an endorsed AHO (macrobenchmark). In fact, the maximum overhead of 9.818ms (41.16%) that HomeEndorser adds is to the overall bootup time of HomeAssistant, which is not that frequent, and not perceivable by the user. After the bootup, the overhead to update policies when devices get added or removed is only 4.350 ms. Finally, we note that the endorsement check overhead is not dependent on the policy size, as HomeAssistant's (and hence HomeEndorser's) state machine obtains device state changes in parallel.

7.6 Effort Required to Integrate and Configure HomeEndorser (RQ_6)

We now describe the effort to deploy and integrate HomeEndorser, from the perspective of experts, platform designers, and end-users.

1. Effort by experts: HomeEndorser's process for generating policy templates is a one-time effort, and templates only need to be updated when new functionality emerges for a device category, or when an entirely new category of device is introduced to the market (*i.e.*, not new brands). The only manual effort involves the identification of the endorsement attributes (as described in Sec. 5.3). It took 2 authors 4 workdays to identify the 10 endorsement attributes for the `home` AHO (as described in Sec. 6).

2. Deployment in the User's Home: As HomeEndorser is integrated with the platform (here, HomeAssistant) and is pre-configured to include all endorsement policy templates, it requires minimal effort from the user. We describe the ease of use in an end-to-end manner as follows: (1) The user connects and configures their devices to the platform as usual (*e.g.*, setting names, location). (2) The user selects an AHO-change they want to endorse. *This is the only additional configuration step introduced by HomeEndorser*. (3) HomeEndorser automatically instantiates location specific policies (see Sec. 5.2) for each AHO without incurring any additional user input. This also occurs automatically on boot, or as devices are added/removed. (4) When HomeEndorser makes an enforcement decision resulting in an AHO-change denial, the user is notified. The user can override this by changing the AHO-state through the native app, which HomeEndorser allows by default. However, we expect this to be rare given HomeEndorser's negligible false positives (see Sec. 7.3).

3. Platform integration: The design of HomeEndorser is independent of any single platform. That is, while our proof of concept is implemented as an enhancement of HomeAssistant, we identify 4 key platform properties that would enable HomeEndorser on any smart home platform. We chose to implement HomeEndorser in HomeAssistant because of its open-source nature and the ease of evaluation it allowed (*e.g.*, creating a virtual device).

Property 1 ($Prop_1$) - Ability to obtain device states: HomeEndorser must be able to obtain states from all devices. Ideally, the platform should have a Platform State Machine that can readily provide recent device state changes (**G4**).

Property 2 ($Prop_2$) - Complete mediation and Tamperproofness: The platform must have a central component that intercepts all the API requests (**G2**), which must be unmodifiable by third parties (**G3**).

Property 3 ($Prop_3$) - Timestamp information of device states: HomeEndorser requires *recent* device state information to prevent any false positives that can occur because of devices reporting cached states or the platform itself reporting the old/last known state because of an unresponsive device (**G4**).

Property 4 ($Prop_4$) - Ability to monitor device-changes: HomeEndorser needs to dynamically adapt its policies based on the current setup of the smart home, and hence, the platform needs to monitor the addition, removal, and change in placement of devices.

Table 3 illustrates how 6 popular smart home platforms exhibit $Prop_1 \rightarrow Prop_4$, and particularly, demonstrates that only IoTivity requires a design-level extension (*i.e.*, a state machine to track freshness) for integrating HomeEndorser (in terms of $Prop_3$), and *all other platforms may feasibly integrate HomeEndorser* with negligible engineering efforts. For instance, both SmartThings and OpenHAB satisfy the 4 design properties necessary to integrate HomeEndorser with minor modifications. Both maintain a variant of the state machine, which enables us to collect all device states at any time, and validate their timestamps (i.e., enabling $Prop_1, Prop_3$ and $Prop_4$). Similarly, both enable centralized mediation of AHOs ($Prop_2$), with SmartThings enabling it immediately, whereas with OpenHab we would simply need to hook into the exposed services/bindings, as prior work has done for Android [21] and Linux [61]. Finally, we mark certain properties for NEST, SmartThings and Google Home as ✓* as those properties are exhibited as per the documentation, but source code would be needed to confirm.

8 THREATS TO VALIDITY

With a systematic, data-driven approach, HomeEndorser lays the groundwork for secure and practical endorsement for AHOs. We now discuss the threats to the validity of its approach:

1. Byzantine Fault Tolerance: We rely on devices to not be compromised and to report correct states, as stated in the threat model in Section 4 (although HomeEndorser does dynamically adapt to devices that may be offline/non-responsive). That said, HomeEndorser's integrity validation of AHOs complements prior efforts [4] that aim to validate device states via fingerprinting, and combining the two approaches is a promising future direction.

2. Completeness and Rigor of Policy Generation: The device-attribute map (see Sec. 5.3), consisting of 510 pairs, is an *evolving dataset* that is *as complete as the information sources* used to derive it (*e.g.*, capability maps), and which can accommodate new device types/attributes with minimal effort. Furthermore, we used systematic open-coding to identify endorsement attributes with negligible disagreements (details in online appendix [2]), illustrating high confidence and minimal risk of incompleteness due to expert error.

3. Multi-user smart homes: Similar to most prior work in the area of smart home API misuse [27, 34, 44, 60], HomeEndorser does not claim to address multi-user scenarios, which are a novel but orthogonal design challenge, which we leave as future work. However, in the online appendix [2], we show that HomeEndorser's endorsement with location-specific predicates may be independent of the number of users, and mitigate the implications of multiple simultaneous device-interactions.

4. Device Availability and Placement: As HomeEndorser automatically chooses the most restrictive policy applicable to a user's home based on device availability/placement (see Sec. 5.2), it can adapt to diverse device combinations. However, we assume optimal device placement/configuration to be out of scope, and direct the reader to complementary work that informs on optimal deployment [28].

9 RELATED WORK

Like HomeEndorser, prior work has also worked on policy design and enforcement, as well as complementary approaches such as anomaly detection or device state validation in the smart home. We now discuss how they differ from HomeEndorser, and how HomeEndorser is ideally suited to solve the problem of AHO integrity.

1. Policy model design: Prior work [8, 49, 62, 63] has explored policy models with various properties. For instance, ExPAT [63] captures user expectations as invariants, PatrIoT [62] supports temporal clauses, while Kratos [49] supports multiuser policies. We design HomeEndorser's own policy model for the following reason: unlike prior work, HomeEndorser is designed exclusively for endorsement of AHOs and does not need to accommodate properties from automations into policy invariants (*e.g.*, temporal). Instead, it only considers device-attributes and their locations, with different locations expressed in mutually-exclusive DNF predicates. As other conditions (*e.g.*, temporal) are decoupled from the policy and are part of enforcement, the policy specification is simpler and allows automated, deployment-aware policy instantiation.

2. Policy enforcement: While prior work [7, 8, 15, 36, 42, 62, 63] has explored policy enforcement, they do not focus on AHO integrity as we discuss in Section 2. Recall that AHOs are platform objects accessible via direct API calls to the platform. However, prior work (*e.g.*, PFirewall [8], Maverick [36]) that operate outside the platform cannot intercept direct cloud-cloud communications between the platforms and the devices, rendering them ineffective. Unlike HomeEndorser, Maverick [36] also incurs significant user effort by requiring users to configure the policies, and add devices through the tool's own interface.

Similarly, prior work has supplemented policy enforcement with static analysis [6, 42], runtime rule-based enforcement [7, 62, 63] or predicting app interactions via physical channels [15] to prevent two or more apps from accidentally (or maliciously) triggering one another to reach an unsafe state (*i.e.*, *app chaining*). However, the scenario in the motivation example manifests as an arbitrary/unauthorized API call to change AHO, not as app chains, so it cannot be prevented by instrumenting apps. Additionally, as they analyze installed IoT apps, they may be incompatible with popular platforms (see Sec. 2), while HomeEndorser is app agnostic and compatible.

3. Centralized AHO Modifications: Schuster et al. [47] propose securing shared states (including AHOs) by centralizing them and allowing only trusted third-parties to modify them using "environmental situation oracles (ESOs)". However, the ESO model aims for privacy, not integrity, in allowing one dedicated trusted app per AHO to compute that AHO's value, which may be hard to scale or maintain and also need to be accepted by competing stakeholders (*e.g.*, users, developers). In contrast, HomeEndorser respects user-choice, and provides endorsement in the presence of untrusted services.

4. Anomaly Detection: HomeEndorser builds upon Biba's notion of trusted guards [3] for endorsement, and is inherently orthogonal to anomaly detection systems like HAWatcher [18]. However, HomeEndorser has some key advantages over HAWatcher. First, HomeEndorser is app agnostic and learns correlations from trusted endorsement attributes. This makes it compatible with most platforms (see Sec. 2), and prevents the risk of *false learning* from malicious apps. Further, HAWatcher trains separately for every home, requires private user data and a day of re-training when configurations change while HomeEndorser automatically instantiates based on device availability/placement information of the user (see Sec. 5.3).

5. Device State Validation: HomeEndorser is complementary to work that validate device states such as Peeves [4]. Peeves generates fingerprints of device events based on physical changes they *cause* that are sensed by other trusted sensors to attest device states. However, unlike HomeEndorser, Peeves focuses on the fingerprint accuracy/precision for *individual* device states rather than on AHOs which are platform objects, while HomeEndorser builds AHO endorsement policies involving trusted device-attributes. As the device states that Peeves validates form the building block of HomeEndorser's endorsement, HomeEndorser will benefit from such complementary approaches, although neither promise byzantine fault tolerance.

10 CONCLUSION

We presented the HomeEndorser framework, which uses localized device state changes to endorse proposed changes to abstract home objects (AHOs) by compromised/malicious services with API access, thereby protecting high integrity devices that rely on the AHO values. HomeEndorser provides a policy model for specifying endorsement policies in terms of device state changes, and a platform reference monitor for endorsing all API requests to change AHOs. We evaluate HomeEndorser on the HomeAssistant platform, finding that we can feasibly derive policy rules for HomeEndorser to endorse changes to 6 AHOs, preventing malice and accidents with feasible performance overhead. Finally, we demonstrate that HomeEndorser is backwards compatible with most popular smart home platforms, and requires modest human effort to configure and deploy.

REFERENCES CITED

[1] Alarm Grid. Accessed Feb 2021. Introducing the "Privacy When Disarmed" Feature for Total Connect 2.0 HD Cameras - Alarm Grid. https://www.alarmgrid.com/blog/introducing-the-privacy-when-disarmed-feature-for-total-connect-.

[2] Anonymous. 2020. HomeEndorser Online Appendix. https://sites.google.com/view/homeendorser.

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

[4] Simon Birnbach, Simon Eberz, and Ivan Martinovic. 2019. Peeves: Physical Event Verification in Smart Homes. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1455–1467.

[5] Ethan Cecchetti, Andrew C. Myers, and Owen Arden. 2017. Nonmalleable Information Flow Control. In *Proceedings of the 2017 ACM Conference on Computer and Communications Security*. 1875–1891.

[6] 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.

[7] Z. Berkay Celik, Gang Tan, and Patrick McDaniel. 2019. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. In *Proceedings of the NDSS 2019 Symposium*.

[8] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Lannan Luo. 2021. PFirewall: Semantics-aware customizable data flow control for smart home privacy protection. *Network and Distributed Systems Security (NDSS) Symposium2021* (2021).

[9] D. D. Clark and D. Wilson. 1987. A Comparison of Military and Commercial Security Policies. In *Proceedings IEEE Symposium on Security and Privacy*.

[10] CNET. Accessed May 2021. Google reverses course on cutting off Works with Nest connections. https://www.cnet.com/home/smart-home/google-reverses-course-on-cutting-off-works-with-nest-connections/.

[11] Camille Cobb, Milijana Surbatovich, Anna Kawakami, Mahmood Sharif, Lujo Bauer, Anupam Das, and Limin Jia. 2020. How Risky Are Real Users' {IFTTT} Applets?. In *Sixteenth Symposium on Usable Privacy and Security ({SOUPS} 2020)*. 505–529.

[12] Digital Trends. Accessed May 2021. Google is ending its Works with Nest system. Here's what that means for you . https://www.digitaltrends.com/home/google-is-ending-its-works-with-nest-system/.

[13] Digitized House. Accessed May 2021. Fallout Mounts from Impending Works with Nest Sunsetting. https://digitized.house/fallout-mounts-works-with-nest-sunsetting/.

[14] Wenbo Ding and Hongxin Hu. 2018. On the Safety of IoT Device Physical Interaction Control. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 832–846.

[15] Wenbo Ding, Hongxin Hu, and Long Cheng. 2021. IOTSAFE: Enforcing Safety and Security Policy with Real IoT Physical Interaction Discovery. In *Proceedings of the ISOC Network and Distributed Systems Security Symposium (NDSS)*.

[16] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazieres, Frans Kaashoek, and Robert Morris. 2005. Labels and event processes in the Asbestos operating system. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, Vol. 39. 17–30.

[17] Timothy Fraser, Lee Badger, and Mark Feldman. 1999. Hardening COTS Software with Generic Software Wrappers. In *Proceedings of the IEEE Symposium on Security and Privacy*. 2–16.

[18] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. 2021. {HAWatcher}:{Semantics-Aware} Anomaly Detection for Appified Smart Homes. In *30th USENIX Security Symposium (USENIX Security 21)*. 4223–4240.

[19] Google. Accessed June 2019. Local Home SDK. https://developers.google.com/actions/smarthome/local-home-sdk.

[20] Google. Accessed May 2021. We hear you: updates to Works with Nest. https://blog.google/products/google-nest/updates-works-with-nest/.

[21] Stephan Heuser, Adwait Nadkarni, William Enck, and Ahmad-Reza Sadeghi. 2014. ASM: A Programmable Interface for Extending Android Security. In *Proceedings of the USENIX Security Symposium*.

[22] How-to Geek. 2021. How to Use Your Nest Thermostat as a Motion Detector. https://www.howtogeek.com/249093/how-to-use-your-nest-thermostat-as-a-motion-detector/.

[23] IFTTT. Accessed Feb 2023. Google Nest is Back on IFTTT. https://ifttt.com/explore/new-google-nest-thermostat-phase1.

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

[25] IFTTT. Accessed May 2021. Important update about Nest services. https://help.ifttt.com/hc/en-us/articles/360022524734-Important-update-about-the-Nest-services.

[26] IoTivity. Accessed June 2019. IoTivity Wiki: IoTivity Initialization and Setting. https://wiki.iotivity.org/initialize_setting.

[27] 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)*.

[28] Arun Cyril Jose and Reza Malekian. 2017. Improving smart home security: Integrating logical sensing into smart home. *IEEE Sensors Journal* 17, 13 (2017), 4269–4286.

[29] Kaushal Kafle, Kevin Moran, Sunil Manandhar, Adwait Nadkarni, and Denys Poshyvanyk. 2019. A Study of Data Store-based Home Automation. In *Proceedings of the 9th ACM Conference on Data and Application Security and Privacy (CODASPY)*.

[30] Kaushal Kafle, Kevin Moran, Sunil Manandhar, Adwait Nadkarni, and Denys Poshyvanyk. 2020. Security in Centralized Data Store-based Home Automation Platforms: A Systematic Analysis of Nest and Hue. *ACM Transactions on Cyber-Physical Systems (TCPS)* 5, 1 (Dec. 2020).

[31] Kami. Accessed May 2021. How to use Home and Away mode in YI Home app . https://help.yitechnology.com/hc/en-us/articles/360041767214-How-to-use-Home-and-Away-mode-in-YI-Home-app.

[32] David H. King, Susmit Jha, Divya Muthukumaran, Trent Jaeger, Somesh Jha, and Sanjit Seshia. 2010. Automating Security Mediation Placement. In *Proceedings of the 19th European Symposium on Programming (ESOP '10)*. 327–344.

[33] 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.

[34] Sanghak Lee, Jiwon Choi, Jihun Kim, Beumjin Cho, Sangho Lee, Hanjun Kim, and Jong Kim. 2017. FACT: Functionality-centric access control system for IoT programming frameworks. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*. 43–54.

[35] Sunil Manandhar, Kevin Moran, Kaushal Kafle, Ruhao Tang, Denys Poshyvanyk, and Adwait Nadkarni. 2020. Towards a Natural Perspective of Smart Homes for Practical Security and Safety Analyses. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (Oakland)*. San Fransisco, CA, USA.

[36] M Hammad Mazhar, Li Li, Endadul Hoque, and Omar Chowdhury. 2023. MAVERICK: An App-independent and Platform-agnostic Approach to Enforce Policies in IoT Systems at Runtime. *arXiv preprint arXiv:2302.01452* (2023).

[37] M. Douglas McIlroy and James A. Reeds. 1992. Multilevel security in the UNIX tradition. *Software: Practice and Experience* (1992).

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

[39] 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)*.

[40] Nest Labs. Accessed Feb 2021. Turn your Nest camera on or off - Google Nest Help. https://bit.ly/33lkfPE.

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

[42] Dang Tu Nguyen, Chengyu Song, Zhiyun Qian, Srikanth V. Krishnamurthy, Edward J. M. Colbert, and Patrick McDaniel. 2018. IotSan: Fortifying the Safety of IoT Systems. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. 191–203.

[43] OCF. 2020. Open Connectivity Foundation - OCF. https://openconnectivityfoundation.github.io/devicemodels/docs/index.html.

[44] Amir Rahmati, Earlence Fernandes, Kevin Eykholt, and Atul Prakash. 2018. Tyche: A Risk-Based Permission Model for Smart Homes. In *2018 IEEE Cybersecurity Development (SecDev)*. 29–36.

[45] Ring. Accessed Feb 2021. Control All your Ring Cameras with Modes - Ring Help. https://support.ring.com/hc/en-us/articles/360036107792-Control-All-your-Ring-Cameras-with-Modes.

[46] Samsung. 2018. Samsung SmartThings SmartApp Public Repository. https://github.com/SmartThingsCommunity/SmartThingsPublic.

[47] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2018. Situational access control in the internet of things. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1056–1073.

[48] Umesh Shankar, Trent Jaeger, and Reiner Sailer. 2006. Toward Automated Information-Flow Integrity Verification for Security-Critical Applications. In *Proceedings of the ISOC Network and Distributed Systems Security Symposium (NDSS)*.

[49] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. 2020. Kratos: Multi-user multi-device-aware access control system for the smart home. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 1–12.

[50] Simplisafe. Accessed Feb 2021. What is the privacy shutter? https://support.simplisafe.com/hc/en-us/articles/360029760591.

[51] SimpliSafe. Accessed May 2021. Everything You Need to Know About Home Mode. https://simplisafe.com/blog/home-mode.

[52] SmartThings. Accessed Aug 2022. The End of Groovy Has Arrived. https://community.smartthings.com/t/the-end-of-groovy-has-arrived/246280.

[53] SmartThings. Accessed Dec 2018. Capabilities Reference. https://docs.smartthings.com/en/latest/capabilities-reference.html.

[54] SmartThings. Accessed June 2019. The SmartThings Ecosystem. https://smartthings.developer.samsung.com/docs/index.html.

[55] Statista. 2020. Smart Home - Worldwide. https://www.statista.com/outlook/279/100/smart-home/worldwide.

[56] 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.

[57] The Ambient. Accessed May 2021. Life360 guide: How to smarten up your home with the geolocation app. https://www.the-ambient.com/guides/life360-complete-guide-804.

[58] 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*.

[59] TP-Link. Accessed May 2021. Does Nest work with TP-Link Kasa. https://www.tp-link.com/us/support/faq/2014/.

[60] 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*.

[61] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, and Greg Kroah-Hartman. 2002. Linux Security Modules: General Security Support for the Linux Kernel. In *Proceedings of the 11th USENIX Security Symposium*.

[62] Moosa Yahyazadeh, Syed Rafiul Hussain, Endadul Hoque, and Omar Chowdhury. 2020. PATRIOT: Policy assisted resilient programmable IoT system. In *Runtime Verification: 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6–9, 2020, Proceedings 20*. Springer, 151–171.

[63] Moosa Yahyazadeh, Proyash Podder, Endadul Hoque, and Omar Chowdhury. 2019. Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms. In *Proceedings of the 24th ACM symposium on access control models and technologies*. 61–72.

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

[65] Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. 2002. Secure program partitioning. *ACM Transactions on Computing Systems* 20, 3 (2002), 283–328.

[66] 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.

Table 4: Real and Virtual Devices in Evaluation

Device	real/virtual	Number
August Door Lock	*real*	1
Blink Camera	*real*	1
Philips Hue Motion+Illuminance+Temp. Sensor	*real*	1
Aotec Door Sensor	*real*	1
Security Panel	*virtual*	1
Presence Sensor	*virtual*	1
Beacon Sensor	*virtual*	1
Thermostat	*virtual*	1
Wemo Switch	*real*	1
Philips Hue Lamp	*real*	1
Google Home	*real*	1

Table 5: AHOs inferred from Endorsement Attributes

AHO	Endorsement attributes
`home`	<security-panel, disarmed>
`home`	<motion-sensor, active>
`fire`	<temperature-sensor, temperature>
`fire`	<smoke-detector, smoke-alarm-state>
`safety_state`	<co-detector,co-alarm-state>
`illuminance`	<blind,openLevel>

A APPENDIX

A.1 Device List for Evaluation

Table 4 shows the real as well as the virtual devices we integrated into HomeAssistant to conduct the experiments. Our choice of devices was limited by compatibility with HomeAssistant, with a bias towards popular brands and device types that would allow us to evaluate HomeEndorser's endorsement.