Correctness and Security for Home Automation

Chandrakana Nandi
Research Advisor: Michael Ernst
Computer Science & Engineering, University of Washington
email: cnandi@cs.washington.edu

1 Problem and Motivation

Home automation is gaining popularity as one of the most notable applications of Internet of Things. Smart homes involve numerous gadgets and sensors which communicate and act autonomously. One major challenge for smart homes is ensuring their correctness and security. What if the controller of one device incorrectly sends commands to another device? What if personal information about the owner is sent to the cloud without permission or proper anonymization? Our goal is to develop a framework to formalize and verify the correctness and security properties for smart homes.

2 Background

Some recent work has discussed possible security loopholes in smart homes [4, 5, 7] and some early solutions have been proposed with limited expressiveness [3] or dynamic enforcements [2]. The problem with dynamic solutions is that if a loophole is encountered during execution, the service has to be stopped which is not convenient. Our approach has greater expressiveness and is static, providing compile time guarantees that the smart home satisfies the correctness and security policies.

3 Approach

To the best of our knowledge, there has not been any work so far on formal reasoning about the correctness and security of home automation systems. In our approach, we formalize a set of policies for the smart home. Our goal is to implement techniques to verify these policies for Google's Android based framework, Brillo [6]. In this work, we propose the formalism. We consider that a smart home has several autonomous smart appliances which can talk to each other. There is no central hub connecting the devices, instead communication happens directly between the devices. The owner of the house can send commands to the devices through his/her smart phone if needed. We assume that messages from the phone to an appliance cannot be spoofed and no one outside the house can listen to the messages. A device executes an action based on the values of sensor variables on which it depends. For example, the front door opens or closes depending on the inputs from a camera outside the house, the location of the owner and the distance of the owner from the door. We want to ensure that no device executes an action unless the conditions are satisfied. We call such a policy a *dependency policy*. Another policy of interest is about controlling rights: what devices are allowed to control a particular device? This policy is called *control policy*. The third type of policy is for ensuring that sensitive data is not leaked to the outside world and is called *information flow policy*.

3.1 Dependency Policy

Let D denote a smart device and A denote an action that it executes, based on the values of the sensors it relies on. There are two possible situations: conditions that *may* trigger the device to act and conditions that *must* trigger the device to act. If D must execute A when some condition C holds, then it is represented as $C \Rightarrow D\ \mathbf{M}\ A$. This means that if C is true, D absolutely must execute A. The other possibility is that D may execute A when some conditions C holds. This is represented as $D\ \mathbf{m}\ A \Rightarrow C$. This means that D may not execute A, but if it does, the condition C must be true. The structure of the dependency policy is:

$$<sensor\_variable := state>^{+} \Rightarrow D\ \mathbf{M}\ A \quad (1)$$

$$D\ \mathbf{m}\ A \Rightarrow <sensor\_variable := state>^{+} \quad (2)$$

3.2 Control policy

An appliance should not be controllable by any arbitrary appliance. However, some appliances may have the permission to send some information to other appliances. For example, a thermostat can control the heater and the cooler and can connect to the owner's phone to find out the location of the owner: if the owner is coming home, it turns on/off the heater or the cooler to regulate the room temperature. The door controller controls the main door depending on the owner's location, but cannot connect to the owner's phone due to lack of support for WiFi. It thus obtains the owner's location from the thermostat using some weaker communication protocol. We want to ensure that the thermostat does not send open/close messages to the door instead of sending the owner's location. To verify this

property, we model the entire home as a finite state machine and check if there is an "illegal" transition label due to unauthorized commands. Figure 1 illustrates this concept. The format of the label is `sender_receiver_command`.

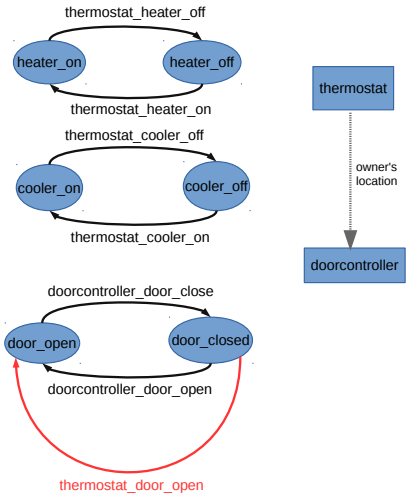

Figure 1: Finite state machine of a smart subsystem: the red arrow is an "illegal" transition label indicating a command from an unauthorized sender

3.3 Information flow policy

We verify the information flow policy using type checking. If d is some data that an appliance gathers from the owner, we annotate it as `sensitive` or `non_sensitive`. Let L_d denote a list of all the destinations to which d may be sent. Then the policy states that if d is not sensitive, then it may be sent to any of the allowed destinations. If it is sensitive, then before sending it anywhere, it should pass through the user's scrutiny. The user may either decide not to send it anywhere or to anonymize it before sending it to some destinations that seem fit. This is represented using an ordered pair, the first element of which describes the action to be performed, and the second element is a list of destinations allowed. The ordered pairs can be of two types,

- `(keep_secret, [])`: if d has to be kept secret, then the list of destinations will be an empty list.
- `(anonymize, [`$D_1, D_2, ..., D_m$`])`: first d is anonymized and then it may be sent to one or more of these m ($\geq$ 1) destinations where $\{D_1, D_2, ..., D_m\} \subseteq L_d$.

To summarize,

- $d.type = sensitive \wedge USER\_DECISION = (keep\_secret, []) \Rightarrow d.destination = []$
- $d.type = sensitive \wedge USER\_DECISION = (anonymize, [D_1, D_2, ..., D_m]) \Rightarrow$ $d.is\_anonymized = true \wedge$ $d.destination \in [D_1, D_2, ..., D_m]$
- $d.type = non\_sensitive \Rightarrow d.destination \in L_d$

4 Contributions

We have used the above policies to formalize a home automation system which has more than 10 appliances. Here we provide some examples for the dependency and information flow policies. Figure 1 in section 3.2 illustrates an example of the control policy.

- Dependency policy: The main door of the house *must* open if it is currently closed and the owner is within 5 feet from the door outside of it and the camera feedback indicates that the person is indeed the owner. Note that the door does not open when there is someone near it inside the house, waiting to go out.
  $(D = \text{“}MAIN\_DOOR\text{”}) \wedge (CURRENT\_STATE = \text{“}closed\text{”}) \wedge (OWNER\_LOCATION = \text{“}outside\text{”} \wedge OWNER\_DISTANCE \leq \text{“}5\ feet\text{”} \wedge CAMERA\_FEEDBACK = 1) \Rightarrow (D = \text{“}MAIN\_DOOR\text{”})\ \mathbf{M}\ (A = \text{“}open\text{”})$

  The Laundry machine *may* start when the machine is full, the doors are closed and the machine is not running already.
  $(D = \text{“}LAUNDRY\_MACHINE\text{”})\ \mathbf{m}\ (A = \text{“}start\text{”}) \Rightarrow (IS\_FULL = true) \wedge (IS\_DOOR\_CLOSED = true) \wedge (CURRENT\_STATE = \text{“}paused\text{”} \vee CURRENT\_STATE = \text{“}off\text{”})$

- Information flow policy: Consider a smart scale [1] which can remember and track the owner's weight. The owner may not want this information to be sent to any website by the appliance. Hence the *type* for the weight would be `sensitive` and the user's decision will most likely be `(keep_secret, [])`.

5 Conclusions and future work

We proposed a technique to define and formalize correctness, security and privacy for smart homes. The next step is to implement a framework that allows us to verify these policies.

References

[1] *Smart Weigh*, 2015 (accessed November 29, 2015). `http://www.smartweighscales.com/`.

[2] Jalal Al-Muhtadi, Manish Anand, M Dennis Mickunas, and Roy Campbell. Secure smart homes using jini and uiuc sesame. In *Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference*, pages 77–85. IEEE, 2000.

[3] Mario Ballano Barcena and Candid Wueest. Insecurity in the internet of things. 2015.

[4] Tamara Denning, Tadayoshi Kohno, and Henry M. Levy. Computer security and the modern home. *Commun. ACM*, 56(1):94–103, January 2013.

[5] Nitsh Dhanjani. Abusing the internet of things: Blackouts, freakouts and stakeouts.

[6] Google. *Brillo*, 2015 (accessed November 28, 2015). `https://developers.google.com/brillo/?hl=en`.

[7] Blase Ur, Jaeyeon Jung, and Stuart Schechter. The current state of access control for smart devices in homes. In *Workshop on Home Usable Privacy and Security (HUPS)*. HUPS 2014, July 2013.