TAESim: A Testbed for IoT Security Analysis of Trigger-action Environment

Xinbo Ban[1,2][0000−0002−3847−7483], Ming Ding[3][0000−0002−3690−0321], Shigang Liu[1][0000−0002−3122−6745], Chao Chen[4][0000−0003−1355−3870], Jun Zhang[1][0000−0002−2189−7801], and Yang Xiang[1][0000−0001−5252−0831]

[1] Swinburne University of Technology, Melbourne 3122, Australia
{XBan, ShigangLiu, JunZhang, YXiang}@swin.edu.au
[2] Data61, CSIRO, Sydney 2015, Australia
Xinbo.Ban@data61.csiro.au
[3] Information Security and Privacy Group, Data61, CSIRO, Sydney 2015, Australia
Ming.Ding@data61.csiro.au
[4] James Cook University, Townsville 4811, Australia
Chao.Chen@jcu.edu.au

Abstract. The Internet of Things (IoT) networks promote significant convenience in every aspect of our life, including smart vehicles, smart cities, smart homes, etc. With the advancement of IoT technologies, the IoT platforms bring many new features to the IoT devices so that these devices can not only passively monitor the environment (e.g. conventional sensors), but also interact with the physical surroundings (e.g. actuators). In this light, new problems of safety and security arise due to the new features. For instance, the unexpected and undesirable physical interactions might occur among devices, which is known as inter-rule vulnerability. A few work have investigated the inter-rule vulnerability from both cyberspace and physical channels. Unfortunately, only few research papers take advantage of run-time simulation techniques to properly model trigger action environments. Moreover, no simulation platform is capable of modeling primary physical channels and studies the impacts of physical interactions on IoT safety and security. In this paper, we introduce TAESim, a simulation testbed to support reusable simulations in the research of IoT safety and security, especially for the IoT activities in home automation that could involve possibly unexpected interactions. TAESim operates over MATLAB/Simulink and constructs a digital twin for modeling the nature of the trigger-action environment using simulations. It is an open-access platform and can be used by the research community, government, and industry who work toward preventing the safety and security consequences in the IoT ecosystem. In order to evaluate the effectiveness and efficiency of the testbed, we conduct some experiments and the results show that the simulations are completed in a few seconds. We also present two case studies that can report unexpected consequences.

Keywords: IoT Security · SmartThings · Inter-rule vulnerability · Simulation.

1 Introduction

Internet of Things (IoT) greatly revolutionizes home automation due to the exponential growth of IoT devices. It is expected to have over 50 billion IoT devices connected to the Internet by the end of 2020 [20]. Although the IoT technologies can offer a lot of convenience, new concerns have been raised about the safety and security of the smart home environment [2][7]. For instance, Mirai malware launched a large-scale Distributed Denial of Service (DDoS) attack through controlling over 600,000 vulnerable IoT devices [31]. The adversary can break into the home network by exploiting the flaw in firmware [44]. More specifically, a worm could self-replicate and spread throughout ZigBee among smart bulbs [42]. Moreover, design flaws have been recently found in the SmartThings platform and vulnerable third-party applications could compromise the platform [22]. Some work investigate the possibility of launching attacks by leveraging the physical capabilities of IoT devices. For example, a smart bulb could eavesdrop the sensitive traffic and expose it by flashing the light stealthily [41].

The recent research has improved the IoT safety and security by working at addressing the traditional issues of IoT security including design flaws [22] [24] [29] [50], malware [22] [44], protocol vulnerabilities [27] [33] [40] and firmware vulnerabilities [26] [44]. Different from these work, we focus on a new type of safety and security issue caused by the inter-rule vulnerability. Due to the increasing complexity of smart home configuration, IoT apps are co-employed in an environment and they can interact with each other via a common device. Besides, some IoT devices can not only communicate via network but also have the functionalities of sensing and affecting the physical environment. These interactions may lead to undesired and unexpected consequences. The attack can be launched to leave the user in a risky state. For example, the door is unlocked when there is no person at home or the heater is turned off to create the 'unpleasant' state when it is winter. In order to alleviate the security problem caused by the interactions of IoT devices and apps, some research recently shed light on the discovery of the risky interactions [10][17][8][36][14].

According to [19], there are two types of the interactions given an IoT environment with IoT apps and devices co-employed:

- *Cyberspace interaction.* The network enables the interaction of apps via the channels in cyberspace such as time, and home mode. For instance, given two apps, a light is turned on when the sunsets and a door is unlocked when this light is on [10]. The event *light.on* is shared in the same device in cyberspace. The term 'cyberspace interaction' represents IoT app interaction when IoT apps operate on the same device.
- *Physical interaction* IoT has a unique feature that devices can interact with each other via physical channels such as temperature, illuminance, and humidity [17][4]. For example, an app turns on a heater and another app opens a window when the temperature is higher than a threshold. The heater and the temperature sensor are connected through a temperature channel then a physical interaction is generated.

Cyberspace interactions consist of one or multiple IoT apps and they can leave users in an unexpected state. Some research such as Soteria [8] and IoTSan [36] utilized a collection of safety policies to assess the safety and security of an IoT ecosystem. More specifically, they discovered the cyberspace interactions that violate the designed safety policies through model checking. For example, conflicts usually happen when several IoT apps control a common IoT device.

Physical interactions can also lead to insecure situations, in which the adversaries can exploit the vulnerability. For instance, an app can control the window when the temperature rises above a threshold and it exposes a potential break-in vulnerability if a burglar manipulates the temperature. IoTMon [17] leveraged static analysis techniques to discover all potentially vulnerable physical interactions. Differently, IoTGuard [10] and IoTSafe [19] are dynamic solutions to enforce the safety policy at run-time. IoTGuard mainly focuses on cyberspace interactions in an IoT ecosystem and IoTSafe aims to capture real physical interactions.

Different from cyberspace interaction, physical interaction faces more challenges for analysis. Firstly, static analysis techniques poorly explore the possible paths for physical interaction since it highly depends on the real-world environment. For example, a program executed on a computer has the same behavior wherever the computer is. However, an IoT app operates variously if the physical channels differ. Secondly, dynamic analysis techniques rely on the development of program simulation. For instance, the fuzzing technique runs the program and mutates the input cases until a crash occurs. However, applying similar techniques for IoT apps cannot resolve this situation because physical interactions affect the operation of IoT apps in a different environment. IoTSafe successfully modeled the physical channels depending on the employment of real devices and sensors. However, the input cases have poor scalability, which means that they cannot represent diverse scenarios. Moreover, recent work depend on the SmartThings simulator, which requires the instrumentation in early-stage and limits the variety of IoT devices. In order to fill this gap, we propose a testbed to simulate the vast number of possible cyberspace and physical interactions among multiple IoT devices and apps.

In this paper, we present a proof of concept of a simulation testbed, TAESim, for IoT security of trigger-action platform, which is not included in previous studies. Our method addresses the main challenges of the IoT trigger action security analysis and makes the IoT environment simulation possible. By taking advantage of MATLAB/Simulink, we implement a testbed with the capacity for expansion, and it can properly model the behavior of the channels and devices. In the proposed testbed, multiple IoT apps can be executed simultaneously, and joint behavior on channels from multiple devices can be represented as well. We implement several devices, two cyberspace channels (i.e., time and home state) and seven physical channels (i.e., temperature, humidity, smoke, motion, illumination, ultraviolet, and water). It is worth noting that more devices and channels can be added to the simulation testbed. Moreover, the testbed is allowed to randomly adopt unexpected factors such as human interaction, sudden

shutdown, etc. Furthermore, the proposed testbed supports several research directions. For example, simulating the IoT system before installing devices and apps at home, or creating the corresponding digital twin to predict future behavior for the inter-rule vulnerability. We also verify the effectiveness and efficiency of the testbed. The simulation results demonstrate that our testbed can properly model the interactions between devices and the joint effects on the physical channels. Although some other testbeds have been proposed in previous work, there is widely adopted testbed for researching safety and security of trigger-action environemnt. Han et al [28] proposed a simulation toolkit, DPWSim, for supporting the development of IoT application that used Devices Profile for Web Services. Lee et al. [32] proposed CyPhySim that leveraged the state machine, continuous-time solver, and discrete-event simulation engine to simulate an cyber-physical system. FIT loT-LAB, presented by Adjih et al. [1] composed thousands of wireless nodes to accelerate the IoT development. Comparatively, our proposed testbed investigates the practical interaction modeling and has substantial scalability and superior performance.

The rest of the paper is organized as follows. Section 2 presents the related work and motivation of this paper. In order to establish this testbed for IoT security research, we discuss the main challenges in modeling the IoT trigger-action environment and assessing its safety and security. Section 4 introduces the details of the components in the implementation of the TAESim. We evaluate the efficiency of the testbed and present the representative case studies in Section 5. Finally, Section 6 concludes this paper.

2 Related Work and Motivation

This paper aims at modeling the trigger-action environment that can be helpful at steps in the assessment of security, safety, and privacy of IoT automation systems. We first reviewed the recent work and present the motivation of our proposed testbed.

2.1 Related work

Previous work have proved that static techniques can identify and improve IoT safety and security. Without executing programs, it provides scalability especially when the large-scale study is performed. Fernandes et al. [22] analyzed the SmartApps obtained from the official store in 2016 and identified that over 55% of them were vulnerable to the over-privileged attacks. Besides, they reported that no sufficient protection was provided from SmartThings for the sensitive data, which leaded to exploitable vulnerabilities such as event spoofing and leakage. SAINT [7] detected the sensitive data flow by tracking the sensitive sources to the external sinks in the information flows. SOTERIA [8] leveraged model checking to discover the violations based on the user-defined security and safety properties. Similarly, IotSan [36] verified the security and safety properties using model checking especially focusing on the interactions between devices

and apps. However, both SOTERIA and IotSan only consider cyberspace interactions in the proposed approaches. IoTMon [17] first discovered all potential physical interaction chains from the IoT apps and reported security and safety risks. Nevertheless, it cannot find the violations for run-time policy violations in real-world IoT deployments as well.

On the other hand, the results from the static analysis on IoT apps can provide rich information to guide run-time enforcement. SmartAuth [48] used the static analysis model obtained from the descriptions of IoT apps to keep the run-time behaviors of IoT apps consistent. It alleviated the security threats of over-privileged IoT apps. FlowFence [23] addressed the data leakage and permission abuse issues through leveraging the information flow resulting in blocking undefined ones. HoMonit [51] analyzed the source code of IoT apps and defined a normal traffic behavior model to detect the risky behaviors at run-time. IoTGuard [10] enforced the policies in multi-app environments to detect the violations by means of chaining rules and analyzing their reachability. IoTSafe [19] practically inspected physical interactions in a IoT environment and dynamically assessed the safety and security of it.

2.2 Motivation

An IoT device can not only be triggered by the cyberspace event and the physical channels but also exert influence on the physical environment (e.g. temperature, humidity, brightness). The attacker can exploit an IoT environment via an insecure and unsafe interaction leading the users to be in an unexpected state [17]. For example, if a robot vacuum is tampered with, the window could be opened via the physical motion interaction. Given three apps, a home mode app, a window app, and a robot app, a potentially unsafe interaction might exist in this smart home. The first app assigns the home mode 'Occupied' when the motion sensor detects a movement.

The window app controls the window to be opened if the temperature rises above a threshold and the home mode is 'Occupied'. The robot app sets a timer to trigger a vacuum operation. In this example, the temperature near the thermometer sensor could be raised above 85F to trigger a window opening action, which may leave home in a potentially unsafe situation, such as burglar break-in.

This type of vulnerability is called inter-rule vulnerability and it is very difficult to be identified by a manual process. Different from the software and hardware vulnerability, inter-rule vulnerability potentially exists in the interactions between devices. It is an unexpected consequence after the devices interact with each other. On the other hand, it is similar to the traditional vulnerability because it directly leaves the user in an unsafe state or can be exploited by adversaries. There are a few factors that might lead to inter-rule vulnerability including malicious apps, broken devices, user's vulnerable configurations, etc. Meanwhile, many research work aim at eliminating the real-world risks through dynamically discovering the vulnerabilities in run-time before the users set up the devices in an IoT ecosystem. The authors deploy the apps on the SmartThings simulator to capture the run-time information including the device status and

user's configurations but there are some limitations. Firstly, although it provides a collection of the devices for selection, new products are often not available in the simulator. Secondly, SmartThings simulator is running on the online server maintained by Samsung. Previous work leveraged a collection of SmartThings commands from its documentation for information collection in run-time. So, it is necessary to instrument the target apps before dynamic analysis starts. Moreover, if the SmartThing server does not allow the data exchange for security consideration, it will be impossible to utilize the SmartThing simulator for security analysis. So, we are motivated to propose a dynamic analysis simulator that is capable of modeling IoT environments including channels, devices, and apps.

3 Challenges in Testbed Simulation

Compared to traditional computing platform, IoT reveals several unique functional characteristics while it poses unusual challenges in terms of code analysis for security. In order to properly model the IoT environment and propose the testbed, we focus on the trigger action environment and discuss the challenges that a simulation testbed faces. In this section, we present five challenges from different aspects, including physical channels modeling, IoT apps modeling, automated test-case generation, multiple apps analysis, and interactions between IoT devices and apps.

Physical channels modeling: A vulnerability can potentially lead the program to crash, thus the system is at risk. IoT devices that execute the program in firmware are in the same danger as well. Differently, the physical channels are interacted into cyberspace connectivity by IoT devices. It can achieve unexpected consequences that the IoT apps deviate from the device functionality caused by the misuse of physical channels. For example, the temperature can be increased through maliciously turning on a heater by an adversary. Once it exceeds a threshold, the window will be opened. The heater-temperature-window interaction leads the room to be insecure and unsafe. Therefore, a burglar can break into the house by controlling the indoor temperature.

Besides, the physical channels and the joint influence of physical channels are different from the consequence of a single device when multiple IoT devices and apps operate together. For instance, the temperature is rising quickly when a heater and an AC operate together. So the safety and security of apps of trigger action platforms not only affect the stability of the program but also raise the concerns on the physical environment.

IoT apps modeling: Most of the IoT devices usually constitute a complex system, and it is hard to conduct a security assessment on them. In other words, these systems cannot be executed and analyzed directly in a short time, which requires appropriate simulation to accurately execute and analyze these kinds of IoT systems. Importantly, the state and computational logic among these devices should be able to be gathered during the simulation process on the heterogeneous IoT system [30]. In addition, it is worth noting that simulating the physical

channels is difficult, including temperature, humidity, illuminance, etc. Similar to cyber-physical system simulation, it must involve the evolution of IoT system state over time. So those requirements prompt to develop a simulator that can execute the IoT apps by means of a discrete-event simulation engine through continuous-time solvers and state machine-based modeling [32]. Many research explored the demands of IoT system modeling and simulation [1][16][28][32]. For example, IoTify provides the virtual device simulation on the cloud for IoT app development [30]. However, existing simulators mainly focus on the development of IoT device functionalities and often adopt the SmartThings web-based IoT simulator. These simulators insufficiently support the diverse IoT devices and apps, which limits the various functionality simulation for IoT apps.

Automated test-case generation: A requirement of the dynamic analysis deployment is the input data for program execution. Generally, inputs are the entry points of a program. For IoT apps, the event triggers of IoT apps can be considered as the inputs. Since input generation needs to be scalable, systematic, and automated, it introduces the difficulties of input generation for IoT apps, which manage multiple devices with different states. For instance, the thermostat has an integer value attribute that introduces a large space for input generation and a large number of test cases.

Fuzzing and symbolic execution are usually utilized for input generation and code coverage increase. Fuzzing feeds the randomly generated inputs to an executed app, while symbolic execution explores the paths using symbolic inputs [5]. For example, IoTFuzzer [11] identified contents of IoT apps through dynamic analysis and discovered the memory corrupted vulnerabilities based on the mutation. Meanwhile, many work leveraged heuristics that intelligently explored the code paths via input generation guidance to avoid redundants [6] [15] [34] [39] [49]. Yet, to our knowledge, tools that automate test input data and event generation to execute IoT apps are non-existent. This motivates us to improve test-case generation techniques as applied to IoT in the future.

Multiple apps analysis: Individual app analysis always focuses on the single app in isolation while multiple apps analysis investigates the joint behavior of several apps. In an IoT environment, apps can interact via the devices or events in two ways: (1) when a device attribute is changed by an event handler and this behavior triggers another event of a device. For instance, when the smoke is detected, a light is turned, then the window is closed because the light is turned on; (2) multiple apps operate on the same device. For example, the water valve is closed when the leak is detected meanwhile it should be opened when the sprinkler is activated by a smoke detector;

Although all individual apps are verified that each of them is secure and safe, the interactions still can cause security and safety issues [9] [13] [18] [37]. To avoid the unexpected consequence through interactions, identifying the interactions is essential for securing the IoT environment with multiple apps employed. It motivates us to develop the dynamic approach for checking that the IoT apps conform to safety properties when interacting with each other.

Interactions between IoT devices and apps: The services from trigger-action platforms can be connected and employed simultaneously, including IFTTT, Zapier, and Apiant. This platform provides a collection of APIs that allow users to authorize services. For instance, a user with a SmartThings IoT platform account can authorize the SmartThings service through the OAuth protocol to communicate with their SmartThings account. REST APIs support the service communication based on HTTP protocol [25]. So users are capable to create their personally customized automation through using rules, which connect the trigger event and the action event. When the event happens as the trigger in service, the action of the rule is automatically operated in another service.

The interactions between IoT apps from different trigger action platform can make the IoT environment insecure and unsafe [3] [10] [46]. The analysis of interactions between IoT apps requires Natural Language Processing for the key information extraction. Specifically, rules called IoT apps, automated the device behavior via either cyberspace interaction or physical interaction. So, it is necessary to figure out the types of devices, channels, and events in the rules. No matter what platforms the IoT system uses, determining the key information from the description of rules can be accomplished using advanced natural language processing techniques.

4 TAPSim: A Simulation Testbed

4.1 Overview

To model the trigger-action environment and address the challenges for the IoT scenario simulation, we propose TAPSim to simulate the behaviors of the IoT devices, apps, and interaction channels. We use MATLAB/Simulink as the simulation engine because it fits the requirements of the proper discrete-event simulation.

MATLAB is a programming and numeric computing platform for data analysis, algorithms development, and model creation. It integrates Simulink that is a block diagram environment for multi-domain simulation and Model-Based Design, enabling the algorithms incorporation and result analysis. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling/simulating dynamic systems. We find that MATLAB/Simulink is proper for the simulation of an interactive IoT system. We create the complex digital twin of smart home through system componentization and reuse components throughout the model with subsystems and model references. The detailed components will be discussed in the following sections.

4.2 Devices

In this section, we present the way to model the IoT devices in this testbed. Simulink provides a block, Data Store Memory, to store a global variable during the simulation. It meets the demands of the modeling status of the device (e.g.

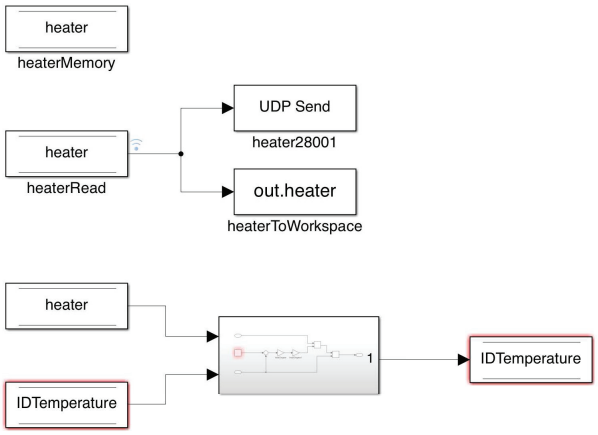

Fig. 1: An IoT device (heater) modeled in Simulink

on and off). For example, in Figure 1, we create a Data Store Memory block for storing the device state.

Initially, the default value is set up with 0 and can be randomly specified for test case generation. In the properties page of each Data Memory block, it allows to visually input a default value from the Signal Attributes tab. Value 0 or 1 for each Data Store Memory block represents on or off for status of the relevant device. We also create a value output module for saving and sending the device state with the time series. It consists of three blocks: Data Store Read, UDP Send, and To Workplace.

Data Store Read block outputs the value of the corresponding device state at every sampling time. We connect this block to the other two blocks for data saving and sending. The upper one, the UDP Sender block could send its received value to a particular IP address based on UDP communication for data storage. We create this block for further research if users need to communicate with the analysis engine. Besides, the To Workspace block saves the input signal to a workspace during simulation. When the simulation is paused or completed, the data written in the workspace can be retrieved or viewed visually.

Our testbed considers smart plugs as the particular smart devices that plug connect to. For instance, a plug is viewed as a bulb when a smart plug connect to a bulb The plug is modeled as an integral part of its connected device because these kinds of devices have limited functions like turning on and off.

A device can affect one or multiple physical channels. So we need to identify the physical channels and interactions between them. Many research discovered the potential physical interactions among IoT devices [9][14][18][37]. They statically analyzed the IoT apps to construct a Dependency Graph and discover the possible physical channels and corresponding interactions. The physical interactions between devices are context-sensitive in a real-world IoT system. To capture the real physical channels, Ding et al. [19] dynamically identified the real and context-sensitive physical interactions using the practical devices. The results are shown in Table 1.

In this study, we model the physical interactions between the heater channel and the temperature channel. We adopted the heater simulation from Simulink [43] and the model of the heating influence can be described as follows:

$$\frac{dQ}{dt} = (T_{heater} - T_{room}) \cdot Mdot \cdot c \tag{1}$$

where $\frac{dQ}{dt}$ represents heat flow from the heater into the room. c is the heat capacity of air at constant pressure. $Mdot$ is the air mass flow rate through the heater (kg/min). T_{heater} is the temperature of hot air from the heater and T_{room} is the indoor temperature at the same time. These parameters are pre-defined before the simulation starts. For different devices, adjusting relevant parameters deals with the various situations. If the device cannot be easily modeled based on this equation, it allows the function to directly change the value of the physical channel by a minute. Once the state of the heater is on, the room gains heat and temperature changes over time.

Table 1: Summary of interactions of 16 IoT devices. ✓represents the physical interaction is identified which means that the IoT device has the influence on the physical channel.

Device	Temperature	Humidity	Smoke	Motion	Illuminance	Ultraviolet	Water
AC	✓	✓					
Heater	✓	✓					
Vent	✓	✓					
Fan	✓						
Window	✓	✓		✓		✓	
Radiator	✓	✓					
Humidifier		✓					
Coffee Machine		✓					
Robot				✓			
Stove	✓	✓					
PC	✓						
TV					✓		
Air Fryer	✓		✓				
Light					✓		
Shade	✓				✓	✓	
Valve							✓

4.3 Channels

In an IoT environment, the automation is achieved by producing interactions via channels. More specifically, the devices can communicate and act under a certain condition via network, such as Wi-Fi, Zigbee. Moreover, the physical environment can enable the device to activate the automation including temperature, illuminance, etc. Thus, there are two kinds of channels including cyberspace channels and physical channels. In this section, we introduce how to model the both types of channels.

Cyberspace channels: In order to model the interaction in cyberspace, we create a time channel measured in minutes. The IoT apps are always executed with one or two minutes' delay [35]. In the TAESim, we assume that channels and devices update their states every minute, since the time in simulation is different from it in the real world. To simulate a dynamic system, we compute

its states at successive time steps over a specified time span. Time steps are time intervals when the computation happens. The size of this time interval is called step size. The process of computing the states of a model in this manner is known as solving the model. Besides, we need a solver that applies a numerical method to solve the set of ordinary differential equations that represent the model. Through this computation, it determines the time of the next simulation step. In the process of solving this initial value problem, the solver also can satisfy the accuracy requirements. We use the Fixed-step solver and set the time step as one, which means that the stop time represents the minutes that the simulation executes.

Time is an important factor to simulate the smart home environment. In order to fit several IoT apps that require time as the condition, we create a time channel through a time block in Simulink. Moreover, we use a Function block to convert minutes to hours and days for data analysis. Similar to modeling devices, the Data Store Memory block stores the value and can be outputted to trigger devices. Importantly, none of the devices can change the time channel.

Besides, many IoT apps complete the automation tasks based on a specified condition. For example, an official SmartThing app usually uses the scheduled-mode-change.groovy to change mode at a specific time of day. To properly capture the cyberspace interactions, we create another channel that is the home mode. According to the practical usage of smart home, we design that there are three home modes including Home, Occupied, and Sleep. The home mode can be a condition in an IoT app and changing of home mode can be either trigger or action. The home mode channel is editable to the IoT apps (Home is 0; Sleep is 1; Occupied is 2). Similarly, it also can be read by a Data Store Read block.

Physical channels: Simulating the physical channels at run-time can properly capture the interactions between devices. The physical modeling process is often difficult to replicate because many complex factors are necessary to be considered such as the house geometry, materials of the house, outdoor weather [12] [21][38][45][47]. These factors collaboratively influence physical channels including temperature, humidity, illuminance, smoke and so on. To address the physical channel modeling challenges, we first identify the physical channels and interactions in an IoT system. Table 1 shows the summary of implicit and explicit physical interactions [19]. In order to simulate both implicit and explicit interactions, we use the Function and Subsystem block in Simulink's library to model the states and changes of physical channels.

We create seven physical channels in this simulation testbed including temperature, humidity, smoke, motion, illuminance, ultraviolet, and water. For each channel, a Data Store Memory block stores the value that represents the corresponding unit on the specific scale. For example, the Data Store Memory of indoor temperature channel stores the degree Celsius that measures the temperature on the Celsius scale. It can be affected by many factors such as the outdoor temperature, heater, fan, etc. We provide a function that defines the daily change of the temperature based on the changes in outdoor temperature. The way device affects the temperature channel is similar.

The model process is adapted from the example officially provided by Simulink [43]. In this model, we present how to model the indoor heat losses and then give detailed parameters.

$$\left(\frac{dQ}{dt}\right)_{losses} = \frac{T_{in} - T_{out}}{R_{eq}} \tag{2}$$

$$\frac{dT_{room}}{dt} = \frac{1}{M_{air} \cdot c} \cdot \left(-\frac{fQ_{losses}}{dt}\right) \tag{3}$$

where $\left(\frac{dQ}{dt}\right)_{losses}$ is the heat loss in the room. T_{in} and T_{out} are the temperature for indoor and outdoor, respectively. Req is the equivalent thermal resistance of the room, which can be calculated by pre-defined parameters including room geometry (size of room; size and number of window) and thermal properties and resistance of the room. $\frac{dT_{room}}{dt}$ is the temperature time derivative. M_{air} represents the mass of air indoor and c is the heat capacity of air at constant pressure. To simulate the environment, we use the default values as the initial set for a few characters of room and outdoor temperature.

The humidifiers are set to vary within the range 0% to 100% since most humidifiers sense and report the relative humidity. The relative humidity is the proportion of water vapor in the air relative to the maximum water vapor that can be held in the air at a given temperature, and thus a temperature-dependent measure. The parameter for modeling humidity is different from the temperature changes. We leave the humidity channel modeling for future work because it is hard to model the humidity. To simplify the problem, if a device has either an implicit or explicit effect on humidity, a function can rise the value to the Data Store Memory of humidity.

In addition, the other physical channels have been set to the default value in advance and we leave the modeling process for future work. To quickly set up a simulation in a very basic configuration, the default value is specified before the simulation starts to fit the modeling requirements.

4.4 Apps

Home automation rules, called IoT apps, are the core of the smart home to automatically trigger the devices to act. Generally, the IoT apps have three elements: trigger event, condition, and action event. The trigger event is either a specific action of a device such as turning on/off or reaching a threshold in a physical channel such as temperature. The condition is optional in IoT apps and can be either from cyberspace or physical space. Many IoT apps have only trigger events and action events to compose automation. SmartThings supports multiple conditions while IFTTT allows users to specify one condition for one applet. The action event is the capability of the device. Importantly, in an IoT ecosystem, an action event is possibly another app's trigger event composing a rule chain. Thus it is difficult to figure out the real rule chain by statically

analyzing the IoT apps. With the help of Simulink, we attempt to model the behaviors of an IoT app using several blocks from the native library.

Figure 2 shows an app named 'Turn off light if motion detected'. It is an official SmartApp from the SmartThings community written in the Groovy programming language. The testbed needs the description of every IoT app because the necessary elements for setting simulation are trigger and action events. The description of this app is 'Turns off a device if there is motion' and it indicates at least two devices exist in this IoT ecosystem: a user-specified device and a motion sensor. In Figure 2, an area contains all blocks and the apps' name is shown on the top. The block named 'motionSensor' is a 'Data Store Read' block, that outputs the state of the motion sensor to an 'if' block. There are basically two ways that could happen. Firstly, if the motion sensor is activated by the movement, the signal from the 'Data Store Read' block is '1' and the 'if' block is executed, which means the 'constant' block sends the signal '0' to the device block. Otherwise, it reaches the 'terminator' block which is used to terminate output signals. Since we create the single model file for reuse, none of the 'Data Store Memory' blocks is added in this file and 'Data Store Read' blocks are missing in the corresponding data store. It leads such 'Data Store Read' blocks to be highlighted as warnings. Once the app is integrated into a complete digital twin of the IoT ecosystem in the testbed with all relevant 'Data Store Memory' blocks, the warnings disappear. In this example, we model the light as the simulated device. Finally, the Data Store Write receive the constant and refresh the state of light.

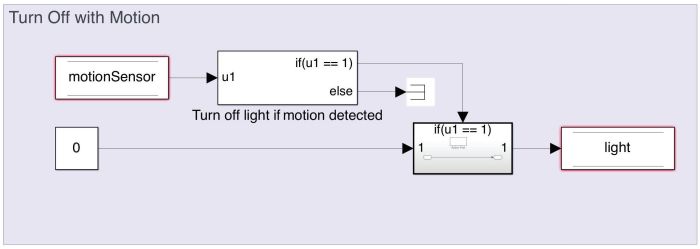

Fig. 2: An SmartApps 'Turn off light if motion detected' from official SmartThings repository

4.5 Unexpected factors

In order to explore all possible situations and discover potential risky interactions in the real world, we deem the unexpected factors as those events that occasionally occur and lead to an unknown consequence. There are two factors we adopt in the testbed: Human interaction and broken devices.

Human interactions: We mainly consider interactions between devices and direct users' interactions in the environment. It needs the dynamic analysis, which may be disturbed by human activities like moving. This actively demonstrates that human activities results in a false-positive interaction in the analysis.

The testbed is expected to randomly imitate human behaviors, that may affect the device automation. We consider this as future work.

Broken devices: Previous research investigates the inter-rule vulnerability under a perfect situation that all devices work appropriately. However, a part or whole home may have no power. During the simulation, the device state changes under the situation of apps usage. We design a random modular to turn off some devices to simulate the broken device.

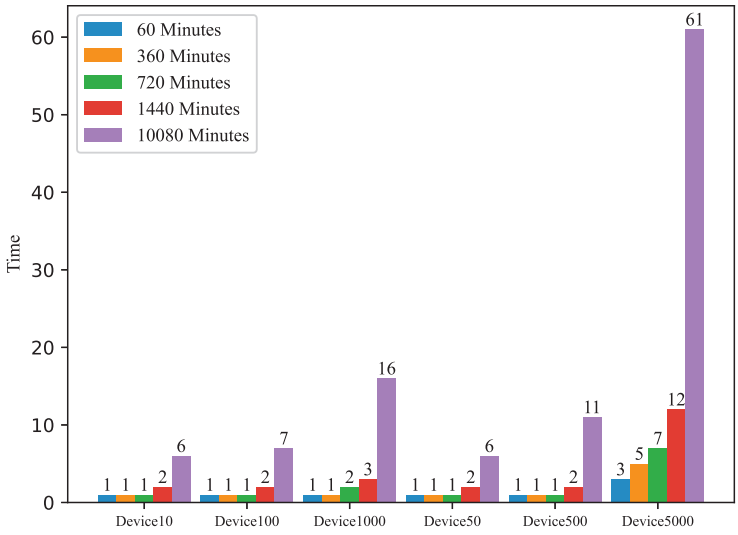

Fig. 3: The overhead of testbed on large scale simulation.

5 Evaluation and Case Study

5.1 Evaluation

For evaluating the overhead of the testbed simulation, we tested the average simulation time of different Stop Time settings in groups of devices. The results are output from the Simulation Manager in Simulink. We performed the experiments on a desktop computer with a 2.1Ghz 2-core Intel Xeon Silver processor and 64GB RAM, using MATLAB 2018b version with one active worker. We ran each test 10 times in each group and reported the average result. For each group, the marked number of devices were modeled in a single file and we simulated with different Stop Time settings including 60, 360, 720, 1440, 10080. The Stop Time is viewed as the minutes in the simulation. So we designed the simulation with a large number of devices that continually ran from minutes to days. For example, the simulation with 5000 devices and 10080 minutes represent that devices are running for 7 days. We selected 10 modeled devices and repeatedly add them into the simulation. As is shown in Figure 3, the time consumption of Stop Time 60, 360, 720 for the four groups (Device 10, Device100, Device50, Device500) is 1 second in real-world and the time consumption of Stop Time

1440 for them is 2 seconds. For simulation of 10080 minutes, four groups (Device 10, Device100, Device50, Device500) cost 6 to 11 seconds, which is roughly 5 times the result of 1440 minutes. The last group with 5000 devices consume 3, 5, 7, 12, 61 seconds for each Stop Time setting, respectively.

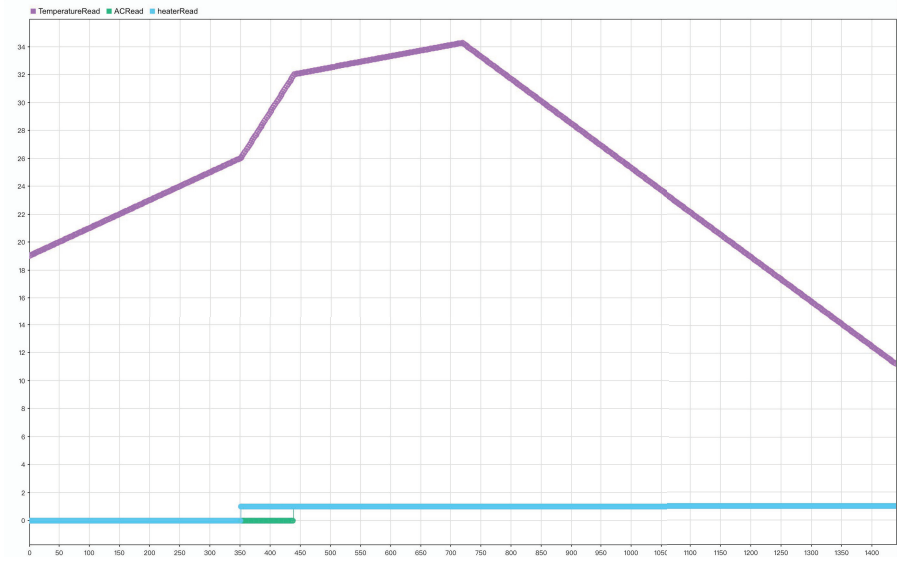

Fig. 4: The results of simulation with two devices, a heater and an AC. Value 1 indicates the state on for device and the value 0 means the device is in off state. A physical channel is modeled to represents the indoor temperature and its change over time.

5.2 Case study

In this section, we demonstrate a few case studies to show the effectiveness and usability of our testbed. The first study case models a smart home described in [10]. In this simple scenario, a misconfiguration causes a policy violation where the AC and heater run at the same time when the temperature thresholds of heating and cooling are not configured properly. Thus these errors depend on the user's configuration of apps' attributes at the installation time. Specifically, we use the same IoT apps obtained from the official SmartThings community and simulate two apps: the first one indicates that if the room temperature is higher than the user's input, then turn on the heater; the second one is opposite to the first one, i.e., if the temperature is greater than a threshold, then turn on the AC. We set the temperature thresholds to 27°C and 32°C for the heater and AC, respectively. The start time for simulation time is 0 minute and the stop time is 1440 minute, which corresponds to 24 hours, i.e., a single day. The temperature randomly varies in the range of 10 to 20, which represents a relatively cold day. Taking advantage of the parallel simulation, we run the simulation 50 times and select the first one that violates the security and safety policy. As shown in Figure

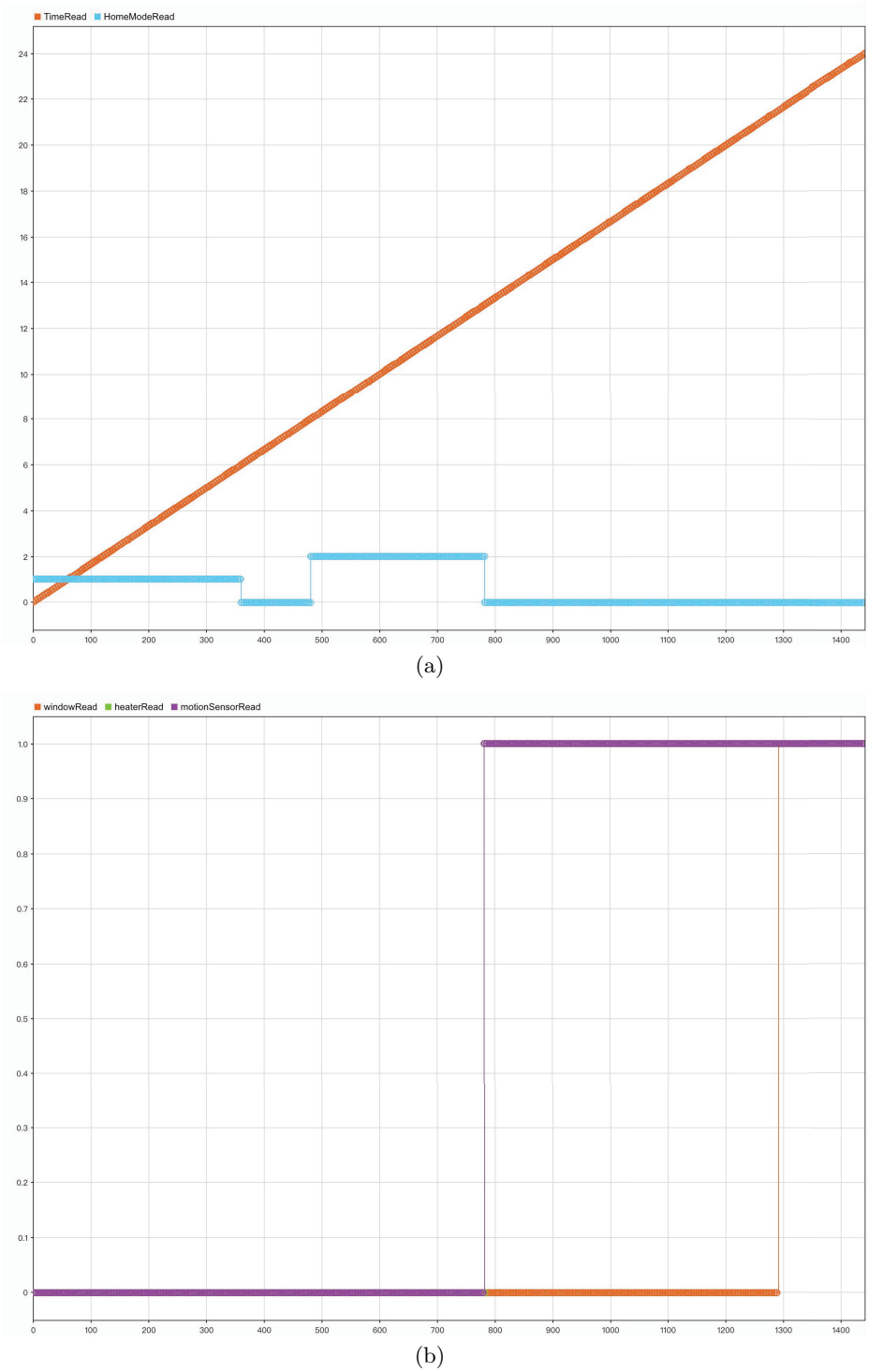

Fig. 5: In (a), time changed in a single day is illustrated and three home mode (Home; Sleep; Occupied) changed over time is represented. In (b), three devices are simulated including a window, a heater, and a motion sensor.

4, the x and y coordinates represent the testbed simulation time in minutes and the temperature value, respectively. The heater is turned on when the simulation time is around 350. When the temperature reaches 32, AC is turned on at time 440. Since the policy claims that heater and AC must not be switched on at the same time, it violates the security and safety policy defined in [10].

We conducted a second case study to demonstrate the effectiveness of the testbed against indirect attacks that exploit temporal physical interactions at run-time. In this scenario, three smart devices are deployed: a motion sensor, a heater, and a smart window. We assume that a vacuum machine unexpectedly operates or is exploited by an attacker. Eventually, it changes the home mode to 'Occupied' due to the trigger of the motion sensor. Then, the change of the home mode leads to the activation of the heater. Further, a window is opened when the temperature reaches a threshold predefined by the user. We define mode 'Sleep' is from 0 to 6 o'clock; 'Home' is from 6 to 8 o'clock and from 18 to 24 o'clock; and 'Occupied' is from 8 to 18 o'clock. And the threshold is defined as 40°C. Similar to the first case study, the initial temperature is randomly generated within the same range. We run the simulation using parallel computations and show the first violation results in Figure 5. The window should have been kept closed when the home mode is 'Sleep', putting users at risk of invasion.

6 Conclusion

In this work, we propose TAESim, a simulation testbed for IoT safety and security analysis of trigger-action environment. The simulation can be viewed as the digital twin of an IoT system with cyberspace and physical interactions. We implement this testbed to correctly model the behaviors of IoT apps, states of devices, and channels. The testbed supports large-scale analysis with no limitation of types of IoT devices and apps. The states of devices and channels can be randomly specified for test case generation. We conduct the experiments to show its efficiency and present the case studies to show its effectiveness. The results show that most simulations only consume 1-3 seconds. Simulating 5000 devices with the Stop Time 43200 minutes takes 275 seconds. We also present two case studies, that show that the testbed can properly simulate the trigger action environment and discover the safety and security violations.

References

1. Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., et al.: Fit iot-lab: A large scale open experimental iot testbed. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT). pp. 459–464. IEEE (2015)
2. Alrawi, O., Lever, C., Antonakakis, M., Monrose, F.: Sok: Security evaluation of home-based iot deployments. In: 2019 IEEE symposium on security and privacy (sp). pp. 1362–1380. IEEE (2019)

3. Bastys, I., Balliu, M., Sabelfeld, A.: If this then what? controlling flows in iot apps. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1102–1119 (2018)
4. Birnbach, S., Eberz, S.: Peeves: Physical event verification in smart homes (2019)
5. Cadar, C., Godefroid, P., Khurshid, S., Pasareanu, C.S., Sen, K., Tillmann, N., Visser, W.: Symbolic execution for software testing in practice: preliminary assessment. In: 2011 33rd International Conference on Software Engineering (ICSE). pp. 1066–1071. IEEE (2011)
6. Carter, P., Mulliner, C., Lindorfer, M., Robertson, W., Kirda, E.: Curiousdroid: automated user interface interaction for android application analysis sandboxes. In: International Conference on Financial Cryptography and Data Security. pp. 231–249. Springer (2016)
7. Celik, Z.B., Babun, L., Sikder, A.K., Aksu, H., Tan, G., McDaniel, P., Uluagac, A.S.: Sensitive information tracking in commodity iot. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 1687–1704 (2018)
8. Celik, Z.B., McDaniel, P., Tan, G.: Soteria: Automated iot safety and security analysis. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18). pp. 147–158. USENIX Association, Boston, MA (Jul 2018), https://www.usenix.org/conference/atc18/presentation/celik
9. Celik, Z.B., McDaniel, P., Tan, G.: Soteria: Automated iot safety and security analysis. In: 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18). pp. 147–158 (2018)
10. Celik, Z.B., Tan, G., McDaniel, P.D.: Iotguard: Dynamic enforcement of security and safety policy in commodity iot. In: NDSS (2019)
11. Chen, J., Diao, W., Zhao, Q., Zuo, C., Lin, Z., Wang, X., Lau, W.C., Sun, M., Yang, R., Zhang, K.: Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing. In: NDSS (2018)
12. Cheng, Z., Shein, W.W., Tan, Y., Lim, A.O.: Energy efficient thermal comfort control for cyber-physical home system. In: 2013 IEEE International Conference on Smart Grid Communications (SmartGridComm). pp. 797–802. IEEE (2013)
13. Chi, H., Zeng, Q., Du, X., Yu, J.: Cross-app threats in smart homes: Categorization, detection and handling. arXiv preprint arXiv:1808.02125 (2018)
14. Chi, H., Zeng, Q., Du, X., Yu, J.: Cross-app interference threats in smart homes: Categorization, detection and handling. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 411–423. IEEE (2020)
15. Choudhary, S.R., Gorla, A., Orso, A.: Automated test input generation for android: Are we there yet?(e). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 429–440. IEEE (2015)
16. D'Angelo, G., Ferretti, S., Ghini, V.: Simulation of the internet of things. In: 2016 International Conference on High Performance Computing & Simulation (HPCS). pp. 1–8. IEEE (2016)
17. Ding, W., Hu, H.: On the safety of iot device physical interaction control. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 832–846. ACM (2018). https://doi.org/10.1145/3243734.3243865, https://doi.org/10.1145/3243734.3243865
18. Ding, W., Hu, H.: On the safety of iot device physical interaction control. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 832–846 (2018)

19. Ding, W., Hu, H., Cheng, L.: Iotsafe: Enforcing safety and security policy with real iot physical interaction discovery (2021)
20. Egham: Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016 (2017)
21. En, O.S., Yoshiki, M., Lim, Y., Tan, Y.: Predictive thermal comfort control for cyber-physical home systems. In: 2018 13th Annual Conference on System of Systems Engineering (SoSE). pp. 444–451. IEEE (2018)
22. Fernandes, E., Jung, J., Prakash, A.: Security analysis of emerging smart home applications. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016. pp. 636–654. IEEE Computer Society (2016). https://doi.org/10.1109/SP.2016.44, https://doi.org/10.1109/SP.2016.44
23. Fernandes, E., Paupore, J., Rahmati, A., Simionato, D., Conti, M., Prakash, A.: Flowfence: Practical data protection for emerging iot application frameworks. In: 25th {USENIX} security symposium ({USENIX} Security 16). pp. 531–548 (2016)
24. Fernandes, E., Rahmati, A., Jung, J., Prakash, A.: Decoupled-ifttt: Constraining privilege in trigger-action platforms for the internet of things. arXiv preprint arXiv:1707.00405 (2017)
25. Fernandes, E., Rahmati, A., Jung, J., Prakash, A.: Decentralized action integrity for trigger-action iot platforms. In: Proceedings 2018 Network and Distributed System Security Symposium (2018)
26. Fisher, D.: Pair of bugs open honeywell home controllers up to easy hacks (2015)
27. Fouladi, B., Ghanoun, S.: Honey, i'm home!!, hacking zwave home automation systems. Black Hat USA (2013)
28. Han, S.N., Lee, G.M., Crespi, N., Heo, K., Van Luong, N., Brut, M., Gatellier, P.: Dpwsim: A simulation toolkit for iot applications using devices profile for web services. In: 2014 IEEE World Forum on Internet of Things (WF-IoT). pp. 544–547. IEEE (2014)
29. Jia, Y.J., Chen, Q.A., Wang, S., Rahmati, A., Fernandes, E., Mao, Z.M., Prakash, A., Unviersity, S.: Contexiot: Towards providing contextual integrity to appified iot platforms. In: NDSS (2017)
30. Kecskemeti, G., Casale, G., Jha, D.N., Lyon, J., Ranjan, R.: Modelling and simulation challenges in internet of things. IEEE cloud computing **4**(1), 62–69 (2017)
31. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.M.: Ddos in the iot: Mirai and other botnets. IEEE Computer **50**(7), 80–84 (2017). https://doi.org/10.1109/MC.2017.201, https://doi.org/10.1109/MC.2017.201
32. Lee, E.A., Niknami, M., Nouidui, T.S., Wetter, M.: Modeling and simulating cyber-physical systems using cyphysim. In: 2015 International Conference on Embedded Software (EMSOFT). pp. 115–124. IEEE (2015)
33. Lomas, N.: Critical flaw identified in zigbee smart home devices (2015)
34. Mao, K., Harman, M., Jia, Y.: Sapienz: Multi-objective automated testing for android applications. In: Proceedings of the 25th International Symposium on Software Testing and Analysis. pp. 94–105 (2016)
35. Mi, X., Qian, F., Zhang, Y., Wang, X.: An empirical characterization of ifttt: ecosystem, usage, and performance. In: Proceedings of the 2017 Internet Measurement Conference. pp. 398–404 (2017)
36. Nguyen, D.T., Song, C., Qian, Z., Krishnamurthy, S.V., Colbert, E.J.M., McDaniel, P.D.: Iotsan: fortifying the safety of iot systems. In: Dimitropoulos, X.A., Dainotti, A., Vanbever, L., Benson, T. (eds.) Proceed-

ings of the 14th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2018, Heraklion, Greece, December 04-07, 2018. pp. 191–203. ACM (2018). https://doi.org/10.1145/3281411.3281440, https://doi.org/10.1145/3281411.3281440
37. Nguyen, D.T., Song, C., Qian, Z., Krishnamurthy, S.V., Colbert, E.J., McDaniel, P.: Iotsan: Fortifying the safety of iot systems. In: Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies. pp. 191–203 (2018)
38. Ott, W.R.: Mathematical models for predicting indoor air quality from smoking activity. Environmental Health Perspectives **107**(suppl 2), 375–381 (1999)
39. Rastogi, V., Chen, Y., Enck, W.: Appsplayground: automatic security analysis of smartphone applications. In: Proceedings of the third ACM conference on Data and application security and privacy. pp. 209–220 (2013)
40. Ronen, E., Shamir, A., Weingarten, A., O'Flynn, C.: Iot goes nuclear: Creating a zigbee chain reaction. IEEE Security Privacy **16**(1), 54–62 (2018)
41. Ronen, E., Shamir, A.: Extended functionality attacks on iot devices: The case of smart lights. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 3–12. IEEE (2016)
42. Ronen, E., Shamir, A., Weingarten, A.O., O'Flynn, C.: Iot goes nuclear: Creating a zigbee chain reaction. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 195–212. IEEE (2017)
43. Simulink, M.: Thermal model of a house, https://www.mathworks.com/help/simulink/examples/thermal-model-of-a-house.html
44. Sivaraman, V., Chan, D., Earl, D., Boreli, R.: Smart-phones attacking smart-homes. In: Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks. pp. 195–200 (2016)
45. Son, N.H., Tan, Y.: Simulation-based short-term model predictive control for hvac systems of residential houses. VNU Journal of Science: Computer Science and Communication Engineering **35**(1) (2019)
46. Surbatovich, M., Aljuraidan, J., Bauer, L., Das, A., Jia, L.: 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. pp. 1501–1510 (2017)
47. TenWolde, A., Pilon, C.L.: The effect of indoor humidity on water vapor release in homes (2007)
48. Tian, Y., Zhang, N., Lin, Y.H., Wang, X., Ur, B., Guo, X., Tague, P.: Smartauth: User-centered authorization for the internet of things. In: 26th {USENIX} Security Symposium ({USENIX} Security 17). pp. 361–378 (2017)
49. Vidas, T., Tan, J., Nahata, J., Tan, C.L., Christin, N., Tague, P.: A5: Automated analysis of adversarial android applications. In: Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices. pp. 39–50 (2014)
50. Yu, T., Sekar, V., Seshan, S., Agarwal, Y., Xu, C.: Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In: Proceedings of the 14th ACM Workshop on Hot Topics in Networks. pp. 1–7 (2015)
51. Zhang, W., Meng, Y., Liu, Y., Zhang, X., Zhang, Y., Zhu, H.: Homonit: Monitoring smart home apps from encrypted traffic. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1074–1088 (2018)