SMARTMON: MONITORING SMART DEVICE STATUS THROUGH NETWORK TRAFFIC

by

PENGFEI PENG

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

Department of Computer and Data Science

CASE WESTERN RESERVE UNIVERSITY

August, 2020

SmartMon: Monitoring Smart Device Status through Network Traffic

Case Western Reserve University
Case School of Graduate Studies

We hereby approve the thesis[1] of

PENGFEI PENG

for the degree of

Master of Science

Dr. An Wang

Committee Chair, Adviser — June 19th 2020
Department of Computer and Data Science

Dr. Shuai Xu

Committee Member — June 19th 2020
Department of Computer and Data Science

Dr. Pan Li

Committee Member — June 19th 2020
Department of Computer and Data Science

[1]We certify that written approval has been obtained for any proprietary material contained therein.

Dedicated to my parents,
Shuxiong Peng and Huifen Cai,
who love me, believe in me, support me every step of the way.

Table of Contents

List of Tables

List of Figures

Acknowledgements

First of all, I wish to thank An Wang, my supervisor, for introducing me to the fantastic world of research of computer science, continually guiding me and giving me advice and knowledge on this road.

Great thanks to Taiyu Li, and Shengjie Ji who always stand by my side and enlighten the darkness above my head and inside my heart. Three Musketeers is always in my mind.

Thanks to Chen Zhang for being my career mentor and leading my way here. Without you I cannot even make it to the states.

Thanks to Wenjie Liao, Yuanyi Xu, Li zheng, Junchen Chi, for your support and believe. Though you are not with me now, I believe our hearts are always together.

Special thanks to Li Zheng and Huolinxiao Kuang for your comfort and encouragement every time when I almost give up myself and lost my mind.

Special thanks to Abbey Zheng for illuminating the way when I lost. You make me believe that I could still be a good photographer and create great works.

Finally, my sincere appreciation to my ex-girlfriend, especially laugh and tear we once spent together. I can't imagine I would be with you for nearly 5 years even if we couldn't see each other among most of those days due to physical distance. We didn't make it, but I never ever regret. Thank you for taking my youth away and let me know what a man should be.

Abstract

SmartMon: Monitoring Smart Device Status through Network Traffic

Abstract

by

PENGFEI PENG

The rapid expansion of Internet of Things (IoT) has brought unprecedented changes to our daily life. Among all, smart home technologies are the most widely adopted. They leverage various devices in home environment to build a connected network, over which automation is implemented for enhancing device interoperability. Such automations usually execute on platforms that are provided by device vendors, such as Samgsung, Google and Amazon. However, back-end cloud may not always be trustworthy due to malware, unknown third-party applications and possible side-channel attacks. Specifically for the IoT platforms, we identify two security threats that may gain unauthorized control of smart home devices: over privilege issue and spooking events. In this thesis, we presents SmartMon, a framework that is designed to detect such security violations by statically analyzing automation application (SmartApp) control logic and comparing them with dynamic execution patterns. Through evaluations, we demonstrate that SmartMon could achieve high precision (> 95%) in detecting both violations. We also evaluate its detection capability in more complex settings, where multiple SmartApps execute simultaneously, resulting in potential dependencies. The evaluation results show that SmartMon remains high accuracy in this scenario as well.

1 Introduction

The rapid expansion of development of Internet of Things (IoT) benefits people's daily lives in recent years, especially help people simplify their lives and create a more efficient living style through smart home. Smart home, or home automation is the technique that enable user control their own home appliance (e.g dryers, oven, washer), lighting, temperature control (heater, air conditioner), sound (music player), security (alarm sensor, smoke detector) systems. Nowadays smart home has been a indispensable and considerable part of the world economy, According to the report[1], the Smart home market is expected to grow from USD 76.6 billion in 2018 to USD 151.4 billion by 2024. A survey from statista[2] claims that the revenue in smart home of United States has reached USD 27649 millions in 2020, and Revenue is expected to show an annual growth rate from 2020 to 2024) of 14.3 percents, resulting in a market volume of USD 47, 119 millions by 2024, and Household penetration is 32.4 percents in 2020 and is expected to hit 52.4 percents by 2024. As the concept of smart home spreads, world-class commercials step in and develop their smart home platforms, such as Samsung's SmartThings[3], Apple's HomeKit[4], Wink[5], Vera Control's Vera3[6], and Google Android Things[7].

These mentioned platforms allow users control their home appliance through a connected gateway (normally a hub) by their mobile phone or cloud server. Devices are allowed to connect to hub through different communication tools (wifi, zigbee) and users are even allow to write their own recipes to control and monitor their devices. Report from securitytoday[8] presents that the number of active IoT devices has reached 26.66 billion till 2019, and by 2025, more than 75 IoT devices billion will be connected to the web. With such a great amount of connected device, the number of installed applications or recipes which are used for controlling devices is also considerable. In early 2015[9], it is reported that nearly 20 million recipes are being executed daily on IFTTT, which is one of the largest third party cloud recipe provider, and approximately 600 million each month. Consider the significance of smart home in the market of economy, and the huge quantity of the connected device and installed apps, smart home security would be a important research topic for us.

As the popularity of the Iot platform spreads and the quantity of installed Iot SmartApps raises, the safety of IoT smart home system has become a widely concerned and important issue. The first obvious threat is malicious SmartApps resulting from logic flaw or malicious code. Some researchers have already revealed security and privacy issues on different part of IoT smar thome system. Potential integrity and secrecy problem of Smart application is reported in recent research[10], 50 percents of 19323 IFTTT recipes are considered as unsafe recipes, while other researchers[11] mention that 30 percents of 279828 applets, plus more than 400 services have privacy leak threat. Some of them have the risk of being exploited by attackers through denial-of-service attacks to leak private information, and other recipes with logic conflict cause unpredictable threat to users. In addition, the design flaw of platform or cloud interfaces could be exploited

and attacked by attackers. Another paper[12] claims the web-based attack possibility towards smart home scenario, the author presents two different ways to circumvent the same-origin policy attacks: one is making use of certain flaws of interfaces of platform by applying malicious script, the other is using DNS rebinding attacks to exploit server. Both attacks reveal the safety issues between platform cloud and user, and the possibility of implementing such attacks. Not only the flaw of SmartApps or cloud interfaces can be exploited by attackers, device communication tool could be another break point for attackers. Researchers from[13] has found specific signal transferring pattern of a certain device using Zigbee as communication tool and imitate the pattern through a Zigbee signal generator from same provider to control the device, and this reveals the possibility of compromising smart home through device communication tools. Apart from these, devices and SmartApps, which work on different IoT platform may cause conflicts and then they are not able to fulfill there functionality or even becoming safety threat to users. Such threats are named as cross-app interfaces threat in the paper[14], which means that unexpected automation, security and privacy issues may be caused by different automation from different apps and platforms.

With considerable researchers concerns about the safety issues towards IoT smart home system, recent research has provided practical solutions to improve security issues in IoT environment. In our design, we consider improving app-level security issues, and other researchers also provide various solutions towards this problem. Professor Ding presents[15] a framework called IotMon, which discovers any possible physical interactions and generates all potential interaction chains from applications after using NLP to analyze application source code and description, monitors the IoT environment using log provided by IoT cloud, and finds the misbehavior according to the result

of pairing the interaction chains. But using log file restrict the framework with a high overhead, and the accuracy of interaction chains are highly relied on the descriptions due to NLP features. Researchers from University of Illinois at Urbana-Champaign[16] design ProvThings, a platform-centralized approach to centralized auditing in the Internet of Things. ProvThings collects data provenance through an API and code instrumentation to SmartApp, transferring such data to backend server and apply real-time device behavior authentication with 5 percents overheads, which is a obvious improvement compared to IotMon. However the load of code instrumentation to every single app is heavy and not elegant, and the result is still only depended on the pair result of interaction chain generated by static analysis and collected data from platform API. Another research[17] claims HoMonit, to monitor SmartApps from encrypted wireless traffic. HoMonit gains device expected behavior through analyzing application source code and UI interface, which is similar to previous research, and it obtain pair data through Zigbee sniffer, which is able to sniff all the device using Zigbee as communication tools in a 10 feet range. The design reaches a high accuracy of discovering malicious behaviors, but such system can only handle Zigbee devices in Sumsung SmartThings environment and it has physical restrictions with the authentication range.

Problem Scope: Our goal is to validate whether the status of smart device is valid, whether the event happened as the app functionality describes, and further apply machine learning model to predict device and device status based on collected data, which could help the design gain a better performance with the decision make validation system together. Since this paper concentrates on clarifying and handling application-level

issues on smart home environment, we assume other components of our model are uncompromising. Thus other type of attack such as attacks towards SmartThings hardware, for example, the DNS rebinding attacks on the browser level mentioned in this paper[18], from which attackers apply a large scale of DNS rebind attack to exploit browser and plug-ins. Another case not include is mentioned in this paper[13], attackers apply device based attack sending request and read the leaked data from a distance of about 100 meters using only cheap available equipment. Web based attack towards smart cloud or specific platform is also not include in our design.

Contributions:

$SmartMon$: We develop SmartMon, a validation system for validating IoT devices status and detecting misbehavior. We ensure our system is both effective and minimal designed, less invasive compared to prior research. And the validation and prediction of our design is relied on multiple source instead of single source, which provides a more reliable result.

$Novel\ Technique$: We apply machine-learning based prediction module, build LSTM (long short time memory) RNN (recurrent neural network) to solve a multi label status real-time prediction problem, integrate it to our system and gain a solid result.

2 Related Works

In this Section, we display the main IoT security issues from different aspects, possible solutions towards them. Internet of things has brought considerable benefit and convenience to human's lives and works from every aspect. However, the security threat from different part of IoT architecture become gradually larger due to the wide deployment of IoT devices, massive amount of IoT applications, incomplete defense strategy of IoT platforms, and the unconsciousness of users.

2.1 IoT System Architecture

The IoT architecture is primarily composed by 4 layers as shown in figure: Application layer, Support layer, Network layer and Perceptual Layer. They are the most common and primary layers among various IoT systems.

Application layer provides specific IoT service to IoT users. Normally users are permitted to access those services through certain application layer interface after authentication. For example, users can access weather report, stock notification after installed the recipes on IFTTT platform and get authorization. People can manage their smart home devices such as light and heater through installing SmartApp on Sumsung SmartThings platform.

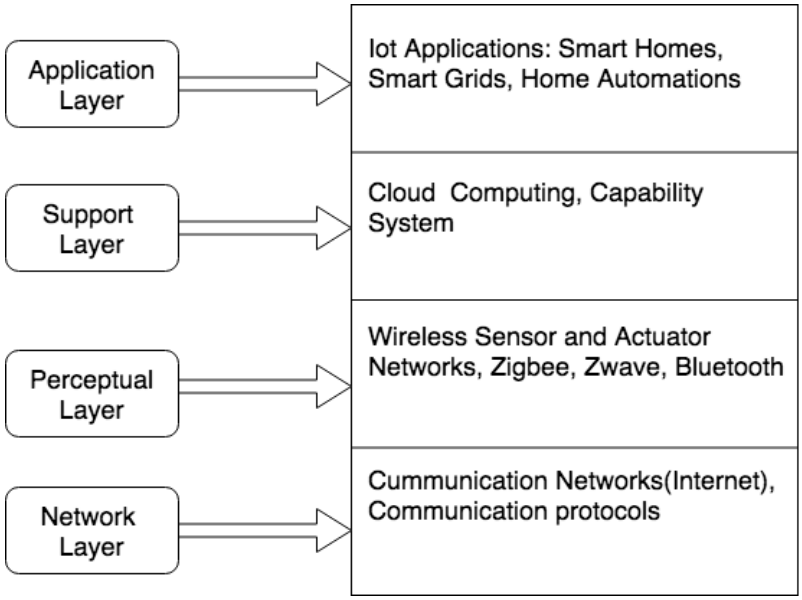

Figure 2.1. IoT Architecture

Support layer consists of those platforms that enable the functionality of applications. They allocate memory storage and computing ability to different resource restricted devices and make applications work properly and effectively. The policy and provided content vary between different platforms. For instance, Sumsung SmartThings allow user writing code to build application using groovy and provide numerous helpful API which cover most of the functions needed, while IFTTT provides the selections of third party services which allow users design their own recipes.

Perceptual layer is composed by devices such as wireless sensors and detectors which obtain real world data like temperature, weather, objects' movement. And such devices sense and transfer obtained data via their specific communication tools. For example, Zigbee and Zwave are supported for SmartThings sensors and bluetooth are enabled for android things. Due to the physical restriction of those sensors, they are prone to be the targets of cyber-attacks.

Network layer represents the network where obtained data, user commands, cloud notifications are transferred. It contains mobile network which allow communication between mobile phone and cloud, and for some city range IoT applications, satellite network are needed. However, the network layer of IoT system still face the core network security issues, such as Dos attacks, eavesdropping attack, man in the middle attack. traffic[19].

2.2 Security Issues on Access Control

The architecture of IoT environment claim similar layers compared to traditional web or mobile network and application environment, which also determines that IoT system still face various types of attack just like reported network security issues. Since most of IoT platforms are performing the function of access control, which help users monitor smart devices, it become a major attack scenario for attackers. As the IoT architecture describe, the command go through network layer, application layer and then finally perceptual layer. At network layer, attackers can apply Dos attacks, eavesdropping attack, man in the middle attack to compromise the process of access control. Attackers can temporarily hijack more than 100000 IP address to construct the bot-network, circumvent the firewall, attack the cloud browser and disable the service[20] of all access control SmartThings. The adversary can also change or recreate the routing information and distribute it in the network to create routing loops, advertising false routes, sending error messages or dropping network traffic[19] to cause a long latency when an access control command is sending. Attackers could also compromise access control in application layer. For example, the adversary can send malicious script to users via internet, the malware is going to compromise the application and run the command they acquire

such as open the door or close the security system after user open and run the script[21]. Apart from above attacks, attacker can also apply malware to steal users' data or disable service[22] in application layer. For example, Trojan horses, is one of the most dangerous malware used by adversaries to exploit a system. Besides application layer and network layer, attacker is capable of compromising access control in the perceptual layer, which is the destination of access control process. If attack have physical access to the actual devices, they can physically damage the device. Or they could apply side channel attack using information of power consumption and electromagnetic radiation from senor to attack encryption mechanisms to attack encryption mechanisms[23]. And attackers are possible to manipulate the device and sending fake command or device status if they know the mechanisms of receiving and sending data[13].

Possible Solutions The permission based access control is one of the most common and effective solution for IoT security problems such as misbehavior and over privilege and has received a lot of attention from research community[16,17,24,25]. Permission-based security models provide controlled access to various system resources. The expressiveness of the permission set plays an important role in providing the right level of granularity in access control[26]. An example of a permission-based security model is Google's Android OS for mobile devices. In Android platform, users must declare their permissions as a list in the manifest of the application to let the application know the exact access. Such strategy restrict access to dangerous resources and advanced functions and user can decide whether or not apply such permission based model on their application when install a new application[27]. The most common permission based model are also widely applied on IoT environment for restrict the available resources or functions. ContexIoT provide contextual integrity to the IoT platforms to fulfill a permission

based access control form both control and data flow levels[25]. ContexIoT collect data from application through code instrumenting and send those data to backend match module. The match module use the permission list which is generated from analyzing application source code to evaluate the risk of events and then apply control demand to actual devices. Compared to previous systems for the smartphone platforms, this system is more fine-grained since it's defined at both control and data flow levels, but it still face the problem of code instrumentation on every application and challenge on handle complicated application static analysis. we would fix such problem in our design and it would be discussed in system design part.

2.3 Security Issues on Authentication Risks

Due to the complexity of IoT system and various demands and requirement for users, IoT system is usually required to combine content and service provider, control of actual SmartDevices, cloud platform with strong computing capability, and such combination demands authentications on each transaction, which could be obvious attack point for adversary. Most of IoT platform apply Oauth authentication mechanism to ensure the safety of user login and require source. In a specific SmartApp, once users are authenticated, they are given a very powerful Oauth token which enable them use all the capabilities of all the SmartDevices installed in the application. Such powerful token is too centralize that nearly include permission of all kind of resource in the application, and is obvious to draw attackers attention and there's numerous reported attack about it. Yang et al.[28] showed that 41 percents of top 600 Android mobile applications, which use Oauth, are susceptible to remote hijacking, and the recent Google Docs Oauth-based phishing attack compromised one million users. And once attackers compromise the

platform, they will be able to leak Oauth tokens for all users. **Possible Solutions** To solve the security issues like malware and hijack attack, and ameliorate the authentication problem for IoT platform, some researchers focus on the Oauth authentication mode. Most of IoT platforms are centralized system, that enable users to create automation rules by combining together online services and physical resources using Oauth tokens. Once the token is compromised by attackers, attackers are able to complete any kind of misuse of SmartDeviecs such as manipulate devices and leak data. To ameliorate this centralized system, a Decentralized Action Integrity platform (DTAP) is presented[29]. Such DTAP splits currently monolithic platform designs into an untrusted cloud service, and a set of user clients (each user only trusts their client). DTAP introduces the concept of Transfer Tokens (X-Tokens) to practically use fine-grained rule-specific tokens without increasing the number of Oauth permission prompts compared to current platforms. In this way, each token performs their own function and decentralize the original IoT authentication mechanism.

2.4 Machine Learning Based Classification and prediction

As the development of machine learning technique, plenty of research community has applied it on IoT security problems and it does have good performance on prediction and classification problems. Recently IotSpot presents a new Machine Learning based unsupervised approach that can accurately identify IoT devices using only very limited network traffic data[30]. IotSpot constructs its dataset using the network traffic data and build random forest model to identify SmartDevice types. The network traffic data is as few as 40 minutes long. And Iotspot select feature such as packets number, packets

bytes from flow which is network traffic short period of time. The model is able to recognize around 15 different SmartDevices under its testbed with around ninety percent accuracy. Other more complex model such as Conventional Neural Networks (CNN) is also applied on prediction of IoT Devices. For instance, recent work[31], deploy Conventional Neural Networks (CNN) and Recurrent Neural Network to predict the unlabeled devices. However, training a model like CNN and RNN demands huge amount of data, which is a significant restriction for those models and it's difficult to find a balance between training the model with a smaller dataset or ensuring the accuracy with a larger dataset.

3 Background

In this section we will go over necessary background about IoT platform and Samsung SmartThings, details of specific smart applications inside SmartThings cloud and third party services supported by SmartThings.

3.1 IoT Platforms

As the trend of IoT spread world widely, numerous new IoT cloud platform are created to satisfy the demands of completing computations and running huge amount of devices and applications. As reported recently, there are more than 450 available IoT platforms in the marketplace[32]. Typically, the main functions of IoT platform is to fulfill the integration of devices from different manufacturers, provide tools or API making user design their own application, and combine more third party services. Many of them, such as SmartThings and AWS IoT[16], integrate a comprehensive set of devices and enable custom IoT applications.

As IoT is a sprawling and diverse ecosystem, in this work we focus on home automation platforms, which have the largest market share of IoT consumer products. Smart home platforms automatically manage the home environments and enable the users to remotely monitor and control their homes. Generally, in a smart home, a hub is a

centralized gateway to connect all the devices; a cloud synchronizes devices states and provide interfaces for remote monitoring and control; an app is a program that manages devices to create home automation. At present, a variety of platforms compete within the smart home landscape such as Sumsung SmartThings, Amazon echo, AndroidThings.

There are two typical IoT platform architectures: cloud-centralized architectures in which apps execute on a cloud backend, and hub-centralized architectures where apps run locally within the home. Currently, the cloud-centralized architecture is the most popular architecture since it utilize the powerful computing ability of cloud to boost the efficiency of the system and its more decentralized architecture compared to hub-centralized architecture. Across all platforms, a central point of mediation exists (i.e., hub or cloud) for control of connected devices.

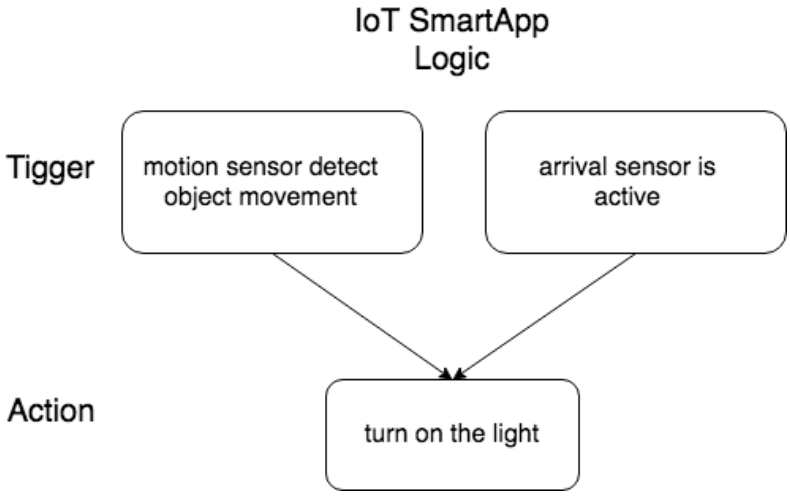

Figure 3.1. An Example of Trigger Action SmartApp

Finally, while not all products feature an app market, the logic of both appified and unappified platforms is largely specified in terms of a trigger-action programming paradigm. As the figure shows, trigger of an application represents an specific event that

cause the execution of action, usually trigger is the status of sensors and the value of mode, for example motion sensor detects object movement. And once the event of trigger is happening, action event would be executed, action events are usually actuation of devices such as turn on the light. Such trigger action logic pattern is the most popular mechanism among IoT application and widely used in most of IoT platforms including SmartThings, IFTTT, Google Home.

3.2 Sumsung SmartThings Platform

Among numerous IoT platforms, Sumsung SmartThings is one of the biggest platform and with the most users and integrating third party services. SmartThings are composed by three major components: the SmartThings cloud backend, SmartThings mobile app and SmartThings hub. The IoT apps are called SmartApps in the SmartThings platform and they are built by groovy in SmartThings IDE. Those SmartApps allow users custom their own application to manage and monitor their home devices according to their needs. Due to the restricted computing ability of IoT devices, IoT app cannot run directly on devices. Instead, SmartApps are running on the SmartThing cloud platform. The iot smart app is executed in a sandbox environment which guarantees the safety of the execution of SmartApp. For safety issues, some of the functions of the object-oriented language is disabled under under the groovy sandbox environment. For example, users cannot initialize a new class or library, but they are allowed to write their own function, define their own variables. The SmartApp also provide some API to respond to HTTP request from from external application, which is protected by Oauth-based

authentication. SmartApps support dynamic method invocation (using the GString feature), which is similar to other programming languages such as Java, a method can be invoked by providing its name as a string parameter.

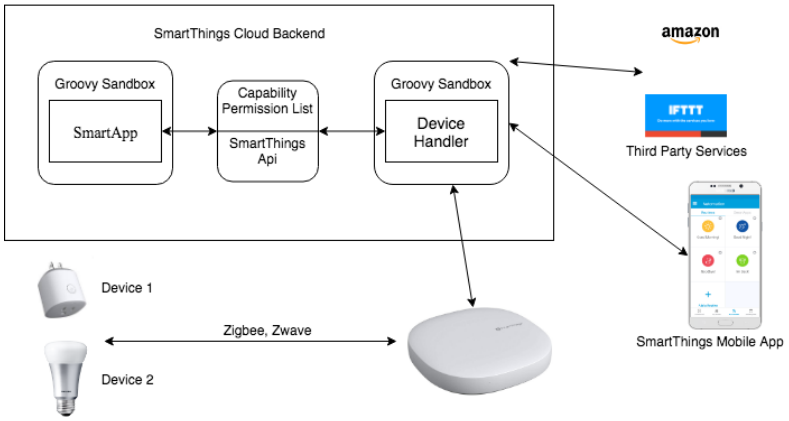

Figure 3.2. SmartThings Architecture

Most of Sumsung SmartThings devices are connected directly to SmartThings hub through z-wave and Zigbee. Communication protocol such as z-wave and Zigbee is extremely low cost and require less computation ability, which is perfectly fit IoT devices considered their small physical size and poor computation ability. For those hub connected devices, when specific event is triggered, hub would perform the function of bridge. For example, if motion sensor detect object's movement, the collected data would be sent to hub first, and then transferred to cloud platform through Zwave and Zigbee. However, for some third party devices such as Yale smart lock, they are not connected to hub through Zigbee and Zwave, instead they are connected through other communication tools such as wifi. A SmartApp and a SmartDevice communicate in two ways, SmartApp can either control the SmartDevice via provide method such as turn on the light, or trigger the SmartDevice through subscribed method when certain subscribed event is happening. Besides these, users can also control SmartDevice directly

such as open the door through SmartApp. However, the communication and control bewteen SmartApp and SmartDevice are restricted by limited capabilities. SmartThings provides a capability model for monitoring SmartDevice. This model is a coarse-grained model that defines the available functions of various SmartDevice and a set of commands for controlling them. For example, the capability switch.on and switch.off make SmartDevice outlet could be turned on and off.

All the SmartDevice in the Sumsung Smartthings platform are fulfilled their functions by SmartApps. The SmartApps are consisted of four parts: Definition, Preference, Predefined Callbacks(installed method, update method), and Event Handlers. Definition determines how the app is described. Preference specifies what kinds of devices and other information is needed in order for the application to run. Predefined Callbacks are functions called during the installation, updating and deletion of applications. Event handlers takes single argument subscription and handle the action logic, and the action could be command of controlling devices, or third platform service. The details of how the application runs and operate will be discussed in following Sumgsung SmartThings SmartApp part.

3.3 Sumsung Smartthings SmartApp

In Sumsung SmartThings Platform, the events of SmartDevices are defined by SmartApps. Inside the SmartApps, specific trigger and actions are handled by capability permission list, which is a fine-grained restricted mechanism designed by SmartThings. As the figure shows, capability permission list are shaped by attributes and commands. Commands are methods for SmartApps to control the SmartDevices while attributes defines the characteristics of the SmartDevices. One SmartDevice may have more than

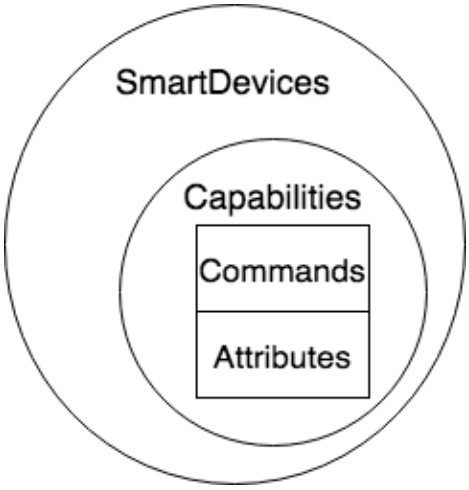

Figure 3.3. Relationship between SmartDevices and Capabilities

one capabilities, for example, SmartThings motion sensor has 6 capabilities, among those capabilities motion is used for detect the object movement while attribute power is used for reporting the device's power consumption. Under the capability permission list, SmartDevices are fulfilled restricted actions, which also secure the safety of SmartDevices and smart home environment.

As the example of Sumsung SmartThings SmartApp is shown on Listing 3.1, this is a SmartApp called Brighthen My Path, which performs the function of automatically open the light when owners coming home. specifically in one Event-Handlers SmartApp, SmartApp is composed of Definition, Preference, Predefined Callbacks, and Event Handlers. Predefined Callbacks usually include installed method, update method.

```
definition(
    name: "Brighten My Path",
    namespace: "SmartThings",
    author: "SmartThings",
    description: "Turn your lights on when motion is detected.",
    category: "Convenience",
    iconUrl: "https://s3.amazonaws.com/smartapp-icons
    iconX2Url: "https://s3.amazonaws.com/smartapp-icons
)

preferences {
```

```
12    section("When there's movement...") {
13      input "motion1", "capability.motionSensor", title: "Where?",
       multiple: true
14    }
15    section("Turn on a light...") {
16      input "switch1", "capability.switch", multiple: true
17    }
18  }
19
20  def installed()
21  {
22    subscribe(motion1, "motion.active", motionActiveHandler)
23  }
24
25  def updated()
26  {
27    unsubscribe()
28    subscribe(motion1, "motion.active", motionActiveHandler)
29  }
30
31  def motionActiveHandler(evt) {
32    switch1.on()
33  }
```

Listing 3.1. An Example of Sumsung SmartThings SmartApp

Definition determines how the app is described in mobile or cloud UI, and usually contains basic information of the application such as application name, author, namespace, description and application category. Preference specifies what kinds of devices and other information is needed in order for the application to run. It is composed of sections of input, which would contains devices' name and their capabilities, in this case, it announce used SmartDevices motion sensor and light, and their capability motionSensor and switch. Predefined Callbacks are functions called during the installation, updating and deletion of applications. Both installed and updated method are commonly found in all smartapps. Installed methods are called when a SmartApp is first installed while updated methods are called when the preferences of the app is updated. For SmartApp Brighten My Path, the eventhandler motionActiveHandler is subscribed

while installation phase and update phase. Event handlers takes single argument subscription and handle the action logic, and the action could be command of controlling devices, sending http request or third platform service. Once the event motion.active is happening, subscribe method would trigger motionActiveHandler to execute command switch1.on to turn on the light.

3.4 Sumsung Smartthings Third Party Service

The available SmartDevices and services provided by a single IoT platform is considered restricted and may not satisfy demands for users daily lives, and Sumsung SmartThings also face the same problem. To copy with such demands, Sumsung SmartThings claims "Works With SmartThings" (WWST) program, which provides available interfaces for the integration third party services and support them work simultaneously with existing SmartThings IoT devices. Among those integrated third party services available in SmartThings, IFTTT and Amazon are two of the most popular third party service providers.

Figure 3.4. An Example of Trigger Action on IFTTT

IFTTT is a cloud-centralized platform which allow users create their own recipes to connect web services. The main content provided by IFTTT is online web services such as weather report and email notification. Recipes are shaped by trigger and action which is similar to SmartThings, and users are able to set their trigger via provided field. Users

are not allowed to define web service functions in a recipe due to the coarse-refined trigger action field mechanism, which is similar to SmartThings capability permission list mechanism. IFTTT can be directly linked to SmartThings through provided action field by IFTTT or available API by SmartThings, and it largely increases the variety of the SmartThings platform architecture. Differently compared to IFTTT, Amazon smart home are mainly provided services about actual devices. Amazon have one of the largest IoT device manufacture industry and own nearly all kind of SmartDevices from various sensors to functional devices. The existence of Amazon AWS cloud platform also enable Amazon smart home running complex application with machine learning model. The integration of Amazon Smart Devices allow SmartThings gain a more completed Smart-Devices options and possibility for building complex smart application.

4 System Design

In this section, we present the details of our system design and implementation. All of the devices and sensors are connected to sumsung smartthings hub and running on the smartthings cloud. The monitoring application is running on the smartthings cloud, collecting and sending device data to device raspberry pi, while in the meantime, event validation application are running on device raspberry pi, processing data and analyzing the security issues. We have another raspberry pi as a programmable router, collecting network traffic of smartthings hub, which would be used as the source of machine learning dataset. And the trained model would provide prediction of the device status, which would help validation application gain a better performance.

To serve as a effective and minimal framework for the validation and prediction of smart devices' behavior in smart home environment, a set of challenges should be concerned, including: 1.How to extract DFA(Control Logic) automatically and properly from various smartapps? 2.How to capture smart devices' wireless traffic in iot environment and how to process validation when data is collected? 3.How to obtain trainning data and select appropriate attributes for construction of dataset for trainning? 4.How to choose appropriate and effective machine learning algorithm for prediction? 5.How to evaluate the prediction and validation accuracy in our testbed?

4.1 Threat Model

In this paper, we consider application level IoT attacks on IoT platform. These application-level attacks comes from either malware or vulnerable apps. To gain control's access without permission, attackers can apply malicious logic during installation of the application. And attackers can also exploited the design flaw to trigger unexpected device. In our paper, two specific attacks towards above vulnerabilities is mainly considered: (1) Over privileged misbehavior: attackers embedded malicious logic into apps to obtain the control of smart devices which is not mentioned in the description of SmartApp. For example, attackers can control the status of smart light in Unlock When Presence through inserting malicious code into handler method and then pass the wrong command through the unpermitted device capabilities, which may cause security issues like data leakage. (2) Event spoofing misbehavior: Attackers can lauch fake event to trigger smart device in SmartApps to impair the security of smart home. For example[33], attacker can apply a fake alarm attack on specific smart apps which include device control commands such us open heater, open the door, which may cause physical security issues like fire and robbery.

4.2 System Overview

We present a IoT Event detector to detect misbehavior and malware under Sumsung SmartThings environment and tackle over privilege and fake event situation. The IoT Event detector is consist of three components: DFA building module, Traffic Collection module, Misbehavior Detection Module. Each module performs their functions separately and works well as a united system with high accuracy and efficiency.

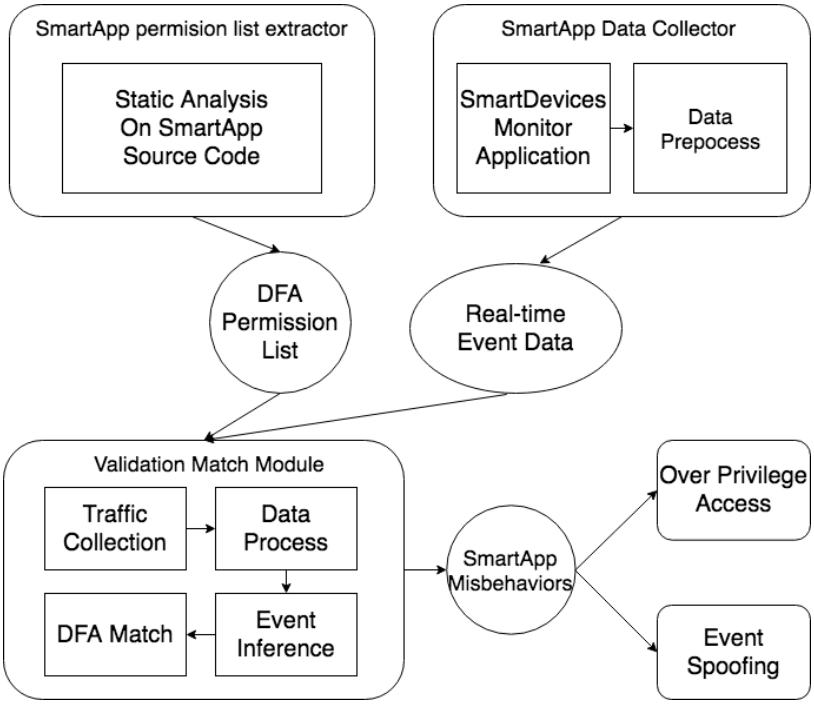

Figure 4.1. System Design FlowChart

The DFA building module performs the functions of analyzing SmartApps source code using static analysis, extract DFA(Control logic) from SmartAppps, and generate the permission white list for further match module. The Traffic Collection module is in charge of collecting SmartDevice data real-timely. Traffic Collection module is a SmartApp written in groovy and it enables all the installed SmartDevices, monitor their status and send their status via provided API when a specific event is happening. Misbehavior Detection Module is running on gateway(Raspberry pi) and it receive SmartDevices data from our Traffic Collection module and cloud server simultaneously, validate them with permission list and generate the validation result.

4.3 DFA Building Module

To validate when the device is triggered, and whether the device's status is correct, the behavior of related SmartApp is required to be learned at first. Researchers apply static analysis to extract key components they need for further processing. There are several practical method to process static analysis from different source. For example, in this paper[25], researcher extracts control logic from application code logic. While other researchers[10] extract and classify trigger and action using machine learning model NLP from IFTTT recipes' dataset, in which information such as device and capabilities are represented in a tuple. Also, researchers[15]can generate DFA from open source applications code and application description using NLP and feature extraction tools. In our design, those key components such as trigger, action, device, SmartApp name are extracted from open source SmartApps as a tuple using Abstract Syntax Tree via static analysis. And we write our groovy code to process those tuples, build DFA logic for all the SmartApps in our testbed automatically. Compared to projects mentioned above, our code consider more possible situations even when the application logic is complex, we could still obtain the needed DFA from them.

The analysis source of our static analysis is application source code from SmartThings open source library. According to the devices we have and capabilities of those devices, as the figure shows, we select 20 applications including all the possible trigger actions our testbed could handle among total more than 100 SmartApps in the library.

SmartThings applications are written by groovy, a Java-syntax-compatible object-oriented programming language. Generally speaking, there are three different kinds of SmartApps: Event-Handlers, Solution Modules, and Service Managers. In our design, we mainly focus on testing Event-Handlers application since they are the most common

apps developed by SmartThings community directly related to smart devices and provide various recipes for users. Specifically in one Event-Handlers SmartApp, SmartApp takes the form of a single Groovy script, which is composed of Definition, Preference, Predefined Callbacks (installed method, update method), and Event Handlers. And we mention the features of SmartApp in Background section. Event handlers takes single argument subscription and handle the action logic, and the action could be command of controlling devices, sending http request or third platform service.

As we mentioned in the previous part 4.1.1 Analysis Source, a typical SmartApp is constructed by several components and each component have its unique function and information. Our goal of the static analysis is finding the trigger action and specific information related to trigger action out of the application source code logic.

To achieve such goal, we must know where we could obtain information about trigger and action and what kind of information form is expected. In SmartThings platform, each device has its own capabilities, which actually means the functionality of this device. For example, motion sensor has there capabilities: motion, temperature. Capability motion allow the device detect the movement status while capability temperature allow the device detect environment temperature. To enable the functionality of a device in SmartApp, user should first announce device name and capability in the preference section, and then claim the event handler once the device event is triggered in the subscribe method in the Predefined Callbacks(installed method, update method) section. Subscribe method takes three variables: device name, device capability, event handler name. Thus, we are able to extract trigger information from subscribe method and the related event handler where usually action lies. Knowing the exact position to extract SmartApp logic, we still need to formalize the logic in a standard and understandable

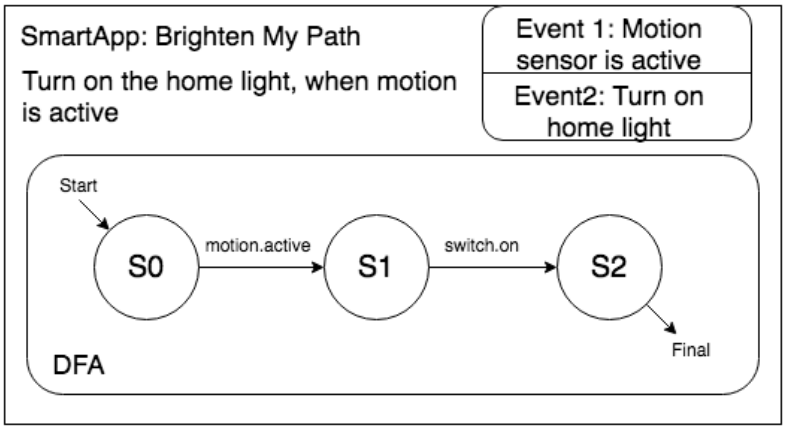

Figure 4.2. An Example of SmartApp DFA

way. Since logic inside SmartApp is usually not complex and with finite devices and devices status, We utilize the Deterministic Finite Automaton (DFA) to represent the logic of SmartApps. As the figure shows, one trigger action usually has three state, S0, S1, and S2. S0 is the initial state, S1 is the transferred state when trigger event happened, S2 is the final state after action event is executed by event handler. In the case of SmartApp Brighten My Path mentioned in Section 2 Backeground, DFA is in S0 when nothing happens, transferred to S1 when motion is active, and reach final state S2 when light is turned on. Compared to recent works, our design are applied to handle some complex logic which may contains more than one DFA and chain DFA. In chain DFA, the final state of one DFA could be the initial state of the other, which make number of states increase. specifically, we formalize the SmartApp DFA as a 4-tuple, (D, S, T, I) where D represents the set of smart devices in SmartApp, S represents the possible states, T represents transition function from one state to another, and I represents extra information about one state such as delay information.

After analyzing the structure of SmartApp, knowing where we could extract information about trigger and action, now we should find how to deal with it. In our design,

Abstract Syntax tree tool is applied to extract the information about trigger and action. an Abstract Syntax Tree (AST), is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code[15]. The details of the implementation and structure of Ast may vary towards different language. Since SmartApp are written by groovy, the Ast is a m-ary tree which means each node may have more than two children nodes and exact one parent node, each node could be a section, a method, a function or a a line of execution code. And the information of the app become more specific as the tree is traversed from top to bottom. In a specific SmartApp, from top to bottom, the first layer composed of two nodes: Blockstatement, Method. And the Blockstatement has two children: Definition, Preference, which are the main components of SmartApp mentioned in previous section. The children of Method are methond in Predefined Callbacks(installed method, update method), and method in event handlers. As shown in figure, we implement the dfa extractor in groovy. For each SmartApp in our dataset, we first initialize a CompilationUnit object and then get the Absrtact Syntax Tree first through method getAST. The AST would be traversed in preorder from top to bottom layer by layer. Next we apply getModules method on ast object, which returns nodes Blockstatement and Method. Then we apply getStatement method on Blockstatement and Method to obtain nodes of Definition, Preference, Predifined Callbacks and Event Handlers. After that we could visit those nodes separately in preorder using visit method. During the process of traversing the AST, trigger information we extract from subscribe method in Preference would be put into a trigger dictionary using event handler's name as the key, and action information we extract from event handlers would be put in another dictionary also using

event handler's name as the key. After traversal, we could generate trigger action chain through pairing these two dictionaries using the event handler's name key.

In addition, while static analysis in other mentioned papers[15] only consider the trigger action under the simplest logic, our design is able to handle complicated trigger action logic. Since we use event handlers' name as the key to pair trigger dictionary and action dictionary, we are able to extract more than one trigger action chain in a single SmartApp. For example, in the SmartApp Flood Alert, there are two subscribe methods: one is handled by waterWetHandler and the other is handled by Swithch event handler. In this situation, we can extract two trigger action chain through different event handler key. Sometimes such multiple trigger actions in one SmartApp happens with only one subscribe method. That's because users can add if else logic in the event handlers. For example, in the SmartApp Let There Be Light, there is only one subscribe method and the capability is motion. But in event handler, user apply if statement to trigger when smart light when motion is active and turn off light when motion is inactive. To solve this problem, we can obtain the information of if statement while traversing the AST through applying method getBooleanExpression().getText() on Ast node. And then trigger action chain can be generated according to different if conditions.

4.4 Traffic Collection Module

After analyzing the logic of SmartApps source code and knowing the trigger action chain of them, now we focus on acquiring and transferring device data simultaniously as the device status changes. In this part, the monitor application is running on the SmartThings cloud and to enable the functionality of this application, the device type configuration of raspberry pi should be constructed first. In the sumsung SmartThing platform,

each device has own device type handler which define its capabilities and functionalities. we discover that SmartThings cloud provide API to send command to specific IP address through get or put http request in other device type handlers. However, raspberry pi is not officially supported by SmartThings cloud, the device type handler of raspberry pi is not available. Then we write our own raspberry pi device type handlers in 252 Locs in groovy according to the basic logic of device type handler and fulfill the needed sending http request function. In this device type handlers, we enable capability Switch Level, Color Control, Color Temperature, Switch, Refresh for raspberry pi. Among those capabilities Switch Level and Switch are capabilities for controlling the open and close status, Color Temperature is the capability we apply for sending device data since it would send a http put request which contains all the information define in the monitor application to raspberry pi through SetColor method. Inside the SetColor method, we initialize a HubAction object, which is a object handling various data and command transferring of SmartThings hub, defines the sending method Put, Path and data body. Through this HubAction object, we are able to send data from SmartThings platform to raspberry pi.

After implementing the device type handler of raspberry pi, it comes to the implementation of monitor application. The functionality of this SmartApp is monitoring all the device status in our testbed and sending device data to raspberry pi when there's a device status change. The construction of monitor application is same as other groovy SmartApp which has been clarified in 4.1.1. To fulfill the function, all the capabilities that devices of our testbed owns are required to be covered in the monitor application at first. In our monitor application, we include all devices capabilities in Preference section and it allow us acquire all the device during installation phase of monitor application.

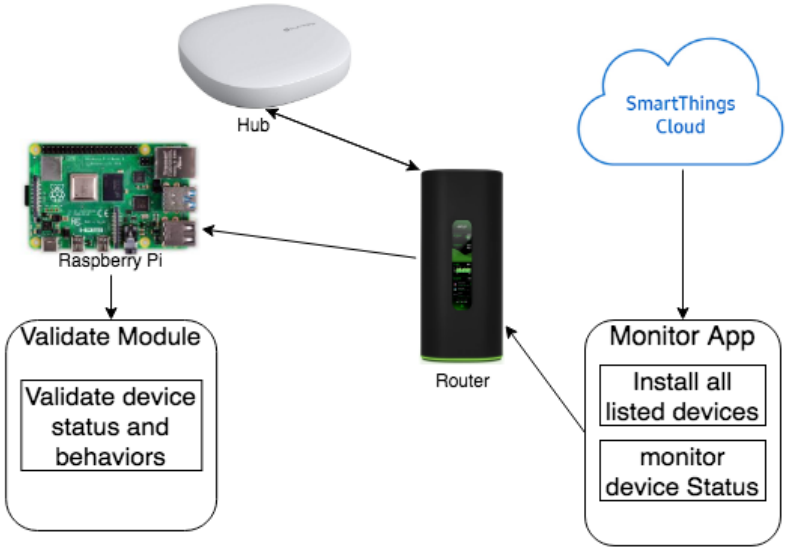

Figure 4.3. Flow Chart of Data collector

Triggering action while the device status change requires the subscribe function in install method and update method, which has been mentioned in 4.1.1. We define a subscribeToEvents() method in the installed method and updated method in Predefined method section. The subscribeToEvents() contains multiple subscribe function that cover all the possible device status variance. And all the subscribe functions in subscribeToEvents method are handler by one event handler for simplicity and efficiency of the design. Therefore, once one of the installed device status is changing, it would trigger the event handler. Event handler takes object event, which is trigger, the change of device status. For example, if the switch is on, then event handler would take this event as variable. The details of Event handler is shown as the figure. Inside the event handler, there are two for loops, which secure that we could traverse all events inside all the install devices. The device event is obtained by API device.event, and the information of the event is obtained by event.time, event.source, event.descriptionText. The event.time records the time that the event is put into the SmartThings cloud. Event.source claims the source

of this event while event.descriptionText claims the description of the event. For example, if this event is triggered by user or sensor, the source would be device, and the event.description would be the switch.on is triggered by Device. If this switch.on is triggered by SmartApp, the source would be the app command, and the event.description would be the switch.on is triggered by AppName. Noted that there's a subtle delay when event history is updated compared to the actual time when event happens. To avoid errors of obtaining device status data, instead of getting the most recent event of the device, we compare the event.time and the time of 10 most recent event in event history, and get the one with the minimal difference. After considerable test, we find the most appropriate time delay value for our testbed is 600ms. After we get the eventtime, evensource and eventtext, we send them to validation match module.

4.5 Misbehavior Detection Module

As mentioned before, nowadays validation solution towards IoT environment are most provenance or context based method, which concludes collecting data and processing data, and there are several practical solutions to reach such goal. In this paper[25], researcher apply code instrumentation on every application to get update of devices' data and then send to backend server for processing and validating. In another research[15], researcher exploit provided SmartThings API to send related devices' log data to their gateway and run several validation apps on the gateway for further processing. Among the solutions mentioned above, plenty of adjustment are required on every single app for accessing device data and sending them to server, which is laborious and time-consuming. In our design, since we apply a raspberry pi as a gateway for data processing and connect it directly to SmartThings cloud as a device, we are able to send

Algorithm 1 Detection Module Pseudo Code

Input:

D, tuple of device status $D(Device, Status, AppName, Time...)$
Sc, set of devices status collected by validate module from SmartThings cloud
Sa, set of devices status received from Monitor App
W White list of DFA, contains all the legal behaviors of SmartApps extracted from DFA generator

Output:

Rs,set of status validation result
Rt,set of trigger action validation result
Initialize $V(s) = 0$, for all $s \in \mathscr{S}^+$
Thread 1
for each $Di \in Sa$ **do**
    for each $Dj \in Sc$ **do**
        if $Di == Dj$ **then**
            $Rs \leftarrow valid$
        else
            $Rs \leftarrow fakeevent$
            if $APP-Command\ tag\ in\ Di$ **then**
                $Rs \leftarrow an\ event\ spoofing\ happened$

Thread 2

sort(Sa) according to time info inside D
for each $i \in Sa$ **do**
    for each $j \in Sa$ **do**
        if *TimeDif(Di, Dj)<threshold* **then**
            if *TA (Di, Dj) in W* **then**
                $Rt \leftarrow valid$
            else
                $Rt \leftarrow an\ over\ privilege\ happened$

Output: Rs, Rt

SmartThings device data direcetly to raspberry pi in a tuple form when devices' status changing via our monitoring SmartApp, which allow us circumvent doing code instrumentation or applying API on every SmartApp and thus keep our design simple but effective. Furthermore, since we connect raspberry pi as device to SmartThings cloud,

authorization problem won't happens. To be more specific, we have an application running on SmartThings cloud as a monitor app, collecting device data when there is a status change and sending them back to raspberry pi. And we have a validation application running on raspberry pi, which can simultaneously receive and process the data from monitor app and query device data from SmartThings cloud server, and then validate whether the device status is valid and whether the trigger action is truly happening. To fulfill the requirement of SmartDevice status validation, match resource is needed and it consists of three components in our design. The first component is SmartApp permission list generated by our permission list extractor, this SmartApp permission list perform the function of white list which provides the module valid SmartDevice behaviors. The second component is data collected by SmartApp monitor App from our data collector. This component is sent to our backend server as a tuple form. The tuple consists of all the device information with the form deviceName, sensorType, sensorStatus, Time, TimeNumber, when an event is happening. The third component is the SmartDevice status queried by our backend server. Sumsung Smartthings cloud platform provides various API and interface for developers to get resources and create their own project, in this component, we use SmartThings Switches CLI to acquire device status. First we send request to cloud for SMARTTHINGS-CLI-TOKEN which authorizes us the permission to gain resource from cloud and run command on SmartDevice. And then we are able to run sthelper status command to get all the device data directly from the cloud database. The Switches CLI also allow us run command like turn switch on to control SmartDevice if needed. After completing analysis on SmartApp and data collection, we match the data mention above and evaluate the risk of each event in validation match module. The match module is design in a multiple processes way to raise the efficiency

and handle tasks simultaneously. The whole module is written in python with 951 locs. The validation match module consist two processes: statuscheck and validate module.

Statuscheck The first process is the status check module which consistently check all the installed SmartDevice status from cloud. Status Check process is constructed by data query and data process. Data acquisition is implemented by SmartThings Switches CLI tool which is mentioned above. Since the data update speed varies for the cloud database and the data transferred by our data collector from SmartApp, the statuscheck module request data in a certain and appropriate interval that it reduce the error and latency caused by update speed from different source and raise accuracy. The data returned by sthelper command is a string of all the device information which conclude its capability, manufactory, current status and so on. We prepossess the returned string, extract the SmartDevice status information needed in a tuple form and put them in the status stack. Status stack ensure we could always obtain the most recent status for match module, if needed, we could pop out the top tuple inside the stack, which is the most recent SmartDevice status.

Detection Module The main function validate module is validate the collected SmartDevice data real-timely and make classification to judge whether there's a misbehavior. There's a real-time network sniffer inside the validate module collecting the network packet sent from cloud to raspberry pi. Since we edit the SmartDevice status as a tuple in Data Collector from SmartThings cloud, the sniffer is able to extract the tuple and process them for later validation. As the logic of our monitor application inside Data collector describes, the packet of SmartDevice status is sent to our validate module when

an event of installed SmartDevice is happening. In the mean time, the validation module start to validate whether the event is valid after receiving the packets. Since multiple events from different SmartDevice may happen together and with very short time interval, we implement a multiple thread design for our validate module to handle multiple events cases, eliminate events conflicts and increase the processing efficiency of our design. For the fake event condition, for example if attacker apply malware to compromise SmartApp and plan to open the door through a fake event, the validate module would acquire the recent actual SmartDevice status from status stack and judge whether the event is truly happened, if not, this event would be labeled fake event. Fake event produce incomplete trigger action chain, and it makes our data collector receive less packets and the validate module would recognize this situation from the comparison between the SmartDevice data extract from received packets and SmartDevice information on the permission white list, then such attack would be detected as fake event. For the over privilege condition, attackers may acquire unallowed capability or function. For example, attackers may acquire capability of turn on heater in an application which is supposed to handle turning off heater. Since under over privilege condition, the trigger action chain is complete, validate module is able to match collected data with the permission list and detect the over privilege event if it's not on the permission list.

4.6 Machines Learning based Event Classification

Machine learning is probably the most popular research topic due to its capability of handling different kinds of data and the generalization of its models. Security solutions for various security issues under IoT environment have received a great deal of attention in the recent years, researchers are attempting to apply technique of machine learning

to solve prediction and classification problem of SmartDevices. Under our system design, we already implement a complete data collect and validate module and reach a considerable efficiency and accuracy. Our system is able to normalize the inside logic of SmartApps, collect and validate SmartDevices status real-timely and detect anomalies. But one the major restriction is that the collected data used for validation is from one source, which is the SmartThings cloud. Though we assume the cloud is uncompromising and in fact it would very difficult for attackers to compromise cloud platform, it's still a restriction. To ameliorate this, we design and integrate this machine learning based prediction model, which use the source of previous knowledge of SmartDevices status and provide real-time predictions for our system. And there are several researchers also try to solve similar problems. Recently, a number of solutions have been proposed to address the different security vulnerabilities in wireless medical devices[34], researchers collect network traffic, use it building dataset, and aim to build machine learning model to recognize SmartDevices and detect anomalies. They select a period time of packets as fundamental data unit and set a tuple of features of such data unit such as packets volume, packets size, interval. The model works well on recognizing medical smart devices and perform a very solid accuracy. However, such model is not able to classify the actual status of device and several identities feature like port number, mac address has been added to the model, which are not generalized features for all circumstances and further reduce the generalization of this model. There's another paper[17], using zigbee and zwave traffics as dataset, finds the characteristics of packets length variance for different device status, and predict device status according to such feature. But such prediction mechanism relies on the environment setting heavily and could be affected by the variation of the IoT environment. Considering above works' contributions and limitations,

our machine learning based prediction module circumvent those limitations and provide a more comprehensive solution.

In this module, we collect network traffic between SmartThings cloud and hub and build our dataset. As the Architecture figure shows, when a specific event is triggered, the event handler inside SmartApp would send command to SmartThings hub trough SmartThings cloud, and then hub would tranfer such command to connected IoT devices using Zigbee or Zwave as communicate tools. We arrange OpenWRT on Raspberry pi, which is is a linux based system that turn micro controller such as raspberry pi to a programmable router. In this way, we can allocate traffic and of coarse we are able to sniff and collect the traffic between cloud and hub. As mentioned before, recent works concentrate more on traffic between device and hub, or the overall traffic in the local network environment. But the communicate tools between hub and devices is various which causes the design lack of generalization while sniffing the whole environment traffic brings plenty of noises. Our design select collecting traffic between hub and cloud, which allow us circumvent above limitations. The SmatThings hub would be allocate a IP address when connected to network, and we run our sniff code on the programmable router, sniffing and collecting the network traffic between router's IP and hub's IP.

The machine learning module we apply is LSTM (long short time memory), a model takes time series as data, and handle classification and prediction problem about problem related to time series. Since when a certain event is triggered, a series of network traffic would be generated instead of several command packets, and those packets with command information are usually encoded, it's suitable to apply LSTM to solve such time series packet level devices status classification problem. We take packets in 1 minute

as a unit for building our dataset. The time series are in the form of packets information tuple sorting according to time. Packet tuple contains the packet level feature we select for prediction such as packet size, packet type, packet direction, packet time, packet cloud query respond feature. To collect data, we manually triggered SmartDevices one time per minutes. And them we prepossess data, truncate the data flow to length 50 or enlarge the data flow to 50 with 0. Finally we normalize the dataset, reshape all value into range from 0 to 1. The basic unit of the model is 50 long time series which has a label representing its device and status. The model consist of 4 layers: input layers,one LSTM layer, one dropout layer and one output layer.

5 Evaluation

In this section, we evaluate the performance of SmartMon system in 3 sections: the performance of DFA extraction, the efficiency of detection of misbehavior, the accuracy of LSTM prediction of device status. We evaluate different session on the same SmartThings testbed.

5.1 SmartThings Testbed

We deploy our IoT environment in a private apartment, and our test devices are mainly from Samsumg SmartThings[24], one of the largest SmartThings platform in the market, which enable users edit their own recipes to monitor their devices and allow huge variety of third party services such google and amazon. The environment is composed of one SmartThings hub, 4 sumsung SmartThings sensors: motion sensor, multipurpose sensor, waterleak sensor, arrival sensor, 4 SmartThings devices: Yale smart lock, SmartTings outlet, philips hue light, raspberry pi, and a raspberry pi applied as a programmable router. All the SmartThings sensors are connected to the SmartThings hub through Zigbee[35], which is an IEEE 802.15.4-based specification with lower power and bandwith compared to bluetooth and wifi. For the connection way of SmartThings devices, the SmartThings outlet and Phillips hue light are connected through Zigbee, the Yale smart

lock is connected through Zwave, Z-Wave[36] is another popular low-power consumption communication protocol, and the Raspberry pi is connected through wifi. To monitor the network traffic and collect raw data from IoT environment, we set up a raspberry pi 3 B model as programmable router, which is connected by the SmartThings hub to sniff the network traffic of hub. And we apply another raspberry pi 3 B model as one of the SmartThings device, connecting directly to SmartThings cloud through wifi, which allow us gain the data from our monitoring app to raspberry pi. The whole testbed is set in a local network environment.

5.2 Extraction of DFA

To evaluate the performance of our DFA extractor, we apply the open source SmartApp source code from SmartThings Community github repository. Sumsung SmartThings updates the SmartApp repository frequently and allow developers upload their own application to the repository. We separate the dataset into three parts: developers apps, SmartThings apps, SmartApps in my testbed. Among those SmartApps bulit by developers, the basic logic may not be same as SmartApp built by SmartThings due to personal designed constructions and functions. And some of those SmartApps are even consist of multiple trigger actions, delayed actions and third party service, which makes the application become difficult to analyze. SmartApps built by SmartThings are based on the logic on groovy SmartThings documentation and most of those apps have single function which means they only have one trigger actions. Smart apps in our testbed cover all the possible situation we need to evaluate such as multiple trigger actions, complex logic, delayed actions and it's important to secure the accuracy of this part. In our experiment, We totally test 169 SmartApps from Sumsung SmartThings SmartApp repository,

Table 5.1. Performance of DFA Building Module

SmartApps Source	Total Number	Analyzed Number	Correct Number	Analyzed Rate	Acc
SmartThings	82	66	58	0.81	0.88
Developers	63	41	34	0.65	0.83
Apps in Our Testbed	24	22	20	0.91	0.92
Overall	169	131	112	0.78	0.86

and our DFA extractor performs a solid accuracy, extracting 131 SmartApps DFA logic and achieving a 0.86 accuracy. The analyzed rate and accuracy of apps in our testbed are 0.91 and 0.92 separately, which clarify the DFA extractor is able to handle different types of situations and secure the reliability of our testbed on further validation experiment. The analyzed rate and accuracy of apps for the apps from SmartThings are 0.81 and 0.88 separately, which clarify our design gain a good performance on standard SmartApp which follow the basic groovy SmartApp architecture. The analyzed rate and accuracy of apps in our testbed are 0.65 and 0.83 separately, which prove that our design is capable of handling SmartApps with complex logic even if developer don't built their SmartApps in a basic and formal way. The relatively low analyzed rate compared to SmartApps from SmartThings and our testbed is because nearly one thirds of developers SmartApps designed for integrate informal third party service to SmartThings platform, which makes the SmartApps contains lots of http request, authentication logic, and some of them perform the function of connection only instead of access control. Since then, our design gain a relatively low analyzed rate. But once the SmartApp contains any access control logic and is able to be analyzed, the accuracy is over 0.83, which is solid. Overall, the DFA extractor gain a good performance under our experiments and clarify its ability of handling various types of SmartApps from different source with high accuracy.

5.3 Detection of Misbehaviors towards Single SmartApps

We run the experiment of Detection of Misbehavior under our network testbed, which is declared in 5.1. To evaluate the performance of our validation system towards over privilege and event spoofing, we would randomly and manually trigger SmartApps in a period of time and evaluate the result. The experiment would also test single SmartApp and multiple SmartApps separately to observe the performance of our validation system towards multiple trigger action in short time period. Our experiment would also evaluate the time interval threshold influence on SmartApps misbehaviors validations. The time threshold is separated in two parts: one is the time threshold on capturing trigger action, which affects the performace of validating over-privileged behavior, the other is time threshold on validating SmartDevices' status, which affect the performance of event spoofing misbehavior. For over-privileged misbehavior detection, since we are doing validation while sniffing the network traffic, the incoming packets pairs are divided to trigger action according to the DFA information and the event triggered time interval, which is time interval threshold. This value could influence the probability of capturing correct trigger action, which further affect the accuracy of our model. For the validation of SmartDevices' status, due to updating speed of cloud, different reaction time for SmartDevices, time threshold here may further affects the performance of detecting event spoofing misbehavior.

we first deploy the over-privileged experiment by adding malicious code to the original SmartApps. For example, in the SmartApp Brighten My Path, the original trigger action is switch.on when motion.active, and we could manipulate the app, insert logic to turn off then light when motion sensor is active. And we could also add other device and capabilities to allow more trigger action happen. So in such way, edited Brighten My

Path become a over-privilege SmartApp for its violation of sending command switch.off and its illegal use of other devices capabilities. We have total 20 SmartApps in our test bed which represents different categories and include various kinds of SmartDevices such as Philips hue light, Yale lock, outlet, SmartThings sensors. For each SmartApp, we manually triggered the event 10 times randomly in each minute during total 10 minutes test period. During the experiment period, the SmartMon is real-timely sniffing the incoming traffic and detect misbehavior of SmartApps, the result is shown by the figure 5.1 and 5.2. Then we deploy event-spoofing misbehavior test by adding malicious logic to the original SmartApps. We install virtual device while the installation phase of SmartApp and trigger SmartApp through change the status of virtual devices. Since virtual devices are not on the DFA permission list and are not allowed to trigger the action of certain SmartApps, when such situation happens, it becomes event spoofing misbehavior. we manually triggered the event 10 times randomly in each minute during total 10 minutes test period. During the experiment period. The following two figures 5.1 and 5.2 describe precision and recall of our experiment. To define the precision and recall of our experiment, we must first declare the definition of TP (True Positive), FP (False Positive), TN (True Negative), and FN (False Negative) in our experiment. TP represents the correctly labeled misbehavior of SmartApp; a TN is a correctly labeled benign trigger action; a FP is incorrectly labeled benign trigger action; a FN is a incorrectly labeled misbehavior of SmartApp.

$$precision = TP/(TP + FP) \tag{5.1}$$

$$Recall = TP/(TP + FN) \tag{5.2}$$

The way to calculate precision and recall is described in equation 5.1 and 5.2. Precision represents the accuracy of positive labeled events among all positive predicted events while recall describes the accuracy of detected misbehavior among all actual happened misbehavior. As the following two figures shows, the overall precision of over privilege is all over 0.93 and could be as high as 0.97, which clarify that our validation system has a very high accuracy on label misbehavior when such event is happening. The overall recall is all over 0.83 and could be as high as 0.99, which means our validation system has a very high accuracy on recognizing misbehavior and seldom omits misbehavior when it's happening. On the other hand, the performance of event spoofing presents a similar pattern compared to over privilege. The overall precision is all over 0.88 but the recall value could be low as 0.65. Such performance shows that our validation system has a very high accuracy on recognized event spoofing misbehavior but omits some event spoofing misbehavior when it's happening.

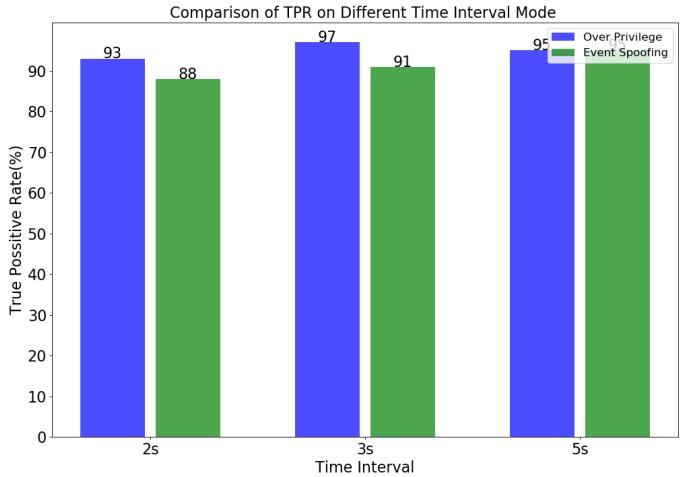

Figure 5.1. Precision on Different Time Interval

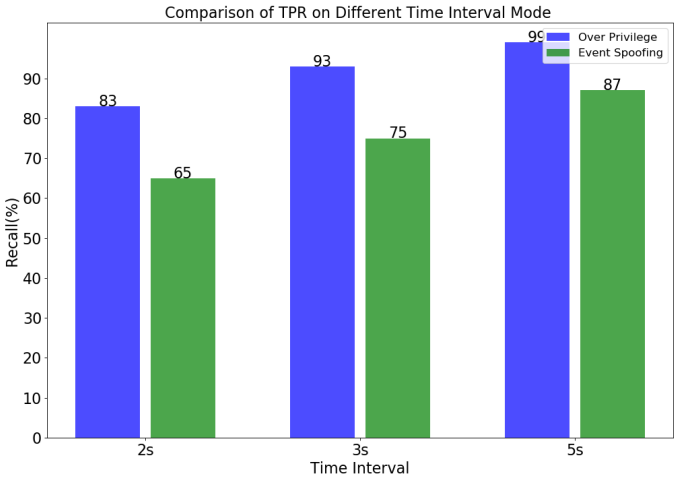

Figure 5.2. Recall on Different Time Interval

We make the experiments with the variance time interval threshold while other variables remain unchanged. We select 2 seconds, 3 seconds and 5 seconds as time interval threshold value to do separate experiments due to the features of communication between SmartThings cloud platform and hub. trigger and action are supposed to happen in the same time, but due to time used by computing and sending command, and delay of network or cloud platform, sometimes there's 1 or 2seconds delay between trigger event and action event. For some specific third party services or devices, such time interval may be longer, which makes the experiments of performance over different time interval threshold meaningful. As the figure 5.1 shows, the overall precision is all above 0.93 which means time interval threshold has few impact on the accuracy of recognized over-privilege misbehavior. However, there's a obvious difference on recall performance over-privilege misbehavior. The recall of time threshold 2 seconds is 0.83 while the recall of time threshold 5 seconds is 0.95. This result shows that though difference on time interval threshold would not affect the accuracy of recognized over-privilege misbehavior,

but it does affect the detect rate of over-privilege misbehavior. Some of the trigger action may not be captured due to the short time threshold and cause a loss of recall. On the other hand, The recall of event spoofing on time threshold 2 seconds is 0.65 while the recall of event spoofing on time threshold 5 seconds is 0.87. This claim that short time threshold does affect the possibility of detecting a event spoofing a lot. Since our system query the most recent status of SmartDevices from cloud for several time and validate with real-time data. Thus if the time threshold here is too short, the validation result would be poor for some specific SmartDevies with a longer reaction time. For example, for Yale Smart Lock, usually it take more than 3 seconds to react commands, and for waterleak sensor, the delay is around 2 seconds. That's the reason why the recall value is low when the time threshold is short. To ameliorate such condition, we can choose a longer threshold, as the figure 5.2 shows, when threshold is 5s, the recall value would be 0.87. On the other hand, we set delay value for specific SmartDevice such as Yale lock, which could solve such problem and increase the accuracy of our system. And since the validation module is designed in a multiple process and multi thread way, such revise won't affect the efficiency of our system, and the we would clarify the improvement in figure 5.3 and 5.4.

The following table describe the precision, recall value and devices' capabilities for each SmartApp when the time threshold is 5 seconds in our testbed. Overall we can see our validation module gain a high performance for each SmartApp, most of precision and recall value are over 0.9 and nearly 1.0 and the lowest precision value and recall value are 0.76 and 0.9, which is still solid performance.

Table 5.2. Performance of Detection Module on Each SmartApp

SmartApps	Devices	Capabilities	Pre	Rec
Brighten Dark Places	Philips Hue Light, Multipurpose Sensor	switch, multipurposeSensor	1.00	.1.00
Brighten My Path	Outlet, Motion Sensor	switch, motionSensor	0.87	1.00
Flood Alert	Outlet, Waterleak Sensor	switch, waterleakSensor	0.76	0.95
It Moves	Outlet, Multipurpose Sensor	switch, multipurposeSensor	0.95	1.0
Let There Be Dark	Outlet, Multipurpose Sensor	switch, multipurposeSensor	1.0	1.0
Light Follows Me	Motion Sensor, Outlet	switch, motionSensor	0.8	0.95
Lights Off When Closed	Outlet, Multipurpose Sensor	switch, multipurposeSensor	0.85	1.0
Lock It When I Leave	Yale Lock, Arrival Sensor	lock, presenceSensor	0.93	0.93
Monitor On Sense	Outlet, Multipurpose Sensor	switch, accelerateSensor	0.95	1.0
My Light Toggle	outlet, Motion Sensor, Philips Hue Light	switch, motionSensor	1.0	1.0
Turn It On For 5 Min	Outlet	switch, cloud	1.0	1.0
Turn It On When It Opens	Philips Hue Light, Multipurpose Sensor	switch, multipurposeSensor	0.95	1.0
Turn Off With Motion	Outlet,Motion Sensor	switch, motionSensor	0.87	1.0
Unlock It When I Arrive	Yale Lock,Arrival Sensor	lock, presenceSensor	0.95	0.9
When Present Turn On	Outlet, Arrival Sensor	switch, presenceSensor	0.95	1.0
Turn It On every 1 Min	Philips Hue Light	switch, cloud	1.0	1.0
Left It Open	Philips Hue Light, Arrival Sensor	switch, presenceSensor	0.9	0.95
The Gun Case Moves	Outlet, Multipurpose Sensor	switch, accelarationSensor	0.85	0.9
When Presence Open The Light	Philips Hue Light, Arrival Sensor	switch, presenceSensor	1.0	1.0
Smart Turn It On	Philips Hue Light	switch, cloud	0.9	1.0

5.4 Detection of Misbehavior towards Multiple SmartApps

For the detection of misbehavior on multiple SmartApps, we apply similar test method as the single SmartApp test. We install two or three SmartApps and test them simultaneously. We randomly trigger multiple SmartApps in 10 minute for 10 times. To evaluate the ability of handling multiple trigger action validation in short time, we trigger multiple trigger action in the same time. Figure 5.3 and 5.4 shows the precision and recall value of performance of our system respectively. As the figure 5.3 and 5.4 shows, the recall value are around 0.9 for event spoofing, which prove that the system doesn't omit happened event-spoofing misbehavior after adding delay option for specific SmartDevices. The overall precision value is all above 0.88 for detection of both over-privilege and event spoofing misbehavior, which clarify that our validate module is able to handle high volume trigger action in short time and retains a high performance. The highest recall value happens when the time threshold is 5 seconds and it means with longer time

threshold, more trigger action and devices status are able to be validated. However, the highest precision happens when time interval was 3 seconds instead of 5 seconds, which clarify that though short time threshold omit some trigger action, it produce less false positive compared to longer time threshold.

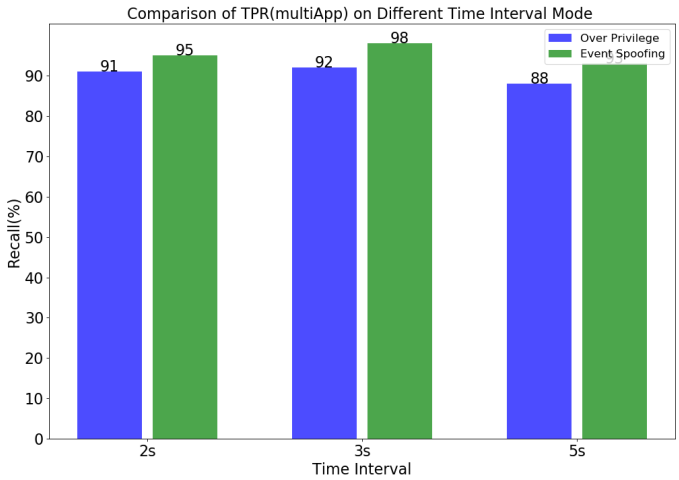

Figure 5.3. Precision on Different Time Interval (multiple App)

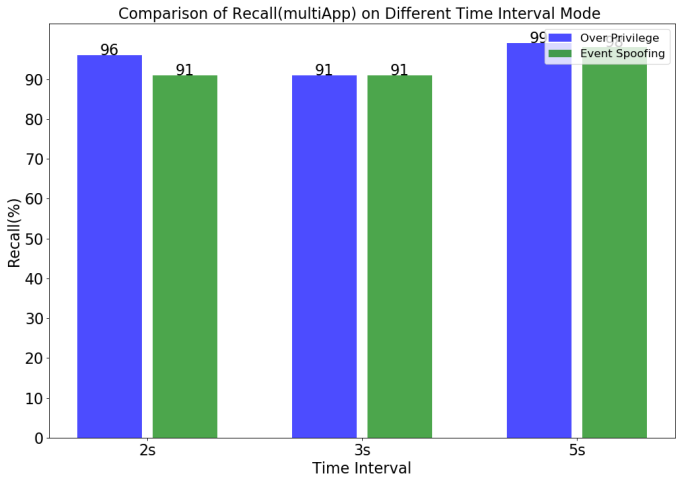

Figure 5.4. Recall on Different Time Interval (multiple App)

Table 5.3. Performance of Event Classification

SmartDevices	Accuracy
Outlet	0.85
Philips Hue Light	0.85
Yale Smart Lock	0.7
Motion Sensor	0.8
Multipurpose Sensor	0.6
Waterleak Sensor	0.6
Overall	0.74

5.5 Event Classification of Device Status

To fulfill the evaluation of event classification, we find the appropriate variable setting for our model through tuning the value of epoch and batch size. We select softmax as activation function since its popularity on multi-label classification problems. The test method is the same as method we apply in misbehavior detection module. We randomly trigger Smart devices 10 times in 10 minutes and validate whether the classification result is correct. As the figure shows, The overall classification accuracy is 0.74, which is solid. And the accuracy of outlet, Philips light and Yale lock is higher than sensors, which mean this module performs better on classifying Smart devices than sensor status. Sensors have lower accuracy which is because they are more similar. They are all from SmartThings, follow the same communication mechanism.

6 Conclusions

In this paper, we present SmartMon, a misbehavior detection system for Sumsung SmartThings platform to detect misbehavior such as event spoofing and over privilege in SmartApps. Our SmartMon are able to infer and validate smart devices status and analyze them real-timely. The key components of SmartMon includes DFA building module, traffic collection module, misbehavior detection module and event prediction module. DFA building module can successfully extract SmartApp logic from source code through static analysis with a solid accuracy. Traffic collection module sniffs smart devices status via monitor SmartApp and real-timely transfer data to misbehavior detection module. Misbehavior module is able to validate device status from multiple source traffic collector and events prediction module with high performance. Events prediction module provides prediction based on LSTM RNN model training by net work traffic data, presents a solid prediction result and provides validation source for misbehavior detection module. Overall the system work well together and provides a novel and complete security plan for IoT environment.

7 Future Work

Though we design a complete system with high performance and efficiency, there's still further work could improve and advance our design. The data collection is fulfilled by traffic collection module, inside the module we have a monitor app which is able to monitor all the smart devices and we integrate raspberry pi to SmartThings which make it perform the function of smart device and gateway simultaneously. However, not all of the IoT platform support such integration idea, and it would be difficult for some platform with restricted support for third party device integration such as IfTTT. To compensate such limitations and generalize the design, we need to further concentrating on how we could find a way overcoming data collection problems with restricted platform. Another improvement we could apply is in the events prediction module. Due to the scale of our manually designed and made dataset, lack of smart devices and selection of model, the performance of this part is expected to have a possibility to be further improved. We could wait official smart devices status dataset or attempt to apply transferring learning on our design. And we could construct the module with other machine learning model to see the performance.

Complete References

[1] Marketsandmarkets. 2020. https://www.marketsandmarkets.com/Market-Reports.

[2] Statista. 2020. https://www.statista.com/outlook/279/109/smart-home/united-states//.

[3] Sumsung. 2018. Smartthings. www.smartthings.com/.

[4] Apple. 2018. Apple homekit. https://developer.apple.com/homekit//.

[5] Wink. 2018. https://www.wink.com//.

[6] Vera. 2018. https://getvera.com//.

[7] Google. 2018. Android things. https://developer.android.com/things/.

[8] Marketsandmarkets. 2020. https://securitytoday.com/Articles/2020/01/13/The-IoT-Rundown-for-2020.aspx?Page=2//.

[9] Marketsandmarkets. 2020. http://qz.com/346767/ifttt-pares-down-its-automation-serviceto-prepare-for-the-one-\click-smartwatchfuture.

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

[11] Iulia Bastys, Musard Balliu, and Andrei Sabelfeld. If this then what? controlling flows in iot apps. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, page 1102–1119, New York, NY, USA, 2018. Association for Computing Machinery.

[12] Gunes Acar, Danny Yuxing Huang, Frank Li, Arvind Narayanan, and Nick Feamster. Web-based attacks to discover and control local iot devices. In Proceedings of the 2018 Workshop on IoT Security and Privacy, pages 29–35, 2018.

[13] Eyal Ronen and Adi Shamir. Extended functionality attacks on iot devices: The case of smart lights. In 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pages 3–12. IEEE, 2016.

[14] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Jiaping Yu. Cross-app interference threats in smart homes: Categorization, detection and handling. arXiv preprint arXiv:1808.02125, 2018.

[15] Wenbo Ding and Hongxin Hu. On the safety of iot device physical interaction control. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 832–846, 2018.

[16] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. Fear and logging in the internet of things. In Network and Distributed Systems Symposium, 2018.

[17] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. Homonit: Monitoring smart home apps from encrypted traffic. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 1074–1088, 2018.

[18] Dennis Tatang, Tim Suurland, and Thorsten Holz. Study of dns rebinding attacks on smart home devices. In Computer Security, pages 391–401. Springer, 2019.

[19] Di Wu, Gang Hu, and Gang Ni. Research and improve on secure routing protocols in wireless sensor networks. In 2008 4th IEEE International Conference on Circuits and Systems for Communications, pages 853–856. IEEE, 2008.

[20] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. Protecting browsers from dns rebinding attacks. ACM Transactions on the Web (TWEB), 3(1):1–26, 2009.

[21] Tobias Heer, Oscar Garcia-Morchon, René Hummen, Sye Loong Keoh, Sandeep S Kumar, and Klaus Wehrle. Security challenges in the ip-based internet of things. Wireless Personal Communications, 61(3):527–542, 2011.

[22] Hui Suo, Jiafu Wan, Caifeng Zou, and Jianqi Liu. Security in the internet of things: a review. In 2012 international conference on computer science and electronics engineering, volume 3, pages 648–651. IEEE, 2012.

[23] Kai Zhao and Lina Ge. A survey on the internet of things security. In 2013 Ninth international conference on computational intelligence and security, pages 663–667. IEEE, 2013.

[24] Jasmin Guth, Uwe Breitenbücher, Michael Falkenthal, Paul Fremantle, Oliver Kopp, Frank Leymann, and Lukas Reinfurt. A detailed analysis of iot platform architectures: concepts, similarities, and differences. In Internet of Everything, pages 81–101. Springer, 2018.

[25] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and SJ Unviersity. Contexlot: towards providing contextual integrity to appified iot platforms. In NDSS, 2017.

[26] David Barrera, H Güneş Kayacik, Paul C Van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In Proceedings of the 17th ACM conference on Computer and communications security, pages 73–84, 2010.

[27] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. IEEE security & privacy, 7(1):50–57, 2009.

[28] Ronghai Yang, Wing Cheong Lau, and Tianyu Liu. Signing into one billion mobile app accounts effortlessly with oauth2. 0. blackhat Europe, 2016.

[29] Earlence Fernandes, Amir Rahmati, Jaeyeon Jung, and Atul Prakash. Decentralized action integrity for trigger-action iot platforms. In Proceedings 2018 Network and Distributed System Security Symposium, 2018.

[30] Liangdong Deng, Yuzhou Feng, Dong Chen, and Naphtali Rishe. Iotspot: Identifying the iot devices using their anonymous network traffic data. In MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM), pages 1–6. IEEE, 2019.

[31] Jorge Ortiz, Catherine Crawford, and Franck Le. Devicemien: network device behavior modeling for identifying unknown iot devices. In Proceedings of the International Conference on Internet of Things Design and Implementation, pages 106–117, 2019.

[32] IoT Analytics. Iot platform comparison: How the 450 providers stack up, 2017.

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

[34] Sida Gao and Geethapriya Thamilarasu. Machine-learning classifiers for security in connected medical devices. In 2017 26th International Conference on Computer Communication and Networks (ICCCN), pages 1–5. IEEE, 2017.

[35] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. Iot goes nuclear: Creating a zigbee chain reaction. In 2017 IEEE Symposium on Security and Privacy (SP), pages 195–212. IEEE, 2017.

[36] Muneer Bani Yassein, Wail Mardini, and Ashwaq Khalil. Smart homes automation using z-wave protocol. In 2016 International Conference on Engineering & MIS (ICEMIS), pages 1–6. IEEE, 2016.