IoT-Praetor: Undesired Behaviors Detection for IoT Devices

Juan Wang, *Member, IEEE*, Shirong Hao, Ru Wen, Boxian Zhang, Liqiang Zhang, Hongxin Hu, *Member, IEEE*, and Rongxing Lu, *Senior Member, IEEE*

***Abstract*—Due to insecure design and configuration, the Internet-of-Things (IoT) devices are vulnerable to various security issues. In most attacks against IoT, e.g., Mirai, attackers control devices to perform malicious behaviors that are not expected by owners and administrators. Therefore, how to effectively detect malicious behaviors is crucial to protect the security of IoT devices. Different from powerful PCs and servers, resource-constrained IoT devices are generally used to execute the specific function and their behaviors are limited. Based on this observation, we propose IoT-Praetor, an undesired behavior security detection system for IoT devices. In IoT-Praetor, a new device usage description (DUD) model is proposed to construct an IoT device behavior specification, including communication and interaction behaviors. Furthermore, automatic behavior extraction approaches are presented. We also design a behavior rule engine to detect device behaviors in real time. To evaluate the effectiveness of IoT-Praetor, we implemented our methods on Samsung SmartThings and performed a security test. The evaluation results show that the successful detection rate of malicious interaction behavior is 94.5% on average, and the detection rate of malicious communication behavior is above 98%, and system running time delay is only in millisecond level.**

***Index Terms*—Deep learning, device identification, Internet of Things (IoT), security.**

I. Introduction

THE Internet-of-Things (IoT) technology has received considerable attention in various applications, including smart healthcare, smart city, smart building, and smart home. Among these applications, smart home [1] is one of the most popular ones. Essentially, smart home integrates various home IoT devices, and those home IoT devices can communicate and share data with each other, enabling homeowners to freely control them for a better convenience life. Recently, some integrated platforms have emerged, including Samsung SmartThings [2], Apple HomeKit [3], AWS IoT [4], OpenHAB [5] (open source), and HomeAssistant (open source) [6]. In addition to these control platforms, there are also some third-party platforms such as IFTTT [7], which can be linked to existing professional devices and communicate with mainstream platforms to provide more customized services.

Nevertheless, with the proliferation of IoT devices, security issues [8], [9]–[14], such as DDoS attacks and privacy leaks have become increasingly serious. A simple IoT system may contain a large number of resource-constrained and low-power IoT devices so that those traditional defense methods such as anti-virus software, firewalls, etc., are difficult to install directly in the devices to resist security threats. In addition, a large number of legacy IoT devices have more or less design and configure defects, such as hardcoded or weak passwords, and those flaws may not be fixed through firmware update because manufacturers may stop the maintenance of the devices. Hence, there are still lots of vulnerable devices running on the market, which further makes the security of IoT devices a major challenge. In most attacks against IoT devices, e.g., Mirai [15], attackers control devices to perform malicious behaviors that are not expected by owners and administrators. Therefore, how to effectively detect malicious behaviors becomes crucial to protect the security of IoT devices.

Existing research work on IoT security, such as Homonit [16] and SmartAuth [17] that apply code analysis and natural language processing (NLP) to infer whether SmartApps follow the original design goal and execute some unauthorized behaviors. FlowFence [18] ensures that authorized users can access data legally by using information flow tracking technology. However, these solutions mainly focus on IoT applications security. Currently, CISCO has proposed manufacturer usage description (MUD) [19] as a standard specification to formulate behaviors of IoT devices. However, it only includes some network access rules specified by device manufacturers, resulting in that MUD can only define very limited device communication behaviors.

In this article, we present IoT-Praetor, a new behavior-based security detection system for IoT devices. In IoT-Praetor, aiming at formally specifying the expected behaviors of

Manuscript received January 9, 2020; revised March 21, 2020 and June 9, 2020; accepted June 30, 2020. Date of publication July 17, 2020; date of current version January 7, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61872430; and in part by the National Basic Research Program of China (973 Program) under Grant 2014CB340600. *(Corresponding authors: Liqiang Zhang; Boxian Zhang.)*

Juan Wang and Liqiang Zhang are with the School of Cyber Science and Engineering, Key Laboratory of Aerospace Information Security and Trust Computing, and Ministry of Education, Wuhan University, Wuhan 430072, China (e-mail: jwang@whu.edu.cn; zhanglq@whu.edu.cn).

Shirong Hao and Ru Wen were with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China. They are now with the OS Security Team, Alibaba Group, Hangzhou 311121, China, and also with JD Retail, Beijing 100176, China (e-mail: shirong@whu.edu.cn; ruyulh@whu.edu.cn).

Boxian Zhang is with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China (e-mail: boxzhang@whu.edu.cn).

Hongxin Hu is with the School of Computing, Clemson University, Clemson, SC 29634 USA (e-mail: hongxih@clemson.edu).

Rongxing Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada (e-mail: rlu1@unb.ca).

Digital Object Identifier 10.1109/JIOT.2020.3010023

an IoT device, we propose a new device usage description (DUD) model to construct the behavior specification of IoT devices, including the desired communication behaviors and interaction behaviors. In addition, we present automatic approaches based on crawling and NLP to extract device interaction behavior rules and expected communication behavior rules. Furthermore, we design a behavior rule engine to monitor and detect device behaviors in real time.

We implement IoT-Praetor based on Samsung SmartThings, one of the most popular smart home platforms. To evaluate the effectiveness of IoT-Praetor, we leveraged malicious SmartApps and IFTTT Applets, Mirai Botnet attack for security testing. The results indicate that the successful detection rate of malicious interaction behavior is 94.5% on average and the detection rate of malicious communication behavior is above 98%, and system running time delay is in millisecond level. As a result, our system can effectively detect and defend against malicious behaviors of IoT devices while introducing acceptable performance overhead.

Specifically, our main contributions can be summarized as follows.

1) We propose a new DUD model to construct the IoT device behavior specification. Our model can specify the fine-grained interaction behaviors and communication behaviors of the whole system over the entire lifecycle for IoT devices.
2) We propose the approaches to automatically extract device usage rules (DURs), which can extract device interaction rules from SmartApps and IFTTT applets and extract communication behavior rules including hidden characteristics by analyzing the communication traffic of incoming and outgoing devices.
3) We design and implement IoT-Praetor, a behavior security detection system for IoT devices based on DUD, and evaluate its effectiveness.

The remainder of this article is organized as follows. Section II introduces the background and Section III discusses the threat model. Section IV describes our system overview and Section V describes the basic structure of DUD. The automatic extraction of device behavior rules is described in Section VI. We present the design of malicious behavior detection in Section VII. The implementation and evaluation of our system are described in Section VIII. We discuss limitations and future work in Section IX. Finally, we present related work in Section X and conclude this article in Section XI.

II. Background

A. Samsung SmartThings

As one of the most popular smart-home platforms [20], SmartThings can integrate different devices from different manufacturers through networks and cloud frameworks. In addition, SmartThings can enable users to create personalized rules based on their needs by the trigger–action programming paradigm to implement complex operations across devices. A SmartThings platform includes three major components: 1) SmartApps; 2) SmartThings cloud backend; and 3) a Smart Hub.

SmartApps are the small Groovy programs running in the Groovy sandbox environment provided by the SmartThings cloud backend, which allows users to connect their devices making their home more intelligent, such as "turn on the light when the motion sensor is active". SmartApps allow external system API access to communicate with external Web services such as sending an email, which is protected by OAuth2 authentication.

SmartThings cloud backend provides the Groovy sandbox environment for SmartApps and Device Handler. Device Handler is the virtual representation of a physical device, which is responsible for communicating between the physical devices and the SmartThings platform. Developers can create Device Handler to integrate new devices into the SmartThings system. The SmartApps subscribe to the events from Device Handler and send the corresponding commands to the Device Handler to control the device. The communication between the SmartApps and Device Handler is based on capabilities that are the core of the SmartThings architecture. Capabilities are decomposed into a set of commands and attributes that devices can support. Smart Hub is a hardware physical device.

The hub is responsible to send commands from the SmartThings Apps to the connected devices and send the states of devices to the user's phone.

B. IFTTT Platform

Triggering operating platforms, such as IFTTT, Zapier [21], and Apiant [22] allows users to connect services together and implement "trigger–action" operations. The users of the IoT platform can authorize their apps to automatically access the services of trigger operating platforms. For example, using the OAuth protocol, SmartThings users can authorize IFTTT to communicate with their services. IFTTT (If-this-then-that) [7] enables users to create automation rules (or recipes) across multiple platforms according to the "trigger–action" paradigm. It combines trigger and action to define various automation tasks, providing a simplified way that users can easily follow.

In addition, IFTTT provides a powerful and simple Web UI for mobile Applets development. By operating on the interface, users can splice different IoT devices and services, create new functions, and customize device behavior rules to achieve intelligent devices easily. At the same time, IFTTT has storage capabilities of rules, allowing users to reuse automated rules provided by third-party developers or others.

In this article, we design and implement the IoT-Praetor architecture and system based on Samsung SmartThings and IFTTT platform. In IoT-Praetor, devices are managed by Samsung SmartThings and can communicate with other IFTTT platforms to use their services for intelligent functions.

C. MUD

MUD [19] is an IETF specification for defining behaviors of IoT devices. In August 2016, the relevant draft of MUD was submitted by Cisco. Recently, MUD has been officially approved as an Internet Standard by IETF. MUD allows IoT device manufacturers to define behaviors of IoT devices such as communication patterns. The specification [23] can be used to find network threats for IoT devices because IoT devices are expected to be special purpose and have a small number of predictable traffic flows, which can be captured via relatively simple policies.

MUD is provided in the form of a JSON file, which contains an abstract device policy describing the normal communication access rules of IoT devices. An MUD controller is used to request and receive information from the MUD file server. After an MUD file is received, the abstract information in the MUD file is converted to the specific network element's configuration while maintaining and updating any necessary mappings.

However, MUD only contains the network access rules specified by the manufacturer at the initial stage of device joining the network, failed to consider adequately device interaction rules according to the user's requirements. This results in the consequence that MUD can only define limited device communication behaviors and miss other threats to IoT devices. Additionally, the goal of MUD is specifying the device behaviors by manufacturers, hence it does not fully consider the behaviors during devices running in a real environment. To specify the fine-grained behaviors of the whole system over the entire lifecycle for IoT devices, we propose a new IoT DUD model, which consists of device interaction behaviors and device communication behaviors. In our work, the DUD rules can be automatically extracted from the device information, SmartApps description, IFTTT Applets rules description, and device communication packets. Then, a white list of device behaviors is generated.

III. Threat Model

We consider adversaries can compromise IoT devices through malicious network communication, such as Mirai and Hajime [24], which are the most notorious DDoS attacks to IoT devices. Moreover, we assume SmartApps and IFTTT Applets may be malicious or compromised so that adversaries can exploit security design flaws in SmartThings' capability model and event subsystem of smart applications, causing that SmartApps deviate from the original design goal. For example, Fernandes *et al.* [25] summarized SmartApps misbehaviors including: 1) over-privileged access, which is performed to gain control of devices in a manner not specified by its intended function and 2) event spoofing, which can be used to perform fake events, causing SmartApps to activate some actions mistakenly [16].

However, we assume that devices will not be attacked at the initial stage of connecting to network because attackers need to spend some time, resources, and energy to find out vulnerabilities of devices, and how to exploit them before carrying out attacks. Therefore, our IoT-Praetor system can extract behaviors from clean data as a baseline.

Besides, we do not consider physical channel attacks, such as the security of Bluetooth, ZigBee, and the side-channel attacks to IoT devices [4], [26]–[28].

IV. Design Overview of IoT-Praetor

IoT devices are expected to be special purpose and have a small number of predictable traffic flows, which can be captured via relatively simple policies. Based on this observation, we propose IoT-Praetor, a system for automatically extracting the behavior baseline of IoT devices and detecting malicious behaviors. In IoT-Praetor, a new DUD model is proposed. DUD can formulate the device behaviors from two aspects: 1) device interaction behaviors and 2) network communication behaviors. Furthermore, we design and implement a behavioral security detection system of IoT devices by using DUD and automatic behavior extraction and monitoring.

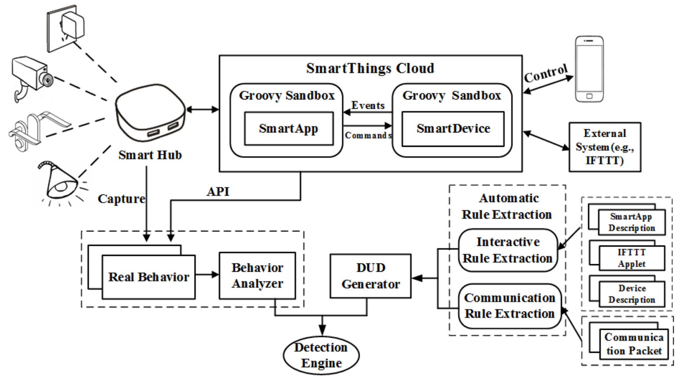


Fig. 1. System architecture of IoT-Praetor.

IoT-Praetor, the behavioral security detection system of IoT devices, is designed and implemented based on the Samsung SmartThings platform. The architecture of the system is shown in Fig. 1, which includes an automatic rule extraction module, a DUD generation module, and a behavior detection engine. The automatic rule extraction module consists of an interaction rule extraction module by analyzing SmartApps and IFTTT Applets, and a communication rule extraction module by analyzing communication packets of devices. After extracting rules by the automatic rule extraction module, the DUD generation module generates DURs from the extracted behavior rules based on DUD. Then, the behavior detection engine captures the real-time behaviors of devices and uses DUR as a baseline to detect the security of devices. In IoT-Praetor, the real-time behaviors of devices are obtained from the Smart Hub or using the SmartThings API to call device interfaces.

A. Automatic Rule Extraction Module

The automatic rule extraction module is used to automatically extract behavioral rules of devices. The module is divided into two submodules: 1) interactive rule extraction module and 2) communication rule extraction module. The interactive rule extraction module first uses crawling to obtain basic information of the current device from UI provided by SmartThings and IFTTT platform. Then, NLP tools are used to extract device interaction behavior rules from device information, SmartApps description, and IFTTT Applets rules description. The communication behavior rule extraction module obtains basic and hidden characteristics of device communication by analyzing network communication packets to extract device communication behavior rules.

B. DUD Generation Module

This module is used to generate expected DURs as a white list of device behavior. It can obtain the device behavioral

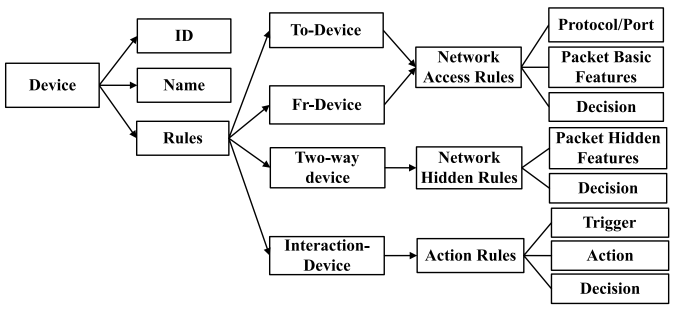


Fig. 2. Node structure of DUD.

rules from the automatic rule extraction module consisting of the interaction rule extraction module and communication rule extraction. Then, the DUD generation module translates the extracted information to the formulated DURs. The rules are specified by a YANG tree.

C. Behavior Detection Engine

By capturing device communication traffic on a router and using SmartThings API, we analyze the communication packets to obtain real-time behaviors of devices. Based on the Drools [29] rule engine, the behavior detection engine compares the real-time device behaviors with DURs to detect suspicious behaviors and performs automatic action, such as alarming or logging.

V. Device Usage Description

MUD is a draft standard for defining behaviors of IoT devices. However, it only includes some simple network access rules specified by device manufacturers, resulting in that MUD can only define very limited device communication behaviors. To solve the limitations of MUD, we propose a DUD model and define the whole system behavior rules of IoT devices. In IoT-Praetor, any undefined behaviors in the DUD file are not allowed. Just like the MUD file, DUD is based on YANG, and Fig. 2 shows the important node structure.

DUD includes the following elements.

A. ID

Each IoT device has a DUR file generated according to DUD to specify its behavior rules. The device ID is a unique identifier of each device in IoT-Praetor and indicates the correspondence between a device and a DUR file.

B. Name

The name field is usually a complex string generated by a combination of numbers and letters, indicating the device name corresponding to the DUR file.

C. Rules

Rules represent device behaviors, including Fr-device, to-device, two-way device, and interaction device. Fr-device and to-device represent incoming and outgoing device communication behavior rules, which are network access rules and specify the basic characteristics of the communication packets, such as communication direction, IP, protocol, port, etc. A two-way device includes hidden characteristics of communication packets, which are depicted in Section VI-B for details. An interaction device is used to represent the interaction behavior rules, which mainly include the "trigger and action" rules extracted from smart applications.

```
module: smart-device-dud
   +--rw dud
      | +--rw dud-version      uint8
      | +--rw last-update      yang:date-and-time
      | +--rw cache-validity?  uint8
      | +--rw is-supported     boolean
      | +--rw deviceinfo
      +--rw from-device-policy
      +--rw to-device-policy
      +--rw hidden-device-policy
      +--rw interaction-device-policy
      augment /acl:acls/acl:acl/acl:aces/acl:ace/
acl:matches/acl:l4/acl:tcp/acl:tcp:
        +--rw direction-initiated?  direction
      augment /acl:acls/acl:acl/acl:aces/acl:ace/
acl:matches/acl:l3/acl:ipv4/acl:ipv4:
        +--rw src-dnsname?  inet:host
        +--rw dst-dnsname?  inet:host
      augment /acl:acls/acl:acl/acl:aces/acl:ace/
acl:matches:
        +--rw packet-fre
        +--rw service-fre
        +--rw packet-order?  decimal64
        +--rw rule
          +--rw triggers* [id]
          +--rw device-action
```

Fig. 3. YANG tree structure of DUD.

The YANG tree is a simple graphical representation of the data model that can be automatically generated by the Pyang tool. Through the tree structure, node information in the YANG file is displayed more simply and clearly. Fig. 3 shows the simple DUD YANG tree structure.

Based on DUD, we use device ID as a key value to generate a YANG-based JSON file for each device as the DUR. Each rule file does not affect each other and works independently, which improves the performance and security of the system to a certain extent.

VI. Automatic Extraction of Device Behavior Rules

A. Automatic Extraction of Interaction Rule

In an IoT system, smart applications can specify the interactive behaviors of IoT devices by setting trigger–action rules to realize IoT automation. For example, SmartThings provides SmartApps to control IoT hardware devices. In addition, the IFTTT platform can be authorized to access API and set the corresponding IFTTT rules to control SmartThings devices. Based on the observations, we extract device interaction behavior rules from the following three aspects.

1) *Device Information:* In the SmartThings platform, SmartThings API is provided, which allows authorized

TABLE I
DEVICE INTERACTIVE RULES

Event	Device	Capability
Trigger	DeviceX	Attribute
Action	DeviceY	Command

```
{
     "Label":"Smart Light",
     "Date Dreated":"2018-12-30 8:00:38 AM UTC",
      "Settings":[{
           "Name": themotion,
           "Type": capability.motionSensor,
           "Value":{
                 "ID": 1d713e0a-48fb-4944-9737-053dec76d1cf,
                 "Name": Motion Sensor  }
               },{
           "Name": theswitch,
           "Type": capability.switch,
           "Value":{
                 "ID": 2385b786-cbcb-46ab-b710-5de0d9e293a8,
                 "Name": Hue color lamp 1 (Hue Extended Color) }
               }],
     " Event Subscriptions":{
           "Name": motion.active,
           "Handler": motionDetectedHandler,
           "Source":{
                  "ID": 1d713e0a-48fb-4944-9737-053dec76d1cf,
                  "Name": Motion Sensor }
     }
     }
```

Fig. 4. JSON information of Smart Light.

users to quickly and easily obtain information about IoT devices.

2) *SmartApps:* SmartApps include a description field that uses natural language to interpret the functionality of current SmartApp.
3) *IFTTT Rules:* IFTTT rules can define the interaction rules to control the device through third-party platforms.

As shown in Table I, the extracted device interaction rules include several key elements: trigger, action, device, and capability.

1) Extracting Rules From SmartApps: The description field of SmartApp includes the function definition of IoT devices. However, the information is relatively brief. Hence, it is difficult to extract the rules directly and the accuracy is low. Therefore, we combine the information provided by SmartThings API and UI, and then extract SmartApp rules to improve the accuracy of rule extraction. The SmartApp extraction process is divided into the following two steps.

SmartThings provides users with a UI as SmartThings IDE (https://graph.api.smartthings.com/), which can configure and view current system information, such as location, hub, device, Device Handler, and installed SmartApp. Taking Smart Light as an example, we crawled the main information from UI and converted it to a JSON format information as shown in Fig. 4.

The description contained in the settings field indicates device information. For example, the type field describes the capability used by SmartApp and the value field specifies

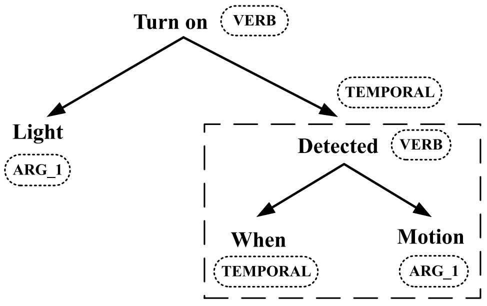


Fig. 5. SRL Tree (turn light on when motion is detected).

the device ID and name. As shown in Fig. 4, the capability of the trigger device involved in Smart Light is capability.motionSensor of Motion Sensor, and the capability of action device is capability.switch of Hue color lamp 1. The event subscription field indicates trigger information, including trigger condition, handler, and device. Similarly, the trigger device in Smart Light is Motion Sensor and the trigger condition is motion.active.

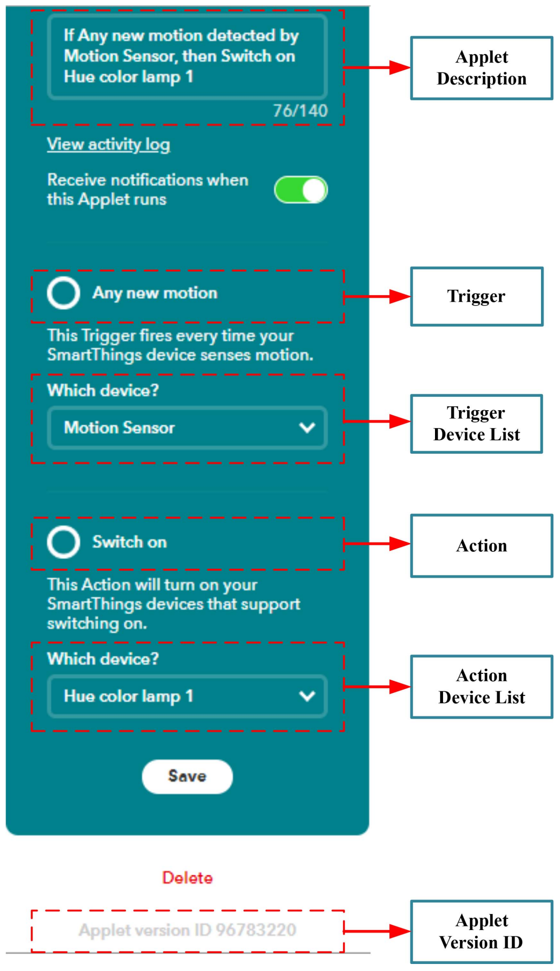
Through the above step, we successfully obtain a trigger and action device and its capability. Then, we can get all the possible commands from the Samsung capability document, such as Switch has on () and off (), as the command candidate set of the action device. This successfully narrowed the scope of the device, and then we used the NLP technology to analyze the SmartApp text description.

First, we use semantic role labeling (SRL) to analyze the SmartApp description. SRL is a shallow semantic analysis technique that assigns labels for words or phrases in sentences to express their relationship with predicates. An SRL classifies them into specific roles, such as core semantics and ancillary semantic roles.

After each sentence is analyzed, it generates a file in JSON format, which is represented as an SRL tree structure. For example, given the sentence, turn on light when motion detected, SRL analyzes and yields an SRL tree structure as shown in Fig. 5.

Two ARG_1 light and motion in the SRL tree represent the objects of capability. The two VERBs usually represent attributes or commands. We use AllenNLP [30] to achieve deep SRL processing. Then, by using the Word2Vec [31] model to analyze the similarity of words to determine the relationship between sentence words and related devices, we obtain which ARG_1 is the object of the capability of action device and which VERB is the command of capability. Similarly, Word2Vec is used to analyze the correlation between VERB and command candidate set of action devices so as to analyze the commands used by SmartApps. Because the words in SmartThings usually have special meaning, we use Samsung SmartThings capability documentation as a corpus for training.

2) IFTTT Rules Extraction: IFTTT can access SmartThings services and define Applets based on the "If this, then that" form. As shown in Fig. 6, on the IFTTT Web interface, we can view and configure Applets set in the current system, including version ID, description, devices, and capability of



If Any new motion detected by Motion Sensor, then Switch on Hue color lamp 1

76/140

View activity log

Receive notifications when this Applet runs

Any new motion

This Trigger fires every time your SmartThings device senses motion.

Which device?

Motion Sensor

Switch on

This Action will turn on your SmartThings devices that support switching on.

Which device?

Hue color lamp 1

Save

Delete

Applet version ID 96783220

Fig. 6. Configuration interface of an IFTTT Applet.

the Applets. Therefore, we can crawl the Applet's information from IFTTT UI.

The IFTTT platform provides 16 triggers and 6 actions of SmartThings. By comparing with the Samsung capability document, we associate the SmartThings service provided by each IFTTT with the capability of the device. In this way, the capability of the device can be obtained directly through the services provided by the IFTTT platform.

The Applet description information generally follows "If this, then that" paradigm, such as "If any new motion detected by Motion Sensor, then Switch on Hue color lamp 1." If a statement must contain a trigger, then the statement must contain action and a clear device name and capability. In the previous step, we get a device list, trigger, and action in Applets. Then, we analyze the If and Then statements separately and obtain the device name and a specific trigger and action phrase in the sentence. Therefore, comparing the elements of the sentence with the device list, we can obtain the device name, trigger, and action. The file in the JSON format generated by analyzing Applets is shown in Fig. 7.

Through the above step, we obtain commands or attributes corresponding to SmartThings services and devices provided by IFTTT applets so that the device interactive behavior rules of DUD are generated.

```
{
  "Description":"If Any new motion detected by Motion
  Sensor, then Switch on Hue color lamp 1",
  "TriggerDevice":"Motion Sensor",
  "Trigger":"Any new motion",
  "ActionDevice":"Hue color lamp 1",
  "Action":"Switch on"
}
```

Fig. 7. JSON information of the IFTTT Applet.

B. Automatic Extraction of Communication Rules

Except for the interactive actions from SmartApps and IFTTT applets which can control IoT devices and result in the undesired behaviors, network communication also may cause IoT devices to execute malicious behaviors.

Since IoT devices are generally designed with special purpose and have a small number of predictable traffic flows, their network communication behaviors are relatively simple and stable. By analyzing the real communication of IoT devices, we observe that they have the following main features: 1) destination IP address in the communication process is within a certain range; 2) the communication protocol and service type are relatively fixed; 3) the transmitted data content is relatively stable; and 4) the transmission frequency of communication packets has a certain regularity. Therefore, we capture communication packets of IoT devices in the router and analyze the communication behavior of packet header, such as IP, port, protocol, communication frequency, etc.

1) Data Set: The device communication traffic analyzed in this section comes from two parts.

1) A small IoT system built by ourselves, including three Samsung devices (Smart Hub, Motion Sensor, and Outlet), three cameras (Arlo Camera, Yi Camera, and Woshida Camera), and a Philips Hue Light. These devices are connected to the network through a router that has installed an OpenWrt system and contained tcpdump for capturing device traffic.
2) Due to the limited IoT devices, we have extended our research work using a public data set provided by Sivanathan *et al.* [32]. They build a smart environment with more than 28 different IoT devices including sensors, switches, lights, medical devices, cameras, and hubs. Beginning on September 23, 2016, the network traffic of devices were collected daily and published on the Internet. The size of the daily logs varies between 61 and 2 GB, with an average of 365 MB.

2) Basic Features: We first analyze the following basic features of IoT device traffic for extracting communication behavior rules.

1) *Communication Direction (From/To Device):* Normally, the information exchange of the IoT device is balanced when the device communicates with the cloud server. In the case of attack, such as DDoS, the device only sends packets without receiving reply packets. Hence, we should analyze communication behaviors from different directions, such as from device and to device.

TABLE II
IP AND PORT INFORMATION OF DEVICES

Device	Number	IP	Port
Yi Camera	8	139.129.76.123	443
		101.201.149.73	32100
		106.14.51.2	80
		······	443
Smart Hub	10	34.195.26.49	443
		13.124.2.28	80
		54.194.110.229	443
		······	80
TPLink Smart plug	7	52.220.7.83	50443
		54.254.250.149	50443
		······	50443
Triby Speaker	6	46.105.38.79	5228
		188.165.38.151	8090
		······	
Arlo	9	54.229.110.181	443
		34.242.116.196	443
		······	443
Philips Hue Light	8	34.249.77.109	80
		54.76.81.242	443
		130.211.67.12	443
		······	443
NEST Smoke Alarm	6	54.225.171.100	11095
		23.20.212.235	11095
		······	11095
Withings Smart scale	1	89.30.121.150	80

2) *Destination IP Address:* An important feature of the IoT device communication traffic is that the destination IP address set is usually fixed. Different from PC, which can communicate with a variety of network servers using different protocols, IoT devices only need to communicate with some specific servers. Therefore, the destination IP addresses rarely change.
3) *Port:* IoT devices only need to communicate with specific servers designed by the manufacturer to interact with specific ports. Moreover, in order to prevent security attacks such as a remote attack, devices usually close most of the ports. Therefore, ports used in the communication process of the IoT device are basically fixed and are not change at will. The attack traffic launched by malicious attackers usually uses diverse ports. Table II shows examples of IP and port information for eight IoT devices. It can be seen that the IP address used in the communication of the IoT device is usually within a certain range, and the ports are fixed.
4) *Protocol:* Because the IoT device service is relatively simple, protocols used in communication are relatively fixed. For example, it updates and downloads the firmware via HTTP; uses NTP to implement time synchronization and uses DNS to query server domain name. DNS is one of the most popular protocols for IoT devices. To distinguish different functions, the device communicates with servers using a different domain name that is resolved through DNS. The device performs DNS resolution only on a limited number of domain names (mainly vendor or server domain names). Malicious attackers can resolve any domain name arbitrarily, and there may be more than 300 domain names queried in a few hours. Table III shows the information of domain names used in the communication of six

TABLE III
DOMAIN INFORMATION OF DEVICES

Device	Number	Domain Name
Yi Camera	3	api.xiaoyi.com
		log.xiaoyi.com
		motiondetection.oss-cn-qingdao.aliyuncs.com
Smart Hub	3	pool.ntp.org
		api.smartthings.com
		fw-update2.smartthings.com
TP-Link Smart plug	4	uk.pool.ntp.org
		ca.pool.ntp.org
		devs.tplinkcloud.com
		······
NEST Smoke Alarm	4	frontdoor.nest.com
		weave-logsink.nest.com
		······
Arlo	11	ntp01.arlo.com
		mcs.arlo.com
		registration.arloxcld.com
		advisor.arloxcld.com
		······
Philips Hue Light	7	www2.meethue.com
		time2.google.com
		dcp.cpp.philips.com
		······
Triby Speaker	9	sip.aws.invoxia.io
		crossbar.invoxia.io
		0.invoxia.pool.ntp.org
		······
Withings Smart scale	1	Scalew.withings.net

devices. It can be seen that the number of domain names used by these IoT devices is very limited. In addition, devices use HTTP to update firmware and time synchronization. An HTTP request mainly includes requested methods, such as GET/HEAD/POST and requested URL, which is definite for completing a specific service.

3) Hidden Features: In addition to the above basic features, we have found that the network traffic of IoT devices has the following hidden features.

1) *Time Interval of Packets:* In order to complete specific services, IoT devices need to send packets within a fixed-time interval. For example, Yi Camera uses UDP protocol to maintain the heartbeat connection with the cloud server at a time interval of 25 s. Amazon Echo, Samsung SmartThings, and Belkin Wemo motion sensors send DNS requests every 5, 10, and 30 min [33] to query DNS with a fixed frequency. SmartThings, LiFX bulb, and Amazon Echo send NTP requests every 600, 300, and 50 s for time synchronization. However, malicious attackers usually send packets with a shorter and unstable time interval.
2) *Number of Packets:* IoT devices have requirements for traffic stability, and there is a limitation on the surged traffic of normal devices. But for attack traffic such as DDoS, the attackers hope to spend the minimum cost and establish as many network connections as possible in the shortest time to achieve the purpose of exhausting target server resources quickly. Hence, the number of packets over a period of time increases sharply compared

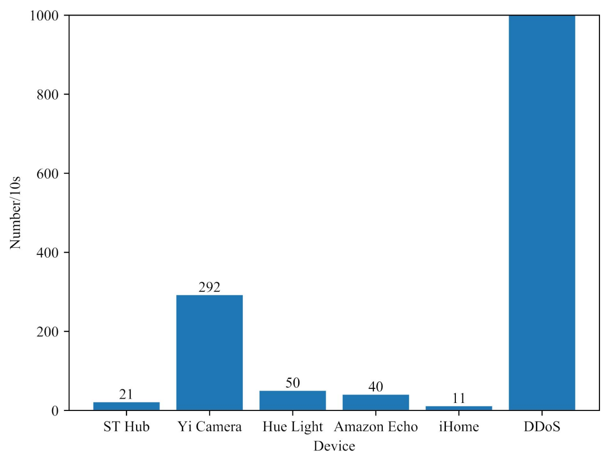


Fig. 8. Number of packets sent in 10 s.

TABLE IV
PACKET CHARACTERISTICS USED BY PACKET SEQUENCE

Description	Symbol	Value
direction	c_1	0 = from, 1 = to
local port type	c_2	binary value of local port type
remote port type	c_3	binary value of remote port type
protocol type	c_4	binary value of protocol type
packet size	c_5	binary value of packet size
connection flag	c_6	binary value of flag
time interval	c_7	binary value of packet interval time

with normal traffic. Fig. 8 shows the number of packets sent within 10 s of IoT devices, such as Smart Hub and Yi Camera. It can be clearly seen that among the five IoT devices, Yi Camera sends the largest number of packets within 10 s. However, in order to achieve the maximum attack effect in Mirai DDoS, the number of packets sent by Yi Camera and other devices in 10 s is as high as 20 000.

3) *Packets Sequence:* Packets of IoT devices generally follow certain sequence characteristics. For example, TCP flags should be SYN, SYN+ACK, ACK, PUSH, and FIN. However, the attack traffic does not follow the above sequence. SYN Flood only sends a large number of SYN packets to exhaust target server resources. In order to extract the feature of packets sequence, the device communication packets captured on the router are grouped into $pkt_1, pkt_2, \ldots$ Then, we extract packet header information, including direction, port, protocol, packet size, connection flag, and time from the packets. According to this information, each packet pkt_i is represented by a symbol as s_i that represents a seven-tuple $(c_1, c_2, c_3, c_4, c_5, c_6, c_7)$, as detailed in Table IV. Then, we calculate the probability p_i of each packet symbol s_i under the given sequence of k preceding symbols s_{i-k}, $s_{i-k+1}, \ldots, s_{i-1}$

$$p_i = P(s_i | < s_{i-k}, s_{i-k+1}, \ldots, s_{i-1} >).$$

Here, gated recurrent unit (GRU) [34] is used to estimate the probability of the next packet based on k previous packets symbols by inputting a pretrained model. GRU is a new type of RNN that provides similar accuracy to other RNN with a lower computational cost. By analyzing normal packet probability, we set a detection threshold β. If $p_i < \beta$, it is expressed as attack traffic and identified as malicious communication behavior of the device. In addition, when reading packets, it is necessary to record the number of packets sent by the device per minute and time interval of packets, then write them into the DUR file. Thereby, the communication behavior rule will be automatically extracted.

VII. MALICIOUS BEHAVIOR DETECTION

A. Monitoring IoT Device Behaviors

As we know, the Samsung SmartThings platform is a closed source, and we can only write SmartApp and Device Handler through IDE provided by SmartThings or use SmartThings API to obtain and configure basic information. The communication packets of devices are encrypted using sophisticated encryption algorithms and are extremely difficult to capture device behaviors through password cracking. Therefore, we monitor interactive behavior through SmartThings API and obtain communication behaviors by analyzing packets' header information in the router.

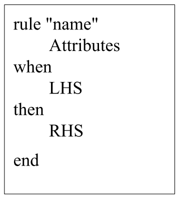
1) Monitoring Device Interactive Behaviors: The device information about location, device, Apps, and installedApps can be obtained through the SmartThings API. All the SmartThings resources are protected by OAuth2 bearer tokens and need to specify the scope of OAuth2 to grant user's permission. We get the states of devices in real time by using SmartThings API, then call the observable class of java.util toolkit and observer device interfaces to monitor device behaviors.

2) Monitoring Device Communication Behaviors: For the communication behavior of devices, we obtain real-time behavior of devices in real time by monitoring and analyzing the communication packets between devices and cloud servers in the router. The analysis process includes basic and hidden communication features about the device proposed in Section VI, such as destination IP address, port, and packets sequence. Then, communication behavior rules are generated automatically and written to device DUR file as a white list of communication behaviors.

We capture and analyze packets by calling pcap4j [52], which is a Java library for capturing, making, and sending packets. Pcap4j supports multiple protocol types and can add new protocol support without modifying the contents of the library itself. All of its built-in packet classes are serializable and thread safe. It can also dump and read files in the PCAP format.

B. Detection Engine

Rule detection engine detects whether the real-time behaviors of a device are consistent with the rules in DUR file, and then performs the corresponding operation, such as alert and logging. The detection engine of IoT-Praetor is based on Drools. Drools is an open-source rules engine written in Java. Drools is implemented based on the RETE pattern matching algorithm [35]. Its rule files are suffixed with ".drl" and are



```
rule "name"
    Attributes
when
    LHS
then
    RHS
end
```

Fig. 9. Basic elements of Drools rules.

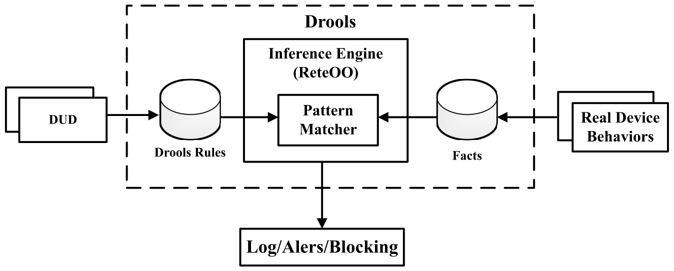


Fig. 10. Architecture of Drools-based detection engine.

written in the native language. The basic elements of each rule are shown in Fig. 9.

Each rule needs to use "name" as a unique identifier, which mainly includes three parts. Attributes are optional lists that may affect rules such as priority. The left-hand side (LHS) refers to a specific set of conditions, which can be composed of zero or more conditions. When LHS is empty, the system will return "true" for it. Multiple or different LHSs can be connected with "and" or "or." The right-hand side (RHS) refers to an operation that needs to be specified when all the conditions of LHS are met. It should be atomic and does not contain conditional code.

Fig. 10 shows the architecture of our detection engine. In Sections VI-A and VI-B, by analyzing device interactive behavior and communication behavior, DUD-based DURs are generated for the current devices of the system. We convert DUR into Drools standard rule and monitor real behaviors as input to our detection engine. Once a suspicious behavior is detected, the corresponding actions are performed, such as alarming or logging.

VIII. Implementation and Evaluation

A. Implementation

Our system is implemented based on Samsung SmartThings, which includes Smart Hub, SmartThings Motion Sensor, SmartThings Outlet, Philips Hue Light Suite, Arlo HD Smart Wireless Camera Kit, Yi 720P Camera and a Woshida Camer, and mobile phone with SmartThings Mobile App. All the devices are connected to the network through a Xiaomi wireless router (installing OpenWrt system), and the device IP address is assigned by DHCP in the network range of 192.168.1.0/24. The network configuration of the device is shown in Table V.

We evaluate the performance of our system using a physical machine with an Intel i7-6700 processor, 8-GB memory, and a 500-GB hard disk. The machine runs Ubuntu 14.04. We use Drools 6.3 development environment that is installed under

TABLE V
Hardware Device Configuration

Device	MAC address	IP
Mi Router	64:09:80:7D:48:E6	192.168.1.1
Smart Hub	D0:52:A8:63:4E:E4	192.168.1.180
Philips Hue Light	B0:D5:9D:1B:36:AB	192.168.1.160
Arlo Camera	A0:04:60:71:FBb:8F	192.168.1.233
Yi Camera	0C:8C:24:44:31:97	192.168.1.173
Woshida Camera	54:C9:DF:E5:5F:1E	192.168.1.110

TABLE VI
Accuracy of Interaction Rule Extraction

Rule Type	Source	Accuracy Rate
Interaction rule	50 SmartApps	90%
Interaction rule	30 IFTTT Applets	100%

MyEclipse10. NLP tools, Word2Vec, and neural network algorithm GRU used in the system are selected to install the corresponding libraries AllenNLP, gensim, and keras under Python 3.6.7.

B. Evaluation

IoT-Praetor is an undesired behavior detection system for IoT devices. It can work in a household setting or an enterprise setting. When we evaluate the performance of IoT-Praetor, we can deploy it in a smart home environment based on Samsung's SmartThings platform.

1) Effectiveness of Rule Extraction: The device rule extraction testing is divided into three parts: 1) SmartApp rules extraction; 2) IFTTT rules extraction; and 3) communication rules extraction.

1) *SmartApp Rules Extraction:* In order to evaluate the effectiveness of the SmartApp rules extraction method proposed in this article, we selected 50 SmartApps from SmartThings Public GitHub repository [36] for testing. Because hardware devices in this lab environment are limited, we created a virtual device corresponding to SmartApp using Device Handler in SmartThings Public GitHub repository. In 50 open-source SmartApps, we can correctly extract 45 (90%). Three of them cannot correctly be extracted because the description field is not detailed enough, there is no way to extract the capability involved in device correctly. The remaining two are Word2Vec errors in the judgment of similar words.
2) *IFTTT Rules Extraction:* Extracting IFTTT Applet needs to load them into the system. Otherwise, the list of devices involved in Applets cannot be obtained correctly. Some of the Applets about SmartThings are only related to one capability of the device, which does not involve interactive behaviors. Hence, we select 30 Applets related to our lab environment on IFTTT, then analyze and automatically extract the device interactive behaviors, and successfully extract all 30 IFTTT Applets. Table VI shows the accuracy of device interactive rule extraction.
3) *Communication Rules Extraction:* In order to test the effectiveness of communication rules extraction by analyzing device communication traffic, we test PCAP files generated by eight IoT devices and extracted the traffic

TABLE VII
COMMUNICATION RULE

Device	Basic	Hidden Characteristics		
		Number/10s	Packet frequency/protocol	Packet sequence
Yi Camera	23	392	25s/UDP	0.01
Arlo Camera	35	841	No	0.01
Smart Hub	10	21	10min/DNS,5min/NTP	0.01
Hue Light	33	50	30s/NTP	0.01
Amazon Echo	66	39	5min/DNS, 50s/NTP	0.01
LiFX Bulb	15	43	50s/NTP	0.01
Wemon motion sensor	35	23	30min/DNS	0.01
Nest Smoke alarm	32	44	No	0.01

TABLE VIII
TESTING OF INTERACTION RULES

Operation	Number	Malicious Behavior	Detection rate
Interactive behavior rules	20 SmartApps 10 IFTTT Applets	Over-privileged Accesses	96%
		Event Spoofing	93%

rules. Other data come from device data set provided by Sivanathan *et al.* [32]. Table VII shows communication rules information generated by each device, including the number of basic feature rules and three hidden feature rules.

2) Interactive Behavior Detection: Fernandes *et al.* [25] summarized several security vulnerabilities in the design of the capability model and the event system of SmartThings, which may cause SmartApps to perform malicious behaviors resulting in security issues in the IoT system. It includes the following two aspects.

1) *Over-Privileged Accesses:* SmartThings have a coarse-grained capability model for SmartApps. A SmartApp that only needs an attribute or command can access the entire capability so that the SmartApp that is authorized to access the capability of a device can access all capability of this device. Therefore, a malicious SmartApp only requires access to some of the permissions, and it is also possible to access and control the entire device.
2) *Event Spoofing:* Each device connected to a hub is assigned a 128-b device identifier. Once a SmartApp obtains the identifier, it can spoof all events of the device without accessing any capability and pass it to all SmartApps who subscribe to related capability.

There is no public malware data set about SmartApp on the SmartThings. In order to test interactive rules and exclude SmartApps with inaccurate description, we customized 20 SmartApps, including five SmartApps with over-privileged accesses, five SmartApps with event spoof, and ten normal SmartApps. We used ten misbehaving IFTTT Applets. The interactive behavior test results are shown in Table VIII.

During the testing, we manually triggered each malicious SmartApp 20 times, in which the detection rate is represented the number of malicious behaviors/total malicious behavior. The result of over-privileged accesses is 96%. For example, SmartApp named "Smart Light" who has excessive privileged behavior for accessing off() command can be successfully detected. The detection rate of the event spoof is 93%. The false-positive rate of interactive behavior detection is 1.6%. The main factor affecting the detection rate is a time offset existing during monitoring device behaviors, which leads

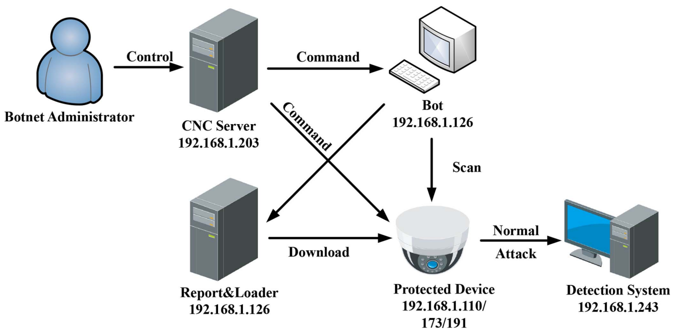


Fig. 11. Simulated lab environment.

to imprecisely detect the triggering operation relationship between devices behaviors.

3) Communication Behavior Detection: We verify the functionality of device communication behavior rules by simulating Mirai Botnet attacks because Mirai is open-source available and is the basis for other main types of IoT malicious software. Our attack data set was collected from three devices. The simulated experimental environment is shown in Fig. 11.

We created the CNC server, Report&Loader server, and infected bots by creating three virtual machines in VMware. The IP address of the CNC server is 192.168.1.203. The administrator sends attack commands to bots by controlling the CNC server. The IP of Report&Loader server is 192.168.1.183, which is used to receive the crack result and download malicious programs to the attack device. The IPs of Bot devices are 192.168.1.110, 192.168.1.173, and 192.168.1.191, which receive attack commands from the CNC server and launch DDoS attack. The IP of the behavior detection system is 192.168.1.243, running our detection system, analyzing the device traffic, and determining whether device behavior is legal.

We tested three vulnerable devices by launching a Mirai attack. As we can see from Section VI-B, the maximum number of packets sent by IoT devices per 10 s is limited. For example, the maximum number of packets for Arlo Camera is 841/10 s. Hence, we set up a $w = 1000$ packet window. According to the types of packets, we divide Mirai into four different stages. Before infection, bots brutely crack infected devices with username and password dictionary. Once successfully cracked, the loader remotely logs in the device and determines the device environment. After infection, the loader uses wget, tftp, and echo methods to download the Mirai malicious program to an infected device. During the scanning phase, the infected device initiates a scanning crack to find other vulnerable devices in the network. In the DDoS attack phase, the infected device receives a command from the CNC server and sends ten different types of attacks to the target server. This document selects five attack modes: 1) UDP Flood; 2) SYN Flood; 3) ACK Flood; 4) DNS resolver Flood; and 5) Http Flood. Table IX shows the results of malicious communication behavior detection.

Before infection, we repeatedly infected each vulnerable device 20 times, each time generating about 2000 attack packets that are mainly Telnet packets. The detection rate was

TABLE IX
TESTING OF COMMUNICATION RULES

Device	Mirai				Average detection rate
	Before infection	After infection	Scanning	DDoS	
Yi 720p	100%	99%	98%	97%	98.5%
Woshida Camera	100%	98%	100%	99%	99.3%
TP-link Router	100%	99%	100%	97%	99%

100% because 53 ports were not allowed in the rules file. After infection, wget was used in this experiment, and each infection-prone device was repeatedly infected 20 times, each time generating about 700 attack packets, which are mainly TCP, Http, and Telnet packets. In the scanning phase, we set each device to perform scanning for 3 min, generating more than 100 000 packets per device. During the DDoS attack phase, each attack lasts for 3 min while each device generates 10 million attack packets.

In the process of communication behavior detection, the attack detection rate is over 98% and the false positive rate is zero. These results show that IoT-Praetor does not introduce false alarms. To extract the feature of packets sequence, IoT-Praetor uses the GRU model to estimate the probability of the next packet based on k previous packet symbols by inputting a pretrained model. To determine appropriate values for the detection threshold and anomaly triggering threshold, we evaluated the system accuracy using the normal data set and the attack data set. There are two reasons why a few packets are not detected correctly: 1) the detection threshold in the GRU model may have a slight deviation and 2) the automatic rule extraction module cannot perfectly obtain the normal and abnormal communication rules of all packets.

C. Performance

To evaluate additional overhead generated by the behavior detection system, we test the performance from the following aspects: 1) preprocessing performance includes the time for automatic extraction and generation of rules and 2) system running performance.

1) Preprocessing Performance: The preprocessing performance evaluation includes the rules extraction time for SmartApps, IFTTT Applets, and communication traffic.

1) *SmartApp Rules Extraction:* To test the performance of extracting device interaction behavior rules from SmartApps, we calculated the time of 50 open-source SmartApps and tested ten times per SmartApp. Thus, the average time of SmartApp extraction is 7 s.
2) *IFTTT Rules Extraction:* In order to test the time of IFTTT Applets extraction, we also performed ten times automatic extractions on 30 IFTTT Applets, resulting in the average time overhead of 363 ms.
3) *Communication Behavior Rule Extraction:* We capture device traffic in a trusted environment for 15 min. During this time, we repeatedly perform various operations to the devices on mobile apps. Then, the normal communication traffic of the devices is taken as an input and a DUD-based device communication behavior rule is generated. In this process, the time of communication behavior rule generation is about 2520 ms.

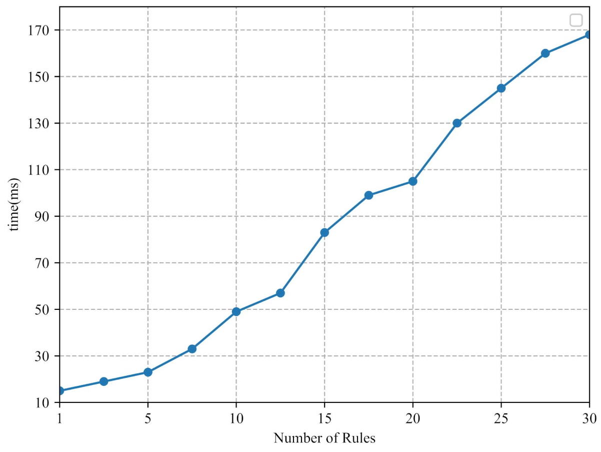


Fig. 12. Time cost of device interaction rule detection.

The time of preprocessing device behavior rules is mainly to obtain trusted device communication behaviors and capture the network communication traffic, but the preprocessing operation of rule generation is a one-time cost and can be completed in the initial stage of the system. So the performance overhead is acceptable.

2) System Performance: To evaluate the system run-time performance, we extracted 30 "trigger–actions" rules from SmartApps and Applets to generate DURs. Then, our system detects whether interactive behaviors violate the DUR white list by monitoring the changes of device states. Because interactive rules of devices can be deleted, modified, and added according to the needs of users, the rules may change in quantity over time. Therefore, we tested the behavior detection time by setting a different number of interactive rules. As shown in Fig. 12, when the number of rules increases, the system running time also increases.

We import traffic to the local host through port mirroring in the router, classify the traffic according to the MAC addresses of the devices, and start multithread to detect the communication behavior of different devices. The average time of detection for each packet is approximately 124 ms. The time consumption is mainly caused by monitoring and analyzing the communication packets between devices and cloud servers in the router, including capturing the real-time packets, extracting the basic and hidden communication behaviors of devices, and comparing the real-time communication behaviors of devices with DUD communication rules. Due to the large number and complexity of Drools rules, the DUR translation and matching process introduces a lot of performance overhead. The communication behavior rule is generated based on a large amount of communication traffic in a trusted environment. Generally, the rule is basically fixed and will not change excessively with time.

D. System Security Analysis

DURs include the information of communication behavior and interactive behavior rules, which is the security baseline for ensuring device security in our system. Its security is very important and must ensure that the file cannot be tampered with. An effective solution is provided in MUD to guarantee the security of the rule file, which we can draw on for DUD. Singing the DUD file by using the CMS message encryption algorithm (cryptographic message syntax), and stored in the designated location and transmitted using "application/pkcs7-signature" content type. In addition, the validity time of the rule file signature must be set. Once it expires, we need to resign the file. Before using a DUR file, the system must retrieve the signed value and verify the signature.

The key to detecting device interactive behavior is whether the rules contained in the DUR file are correct. In our system, SmartApps and IFTTT Applets are often signed and accessed by the administrator through OAuth2 protocol authentication. In addition, the process of automatic extraction of rules is a one-time effort and can be completed offline.

The automatic extraction of communication behavior rules is implemented by capturing device traffic in a clean environment where the device connects the network. Meanwhile, device communication behavior has strong stability. Under normal circumstances, it will not change too frequently. Therefore, it is secure that we extracted communication behavior rules at the beginning of the device joining the network.

Once malicious behaviors have been detected, the following actions can be taken. First, the detection engine can block malicious behaviors in the smart hub. Then, the violation information can be written to the system log of the smart hub or sent to device administrator for further analysis.

IX. Discussion

IoT-Praetor is a DUD-based behavior detection system for IoT devices. It is designed and implemented based on Samsung's SmartThings platform. Although our approaches can be applied to other IoT platforms, it still needs some efforts to modify the system design and implementation modules according to the features of each platform. In the future, we plan to extend our approaches to support more platforms.

The evaluation results show that IoT-Praetor can effectively detect malicious behaviors of IoT devices and cause small performance overhead. However, due to the very high real-time requirements for IoT devices, the delay time still needs to be reduced. Therefore, the efficiency of the behavior detection engine and the performance overhead of the system should be further improved.

Finally, the extraction of device communication behavior rules is based on the assumption that device communication traffic is captured in a clean environment when devices first join networks at an initial stage. Although the device communication usually does not change too much in an entire life cycle, the device update sometimes may change the corresponding network communication rules, such as IP addresses of cloud servers. Hence, how to capture the update and successfully reinstall DUR rules when the communication rules in the DUR need to be updated, should be considered in the future work.

X. Related Work

In recent years, with the booming of IoT, its security issue [37]–[40] has become one of the hot topics in academic and industrial research. Antonakakis *et al.* [41] analyzed the principle, propagation model, and attack mode of malicious code Mirai, which is a kind of DDoS attack on IoT devices. Cao *et al.* [42] analyzed the attack mode and attack process of Mirai in detail. They designed and implanted "white" Mirai in IoT devices, expelled vulnerable ports occupied by other malicious Mirai, and maintained heartbeat connection with manufacturers to achieve a secure Mirai defense system. Costin *et al.* [43], [44] performed static and dynamic security analysis on a large number of IoT firmware and highlighted several important security challenges in future research.

In addition, there are many works focusing on the security of SmartApps. Fernandes *et al.* [25] used Samsung SmartThings as an example to conduct an in-depth security analysis of a smart home platform and discovered security issues such as excessive privileges. By exploiting this vulnerability, pincode of door lock can be leaked through malicious monitoring battery SmartApp, causing serious security hazards. Then, Fernandes *et al.* continued to propose FlowFence [18], by using the information flow tracking technology in IoT, users who obtain rights can use data safely and legally. SmartAuth [17] proposes a user-centric, semantic-based intelligent authorization system. It collects security-related information from IoT App descriptions, code and comments automatically, and generate user authorization interfaces to enhance the generation of existing platform security policies. IoTMon [45] discovers the possible physical interactions across IoT applications and assesses the safety risk of each discovered interapp interaction. IoTGuard [46] collects an app's information at runtime using a code instrumenter that adds extra logic to the app's source code and checks behavior violations by comparing it with predefined policies. However, the policies in IoTGuard are defined and labeled manually. In addition, IoTGuard only considers the behaviors of apps and does not consider the communication behavior of IoT devices. HoMonit [16] monitors SmartApps from encrypted wireless traffic by comparing the SmartApps activities inferred from the encrypted traffic with their expected behaviors dictated in their source code or UI interfaces. However, Homonit needs the specific hardware-based wireless sniffers to monitor the encrypted wireless traffic so as to infer the state transition of IoT device behaviors.

Comparing the previous work, our work proposes a new DUD model for building IoT device behavior security specifications, which comprehensively considers the device interactive behaviors and communication behaviors. Furthermore, we present the behavior extraction method for SmartApps, IFTTT applets, and communication traffic. In addition, we design and implement a behavioral monitoring system based on Drools and a detection engine.

XI. Conclusion

In this article, we proposed IoT-Praetor, a DUD-based behavior detection system for IoT devices. Aiming to specify the fine-grained behaviors of the whole system over the entire lifecycle for IoT devices, we presented a new DUD model, which can define the interaction behavior rules and communication behavior rules of devices. Moreover, we proposed automatic extraction approaches based on crawling and NLP to automatically extract and generate DURs as a white list. By monitoring device behaviors in real time, we designed a device behavior detection engine based on Drools. Finally, we implemented and evaluated our system on the Samsung SmartThings platform. Our evaluation results showed that IoT-Praetor can effectively detect the malicious behaviors of IoT devices and cause small performance overhead.

REFERENCES

[1] *Wink: A Simpler, Smarter Home*, Wink, New York, NY, USA, 2018. [Online]. Available: https://www.wink.com/

[2] *SmartThings*, SmartThings, Mountain View CA, USA, 2017. [Online]. Available: https://www.smartthings.com/

[3] *HomeKit*, Apple HomeKit, Cupertino, CA, USA, 2018. [Online]. Available: https://developer.apple.com/homekit/

[4] *AWS IoT Documentation*, AWS Serv., Seattle, WA, USA, 2016. [Online]. Available: https://aws.amazon.com/de/documentation/iot/

[5] *openHAB*, openHAB, San Francisco, CA, USA, 2018. [Online]. Available: https://github.com/openhab/openhab/wiki

[6] *Home Assistant*, Home Assist., San Diego, CA, USA, 2018. [Online]. Available: https://www.home-assistant.io/

[7] *IFTTT Documentation*, IFTTT, San Francisco, CA, USA, 2016. [Online]. Available: https://ifttt.com/

[8] S. Notra, M. Siddiqi, H. H. Gharakheili, V. Sivaraman, and R. Boreli, "An experimental study of security and privacy risks with emerging household appliances," in *Proc. IEEE Conf. Commun. Netw. Security*, San Francisco, CA, USA, 2014, pp. 79–84.

[9] T. Oluwafemi, T. Kohno, S. Gupta, and S. Patel, "Experimental security analyses of non-networked compact fluorescent lamps: A case study of home automation security," in *Proc. LASER (LASER)*, 2013, pp. 13–24.

[10] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart locks: Lessons for securing commodity Internet of Things devices," in *Proc. 11th ACM Asia Conf. Comput. Commun. Security*, 2016, pp. 461–472.

[11] D. Formby, P. Srinivasan, A. Leonard, J. Rogers, and R. A. Beyah, "Who's in control of your control system? Device fingerprinting for cyber-physical systems," in *Proc. Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2016, pp. 1–15.

[12] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms," in *Proc. 28th USENIX Conf. Security Symp.*, 2019, pp. 1133–1150.

[13] Q. Wang, Y. Zhang, X. Lu, Z. Wang, Z. Qin, and K. Ren, "Real-time and spatio-temporal crowd-sourced social network data publishing with differential privacy," *IEEE Trans. Depend. Secure Comput.*, vol. 15, no. 4, pp. 591–606, Jul./Aug. 2018.

[14] Q. Zou, Y. Wang, Q. Wang, Y. Zhao, and Q. Li, "Deep learning-based gait recognition using smartphones in the wild," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3197–3212, Apr. 2020, doi: 10.1109/TIFS.2020.2985628.

[15] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[16] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "HoMonit: Monitoring smart home apps from encrypted traffic," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2018, pp. 1074–1088.

[17] Y. Tian *et al.*, "SmArtauth: User-centered authorization for the Internet of Things," in *Proc. 26th USENIX Conf. Security Symp. (USENIX Security)*, 2017, pp. 361–378.

[18] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "FlowFence: Practical data protection for emerging IoT application frameworks," in *Proc. 25th USENIX Conf. Security Symp. (USENIX Security)*, 2016, pp. 531–548.

[19] E. Lear, R. Droms, and D. Romascanu, "Manufacturer usage description specification (work in progress)," Internet Eng. Task Force, Internet Draft, 2018.

[20] M. Prospero. (2020). *The Best Smart Home Hubs of 2020*. [Online]. Available: https://www.tomsguide.com/us/best-smart-home-hubs,review-3200.html

[21] *Zapier: Connect Your Apps And Automate Workflows*, Zapier, San Francisco, CA, USA, 2018. [Online]. Available: https://zapier.com/

[22] *Apiant: Connect Your Apps, Automate Your Business*, Apiant, Philadelphia, PA, USA, 2018. [Online]. Available: https://apiant.com/

[23] A. Hamza, H. H. Gharakheili, and V. Sivaraman, "Combining MUD policies with SDN for IoT intrusion detection," in *Proc. Workshop IoT Security Privacy*, 2018, pp. 1–7.

[24] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and analysis of hajime, a peer-to-peer IoT botnet," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2019, pp. 1–15.

[25] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proc. IEEE Symp. Security Privacy (SP)*, San Jose, CA, USA, 2016, pp. 636–654.

[26] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Spot me if you can: Uncovering spoken phrases in encrypted voip conversations," in *Proc. IEEE Symp. Security Privacy (SP)*, Oakland, CA, USA, 2008, pp. 35–49.

[27] Z. Zhou, W. Diao, X. Liu, and K. Zhang, "Acoustic fingerprinting revisited: Generate stable device ID stealthily with inaudible sound," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2014, pp. 429–440.

[28] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "DolphinAttack: Inaudible voice commands," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 103–117.

[29] M. Proctor, M. Neale, P. Lin, and M. Frandsen, *Drools Documentation*, vol. 5, JBoss, Atlanta, GA, USA, 2008.

[30] M. Gardner *et al.*, "AllenNLP: A deep semantic natural language processing platform," 2018. [Online]. Available: arxiv:1803.07640.

[31] X. Rong, "Word2Vec parameter learning explained," 2014. [Online]. Available: arxiv:1411.2738.

[32] A. Sivanathan *et al.*, "Characterizing and classifying IoT traffic in smart cities and campuses," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Atlanta, GA, USA, 2017, pp. 559–564.

[33] P. Hunt, "Chain grant type for oauth2," Int. Telecommun. Union, Internet Draft, 2012.

[34] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. [Online]. Available: arxiv:1412.3555.

[35] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," in *Artificial Intelligence*, Elsevier, 1989, pp. 547–559.

[36] *Smartthings Public Github Repo*, SmartThings, Mountain View CA, USA, 2016. [Online]. Available: https://github.com/SmartThingsCommunity/SmartThingsPublic

[37] B. Ur, J. Jung, and S. Schechter, "The current state of access control for smart devices in homes," in *Proc. Workshop Home Usable Privacy Security (HUPS)*. 2014, pp. 209–218.

[38] J. Valente and A. A. Cárdenas, "Using visual challenges to verify the integrity of security cameras," *Challenge*, vol. 1, no. 4, p. 2, 2015.

[39] *The Internet of Things: Security Research Study*, Veracode, Burlington, MA, USA, 2015. [Online]. Available: https://www.veracode.com/veracode-study-reveals-internet-things-poses-cybersecurity-risk

[40] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-Things," in *Proc. 14th ACM Workshop Hot Topics Netw.*, 2015, p. 5.

[41] M. Antonakakis *et al.*, "Understanding the mirai botnet," in *Proc. 26th USENIX Conf. Security Symp.*, 2017, pp. 1093–1110.

[42] C. Cao, L. Guan, P. Liu, N. Gao, J. Lin, and J. Xiang, "Hey, you, keep away from my device: Remotely implanting a virus expeller to defeat mirai on IoT devices," 2017. [Online]. Available: arxiv:1706.05779.

[43] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *Proc. 23rd USENIX Conf. Security Symp. (USENIX Security)*, 2014, pp. 95–110.

[44] A. Costin, A. Zarras, and A. Francillon, "Automated dynamic firmware analysis at scale: A case study on embedded Web interfaces," in *Proc. 11th ACM Asia Conf. Comput. Commun. Security*, 2016, pp. 437–448.

[45] W. Ding and H. Hu, "On the safety of IoT device physical interaction control," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2018, pp. 832–846.

[46] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2019, pp. 1–15.

Juan Wang (Member, IEEE) received the M.E. and Ph.D. degrees from the School of Computer Science, Wuhan University, Wuhan, China, in 2004 and 2008, respectively.

She is an Associate Professor with the School of Cyber Science and Engineering, Wuhan University. In 2018 and January 2010, she was a Visiting Scholar with Pennsylvania State University, State College, PA, USA, and Arizona State University, Tempe, AZ, USA. Her research has been supported by NSF projects. She has authored and coauthored over 40 papers and holds 10 patents in security areas. Her current research interests include cloud and IoT security, trust computing, and SDN and NFV security.

Shirong Hao received the bachelor's degree in information security from Wuhan University, Wuhan, China, in 2017, where she is currently pursuing the master's degree with the School of Cyber Science and Engineering.

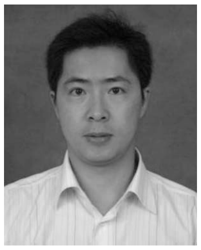
Her research interests include NFV, system security, and IoT security.

Ru Wen received the bachelor's degree in information security from Wuhan University, Wuhan, China, in 2016, where she is currently pursuing the master's degree with the School of Cyber Science and Engineering.

Her research interests include software-defined network, system security, and IoT security.

Boxian Zhang is currently pursuing the bachelor's degree with the School of Cyber Science and Engineering, Wuhan University, Wuhan, China.

His research interests include software security, Internet security, and IoT security.



Liqiang Zhang received the Ph.D. degree in information security from Wuhan University, Wuhan, China, in 2018.

He is an Assistant Professor with Cyberspace Science and Engineering School, Wuhan University. His current research interests include trusted computing, software analysis, AI security, and system evaluation.

Hongxin Hu (Member, IEEE) received the Ph.D. degree in computer science from Arizona State University, Tempe, AZ, USA, in 2012.

He is an Associate Professor with the Division of Computer Science, School of Computing, Clemson University, Clemson, SC, USA. He has published over 100 refereed technical papers, many of which appeared in top conferences and journals. His current research spans security, privacy, networking, and systems.

Dr. Hu received the NSF CAREER Award in 2019. He was a recipient of the Best Paper Award from ACM SIGCSE 2018 and ACM CODASPY 2014 and the Best Paper Award Honorable Mention from ACM SACMAT 2016, IEEE ICNP 2015, and ACM SACMAT 2011.

Rongxing Lu (Senior Member, IEEE) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, in 2012.

He is currently an Associate Professor with the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Fredericton, NB, Canada. Before that, he worked as an Assistant Professor with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, from April 2013 to August 2016. He worked as a Postdoctoral Fellow with the University of Waterloo from May 2012 to April 2013. His research interests include applied cryptography, privacy enhancing technologies, and IoT-big data security and privacy. He has published extensively in his areas of expertise.

Dr. Lu was a recipient of the 8 best (student) paper awards from some reputable journals and conferences. He was awarded the most prestigious Governor General's Gold Medal and won the 8th IEEE Communications Society (ComSoc) Asia–Pacific Outstanding Young Researcher Award in 2013. He is currently a Senior Member of IEEE Communications Society. He currently serves as the Vice-Chair (Conferences) of IEEE ComSoc Communications and Information Security Technical Committee. He is the Winner of 2016–2017 Excellence in Teaching Award, FCS, UNB.