IFIP Networking 2024 1570998142

Supervising Smart Home Device Interactions: A Profile-Based Firewall Approach

François De Keersmaeker, Ramin Sadre, Cristel Pelsser
UCLouvain, *ICTEAM*
Louvain-la-Neuve, Belgium
{francois.dekeersmaeker,ramin.sadre,cristel.pelsser}@uclouvain.be

***Abstract*—Despite their ubiquity, the security of Internet of Things devices is unsatisfactory, as demonstrated by several attacks. The IETF's MUD standard aims to simplify and automate the secure deployment of network devices. A MUD file specifies a device-specific description of allowed network activities (e.g., allowed IP ports or host addresses) and can be used to configure for example a firewall. A major weakness of MUD is that it is not expressive enough to describe device interactions, which often occur between devices in modern Smart Home platforms.**

In this article, we present a new language for describing such traffic patterns. The language allows writing device profiles that are more expressive than MUD files and can describe complex traffic patterns. We show how these profiles can be translated to efficient code for a lightweight firewall. We evaluate our approach on traffic generated by various Smart Home devices, and show that our system can accurately block unwanted traffic while inducing negligible latency.

***Index Terms*—IoT, smart home, security, firewall, device profiling, home automation**

I. Introduction

Cyberattacks against Smart Home IoT devices are a major concern for the privacy and security of their owners. Attackers could, for example, access IP cameras, sabotage smoke sensors, or manipulate smart locks. Unfortunately, the current state of the security of IoT networks in general and Smart Home networks in particular is unsatisfactory. Due to resource constraints, it is often difficult to run state-of-the-art security solutions, such as host-based intrusion detection systems, on IoT devices [18]. In addition, the devices are for easy use by non-expert, so it often depends entirely on the manufacturer whether and how often security updates are applied and best practices are followed. Finally, their large number makes it possible for attackers to misuse them to perform powerful attacks against other Internet hosts [6]. Improving the security of Smart Homes has therefore become an urgency.

An assumption which is present in numerous research works [9], [25], [10] is that an IoT device has a network behavior governed by a limited set of network access patterns that are simpler than what can be observed for general purpose devices such as PCs. Consequently, one can express the device's expected network behavior in a compact form that we will call a *profile* in the following.

ISBN 978-3-903176-63-8 © 2024 IFIP

Leveraging this concept, the Internet Engineering Task Force (IETF) defined the Manufacturer Usage Description (MUD) standard in RFC 8520 [17]. A MUD file is a description of the network connections allowed for a device. It specifies one or more traffic access control lists based on basic traffic features such as IP addresses, ports, connection direction, etc. The MUD file of a device can be used to configure local firewalls and switches such that the device can perform its normal functions, while other activities are blocked. Suggestions exist to extend MUD to support traffic statistics [16], such as packet rate or count, and some application-layer protocols [19].

A major weakness of MUD and its extensions is the fact that they do not sufficiently cover the reality of today's Smart Home networks [21]. In the latter, devices can interact with each other directly or through one or more intermediate hosts. As a result, communication patterns emerge, consisting of temporally and logically interdependent connections. Such patterns cannot be described in MUD at all, and, consequently, malicious activities breaking those patterns cannot be blocked, which makes unauthorized device control and data exfiltration attacks possible [27], [28].

In this article, we present a system to efficiently enforce legitimate device interactions in Smart Home networks. Our contributions are summarized as follows:

We propose a new language to express profiles that model legitimate network communication patterns of IoT devices. The language can express dependencies between traffic connections caused by device interactions.

We describe a firewall that leverages NFTables and NFQueue, lightweight firewall software available by default in Linux, to protect Smart Homes and a tool to automatically translate our device profiles into code for the firewall.

We evaluate our approach on different types of Smart Home devices and show that our firewall correctly and efficiently forwards legitimate traffic while it blocks deviating traffic patterns with little overhead.

We publish the source code of our translation tool and firewall, the profiles that we wrote for the tested devices, and the traffic traces used in the evaluation, at https://github.com/fdekeers/smart-home-firewall.

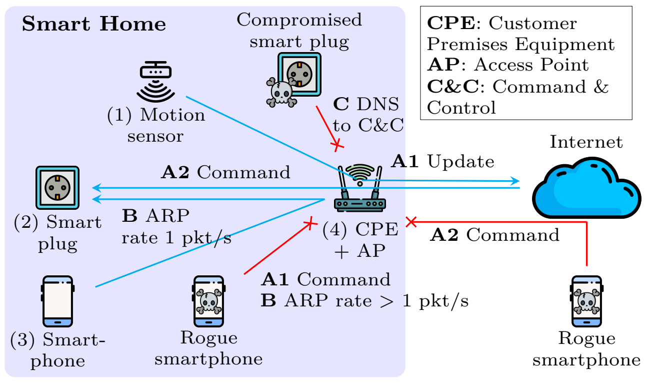


Fig. 1: Example smart home network, including attacks

It should be noted that it is not the aim of this paper to prove that IoT device communications follow patterns that can be expressed in machine-readable descriptions. This has been discussed in the existing literature, to which we will refer in the relevant sections of this paper.

This article is organized as follows. Section II gives the motivation for our work and the considered threat model. We describe our system in Section III. Section IV presents the experimental setup for the evaluation of our approach, followed by Section V which presents our results. Section VI discusses the strength and limitations of our system. Section VII presents related works. The article concludes in Section VIII.

II. Motivation and threat model

We introduce Smart Home networks and the concept of device interactions, potential attacks, the limitations of MUD, and the threat model used in the rest of the paper.

A. Smart Homes and device interactions

Smart Homes involve everyday household objects, from light bulbs to washing machines, enhanced with the capabilities of sensing (and, in some cases, acting), and communication. Such “smart” objects are connected to the Local Area Network (LAN) of the home, allowing them to communicate with each other, with other IT devices (e.g., laptops or smartphones), and with hosts in the Internet. The ones capable to speak the TCP/IP protocols can be connected through Wi-Fi or Ethernet. Some devices might only speak protocols specifically designed for local IoT communication, such as Zigbee [8], and need a gateway to communicate with IP devices.

Fig. 1 shows a Smart Home network containing: (1) a motion sensor and (2) a smart plug, both using Wi-Fi; (3) a smartphone (also using Wi-Fi) to control the Smart Home devices with a vendor-specific companion app; (4) a customer premises equipment (CPE) supplied by the ISP that acts as a wireless access point and router.

Device interactions are a core principle of Smart Homes. To perform home automation tasks, devices need to seamlessly communicate with each other. In our example, the user might want that the smart plug turns on when the motion sensor detects movement. On network level, the following activities take place: (A1) The motion sensor indicates a movement detection to the cloud; (A2) The cloud processes the message, and issues a command toward the smart plug. In this interaction, the message that actually changed the plug’s state is the second one. However, this activity could not have occurred without the first one. Obviously, the network activities carry a certain semantic and therefore happen in a specific order between specific hosts. While a traditional firewall is limited to either allow or block each activity, any time and in any possible order, *an interaction-aware firewall, as proposed in this paper*, can block communications that break the expected pattern.

B. Attacks and the limitations of MUD

The Manufacturer Usage Description (MUD) standard was defined by the IETF [17] with the goal of exhaustively describing the intended network behavior of devices. In practice, a MUD file consists of a set of network Access Control Lists (ACL) which specify authorized network traffic. MUD files are machine-readable and are intended to ease the configuration of firewalls and other network equipment.

In the following, we give examples of the type of attacks we are interested in and explain why MUD is not sufficient to describe the legitimate behavior of the devices.

Our first example showcases the concept of interactions. For the network shown in Fig. 1, an attacker who has hacked the cloud account of the Smart Home user can send commands to the smart plug. In MUD, we have to allow the connection between the cloud and the smart plug since it can occur in legitimate interactions. The fact that the communication should be allowed only if it is preceded by a communication between the motion sensor and the cloud cannot be expressed in MUD. In this paper, we propose a profile language that can describe legitimate communication patterns arising from device interactions, and a firewall that can enforce them, therefore blocking any smart plug command that was not preceded by traffic from the motion sensor. Other attacks that would now be blocked in such configuration are: an attacker issuing a command toward the plug from the local or a different network; any unwanted traffic between the plug and its vendor’s cloud, e.g. to exfiltrate privacy-sensitive data.

Our second example highlights the need for descriptors in device profiles that allow to specify traffic rate limits. An attacker controlling a rogue device in the LAN can flood the network with unwanted packets (pattern *B* in Fig. 1). In contrast to MUD, the profile language proposed by us allows specifying connection specific limits.

Finally, we illustrate the usefulness of application protocol layer descriptors in device profiles. We consider the scenario of an attacker who has infected an IoT device with botnet software. The device will try to contact a C&C server in the Internet and send a DNS query to resolve its name. In MUD, we are limited to open port 53 since sending DNS queries is part of the normal behavior of devices. In our profile language, we can specify which domain names can be resolved.

C. Threat model

Our focus is on attacks that manifest themselves as changes in the communication characteristics and patterns of the devices. Such changes can be caused by the attack itself or be consequences of a successful attack. The considered attacks can be roughly divided into two categories:

- Attacks in which the attacker attempts to compromise, remotely control, or render a device unusable through the network. Examples include DoS attacks, scans, unauthorized remote access, and messages that were not initiated by the Smart Home resident.
- Attacks in which the attacker misuses a compromised device to perform the above attacks against other devices or to exfiltrate data.

We assume that the attacker has the capability to attempt the above attacks from inside and outside the Smart Home network. Obviously, our approach cannot protect against an attack that does not affect the network communication that is visible to the firewall. In the envisaged scenario where the firewall is deployed at a central point of the home network, this includes attacks using ad-hoc and long-range communication (e.g., 4G).

We assume that the attacker does not have the capability to (a) compromise the firewall software or hardware; (b) compromise the device profiles. The latter deserves further explanation. In the MUD standard [17], the idea is that the MUD file is provided by the manufacturer. It can be debated whether end users should trust the manufacturer when it comes to the protection of their Smart Homes. In fact, manufacturers have been caught collecting sensitive information from their customers' devices [30]. Therefore, the possibility that a manufacturer distributes compromised profiles cannot be excluded. We have designed our profile language such that the profiles are easy for experts to understand. We envisage a trusted platform from which profiles can be securely downloaded and where experienced users can review them, and report any security or privacy misuse, as a community effort to help less savvy users. Since this model has already been discussed for other use cases in the literature [30], [11], we consider such a platform to be outside the scope of this paper.

III. System overview and description

We first give a high-level introduction to our security system for Smart Home networks. After that, we describe its components in detail.

A. System overview

Our approach is based on extended traffic profiles that describe the allowed communication patterns, including device interactions, of Smart Home devices. Our goal is to leverage the information contained in the traffic profiles to identify and block unwanted or unexpected communications. We assume that the Smart Home to be protected is based on a (wireless) LAN infrastructure. An obvious place for a firewall is therefore the central home router or access point, where the communication between the devices and with the outside world can be monitored. If other (non-LAN) communication channels exist for the devices, they have to be monitored, too.

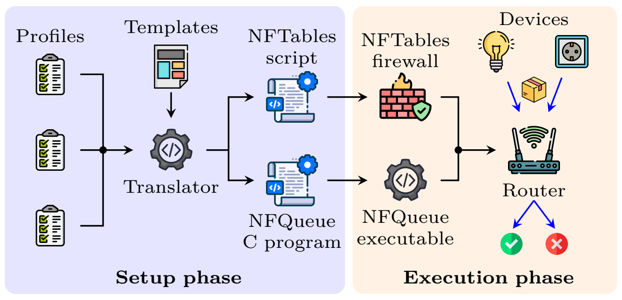


Fig. 2: System components

Fig. 2 shows the components of our proposed system. It is intended for inspection of live traffic on consumer-grade home routers and access points. As such, for efficiency reasons, its workflow is divided in two phases: the *setup phase* and the *execution phase*.

The *setup phase* takes, as input, the profiles for the Smart Home devices. Profiles are written in a human-readable language by the device manufacturer, the device owner, or other users. They describe the allowed communication activities at different protocol and abstraction levels. This comprises connection parameters (endpoints, protocols, etc.), traffic statistics (number of packets, byte rate, etc.), application layer information (e.g., URIs or domain names), and the relationships between connections (order, etc.).

The *execution phase* occurs when the firewall is deployed in the network, and inspects live traffic. Our firewall, running on a router in Fig. 2, builds on NFTables [1], the Linux kernel firewall, to filter traffic from and to the Smart Home devices. It installs NFTables rules that, where necessary, offload matching packets to a user-space program using the NFQueue infrastructure. That program is responsible for performing the complex matching and filter actions specified in the profiles.

During the *setup phase* a translator (written by us) makes the interface between the device profiles and the NFTables and NFQueue-related files required for the *execution phase*, by parsing the former to produce the latter.

B. Devices profiles

A device's profile is a model of its legitimate communication patterns. We use YAML as file format, instead of JSON and XML used by MUD. The reason for this

```
device-info:
  name: my-device
  mac: 00:11:22:33:44:55
  ipv4: 192.168.1.100

patterns:
  dns-p:
    protocols:
      dns:
        qtype: A
        domain-name: my.server.com
      udp:
        dst-port: 53
      ipv4:
        src: self
        dst: gateway
    bidirectional: true
    stats:
      packet-count: 4
```

```
[...]

interactions:
  dns-https-server:
    dns-server: !include patterns.dns-p
    https-server:
      protocols:
        tcp:
          dst-port: 443
        ipv4:
          src: self
          dst: my.server.com
      bidirectional: true
      stats:
        rate: 50/second

[...]
```

Fig. 3: Minimal device profile example. User defined labels are highlighted in green. Builtin keywords are shown in black.

choice is that the YAML syntax is more flexible, allowing, amongst others, the use of references to other fields in the same file. We extend this concept to allow referencing fields present in *other* files and also placeholders. This improves readability and reduces cluttering in the case of repeating patterns. To this end, our profile language supports a new `include` directive, which we have implemented thanks to `PyYAML`'s support for custom YAML parsers in Python. It should be noted that the features supported by our profiles are a superset of MUD, i.e., any MUD file can be trivially translated into our profile format.

Fig. 3 shows an excerpt from a profile. It consists of two main sections, namely the `device-info` section and the `interactions` section, that we describe in the following. The example also showcases the new `include` directive that allows to move parts of a declaration to another location in the profile and to refer to it by a qualified name. Concretely, the identifier `patterns.dns-p` refers to the user-defined field `dns-p` under the field `patterns`.

The `device-info` section specifies metadata used to identify the device, that is its name, MAC address, and IP address (v4 or v6). In the remaining of the profile, the keyword `self` can be used to reference the device.

The `interactions` section defines a series of *interactions*. In the example on Fig. 3, we only have one, which we (the profile author) have named `dns-https-server`. Each interaction defines a series of *policies*. In our example, the `dns-https-server` interaction defines two policies that we have named `dns-server` and `https-server`. We will first discuss them before explaining what interactions do.

The firewall only lets through packets matching a policy. The latter describes the packet features to match on. We allow all features supported by MUD (e.g., IP addresses, TCP/UDP ports, ARP message types, ICMP message types). In addition, we added support for more protocols, in particular for the application layer protocols (m)DNS, DHCP, HTTP, SSDP, CoAP, and IGMP, as these protocols are used by many Smart Home devices.

To describe more complex network activities, multiple policies can be combined to an interaction (`dns-https-server` in the example). In an interaction, not all its policies are active at the same time. An interaction starts when its first policy sees a matching packet. That first policy becomes then the active policy of the interaction. The interaction will at some point deactivate its currently active policy and activate its next policy. How this is done depends on the type of the currently active policy:

a) By default, a policy is a *one-off* policy. When it matches a packet, it is deactivated and the next policy is activated. If the property `bidirectional` is set, the policy requires a matching packet in each direction: it first matches on the fields as specified and then reverses the relevant fields (source and destination addresses and ports, DNS or ICMP message types, etc.) for the next packet.

b) A *transient* policy has a maximum duration or maximum packet count specified in its `stats` section. The interaction moves to the next policy if the transient policy has reached the specified maximum duration or number of matching packets *or* when a packet is seen that matches the next policy. If the property `bidirectional` is set, the policy matches packets in both directions.

c) A *periodic* policy contains a packet rate specification in the `stats` section. The policy stays active until a packet is seen that matches the next policy. Packets exceeding the specified rate are dropped.

C. The Firewall

In the following, we describe a firewall that is able to enforce the traffic patterns described in our profile language. It leverages NFTables, the next-generation Linux kernel firewall [1] and successor of IPTables. Since NFTables is shipped by default in many recent Linux distributions, our firewall can run on many platforms and does not require installing third-party tools. NFTables provides simple rule-based packet header matching for the most prominent layer 2, 3 and 4 protocols, such as ARP, IPv4/6, TCP, UDP and ICMP, as well as matching capabilities based on rate and size. NFTables allows intercepting packets traversing the Linux network stack with *Netfilter kernel hooks* [2]. Our firewall registers to the `prerouting` hook, which is the first hook able to handle packets coming from different interfaces.

By itself, NFTables' matching and filtering capabilities are quite limited. Its flexibility comes from one of its libraries, `libnetfilter_queue` (hereafter referred to as NFQueue), which allows NFTables rules to offload a matching packet to a so-called "queue", implemented as a callback function in a user-space program [3]. The latter can, in turn, process the packet and issue the final verdict (i.e., accept or drop), as summarized in Fig. 4.

We use NFTables' limited packet header matching capabilities to execute the basic matching rules defined in the policies of our profiles, e.g., the matching by IP address or packet rate. We implement the advanced features of our firewall in external binaries that receive the packets from NFTables using NFQueue. Those advanced features are:

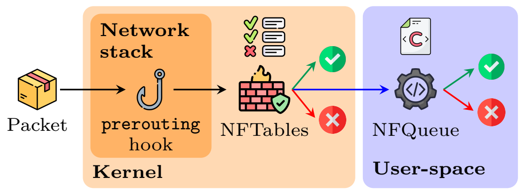


Fig. 4: General NFTables/NFQueue firewall operation

1) The interaction mechanism described in Section III-B.
2) Matching on maximum duration and packet count, as needed for transient policies.
3) Support for the protocols CoAP, DHCP, HTTP, IGMP, (m)DNS, and SSDP.
4) Support for domain names. Our firewall monitors DNS responses and caches the resolved names. This allows using domain names in policies instead of IP addresses, which is necessary for our profiles to stay up-to-date, as the IP addresses of the cloud servers contacted by an IoT device might regularly change.

When the firewall is running, there is one user-space process per device, containing the NFQueue callback code for that device. This process is further split into multiple threads, one per queue, that is, one per NFTable rule installed for that device. Note that a packet can match multiple rules, possibly even from several devices. For example, it could be affected by policies of the sending and of the receiving device. In that case, it will be forwarded to all corresponding NFQueue functions. The packet is admitted if it complies to any of the prompted policies.

D. State machine for interactions

The interaction-based matching and filtering mechanism described in Section III-B requires a stateful firewall. Our current prototype maintains one Finite State Machine (FSM) per interaction to keep track of its progress. The number of FSM states depends on the number, types, and properties of the policies in that interaction.

We explain the functioning of the FSM with an example of an interaction with three policies A, B, and C. Policy A is one-off and bidirectional, B is periodic, and C is transient. The resulting FSM is shown in Fig. 5.

State 0 is the initial state of the FSM. If a packet matching policy A is seen, we move to state 1. Since A is bidirectional, it now waits for the packet in the opposite direction. After that, it moves to state 2, which presents policy B. The latter is periodic, i.e., it will accept all matching packets and stay in state 2 until a packet matching policy C is seen, which causes the FSM to move to state 3. As a transient policy, C accepts all matching packets, but, unlike policy B, it will stop matching when the specified maximum duration or packet count is reached. In that case, the interaction restarts in state 0 or the FSM directly moves to state 1 if the last packet matched policy A. All non-matching packets are dropped.

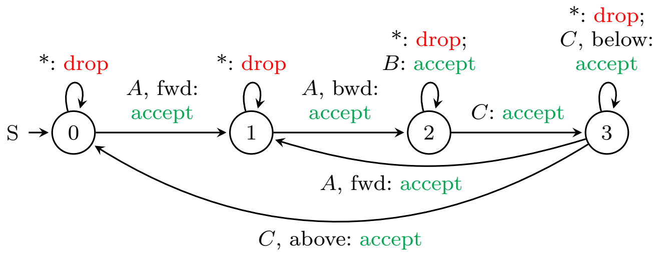


Fig. 5: Finite State Machine for an interaction with an one-off policy A, a periodic policy B and a transient policy C. “fwd” and “bwd” specify the direction of the packet. “below” indicates a packet within the limits of a transient policy. “*” stands for all non-matching traffic.

Note that the rate limitation of the periodic policy is directly enforced by NFTables rules and not by the FSM.

E. The Translator

The *translator* is a tool written by us in Python that translates the device profiles, during the *setup phase* (mentioned in Section III-A), into NFTables configuration scripts and C code for the NFQueue part of the firewall, that is for the features of the firewall beyond the simple matching rules of NFTables. To simplify the code generation, we have prepared C code templates that are completed with the information extracted from the profiles using the Jinja templating engine [4]. The C code is compiled to binary files that will be executed, upon deployment, on the device hosting the firewall (e.g. the network's wireless access point).

IV. Experimental setup

This section describes the experimental setup that we use for the evaluation of our approach.

A. Devices

To mimic a typical Smart Home network, we use commercial, off-the-shelf devices (Table I). They can be divided into two groups, namely *end devices* and *Smart Hubs*. The former include sensors and actuators, i.e., devices that sense and manipulate the physical environment. Some end devices use Zigbee [8] and require an intermediate device (a *gateway*) to communicate with IP networks. Smart Hubs are more complex devices that allow to control other devices and often have additional functions such as music streaming. Many commercially available hubs have multiple network interfaces for different types of networks and can therefore also act as a gateway.

We connect the IP devices, either via Wi-Fi or Ethernet, to a router acting as a wireless Access Point (AP), running OpenWrt [5], an open-source Linux distribution for network appliances. The AP is in turn wired to a router, also running OpenWrt, which acts as the gateway between the home LAN and the Internet. Our firewall is deployed

ID	Name	Type	Protocol	#Policies	#Interactions	#Queues
1	TP-Link HS110 smart plug	End device	Wi-Fi	35	13	20
2	Xiaomi MJSXJ02CM camera	End device	Wi-Fi	37	15	21
3	D-Link DCS-8000LH camera	End device	Wi-Fi	58	18	24
4	Tuya Wi-Fi PIR motion sensor	End device	Wi-Fi	27	10	22
5	Philips Hue White and Color lamp A60	End device	Zigbee	No rules: Zigbee devices		
6	SmartThings door sensor (Multipurpose)	End device	Zigbee			
7	SmartThings plug	End device	Zigbee			
8	SmartThings presence sensor	End device	Zigbee			
9	Amazon Echo Dot (2nd gen)	Hub	Wi-Fi	No rules: only used as hub		
10	Philips Hue bridge	Gateway	Ethernet, Zigbee	96	33	26
11	SmartThings V3 hub	Hub	Ethernet, Zigbee	41	18	17

TABLE I: Testbed devices. The ID is used to identify the device in Fig. 6.

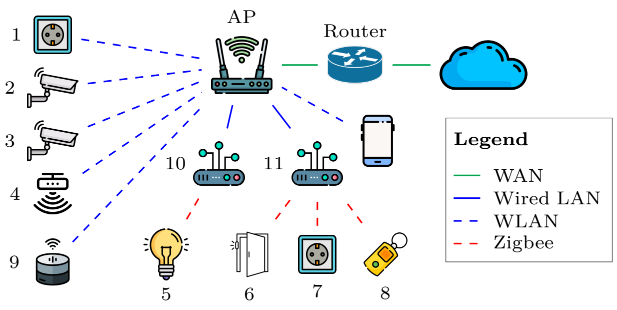


Fig. 6: Experimental Smart Home network. The numbers refer to the devices in Table I.

on the AP, which is a very modest dual-band AP (TP-Link TL-WDR4900, v1) with a 800 MHz Freescale PPC CPU and 128 MBytes of RAM.

B. *Traffic capture*

We capture the traffic from and to the devices. We use the resulting datasets for the creation of the device profiles.

To collect network traffic, we execute `tcpdump` on the four network interfaces of the LAN AP, namely the 2.4/5.0 GHz WLAN, the wired local LAN, and the WAN interfaces. For the recorded traffic to be as exhaustive as possible, we physically interact with the devices. We also communicate with all devices through their respective vendor-provided companion app running on a mobile phone, first connected to the same LAN as the device, then to an external network. Device-specific commands are issued, e.g., switching the smart plug on and off, or streaming from the camera. For the devices compatible with the Amazon or SmartThings hubs, similar interactions are also done through the respective hubs' apps and through Amazon Alexa's voice control. In this way, we make sure to cover various types of device behavior, such as derived in [24]. Finally, we also leave the devices idle to capture background traffic and device activities not linked to user activities, such as checks for software updates.

C. *Profile creation*

We manually build device profiles from the traffic captures. Since the Philips Hue lamp and the three SmartThings end devices are not IP devices, their behavior is described in the profile of the gateway they are connected to, that is the Philips Hue Bridge and the SmartThings Hub, respectively. The Amazon Echo Dot controls the other devices via WiFi. The legitimate traffic between the Echo and the WiFi devices is rich and confined to the home. We thus do not create a profile for it. However, the Echo Dot *does* appear in the profiles of the other devices as part of their interactions with it.

The profiles are then automatically translated to script and C code files by our translation tool (see Section III-E). Table I gives the number of different policies, interactions, and NFQueue queues for the devices. The most "complex" devices are the Philips Hue bridge and the attached lamp. Unlike a smart plug, the lamp allows different types of interactions (e.g., adjusting the brightness, the color, etc.). Moreover, the bridge has more connectivity options (mDNS, SSDP) than simpler devices and can directly communicate with devices such as the SmartThings hub without an Internet connection. This results in a greater number of NFTables rules for the different protocols and in a larger NFQueue program to process the interactions.

V. Evaluation

We validate the performance of our firewall in attack-free and attack scenarios.

A. *Fuzzing-based validation*

We first verify that our system accepts traffic which conforms to the configured profiles, whilst dropping any traffic which does not conform. We developed a fuzzer which takes a packet capture as input and modifies protocol fields of randomly selected packets. The modification is always applied to the highest supported protocol layer (e.g., the DNS layer for a DNS packet) in order to stress the NFTable rules as well as the NFQueue code that handles the highest protocol layers.

Our testing methodology is the following:

Base PCAP creation: We take the device traffic collected in Section IV-B. The firewall should not block any of those packets.

Fuzzing PCAP files: We run our fuzzer on each PCAP file five times to produce mutated versions.

Labeling packets: If the packet was modified and does not conform to the device profile anymore, its expected verdict is DROP. If the packet was *not* edited, we must check if a previous packet in the same interaction was modified. If yes, and if this should prevent the current packet from being accepted because it corrupts the interaction, the expected verdict is DROP, otherwise ACCEPT.

Replaying PCAP files: We replay the mutated PCAPs through the firewall, log its verdict for each packet, and compare it with its label.

We replayed a total of 10,894 packets. Among them, 4,348 were supposed to be accepted, and 6,546 to be dropped. Our firewall issued the correct verdict for all packets.

B. Validation on attack-free scenarios

Our system must be transparent to the user of the Smart Home network if no attack occurs. Traffic should flow without noticable additional delay. To validate the correct behavior of our firewall in a realistic setting, we deploy it in the testbed depicted in Fig. 6, consisting of the devices shown in Table I. As explained in Section IV-C, the behavior of the Zigbee devices (the Philips Hue lamp, and the SmartThings plug, door sensor, and presence sensor) is embedded in the profile of their related gateway (Philips Hue Bridge or SmartThings Hub), whereas the Amazon Echo Dot is only used to control the WiFi devices, and does not have a dedicated profile.

First, we activate one device at a time and interact with it. For each device, we record traffic on the four AP network interfaces. We compute the latency induced by processing on the AP, by taking the difference between the packet timestamp on the egress and ingress interfaces. We repeat this measurement in four different scenarios:

- `no-firewall`: without any firewall.
- `base-firewall`: with the default built-in OpenWrt firewall rules active on the AP. These rules accept all packets, but introduce some rate limiting, e.g., for incoming TCP SYN packets.
- `my-firewall`: with our firewall. We only use one device at a time and activate only the profile for that device. We repeat this for all devices. The goal is to measure the influence of a device's profile on the firewall performance. Our firewall processes 2,105 packets (1.7 MBytes) for the D-Link camera (during 10 minutes, of which the camera was streaming during 20 seconds), 3,862 packets (0.8 MBytes) for the Philips Hue, 1,071 packets (0.2 MBytes) for the SmartThings hub, 682 packets (0.2 MBytes) for the TP-Link plug, 31,816 packets (17.2 MBytes) for the Xiaomi camera (during 30 minutes, of which the camera was streaming during 120 seconds), and 303 packets (0.04 MBytes) for the Tuya motion sensor.
- `all-devices`: with our firewall. All device profiles are activated, and we interact with the six devices at the same time to mimic a busy Smart Home network. In this test, 12k packets (4.7 MBytes) were exchanged during 48 minutes.

No false positives are observed, i.e., the firewall correctly lets pass all packets.

Fig. 7a shows the mean and 95-percentile range (the interval from the 10 to the 90-percentile) of the latency induced by the firewall in the four scenarios for the five tested end devices. We observe that the 95-percentile interval of the latency is under 0.6 ms in all test cases. The higher latency observed for the Philips Hue device can be explained by the fact that this device transmits packets that require more processing, namely a lookup in the DNS cache. Nevertheless, the latency is negligible compared to the network latency, which is typically of the order of 2 to 5 ms between devices in the same LAN. We conclude that the latency caused by the firewall will not be noticed by the user when interacting with the devices.

In the `all-devices` scenario where all device profiles are active at the same time, the firewall latency increases. This is expected, as the system must process more packets and rules. Nevertheless, the latency is still acceptable. This indicates that our system is suitable for most current Smart Home network, where the number of devices hardly exceeds ten [15].

C. Validation on attack scenarios

As argued in Section II-C, our approach is, by design, limited to attacks that have an impact on the network activities. Such attacks include reflection attacks and other types of brute-force DoS attacks, network scans, data exfiltration, attempts to remotely control a device from an unknown host, send unexpected messages, etc. We do not show evaluation results for all of these attacks since many of them are trivially blocked because they use addresses, ports, or protocols that are not defined in the profiles.

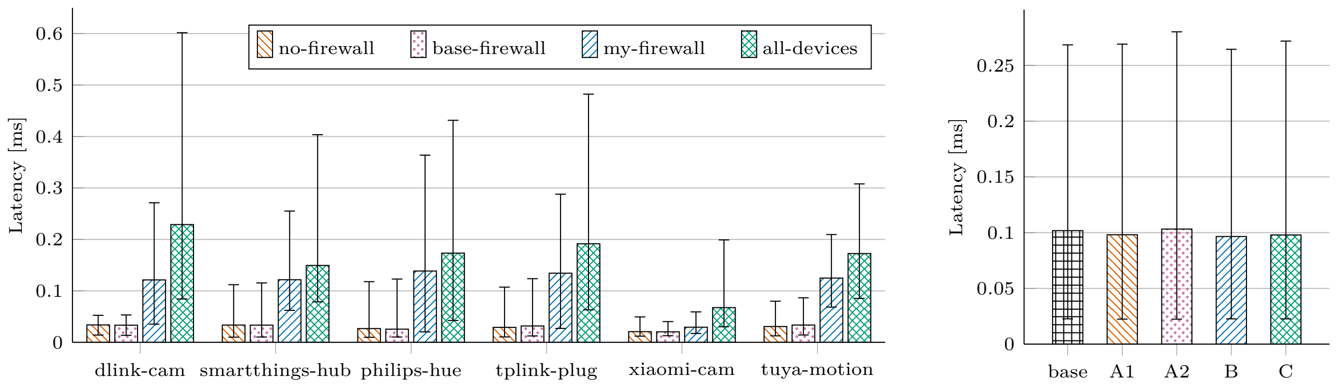
In the following, we revisit the three attacks described in Section II-B, as they concern key properties of our system, that is device interactions, traffic statistics, and support for higher-layer protocols. We run the attacks in the presence of our firewall and record the accepted and dropped packets. We also do latency measurements.

We implement the network shown in Fig. 1 with a Tuya motion sensor, a TP-Link smart plug, and a smartphone. As explained before, the expected behavior is: The sensor, upon motion detection, transmits a message to the vendor's cloud, indicating the detection. The cloud processes the message, and instructs the smart plug to change its state. We run the firewall with the corresponding profiles. Among others, this means that the firewall is aware of the above device interaction, of the expected traffic rates, and of the domain names of the servers (NTP servers, cloud servers, etc.) contacted by the devices.

We test the following attacks:

A1 The attacker issues commands to the plug from a host located in the local network.

A2 The attacker issues commands to the plug from a host outside the local network.



(a) Latency in attack-free scenarios

(b) Latency in attack scenarios

Fig. 7: Mean latency induced by the AP hosting the firewall, per experimental scenario

B The attacker sends two ARP packets per second from the smartphone. The profile specifies a maximum rate of one ARP packet per second.

C A device tries to resolve domain names of servers not mentioned in the profiles. This mimics a scenario where an attacker has compromised the device and lets it communicate with a malicious server.

The firewall successfully blocks all attack packets, without affecting the normal function of the devices.

Finally, we deploy again the full testbed network as illustrated on Fig. 6, and we measure the latency added by our firewall, in a scenario similar to the `all-devices` presented in Section V-B, when the aforementioned attacks are ongoing. Fig. 7b shows the mean and 95-percentile interval of the latency for legit packets. In all cases, the latency stays low.

D. CPU and Memory usage under stress

We measure the CPU and memory usage of our system to study its robustness and processing capabilities in case of higher network load. To this end, we deploy the TP-Link smart plug, along with our system configured with the profile of this device. We evaluate four scenarios with increasing computational resources requirements:

- **Normal**: Normal operation, i.e., identical to the latency measurements presented in Section V-B.
- **State**: Under a steady flow of packets rejected by NFQueue because of the FSM state. The firewall must only check the FSM state to issue the reject verdict.
- **String**: Under a steady flow of DNS requests with an unexpected domain name. The firewall must perform name comparisons to issue the reject verdict.
- **Lookup**: Under a steady flow of packets directed towards an IP address not present in the DNS table. The firewall must perform a lookup in the DNS cache to issue the reject verdict.

For the last three scenarios, the packet rate is set to 1,000 packets/s.

We measure mean CPU usages of 0.1% (Normal), 2.5% (State), 3.0% (String), and 3.2% (Lookup) for the respective scenarios. The memory usage stays below 5% for all five scenarios. We conclude that the firewall resource usage is very low, even on our modest hardware. This shows our firewall can handle traffic loads which are considered high for Smart Home IoT networks. Additionally, it is very lightweight and can be deployed on virtually any home router supporting OpenWrt.

VI. Discussion

1) What types of attacks can/cannot be blocked?: Our approach focuses on attacks with an observable impact on the network behavior of the affected devices. Since the firewall monitors all its interfaces, it can also block attacks *from* Smart Home devices, such as outgoing DDoS attacks or scans. Our firewall performs a limited form of deep packet inspection, for example to match the protocol fields of DNS and HTTP traffic. Apart from that, payload is not further analyzed, which means that attacks that do not look suspicious on metadata level or traffic statistics level (such as an SQL injection attack) cannot be stopped.

Our firewall cannot inspect protocols fields of encrypted protocols, such as the URL in HTTPS requests. Note, however, that many IoT devices use clear text protocols due to resource constraints or because the manufacturer is shy of the associated effort (updating certificates, etc.). Furthermore, previous research has shown that the headers and metadata of network traffic can provide sufficient information for detecting many types of attacks [23], [25].

Data exfiltration attacks targeting privacy in Smart Homes can be blocked depending on their nature. Apart from blocking data transfers to unknown destinations, the firewall can limit the rate, duration, and size of network activities. Therefore, even if the destination is generally trusted, the firewall can block data transfers that deviate from the expected behavior, such as a smart speaker with a microphone sending more data than expected.

Finally, traffic related to user events (e.g., activating a device) might be indistinguishable from non-user-initiated traffic (e.g., keep-alives). In that case, our firewall might prove to be imprecise, if a specific user event must be blocked whilst other traffic should still be accepted. Nonetheless, for most devices, this type of traffic *can* be distinguished, thanks to different packet signatures, or an identifiable periodicity.

2) Who creates the profiles?: Device profiles must accurately describe the devices' communication patterns. In the context of this research, we assumed profiles are created and distributed by the devices' manufacturers (an assumption identical to MUD's) or a group of experienced users. Other ways to create device profiles can be imagined. Inspired by the work of Hamza *et al.* [12] for MUD, one could design a tool that automatically creates our profiles from traffic captures. Hu *et al.* recently proposed BehavIoT [14] to extract models of the behavior of Smart Home networks from labeled datasets. We see it as future work to explore whether and how those models could be translated into device profiles for our firewall.

Finally, it should be noted that profiles might need to be updated during the lifetime of the firewall. During our experiments we had to do this for one of the devices after it received a firmware update that changed its behavior.

VII. Related work

Related work can be divided into three areas: The usage of device profiles to enforce security policies, protection systems for Smart Homes, and MUD extensions.

1) IoT device profiling: Similarly to MUD, Barrera *et al.* [7] present a methodology to derive network behavior profiles for Smart Home devices from traffic captures. Their profiles are limited to basic network characteristics that can easily be translated into IPTables rules. In our work, profiles are much more expressive and describe interactions between devices, which is beyond the capabilities of IPTables rules.

Hamza *et al.* [12] developed the *MUDgee* tool, which can produce a MUD file for a device from traffic captures. As the created profiles comply with the base MUD specification, they suffer from MUD's shortcomings.

2) IoT and Smart Home protection systems: Numerous authors have developed security systems for IoT or Smart Home networks. The vast majority leverages Machine Learning (ML) models. A typical example of such ML-based systems is *Passban IDS*, developed by Eskandari *et al.* [9]. It is an anomaly-based IDS which can be deployed on gateways with commodity hardware. The IDS proposed by Pashamokhtari *et al.* [25] employs sequential ML models that are specialized on subsets of traffic features describing the expected traffic of IoT devices in Smart Homes. They employ Software Define Networking (SDN) to dynamically adapt the traffic forwarding rules. ML-based IDSs often require heavy resources as well as extensive training data. Our approach is inherently opposite: by being based on a simple model of the intended device network behavior, it only requires device profiles as input, and is very lightweight. Hamza *et al.* [13] were the first to leverage MUD profiles to provide network security. They did so by translating the MUD profiles' ACLs into SDN rules.

3) Extensions to the MUD standard: As discussed in Section II-B, the MUD standard suffers from multiple shortcomings. Several authors have tried to overcome those shortcomings. Lear *et al.* [16] proposed new MUD fields which allow matching on the traffic rate. Our profile language supports rate limitation as well as new matchable traffic statistics such as size and duration.

Reddy *et al.* [26] developed a MUD extension for TLS features, e.g., encryption, signature and key exchange algorithms. Matheu *et al.* [19] extend MUD with the capability of matching application layer protocols by their name, as well as the number of concurrent connections using a specific protocol and the accessed resources. Contrary to them, we provide protocol-specific matching capabilities and not just protocol name matching.

Morais and Farias [22] added *Malicious Traffic Description* fields to MUD in an effort to integrate signature-based intrusion detection. Finally, Matheu *et al.* [20] leverage the Medium-level Security Policy Language (MSPL) [29] to provide data privacy, resource authorization, and channel protection capabilities to MUD.

VIII. Conclusion

Modern Smart Home installations exhibit complex communication patterns caused by the interactions of the deployed devices. Simple firewalling rules ignoring these interactions leave opportunities to a number of attacks.

Taking inspiration from the IETF's MUD standard, we defined a new language for communication profiles of IoT devices. Contrary to MUD, the language is expressive enough to describe communication patterns involving multiple devices and their interactions. We designed a lightweight firewall that uses the profiles to enforce communication policies in Smart Home networks. We performed experiments with various types of IoT devices and showed that the firewall is effective in blocking unwanted network activities, while remaining transparent to the devices and the users.

Acknowledgements

F. De Keersmaeker is funded by the F.R.S.-FNRS Research Fellow [ASP] grant with number 40006064. R. Sadre is supported by the Walloon Belgian region project CyberExcellence. Grant #1318167 from the Cisco University Research Program Fund, an advised fund of Silicon Valley Foundation, partially supports C. Pelsser. All icons are from flaticon.com.

References

[1] https://www.netfilter.org/projects/nftables/index.html.

[2] https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks.

[3] https://www.netfilter.org/projects/libnetfilter_queue/index.html.

[4] https://jinja.palletsprojects.com/en.

[5] https://openwrt.org.

[6] M. Antonakakis *et al.* Understanding the Mirai Botnet. In *USENIX Security '17*. USENIX.

[7] D. Barrera *et al.* Standardizing IoT network security policy enforcement. In *Workshop on decentralized IoT security and standards (DISS)*, 2018.

[8] S. C. Ergen. ZigBee/IEEE 802.15. 4 Summary. *UC Berkeley, September*, 10(17), 2004.

[9] M. Eskandari *et al.* Passban IDS: An Intelligent Anomaly-Based Intrusion Detection System for IoT Edge Devices. *IEEE Internet of Things Journal*, 7(8), August 2020.

[10] C. Fu, Q. Zeng, and X. Du. HAWatcher: Semantics-Aware anomaly detection for appified smart homes. In *USENIX Security '21*. USENIX.

[11] S. Guo, M. Lyu, and H. H. Gharakheili. Realizing Open and Decentralized Marketplace for Exchanging Data of Expected IoT Behaviors, 2023. http://arxiv.org/abs/2401.00141.

[12] A. Hamza *et al.* Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles. In *IoT S&P '18*. ACM.

[13] A. Hamza *et al.* Combining MUD Policies with SDN for IoT Intrusion Detection. In *IoT S&P '18*. ACM.

[14] T. Hu *et al.* BehavIoT: Measuring Smart Home IoT Behavior Using Network-Inferred Behavior Models. In *IMC '23*. ACM.

[15] D. Y. Huang *et al.* IoT Inspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale. *ACM IMWUT*, 4(2), June 2020.

[16] E. Lear and O. Friel. Bandwidth Profiling Extensions for MUD. Technical Report draft-lear-opsawg-mud-bw-profile-01, IETF, July 2019.

[17] E. Lear *et al.* Manufacturer Usage Description Specification. RFC 8520, March 2019.

[18] I. Martins *et al.* Host-based IDS: A review and open issues of an anomaly detection system in IoT. *Future Generation Computer Systems*, 133, 2022.

[19] S. N. Matheu *et al.* Extending MUD Profiles Through an Automated IoT Security Testing Methodology. *IEEE Access*, 7, 2019.

[20] S. N. Matheu *et al.* Security Architecture for Defining and Enforcing Security Profiles in DLT/SDN-Based IoT Systems. *Sensors*, 20(7), January 2020.

[21] N. Mazhar *et al.* Role of Device Identification and Manufacturer Usage Description in IoT Security: A Survey. *IEEE Access*, 9, 2021.

[22] S. Morais and C. Farias. INXU - A Security Extension for RFC 8520 to Give Fast Response to New Vulnerabilities on Domestic IoT Networks. In *WPIETF '20*. SBC.

[23] M. Nobakht *et al.* A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow. In *ARES '16*. ACM ICPS.

[24] T. OConnor *et al.* HomeSnitch: behavior transparency and control for smart home IoT devices. In *WiSec '19*. ACM.

[25] A. Pashamokhtari *et al.* Progressive Monitoring of IoT Networks Using SDN and Cost-Effective Traffic Signatures. In *ETSecIoT '20*. IEEE.

[26] T. Reddy *et al.* MUD (D)TLS profiles for IoT devices. Technical Report draft-reddy-opsawg-mud-tls-12, IETF, April 2023.

[27] E. Ronen and A. Shamir. Extended Functionality Attacks on IoT Devices: The Case of Smart Lights. In *EuroS&P '16*. IEEE.

[28] D. Uroz and R. J. Rodríguez. Characterization and Evaluation of IoT Protocols for Data Exfiltration. *IEEE Internet of Things Journal*, 9(19), October 2022.

[29] A. M. Zarca *et al.* Security management architecture for NFV/SDN-aware IoT systems. *IEEE Internet of Things Journal*, 6(5), 2019.

[30] I. Zavalyshyn *et al.* My house, my rules: A private-by-design smart home platform. In *MobiQuitous '20*. EAI.