Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles

Ayyoob Hamza
UNSW Sydney
ayyoobhamza@student.unsw.edu.au

Dinesha Ranathunga
ACEMS, University of Adelaide
dinesha.ranathunga@adelaide.edu.au

Hassan Habibi Gharakheili
UNSW Sydney
h.habibi@unsw.edu.au

Matthew Roughan
ACEMS, University of Adelaide
matthew.roughan@adelaide.edu.au

Vijay Sivaraman
UNSW Sydney
vijay@unsw.edu.au

ABSTRACT

IoT devices are increasingly being implicated in cyber-attacks, raising community concern about the risks they pose to critical infrastructure, corporations, and citizens. In order to reduce this risk, the IETF is pushing IoT vendors to develop formal specifications of the intended purpose of their IoT devices, in the form of a Manufacturer Usage Description (MUD), so that their network behavior in any operating environment can be locked down and verified rigorously.

This paper aims to assist IoT manufacturers in developing and verifying MUD profiles, while also helping adopters of these devices to ensure they are compatible with their organizational policies. Our first contribution is to develop a tool that takes the traffic trace of an arbitrary IoT device as input and automatically generates the MUD profile for it. We contribute our tool as open source, apply it to 28 consumer IoT devices, and highlight insights and challenges encountered in the process. Our second contribution is to apply a formal semantic framework that not only validates a given MUD profile for consistency, but also checks its compatibility with a given organizational policy. Finally, we apply our framework to representative organizations and selected devices, to demonstrate how MUD can reduce the effort needed for IoT acceptance testing.

CCS CONCEPTS

• **Security and privacy** → *Formal methods and theory of security*;

KEYWORDS

IoT, MUD, Policy Verification

ACM Reference Format:
Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Matthew Roughan, and Vijay Sivaraman. 2018. Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles. In *IoT S&P'18: ACM SIGCOMM 2018 Workshop on IoT Security and Privacy , August 20, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3229565.3229566

IoT S&P'18, August 20, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5905-4/18/08…$15.00
https://doi.org/10.1145/3229565.3229566

1 INTRODUCTION

Many connected IoT devices can be found on search engines such as Shodan [13], and their vulnerabilities exploited at scale. For example, Dyn, a major DNS provider, was subjected to a DDoS attack originating from a large IoT botnet comprising thousands of compromised IP-cameras [9]. IoT devices, exposing TCP/UDP ports to arbitrary local endpoints within a home or enterprise, and to remote entities on the wider Internet, can be used by inside and outside attackers to reflect/amplify attacks and to infiltrate otherwise secure networks.

These security concerns have prompted standards bodies to provide guidelines for the Internet community to build secure IoT devices and services [15–17], and for regulatory bodies (such as the US FCC) to control their use [6]. The focus of our work is an IETF proposal called Manufacturer Usage Description (MUD) [11] which provides the first formal framework for IoT behavior that can be rigorously enforced. This framework requires manufacturers of IoTs to publish a behavioral profile of their device, as they are the ones with best knowledge of how their device will behave when installed in a network; for example, an IP camera may need to use DNS and DHCP on the local network, and communicate with NTP servers and a specific cloud-based controller in the Internet, but nothing else. Such requirements vary across IoTs from different manufacturers. Knowing each device's requirements will allow network operators to impose a tight set of access control list (ACL) restrictions for each IoT device in operation, so as to reduce the potential attack surface on their network.

The MUD proposal therefore provides a light-weight model of achieving very effective baseline security for IoT devices by allowing a network to automatically configure the required network access for IoT devices, so that they can perform their intended functions without having unrestricted network privileges. This open and standards-based approach that leverages the expertise of device manufacturers allows enterprises to scale their IoT deployments by automating much of the effort needed to achieve baseline security in their network.

MUD is a new and emerging paradigm, and there is little collective wisdom today on how manufacturers should develop behavioral profiles of their IoT devices, or how organizations should use these profiles to secure their network. This paper[1] is our attempt to address both these shortcomings. Our first contribution helps

[1]This project was supported by the Australian Research Council (ARC) through Linkage Grant LP150100666, Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS) and Google Faculty Research Awards.

IoT manufacturers generate and verify MUD profiles: we develop a tool that takes as input the packet trace containing the operational behavior of an IoT device, and generate as ouput a MUD profile for it. We contribute our tool as open source, apply it to 28 consumer IoT devices, and highlight insights and challenges encountered in the process. Our second contribution applies a formal semantic framework that not only validates a given MUD profile for compliance and consistency, but also checks its compatibility with a given organizational policy. We apply our semantic framework to representative organizations and selected devices, and demonstrate how MUD can greatly simplify the process of IoT acceptance into the organization.

The rest of the paper is organized as follows: §2 describes relevant background work on IoT security and formal policy modeling using metagraphs. §3 describes our open-source tool for automatic MUD profile generation, and provides insights and challenges encountered in applying it to 28 real IoT devices. Our verification framework for MUD policies is described in §4, followed by evaluation results in §5. We discuss some potential enhancements to the MUD specifications in §6, and conclude the paper in §7.

2 BACKGROUND AND RELATED WORK

Securing the IoT has been secondary to innovating new devices and services, creating substantial safety and economic risks for the Internet [14]. Today, many IoT manufacturers do not incorporate even basic security measures into their devices [12], while network operators have poor visibility into the network activity of their connected devices and are unsure of what access control policies to apply to them [27]. As IoT botnets grow in size and increase in sophistication, attackers are using them to launch large-scale DDoS attacks [3]; devices such as baby monitors, refrigerators and smart plugs have been hacked and controlled remotely [25]; and many organizational assets such as cameras are being accessed publicly [1, 29].

Existing IoT security guidelines and recommendations [6, 15–17] are largely qualitative and subject to human interpretation, and therefore unsuitable for automated and rigorous application. The IETF MUD specification [11] on the other hand defines a formal framework to capture device run-time behavior, and is therefore amenable to rigorous evaluation. When coupled with the observation that most IoT devices have a small and recognizable pattern of communication (as demonstrated in our previous work [24]), MUD allows IoT device behavior to be captured succintly, verified formally for compliance with organizational policy, and assessed at run-time for anomalous behavior that could indicate an ongoing cyber-attack.

A valid MUD profile contains a root object called "access-lists" container [11] that comprises several access control entries (ACE), serialized in JSON format. Access-lists are explicit in describing the direction of communication, *i.e., from-device* and *to-device*. Each ACE would match on source/destination port numbers for TCP/UDP, and type and code for ICMP. The MUD specifications also distinguish *local-networks* traffic from *Internet* communications.

We provide here a brief background on the formal modeling and verification framework used in this paper. We begin by noting that the lack of formal policy modeling in current network

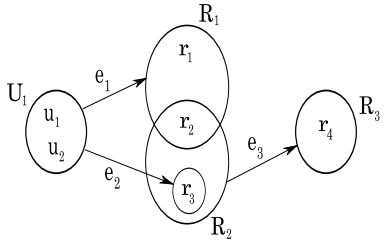


Figure 1: A metagraph consisting of six variables, five sets and three edges.

systems contribute to frequent misconfigurations [22, 23, 28]. We use the concept of a *metagraph*, which is a generalized graph theoretic structure that offers rigorous formal foundations for modeling and analyzing communication-network policies in general. A metagraph is a directed graph between a collection of sets of "atomic" elements [2]. Each set is a node in the graph and each directed edge represents the relationship between the sets. Fig. 1 shows an example where a set of users (U_1) are related to sets of network resources (R_1, R_2, R_3) by the edges e_1, e_2 and e_3 describing which user u_i is allowed to access resource r_j.

Metagraphs can also have attributes associated with their edges. An example is a *conditional metagraph* which includes propositions – statements that may be true or false – assigned to their edges as qualitative attributes [2]. The generating sets of these metagraphs are partitioned into a variable set and a proposition set. A conditional metagraph is formally defined as follows:

Definition 1 (Conditional Metagraph). *A conditional metagraph is a metagraph $S=\langle X_p \cup X_v, E\rangle$ in which X_p is a set of propositions and X_v is a set of variables, and:*

1. at least one vertex is not null, i.e., $\forall e' \in E, V_{e'} \cup W_{e'} \neq \phi$

2. the invertex and outvertex of each edge must be disjoint, i.e., $X = X_v \cup X_p$ *with* $X_v \cap X_p = \phi$

3. an outvertex containing propositions cannot contain other elements, i.e., $\forall p \in X_p, \forall e' \in E$, *if* $p \in W_{e'}$, *then* $W_{e'} = p$.

Conditional metagraphs enable the specification of stateful network-policies and have several useful operators. These operators readily allow one to analyze MUD policy properties like consistency. To the best of our knowledge, this is the first attempt to automatically generate MUD profiles, formally check their consistency and compatibility with an organizational policy, prior to deployment.

3 MUD PROFILE GENERATION

The IETF MUD specification is still evolving as a draft. Hence, IoT device manufacturers have not yet provided MUD profiles for their devices. We, therefore, developed a tool – *MUDgee* – which automatically generates a MUD profile for an IoT device from its traffic trace in order to make this process faster, cheaper and more accurate. In this section, we describe the structure of our open source tool [7], apply it to traces of 28 consumer IoT devices, and highlight insights.

We captured traffic flows for each IoT device during a six month observation period, to generate our MUD rules. The rules reflect an application whitelisting model (*i.e.,* only 'allow' rules with default 'drop'). Having a combination of 'accept' and 'drop' rules requires a

Table 1: Flows observed for Blipcare BP monitor (*: wildcard, proto: Protocol, sPort: source port number, dPort: destination port number).

Source	Destination	proto	sPort	dPort
*	192.168.1.1	17	*	53
192.168.1.1	*	17	53	*
*	tech.carematix.com	6	*	8777
tech.carematix.com	*	6	8777	*

notion of rule priority (*i.e.,* order) and is not supported by the current IETF MUD draft. For example, Table 1 shows traffic flows observed for a Blipcare blood pressure monitor. The device only generates traffic whenever it is used. It first resolves its intended server at `tech.carematrix.com` by exchanging a DNS query/response with the default gateway (*i.e.,* the top two flows). It then uploads the measurement to its server operating on TCP port 8777 (described by the bottom two rules).

3.1 MUDgee Architecture

MUDgee implements a programmable virtual switch (vSwitch) with a header inspection engine attached and plays an input PCAP trace (of an arbitrary IoT device) into the switch. *MUDgee* has two separate modules; (a) captures and tracks all TCP/UDP flows to/from device, and (b) composes a MUD profile from the flow rules.

Capture intended flows: Consumer IoT devices use services provided by remote servers on the cloud and also expose services to local hosts (*e.g.,* a mobile App). As required by MUD specifications, we track (intended) device activities for both remote and local communications using separate flow rules.

It is challenging to capture services (*i.e.,* especially those operating on non-standard TCP/UDP ports) that a device is either accessing or exposing. This is because local/remote services operate on static port numbers whereas source port numbers are dynamic (and chosen randomly) for different flows of the same service. We note that it is trivial to deduce the service for TCP flows by inspecting the SYN flag, but not so easy for UDP flows. We, therefore, developed an algorithm (Fig. 2) to capture bidirectional flows for an IoT device.

We first configure the vSwitch with a set of proactive rules, each with a specific action (*i.e.,* "forward" or "mirror") and a priority (detailed rules can be found in our technical report [8]). Proactive rules with a 'mirror' action will feed the header inspection engine with a copy of the matched packets. Our inspection algorithm, shown in Fig. 2, will insert a corresponding reactive rule into the vSwitch.

Giving insights into our algorithm, for example, a DNS reply packet is matched to a top priority flow and our algorithm extracts and stores the domain name and its associated IP address into a DNS cache table. This cache is dynamically updated upon arrival of a DNS reply matching an existing request.

Looking into the priority of rules in our vSwitch, the MUD specification requires the segregation of traffic to and from a device for both local and Internet communications. Our algorithm achieves this by assigning a unique priority to the reactive rules associated with each of the groups: from-local, to-local, from-Internet and to-Internet. We use a specific priority for flows that contain a TCP

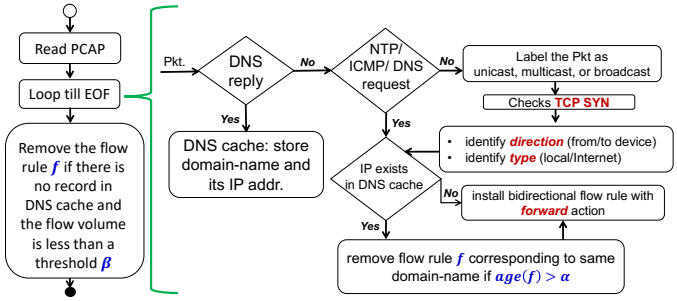


Figure 2: Algorithm for capturing device flows and inserting reactive rules.

SYN to identify if the device or the remote entity initiated the communication.

Flow translation to MUD: *MUDgee* uses the captured traffic flows to generate the MUD profiles for the devices. We, primarily convert each flow to a MUD ACE by considering the following:

Consideration 1: We use the DNS cache to reverse lookup the IP address of the remote endpoint to a domain name, if any.

Consideration 2: Some consumer IoTs, especially IP cameras, typically use the Session Traversal Utilities for NAT (STUN) protocol to verify that the user's mobile app can stream video directly from the camera over the Internet. If a device uses the STUN protocol over UDP, we must allow all UDP traffic to/from Internet servers because the STUN servers often require the client device to connect to different IP addresses or port numbers.

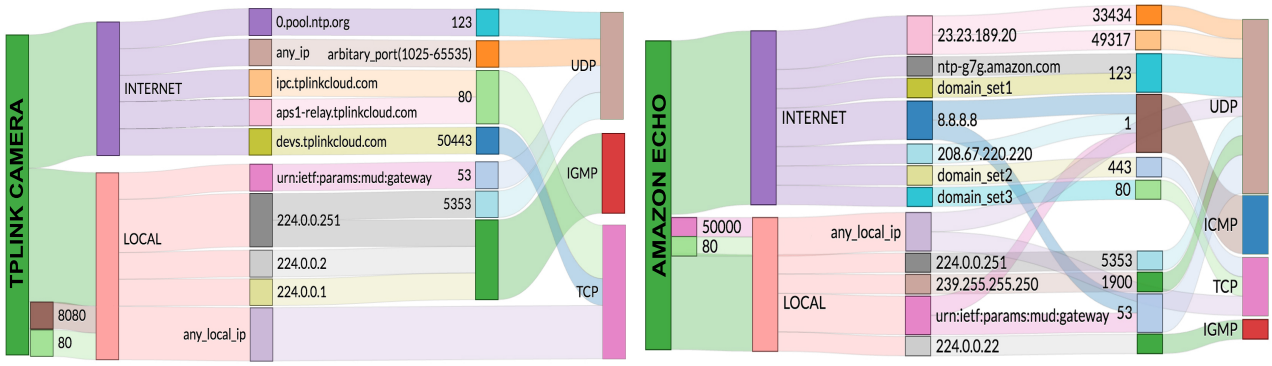
Consideration 3: We observed that several smart IP cameras communicate with many remote servers operating on the same port (*e.g.,* Belkin Wemo switch). However, no DNS responses were found corresponding to the server IP addresses. So, the device must obtain the IP address of its servers via a non-standard channel (*e.g.,* the current server may instruct the device with the IP address of the subsequent server). If a device communicates with several remote IP addresses (*i.e.,* more than our threshold value 5) all operating on the same port, we allow remote traffic to/from any IP addresses (*i.e.,* *) on that specific port number.

Consideration 4: Some devices (*e.g.,* TPLink plug) use the default gateway as the DNS resolver, and others (*e.g.,* Belkin WeMo motion) continuously ping the default gateway. The existing MUD draft maps local communication to fixed IP addresses through the controller construct. We consider the local gateway to act as the controller, and use the name-space `urn:ietf:params:mud: gateway` for the gateway.

The generated MUD profiles of the 28 consumer IoT devices we analyzed are listed in Table 2 and are publicly available at: `http://iotanalytics.unsw.edu.au/mud/`.

3.2 Insights and challenges

The Blipcare BP monitor is an example device with static functionalities. It exchanges DNS queries/responses with the local gateway and communicates with a single domain name over TCP port 8777. So its behavior can be locked down to a limited set of static flow rules. The majority of IoT devices that we tested (*i.e.,* 22 out of 28) fall into this category (marked in green in Table 2).



(a) TP-Link camera.

(b) Amazon Echo (Listing 1 for description of domain_set1-3).

Figure 3: Sankey diagrams of MUD profiles for: (a) TP-Link camera, and (b) Amazon Echo.

Listing 1: Example list of domains accessed by Amazon Echo corresponding to Figure 2(b).

```
domain_set1:
0.north-america.pool.ntp.org,
1.north-america.pool.ntp.org,
3.north-america.pool.ntp.org
domain_set2:
det-ta-g7g.amazon.com,
dcape-na.amazon.com,
softwareupdates.amazon.com,
domain_set3:
kindle-time.amazon.com,
spectrum.s3.amazonaws.com,
d28julafmv4ekl.cloudfront.net,
live-radio01.mediahubaustralia.com,
amzdigitaldownloads.edgesuite.net,
www.example.com
```

In Fig. 3, we use Sankey diagrams to represent the MUD profiles in a human-friendly way. Fig. 3(a) exemplifies the second category of our generated MUD profiles. The TP-Link camera accesses/exposes limited ports on the local network. It gets its DNS queries resolved, discovers local network using mDNS service over UDP 5353, probes members of certain multicast groups using IGMP, and exposes two TCP ports 80 (management console) and 8080 (unicast video streaming) to local devices. All these activities can be defined by a tight set of ACLs. But, over the Internet, the camera communicates to its STUN server (accessing an arbitrary range of IP addresses and port numbers shown by the top flow), to port numbers on specific endpoints including time synchronization with `pool.ntp.org`. Such IoT devices with static functionalities that are loosely defined, due to use of STUN protocol fall in to this second category (marked in blue in Table 2). This category device manufacturers can configure their STUN servers to use a specific set of endpoints and port numbers, instead of a wide and arbitrary range.

Amazon Echo, represents devices with complex and dynamic functionalities augmentable using custom recipes or skills. Such devices (marked in red in Table 2), are able to communicate with a growing range of endpoints on the Internet, which the original manufacturer cannot define in advance. For example, our Amazon Echo communicates with `meethue.com` over TCP 443, to interact with the Hue lightbulb in the test bed. It can also contact the news website `abc.net.au` when prompted by the user. For these type of devices, the biggest challenge is how manufacturers can dynamically update their MUD profiles to match the device capabilities. It is important to note that even the initial MUD profile would be useful for a minimum network communication permission set that can be amended over time.

Table 2: List of IoT devices for which we have generated MUD profiles. Devices with purely static functionality are marked in green. Devices with static functionality that is loosely defined (*e.g.,* due to use of STUN protocol) are marked in blue. Devices with complex and dynamic functionality are marked in red.

Type	IoT device
Camera	Netatmo Welcome, Dropcam, Withings Smart Baby Monitor, Canary camera, TP-Link Day Night Cloud camera, August doorbell camera, Samsung SmartCam, Ring doorbell, Belkin NetCam
Air quality sensors	Awair air quality monitor, Nest smoke sensor, Netatmo weather station
Healthcare devices	Withings Smart scale, Blipcare Blood Pressure meter, Withings Aura smart sleep sensor
Switches and Triggers	iHome power plug, WeMo power switch, TPLink plug, Wemo Motion Sensor
Lightbulbs	Philips Hue lightbulb, LiFX bulb
Hub	Amazon Echo, SmartThings
Multimedia	Chromecast, Triby Speaker
Other	HP printer, Pixstar Photoframe, Hello Barbie

4 MUD PROFILE VERIFICATION

Network operators should not allow a device to be installed without first checking its compatibility with organizational security policy. We develop a tool called *MUDdy* that uses the concept of Metagraphs to check a MUD profile for internal consistency, and its compliance with the organizational policies.

4.1 Syntactic correctness

A MUD profile consists of a YANG model which describes device-specific network behavior. In the initial version of MUD, this model is serialized using JSON [11]. A MUD profile is limited to the serialization of only a few YANG modules (*e.g.,* ietf-access-control-list) [11]. *MUDdy* will throw an invalid syntax exception when parsing

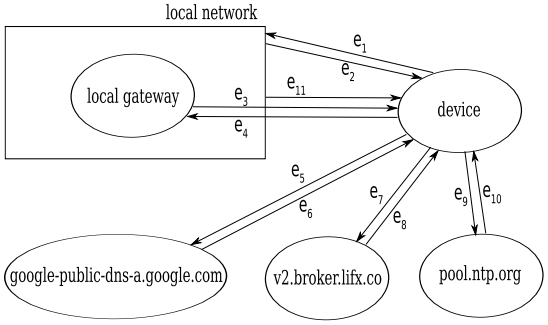


Figure 4: Metagraph model of a Lifx bulb's MUD policy. The policy describes permitted traffic flow behavior. Each edge label has attached a set of propositions of the metagraph. For example e_4=$\{protocol = 17, UDP.dport = 53, UDP.sport = 0-65535, action = accept\}$.

a MUD profile if it encounters any schema beyond these permitted YANG modules.

In addition, *MUDdy* also rejects MUD profiles containing IP addresses (in particular those with local significance). As per the IETF specification, publishers of MUD profiles are advised to use the abstractions provided in the specification and avoid using hardcoded IP addresses. *MUDdy* will also reject a MUD profile if it contains actions other than 'accept' or 'drop'.

4.2 Semantic correctness

Checking a MUD policy's syntax partly verifies its correctness. A syntactically correct policy must also be semantically correct; so we must check the policy, for instance, for inconsistencies. These can stem from two sources; (a) overlapping rules with different access-control actions (*i.e.,* intent-ambiguous rules); and/or (b) overlapping rules with identical actions (*i.e.,* redundancies). Our whitelisting model prevents the former by design. Redundancies are still possible and need to be checked.

We use metagraphs to model MUD policies in *MUDdy*. Metagraph algebras [2] can then be used to check the consistency of these policy models. Below is a summary of this process.

4.2.1 Policy modeling. Access-control policies are often represented using the five-tuple: source/destination address, protocol, source/destination ports [5, 10, 18]. We construct MUD policy metagraph models leveraging this idea. An example is shown in Fig. 4 for a Lifx bulb. Here, the source/destination addresses are represented by the labels *device*, *local-network*, *local-gateway* and a domain-name (*e.g., pool.ntp.org*). Protocol and ports are propositions of the conditional metagraph.

4.2.2 Policy definition and verification. We wrote *MGtoolkit* [20] – a package for implementing metagraphs – to define our policy models. *MGtoolkit* is implemented in Python 2.7. The API allows users to instantiate metagraphs, apply metagraph operations and evaluate results.

Mgtoolkit offers a `ConditionalMetagraph` class which extends a `Metagraph` and supports proposition attributes. The class inherits the base properties and methods of a `Metagraph` and additionally supports members to check consistency. We use this class to instantiate our MUD policy models and check their consistency.

4.2.3 Compatibility with best practices. Policy consistency checks partly verify if a MUD policy is semantically correct. It may also be necessary to check MUD policy semantics against local security policy or industry recommended practices: *e.g.,* ANSI/ISA- 62443-1-1, for compliance. Doing so, is critical when installing an IoT device in a critical network such as a SCADA network, where more restrictive practices are required to prevent serious injury of people, or even death!

SCADA best practices offer a wide spectrum of security policies, representative of various organizations that we check our MUD policies against them. For instance, they include policies for the highly protected SCADA zone (which, for instance, might run a power plant) as well as the more moderately restrictive 'Enterprise' zone.

We define a MUD rule to be SCADA (or Enterprise) zone compatible if its corresponding traffic flow complies with SCADA (or Enterprise) best practice policy. For instance, a MUD rule which permits a device to communicate with the local network using DNS is compatible with the Enterprise zone policy. However, a rule which allows a device to communicate with an Internet server using HTTP will violate the SCADA zone policy.

We have investigated the problem of policy comparison using formal semantics, in the SCADA domain for firewall access-control policies [21]. We adapt the methods and algebras developed there, to also check MUD policies against SCADA best practices. Key steps enabling these formal comparisons are summarized below.

Policies are mapped into a unique canonical decomposition. Policy canonicalisation can be represented through a mapping $c : \Phi \rightarrow \Theta$, where Φ is the policy space and Θ is the canonical space of policies. All equivalent policies of Φ map to a singleton. For $p^X, p^Y \in \Phi$, we note the following (the proof follows the definition)

Lemma 2. *Policies* $p^X \equiv p^Y$ *iff* $c(p^X) = c(p^Y)$.

MUD policy compliance can be checked by comparing canonical policy components. For instance

$$\text{Is } c(p^{device \rightarrow controller}) = c(p^{SCADA \rightarrow Enterprise}) \text{ ?}$$

A notation also useful in policy comparison is that policy P^A *includes* policy P^B. In SCADA networks, the notation helps evaluate whether a MUD policy is compliant with industry-recommended practices in [4, 26]. A violation increases the vulnerability of a SCADA zone to cyber attacks.

We indicate that a policy *complies* with another if it is more restrictive or included in and define the following

Definition 3 (Inclusion). *A policy* p^X *is* included *in* p^Y *on A iff* $p^X(s) \in \{p^Y(s), \phi\}$, i.e., *X either has the same effect as Y on s, or denies s, for all* $s \in A$. *We denote inclusion by* $p^X \subset p^Y$.

A MUD policy (*MP*) can be checked against a SCADA best practice policy (*RP*) for compliance using inclusion

$$\text{Is } p^{MP} \subset p^{RP} \text{ ?}$$

The approach can also be used to check if a MUD policy complies with an organization's local security policy, to ensure that IoT devices are plug and play enabled, only in the compatible zones of the network.

Table 3: MUD policy analysis summary for our test bed IoT devices (*Safe to install?* indicates where in a network (*e.g.,* Enterprise Zone, SCADA Zone, DMZ) the device can be installed without violating best practices, DMZ - Demilitarized Zone, Corp Zone - Enteprise Zone).

Device name	#MUD profile rules	#Redundant rules	Safe to install ?	% Rules violating SCADA Zone	% Rules violating Corp Zone
Blipcare bp	6	0	DMZ, Corp Zone	50	0
Netatmo weather	6	0	DMZ, Corp Zone	50	0
SmartThings hub	10	0	DMZ, Corp Zone	60	0
Hello barbie doll	12	0	DMZ, Corp Zone	33	0
Withings scale	15	4	DMZ, Corp Zone	33	0
Lifx bulb	15	0	DMZ, Corp Zone	60	0
Ring door bell	16	0	DMZ, Corp Zone	38	0
Awair air monitor	16	0	DMZ, Corp Zone	50	0
Withings baby	18	0	DMZ, Corp Zone	28	0
iHome power plug	17	0	DMZ	41	6
TPlink camera	22	0	DMZ	50	4
TPlink plug	25	0	DMZ	24	4
Canary camera	26	0	DMZ	27	4
Withings sleep	28	0	DMZ	29	4
Drop camera	28	0	DMZ	43	11
Net smoke sensor	32	0	DMZ	25	3
Hue bulb	33	0	DMZ	27	3
Wemo motion	35	0	DMZ	54	8
Triby speaker	38	0	DMZ	29	3
Netatmo camera	40	1	DMZ	28	2
Belkin camera	46	3	DMZ	52	11
Pixstar photo frame	46	0	DMZ	48	28
August door camera	55	9	DMZ	42	13
Samsung camera	62	0	DMZ	39	19
Amazon echo	66	4	DMZ	29	2
HP printer	67	10	DMZ	25	9
Wemo switch	98	3	DMZ	24	6
Chrome cast	150	24	DMZ	11	2

5 EVALUATION OF RESULTS

We used *MUDgee* to generate the MUD profiles for 28 IoT devices in our test bed. *MUDdy* then models each MUD policy automatically using a conditional metagraph. A high-level summary of these MUD profiles and their metagraphs are given in Table 3.

We identified MUD policy inconsistencies using *MUDdy*. Our adoption of an application whitelisting model detects inconsistencies and redundancies. There were, for instance, three redundant rules present in the Belkin camera's MUD policy (Table 3). These rules enabled ICMP traffic to the device from the local network as well as the local controller, making the policy inefficient.

Table 3 also shows the results of our best practice compliance checks of the MUD policies. For instance, a Blipcare blood pressure monitor can be safely installed in the Demilitarized zone (DMZ) or the Enterprise zone but not in a SCADA zone: 50% of its MUD rules violate the best practices, exposing the zone to potential cyber-attacks. Policy rules enabling the device to communicate with the Internet directly, trigger these violations.

In comparison, an Amazon echo speaker can only be safely installed in a DMZ. Table 3 shows that 29% of the device's MUD rules violate the best practices if it's installed in the SCADA zone. Only 2% of the rules violate if it's installed in the Enterprise zone. The former violation stems from rules which for instance, enable HTTP to the device. The latter originates from rules which enable ICMP to the device from the Internet.

MUDdy's ability to pinpoint to MUD rules which fail compliance, allows us to identify possible workarounds to overcome the failures. For instance, in the Belkin camera, local DNS servers and Web servers can be employed to localize the device's DNS and Web communications and achieve best practice compliance.

6 DISCUSSION

There are no existing tools which help IoT device manufacturers to automatically generate MUD profiles of their devices. Our tool – MUDgee – achieves this using a device's traffic trace as input.

A MUD profile generated from a device's traffic trace can be incorrect if the device is compromised, as the trace might include malicious flows. We use workarounds to reduce the impact of possible malicious flows by observing traffic flows for each device over a period of six months and extracting the most common flows. In addition, the generated MUD profile is limited to the input trace. Our tool can be extended by an API that allows manufacturers to add rules that are not captured in the PCAP trace.

Zigbee, Z-wave and bluetooth technologies are increasingly being used for IoT devices. Thus, these IoT devices come with a hub device capable of communicating with the Internet. In such cases, a MUD profile can be generated only for the hub.

At present, the MUD specification allows both accept and drop rules but does not specify priority, allowing ambiguity. This ambiguity is removed if only accept rules (*i.e.,* whitelisting) is used. Whitelisting means metagraph edges describe enabled traffic flows. So, the absence of an edge implies two metagraph nodes don't communicate with one another. But when drop rules are introduced, an edge also describes prohibited traffic flows, hindering easy visualization and understanding of the policy. We recommend the MUD proposal be revised to only support explicit 'accept' rules.

The MUD proposal also does not support private IP addresses, instead profiles are made readily transferrable between networks via support for high-level abstractions. For instance, to communicate with other IoT devices in the network, abstractions such as *same-manufacturer* is provided.

The MUD proposal however, permits the use of public IP addresses. This relaxation of the rule allows close coupling of policy with network implementation, increasing its sensitivity to network changes. A MUD policy describes IoT device behavior and should only change when its actual behavior alters and not when network implementation changes! Hardcoded public IP addresses can also lead to accidental DoS of target hosts. A good example is the DoS of NTP servers at the University of Wisconsin due to hardcoded IP addresses in Netgear routers [19]. We recommend that support for explicit public IP addresses be dropped from the MUD proposal.

The tools we propose allow to check if a MUD policy complies with an organizational policy, prior to deployment. This capability reduces the IoT acceptance testing effort required as we need not test the device in network segments where its MUD policy fails compliance.

7 CONCLUSION

In this paper, we have proposed a suite of tools that allow to automatically generate and formally verify IoT device MUD profiles, to ensure the MUD policies are consistent and compatible with organizational policies. We have used these tools to demonstrate how MUD can reduce the effort needed to secure IoT devices.

REFERENCES

[1] 2018. MUD maker. http://www.insecam.org/en/bycountry/US/. (2018).
[2] Amit Basu and Robert Blanning. 2007. *Metagraphs and their applications*. Vol. 15. Springer Science & Business Media.
[3] Sara Boddy and Justin Shattuck. 2017. *The Hunt for IoT: The Rise of Thingbots*. Technical Report. F5 Labs.
[4] Eric Byres, John Karsch, and Joel Carter. 2005. NISCC good practice guide on firewall deployment for SCADA and process control networks. *NISCC* (2005).
[5] Cisco Systems. 2013. *Cisco ASA Series CLI Configuration Guide, 9.0*. Cisco Systems, Inc.
[6] FCC. 2016. Federal Communications Comssion Response 12-05-2016. https://goo.gl/JdLofa. (2016).
[7] Ayyoob Hamza. 2018. MUDgee. https://github.com/ayyoob/mudgee. (2018).
[8] A. Hamza, D. Ranathunga, H. Habibi Gharakheili, M. Roughan, and V. Sivaraman. 2018. Clear as MUD: Generating, Validating and Applying IoT Behaviorial Profiles (Technical Report). *ArXiv e-prints* (April 2018). arXiv:cs.CR/1804.04358
[9] Scott Hilton. 2016. Dyn Analysis Summary Of Friday October 21 Attack. https://goo.gl/mCdQUF. (2016).
[10] Juniper Networks, Inc. 2016. *Getting Started Guide for the Branch SRX Series*. 1133 Innovation Way, Sunnyvale, CA 94089, USA.
[11] Eliot Lear, Ralph Droms, and Dan Romascanu. 2018. *Manufacturer Usage Description Specification (work in progress)*. Internet-Draft draft-ietf-opsawg-mud-18. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-ietf-opsawg-mud-18.txt
[12] Franco Loi, Arunan Sivanathan, Hassan Habibi Gharakheili, Adam Radford, and Vijay Sivaraman. 2017. Systematically Evaluating Security and Privacy for Consumer IoT Devices. In *Proc. ACM IoT S&P*. Dallas, Texas, USA.
[13] John Matherly. 2018. Shodan. [Online]. Available: https://www.shodan.io/. (2018).
[14] Diego M Mendez, Ioannis Papapanagiotou, and Baijian Yang. 2017. Internet of Things: Survey on Security and Privacy. *CoRR* abs/1707.01879 (2017). arXiv:1707.01879
[15] European Union Agency For Network and Information Security. 2017. Communication network dependencies for ICS/SCADA Systems. https://www.enisa.europa.eu/publications/ics-scada-dependencies. (2017).
[16] NIST. 2016. Systems Security Engineering. https://goo.gl/Qo9GfD. (2016).
[17] U.S. Department of Homeland Security. 2016. Strategic Principles For Securing the Internet of Things (IoT). https://goo.gl/PaXbc4. (2016).
[18] Palo Alto Networks, Inc. 2017. *PAN-OS Administrator's Guide, 8.0*. 4401 Great America Parkway, Santa Clara, CA 95054, USA.
[19] Dave Plonka. 2013. Flawed Routers Flood University of Wisconsin Internet Time Server. www.pages.cs.wisc.edu/~plonka/netgear-sntp/. (2013).
[20] Dinesha Ranathunga, Hung Nguyen, and Matthew Roughan. 2017. MGtoolkit: A python package for implementing metagraphs. *SoftwareX* 6 (2017), 91–93.
[21] Dinesha Ranathunga, Matthew Roughan, Phil Kernick, and Nick Falkner. 2016. Malachite: Firewall policy comparison. In *IEEE Symposium on Computers and Communication (ISCC)*. 310–317.
[22] Dinesha Ranathunga, Matthew Roughan, Phil Kernick, Nick Falkner, Hung Nguyen, Marian Mihailescu, and Michelle McClintock. 2016. Verifiable Policy-defined Networking for Security Management.. In *SECRYPT*. 344–351.
[23] Dinesha Ranathunga, Matthew Roughan, Hung Nguyen, Phil Kernick, and Nickolas Falkner. 2016. Case studies of scada firewall configurations and the implications for best practices. *IEEE Transactions on Network and Service Management* 13 (2016), 871–884.
[24] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2017. Characterizing and classifying IoT traffic in smart cities and campuses. In *Proc. IEEE INFOCOM workshop on SmartCity*. Atlanta, Georgia, USA.
[25] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. 2016. Smartphones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 195–200.
[26] Keith Stouffer, Joe Falco, and Karen Scarfone. 2008. Guide to Industrial Control Systems (ICS) security. *NIST Special Publication* 800, 82 (2008), 16–16.
[27] Cisco Systems. 2018. *Cisco 2018 Annual Cybersecurity Report*. Technical Report.
[28] Avishai Wool. 2010. Trends in firewall configuration errors: Measuring the holes in Swiss cheese. *IEEE Internet Computing* 14, 4 (2010), 58–65.
[29] PC World. 2018. Backdoor accounts found in 80 Sony IP security camera models. https://goo.gl/UUvc2x. (2018).