# Combining MUD Policies with SDN for IoT Intrusion Detection

Ayyoob Hamza
University of New South Wales
Sydney, Australia
ayyoobhamza@student.unsw.edu.au

Hassan Habibi Gharakheili
University of New South Wales
Sydney, Australia
h.habibi@unsw.edu.au

Vijay Sivaraman
University of New South Wales
Sydney, Australia
vijay@unsw.edu.au

## ABSTRACT

The IETF's push towards standardizing the Manufacturer Usage Description (MUD) grammar and mechanism for specifying IoT device behavior is gaining increasing interest from industry. The ability to control inappropriate communication between devices in the form of access control lists (ACLs) is expected to limit the attack surface on IoT devices; however, little is known about how MUD policies will get enforced in operational networks, and how they will interact with current and future intrusion detection systems (IDS). We believe this paper is the first attempt to translate MUD policies into flow rules that can be enforced using SDN, and in relating exception behavior to attacks that can be detected via off-the-shelf IDS. Our first contribution develops and implements a system that translates MUD policies to flow rules that are proactively configured into network switches, as well as reactively inserted based on run-time bindings of DNS. We use traces of 28 consumer IoT devices taken over several months to evaluate the performance of our system in terms of switch flow-table size and fraction of exception traffic that needs software inspection. Our second contribution identifies the limitations of flow-rules derived from MUD in protecting IoT devices from internal and external network attacks, and we show how our system is able to detect such volumetric attacks (including port scanning, TCP/UDP/ICMP flooding, ARP spoofing, and TCP/SSDP/SNMP reflection) by sending only a very small fraction of exception packets to off-the-shelf IDS.

## CCS CONCEPTS

• **Security and privacy** → *Intrusion detection systems*; • **Networks** → **Programmable networks**;

## KEYWORDS

IoT, MUD, SDN, Intrusion Detection

**ACM Reference Format:**
Ayyoob Hamza, Hassan Habibi Gharakheili, and Vijay Sivaraman. 2018. Combining MUD Policies with SDN for IoT Intrusion Detection. 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/10.1145/3229565.3229571


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

ACM ISBN 978-1-4503-5905-4/18/08…$15.00
https://doi.org/10.1145/3229565.3229571

## 1 INTRODUCTION

Billions of Internet-connected devices such as cameras, light-bulbs, medical sensors, and smoke alarms have severe security limitations [13] that make them potential victims of large scale cyber-attacks. Security firms [5] warn of attackers continuing to exploit IoTs to launch attacks in the form of DoS, DDoS, brute force, TCP SYN/UDP flooding, and scanning, while the growth of botnets [10] such as Mirai and Persirai that infect millions of IoTs is facilitating destructive campaigns of unprecedented magnitude.

Current practise for securing organizational networks is to rely on Intrusion Detection Systems (IDS) that inspect network traffic to detect attacks. However, such solutions are either extremely expensive if they are hardware-based, or unscalable to high data-rates if they are software-based. Further, the myriad variety of IoT devices, each with its own specific behavior and security vulnerabilities, makes it challenging for the IDS to distinguish normal from abnormal traffic that could be symptomatic of an attack.

In this paper[1] we aim to increase the scalability and efficacy of IDS using a combination of MUD and SDN. Manufacturer Usage Description (MUD) [12] is an emerging IETF framework for formally specifying the expected network behavior of an IoT device. IoT devices generally perform a specific function, and therefore have a recognizable communication pattern [16], which can be captured formally and succinctly as a MUD profile. Using the Software Defined Networking (SDN) paradigm, this formal behavioral profile can be translated to static and dynamic flow rules that can be enforced at run-time by the network – traffic that conforms to these rules can be allowed, while unexpected traffic inspected for potential instrusions. Such an approach dramatically reduces load on the IDS, allowing it to scale in performance and identify device-specific threats.

We first develop and implement a system that translates MUD policies to flow rules – some of these can be configured proactively into network switches, while others are inserted reactively based on run-time DNS bindings. We then simualte these flow rules for 28 consumer IoT devices with given MUD profiles using real traffic traces collected over several months, and evaluate aspects of system performance such as switch flow-table size and fraction of traffic that misses flow rules and therefore needs software inspection. Finally, we highlight the benefits and limitations of flow-rules derived from MUD in protecting IoT devices from internal and external network attacks, and demonstrate that our system is able to detect volumetric attacks (including port scanning, TCP/UDP/ICMP flooding, ARP spoofing, and TCP/SSDP/SNMP reflection) by sending only a very small fraction of exception packets to off-the-shelf IDS.

[1]This project was supported by the Australian Research Council (ARC) through Linkage Grant LP150100666 and Google Faculty Research Awards.

## 2 RELATED WORK

The IETF MUD specification [12] allows a device manufacturer to define the behavior of their device in the form of access control lists. This specification can be fed to an IDS to detect observed behavior that is not as specified, thereby indicating an anomaly or threat. In general, specification-based IDS for general-purpose devices is challenging as polices can be complex, unmanageable, and difficult to define manually as they are highly dependent on user activity [18]. However, since IoT devices are expected to be special-purpose, it is expected that they will have a small number of predictable traffic flows [16], which can be captured via relatively simple policies.

An example proposal for using network policy to secure IoT devices in an SDN environment is [19]; however, their policy grammar requires fine grained access controls that capture the state change (such as smoke sensed or windows opened) associated with IoT devices, which may be infeasible if manufacturers encrypt their sensing data, and undesirable for network operators who do not want to make semantic interpretations of sensing data; furthermore, their proposed theoretical framework has not been demonstrated in implementation. MUD, instead, allows operators to impose a tight set of rules (down to the port level) for each device, thus limiting its communication to only intended traffic flows. In HanGuard [6], the authors propose an access control model to block unauthorized access to IoT device from mobile devices. This proposed framework is only limited to local network traffic and mobile to IoT device communication. In [4], the authors propose a specification-based approach for a wireless sensor network, and expect the network operator to define the rules. We believe this is too onerous for the network operator; the behavior is better defined by the manufacturer of the IoT device, which is exactly what MUD intends.

MUD is a relatively new IETF framework [12], and the specification is still evolving. A valid MUD profile contains a root object called "access-lists" container 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. The MUD proposal defines how a MUD profile needs to be fetched and how the behavior of an IoT device needs to be defined. The MUD profile will be downloaded using a MUD url and this will be passed to the network by the IoT device through either DHCP option, LLDP, or IEEE 802.1ar. Our work [9] describes our open-source tool MUDgee [7] that is able to generate the MUD profile for an IoT device using its traffic trace as an input. To the best of our knowledge, our work is the first study on the use of MUD profiles for security enforcement and analysis in an SDN environment.

## 3 TRANSLATING MUD PROFILE TO DYNAMIC FLOW RULES

In this section, we outline our SDN-based system to enforce MUD policies and dynamically inspect exception traffic which is a small fraction of total packets to/from IoT devices. Our system uses as input MUD profiles of 28 consumer IoT devices that we have automatically generated by the MUDgee tool [9] using packet traces collected over several months. We next begin with the architecture of our system.

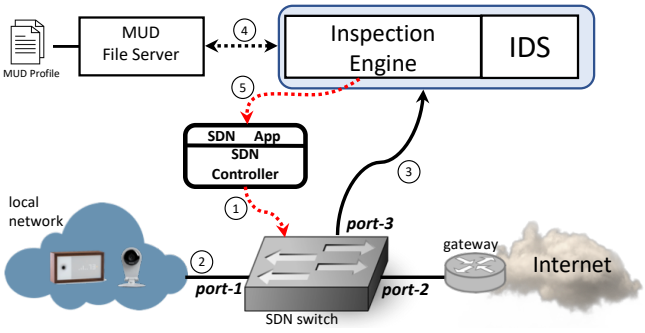



**Figure 1: Our SDN-based system architecture.**

### 3.1 System Architecture

Fig. 1 shows the functional blocks in our system architecture applied to a typical home or enterprise network. IoT devices on the left can communicate with the local network as well as with Internet servers via a gateway. Our system comprises a switch whose flow-table rules are managed dynamically by the SDN controller, a packet inspection engine, and a signature-based IDS.

The switch is initially configured by a default rule to mirror all traffic to the inspection engine on port-3, as shown by step ①. Packets from an IoT device, that has not yet been discovered, are forwarded (on port-2) and mirrored (on port-3), as shown by step ②. The inspection engine keeps track of already discovered devices on the local network (by maintaining a table of known MAC or IP addresses). Upon connection of a new device, its traffic is mirrored which helps the inspection engine detect this device and obtain the MUD URL from its initial packets (*e.g.,* DHCP request) shown by step ③ in Fig. 1. Thereafter, the inspection engine fetches the corresponding MUD profile from a MUD file server as shown by step ④. The MUD profile is stored (till its validity period), and its access control entries (ACEs) will be translated into a set of flow rules ("proactive"). Proactive rules then are inserted into the switch via the SDN controller as shown by step ⑤. Access control entries can be directly translated to flow rules, but it requires a notion of rules priority which is not captured by the current MUD draft. It is important to note that the order of flows becomes important when generating flow rules from a device MUD profile for preventing unwanted traffic to/from the device

We note that MUD specifications allow manufacturers to specify Internet endpoints by their domain-name. Therefore, MUD ACEs pertinent to Internet communications (with domain-name) can not be directly translated to flow rules. This means that we need to inspect DNS responses to find their bindings at run-time and store them in a DNS cache (maintained by the inspection engine). We also mirror all Internet traffic of individual IoT devices to check whether their remote IP address exists in the DNS cache; if yes, a "reactive" flow rule will be inserted into the switch. Note that packet inspection is conducted for flows associated with a domain-name to check whether the flow is compliant with policies specified by the device MUD profile.

**Table 1: Flow rules for Canary camera.**

| flow-id | sEth | dEth | typeEth | Source | Destination | proto | sPort | dPort | priority | action |
|---|---|---|---|---|---|---|---|---|---|---|
| I | <devMAC> | * | 0x888e | * | * | * | * | * | 20 | forward |
| II | <devMAC> | FF:FF:FF:FF:FF:FF | 0x0800 | * | * | * | * | 67 | 20 | forward |
| III | <gwMAC> | <devMAC> | 0x0800 | * | * | 1 | * | * | 20 | forward |
| IV | <devMAC> | <gwMAC> | 0x0800 | * | gateway IP | 17 | * | 53 | 20 | forward |
| V | <gwMAC> | <devMAC> | 0x0800 | gateway IP | * | 17 | 123 | * | 20 | forward |
| VI | <devMAC> | <gwMAC> | 0x0800 | * | gateway IP | 17 | * | 123 | 20 | forward |
| VII | <gwMAC> | <devMAC> | * | * | * | 17 | 53 | * | 20 | forward & mirror |
| VIII | <gwMAC> | <devMAC> | 0x0800 | h.canaryis.com | * | 6 | 80 | * | 11 | forward |
| IX | <devMAC> | <gwMAC> | 0x0800 | * | h.canaryis.com | 6 | * | 80 | 11 | forward |
| X | <gwMAC> | <devMAC> | 0x0800 | h.canaryis.com | * | 6 | 443 | * | 11 | forward |
| XI | <devMAC> | <gwMAC> | 0x0800 | * | h.canaryis.com | 6 | * | 443 | 11 | forward |
| XII | <gwMAC> | <devMAC> | 0x0800 | o.canaryis.com | * | 6 | 443 | * | 11 | forward |
| XIII | <devMAC> | <gwMAC> | 0x0800 | * | o.canaryis.com | 6 | * | 443 | 11 | forward |
| XIV | <gwMAC> | <devMAC> | 0x0800 | b.canaryis.com | * | 6 | 443 | * | 11 | forward |
| XV | <devMAC> | <gwMAC> | 0x0800 | * | b.canaryis.com | 6 | * | 443 | 11 | forward |
| XVI | <gwMAC> | <devMAC> | 0x0800 | i.canaryis.com | * | 6 | 443 | * | 11 | forward |
| XVII | <devMAC> | <gwMAC> | 0x0800 | * | i.canaryis.com | 6 | * | 443 | 11 | forward |
| XVIII | <gwMAC> | <devMAC> | 0x0800 | m.canaryis.com | * | 6 | 443 | * | 11 | forward |
| XIX | <devMAC> | <gwMAC> | 0x0800 | * | m.canaryis.com | 6 | * | 443 | 11 | forward |
| XX | <devMAC> | <gwMAC> | 0x0800 | * | * | * | * | * | 10 | forward & mirror |
| XXI | <gwMAC> | <devMAC> | 0x0800 | * | * | * | * | * | 10 | forward & mirror |
| XXII | * | <devMAC> | 0x0806 | * | * | * | * | * | 7 | forward |
| XXIII | <devMAC> | * | 0x0806 | * | * | * | * | * | 7 | forward |
| XXIV | * | * | * | * | * | * | * | * | 1 | forward & mirror |

It is important to note that IoT local communications are limited to a handful of standard flows which are typically fairly static (*e.g.,* ARP and DHCP), whereas Internet communications are carried by a larger number of flows (*e.g.,* up to 50 for some IoTs) whose endpoints are dynamic (*e.g.,* HTTPS to `h.canaryis.com`). We, therefore, permanently keep flow rules corresponding to local communications, and set a idle-timeout for reactive flow rules of Internet communications, for efficient use of limited-capacity TCAM inside the SDN switch.

As an example, in Table 1 we show flow rules translated from the MUD profile of Canary camera. Highlighted rows correspond to a snapshot of reactive flow rules since they vary over time. We show domain-names for Internet-based source/ destination to make it easier to visualize (in actual flow-table, IP addresses are used). It is seen that the camera communicates with five subdomains of `canaryis.com` – one is HTTP, and five are HTTPS. Non-highlighted rows correspond to proactive rules, and the default rule (at the very bottom of Table 1). As discussed earlier, proactive rules IV, XX, and XXI respectively mirror DNS replies as well as outgoing/incoming Internet traffic for each IoT device. Note that reactive rules would have a priority slightly higher than of flows mirroring Internet traffic, but lower than of the DNS reply flow. This way, we stop mirroring packets of Internet flows that are correctly captured by the inspection engine.

Lastly, we see essential rules for basic operations over the network such as: rules I, II and III respectively correspond to EAPoL (Extensible Authentication Protocol over LAN) packets, DHCP requests, and ICMP replies from default gateway; rules V and VI relate to direct communication between the device and the default gateway for NTP communications; rules XXII and XXIII specifically match on ARP packets from/to the device.

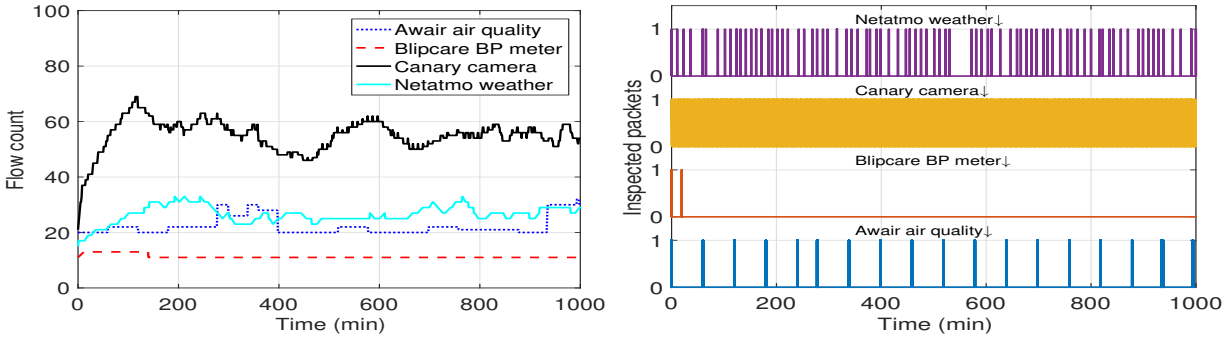
## 3.2 System Implementation and Performance Evaluation

We have implemented our proposed system that uses our generated MUD profiles (according to the syntax of current MUD specifications [12]), replays our IoT traffic traces [16] into an SDN simulator, and performs packet inspection for a fraction of total traffic using Snort [3], a widely deployed, open-source, signature based IDS.

**MUD Profiles:** We collected benign traffic traces of 28 consumer IoT devices over an 18-week period, and used our open source tool called MUDgee [7] to automatically generate MUD profile of these individual IoT devices – these profiles are available at: `http://iotanalytics.unsw.edu.au/mud/` [9]. Note that current IoT manufacturers do not yet support MUD specifications and mechanism. We therefore use the MAC address of IoT devices mapping them to their MUD profile in our repository/cite.

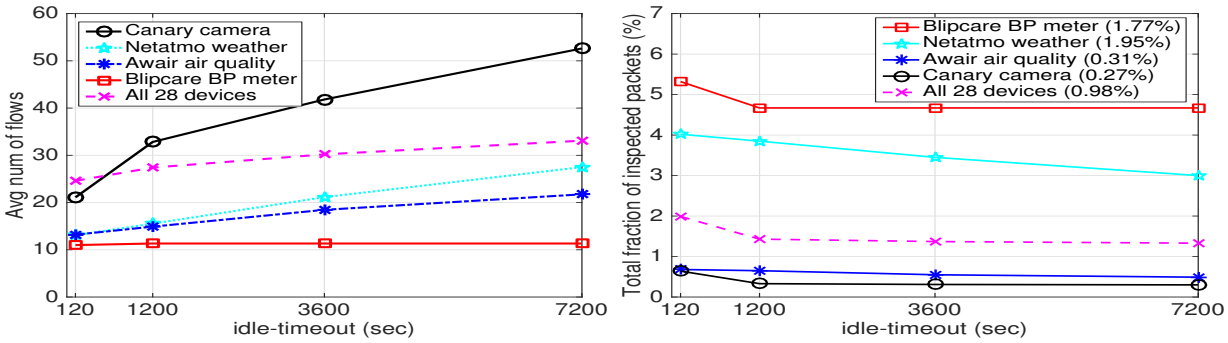
**SDN Simulator and IDS:** We wrote a native SDN simulator [8] that translates MUD policies into flow rules inside the SDN switch. Our simulator also takes an input PCAP trace, performs packet-by-packet service inside the switch with an inspection engine attached to it. The simulator also dumps a small fraction of packets (only those that are not matched to expected flows and need inspection) into an output PCAP file. We utilized `tcpreplay` tool to replay this output PCAP into Snort [15].

**Performance Metric and Evaluation:** As discussed earlier, inspection of certain packets is inevitable to capture a tight rule associated with each of TCP, UDP, or ICMP communications. But these inspections impose compute cost to our system. Another consideration is the dynamic management of flow-table for efficient usage of static resource in the switch. Therefore, we track the number of packets inspected, and the total number of flow rules at run time. The key tuning parameter is the idle-timeout




(a) Number of flows. (b) Inspected packets.

**Figure 2: Dynamics of (a) flow count, and (b) inspected packets, when idle-timeout equals to two hours (7200s).**




(a) Average number of flows . (b) Total fraction of inspected packets.

**Figure 3: Performance metrics: (a) average flow count, and (b) fraction of inspected packets.**

chosen for reactive flow rules. In our experiments, we vary the idle-timeout ranging from fairly short duration 2min to longer durations 20min, 60min, and 120min. We experimented with traffic traces of 28 devices collected over 4.5 months. To make it easier we plot our results for selected IoT devices including Awair air quality, Blipcare blood-pressure meter, Canary camera, and Netamo weather station.

Fig. 2 shows the dynamics of flow count and inspected packets over an experiment duration of 1000 minutes when we set the idle-timeout to 120min. It can be seen in Fig. 2(a) that we have on average 55 rules for canary camera (shown by solid black lines), followed by Netatmo weather and Awiar air quality with 27 and 22 average number of flows (shown by blue and cyan lines in the middle of the plot). However, for Blipcare (shown by dashed red lines), we initially see a total of 13 flow rules, then comes to 11 and stays at that level. This is because Blipcare generates traffic when only it is used by the user, and over the Internet it talks to one cloud-server `tech.carematix.com` using TCP 8777 (*i.e.,* two reactive flow rules) – thus no significant dynamics is intended for the Blipcare blood pressure(BP) meter.

Focusing on inspected packets in Fig. 2(b), we see that Canary has the highest rate of inspected packets (*i.e.,* 1.46 packet-per-min), followed by Netatmo weather station and Awair air quality with the rate of 1.05 and 0.16 inspected packets per minute respectively. Unsurprisingly, Blipcare BP monitor has only 3 initial packets inspected (*i.e.,* a DNS reply, and two packets to/from the remote TCP 8777).

We expect that choosing a low value for the idle-timeout would reduce the average flow count. But, such reduction comes at a cost of more inspected packets. This trade-off is shown in Fig. 3. In this set of experiments, we played the whole traffic captures (for duration of 4.5 months) of all IoT devices, and varied the idle-timeout for each run. Looking into the average flow count in Fig. 3(a), reducing the the idle-time-out from 2hr to 2min monotonically reduces the average number of flows by 60% for Canary camera, as shown by solid black lines and circle markers. A slightly lower reduction (*i.e.,* 52% and 41%) is observed for Netatmo weather station and Awair air quality monitor respectively. Considering all 28 devices (shown by dashed pink lines and cross markers), the average number of flows is about 33 when idle-timeout is set to 2hr, and is reduced to 25 when idle-timeout equals to 2min. We note that in real practice the idle-timeout should be tuned based on the TCAM capacity of the SDN switch and total number of IoT devices in the network.

Lastly, Fig. 3(b) shows the total fraction of packets inspected for each device. It is seen that the fraction of inspected packets initially falls when increasing the idle-timeout from 2min to 20 min and persists afterwards, except for Netatmo weather station. Considering all devices, in worst case (*i.e.,* idle-timeout = 2min) 1.99% of packets are inspected. We note that 0.98% (as mentioned in the legend) of total packets are DNS replies that are always inspected – half of inspected packets are DNS replies. Overall, results confirm that with our system the cost of packet inspection is kept at a minimum.

## 4 SECURITY ANALYSIS OF MUD

In this section, we start analyzing the MUD profile of real consumer IoT devices that we have generated, and highlight attack types that can be prevented. Then, we will use traces collected in our lab, when we launched a number of volumetric attacks to four of IoT devices, to show how our system can detect these attacks using off-the-shelf IDS in an operational environment.

**Table 2: Resistance of IoT devices against various categories of attacks.**

| IoT Device | To Internet | | | From Internet | | | From/To Local | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | UDP | TCP | ICMP | UDP | TCP | ICMP | ARP | UDP | TCP | ICMP |
| Amazon Echo | | | | | | | | | | |
| August doorbell camera | | | | | | | | | | |
| Awair air monitor | | | | | | | | | | |
| Belkin NetCam | | | | | | | | | | |
| Blipcare BP monitor | | | | | | | | | | |
| Canary camera | | | | | | | | | | |
| Chromecast | | | | | | | | | | |
| Dropcam | | | | | | | | | | |
| Hello Barbie | | | | | | | | | | |
| HP Envy printer | | | | | | | | | | |
| Hue lightbulb | | | | | | | | | | |
| iHome power plug | | | | | | | | | | |
| LiFX bulb | | | | | | | | | | |
| Nest smoke sensor | | | | | | | | | | |
| Netatmo welcome | | | | | | | | | | |
| Netatmo weather station | | | | | | | | | | |
| Pixstar Photoframe | | | | | | | | | | |
| Ring doorbell | | | | | | | | | | |
| Samsung SmartCam | | | | | | | | | | |
| SmartThings | | | | | | | | | | |
| TP-Link Day-Night | | | | | | | | | | |
| TP-Link plug | | | | | | | | | | |
| Triby Speaker | | | | | | | | | | |
| Wemo Motion Sensor | | | | | | | | | | |
| WeMo power switch | | | | | | | | | | |
| Withings Baby Monitor | | | | | | | | | | |
| Withings scale | | | | | | | | | | |
| Withings sleep sensor | | | | | | | | | | |

### 4.1 Resistance Against Attacks

We analyze the efficacy of MUD profiles considering four categories of attacks: To/From Internet, and To/From local network. Details of our analysis are shown in Table 2. For ease of visualization, we use color codes to indicate the device resistance: green for being secure, yellow for moderately-secure, and red for being insecure.

**To Internet:** Many IoT devices have been compromised due to insufficient (or even lack of) authentication enforcement, and have been employed in large scale botnet attacks to popular Internet servers [5]. Mirai, Brickerbot and Hajime are examples of major IoT-botnet-based attacks that were launched [10]. MUD helps isolate exception packets related to any attempts for botnet injection only if the device access controls are tightly defined, or the device does not expose telnet or ssh services. In case of already compromised devices, MUD is again able to isolate exception packets related to attack flows that are not specified by the device profile.

We observe that the MUD policy for five devices (*i.e.,* August doorbell camera, Belkin netcam, Ring doorbell, Samsung smart camera, and TPLink camera) cannot be tightly defined [9]. These devices allow peer-to-peer communications for streaming video to their corresponding mobile App (by using STUN or a stream server to initiate the handshake). These types of Internet communications necessitate the device to access an arbitrary range of IP addresses and port numbers, thus making the device vulnerable. Therefore, we mark these devices by red cells (in Table 2 under *To-Internet* heading) for their insecurity against either TCP- or UDP-based attacks.

IoT devices have been used as reflector to amplify TCP, UDP or ICMP floods to Internet-based victims [11, 14]. Another observation is that all of our IoT devices are vulnerable to DNS spoofing, since they don't implement DNSSEC, thus not able to verify the integrity of reply packets. However, enforcing devices to use local DNS servers (*e.g.,* in enterprise networks) can reduce the possibility of DNS spoofing attack. Yellow cells in Table 2 indicate the DNS-related vulnerability. Overall, 23 out of 28 IoT devices are moderately secure against reflection attacks on Internet-based victims.

**From Internet:** Attacks from the Internet are typically successful if a perimeter security by NAT/firewall is not present, or a malware exposes the local network to the Internet via port forwarding [17]. Shodan [2] runs frequent scanning to identify devices publicly accessible, and publishes their IP address and open ports. Furthermore, Insecam [1] lists publicly available video camera feeds from various countries. If proper policy has been applied then these unintended video feeds would have been blocked. Attackers, therefore, would be able to lookup these public repositories to hunt vulnerable IoT devices.

MUD specifications require manufacturers to separately define local and Internet communications for their device. This helps protect devices against port forwarding exposure – if the device offers a service to the local network, unintended remote access from the Internet can be prevented to some extent (excluding the spoofing of device server's IP address). Moreover, unintended endpoints from Internet can be limited if the MUD policies specify the intended endpoints tightly (*i.e.,* by a domain-name).

We see that three devices including August doorbell, Belkin camera, and Samsung smart camera are vulnerable to intrusions from Internet, as marked by red cells in Table 2. This is because, for example, August doorbell camera is allowing Internet traffic from TCP 443 (with no specific domain-name or IP address). This device serves the local network on TCP 80. This enables the attacker (with help of port forwarding) to access TCP 80 on the device by having the source TCP port equals to 443. We note that if the local or Internet service was on UDP (*i.e.,* mismatched transport protocols like in Ring doorbell and TP-Link camera), this type of attack can not succeed. We note that spoofing of IP for device's server is still possible, therefore all devices become somewhat vulnerable to attacks from Internet if they talk same protocol (UDP or TCP) on both local and Internet channels, as shown by yellow cells.

**From/To Local:** It has been shown [6, 13] that communication between an IoT device and its corresponding mobile App over the

local network is typically unauthenticated and/or unencrypted. Additionally, consumer IoT devices expose themselves to the local network using SSDP and/or mDNS protocols that are typically used for local discovery. This feature enables attackers (or their malware hosted on a local device) to get in the middle of communication to eavesdrop or impersonate (*i.e.,* man-in-the-middle can take full control over the device). These devices, with exposed local services, do not have any protection against flooding /denial-of-service attacks. We, therefore, deem devices that communicate over the local network to be at risk for this category of attacks, even though their MUD policies are well defined. This is because intended endpoints for local communications are not tightly defined. Moreover, all devices are vulnerable to ARP spoofing attack, as marked by red cells in Table 2. We note that MUD specifications allow a local controller to restrict the local communications to a limited group – this is more conducive for enterprise networks.

Lastly, we see IoT devices that have large number of open ports which are not used (*e.g.,* Telnet for HP Envy printer) [13], thus not specified in the MUD profile. These open ports make the device vulnerable, but having MUD policies in place would protect devices from being compromised by botnets such as Mirai, Bricker Bot, or Hajime that actively scan for open telnet ports and launch brute-force attack [10].

## 4.2 Detection of Volumetric Attacks using Snort IDS

We now show the efficacy of our system when real attacks are launched on IoT devices. We use traffic traces that we collected in November 2017, comprising a number volumetric attacks including reflection/amplification (SNMP, SSDP, TCP SYN, and Smurf), flooding (TCP SYN, Fraggle, and Ping of death), ARP spoofing, and port scanning that were launched on four IoT devices including Belkin Netcam, Wemo motion sensor, Samsung smart-cam and Wemo switch (listed in Table 3). These attacks were sourced from within the local network as well as from the Internet. For Internet sourced attacks, we had enabled port forwarding (emulating a malware behavior) on the gateway.

We found that a majority of attacks didn't match on any of intended flow rules translated from MUD policies, thus were mirrored for inspection. For example, out of 14 attacks on Samsung camera, traffic of 9 attack types were mirrored for inspection – the rate of mirrored packets rises significantly, proportional to the rate of attack traffic. This measure was 13 out of 14 for Wemo motion sensor. For Samsung camera, attacks including direct TCP flooding, ARP spoofing, Ping of Death, and reflective TCP and ICMP from within local network were successful to pass undetected (*i.e.,* get matched to intended flow rules). However, in case of Wemo motion, only ARP spoofing was successful. This is because our TCP related attacks targeted port numbers that are not defined in the MUD profile of Wemo motion, the same for ICMP. Whereas for Samsung camera, target port numbers were defined as intended service, allowing attacks to pass. We can enhance our system by monitoring the behavior of intended flows using flow-level telemetry – this is planned for future work. Visibility into activity of flows, for example on/off pattern or volume/rate, would reveal malicious traffic covered by intended rules. Furthermore, our observation for Wemo switch and Belkin Netcam was similar to Samsung smart camera. Interestingly,

**Table 3: List of attacks launched against our IoT devices.(*L: local, D: device, I: Internet*)**

| | | Device | | | | Category | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Attacks | | Wemo motion | Wemo switch | Belkin Netcam | Samsung smart-cam | L→D | L→D→L | L→D→I | I→D→I | I→D→L | I→D |
| Reflection | SNMP | | | | ✓ | | ✓ | ✓ | ✓ | | |
| | SSDP | ✓ | | ✓ | | | ✓ | ✓ | ✓ | | |
| | TCP SYN | ✓ | ✓ | ✓ | ✓ | | ✓ | | | ✓ | |
| | Smurf | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | | |
| Direct | TCP SYN | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | ✓ |
| | Fraggle | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | ✓ |
| | ICMP | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | |
| ARP spoof | | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | |
| Port Scan | | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | |

MUD profile was able to protect all four devices against attacks traversing the gateway (*i.e.,* to/from Internet).

We then fed the mirrored packets to Snort IDS to see how unsuccessful attack traffic can be detected. Snort was only able to detect SSDP/SNMP reflection attacks from Internet as well as a few port scanning instances. However, port scanning attacks were flagged by incorrect alarms, *e.g., suspicious inbound traffic for mssql port 3306.* This is because existing signatures of Snort primarily relate to generic servers, not specific to IoT devices.

Interestingly, we discovered from Snort output that there were other attacks from the Internet (not from our source). These attacks detected by the Snort include an attack from blacklisted remote IP address `66.240.192.138`, DNS amplification attack on Wemo motion, and `nmap` port scanning. This was essentially because we had enabled port forwarding for local services offered by each IoT device. For example, TCP port 554, 81, 53, and 49153 for Samsung camera, Belkin Netcam, Wemo motion, and Wemo switch respectively. The detailed outputs from IDS inform the network operator about vulnerable/exposed ports in the network. This also highlights that the need for further inspection of selected packets instead of blocking them.

## 5 CONCLUSION

In this paper, we have combined MUD and SDN to make an efficient and scalable intrusion detection system for IoT devices. We have implemented a system that translates MUD policies of 28 consumer IoT devices to flow rules configured into the network switch. Some of rules are inserted pro-actively while others are configured re-actively based on run-time DNS bindings. We have evaluated the performance of our system in terms of flow-table size and fraction of traffic misses using real traffic traces collected over several months. Lastly, we have highlighted the benefits and limitations of MUD in limiting attack surface to various IoT devices, and demonstrated that how our system can detect volumetric attacks by inspecting a very small fraction of exception packets using off-the-shelf IDS. We have shown that with MUD in place, compromising IoT devices becomes non-trivial for attackers. In future, MUD can be used as a baseline to build sophisticated anomaly-/signature-based IDSs.

## REFERENCES


[1] 2018. Insecam. http://www.insecam.org/
[2] 2018. Shodan. https://www.shodan.io/.
[3] 2018. Snort. https://snort.org/
[4] J. P. Amaral, L. M. Oliveira, J. J. Rodrigues, G. Han, and L. Shu. 2014. Policy and network-based intrusion detection system for IPv6-enabled wireless sensor networks. In *Proc. IEEE International Conference on Communications (ICC)*. Sydney, NSW, Australia, 1796–1801.
[5] S. Boddy and J. Shattuck. 2017. *The Hunt for IoT: The Rise of Thingbots.* Technical Report. F5 Labs.
[6] S. Demetriou, N. Zhang, Y. Lee, X. Wang, C. A. Gunter, X. Zhou, and M. Grace. 2017. HanGuard: SDN-driven protection of smart home WiFi devices from malicious mobile apps. In *Proc. ACM Conference on Security and Privacy in Wireless and Mobile Networks*. Boston, Massachusetts.
[7] A. Hamza. 2018. MUDgee. https://github.com/ayyoob/mudgee
[8] A. Hamza. 2018. SDN pcap simulator. https://github.com/ayyoob/sdn-pcap-simulator
[9] A. Hamza, D. Ranathunga, H. Habibi Gharkheili, M. Roughan, and V. Sivaraman. 2018. Clear as MUD: Generating, Validating and Applying IoT Behaviorial Profiles. In *Proc. ACM workshop on IoT S&P*. Budapest, Hungary.
[10] Cisco Systems Inc. 2017. *Midyear Cybersecurity Report.* Technical Report.
[11] Cisco Systems Inc. 2018. *Annual Cybersecurity Report.* Technical Report.
[12] E. Lear, R. Droms, and D. 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
[13] F. Loi, A. Sivanathan, H. Habibi Gharakheili, A. Radford, and V. Sivaraman. 2017. Systematically Evaluating Security and Privacy for Consumer IoT Devices. In *Proc. ACM workshop on IoT S&P*. Dallas, Texas, USA.
[14] M. Lyu, D. Sherratt, A. Sivanathan, H. Habibi Gharakheili, A. Radford, and V. Sivaraman. 2017. Quantifying the Reflective DDoS Attack Capability of Household IoT Devices. In *Proc. ACM WiSec*. Boston, Massachusetts.
[15] M. Roesch. 1999. Snort - Lightweight Intrusion Detection for Networks. In *Proc USENIX Conference on System Administration*. Seattle, Washington.
[16] A. Sivanathan, D. Sherratt, H. Habibi Gharakheili, Adam Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. 2017. Characterizing and classifying IoT traffic in smart cities and campuses. In *Proc. IEEE INFOCOM workshop on SmartCity*. Atlanta, Georgia, USA.
[17] V. Sivaraman, D. Chan, D. Earl, and R. Boreli. 2016. Smart-phones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 195–200.
[18] R. Sommer and V. Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proc. IEEE Security and Privacy (SP)*. Berkeley, CA, USA.
[19] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu. 2015. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-Things. In *Proc ACM Workshop on HotNets*. Philadelphia, PA, USA.