Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity

Ayyoob Hamza[1], Hassan Habibi Gharakheili[1], Theophilus A. Benson[2], and Vijay Sivaraman[1]

UNSW Sydney[1], Brown University[2]

ABSTRACT

Smart environments equipped with IoT devices are increasingly under threat from an escalating number of sophisticated cyber-attacks. Current security approaches are inaccurate, expensive, or unscalable, as they require static signatures of known attacks, specialized hardware, or full packet inspection. The IETF Manufacturer Usage Description (MUD) framework aims to reduce the attack surface on an IoT device by formally defining its expected network behavior. In this paper, we use SDN to monitor compliance with the MUD behavioral profile, and develop machine learning methods to detect volumetric attacks such as DoS, reflective TCP/UDP/ICMP flooding, and ARP spoofing to IoT devices.

Our **first** contribution develops a machine for detecting anomalous patterns of MUD-compliant network activity via coarse-grained (device-level) and fine-grained (flow-level) SDN telemetry for each IoT device, thereby giving visibility into flows that contribute to a volumetric attack. For our **second** contribution we measure network behavior of IoT devices by collecting benign and volumetric attacks traffic traces in our lab, label our dataset, and make it available to the public. Our **last** contribution prototypes a full working system (built with an OpenFlow switch, Faucet SDN controller, and a MUD policy engine), demonstrates its application in detecting volumetric attacks on several consumer IoT devices with high accuracy, and provides insights into cost and performance of our system. Our data and solution modules are released as open source to the community.

SOSR '19, April 3–4, 2019, San Jose, CA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6710-3/19/04. . . $15.00
https://doi.org/10.1145/3314148.3314352

CCS CONCEPTS

• **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**; **Denial-of-service attacks**; • **Networks** → **Programmable networks**;

1 INTRODUCTION

The proliferation of insecure Internet-connected devices is making it easy [26] for cyber-hackers to attack home, enterprise, and critical infrastructures at large scale. Recent reports [25] show that attackers continue to exploit insecure IoT devices to launch volumetric attacks in the form of DoS, DDoS, brute force, and TCP SYN/UDP flooding. Moreover, the progression of botnets [5, 33] such as Mirai and Persirai, infecting millions of IoT devices, is enabling destructive cyber-campaigns of unprecedented magnitude to be launched.

Network operators today lack the tools to know whether the IoT devices connected to their network are behaving normally or have been cyber-breached [33]. In fact, most operators would not even know what "normal" behavior is, given the myriad IoT devices in the market with different functionalities and from various manufacturers. To alleviate this issue, the IETF has recently proposed the Manufacturer Usage Description (MUD) [14] framework, which requires vendors to formally specify the intended network behavior of the IoT devices they put into the market. MUD specification has not been adopted yet. But, large organizations and critical infrastructure are increasingly deploying IoT devices at scale, and thus demand an automated enforcement of baseline security for multitude of IoT devices across their network. Note that the MUD specification is approved for publication as an RFC which motivates manufacturers to embrace this standard for competing with other players such as Google [6] and Cisco [2] who are strongly backing this standard. This specification allows an operator to lock down the network traffic of the IoT device using access control lists (ACLs) derived from its MUD profile; indeed, our recent work [8] has used software defined networking (SDN) as a vehicle to translate MUD profiles into static and dynamic flow rules that can be applied at run-time on OpenFlow-capable switches to limit IoT traffic, thereby significantly reducing their attack surface.

The focus on this paper is on attacks that can be launched on IoT devices while still conforming to their MUD profiles. Specifically, we consider volumetric attacks that are not prevented by the MUD profile, since it's ACLs simply allow or deny traffic, and there is no provision to limit rates. In this paper we show that a range of volumetric attacks (including ones directly on the IoT device and ones that reflect off the IoT device) are feasible in spite of MUD policy enforcement in the network. Fending off such attacks requires more sophisticated machinery that monitors the level of activity associated with each policy rule to detect anomalies. We leverage earlier studies [9] showing that IoT devices exhibit identifiable traffic patterns (with limited diversity of activity cycles and protocol use), making it feasible to develop machine learning methods for detecting abnormal behavior, which is otherwise difficult for general-purpose computers that exhibit much wider diversity in network behavior [23]. The specific contributions of our work are as follows:

First, we develop a system that learns expected patterns of MUD-compliant behavior for each IoT device by monitoring its activity via a combination of coarse-grained (per-device) and fine-grained (per-flow) SDN telemetry at various time scales, and has the capability to detect volumetric attacks and the specific traffic streams that contribute to it. **Second**, we measure network behavior of real IoT devices under normal and attacks conditions – we subject our devices to volumteric attacks including ARP spoof, TCP SYN flooding, Fraggle, Ping of Death, and SSDP/SNMP/TCP /ICMP reflection. We label our traffic traces (*i.e.,* benign and attack) collected in our lab and make our data openly available to the research community. **Lastly**, we prototype our system (using an OpenFlow switch, Faucet SDN controller, and a MUD policy engine), and quantify the efficacy of our scheme in detecting volumetric attacks on several IoT devices, and release our solution modules as open source to the community.

2 RELATED WORK

Intrusion detection systems for computer networks have been studied extensively by the research community, and look either for *signatures* of known attacks, *anomalies* indicative of deviation from normal behavior, or *specification* of allowed traffic. However, there are limited studies on intrusion detection solution for IoT devices [22]. Security of IoT devices is increasingly becoming important due to their limited protection, if any.

Signatures-based intrusion detection: Nearly all deployed solutions, including software tools like Bro[37] and Snort [40], and commercial hardware appliances belong to this category. There are studies that apply signature-based intrusion detection/prevention in SDN environments [11, 45]. Signature-based approach is not sufficient for addressing the new and growing security issues that come with the proliferation of IoT devices. Attack signatures can not be developed for a growing number of IoT devices at scale. We will show (in §5.6) that signature-based tools are only able to detect limited number of attacks (to IoTs) those that are common for general purpose computers.

Anomaly-based intrusion detection: Anomaly detection holds promise as a way of detecting new and unknown threats, but despite extensive academic research [28], has had very limited success in operational environments. The reasons for this are manifold [23]: "normal" network traffic can exhibit much more diversity than expected (particularly for general-purpose devices); obtaining "ground truth" on attacks in order to train the classifiers is difficult; evaluating outputs can be difficult due to the lack of appropriate datasets; false positives incur a high cost on network administrators to investigate; and there is often a semantic gap between detection of an anomaly and actionable reports for the network operator. There are many studies that employ either entropy-based [34, 42] or machine learning [21, 43, 44] techniques to detect new attacks in SDN environments. Works in [21, 43, 44] use two-class classification (*i.e.,* benign and attack). This contradicts with the expectation from anomaly-based technique that needs to flag deviation from normal behavior [23]. Authors of [21, 44] propose to use features including flow-level stats (*i.e.,* packet/byte count and duration), percentage of bidirectional flows, growth rate of unidirectional flows, and growth rate of number of unique ports, for their classifier. Work in [43] employs deep learning algorithms using a similar set of features to classify normal and abnormal traffic. Authors of [24] applied a techniques in [21] to IoT devices. However, their evaluation is limited to simulated traffic in mininet that does not represent behavior of real IoT devices.

Specification-based intrusion detection: Specifying allowed rules for general-purpose devices has been a challenge [20] as traffic pattern highly depend on applications and user activity. In [17], 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 IETF's MUD proposal [14] intends. Our work in [8] is the first that proposes an IDS for IoT devices using a combination of MUD and SDN and detects attack flows that are not specified in the device formal MUD profile. This work employs a collection of anomaly workers each trained by MUD behavioral profile that detect attacks that conform with MUD profile but display a deviated traffic profile.

3 ANOMALY DETECTION USING SDN

In this section we describe our attack detection solution, including a brief summary of MUD profile (§3.1), the SDN-based system architecture (§3.2), the anomaly detector (§3.3).

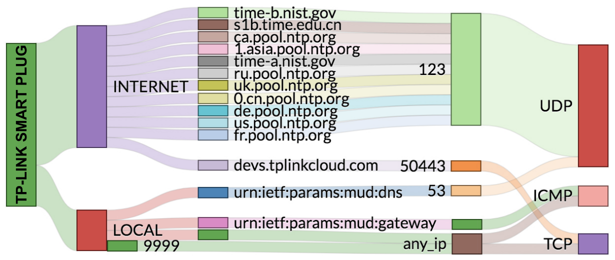


Figure 1: TP-Link smart plug.

3.1 MUD Profile

MUD is a relatively new IETF framework [14], 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 (*e.g.,* via DHCP option). IoT device manufacturers have not yet provided MUD profiles for their devices. But, we released the MUD profiles (automatically generated from packet traces) for 28 consumer IoT devices [7] – in this paper, we use a subset of those profiles corresponding to devices that we experiment with.

We use Sankey diagram in Fig. 1 to represent the MUD profile of TP-Link smart plug. It is seen that this IoT device exchanges DNS queries/responses with the local DNS server, communicates with a range of Internet domains for NTP services (*i.e.,* UDP port 123), and talks to its manufacturer server (*i.e.,* `devs.tplinkcloud.com`) over TCP port 50443. In addition, the TP-Link smart plug exposes TCP port 9999 on the local network to its mobile app for user interaction with the device. We also see that the smart plug and its mobile app send periodic pings to the gateway and the plug respectively for connectivity check.

3.2 SDN-Based System Architecture

An IoT device advertises its MUD profile through a MUD URL. According to the MUD standard, there are three options for emitting the MUD URL namely DHCP, LLDP, and X.509 [14]. If a device is compromised, the MUD URL emitted can potentially be spoofed in case of either DHCP or LLDP. But, it is secure when the device uses the X.509 extension since the MUD URL is added to the certificate by the manufacturer. This means that the MUD URL emitted by an X.509 device can not be spoofed without detection, even if the device is exploited.

Fig. 2 shows the functional blocks in our architecture applied to a typical home or enterprise network. IoT devices on the left can communicate with other devices on the local network and also with Internet servers via a gateway. The architecture comprises an SDN switch whose flow-table rules will be managed dynamically, a MUD engine in conjunction with our App on the SDN controller, a MUD collector and a combination of anomaly-based and specification-based threat detection. These components interact with each other to dynamically manage the flow-table rules inside the switch whilst monitoring network activity of various flows pertinent to each device.

It is important to note that our SDN switch does not send any data packets to the controller; instead, packets that need to be inspected in software are sent as copies on a separate interface of the switch, to which a software inspection engine is attached. This protects the controller from overload from the data-plane, allowing it to scale to high rates and to service other SDN applications.

The operational flow of events in Fig. 2 is as follows: the switch is initially configured by a default rule, as shown by step ①, to mirror packets (on port-3), as shown by step ②, that reveal the device identity (*e.g.,* DHCP), and all other packets are forwarded normally (on either port-1 or port-2 depending on local or Internet communications). We note that DHCP contains the MAC of the device and may provide a *mud-url* if the device manufacturer adopts the MUD standard. This helps our MUD policy engine discover a new IoT device connected to the network – the MUD engine keeps track of already discovered devices. Thereafter, the MUD engine fetches the corresponding MUD profile from a MUD file server, as shown by step ③ – The MUD engine stores the fetched profile till its validity period. In real scenario, the MUD file server is operated by the manufacturer who can keeps updating the profile if needed (*e.g.,* firmware upgrade).

The MUD policy engine translates access control entries (ACEs) of the MUD profile into a set of flow rules (explained in §3.3). We note that MUD specifications allow manufacturers to specify Internet endpoints by their domain-name in ACEs. These ACEs can not be directly translated to flow rules and need further inspection to infer DNS bindings. The MUD engine, therefore, inserts proactive flow entries, shown by step ④, for ACEs with known endpoints (*i.e.,* static IPs) while others are reactively inserted based on run-time DNS bindings. We set an idle-timeout for reactive flow rules that are associated with a domain name, as DNS bindings could be dynamic.

Following insertion of device flow rules, we mirror all DNS responses in addition to exception packets that do not match on any proactive or reactive flow rule (*i.e.,* default mirror of local and Internet traffic). These mirrored packets are inspected by a module inside the MUD policy engine

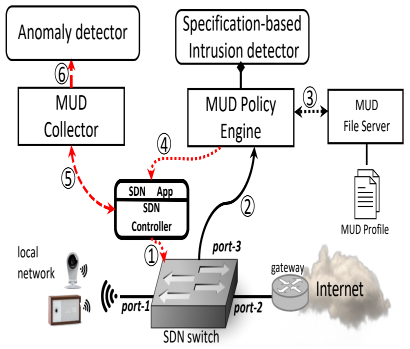


Figure 2: Our SDN-based intrusion detection system.

Table 1: Flow rules for TP-Link Smart Plug.

flow-id	sEth	dEth	typeEth	Source	Destination	proto	sPort	dPort	priority	action
a.1	`<gwMAC>`	`<devMAC>`	`0x0800`	[ntp domain names]	*	17	123	*	20	forward
a.2	`<devMAC>`	`<gwMAC>`	`0x0800`	*	[ntp domain names]	17	*	123	20	forward
b.1	`<gwMAC>`	`<devMAC>`	`0x0800`	devs.tplinkcloud.com	*	6	50443	*	20	forward
b.2	`<devMAC>`	`<gwMAC>`	`0x0800`	*	devs.tplinkcloud.com	6	*	50443	20	forward
c	`<devMAC>`	*	`0x888e`	*	*	*	*	*	11	forward
d.1	`<devMAC>`	`FF:FF:FF:FF:FF:FF`	`0x0800`	*	*	17	*	67	11	forward
d.2	`<gwMAC>`	`<devMAC>`	`0x0800`	*	*	17	67	*	11	forward
e.1	`<gwMAC>`	`<devMAC>`	`0x0800`	gateway IP	*	1	*	*	11	forward
e.2	`<devMAC>`	`<gwMAC>`	`0x0800`	*	gateway IP	1	*	*	11	forward
f.1	`<devMAC>`	`<gwMAC>`	`0x0800`	*	gateway IP	17	*	53	11	forward
f.2	`<gwMAC>`	`<devMAC>`	`0x0800`	gateway IP	*	17	53	*	11	forward & mirror
g.1	`<devMAC>`	`<gwMAC>`	`0x0800`	*	*	*	*	*	10	forward & mirror
g.2	`<gwMAC>`	`<devMAC>`	`0x0800`	*	*	*	*	*	10	forward & mirror
h.1	*	`<devMAC>`	`0x0806`	*	*	*	*	*	7	forward
h.2	`<devMAC>`	*	`0x0806`	*	*	*	*	*	7	forward
i.1	`<devMAC>`	*	`0x0800`	*	*	6	9999	*	6	forward
i.2	*	`<devMAC>`	`0x0800`	*	*	6	*	9999	6	forward
j.1	`<devMAC>`	*	`0x0800`	*	*	1	*	*	6	forward
j.2	*	`<devMAC>`	`0x0800`	*	*	1	*	*	6	forward
k	*	`<devMAC>`	`0x0800`	*	*	*	*	*	5	forward & mirror

called "specification-based intrusion detector" to detect traffic that does not conform to the MUD specification. This module maintains an intermediate set of rules translated from the MUD profile along with a DNS cache (all in memory) to verify if headers of the mirrored packet match with intended profile of the device.

We note that a sophisticated attack traffic can still pass undetected [8] using spoofing techniques. In order to identify such threats we monitor the activity of all device flows that specified by the MUD profile. To do so, we use MUD collector to periodically pull flow counters (denoted by step ⑤ in Fig. 2) from the switch, compute the attributes for each device, and stream them to the corresponding anomaly detector, as denoted by step ⑥. In what follows we explain our features and anomaly detection algorithm.

3.3 Anomaly Detection Method

We develop a machine learning technique (explained in §3.3.4) to determine if an IoT device is involved in a volumteric attack or not (the "attack detection"), and if so, to determine the flow that contributes to the attack (the "attack flow(s) identification"). Our objective is to train our machine with benign traffic profile of each device, and detect attacks by detecting deviation from expected traffic pattern in a device flows that are defined by the device MUD profile.

3.3.1 Device Flow Rules. As briefly explained in §3.2, we translate a given MUD profile [7] to flow-table rules and monitor the expected traffic of the device. For example in Table 1, we show flow rules translated from the MUD profile of the TP-Link smart plug. Highlighted rows (*i.e.,* flow-IDs *a.1*, *a.2*, *b.1* & *b.2*) correspond to a snapshot of reactive flow rules as they vary over time. Note that reactive rules have a priority slightly higher than of flows mirroring the Internet traffic. This way, we stop mirroring packets of Internet flows that conform to the MUD profile. We show domain-names for Internet source/destination to make it easier to visualize (in actual flow-table IP addresses are used). Non-highlighted rows correspond to proactive rules. Proactive rules *f.2*, *g.1* & *g.2*, and *k*, respectively mirror: DNS replies, default Internet traffic from/to, and the local traffic to this device. We only have one direction of local traffic (*i.e.,* to the IoT device) to avoid conflicting with matching on flows of other devices. Mirroring traffic coming to the device allows us to inspect any attempt to access standard vulnerable services such as Telnet, SSH, or HTTP that may be open on IoT devices.

3.3.2 Detection Machines. We now present our two-stage method for detecting anomalies. For each device, we train a specific machine based on its MUD profile. We note that our method is able to detect an attack on the device and also to identify flow(s) contributing to the attack. Fig. 3 depicts the machine structure specific to TP-Link smart plug for detection of anomalies caused by volumetric attacks traffic. In this structure, we first identify whether anomaly occurs over local or Internet communication using two channel workers (stage-1) – these workers utilize coarse-grained (device-level) telemetry. A true alarm from the stage-1 workers would trigger corresponding workers at stage-2 where we identify the flow over which the attacker causes the anomaly using specialized flow workers, each corresponding to a flow in Table 1 – these workers utilize fine-grained (flow-level) telemetry. Note that an attack is triggered only when workers of the two stages indicate an anomalous behaviour. In §5.3 we will show that the combination of the two stages allows us to reduce false positives while maintaining high true positives.

3.3.3 Features Extractor. Having captured device flow rules of each IoT device (*e.g.,* Table 1 for TP-Link smart plug), we now extract corresponding features of network activity.

We use the count of packets and bytes provided by each flow rule as features. This is because that the size of the packet can vary for a given protocol. For example, Fig. 4(a) shows the scatter plot of packet count versus byte count of DNS downstream traffic captured for Samsung camera over one month in our lab. It is seen that for a given packet count, the byte count varies in a range of 1 KB or more – packet count and byte count are not highly correlated. However, for

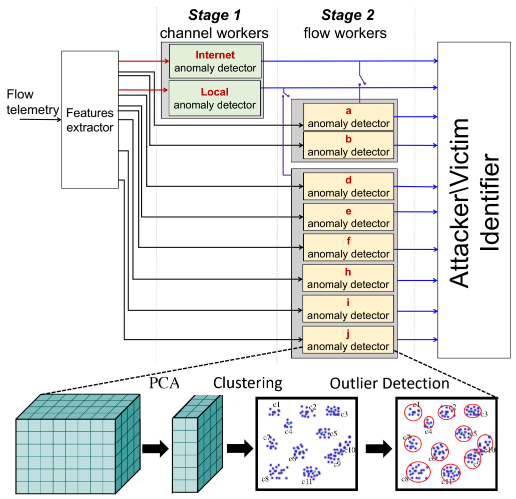


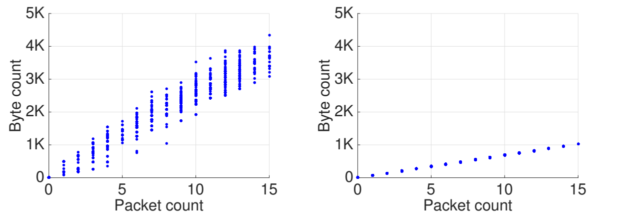
Figure 3: Structure of our anomaly detection machines for TP-Link smart plug.

TCP Port 465 downstream traffic for the same device, shown in Fig. 4(b), packet count and byte count are highly correlated (meaning a fairly consistent packet size). We collect flow counters every one minute and additionally we consider the total, mean, and standard-deviation of packet/byte count over sliding windows of 2-, 3- and 4-minute (explained in §5). This generates a total of 20 features per each flow rule at any point in time. It should be noted that we only considered flow-level attributes for features of our models to avoid the cost of packet inspection. Our primary objective in this paper is to identify the anomalous flow(s). Upon detection of attack flow(s), we may use packet-level attributes (*e.g.,* IP destination cardinalities or inter-arrival times) to identify the attacker/victim host, which is beyond the scope of this paper.

For workers of the stage-1, we use attributes of a set of flows that share the channel specified by the MUD profile (*i.e.,* local or Internet) – for example flows a.1, a.2, b.1, and b.2 for the Internet channel. Each worker of the stage-2 corresponds to a bidirectional traffic flow (*i.e.,* a couple of flow rule to/from the device). For example, machine "**a**" of the stage 2 in Fig. 3 uses features of two flows *a.1* and *a.2* from Table 1.

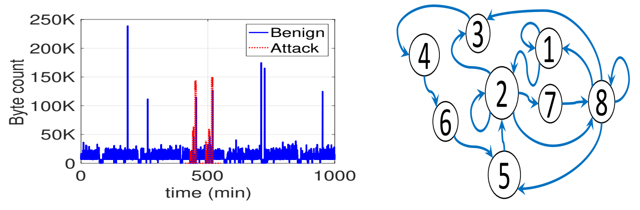
Note that we may have multiple reactive rules for an Internet flow due to dynamic DNS bindings. We therefore aggregate these rules by wild-carding the Internet endpoint. It is important to note that we don't consider default rules (*i.e., g.1*, *g.2*, and *k*) for anomaly detection, as they are taken care by the specification-based intrusion detector (explained in §3.2).

3.3.4 Anomaly Detection Workers. We build our anomaly detection workers (both in stage-1 and stage-2) based on the concept of one-class classification [18] – device workers are trained by features of benign traffic from their respective IoT device, and are able to detect whether a traffic observation belongs to the trained class or not. We employ a clustering-based outlier detection algorithm comprising three steps, as shown in Fig. 3. This is because a simple thresholding would not be able to distinguish volumetric attacks from benign traffic. To better illustrate the case, we show in Fig. 5(a) profiles of benign traffic (solid blue lines) from an Internet server TCP 443 to Netatmo camera and TCP SYN reflection attack (dashed red lines) to the same device. It is seen that no threshold value would detect the attack. This means that even having static rate-limits in the MUD profile may not be sufficient in detecting all attacks, instead it is needed to model and learn the dynamics of traffic profile for various flows. For our one-class classification, we tried three main techniques [18] including probabilistic (*i.e.,* Gaussian mixture models), domain based (*i.e.,* one-class SVM), and cluster-based (*i.e.,* DBSCAN, Kmeans), and found that clustering approach performed the best in modeling the benign behavior of our IoT types.



(a) UDP port 53 downstream. (b) TCP port 465 downstream.

Figure 4: Byte count vs. packet count of downstream remote traffic to Samsung smart-cam.



(a) (b)

Figure 5: (a) Traffic profile from an Internet server TCP port 443 to Netatmo camera. (b) Markov Chain of states transition for the worker "a" of TP-Link plug.

Principle Component Analysis (PCA): We note that if a device contains 17 flow rules, then it would have 238 features in total (each flow contributes to 14 features). This makes it computationally expensive to analyze this large number of features. However, there are features which are highly correlated (*e.g.,* Fig. 4(b)) and can be transformed, reducing the space dimension. We, therefore, employ PCA [29] to extract the principal components of our features that are orthogonal to each other. We use Kaiser rule [15] (eigenvalues >1) to deduce and select the most suitable set of principle components that capture as much variation across benign instances in our dataset. As per PCA requirement, we normalize all features using z-scores method.

Table 2: Anomaly detection machine for TP-Link plug.

Worker	Local	Internet	a	b	d	e	f	h	i	j
# Features	261	81	41	41	41	41	41	41	41	21
# PCA	18	9	4	5	5	2	4	4	4	2
Coverage (%)	97.14	94.9	93.51	96.27	96.46	99.99	98.69	96.62	99.99	99.99
# Clusters	53	48	**8**	50	4	4	36	14	2	4

Clustering: As discussed earlier, we employ several anomaly detection workers for each device, an efficient and inexpensive clustering algorithm is needed that (a) can set the parameters automatically (*i.e.,* self-tuned), and (b) is able to deal with our benign dataset that contains mixture of sparse and dense regions. Among many possible clustering algorithms, we use X-means [12] (*i.e.,* a flavor of K-means algorithm) that is a fairly lightweight yet efficient clustering method. Accuracy of high dimensional data clustering depends on the distance function used [10]. Conducting several experiments, Manhattan distance function provided us with the most optimal result. Lastly, we train the clustering algorithm by the principle components of our training dataset (obtained from PCA). As a result, the output of X-means algorithm are the coordinates of the cluster heads, as shown at the bottom of Fig. 3 by green dots labeled by c_i. Table 2 summarizes the count of features identified for each worker of TP-Link plug. It is seen that how PCA reduces the feature dimension significantly while a high level of variations in the training dataset is covered. The last column shows the number of benign clusters created for each worker.

Outlier Detection: We employ two outliers detection techniques (*i.e.,* boundary detection and Markov Chain[19]) to determine whether an instance is anomalous.

Boundary detection: An anomaly is raised when an observation deviates from benign clusters. Given cluster heads and training dataset, we compute the 97.5th percentile as a boundary for each cluster. Therefore, anomalies observed outside these boundaries are alarmed – this may lead to a minor detection of benign traffic as anomaly (false positive alarm).

Markov Chain: This technique flags anomalous instances that belong to one of expected clusters but their sequence of transition from the previous cluster (*i.e.,* state) is not normal. For this technique, we develop a Markov Chain for each worker capturing states transition across normal clusters. Fig 5(b) illustrates an example of Markov Chain for one worker of TP-Link smart plug – these 8 states correspond to clusters of worker "**a**" in Table 2. Any transition outside of this chain will be raised as anomaly.

4 ATTACK TOOL & DATA COLLECTION

In this section we explain our attack scenarios, tool, testbed and dataset (benign and attack traffic) collected in our lab.

Attack Types and Scenarios: We design two types of attacks namely, (a) direct and (b) reflection. Our direct attacks include ARP spoofing, TCP SYN flooding, Fraggle (UDP flooding), and Ping of Death. Reflective attacks include SNMP, SSDP, TCP SYN, and Smurf. IoT devices (e.g. WeMo switch and WeMo motion) have limited processing capability and become non-functional when they receive a relatively high rate traffic – the actual value of a "high" rate traffic varies across devices from WeMo motion to Amazon Echo. Also, for reflective attacks, it is important to keep the traffic rate low, ensuring the device remains functional during attack and reflects the attack traffic to the victim – for example, WeMo switch becomes non-functional under high rate attack traffic, and thus makes the intended attack unsuccessful. Due to these reasons and also to show the ability of our detection method, we use low-rate and high-rate attacks in our experiments. As depicted in Table 3, we launched various types of attacks at different rates, *i.e.,* low: 1 packet-per-second (pps), medium: 10 pps, and high: 100 pps, and with diversity of location for both attackers and victims being either from Internet (*i.e.,* I) or local (*i.e.,* L). All of our attacks sustained for 10 minutes. In total, we have launched 200 attacks, each lasts for 10 minutes.

We designed these specific attacks to analyze how different rates of attack would impact the traffic in various protocols including ARP, TCP, UDP, and ICMP – application layer attacks (*e.g.,* HTTP, HTTPS, DNS, and SMTP) will also impact these protocols. Our intention was to launch attacks (to the device or Internet servers) without being detected by the specification-based intrusion detector, meaning conforming to MUD profile. Furthermore 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 [35]. For local attacks we employed IP and port spoofing, and for Internet attacks we employed DNS spoofing followed by IP and port spoofing.

Tool: Our modular tool, written in Python, comprises a suite of attacks specific to several real consumer IoT devices. Our tool automatically identifies vulnerabilities of a device (*i.e.,* SSDP, SNMP, exposed ports, weak encryption or unencrypted communication) via launching various tests against the device on the local network. Once vulnerabilities identified, our tool launches pertinent attacks. During attacks, the tool generates appropriate annotations including the victim IP, the attacker host informations, start-time, end-time, bitrate, attack protocol, and attack port number.

Testbed: The lower section of Fig. 6 shows our testbed comprising a TPLink gateway with OpenWrt firmware that serves a number of IoT devices including one unit of WeMo switch, WeMo motion sensor, Samsung smart-camera, TP-Link smart plug, Netatmo camera, Chromecast Ultra, Amazon Echo, LiFX bulb, Phillips Hue bulb and iHome Smart plug. We also consider two attackers in our testbed; locally (inside LAN) and remotely (on the Internet) with two victims – both attackers are able to attack both victims.

We connected a 1 TB external hard disk to the gateway to store packet trace (*i.e.,* `pcap` files) of all network traffic (*i.e.,*

Table 3: Attacks launched against our IoT devices. (*L:local, d:device, I:Internet*)

		Maximum packet rate			Device label										Attack scenario				
Attacks		1 pps	10 pps	100 pps	WM	WS	SC	TP	NC	CU	AE	PH	IH	LX	L→d	L→d→L	L→d→I	I→d→I	I→d
Reflection	SNMP	✓	✓	✓			✓									✓	✓	✓	
	SSDP	✓	✓	✓	✓					✓		✓				✓	✓	✓	
	TCP SYN	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓				✓		✓	
	Smurf	✓	✓	✓			✓	✓				✓		✓		✓			
Direct	TCP SYN	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓				✓
	Fraggle	✓	✓	✓			✓				✓			✓					✓
	Fraggle	✓	✓	✓	✓		✓				✓			✓	✓				
	Ping of Death	✓	✓	✓	✓	✓	✓	✓				✓		✓	✓				
	ARP Spoof	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				

Table 4: Size of our dataset.

Device	# train inst.(min)	# test inst.(min)	# attack inst(min)	Device label
WeMo motion	15000	55400	300	WM
WeMo switch	15000	55361	180	WS
Samsung smartcam	15000	55364	357	SC
TP-Link smart plug	15000	55372	178	TP
Netatmo camera	15000	55359	147	NC
Chromecast Ultra	15000	28730	252	CU
Amazon Echo	15000	28730	79	AE
Phillips Hue bulb	15000	28730	297	PH
iHome Smart plug	15000	28730	30	IH
LiFX bulb	15000	28730	150	LX

locally and remotely) using `tcpdump` tool. We had collected packet traces of benign and attack traffic from our testbed for a period of 16 days – given known attackers/victims we annotated the attack traffic in our dataset. Table 4, lists our dataset. Interestingly we found that there were other attacks launched from the Internet (*i.e.,* wild attacks explained in §5.6) as we enabled port forwarding. In order to capture benign behaviour of IoT devices in our testbed, we installed a touch replay tool on a Samsung galaxy tab recording all possible user interactions (*e.g.,* turning on/off lightbulb, or streaming video from camera) with individual IoTs – each device has a limited number of functions available. We then replayed recorded interactions (spread randomly over hour and day) emulating a real personal activity. For Amazon Echo specifically, we used a simple text-to-speech program that randomly picks a statement from a pre-configured list (*e.g.,* "Alexa! How is the weather today?", "Alexa! Sing X", etc).

Dataset: We collected two datasets namely raw packet traces and derived flow counters. We release our datasets (spanning one month period of benign and attack traffic relating to ten IoT devices and annotation of those attacks) via [31]. The released dataset contains 30 pcap files and each file corresponds to a trace collected over a day. Note that there are 17 other IoT devices (*e.g.,* TPLink camera, DLink camera) in our testbed that are not studied or experimented in this work but the benign data of these devices are included in our traces – we only focused on selected devices with more complex behavior. There are two annotation files comprising: (a) start time, end time, flows that are influenced during the attack, attack type, bitrate of attack; and (b) pcap file number, attacker, and victim IP address; that will be released along with the dataset. Our derived dataset contains counters of flows (computed over a minute) for 10 IoT devices listed in Table 4. The second column shows the number of training instances (*i.e.,* minutely count of packets and bytes per flow rule) for each device. Our training instances only contain benign traffic. For testing phase, we collected more than 28,730 instances for each device containing both benign and attack traffic. Out of these many testing instances, the number of attack instances is shown in the fourth column.

5 PROTOTYPE AND EVALUATION

We prototyped our scheme in a small testbed, depicted in Fig. 6. The objectives of this experimental setup are to demonstrate the feasibility of our scheme with real equipment and traffic, and to evaluate the efficacy of anomaly detection and identification.

5.1 Prototype Implementation

For our system we have developed an application on top of the Ryu along with the Faucet/Gauge [27] SDN controllers, the MUD policy engine, the MUD collector, and implemented the NATS messaging system and used the InfluxDB and H2 databases. Each of these components operates on a separate docker container over an Ubuntu 16.04 server. In addition, the MUD file server is a repository of MUD profiles (obtained from [7]) that runs as an HTTP server on a separate VM in our University cloud [4]. We release our prototype as open source [32].

SDN controllers and application: We used open-source SDN controllers Faucet, Gauge and Ryu in our prototype: Faucet is used for inserting proactive flows via its configuration file; Ryu is used for inserting reactive flows that are dynamic with idle-timeout (60 minutes); Gauge is used for collecting flows counters. The configuration file of the Faucet consists of a set of access control rule that are generated/augmented by the MUD policy engine (explained next). Ryu exposes a REST API for insertion of reactive flows that is convenient to use. But this allows any application to manipulate flow tables that may cause conflict – both Faucet and Ryu controllers manipulate the flow-tables. Therefore, we developed a Python application in Ryu that subscribes to a specific channel of NATS broker (listening for reactive flow messages), enforces them into table-0 with high priority (*i.e.,* 15000 in our prototype) inside the switch. We developed an application that listens to Faucet Unix data-stream, retrieves newly discovered device details (including the device MAC, the switch dpid, VLAN id), and publishes them into the NATS – this channel is listened by the MUD engine. We also configured Gauge to collect flow counters every minute from the switch and thereafter writes into the InfluxDB.

SDN switch: We installed OpenWrt firmware (v18.06) and OVS (v2.8.2) on a TP-Link Archer C7 gateway for our SDN switch and default gateway. Our switch is configured with three OpenFlow controllers, as explained above. We created a VX-LAN tunnel interface between the switch and MUD policy engine to mirror selected packets that requires further inspection.

MUD policy engine: We built the MUD policy engine in Java. It inspects incoming packets from the VX-LAN tunnel

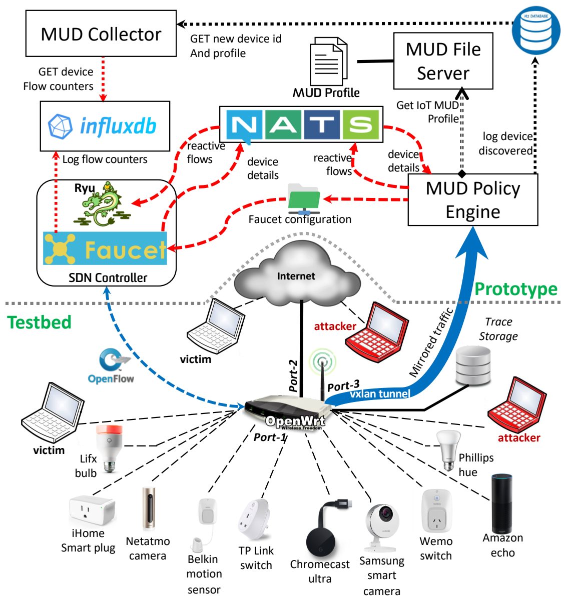


Figure 6: Our system prototype and testbed.

interface. We developed our asynchronous and non-blocking packet listener using `netty` [36]. The engine is responsible for various tasks including generating/augmenting Faucet configuration file (based on device MUD profile) upon discovery of a new device, publishing reactive flow rule based on run-time DNS bindings, and detecting non-compliant traffic (*i.e.,* specification-based intrusion detector).

We note that the Faucet configuration directory is shared by both the Faucet controller and the MUD engine. We designed 3 hierarchical levels of configurations file: (a) switch configuration (*i.e.,* `faucet.yaml`) – this contains a default access control rule for each port of the switch, mirroring all DHCP packets to the MUD policy engine. This configuration file is generated manually by the network admin who has prior knowledge of switch, ports, and VLANs; (b) VLAN configuration (*e.g.,* `switch-mac.yaml`) that lists all devices against each VLAN – in our testbed all devices belong to one VLAN for simplicity; (c) device configuration (*e.g.,* `device-mac.yaml`) that lists ACEs for a given device, as described in §3.3. Once Faucet configuration file is updated, the MUD policy engine writes the device MAC and its MUD profile into an H2 [30] database, informing the MUD collector about the newly connected device, thus enabling it to compute required features for the new device.

MUD Collector: MUD collector periodically (every minute) polls the H2 to check whether a new device is discovered, if so then the device MUD profile will be fetched and interpreted to identify the required features of that specific device. Thereafter, the collector will poll (every minute) the InfluxDB for the device flows counters, compute the features, and writes feature into a device-specific CSV file. Note that we replay these features into our anomaly detectors.

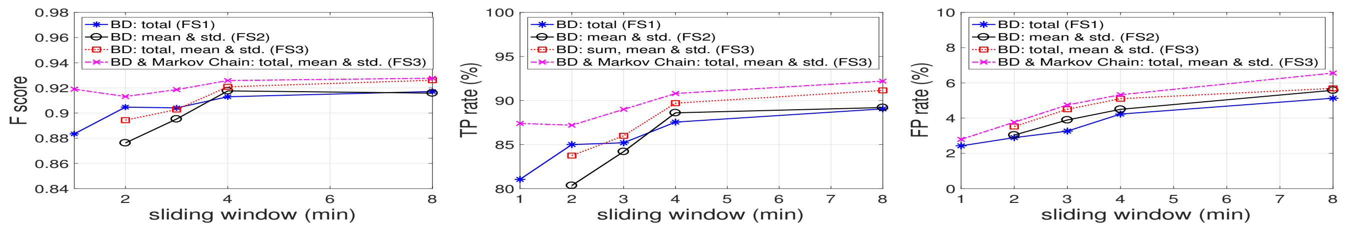
Anomaly detector: We developed our attack detection application using R [39] and Weka [13] tools. PCA library is available in R, and therefore we implemented our boundary detection in R. We employed X-means package of Weka for clustering and used RWeka extension of R to interface with Weka. We replayed (streamed) the CSV outputs of the MUD collector into our anomaly detection application. We then correlated our attack annotation log with the result of the anomaly detection to compute the performance of our scheme.

5.2 Feature Analysis

We now evaluate the importance of features in performance of our anomaly detection. As explained in §3.3.3, we collect flow statistics every minute and construct attributes using these statistics in three possible scenarios: (a) **feature-set-1 (FS1)**: only total count over sliding windows,(*e.g.,* for window size of 2 min., features are 1-min and 2-min total count of flow bytes and packets), (b) **feature-set-2 (FS2)**: total count for the last one minute, and mean and standard deviation over the window (*e.g.,* for a 2 min. window size, features are 1-min total count, 2-min mean and 2-min standard deviation of byte/packet counts), and (c) **feature-set-3 (FS3)**: a combination of FS1 and FS2. Note that for a 1 min. sliding window all FS1, FS2, and FS3 correspond to the same set of features. In Fig. 7, we plot F-score (*i.e.,* a measure of accuracy in binary classification), True Positive rate (TPR), False Positive rate (FPR) for each of these feature sets when sliding window varies from 1 to 8 minutes. This figure also illustrates the performance of two anomaly detection techniques: only boundary detection (BD) and combination of BD and Markov Chain.

Impact of window size: According to Fig. 7(a), for BD only, as the window size increases the performance is improved steadily, with FS3 outperforming FS1 and FS2. Note FS1 performs better in smaller windows sizes, since mean/ standard deviation would not give extra information for small number of data points. Looking into Figures 7(b) and 7(c), we observe that a larger window size results in high rate of true positive and false positive. In order to detect low rate attacks, we need to choose a larger window size. It is also important to note that a larger window imposes computation cost and demand a higher memory to compute features.

Impact on detection: Use of just Markov Chain (without boundary) results in 54.55% of TPR and 0.69% of FPR with 1 minute window for FS1 – a significant portion of attacks are missed. Applying just boundary, instead, gives a high TPR 81.05%. Combining both boundary detection and Markov Chain gives 87.40% of TPR (with 1 min window) – 5 attacks (from a total of 200) that are low rate get missed. The TPR can be further improved to 90.80% by incorporating a richer feature set FS3 and increasing the sliding window to 4 min



(a) F-Score (b) True Positive rate (c) False Positive rate

Figure 7: Impact of various features on performance of anomaly detection. (BD: boundary detection)

(all low-rate attacks get detected) – this is gained at the cost of higher FPR in Fig. 7(c). We note that this TPR can be achieved by use of just boundary detection with FS3 and window size of 4 min (as shown by red lines with square markers in Fig. 7(b)). This shows that traffic characteristics captured by the Markov Chain can be captured by the boundary detection but over a larger window.

Summary: If low-rate attacks are not of interest to the network operator then a small window size with FS1 using both boundary detection and Markov Chain is recommended (lower cost and better FPR). If the operator wants to detect all possible attacks (both low-rate and high-rate), a large window size with FS3 (using only boundary detection) would be an efficient approach – note that use of a larger window comes with the cost of maintaining states for computing features and results in a slightly higher false positive rate. In next, we will employ the latter approach to detect all attacks and operate over the sliding window of 4 minutes – going beyond 4 minutes does not significantly change both TP and FP rates, but it requires double the amount of states. Note that this does not affect the responsiveness of our detection method – it still responds every one minute.

5.3 Attack Detection

We start with performance evaluation of our attack detector when considering IoT devices all together as well as individually. The result is illustrated in Table 5. Each row shows the performance results for various combinations of anomaly detectors shown in Fig. 3.

Accuracy and False-Positive Rate: Focusing on aggregate of all devices, it is seen that the combination of stage-1 and stage-2 workers (*i.e.,* full structure) gives the highest accuracy of 94.9% (*i.e.,* the percentage of correctly classified benign and attack instances). We are able to detect 89.7% of all attacks (TPR: true positive rate) across all IoT devices, when two-stage anomaly detection is employed. As we expect, in this situation the lowest false positive rate (*i.e.,* FPR 5.1%) is achieved. Even this rate of false positive may not be very attractive for real network settings with a large number of connected devices. To reduce FPR a **time-based filtering** can be employed. We use a simple threshold for raising alarm if the anomaly detection is triggered continuously for more than "*t*" minutes. As shown in the second row of Table 5, having a 2-minute filter reduces the FPR to 2.4%. However, the TPR is also reduced to 72.3% – because attacks were not detected for their first two minutes due to time-based alarm filtering. Increasing this time threshold would enhance the FPR but it is detrimental to detection responsiveness.

Unsurprisingly, when workers of only stage-1 or stage-2 are used, the overall accuracy drops. We note that stage-2 workers perform slightly better than stage-1 workers – each stage has its own specialty. Stage-1 deals with coarse-grained device-level activity whereas stage-2 deals with fine-grained flow-level activity. Thus, combination of these stages provides better accuracy and false-positive.

We now consider per-device performance of anomaly detection. The bottom two rows in Table 5 show the performance, when local and Internet attacks are separately considered. For the local detector, the lowest true positive rate (*i.e.,* TPR 80.7%) is achieved for Chromecast Ultra (*i.e.,* device label "CU"). We found that some of our reflection attacks originated from local attacker to external victim (*i.e.,* L→d→I) are missed by this worker, meaning that local traffic features are not impacted sufficiently to raise an attack alarm. However, these reflection attack instances are detected by the Internet worker. Similarly, we observe that the Internet detector for Amazon Echo (*i.e.,* device label "AE") suffers from a fairly low TPR of 68.4% – only a few instances for 1pps and 10pps TCP SYN attack from Internet were missed, but ultimately both the attacks were detected over time. Overall, our machine successfully detected all types (high-, med-, low-rate) of attacks.

Another interesting observation is that for the stage-1 detector of Samsung camera(*i.e.,* device "SC"), the false positive is very high (30.5%) however when it is combined with stage-2 then the false positive drops to 3.1%. This shows that the coarse-grained behavior (*i.e.,* aggregate of flows) of this device was not fully learned by our training dataset but flow-level behavior was well captured and learned.

Detecting Various Attack Types: In Table 6 we show the number of detected attack instances for each IoT device per attack type – each instance is one minute worth of traffic. For example in the first row, we launched 30 instances of TCP SYN remote reflection attacks (*i.e.,* L→d→L) to device

Table 5: Performance of our anomaly detectors.

Anomaly Detectors	All devices			WM		TP		SC		NC		CU		AE	
	Accuracy (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)
Stage-1 and Stage-2 combined	**94.9**	**89.7**	**5.1**	95.7	5	95.7	2.3	93.8	3.1	80.3	4.2	79.8	19.7	83.5	3.8
Stage-1 & -2 combined (2-min filtering)	97.5	**72.3**	**2.4**	77.7	2.4	76	1	75.6	0.9	71.4	2.1	63.1	**13.2**	67.1	**0.6**
Only Stage-2 detector	89.1	92	10.9	96	11.3	96.2	3.4	94.1	6.5	83.7	6.1	86.1	52.3	91.1	16.3
Only Stage-1 detector	85.7	93.7	14.4	97	14.8	96.2	3.9	98.3	**30.5**	88.4	8.7	88.9	27.7	93.7	23.5
Only Local detector	90.6	91.5	9.4	87.1	8.6	96	2.2	96.2	28.4	94.3	1.5	**80.7**	7.1	93.3	15.9
Only Internet detector	93.9	88.5	6.1	95	7.2	93.2	2.1	94	3.3	75	8	84.8	23.2	**68.4**	9.2

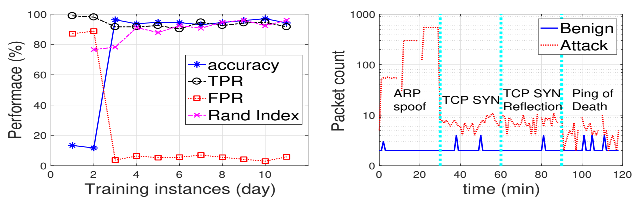
Table 6: Detected attacks for individual devices. (*L:local, d:device, I:Internet*)

Attack Type	Attack Scenario	All device			WM			TP			SC			NC			CU		
		Launched	Detected	Identified	Launched	Detected	Identified	Launched	Detected	Identified	Launched	Detected	Identified	Launched	Detected	Identified	Launched	Detected	Identified
TCP SYN reflection	L→d→L	208	198	178	30	30	**21**	29	28	28	30	29	20	30	29	28	30	29	28
TCP SYN reflection	I→d→I	221	186	177	30	28	28	30	29	28	30	28	27	30	**20**	17	41	24	21
SSDP reflection	L→d→L	90	63	18	30	26	18										30	**12**	0
SSDP reflection	I→d→I	91	89	85	30	29	29										31	31	27
SSDP reflection	L→d→I	89	85	76	30	29	29										30	28	19
SNMP reflection	L→d→L	27	22	0							27	22	0						
SNMP reflection	I→d→I	30	28	28							30	28	28						
SNMP reflection	L→d→I	30	29	29							30	29	29						
Smurf	L→d→L	120	102	97				30	27	27	30	28	27						
Fraggle	L→d	120	110	109	30	29	29				30	28	28						
Fraggle	I→d	108	85	79							30	29	29						
TCP SYN	L→d	210	203	184	30	30	21	30	30	30	30	29	20	30	28	28	30	30	29
TCP SYN	I→d	179	145	139	30	29	28	29	27	26	30	28	27	30	**16**	14	30	**18**	16
Arp Spoof	L→d	297	282	276	30	29	29	30	28	28	30	28	28	27	25	24	30	29	28
Ping of death	L→d	180	167	158	30	28	28	30	30	30	30	29	29						

label "A" that 30 of them are detected. Our anomaly detection machine was able to detect all of these 30 attacks just one minute after their commencement. Results shown in Table 6 highlight the fact that our method is able to detect volumetric attacks of all types during their lifetime (*i.e.,* 10 minutes or more).

We note that our scheme may miss certain types of reflection attacks within the local network (*i.e.,* L→d→L) those that are broadcast with the source address spoofed as a local victim. For this specific type, the original attack traffic does not match on any device-specific flow rule (*e.g.,* SNMP reflection attack for Samsung camera and SSDP reflection for Chromecast Ultra) since we only capture incoming traffic for the local network (as explained in §3). But, the reflected traffic may (may not) hit one of the device flows. For example, we found that the local SSDP reflection attack on WeMo motion was detected. This is because for this attack the reflected packets happened to match on one flow of the WeMo motion. But if the victim is an IoT device, even the local broadcast reflection attack will be detected by the victims machines.

We also see a low detection, shown by red text in Table 6, for SSDP reflection (L→d→L) and TCP SYN (I→d) in Chromecast Ultra(*i.e.,* device "CU") , and TCP SYN reflection (I→d→I) and TCP SYN (L→d) in Netatmo camera (*i.e.,* device "NC"). We emphasize that these undetected attack instances in fact correspond to only the beginning (*i.e.,* first few minutes) of low-rate attacks from their 10-minute total duration. This means that for low-rate attacks, our scheme



(a) perofrmance vs. training. (b) ARP profile.

Figure 8: (a) Performance of anomaly detection for Samsung camera, (b) ARP traffic profile for TP-Link smart plug (benign versus local attack).

is able to flag them if they last for long duration (*e.g.,* more than four minutes) which is typically the intention of the attacker. In other words, all of our attacks were ultimately detected (*i.e.,* on average after 1.92 minutes from the attack commencement).

Impact of Training on Performance: The accuracy of our attack detection highly depends on the benign states that are learned during the training phase. We see in Fig. 8(a) that the overall accuracy for Samsung camera is less than 50% when the machine is trained by only 2-day worth of data, and it steeply rises to 96.08% when machines are introduced by instances from one additional day that contain new benign (expected) states. Instead, the TPR rate is consistently high (*i.e.,* above 80%) as all attack instances (including low rate ones) deviate from limited number of trained states. Therefore it is essential to capture all benign states (*i.e.,* normal) of each device during the training phase.

We use "Rand index" [16] (a measure of the similarity between two data clusters) to identify the minimum amount of training dataset (in terms of number of days) for building a well-trained model. It is shown by dashed pink lines with cross markers in Fig. 8(a). A consistently high Rand index indicates that the training data is sufficient. We can see that 4 days of training instances would result 91% of Rand index and will relatively persist with more instances trained. This metric can be used to determine if a model generated from an environment is deployable for other environments – such exercise is beyond the scope of this paper.

5.4 Attack Flow Identification

We now look at the performance of our scheme in identifying attack flow. In Table 6, the "Identified" column under each device shows the number attack instances in which the contributing flow was correctly identified. It can be seen that for TCP SYN local reflections (L⟶d⟶L) in the first row, there were 30 attack instances launched on WeMo motion of which all 30 instances were detected but only in 21 of those the attack flow was identified correctly. In the remaining 9 instances the ARP flow was only flagged – however our attack was not launched over ARP. We found that the ARP anomaly worker is sensitive to (*i.e.,* raises alarms for) most of local attacks, highlighted by bold blue text under the WeMo motion in Table 6, while the actual contributing flow was not identified for some instances. In Fraggle and Ping-of-death, instead, the attack flow was flagged by the respective worker along with the ARP worker. We note that it is essential to identify the attack flow for blocking purpose – attack mitigation is out of the scope of this work.

Correctness of Flow Alarms: We now look into the performance of individual flow workers (*i.e.,* in our stage-2 anomaly detectors). Table 7 lists detected attacks and corresponding flows identified for the TP-Link smart plug. The "Malicious Flow" column shows the flow (from Table 1) that we used in our attack. For TCP SYN reflection (L⟶d⟶L), we used TCP port 9999 (flow *i*). It is seen that 28 out of 29 attack instances were correctly detected and all true alarms flagged the correct flow *i*, though ARP flow (*i.e.,* flow *h*) is also flagged in 28 alarms. Such high rate of incorrect ARP alarms is seen for local TCP SYN, Smurf, Ping-of-Death, highlighted in red text. In order to better understand the reason for ARP alarms, we plot in Fig. 8(b) the profile of ARP flow in benign (shown by solid blue line) versus local attack (shown by dashed red line) traffic from our training dataset. It is clearly seen that ARP profile is deviated from its normal pattern even for attacks that are not directly related to this flow. We note that during Internet attacks the device ARP profile does not get impacted significantly to raise alarms.

Another interesting observation is when the ARP spoofing attack is launched (the last row in Table 7). We see 15 and 12 alarms respectively for DNS (flow *f*) and local ICMP (flow *j*). We note that the ARP spoof causes all victim traffic to be redirected to the attacker (instead of expected gateway). Since the TP-Link smart plug was communicating ICMP and DNS during the ARP spoof attack, as a result the anomaly was detected by respective flow workers.

These observations can help us determine a weight for individual workers when identifying attack flow(s). For example, if ARP and local TCP port 80 workers flag an anomaly simultaneously, then we may want to start investigating bidirectional TCP flows to/from port 80 – deprioritizing alarms form the ARP worker.

5.5 Processing Cost

We now quantify the system performance in terms of processing cost of our packet inspection (done by the MUD policy engine) to extract DNS bindings, time taken to train anomaly detection, storage size of training dataset, size of trained model, memory demand of features computed at runtime and time taken for a prediction. The results are listed in Table 8.

Packets inspection is only used for tightening up flow entries inserted into the switch. It is also used for further investigation when an anomalous traffic is flagged – attack analysis is beyond the scope of this paper. We record the total fraction of inspected packets measured over a 16-day period in the second column of Table 8. It is seen that a small fraction of packets (*i.e.,* less than 1.16%) is mirrored to the inspection engine (averaged across all devices) – this promises the scalability of our SDN-based approach.

We note that IoT devices (as opposed to typical computers or phones) generate limited number of flows and use them frequently (*i.e.,* MUD profile). As explained in §3, some of these frequent flows are captured (right after their occurrence) by reactive flow rules inserted by our system, and therefore subsequent packets don't need to be inspected, resulting a minimal load on engine. It is seen that the fraction inspected packets is the highest (*i.e.,* . 1.61%) in Table 8 for the WeMo motion. Interestingly, we found that 99.5% of inspected packets were DNS responses, with an average rate of 3.13 packets per minute.

In terms of training time, it took less than 7 minutes for a given device (the longest time observed for Samsung camera) to train the entire model set in sequence – training a single flow-level model took less than 50 seconds. We used Intel Core CPU 3.1 GHz laptop with 16GB of RAM running Mac OSX. The size of the model for each device is 14.96 MB on average. This can be significantly optimized if we use other tools such as Scikit-learn[38], as R and Weka retain the training data in the model itself. In the second last column of Table 8, we record the memory requirement for computing features of FS3 over a window of 4 minutes – this footprint varies with different features sets and window sizes. Lastly,

Table 7: Identified malicious flow for TP-Link smart plug under attack.

Attack type	Attack scenario	Launched	Detected	Malicious flow	Identified flows							
					a	b	d	e	f	h	i	j
TCP SYN reflection	L→d→L	29	28	i	0	0	0	0	0	28	28	0
TCP SYN reflection	I→d→I	30	29	b	0	28	0	0	3	3	0	3
Smurf	L→d→L	30	27	j	0	0	0	0	0	27	0	27
TCP SYN	L→d	30	30	i	0	0	0	0	0	30	30	0
TCP SYN	I→d	29	27	b	0	26	0	0	0	0	0	0
Ping of death	L→d	30	30	e	0	0	0	30	0	26	0	0
ARP spoof	L→d	30	28	h	0	0	0	0	15	28	0	12

Table 8: Performance metrics.

IoT device	inspected packets (%)	time taken for training (s)	training instances (MB)	model size (MB)	runtime features size (KB)	model response time (ms)
WeMo motion	**1.61**	270	3.04	18.93	4.85	15.12
Samsung smartcam	0.07	419	4.14	29.75	5.68	26.68
All devices	0.92	221.9	2.26	14.96	3.56	13.04

Table 9: Comparison of attacks detection between our solution and Snort

IoT device	Detected wild attackers (by our solution)	Detected attacks (by Snort)
WM	{107.170.227.13}	{107.170.227.13}, SSDP reflection(I→d→I)
WS	{107.170.228.161}	{107.170.228.161}
SC	{103.29.71.94, 45.55.2.34, 107.170.229.67, 45.55.14.102, 181.214.206.55, 216.98.153.254, 54.215.173.102, 14.134.5.4, 205.209.159.120}	{103.29.71.94, 45.55.2.34, 107.170.229.67, 45.55.14.102}, SNMP reflection(I→d→I)
TP	{107.170.226.164, 185.170.42.66, 46.182.25.42, 45.227.254.243, 185.156.177.13, 17.136.0.172, 125.212.217.214, 107.170.225.175, 217.182.197.186}	{107.170.226.164, 185.170.42.66, 46.182.25.42, 45.227.254.243}
NC	{58.182.245.89, 27.75.133.76, 14.234.90.16, 103.4.117.85, 177.74.184.229, 176.36.241.230, 81.17.18.221, 201.174.9.186, 194.208.107.25, 161.97.195.135, 189.165.40.237}	{58.182.245.89, 27.75.133.76, 14.234.90.16, 103.4.117.85}
CU, PH	N/A	SSDP reflection(I→d→I)
AE, IH, LX	N/A	

Table 10: Performance of other anomaly detectors.

IoT device	TPR (%)	FPR (%)
WeMo motion	74.00	3.32
WeMo switch	60.55	1.10
Samsung smartcam	67.78	2.31
TP-Link smart plug	95.19	0.63
Netatmo camera	0.00	0.10
Chromecast Ultra	32.93	3.35
Amazon Echo	15.18	3.27
Phillips Hue bulb	19.86	3.26
iHome Smart plug	70.00	1.94
LiFX bulb	82.00	2.52

it takes on average 13 ms to get a response from the model when both stage-1 and stage-2 are triggered, highlighting the efficacy of our scheme in real-time.

5.6 Comparison with Existing Methods

Lastly, we compare the performance of our scheme with existing tools and proposals. We start with Snort [41], a widely deployed, open-source, signature based IDS, and then reevaluate our machines with the features proposed by other researchers.

Snort IDS: We configured Snort IDS with the community rule-set [3] and replayed our packet traces to Snort IDS using the `tcpreplay` tool. Table 9 lists the IP address of endpoints on the Internet that attacked our testbed during the experiments, and were detected by our specification-based intrusion detector – since these Internet endpoints were not specified by the MUD profile of respective devices. Note that these wild attacks from the Internet were seen after port forwarding was enabled on the gateway. According to AbuseIPDB[1], most of these endpoints have been reported as abusive IP addresses (*e.g.,* 181.214.206.55 has a probability of 46% as being an abusive IP address). We can see that the Snort detects a subset of these attacks – attacks from IP addresses in red text are not detected by the Snort. In addition, out of 40 types of our own attacks, the Snort detected only two namely SSDP reflection (I→d→I) to WeMo motion and SNMP reflection (I→d→I) to Samsung camera, shown by blue text in Table 9. These two types of attacks (detected by Snort) carry traffic towards the local network and their signature was known to Snort. While majority of our attacks were specifically designed for IoT devices that Snort does not have the signature for most of them.

Other machine learners: Works in [21, 24, 43, 44] also use a machine learning based approach to detect anomalies. However, the main issue of their approach is that their machines are based on binary classification and use both benign and attack traffic for the training. This limits the scalability of using such solution in an operational network. We note they also employ packet/byte counters as the feature for their machines, but at device-level only (*i.e.,* two features: aggregate bytes and packets of all flows). We reevaluated our anomaly detection algorithm with these two attributes for comparison purpose. The results are illustrated in Table 10. We can see that the overall performance (across 5 devices) is very poor compared to our scheme, with no attacks detected for the Netatmo as well as half of the attacks are missed for WeMo motion and Samsung smartcam.

6 CONCLUSIONS

Vulnerable IoT devices are increasingly putting smart environments at risk by exposing their networks unprotected to cyber attackers. MUD framework aims to reduce the attack surface on IoTs by formally defining their expected network behavior. In this paper, we have focused on detecting volumetric attacks that are not prevented by the MUD profile. We developed an SDN-based system empowered by machine learning to monitor and learn behavioral pattern of MUD rules at coarse-grained (per-device) and fine-grained (per-flow). We then subjected real IoT devices to a range of volumteric attacks (designed to conform to MUD profiles) in our lab, collected and labeled our traffic traces. Lastly, we prototyped our system, and quantified the efficacy of our scheme in detecting volumetric attacks. We made our dataset and system available to the public.

REFERENCES

[1] 2018. AbuseIPDB. (2018). https://www.abuseipdb.com Accessed: 2018-06-05.

[2] 2018. Manufacturer Usage Description. (2018). https://developer.cisco.com/site/mud/

[3] 2018. Snort. (2018). https://snort.org/

[4] 2018. UNSW MUD repository. (2018). https://iotanalytics.unsw.edu.au/mudprofiles

[5] Cisco. 2018. *Cisco 2018 Annual Cybersecurity Report*. Technical Report.

[6] daq 2018. Device Automated Qualification for IoT Devices. (2018). https://github.com/faucetsdn/daq

[7] A. Hamza et al. 2018. Clear as MUD: Generating, Validating and Applying IoT Behaviorial Profiles. In *Proc. ACM Sigcomm workshop on IoT S&P*. Budapest, Hungary.

[8] A. Hamza et al. 2018. Combining MUD Policies with SDN for IoT Intrusion Detection. In *Proc. ACM Sigcomm workshop on IoT S&P*. Budapest, Hungary.

[9] A. Sivanathan et al. 2017. Characterizing and classifying IoT traffic in smart cities and campuses. In *Proc. IEEE INFOCOM workshop on SmartCity*. Atlanta, Georgia, USA.

[10] C. C Aggarwal et al. 2001. On the surprising behavior of distance metrics in high dimensional spaces. In *Proc ICDT*. Springer, Berlin, Heidelberg.

[11] C. Liu et al. 2017. Piggybacking Network Functions on SDN Reactive Routing: A Feasibility Study. In *Proc. ACM SOSR*. Santa Clara, CA, USA.

[12] D. Pelleg et al. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters.. In *Proc ICML*. San Francisco, CA, USA.

[13] E. Frank et al. 2016. The WEKA Workbench. *Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques, 4th edn. Morgan Kaufman, Burlington* (2016).

[14] E. Lear et al. 2018. *Manufacturer Usage Description Specification (work in progress)*. Internet-Draft draft-ietf-opsawg-mud-25. IETF Secretariat.

[15] H.F Kaiser et al. 1960. The application of electronic computers to factor analysis. *Educational and psychological measurement* 20, 1 (1960), 141–151.

[16] H. Lawrence et al. 1985. Comparing partitions. *Journal of classification* 2, 1 (1985), 193–218.

[17] J. P. Amaral et al. 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.

[18] M. Pimentel et al. 2014. A review of novelty detection. *Signal Processing* 99 (2014), 215–249.

[19] P. Garcia-Teodoro et al. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security* 28, 1-2 (2009), 18–28.

[20] P. Uppuluri et al. 2001. Experiences with Specification-Based Intrusion Detection. In *Proc. RAID*. Davis, USA.

[21] R. Braga et al. 2010. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In *Proc. IEEE LCN*. Denver, CO, USA, 408–415.

[22] R. Doshi et al. 2018. Machine Learning DDoS Detection for Consumer Internet of Things Devices. In *Proc. IEEE S&P workshop on Deep Learning and Security*. San Francisco, USA.

[23] R Sommer et al. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proc. IEEE Security and Privacy (SP)*. Berkeley, CA, USA.

[24] S. Bhunia et al. 2017. Dynamic attack detection and mitigation in IoT using SDN. In *Telecommunication Networks and Applications Conference (ITNAC), 2017 27th International*. IEEE, 1–6.

[25] S. Boddy et al. 2017. *The Hunt for IoT: The Rise of Thingbots*. Technical Report. F5 Labs.

[26] F. Loi et al. 2017. Systematically Evaluating Security and Privacy for Consumer IoT Devices. In *Proc. ACM CCS workshop on IoT S&P*. Dallas, Texas, USA.

[27] faucet 2018. Faucet. (2018). https://faucet.nz/

[28] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. 2009. Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges. *Comput. Secur.* 28, 1-2 (Feb. 2009), 18–28.

[29] H. Abdi et al. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459.

[30] h2 2018. H2 Database. (2018). http://www.h2database.com

[31] A. Hamza. 2018. Attack Data. (2018). https://iotanalytics.unsw.edu.au/attack-data

[32] A. Hamza. 2018. MUD-ie. (2018). https://github.com/ayyoob/mud-ie

[33] Cisco Systems Inc. 2017. *Midyear Cybersecurity Report*. Technical Report.

[34] K. Giotis et al. 2014. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Computer Networks* 62 (2014), 122–136.

[35] M. Lyu et al. 2017. Quantifying the Reflective DDoS Attack Capability of Household IoT Devices. In *Proc. ACM WiSec*. Boston, Massachusetts.

[36] netty 2018. Netty. (2018). https://netty.io/

[37] Vern Paxson. 1999. Bro: A System for Detecting Network Intruders in Real-time. *Comput. Netw.* 31, 23-24 (Dec. 1999), 2435–2463.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[39] R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

[40] Martin Roesch. 1999. Snort - Lightweight Intrusion Detection for Networks. In *Proc. USENIX Conference on System Administration*. Seattle, Washington.

[41] M. Roesch. 1999. Snort - Lightweight Intrusion Detection for Networks. In *Proc USENIX Conference on System Administration*. Seattle, Washington.

[42] S. A. Mehdi et al. 2011. Revisiting traffic anomaly detection using software defined networking. In *Proc. Springer RAID*. Berlin, Heidelberg.

[43] T. Tang et al. 2016. Deep Learning Approach for Network Intrusion Detection in Software Defined Networking. In *Proc. IEEE WINCOM*. Fez, Morocco.

[44] Y. Cui et al. 2016. SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks. *Journal of Network and Computer Applications* 68 (2016), 65–79.

[45] Changhoon Yoon, Taejune Park, Seungsoo Lee, Heedo Kang, Seungwon Shin, and Zonghua Zhang. 2015. Enabling Security Functions with SDN. *Comput. Netw.* 85, C (July 2015), 19–35.