Clearer than Mud: Extending Manufacturer Usage Description (MUD) for Securing IoT Systems

Simran Singh(✉), Ashlesha Atrey, Mihail L. Sichitiu, and Yannis Viniotis

North Carolina State University, Raleigh, NC 27606, USA
{ssingh28,amatrey,mlsichit,candice}@ncsu.edu

Abstract. Internet of Things (IoT) devices, expected to increase exponentially over the next several years, are easy targets for attackers. To make these devices more secure, the IETF's draft of Manufacturer Usage Description (MUD) provides a means for the manufacturer of an IoT device to specify its intended purpose and communication patterns in terms of access control lists (ACLs), thereby defining the device's normal behaviour. However, MUD may not be sufficient to comprehensively capture the normal behaviour specification, as it cannot incorporate variable operational settings that depend on the environment. Further, MUD only supports limited features. Our approach overcomes these limitations by allowing the administrator to define the normal behaviour by choosing combinations from a wider set of features that includes physical layer parameters, values of packet headers, and flow statistics. We developed and implemented a learning-based system that captures and demodulates wireless packets from IoT devices over a period of time, extracts the features specified in the normal behaviour specification, and uses a learning algorithm to create a normal model of each device. Our implementation also enforces these normal models by detecting violations and taking appropriate actions, in terms of ACLs on an Internet Gateway, against the misbehaving devices. Hence, our framework makes the specification tighter and clearer than what is possible with MUD alone, thereby making IoT systems more secure.

Keywords: Internet of Things · Security · Manufacturer Usage Description (MUD) · Clustering

1 Introduction

Internet of Things (IoT), defined as an interconnection of things, people, data, and processes meant to achieve some specified business goals, is an important

This material is based upon work supported in whole or in part with funding from the United States Department of Defense (DoD). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any agency or entity of the United States Government.

emerging technology that is being leveraged heavily by companies. The Business Insider predicts that companies will accelerate their investment in IoT solutions, with the aggregate investment forecasted to be $15 trillion between 2017 and 2025 [29]. Wireless sensor networks (WSN) are a key foundation of IoT solutions, helping things to communicate data to people and processes, and vice versa.

WSNs comprise multiple sensors spread across an area, that measure certain physical parameters of the environment and communicate these measurements wirelessly to a gateway, which in turn forwards it to a control server. WSNs may also "close the loop", i.e. the control server may process these measurements and command some actuators to perform actions to control some physical parameters. Advances in microcontroller, wireless communication, networking, and sensor technologies have made WSNs economical, power-efficient, and easy to setup and maintain. These developments are driving the adoption of WSNs for IoT solutions, for example, in smart power grids, smart water networks, intelligent transportation, health care, industrial process monitoring and control, and smart homes. Aided by technologies like WSNs, the IoT world is growing from 2 billion smart "things" in 2006 to a projected 200 billion by 2020, i.e. around 26 smart "things" for every human being on Earth [17].

As WSNs are becoming more common and the IoT is starting to control important infrastructure, ensuring security in IoT is becoming critical. IoT has deep penetration in manufacturing, healthcare, and business. By 2025, the global worth of IoT technologies is projected to be $6.2 trillion, with the maximum value from health care ($2.5 trillion) and manufacturing ($2.3 trillion) [17]. Aruba Networks also predicted that 87% of the health care organizations would have adopted IoT technologies by 2019 [16]. These numbers illustrate IoT's importance, and also imply that security breaches would have serious consequences for people's lives, industry, and the economy.

Security breaches may happen at the level of the device, in the communication protocols, and/or in the cloud. Current approaches to security are mostly reactive, i.e. software, protocol and chip designers take remedial measures after attackers or researchers discover loopholes. Attackers then again find vulnerabilities in the "fixed" designs, and the cycle continues, like a cat and mouse game that gets increasingly difficult as the complexity of the systems increase. For example, in 2015, two cyber-security experts exploited vulnerabilities in the IoT communication layer and used a man-in-the-middle attack to take control of a Jeep on the freeway, leading to the recall of 1.4 million vehicles. Further, in 2015, a UK based telecom and Internet provider, "Talk Talk", was subject to several security breaches, wherein customers's credit card details were exposed, as they were stored unencrypted in the cloud. This prompted cloud providers to improve their service's security.

IoT devices can be easy targets for attackers due to vulnerabilities in the firmware, weak passwords, open telnet ports, and other loopholes. The situation is aggravated by scale and diversity - while administrators may find it easy to keep a few devices secure, it is more challenging to do so for hundreds and thousands of different types of devices, which may be from different vendors.

Cisco Systems predicts that by 2020 there will be some 50 billion networked devices [2]. As the number of devices grow, more security breaches will follow as security solutions and improvements are not keeping pace [28]. IoT botnets (e.g. Mirai, BrickerBot and Hajime) are commonly used in attacks carried out using IoT devices. These botnets have three major characteristics: their setup is fast and easy, their distribution is rapid, and the deployed malware is difficult to detect [2]. The common solution to thwart botnets has been to use stringent policy rules. Unfortunately, policies have not been expressive enough to handle these security breaches [32]. As the policies depend on user's activity and device features, they tend to become complex and difficult to manage. The policies also vary with the manufacturer, as each manufacturer provides different features to the users.

Securing the network has always been a subservient motive in the industry. In 2018, we saw a significant number of companies updating their privacy policies and terms of services. Due to data breaches in major companies, security and privacy has become one of the primary concerns of both the service providers and the consumers. Owners and providers often do not take basic measures to secure their networks [26]. Most of the major attacks took place because users and providers did not upgrade the kernels or did not change the default password on their network devices [2]. With a simple search on "Shodan", a list of default passwords of every device can be found [12].

Prevention is better than cure, so proactive security measures are needed rather than reactive ones. In this work, we implement a framework that leverages learning techniques and proactively secures IoT devices. The rest of this paper is organized as follows. Section 2 compares our study with related work in the academia and the industry. Section 3 explains our theoretical framework, describing our architecture and our chosen learning technique. Section 4 then details the implementation of our framework. Section 4.3 presents our results and finally, Section 5 concludes our paper.

2 Related Work

IoT devices have a specific purpose, and thus have a small number of predictable traffic flows [31]. This has been harnessed in a positive way by the IETF's Manufacturer Usage Description (MUD) framework. In the MUD framework, the manufacturer of an IoT device specifies the device's intended purpose and communication patterns in a MUD file, which is an instance of a YANG model, serialized in JSON format. This MUD file is stored on the manufacturer's website and can be fetched using a MUD Uniform Resource Locator (URL). The IoT device transmits this MUD URL to the Gateway when the device joins the network. An entity in the network management system, termed the MUD Controller, uses this MUD URL to fetch the MUD file. The MUD Controller then ensures that the IoT device's behaviour and communication is constrained accordingly, for example, by applying access control lists (ACLs) at the Gateway.

The promise of the MUD framework for securing networks and the challenge of integrating it has begun to attract the research community's attention.

YANG can be used to describe the normal models and NETCONF deliver them to devices [11,15,18]. Tools like MUDGEE have been developed for generating MUD files from device traffic traces (PCAPs). MUDdy tool uses the concept of metagraphs to check the validity of the MUD file [23]. MUD policies have been integrated with SDN for IoT intrusion-detection by developing and implementing a system that translates MUD policies to flow rules that can proactively be configured on network switches [22].

Due to the predictability in device behaviour, signature-based machine learning techniques have been employed on network traffic to classify and identify the devices [27]. Machine learning is becoming popular in network and security. Machine learning has been used to successfully identify the source Network Interface Card (NIC) over wireless medium by employing Support Vector Machine (SVM) and k-nearest neighbours (kNN) methods [21]. Various intrusion-detection tools, e.g. OSSEC, Snort, Suricata and Bro, can be used to detect and prevent attacks using rule-based approach. Supervised learning systems can be used for detecting malware by analyzing network traffic using tools like WEKA, which can outperform these intrusion-detection tools [19].

While the MUD framework is an important step in constraining the behaviour of IoT devices and keeping them on a leash, it is still in the draft stage and has its limitations. The YANG model in the latest draft only supports specifying inbound and outbound ACLs, which may not be enough to keep the device on a tight leash. For example, an attacker may hijack devices and overwhelm a gateway by increasing the data rate, while using the prescribed port numbers and still remaining within the defined normal "box". To keep the devices on a tighter leash, the normal behaviour description should consist of multiple smaller "boxes" rather than one big "box", i.e., more features have to be specified, with narrower ranges. In the above case, the data rate calculated over a specified interval of time could be included in the normal behaviour specification. Further, it could be a challenge for the manufacturer to create a MUD file for their device if the intended behaviour depends on operational settings. Consider a temperature sensor that may be configured by an administrator to transmit readings at different intervals, depending on the operational environment. In such cases, the normal behaviour cannot be tightly defined by the manufacturer a priori, as it is set by the operator. Hence, the MUD file may be at best incomplete, or worse, unavailable.

In our work, we overcome these limitations by developing a learning-based system that observes the wireless communications of devices over a period of time and learns a model for the normal behaviour of each device. The system validates the communications of all devices against the corresponding normal models in real time. Deviations from the normal model are used to isolate and block misbehaving devices, before they can they do damage. Hence, because our system learns the normal model, it can augment an incomplete behaviour specification in the MUD file, or substitute a missing one. We provide flexibility in both choosing the features to be included in the normal behaviour specification, and the combination in which the features are used to specify the normal behaviour. This helps our system to specify multiple smaller normal “boxes”, giving less leeway to attackers. Further, as our system captures the wireless transmissions of the devices directly off the air, we get access to extra features for defining the normal behaviour, like signal to noise ratio, modulation type and other physical layer information. Therefore, our proposed system can make the normal behaviour specification clearer than that possible using MUD alone, i.e. clearer than MUD.

3 Overview of the Framework Architecture

3.1 System Architecture and Components

Our goal is to design a proactive system that creates normal profiles of IoT devices by observing their wireless transmissions, detects abnormal behaviour, identifies the abnormal device and takes actions against that abnormal device.

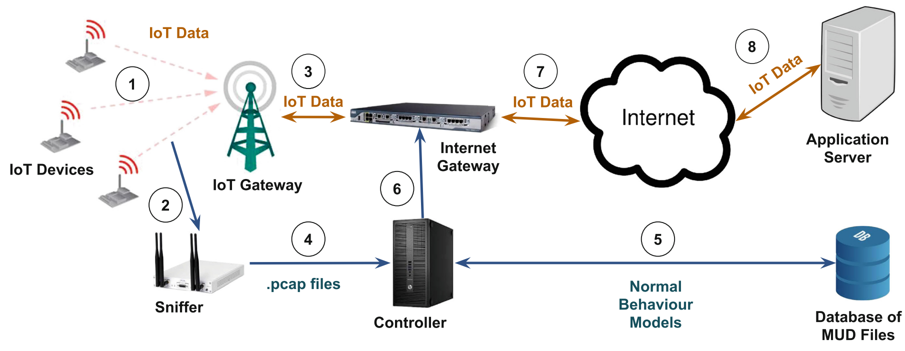

Fig. 1. High level functional blocks of the system’s architecture, with arrows indicating directional or bidirectional flow of traffic

Figure 1 shows the functional blocks in our system’s architecture. In our framework, IoT Devices, which may be sensors or actuators, transmit their measurements wirelessly to the IoT Gateway or receive actuation signals from the IoT Gateway. These communications are shown by ①. In order to create a normal profile of the devices, their communications have to be captured, decoded and then characterized. We use a Sniffer to eavesdrop on and decode the device’s communications as shown by ②. The IoT Gateway demodulates the wireless transmissions of the device and is responsible for forwarding the traffic to the Internet Gateway as shown by ③. The Internet Gateway receives the traffic generated by the sensors via the IoT Gateway and, depending on the actions that are configured on the Internet Gateway, the traffic is then forwarded to the destination, which is the Application Server, ⑧ via the Internet ⑦. All the actions taken by the Controller to thwart the attacks will be implemented on the Internet Gateway. The Sniffer is responsible for capturing and decoding the packets sent or received by the IoT Devices. Packet Captures (pcaps) are generated as an output of the Sniffer and are sent to the Controller for further processing, as shown by ④. The Controller, which is the core of our system and is detailed in Sect. 3.2, receives PCAPs as input and is responsible for extracting relevant features, creating a normal model ⑤ and enforcing the generated model on the Internet Gateway ⑥. The functions of the Controller are detailed in Sect. 3.2.

3.2 Learning Technique

As mentioned above, the Controller is the core of our system's architecture. A simplified block diagram of the Controller is shown in Fig. 2. The feature extractor module receives pcaps from the Sniffer and extracts the desired features. During the learning phase, the features are stored for a duration of time, termed the "learning interval", and then these stored features processed by the learning module to create the normal models of the devices. These normal models are stored as MUD files in the database. The Controller learns the normal models periodically after every learning interval to ensure that the models are updated in case the normal behaviour of the devices change with time. Once the normal behaviour models have been learnt, the Enforcer enforces these models during the enforcing phase. The Enforcer receives the features of the each device's communication, in real time, from the feature extractor. The Enforcer compares these features with the respective normal models read from the database. If any deviation from the normal model is detected, the Enforcer isolates the misbehaving device and blocks it by applying ACLs at the Internet Gateway.

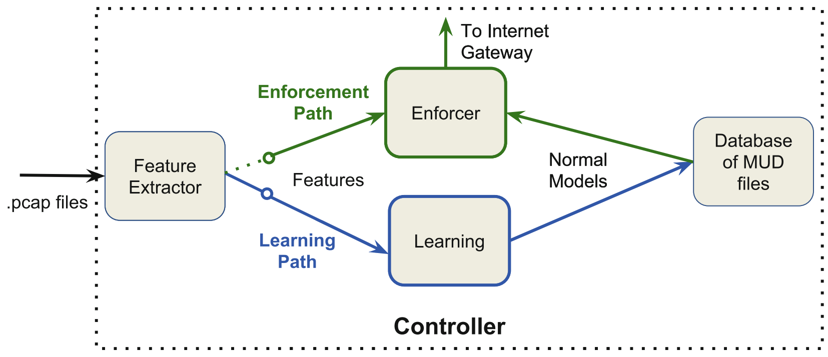

Fig. 2. A simplified version of the Controller, illustrating its main functions - learning the normal model, storing the model into a database, and enforcing the model.

Various learning methods can be used to model the normal behaviour of the device, and these can be categorized into supervised learning and unsupervised learning. Among these methods, it has been shown that performance of unsupervised learning is not affected by unknown characteristics or information and is similar to the performance of the supervised learning [24]. The disadvantage of using supervised methods for characteristics detection is that the labelling procedure of training data is expensive in terms of computation and is time consuming. The unsupervised anomaly detection approach overcomes this problem by making use of data-clustering algorithms, which make no assumptions about the labels or classes of the pattern. Data is grouped such that patterns within the same groups are more similar to each other than they are to patterns belonging to different groups. Since IoT devices have completely different patterns for different features, unsupervised learning using clustering is a good fit.

Among different clustering techniques, hierarchical clustering is preferable when the number of clusters is not known a priori, as it is both more flexible and has fewer assumptions about the distribution of the underlying data. Clustering algorithms like k-means need a criterion to define a correct or acceptable number of clusters, but such information may not be available in some systems [20]. Though hierarchical clustering can be computationally more expensive, it is robust with respect to choice of number of clusters, and can detect outliers (suspicious data points). Hierarchical clustering is also easy to implement, and it is easy to interpret its the results [30]. Hence, a hierarchical clustering learning algorithm is preferable for classifying the behaviour of IoT devices in WSNs, as the number of normal clusters are not known a priori.

Hierarchical clustering starts by treating each data point as a separate cluster. At each step, a distance metric is used to identify the two closest clusters, and these clusters are merged to a single cluster. This continues until all the clusters are merged together. The distance between the two clusters, merged at a given step of the hierarchical clustering algorithm, is termed the "inter-cluster distance". The output of the hierarchical clustering is a dendrogram, which is a tree structure showing the hierarchical relationship between clusters, with cluster numbers on the X-axis and inter-cluster distance on the Y-axis. To obtain a group of clusters, the dendrogram can be "cut" by a horizontal line, corresponding to a specific inter-cluster distance, and a specific cluster merging step.

After performing hierarchical clustering, the next step is to find the best location of this horizontal line, to cut the dendrogram thus obtained, such that the resultant cluster partition is the optimum. The increase in inter-cluster distance between successive cluster merging steps, which we denote by $\delta_{interClusterDistance}$, can be used to guide this search. A large value of $\delta_{interClusterDistance}$ at a merging step indicates that the merged clusters are dissimilar and probably belong to different groups. Hence, the candidate locations to cut the dendrogram are those where the next cluster merging step leads to a large value of $\delta_{interClusterDistance}$. To choose the best location among all these candidates, a metric is needed to measure the "quality" of the corresponding clustering results. We use the silhouette analysis technique as the metric. This technique computes a metric for each data point, termed the "silhouette score", which is a measure of how well that data point lies within its cluster.

The silhouette score for a data point x is calculated as follows:

$$s(x) = \frac{\bar{d_{c'}}(x) - \bar{d_c}(x)}{max\{\bar{d_{c'}}(x), \bar{d_c}(x)\}},$$

where $\bar{d_c}(x)$is the average distance between x and all other datapoints within the same cluster, and $\bar{d_{c'}}(x)$is the smallest average distance between x to all points in any other cluster that does not have x as a member.

The silhouette score ranges from -1 to $+1$, and can be averaged over all the data points to obtain a mean silhouette score. Higher values of the mean silhouette score indicate that the cluster partition is appropriate, while lower values indicate that the partition may have too many, or too few clusters. Among all the candidate clustering results described previously, the optimum is chosen such that it yields the highest mean silhouette score. This learning algorithm is detailed in Sect. 4.2. In this way, based on these optimum clusters obtained with hierarchical clustering and silhouette analysis, normal models are learned for classifying the behaviour of IoT devices.

4 Testbed Implementation, Experiments and Results

To put theory to practice, a testbed was implemented, comprising all the components of the theoretical framework discussed above: a wireless sensor network, a Sniffer that eavesdrops on and decodes all wireless transmissions, and a Controller that supervises the devices. To enable the research and academic community to build on this work, this testbed has been open-sourced [13,14]. In this section, we describe this testbed and present the results of experiments conducted using it.

4.1 System Components

LoRa [7] and LoRaWAN [1], which are predicted to be the dominant physical and medium access control protocols for private LPWANs [8], were chosen to demonstrate the capabilities of our framework and its potential to be used in real life IoT networks. Other IoT communication technologies like SigFox, Narrowband IoT (NB-IoT), and LTE CAT 1, can also be supported over our testbed, with compatible devices, a compatible gateway, and by updating the Sniffer to demodulate and decode the respective communication protocols. As depicted in Fig. 3, Multitech mDot [10] devices were chosen as the programmable IoT Devices, a Multitech Conduit LoRa Gateway [9] as the LoRaWAN Gateway, and an Ettus USRP B210 [3], controlled by the GNU Radio toolkit, as the Sniffer. Linux containers [6], managed using LXD, were used to emulate the IoT Gateway, Application Server and the Controller. An alternative to decoding transmissions "off the air", is to capture packets "on the wire", at the interface of the Gateway towards the IoT Devices, and forward these packets to the Controller.

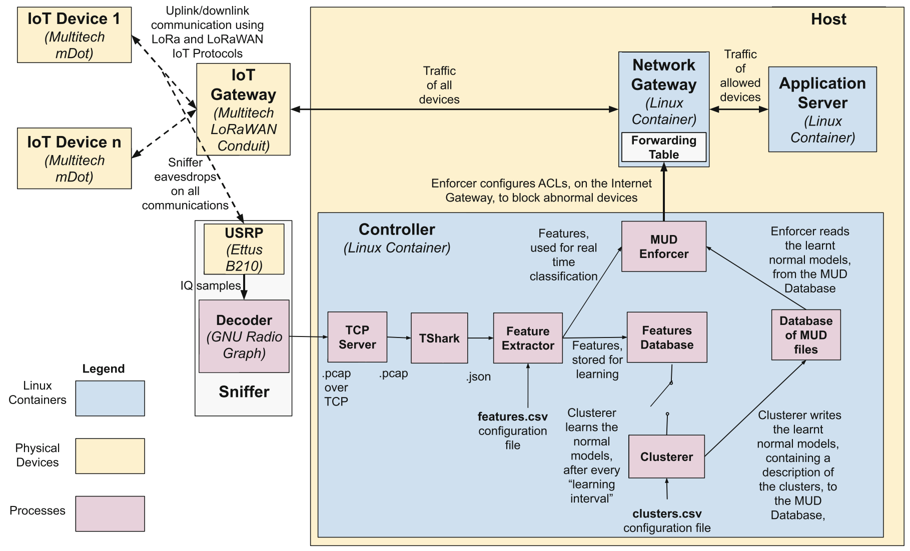
IoT Device 1
(Multitech mDot)
IoT Device n
(Multitech mDot)
Uplink/downlink communication using LoRa and LoRaWAN IoT Protocols
IoT Gateway
(Multitech LoRaWAN Conduit)
Sniffer eavesdrops on all communications
USRP
(Ettus B210)
IQ samples
Decoder
(GNU Radio Graph)
Sniffer
Legend
Linux Containers
Physical Devices
Processes
Host
Traffic of all devices
Network Gateway
(Linux Container)
Forwarding Table
Traffic of allowed devices
Application Server
(Linux Container)
Enforcer configures ACLs, on the Internet Gateway, to block abnormal devices
Controller
(Linux Container)
Features, used for real time classification
MUD Enforcer
Enforcer reads the learnt normal models, from the MUD Database
TCP Server
TShark
Feature Extractor
Features Database
Database of MUD files
.pcap over TCP
.pcap
.json
Features, stored for learning
features.csv configuration file
Clusterer learns the normal models, after every "learning interval"
Clusterer
clusters.csv configuration file
Clusterer writes the learnt normal models, containing a description of the clusters, to the MUD Database,

Fig. 3. Implementation diagram depicting the mapping of the theoretical framework components to actual devices, Linux containers and processes.

The mDots use LoRaWAN's Over the Air Authentication (OTAA) scheme to the join the LoRaWAN Gateway. This scheme requires the mDots to be setup with certain keys. This configuration was performed using MBed's online compiler. To emulate normal and abnormal behaviour, the mDot was also programmed to transmit payloads of desired length, and at desired intervals of time. The mDots were configured to use the US 902–928 MHz industrial, scientific and medical (ISM) band, and to operate in hybrid mode in sub-band 1. Therefore, the mDots hopped in frequency between one of eight channels spaced at intervals of 200 kHz from 902.3 MHz to 903.9 MHz when transmitting with 125 MHz bandwidth, and will transmit at 903 MHz when using 500 MHz bandwidth.

The transmissions of the mDot are decoded by the Multitech LoRaWAN Gateway and encapsulated in TCP datagrams with the destination IP being the interface of the Application Server container connected to the Internet Gateway container. These datagrams are then forwarded to the Internet Gateway, whose forwarding table is configured to forward these packets out on the correct interface. These packets are received and logged by a TCP server at the Application Server container.

The wireless transmissions of the mDot devices are captured off the air and sampled by the USRP, connected to a laptop via a USB 3.0 cable, and controlled by a GNU-Radio flow graph running on that laptop. The USRP is configured to receive at a center frequency of 914.9 MHz and sample at 30 MHz, allowing it to capture all transmissions from approximately 900 MHz to 930 MHz. The IQ samples generated by the USRP are transmitted over the USB cable to the laptop, where they are processed by the flow graph in real time. This flow graph uses a channelizer to split the samples into eight streams, corresponding to the eight channels of sub-band 1, as described previously. Each of these eight streams is then processed by blocks from an open-source library [4], that demodulates and decodes the IQ samples into LoRaWAN packets at a specific spreading factor and bandwidth. The decoded LoRaWAN packets are presented in pcap [5] format to a TCP client, which forwards them to the Controller.

The Controller receives and processes the packets from the Sniffer, and comprises multiple sub-components as illustrated in Fig. 3. The TCP Server receives, from the Sniffer, the packets sent by the devices or the gateway, in .pcap format, and feeds them to a T-Shark process. The T-Shark process translates these packets from .pcap into .json format, where each packet is represented as a JSON object. A JSON object is a set of key value pairs, and, in this case, the key value pairs of a packet's JSON object directly correspond to that packet's pcap fields and their respective values. The feature extractor consumes these JSON objects, extracting and storing relevant features in a database. The features to be extracted are specified in a configuration file, allowing the framework to be flexible and adapt to different protocols. These features may be any header of any layer in the network protocol stack, for example the spreading factor in the LoRa PHY header, or may be statistics calculated from the flow, for example the inter-arrival time. In this way, each packet, transmitted or received by an IoT device, is eventually represented by a set of features, termed as "packet-features". Furthermore, the feature extractor can identify and distinguish between different devices based on certain field(s) in the packet. This enables the feature extractor to store features of each device separately, enabling the framework to learn a normal behaviour model for each individual device. For the LoRaWAN protocol, the "device address" field in the MAC header is used to uniquely identify a device within a LoRaWAN network. This database of features, gathered over a period of time, is used as a training dataset by the Clusterer to create a model for the normal behaviour of each IoT device. Once the normal behaviour models have been learnt, the Controller can start classifying packets in real time, i.e., the feature extractor forwards a packet's features and the device identity to the Enforcer, which validates the features against that device's normal model, as learnt by Clusterer and saved in the database. The learning and classification functions, performed by the Clusterer and the Enforcer respectively, are detailed in the following subsection.

4.2 Learning

The proposed framework enables the administrator to tightly characterize the IoT devices by allowing her to influence the number and the dimensions of the normal "boxes". This influence is exerted by specifying multiple different combinations of features in a configuration file, which is read by the Clusterer. The Clusterer generates normal clusters for each specified feature combination, by processing the features stored in the database. After the features from the captured packets are stored in the database for the learning interval, the Clusterer learns normal behaviour models from these stored features. To better understand the algorithm, we step through it using an example. Assume that the administrator has specified two feature combinations in the configuration file, for a LoRaWAN-compliant GPS sensor: "packet length, inter-arrival time" and "frequency". Assume that, normally, the GPS sensor sends packets of length 45 bytes, at intervals of 60 s, and is configured to operate in US sub-band 1. Assume that, during the learning interval, the GPS sensor loses GPS lock a few times and as a result, for some packets, the inter-arrival time is much higher than 60 s. The Clusterer fetches all "packet-features", for this GPS sensor, from the features database. For the first feature combination, "packet length, inter-arrival time", after extracting the "packet length" and "inter-arrival" fields from each "packet-feature", the Clusterer standardizes these two features. As our cluster analysis algorithm uses Euclidean distance to measure the separation between observations and clusters, standardization is needed to prevent features that are on a larger scale than others, from exerting a stronger influence on the distance measurements and clustering results. For example, measuring the inter-arrival time in milliseconds instead of seconds may change the clustering results. Further, when detecting abnormalities, a deviation in packet length of 10 bytes is more significant than a deviation in inter-arrival time of 10 ms. Hence, the features are standardized, and the recommended approach of using absolute deviation around the mean to detect outliers, rather than using standard deviation, is followed [25]. These standardized feature values, arranged in a matrix, are the input to the hierarchical clustering algorithm. As described in Sect. 3, the optimum clusters are obtained by evaluating different clustering results of the hierarchical clustering algorithm using silhouette analysis. In this case, the optimal clustering would comprise two clusters - one corresponding to the specified behaviour of the sensor, and another corresponding to the observations immediately following the GPS loss. The centroids of these normal clusters represent the normal values for this feature combination, and are added to the normal behaviour specification. In this case, depending on how accurately the GPS sensor follows its specifications, the centroid of the first normal cluster is a point with inter-arrival time approximately 60 s, and packet length approximately 45 bytes, and that of the second normal cluster is a point with packet length approximately 45 bytes, and inter-arrival time greater than 60 s, depending on the delay in obtaining a GPS lock. For the next feature combination, "frequency", the Clusterer identifies eight normal clusters, corresponding to the eight channels in US sub-band 1. The centroids of these clusters, corresponding to the center frequencies of these eight channels, are added to the normal behaviour specification. Hence, in effect, the normal behaviour specification specifies that, for the observed behaviour to be considered normal, it should lie in at least one of the above 11 normal clusters.

Once this normal behaviour model has been learnt, it is enforced by the Enforcer, which validated the captured packets against all the normal clusters of all the specified feature combinations. For a given feature combination, a packet that falls outside any of the corresponding normal clusters is marked as abnormal. For instance, if an attacker hijacks the GPS sensor and starts sending packets frequently, at intervals of 10 s, then the second packet transmitted by the hijacked sensor will have an abnormal inter-arrival time. This packet will fall outside the two normal clusters defined for the combination of packet length and inter-arrival time. Hence, the GPS sensor's behaviour will be classified as abnormal, and that sensor will be blocked by the Enforcer by applying appropriate ACLs at the Gateway.

Table 1. Parameters of the experimental setup

Parameter	Value
Hardware	Ettus USRP B210 connected to a laptop via a USB 3.0 cable as the Sniffer, MultiConnect mDots as the IoT Devices and Multitech Conduit as the LoRaWAN Gateway
Software	GNU Radio and gr-lora in the Sniffer for decoding packets, LXD to manage linux containers, TShark for converting pcaps to json format, sklearn python library for clustering, and pyang python library for parsing and writing yang files
"Normal" pairs of (inter-arrival time, framelength)	(10–11 s, 90 or 91 bytes), (21–23 s, 99 bytes)
"Abnormal" pairs of (inter-arrival time, framelength)	(21–23 s, 99 bytes), (11 s, 98 bytes), (15 s, 94 bytes), (16 s, 97 bytes), (20 s, 92 bytes)
mDot operation mode	Hybrid mode in US sub-band 1 (902–928 MHz)
LoRaWAN uplink spreading factor	8
LoRaWAN uplink bandwidth	500 kHz
LoRaWAN downlink bandwidth	500 kHz
USRP center frequency	914.9 MHz
USRP sampling rate	30 MHz

4.3 Experiments and Results

The testbed described above was used to learn normal behaviour models for two mDots. The mDots were configured to use a spreading factor of 8 and a bandwidth of 500 kHz for their uplink transmissions. In their normal mode, the mDots were programmed to alternatively transmit smaller frames with shorter inter-arrival times, and larger frames with larger inter-arrival times, as specified in Table 1. A training database was created by capturing and decoding frames from the mDots for a duration of eight hours. Then, the Clusterer learnt the

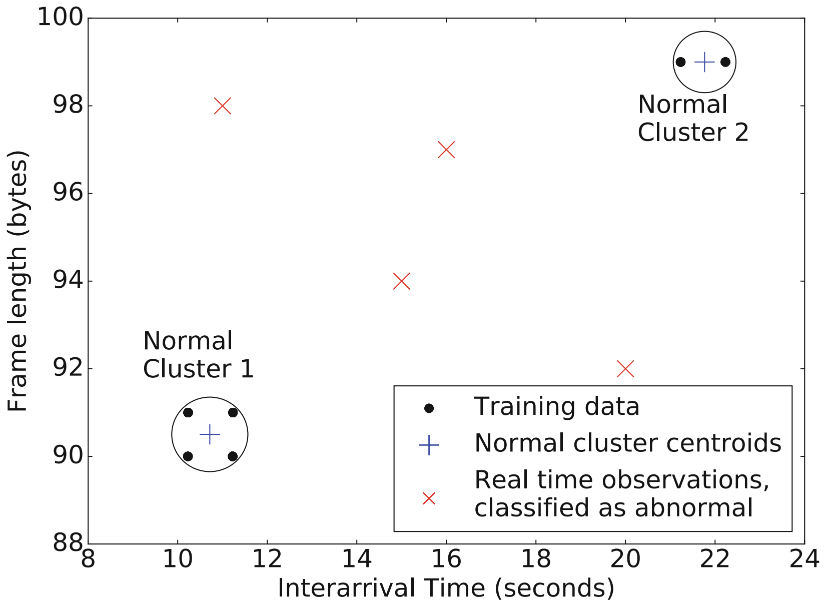

Fig. 4. An illustration of how the normal model is learnt and enforced. The normal model is learnt by clustering the training data, which results in two normal clusters. Real time observations are then classified as normal or abnormal, based on their distance from the centroids of the two normal clusters.

normal model by clustering using the feature combination of "inter-arrival time and frame length", the results of which are shown in Fig. 4.

Abnormal behaviour was emulated by commanding an mDot, via a downlink message, to change the framelength and interval-arrival time to any of the abnormal values listed in Table 1. The resulting observations are depicted by "crosses" in Fig. 4. As these points lie outside the normal clusters, the Enforcer classified the observations as abnormal, and installed ACLs at the IoT Gateway to drop all traffic from the corresponding mDot. The IETF's MUD framework does not currently support specifying the normal behaviour in terms of combinations of inter-arrival time and framelength, with multiple "normal" pairs. Hence, it would not be able to detect this abnormal behaviour. In this way, our framework extends the MUD specification.

5 Conclusion

A framework for enhancing security in IoT WSNs, by learning and enforcing normal behaviour models for IoT devices, was designed, implemented, and demonstrated on a testbed based on LoRaWAN technology. An experiment was conducted to demonstrate the ability of our framework to learn the normal behaviour model, detect abnormal behaviour and thwart potential attacks. This framework extends the MUD specification, allowing an administrator to define the normal behaviour model flexibly and tightly, and thereby enforce the desired level of security in her IoT solutions. The framework needs a Sniffer to be developed for the wireless communication protocol being used, and its application is limited to scenarios where the IoT devices exhibit strong "normal" communication patterns.

Acknowledgements. The authors would like to thank the United States Department of Defense for the funding and the support, and Signalscape for developing the Sniffer.

References

1. About LoRaWAN. https://lora-alliance.org/about-lorawan
2. Cisco's Cybersecurity Report 2017. https://www.cisco.com/c/dam/global/es_mx/solutions/security/pdf/cisco-2017-midyear-cybersecurity-report.pdf
3. Ettus USRP B210. https://www.ettus.com/product/details/UB210-KIT
4. gr-Lora: GNU Radio blocks for receiving LoRa modulated radio messages using SDR. https://github.com/rpp0/gr-lora
5. Libpcap File Format. https://wiki.wireshark.org/Development/LibpcapFileFormat
6. Linux containers. https://linuxcontainers.org/
7. LoRa Modulation Basics, by Semtech. https://www.semtech.com/uploads/documents/an1200.22.pdf
8. LPWAN Market Report 2018–2023, by IoT Analytics. https://iot-analytics.com/lpwan-market-report-2018-2023-new-report/
9. Multitech Conduit Gateway. https://www.multitech.com/brands/multiconnect-conduit
10. Multitech mDot. https://www.multitech.com/brands/multiconnect-mdot
11. NETCONF RFC 6241. https://datatracker.ietf.org/doc/rfc6241/
12. Shodan's search engine for devices connected to the Internet. https://www.shodan.io/
13. Source code for the "Clearer Than Mud" framework, in gzip format. https://research.ece.ncsu.edu/wireless/MadeInWALAN/ClearerThanMUD/ClearerThanMUDFramework.tar.gz
14. Source code for the "Clearer Than Mud" framework, in zip format. https://research.ece.ncsu.edu/wireless/MadeInWALAN/ClearerThanMUD/ClearerThanMUDFramework.zip
15. YANG RFC 7950. https://datatracker.ietf.org/doc/rfc7950/
16. State of IoT, Healthcare, a study by Aruba Networks, February 2017. https://www.arubanetworks.com/assets/infographic/Aruba_IoT_Healthcare_Infographic.pdf
17. A Guide to the Internet of Things, January 2018. https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html
18. Manufacturer Usage Description Specification, June 2018. https://datatracker.ietf.org/doc/draft-ietf-opsawg-mud/
19. Bekerman, D., Shapira, B., Rokach, L., Bar, A.: Unknown malware detection using network traffic classification. In: 2015 IEEE Conference on Communications and Network Security (CNS), pp. 134–142. IEEE (2015)
20. Bharti, K., Shukla, S., Jain, S.: Intrusion detection using unsupervised learning. Int. J. Comput. Sci. Eng. **2**(5), 1865 (2010)

21. Brik, V., Banerjee, S., Gruteser, M., Oh, S.: Wireless device identification with radiometric signatures. In: Proceedings of the 14th ACM International Conference on Mobile Computing and Networking, pp. 116–127. ACM (2008)
22. Hamza, A., Gharakheili, H.H., Sivaraman, V.: Combining mud policies with SDN for IoT intrusion detection. In: Proceedings of the 2018 Workshop on IoT Security and Privacy, pp. 1–7. ACM (2018)
23. Hamza, A., Ranathunga, D., Gharakheili, H.H., Roughan, M., Sivaraman, V.: Clear as MUD: generating, validating and applying IoT behavioral profiles. In: Proceedings of the 2018 Workshop on IoT Security and Privacy, pp. 8–14. ACM (2018)
24. Laskov, P., Düssel, P., Schäfer, C., Rieck, K.: Learning intrusion detection: supervised or unsupervised? In: Roli, F., Vitulano, S. (eds.) ICIAP 2005. LNCS, vol. 3617, pp. 50–57. Springer, Heidelberg (2005). https://doi.org/10.1007/11553595_6
25. Leys, C., Ley, C., Klein, O., Bernard, P., Licata, L.: Detecting outliers: do not use standard deviation around the mean, use absolute deviation around the median. J. Exp. Soc. Psychol. **49**(4), 764–766 (2013)
26. Loi, F., Sivanathan, A., Gharakheili, H.H., Radford, A., Sivaraman, V.: Systematically evaluating security and privacy for consumer IoT devices. In: Proceedings of the 2017 Workshop on Internet of Things Security and Privacy, pp. 1–6. ACM (2017)
27. Meidan, Y., et al.: ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis. In: Proceedings of the Symposium on Applied Computing, pp. 506–509. ACM (2017)
28. Mendez, D.M., Papapanagiotou, I., Yang, B.: Internet of Things: survey on security and privacy. arXiv preprint arXiv:1707.01879 (2017)
29. Newman, P.: The Internet of Things Report, July 2018. https://www.businessinsider.com/internet-of-things-report
30. Nieves, J.F., Jiao, Y.C.: Data clustering for anomaly detection in network intrusion detection. Research Alliance in Math and Science, pp. 1–12 (2009)
31. Sivanathan, A., et al.: Characterizing and classifying IoT traffic in smart cities and campuses. In: 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 559–564. IEEE (2017)
32. Yu, T., Sekar, V., Seshan, S., Agarwal, Y., Xu, C.: Handling a trillion (unfixable) flaws on a billion devices: rethinking network security for the internet-of-things. In: Proceedings of the 14th ACM Workshop on Hot Topics in Networks, p. 5. ACM (2015)