IoT-KEEPER: Securing IoT Communications in Edge Networks

Ibbad Hafeez*, Aaron Yi Ding†, Markku Antikainen*‡ and Sasu Tarkoma*‡
*University of Helsinki, Helsinki, Finland
†Technical University of Munich, Munich, Germany
‡Helsinki Institute of Information Technology, Helsinki, Finland

***Abstract*—Security problems in environments hosting Internet-of-Things (IoT) devices have become apparent, as traditional signature-based anomaly detection techniques fail to secure them due to complex device-to-device (D2D) interactions and heterogeneous traffic patterns. To tackle this emerging security disparity, we propose IOT-KEEPER, a two-tier platform for securing IoT communications within and across edge networks. In specific, IOT-KEEPER secures not only the device-to-infrastructure (Internet) communication, but also D2D communications between devices within edge networks. Different from existing offline solutions that perform network traffic classification over already collected data, IOT-KEEPER continuously inspects the network to identify any suspicious activities and enforce necessary security policies to block such activities. Unlike legacy solutions such as firewall and NIDS, IOT-KEEPER is able to detect and block anomalous activities in the network with its feature-based clustering framework in real time, without requiring explicit traffic signatures nor additional hardware installation. We have deployed a real-world testbed to demonstrate that IOT-KEEPER can identify misbehaving IoT devices based on their network activity with high accuracy, and enforce security policies to isolate such devices in real time. IOT-KEEPER is lightweight, responsive and can effectively handle complex D2D interactions without requiring explicit attack signatures or sophisticated hardware.**

I. INTRODUCTION

IoT-enabled home automation, monitoring, fault detection, and other systems have opened homes and industrial environments to countless new threats [1], [2]. There are several reasons behind the sad state of IoT security. Smart devices are often developed by fast moving teams in large enterprises or independent startups that have to deal with constrained resources, tight deadlines, and pressure to beat their competitors on the market [3]. These issues make it tempting to cut corners in the product development. The developers may, for example, use unverified code snippets and not follow security-by-design principle [4], [5]. It is also common for the manufacturers to prioritize usability over security, for example, by allowing the use of default or otherwise weak passwords [6]. The simple fact, that it is difficult to build secure systems on constrained IoT devices, may also be one predominant factor to the number of insecure IoT devices in the wild.

As a result of the prevalence of insecure IoT devices, network owners can no longer rely on the assumption that all devices in the network are well-behaving and trustworthy [7]. While this, to some extent, applies to every network, it is a particular concern in small office and home (SOHO) environments where the network owners do not have the know-how or resources to improve security. This, together with the fact that IoT devices are rarely updated [8], makes it probable that some devices in the network will, eventually, get compromised by an attacker.

In this paper, we focus on edge network security and, more specifically, the problem of D2D traffic which may originate from compromised IoT devices. We present IOT-KEEPER, a platform that is capable of predicting malicious network traffic including D2D traffic that stays within the local network. IOT-KEEPER consists of two parts. The IoT devices in the edge networks are connected to a hub (KEEPER GATEWAY), which predicts network activity type and enforces security policies. The classification model, based on which the prediction is done, is trained remotely (KEEPER SERVICE) in order to make the system inexpensive to deploy.

Traditionally, edge networks have been secured with network based solutions such as firewalls and network intrusion detection systems, which are placed at the network perimeter. This, however, is problematic in edge networks that predominantly house IoT devices. We approach the problem from a different angle. IOT-KEEPER does not classify traffic only based on the seen packets. Instead, it can also take into account up-to-date contextual information generated on the basis of any devices' prior network activity. We extract features from the collected data and apply clustering to generate a set of rules for predicting the network traffic. Based on the predictions, IOT-KEEPER creates traffic forwarding and filtering rules. For example, if a device exhibits malicious behavior, it is isolated and its D2D communications with other devices, in local and remote network, is restricted. The classification model is bootstrapped with existing data about device vulnerabilities and network attack signatures [9], [10], which is then continuously upgraded through a feedback mechanism.

The main technical insight behind IOT-KEEPER is on how it uses a lightweight and simple clustering in a way that the traffic-class prediction can be done efficiently. Our system demonstrates how a simple yet efficient clustering in conjunction with sophisticated feature extraction enable us to classify network traffic without requiring any heavy operations. As a consequence, IOT-KEEPER is scalable and be realized on legacy hardware.

More specifically, our contributions are:

- We present IOT-KEEPER, a platform which continuously

monitors network activity, accurately identifies devices' malicious behavior, and updates network configuration to isolate such devices using adhoc overlay networks.
- IoT-KEEPER is the first to use a two tier classification model training and prediction mechanism (using Fuzzy C-mean clustering and Fuzzy interpolation scheme) for securing networks in real time, without requiring sophisticated hardware or any changes on the end-hosts.
- We study the individual features' value distributions and their relative importance to identify the set of most useful features for predicting devices' activity type.
- We demonstrate the performance of IoT-KEEPER in a real-world testbed deployment. IoT-KEEPER boasts high prediction accuracy of 98.6% (FPR=0.014 and FNR=0.013) with a low resource footprint (offering high scalability) and minimal increase in latency ($\approx 1.8\%$).

In Sect. II, we discuss the threat model and existing challenges. We present the design of IoT-KEEPER in Sect. III and explain our feature analysis, clustering and prediction techniques in Sect. IV. Sect. V introduces the dataset used for model training and validation. System implementation and performance evaluation are described in Sect. VI. We describe related work in Sect. VII and discuss the potential applications and possible limitations of our work in Sect. VIII, followed by concluding remarks in Sect. IX.

II. Problem Statement

A. Threat Model

We focus on SOHO environment, which is an exemplary type of edge network. A SOHO network often contains a mixture of devices ranging from personal computers to constrained IoT devices. All devices are connected to the Internet through a single gateway and can often communicate with each other in the same local network (i.e. D2D communication). The attacker is considered to be an insider who has gotten foothold in the local network by compromising one or more devices. The details of how the attacker has been able to achieve this is outside the scope of our analysis. The attacker may have, for example, exploited some known vulnerability in an IoT device or simply logged to a device with its default credentials.

The attacker may launch various attacks from the compromised device in the target network. Our analysis focuses on three types of attacks: network scanning, network flooding, and self-replicating malware. Network scanning attacks include *port scan*, where malicious entity scans a single host to find running services with open ports in the network; *port sweep*, where malicious entity searches multiple hosts running a specific service; and *address sweep*, where an attacker tries to map the network, for example, by sending ICMP packets in the hope of replies. Flooding attacks, on the other hand, include various (distributed) denial-of-service (DoS) attacks aimed to disrupt the availability of target devices or services. DoS attacks can also be used as a smoke-screen to hide another attack. Finally, self-spreading malware spreads autonomously and more or less silently in the network. Malware can stay in dormant state for long time before being remotely triggered to be used as a part of another attack. Once activated, the malware can either completely or partially hijack user's devices, for example, for crypto-currency mining, network scanning, or DDoS attack.

Our goal is to secure edge networks (e.g., SOHO) with a heterogeneous device-base against aforementioned attacks. More specifically, we aim to identify and isolate devices which exhibit anomalous behavior. The traffic monitoring, filtering, and device isolation are done at the gateway. The model, based on which the gateway generates traffic filtering rules, is trained using a trusted remote service. Our proposed solution does not require any changes to the end-hosts, although it can utilize logs of IoT devices.

B. Challenges for Securing Edge Networks

Conventionally, there are two types of mechanisms used in securing edge networks. First, network-based solutions such as firewalls and network intrusion detection systems (NIDS) are installed on the gateways or other vantage points to regulate the traffic. Second, there are host-based software solutions such as anti-malware applications and host-based firewalls. The vendors of those solutions further provide updates and security patches in order to mitigate security vulnerabilities latterly discovered in their products. However, all these solutions fall short in addressing the security concerns that result from D2D communication in IoT-based networks:

Complex D2D interactions: Network-based security solutions typically rely on a number of policies or rules to identify traffic flows [11]. Based on the match, they either allow or deny any given traffic flow. Such signature-based techniques minimize the number of false positives, but their performance is limited by the set of available signatures. In addition, NIDSs are typically deployed at network periphery to guard the traffic that is either leaving or entering the network. Thus, they are unable to protect against intra-network D2D-based attacks.

Lack of collective learning: Host-based solutions, such as anti-virus applications, use huge databases of malware and attack signatures to identify anomalous behavior. Due to the sheer amount of IoT devices and different kinds of communication patterns, traditional approaches for learning attack signatures (e.g., honeypots) do not scale to IoT environments. This is a particular concern with newly introduced IoT devices.

Lack of dynamic and automated enforcement mechanism: Typical edge networks provide connectivity to a wide range of devices including smart phones, tablets, computers and IoT devices. The network managers are often responsible for setting up traffic filtering rules to secure incoming/outgoing traffic. However, as the number of IoT devices keeps growing, the amount of manual work will increase (or alternatively, the applied filtering rules become too general). It is clear that such dedicated network administration is an unrealistic requirement, especially for network owners who do not have the know-how nor resources to improve security.

Lack of context awareness: In IoT-based networks, context awareness is required to efficiently distinguish between seem-

ingly similar *normal* and *anomalous* D2D interactions. As an example, the network traffic patterns of a radiator thermostat may vary based on the time of year and outside temperature. Conventional NIDS mechanisms work merely on the network traffic and do not take this kind of contextual information into account.

Heterogeneity of ecosystem: The emerging IoT ecosystem may consist of heterogeneous devices from hundreds of manufacturers. There is huge diversity in firmwares, software stacks and APIs. This diversity makes it impossible to develop generic host-based security solutions. The constrained hardware and the longevity of the devices further add up to the problem. Often, IoT devices cannot even be patched automatically. This applies to low-end IoT-devices in particular: it may be impossible to put any host-based security mechanisms on them because of their constrained nature. In particular for low-end IoT devices, (traditionally popular) advanced host-based security solutions are not available due to lack of OS, software support, hardware and energy resources available.

Given these challenges that D2D communication in IoT-based edge networks causes, we propose IoT-KEEPER, whose design is described in the following section.

III. SYSTEM DESIGN

Our proposed platform, IoT-KEEPER, consists of two main components: KEEPER GATEWAY and KEEPER SERVICE as shown in Figure 1. KEEPER GATEWAY is a lightweight gateway designed to be agnostic of the underlying hardware (we only require that the gateway supports OpenFlow). Owing to this, KEEPER GATEWAY can be deployed using WiFi access points [12] or PC-on-a-card devices such as Raspberry-PI.

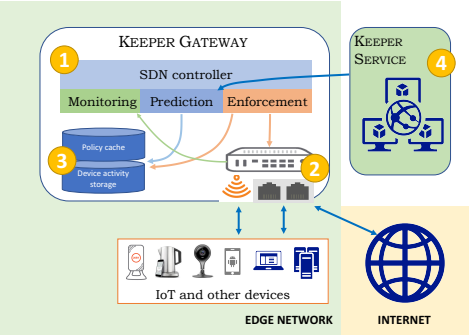


Fig. 1. IoT-KEEPER architecture. KEEPER GATEWAY contains an SDN controller (1), managing OpenFlow switch (2), as well as device activity record and security policy cache (3). KEEPER GATEWAY receives traffic classification model from a remote KEEPER SERVICE (4).

KEEPER GATEWAY has an OpenFlow (OF) switch and an SDN controller running on it. All traffic in the local network goes through the switch via a wired or wireless interface. While the architecture does support multiple OF switches, a typical SOHO environment often contains one switch, which also acts as a gateway. The SDN controller in the KEEPER GATEWAY contains three major modules that are responsible for traffic monitoring, prediction, and enforcement of security policies. The *monitoring* module tracks all network flows within and across the network. It also maintains up-to-date context of the devices' network activity. The *enforcement* module is responsible for security policy enforcement. It tries to match cached security policies with the network flows and generate flow-table rules for the OF switch. The security policies are also used to set up *adhoc overlay networks* (AONs) for isolating devices that exhibit suspicious or malicious behavior.

Figure 2 shows a network where all user and guest devices connect to the same network, which is further partitioned into 3 AONs with varying degrees of network access available for devices. The access for each AON is defined such that D1, D2 and D3 can not communicate with any other devices in the network. Meanwhile, device in *safe* AON have full access to other devices in *safe* AON and restricted access to devices in *suspicious* AON. However, devices in *suspicious* AON can only communicate with each other but can not initiate any communication to devices in *safe* AON.

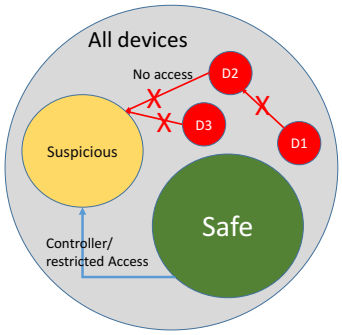


Fig. 2. AONs setup for granting different level of access to user devices connected to the same network.

The AONs allow us to restrict the set of destinations that a device is allowed to communicate with. Thus, it is possible to set up discreet access control on per-device granularity. For example, a gateway may allow a smart fridge to only communicate with user's phone and its own cloud service, and deny communication with all other destinations. AONs are configurable and help us to secure *benign* devices in the network, while still allowing suspicious devices to continue their operation instead of blocking their network access altogether.

We use AONs instead of multiple SSIDs or BSSIDs on the same access points because the number of SSIDs supported by an access points is limited. Due to the limited number of SSIDs, we cannot setup SSID or BSSID with individual network access for each device, therefore, we lose the ability to setup fine grained network access control (at device granularity). Additionally, using multiple SSIDs require forced re-association of devices every time. It will have a negative impact of user experience if they need to re-associate the devices every time network policies are updated or else, device based agents are needed to handle device association to given SSIDs. In case of user devices e.g. laptops, smart phones etc. we can install additional applications to automatically

handle device association to different SSIDs, however, most IoT devices do not support such applications.

If the enforcement module cannot find a security policy matching to a flow from its cache, it requests the *prediction* module to predict the activity type for the traffic flow. The prediction is done using classification rules received from remote KEEPER SERVICE via periodic (asynchronous) updates or on-demand requests. Based on the prediction result, a security policy is generated and stored in local cache. The *enforcement* module uses this security policy to generate corresponding flow-table rules and add them to the OF switch handling given traffic flow.

KEEPER GATEWAY uses *Software-defined Networking* (SDN) for dynamically managing network configuration based on latest network activity. It runs an SDN controller and an OF switch to monitor and filter network traffic across all wired and wireless interfaces. By design, a single KEEPER GATEWAY, running SDN controller, can control more than one OF-capable APs used to setup edge networks. It allows to setup consistent security policies across all the networks setup using those APs.

Finally, KEEPER GATEWAY also runs a web user interface and a REST-API for configuration management and communications with KEEPER SERVICE. The REST-API can be used to interface IOT-KEEPER functionality with third party applications and services.

The backend of the system, KEEPER SERVICE, is primarily responsible for training the traffic classification model. It collects data from a various resources, such as network traffic data, malware databases [13] and public vulnerability databases (CVE [9] and CWE [10]). Classification model training is performed using this data to yield a set of classification rules that are sent to KEEPER GATEWAY. While the architecture does not dictate the use of any specific machine learning algorithms for model training, we use clustering based unsupervised learning technique as this approach saves us from the manual effort of labeling huge amounts of training data.

KEEPER SERVICE is deployed remotely to save the costs and improve the efficiency of model training process. It is possible to use more than one data sources for training data to significantly improve our model generalization and use more resources to minimize the time required for model training. In privacy sensitive use case, KEEPER SERVICE can be deployed locally at user premises and training data can be collected from KEEPER GATEWAY deployed in user network.

Figure 3 shows an exemplary sequence of activities performed for each communication in a typical network setup of IOT-KEEPER. In this example, Alice wants to control Bob's TV connected to the same network. The KEEPER GATEWAY used to setup Bob's network received a set of classification rules from KEEPER SERVICE. These rules were generated by KEEPER SERVICE using previously collected network data, malware signatures and various other sources.

When Alice tries to connect to Bob's TV using her smart phone, the KEEPER GATEWAY sees this as a new activity. It extracts the specified set of features from the connection metadata and looks for a security policy from cache, matching the given request. If a security policy is found, it is used to setup flow-table rules (for allowing or denying the traffic flow) at the OF switch running on KEEPER GATEWAY. In case no matching policy is found, the prediction is performed using the classification rules obtained from KEEPER SERVICE. The prediction results are stored in local cache and flow-table rules are set to handle request traffic flows.

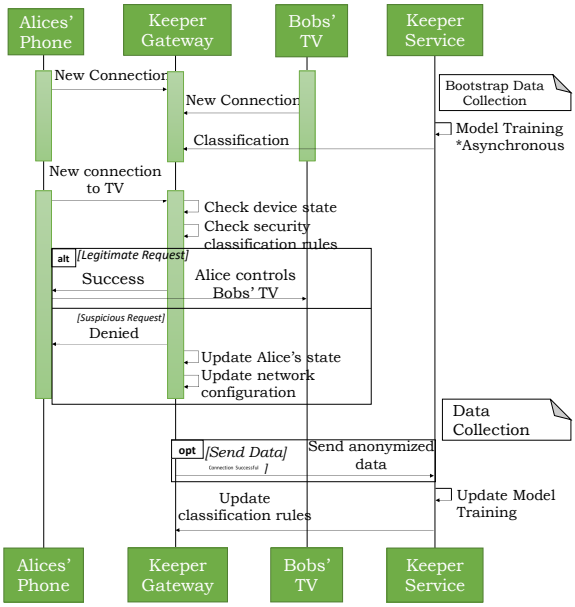


Fig. 3. IOT-KEEPER function diagram to illustrate how different components interact with each other.

IV. TECHNIQUES FOR SECURING D2D

A. Feature extraction

IOT-KEEPER is designed for edge networks that have heterogeneous device base (i.e. IoT devices) and no dedicated network security devices. Therefore, we do not expect that there would be central system logs, such as domain controller or firewall logs, available. Hence, we perform the model training and classification with network logs and, when available, with logs obtained from individual devices.

In the rest of this paper, *device* refers to any host on the network and we assume each device to have a unique IP address. Every feature vector that is used for model training and prediction is identified using source IP, destination IP and timestamp. Features that are extracted from device logs are aggregated and associated with feature vectors via the same identifiers. We extract a total of 39 features including both discrete features and continuous features, as listed in Tab. I and Tab. II respectively.

The same-host, same-service features (i.e. features aggregated over single host and service) are aggregated over n latest connections. This is in contrast to, for example, KDDCup99 dataset [14], where the aggregation is done with a two-second window. Aggregating over a fixed number of connections helps us in identifying probing attacks that employ a wait

TABLE I
DISCRETE FEATURES EXTRACTED FROM NETWORK CONNECTIONS

Type	Feature
L2 Protocol	ARP, LLC
L3 Protocol	IP, ICMP, ICMPv6, EAPoL
L4 Protocol	TCP, UDP
L5 protocol	HTTP, HTTPS, DHCP, BOOTP, SSDP, (M)DNS, NTP
IP Options	Padding, Router Alert

TABLE II
CONTINUOUS FEATURES EXTRACTED FROM NETWORK CONNECTIONS AND DEVICE LOGS

Type	Feature
Source and Destination	No. of unique destination IP addresses No. of unique source and destination ports
Counters	No. of total connections, connections for (same source, same destination, same service) Connection durations, SYN packets, SYN errors, REJ errors, Urgent packets
Data	Total data transferred. Total data from source to destination (SRC2DST) Total data from destination to source (DST2SRC) Packet sizes, payload signatures
Authentication	Total attempts for total login attempts (inc. SSH connection, using default credentials, failed login attempts)

mechanism (i.e. introduce delay between connection requests instead of flooding the target with `SYN` request).

We extract five authentication-based features from device logs including the number of successful and failed login attempts as well as SSH connection attempts and incoming service discovery request. These features are aggregated with timestamp and IP-address of the source device. For this to work effectively, time must be synchronized across all devices. This can be achieved by running network time synchronization services using NTP. In this work, we manually account for these time differences between network and device logs.

B. Feature Analysis

We study the distribution of values for all features to understand their relative importance based on variance and modality. Any features that do not significantly contribute to clustering can be deselected to reduce the model complexity.

Figure 4 shows the cumulative distribution function (CDF) plots for a subset of features extracted from network metadata and device logs. For example, Fig. 4a shows the distribution for the number of unique destination IP-addresses contacted by the devices in the network. The data shown here is collected from our testbed setup (see Sect. V for details). It can be seen that the distribution is not Gaussian, but heavy tailed with majority of the probability mass lying in smaller values, as more than 70% of the devices connect to fewer than 20 unique destination IP-addresses.

On the other hand, the tail of the distribution contains higher values encompassing data points i.e. devices which connect to more than 500 unique destinations. We are primarily interested in the tail of distribution as it contains the data points exhibiting peculiar (potentially *malicious*) behavior.

Figures 4d-4g show that authentication-based features also exhibit non-Gaussian distribution where more than 75% probability mass of feature values are found in range $[0, 5]$. We expect these distributions to result in densely packed clusters with crisp boundaries and clearly marked outliers.

The feature value distributions also show correlation among different features. For example, Fig 4c and Fig 4e show that an *auth-attack* (attacker tries to login to service running on end host) is reflected in network activity as the number of connection requests to given destination increases abnormally. Similarly, SSH attack will result in higher number of network connection between given source-destination host pair over a specific destination port i.e. 22.

These distributions also give useful information about the semantics of various attacks. For example, Fig 4f shows that in an *auth-attack*, majority of authentication attempts are made using default set of credentials. This default set may contain credentials setup by device manufacturers or commonly used by users. Meanwhile, Fig. 4g shows that if the authentication attempts using default credentials are not successful in the first few attempts, most of the following attempts fail, showing that users who tend to change default login credentials choose reasonably unique passwords. However, we can not claim this to be absolutely true because of limited scope of dataset and the number of default (most often used) credentials used for these attacks.

Using *principal component analysis* (PCA) for dimensionality reduction prior to clustering does not improve prediction accuracy in our scenario because of the distribution of feature values in our data. Theoretically, as PCA projects high-dimensional space on to principal components, it retains data variance while reducing data dimensionality by projecting data on first few principal components, each of which points to the direction of maximum variance.

In case of our data, the variance expressibility is affected by the range of feature values and the variance of feature values captured by *principal components* does not reflect its significance in detecting *malicious* activity. In some cases, the principal components obtained by PCA, encompassing most of the variance may fail to capture the outliers. Such data points, found in the tail of distribution, are of maximum interest for us since they represent the deviation from normal activity i.e. highlighting anomalous activities in the network. Therefore, we do not perform PCA analysis for dimensionality reduction in this work.

C. Feature reduction

We use *correlation-based feature selection* (CFS), *deviation method*, and feature values' distributions to identify and remove any features that do not significantly affect the clustering. Feature reduction decreases the classification complexity, reduces resource consumption, and improves generalization.

With CFS, we first identify strongly correlated features by measuring their linear dependencies with Pearson correlation

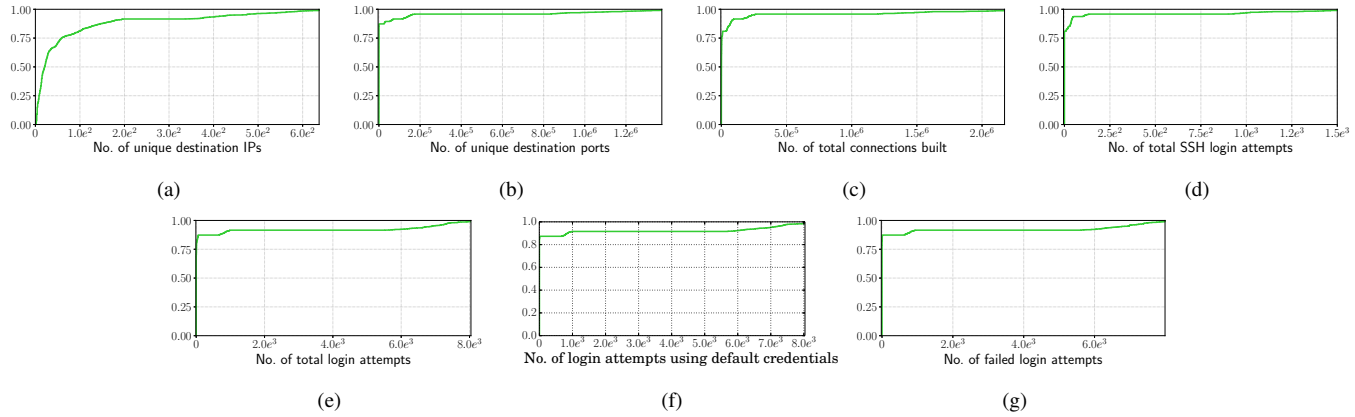


Fig. 4. *Cumulative distribution function* (CDFs) obtained for a subset of features extracted from the data collected using our testbed.

coefficient R. We use Pearson coefficient because it provides fairly accurate results with bounded feature value ranges and when the size of the dataset is fairly large [15]. For any two features which are strongly correlated (i.e. $R \geq 0.99$), we discard one of them as redundant since keeping both features offers limited value to clustering algorithm at the cost of increased complexity and resource consumption.

Using *deviation method*, we first mine 1-length frequent items sets for our feature set to obtain $F_i = [V_1, V_2, \ldots, V_j]$. V_j contains frequent items such that $V_j = [f_i; 1 \leq i \leq n], supp(f_i) \geq m$, where m is minimum support and f_i is frequent item. Frequent item sets for binary features can be mined using algorithms e.g. Apriori [16], FP-Growth [17]. 1–length item sets for continuous variable are calculated by comparing the frequency of the continuous variable against specified minimum support.

The deviation range for each feature is obtained as $D_{v_j} = [f_{max}, f_{min}]$, where $f_{max} = \max(f_j)$ and $f_{min} = \min(f_j)$ for both *malicious* and *benign* classes. If the deviation range of a feature is similar for both classes, the feature is not expected to significantly contribute in defining cluster boundaries, and it is not used in clustering.

Lastly, after clustering, we identify features which do not contribute significantly to the clustering process by studying the normalized feature scores for each cluster. Any features, such as the number of REJ errors, that has similar average scores (within a specified tolerance) for all clusters are considered as non-contributing features. These are pruned off for the next clustering iteration.

In order to make sure that no feature over-influences clustering, all feature values are normalized to range $[0, 1]$.

D. Clustering

We use fuzzy C-mean (FCM) clustering algorithm to generate set of rules used for predicting network activity type. We use the data collected from network traffic and other sources (as detailed in Section V) to bootstrap clustering.

Using FCM, we initially assign a membership value to each data point X_j $(j = 1, 2, ..., n)$ for every cluster C_i $(i = 1, 2, ..., c)$. Each data point X_j is represented as $\left(f_j^{(1)}, f_j^{(2)}, ..., f_j^{(k)}, ..., f_j^{(h)}\right)$ where $f_j^{(k)}$ is value for k^{th} feature in X_j and $1 \leq k \leq 39$ (i.e. 39 features).

The membership value for each data point X_j for each cluster C_i is given as μ_{ij}, $(0 \leq \mu_{ij} \leq 1)$ such that $\sum_{i=1}^{c} \mu_{ij} = 1$ for $1 \leq i \leq c$ and $1 \leq j \leq n$. We then optimize membership values μ_{ij} and cluster centers V_i using Eq. 1 and Eq. 2 respectively, to minimize objective function given in Eq. 3.

$$\mu_{ij} = \left(\sum_{d=1}^{c} \left(\frac{\left\| V_i - X_j \right\|}{\left\| V_d - X_j \right\|} \right)^{\frac{2}{m-1}} \right)^{-1}, \begin{matrix} 1 \leq i \leq c \\ 1 \leq j \leq n \end{matrix} \quad (1)$$

$$V_i = \frac{\sum_{j=1}^{n} (\mu_{ij})^m \times X_j}{\sum_{j=1}^{n} (\mu_{ij})^m}, \begin{matrix} 1 \leq i \leq c \\ 1 \leq j \leq n \end{matrix} \quad (2)$$

$$J_m = \sum_{i=1}^{c} \sum_{j=1}^{n} \mu_{ij}^m \left\| V_i - X_j \right\|^2 \quad (3)$$

where m is fuzziness index [18] and $\left\| V_i - X_j \right\|$ is the Euclidean distance between cluster center V_i for cluster C_i and data point X_j. We use $m = 1$ to perform crisp partitioning.

We first assign labels (from activity types in Tab. III) to clusters based on their feature distributions and verify these labels manually using ground truth. Each of these clusters is translated to a classification rule, to be used by KEEPER GATEWAY for predicting network activity type.

E. Parameter Selection

We determine the optimal number of clusters i by examining the degree of cohesion among data points clustered together, *fuzzy partition coefficient* [19] (FPC), and trade-off between sensitivity, specificity and accuracy of classification model.

We begin with randomly choosing a range of possible values for i. We run FCM algorithm for $n = 3000$ iterations to calculate FPC and *within-cluster-sums-of-distances* (WCSD) for each value of i. We choose i with minimum WCSD to

remove any initialization bias and prevent the output to reside in local minima [15]. WCSD is calculated using Eq. 4, where c is the number of clusters, S_i is the set of data points belonging to i^{th} cluster, and x_{ki} is the k^{th} variable of V_i.

$$WCSD = \sum_{i=1}^{c} \sum_{j \in S_i} \sum_{k=1}^{p} \left\| x_{ki} - x_{ji} \right\| \tag{4}$$

We also calculate silhouette values [20] (using Eq. 5) for all data points x_k to verify our choice of i by studying how well a given data point belongs to the cluster it is assigned to. The optimal choice for i will have minimum WCSD and maximum average silhouette value.

$$s(x) = \frac{b(x) - a(x)}{\max\left(a(x), b(x)\right)} \tag{5}$$

Here $a(x) = \frac{1}{k} \sum_{j=1}^{k} \left\| x - p_j \right\|$, where $p_j \in C_i \wedge x \in C_i$. Similarly, $b(x) = \frac{1}{k} \sum_{j=1}^{k} \|p_k - x\|$, where $p_k \in C_i'$ and C_i' is the closest neighboring cluster for x i.e. $C_i' = C_i \in C$ with $min(\|x - V_i\|) \; \forall \; C_i \in C \wedge x \notin C_i$.

F. Prediction mechanism

We can use three characteristic points, a, b, and c, to represent a fuzzy triangular set A (see Fig. 5), where b is the *center point* with maximum membership value for A and where a and c are the *left* and *right* points respectively, with minimum membership value for A.

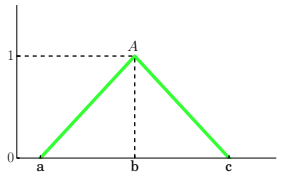


Fig. 5. A triangular fuzzy set shown using its characteristic points a, b, c

We use *fuzzy interpolation scheme* (FIS) to predict the activity type using the rules obtained from clustering. FIS allows us to deduce a conclusion using a sparse fuzzy rule base. Let us consider an sparse fuzzy rule set such as

$Rule\ 1$: $if\ f_1 \in A_{11}, f_2 \in A_{21}, \ldots, f_k \in A_{k1}, \ldots, f_h \in A_{h1} \implies y \in O_1$
$Rule\ 2$: $if\ f_1 \in A_{12}, f_2 \in A_{22}, \ldots, f_k \in A_{k2}, \ldots, f_h \in A_{h2} \implies y \in O_2$
$\vdots$
$Rule\ Q$: $if\ f_1 \in A_{1q}, f_2 \in A_{2q}, \ldots, f_k \in A_{kq}, \ldots, f_h \in A_{hq} \implies y \in O_q$
$Observation$: $f_1 \in A_1^*, f_2 \in A_2^*, \ldots, f_k \in A_k^*, \ldots, f_h \in A_h^*$

$Conclusion$: $y = O^*$

where R_i $(1 \leq i \leq Q)$ is i^{th} rule in sparse fuzzy rule base generated from cluster C_i. A_{ki} and O_i are triangular fuzzy sets for k^{th} antecedent feature $f_k, 1 \leq k \leq h$ and consequent variable y respectively. For any new observation, A_k^* and O^* are triangular fuzzy sets for antecedent and consequent variable obtained as a result of interpolation of spare fuzzy rule base. Figure 6 shows the FIS in practise with n fuzzy rules and h features.

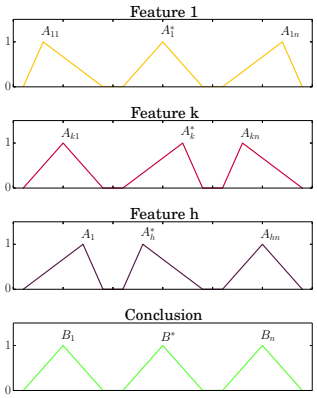


Fig. 6. Sample fuzzy interpolation scheme using n rules with h features.

We use the classification rules obtained from clusters to generate the rule base such that R_i is generated from C_i with h antecedent features and one consequent label assigned to the given cluster.

$$R_i: if\ f_1 \in A_{1i}, f_2 \in A_{2i}, \ldots, f_k \in A_{ki}, \ldots, f_h \in A_{hi} \implies y \in B_i$$

We calculate characteristic points a_{ki}, b_{ki}, c_{ki} for triangular fuzzy set A_{ki} for k^{th} antecedent feature f_k in rule R_i as:

$$b_{ki} = f_q^{(k)}, \quad where\ \mu_{iq} = \max_{1 \leq j \leq n} \mu_{ij}, \tag{6}$$

$$a_{ki} = \frac{\sum_{j=1,2,\ldots,n\ and\ f_j^{(k)} \leq b_{ki}} \mu_{ij} \times f_j^{(k)}}{\sum_{j=1,2,\ldots,n\ and\ f_j^{(k)} \leq b_{ki}} \mu_{ij}}, \tag{7}$$

$$c_{ki} = \frac{\sum_{j=1,2,\ldots,n\ and\ f_j^{(k)} \geq b_{ki}} \mu_{ij} \times f_j^{(k)}}{\sum_{j=1,2,\ldots,n\ and\ f_j^{(k)} \geq b_{ki}} \mu_{ij}}, \tag{8}$$

where b_{ki} has membership value of 1 and a_{ki} and c_{ki} have membership value of 0; $f_j^{(k)}$ is the k^{th} feature's value in sample X_j where $1 \leq k \leq h$ and $1 \leq i \leq c$.

Similarly, we can calculate characteristic variable a_i, b_i, c_i for consequent variable B_i for R_i such as:

$$b_i = O_q, \quad where\ \mu_{iq} = \max_{1 \leq j \leq n} \mu_{ij}, \tag{9}$$

$$a_i = \frac{\sum_{j=1,2,\ldots,n\ and\ O_j \leq b_i} \mu_{ij} \times O_j}{\sum_{j=1,2,\ldots,n\ and\ O_j \leq b_i} \mu_{ij}}, \tag{10}$$

$$c_i = \frac{\sum_{j=1,2,\ldots,n\ and\ O_j \geq b_i} \mu_{ij} \times O_j}{\sum_{j=1,2,\ldots,n\ and\ O_j \geq b_i} \mu_{ij}}, \tag{11}$$

where O_j is expected output class for X_j and $1 \leq i \leq c$. We use Eq. 6–11 to obtain fuzzy triangular sets for the set of classification rules.

The membership value for input feature $f_j^{(k)}$ is $\mu_{A_{k,i}}(f_j^{(k)})$, where $\min_{1 \leq k \leq h} \mu_{A_{k,i}}(f_j^{(k)}) > 0, \quad 1 \leq i \leq p,$ and p is the number of activated fuzzy rules. Using this, we calculate the inferred output O_j^* based on fuzzy rules activated by $f_j^{(1)}, f_j^{(2)}, ..., f_j^{(h)} \in X_j$ using Eq. 12

$$O_j^* = \frac{\sum_{i=1}^{p} \min_{1 \leq k \leq h} \mu_{A_{k,i}}(f_j^{(k)}) \times D_f(B_i)}{\sum_{i=1}^{p} \min_{1 \leq k \leq h} \mu_{A_{k,i}}(f_j^{(k)})} \tag{12}$$

where $D_f(B_i)$ is defuzzified value for B_i in R_i activated by X_j inputs and defuzzified value is calculated using Eq. 13.

$$d(A_{ki}) = \frac{(a_{ki} + 2 \times b_{ki} + c_{ki})}{4} \tag{13}$$

We calculate the weight W_i of given rule R_i $(i = 1, 2, ..., c)$ on the basis of input observations $x_1 = f_j^{(1)}, x_2 = f_j^{2)}, ..., x_h = f_j^{(h)}$ as:

$$W_i = \left(\sum_{d=1}^{c} \left(\frac{\|r^* - r_i\|}{\|r^* - r_d\|} \right)^2 \right)^{-1}, \tag{14}$$

where r^* is the input feature vector $\left(f_j^{(1)}, f_j^{(2)}, ..., f_j^{(h)}\right)$ and r_i is set of defuzzified values for triangular set A_{ki} of antecedent fuzzy sets in R_i. given as $\left(D_f(A_{1,i}), D_f(A_{2,i}), ..., D_f(A_{h,i})\right)$, $1 \leq k \leq h, 0 \leq W_i \leq 1$ and $\sum_{i=1}^{c} W_i = 1$.

The final inferred output is calculated as

$$O_j^* = \sum_{i=1}^{c} W_i \times D_f(B_i) \tag{15}$$

where $D_f(B_i)$ is the defuzzified value of consequent fuzzy variables B_i with $0 \leq W_i \leq 1$ and $\sum_{i=1}^{c} W_i = 1$.

KEEPER GATEWAY predicts the given activity type based on this inferred output. It then generates a corresponding security policy and stores it in cache. The *enforcement* module uses this security policy to update OF-rule at the OF-switch handling the given traffic.

V. DATASET

We performed data collection using a real-world edge network consisting of 38 devices found in a typical SOHO network including smart phones, tablet PCs, laptops, desktop work stations, and smart appliances which run on iOS, Android, Windows, MacOS, Linux, Tizen, and webOS operating systems. All of these devices supported wireless connectivity with 14 devices supporting wired and 32 devices supporting Bluetooth connectivity as well. Table XI gives a list of the devices used for data collection.

For data collection, devices with wireless connectivity are connected to a WiFi network using `hostapd` and WiFi interface of a HP Pavilion Laptop running Ubuntu 16.04 LTS. The WiFi network is connected to local wired network using a TP-Link router. All devices in the testbed have Internet access through one gateway. We also use virtual machines (VM) running Kali Linux and Ubuntu 14.04 LTS to serve as target and attack nodes in different scenarios.

TABLE III
SCENARIOS FOR DATA COLLECTION, REPRESENTING NETWORK ACTIVITY TYPES

Scenario	Description
Authentication attack (A)	A compromised host tries to make multiple login attempts to any host(s) or service(s)
Botnet activity (B)	A (set of) compromised host(s) tries to open many connections to one (or few) destination host(s) where destination host is typically in the remote network.
Normal (N)	Typical usage pattern exhibited by devices e.g. connecting to local devices, cloud services etc.
Port Sweep (P-Sweep)	A compromised host scan set of ports on (one or more) destination host(s).
Port Scan (P-Sweep)	A compromised host scan set of ports on a destination host.
Spying (S)	A compromised host tries to send all user data to a remote destination(s).
Worm (W)	A compromised host tries to scan the network for access to any host and tries to copy malicious content on destination host(s).

Table III shows seven categories for data collection scenarios covering *malicious* and *benign* activity. These scenarios cover devices' normal mode of operation and commonly observed attacks in IoT and SOHO networks [21]–[25] e.g. network and host scanning attacks, privilege escalation, authentication-based attacks, worms etc. We repeated data collection for each scenario $n = 20$ times to avoid any discrepancies and peculiarities in data collection process. For each iteration, average duration of network traffic trace was 600 seconds and only the devices involved in scenario (i.e. host, target) were connected to the network. We recorded all traffic within and across the network on both wired and wireless interfaces. After each iteration, we reset the testbed including all devices involved in data collection. We ran same experiments with non-overlapping set of devices to minimize any redundancy in data points.

Duplicate data points are a common problem in popular datasets (e.g. KDD Cup 1999 dataset [14]) as they may cause bias in learning algorithms [26]. Therefore, we remove any duplicate data points in our dataset. We extract the set of features, listed in Tab. I and II, from the collected traces and use the resulting feature set for clustering. We use the ground truth document listing all the attacks that were included in the data collected to verify cluster labels.

Following real-world edge network model, we expected dataset imbalance issue with *benign* as majority class and *malicious* as minority class. In order to prevent imbalanced dataset from affecting our clustering model, we performed random under-sampling for data points belonging to *benign* activity type. Since, many data points from *benign* traffic are correlated, we do not lose significant ground truth for

majority class with random under-sampling. Meanwhile, we oversampled data points belonging to *malicious* activity type by generating points with SMOTE [27] technique. All six sub-classes of *malicious* activity (given in Tab. III) contain equal number of data points.

VI. Evaluation

A. Evaluation testbed

Our evaluation testbed, shown in Fig. 7, contains few additional devices previously not used in data collection. We used same scenarios (as during data collection) for system evaluation with a different set of devices. That is, if a VM was used to scan a file server during data collection, we use smart phone to perform similar attack on a HTTP server during evaluation. We expect this practice to prevent any learning bias from affecting the accuracy results obtained during evaluation.

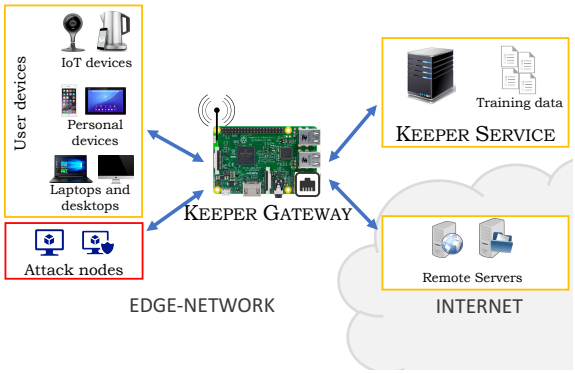


Fig. 7. The testbed used for evaluating IoT-KEEPER. All devices were used for different evaluation scenarios and specialized nodes for generating attack traffic were deployed using VMs. The remote network containing servers was connected to our testbed network via gateway. KEEPER SERVICE used the data collected by us for performing clustering.

The evaluation testbed used a Raspberry PI model 3 running Raspian Jessie Lite to deploy KEEPER GATEWAY and a Core-i5 machine with 32GB memory to run KEEPER SERVICE. We implemented feature extraction, analysis, clustering and prediction scheme in Python using `Flask` framework, `dpkt`, `imbalanced-learn`, and `scikit-fuzzy`. All communication between KEEPER GATEWAY and KEEPER SERVICE was done using REST-API over HTTPS. During evaluation, no data was sent from KEEPER GATEWAY to KEEPER SERVICE and classification model was generated by performing clustering over previously collected data.

We setup KEEPER GATEWAY as a WiFi AP using `hostapd` module [28]. It also runs an Open vSwitch (v2.4.0) (OVS) and KEEPER CONTROLLER, which is based on standard Floodlight SDN controller with our custom modules responsible for traffic monitoring, traffic filtering, state and context management, on-demand security policy enforcement, cache management for security policies and a RestAPI service for communicating with KEEPER SERVICE.

We bridge all wired and wireless interfaces on KEEPER GATEWAY to OVS managed by KEEPER CONTROLLER. In extended deployments, KEEPER CONTROLLER can control multiple OF-capable APs running OVS over stock `OpenWRT` OS, resulting in coherent security across the network.

B. Clustering and prediction accuracy

Figure 8 shows the clusters obtained by performing FCM clustering using our dataset. We performed *multi-dimensional scaling* [29] (MDS) to map 22 dimensional feature space to 2 dimensions surface for better visualization.

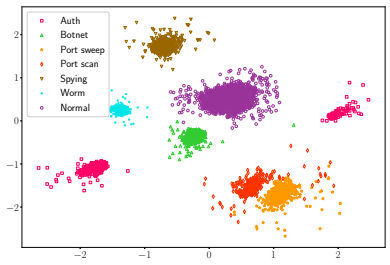


Fig. 8. Clusters obtained as a result of applying FCM clustering algorithm visualized using MDS.

Figure 9 shows the feature value distributions in each of the clusters shown in Fig. 8 using normalized feature scores. We extracted first 18 features from network metadata, where `f1-f9` correspond to connection-related data, `f10-f13` correspond to flagged packets and `f14-f18` correspond to data transferred. `f19-f22` show authentication-related features.

We observe that the distribution of feature values among these clusters show a clear distinction for different type of network activity. For example, distributions of `f1-f6` show a clear difference between network scanning (i.e. P-Scan and P-Sweep) and other types of attacks. Figures 9a and 9b show that port sweep and port scan attacks share the same underlying distribution. This overlapping distribution can results in inaccurate predictions for P-Sweep and P-Scan shown in Tab. V.

TABLE IV
CONFUSION MATRIX FOR ACCURACY ACHIEVED IN IDENTIFYING *benign* AND *malicious* ACTIVITIES

		Predicted		
		benign	***malicious***	**Total**
Actual	***benign***	533	7	540
	malicious	8	532	540
	Total	541	539	1080

Table IV shows the performance of IoT-KEEPER in terms of correctly predicting D2D-centric network activity as *benign* or *malicious*. Using FIS, we achieve a prediction accuracy of 98.61% with precision of 0.985 and an F1 score 0.986. The sensitivity and specificity for our prediction technique was 0.99 and 0.99 respectively. Table V depicts the confusion matrix for prediction accuracy achieved for six types of *malicious* activity (listed in Tab. III).

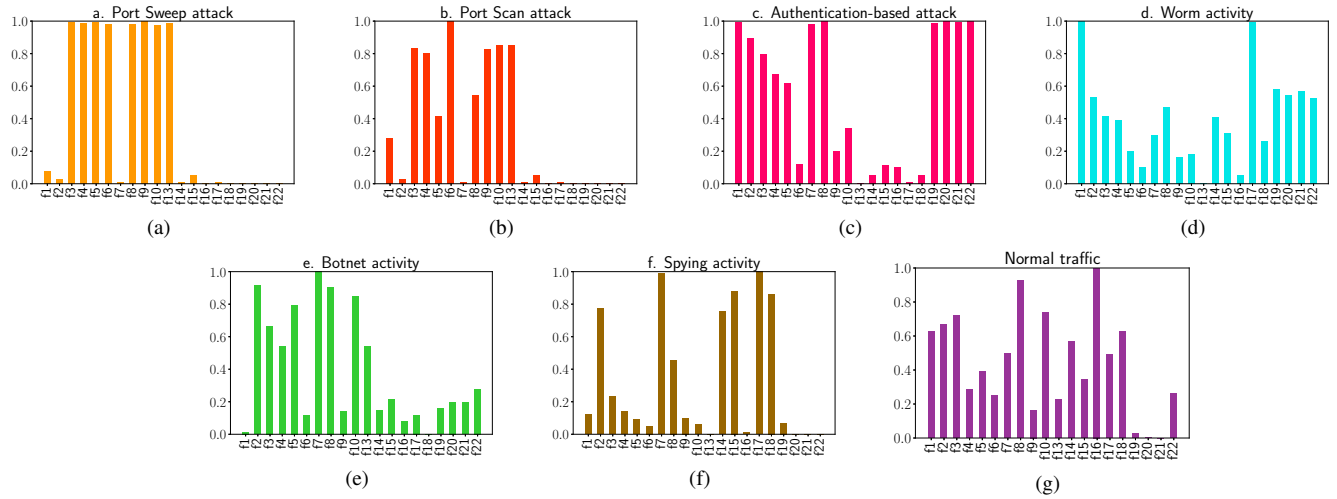


Fig. 9. Normalized feature averages for distinguishing features representing clusters for each attack scenario.

TABLE V
CONFUSION MATRIX FOR SIX TYPES OF *malicious* ACTIVITY. A= ACTUAL, P=PREDICTED

A/P	A	B	PS	Ps	S	W
A	**86**	0	0	0	0	3
B	0	**83**	0	2	2	0
PS	0	0	**81**	9	0	0
Ps	3	0	3	**84**	0	0
S	0	0	0	0	**86**	1
W	3	2	0	0	0	**84**

We observe that although network scanning (P-Scan and P-Sweep) activity is predicted with high accuracy (>0.99 with F1-score of 0.994), the highest number of inaccurate predictions are seen for samples in P-Scan and P-Sweep activities. These inaccurate prediction can be attributed to the overlapping feature values as shown in Fig. 9a-9b.

P-Sweep traffic is clearly differentiable if the attacker scans all 2^{16} ports on a host, opening a large number of connections for source and destination pair (involved in the attack). However, when an attacker sweeps through only a range of ports e.g. $port\_numbers \in [1, 2048], [50000, 65536]$, number of connections built will lower than what is seen in P-Sweep attack and fall in the ranges typically observed in P-Scan attacks. We observe that such boundary cases for P-Sweep traffic are misclassified as P-Scan traffic. Similarly, P-Scan attacks for large range e.g. $[2^0, 2^{15}]$ will result in large number of connections causing these attacks to be classified as P-Sweep attack. It should be noted, that because we use a connection-based aggregation scheme (instead of time-based as explained in Sect. IV-A), we can successfully identify network scanning attacks even when the attacker tries to evade detection by using delays between connections made to different ports.

In some cases, when an IoT device boots up, it starts network broadcasts for polling a range of ports. We presume it as an attempt to scan the network looking for other IoT devices (or possibly an IoT hub) from same manufacturer. This behavior is typically seen in D-Link devices and it can be predicted as *malicious* (P-Scan) resulting in KEEPER GATEWAY updating network configuration to isolate given device. Depending on devices' activity in future, the network isolation can be update to give full network access.

The slightly low prediction accuracy for identifying P-Scan and P-Sweep attacks is not an issue because we achieve a high prediction accuracy for network scanning attacks in general. That is, even though we may mis-classify the exact attack type, we can still identify the *malicious* activity and can set up necessary counter measures in the network.

The reason for why we have emphasized the detection of network scanning attacks is that these attacks are used as a precursor for launching other attacks. For example, before launching a *worm* attack, an attacker first scans network to find out nodes running vulnerable services and then tries to gain access to the these. Similarly, in a *botnet* attack, the attacker first scans target node and then makes a large number of requests to the target. Therefore, if we are able to identify and react to these precursor attacks by hardening network security measures, we can prevent any potential follow-up attacks.

Figure 9d and 9c shows the overlap for average feature values for *worm* and *authentication* attacks. We observe that in cases where an attacker makes several login attempts in an effort to gain access on target host, for planting malicious code, this activity can be inaccurately predicted as an *authentication* attack because of large number of unsuccessful login attempts.

Spying attacks are an interesting example and a good measure to study the crispness of clustering algorithm. Spying activity of any node (i.e. sending user data to unknown destination) is masqueraded to make it indistinguishable from *benign* activity. A small number of unique destinations contacted (only a handful servers where data is offloaded) is similar to activity of an IoT device contacting its cloud services only for uploading data. Looking at the amount of data consumed/(uploaded), the device looks like a smart phone or a

TABLE VI
PERFORMANCE MEASURE FOR PREDICTION TECHNIQUE USED FOR IDENTIFYING SIX SUBCLASSES OF *malicious* ACTIVITY

Measure/Scenario	A	B	PS	Ps	S	W	Mean
Accuracy	0.98	0.98	0.98	0.96	0.98	0.98	0.98
Precision	0.93	0.95	0.96	0.87	0.95	0.95	0.94
Specificity	0.96	0.92	0.90	0.93	0.96	0.93	0.94
Sensitivity	0.99	0.99	0.99	0.97	0.99	0.99	0.99
F1-score	0.95	0.94	0.93	0.90	0.95	0.94	0.94

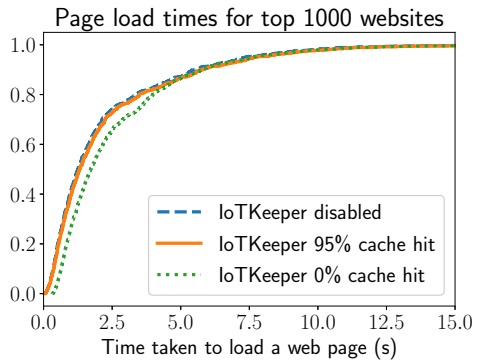


Fig. 10. HTTP page load times for top 1000 websites ranked by Majestic.

laptop consuming a large amount of data over time. However, when we combine the data consumption with number of destination, delay between data bursts and number of connections made, we can identify spying devices with greater confidence. Our technique can successfully differentiate content caching activity of set-top boxes' from spying activity as we take data transfer volumes (source-to-destination and vice versa) into account, for activity prediction.

Table VI shows various performance metrics for our prediction scheme. The results show that our clustering algorithm produces clusters with clear boundaries (see. Fig 8), resulting in high prediction accuracy and low FNR, even though we have overlapping normalized feature scores for different activity types.

C. System performance

We studied the effect of IoT-KEEPER on the latency experienced by the user. For this purpose, we collected page load times for top 1000 websites (ranked by Majestic [30]) in three different scenarios listed as under;

1) IoT-KEEPER disabled i.e. legacy setup.
2) IoT-KEEPER enabled with 95% cache hit rate.
3) IoT-KEEPER enabled with 0% cache hit rate.

Figure 10 show that IoT-KEEPER introduce up to 4.76% and 15.89% increase in latency for 95% and 0% cache hit rate when loading top 1000 most visited websites. For 0% cache hit scenario, we observe that the proportional overhead is much higher (up to 40%) for websites with very small page load times (e.g. google.com and microsoft.com) but low $\leq 7\%$ for websites with longer page load time (e.g. linkedin.com, instagram.com).

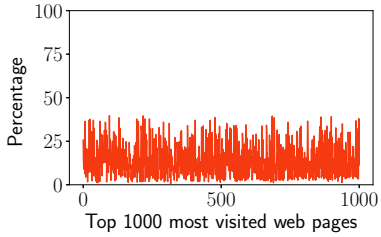


Fig. 11. Percentage of total time required to fetch a web page spent in prediction phase.

The breakdown of time consumed by each step in predicting activity type (see Tab. VII) shows that prediction tasks is largely responsible for increased latency. Figure 11 shows that the time consumed by prediction as a percentage of total time spent in loading top 1000 websites, where prediction accounts for 0.28% up to 39.6% of the total time required to fetch a web page with mean prediction overhead at $13.93\% \pm (9.55\%)$.

TABLE VII
AVERAGE TIME (MS) CONSUMED IN INDIVIDUAL STEPS PERFORMED WHEN PREDICTING TYPE FOR NETWORK ACTIVITY

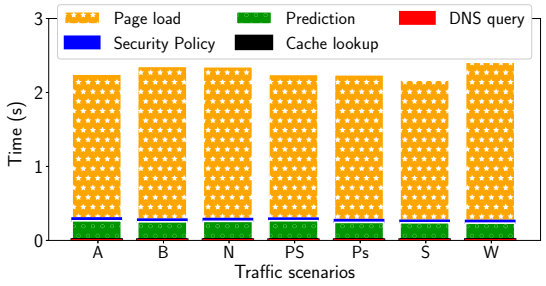
Step	**Time (ms) ($\pm$ StDev)**
Feature extraction	7.6 (2.81)
Prediction	209.2 (28.9)
Security policy generation	28.6 (6.7)

Figure 12 shows the breakdown of the time taken by each individual step to fetch a web page (www.youtube.com). We can observe from Fig. 12a that only the prediction step introduces noticeable overhead on page load time. Meanwhile, for 100% cache hit, latency increase only by 1.8% $(\pm 1.49\%)$.

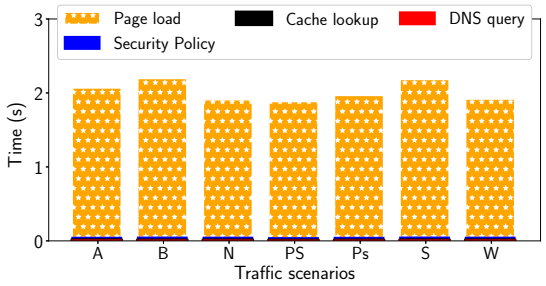
We highlight that IoT-KEEPER only makes prediction when there is no matching security policy available in cache. Once the prediction is performed and a security policy is generated, it is cached so that it can be used for handling similar traffic in the future. Figure 13a shows that with more security policies available in cache, cache hit rate increases, resulting in decrease of number of predictions performed.

Figure 13b further illustrates how the number of rules used by FIS affects the volume of traffic accurately predicted by KEEPER GATEWAY. Specifically, IoT-KEEPER requires ≤ 100 rules to achieve $\geq 93\%$ accuracy.

Because cache-lookup time is negligible compared to time required for prediction and policy generation, IoT-KEEPER caches prediction decisions as security policies. By using hashmap structure for cache storage, we achieve constant lookup time irrespective of cache size, as shown in Fig. 13c. We attribute a few spikes in cache lookup time to underlying hardware and operating system. The deep-memory size of cache increases linearly with the number of security policies



(a) No matching security policy is found in cache and prediction is required



(b) Matching security policy is found in cache, requiring no prediction.

Fig. 12. Breakdown of time taken to fetch a resource (webpage) in different traffic scenarios comparing time taken with *cache miss* versus *cache hit*. *Markers are enhanced for better visualization.*

in cache but it is not considered an issue as modern storage devices offer very high storage densities [31].

We add a user-configurable timeout for every security policy in cache. This time-out refreshes every time the security policy is used to handle network traffic. If the timeout expires and security policy remains unused, it is removed from cache. This approach allows us to maximize cache hit rate without letting cache size increase without limits. We can also bound the cache size to store a maximum number of security policies. In general, such strict measures for limiting cache size may not be required as the cache size only takes 3MB memory for 25000 policies. We observe that majority of network traffic is directed to certain number of destinations e.g. social media, digital media streaming services etc. Therefore, a relatively small number of security policies in cache can result in high cache hit rates (see Fig. 13a).

We study the D2D latency and throughput achieved by IoT-KEEPER for intra- and inter-network communication using both wired and wireless mode of connectivity. These results give a qualitative understanding that our system does not introduce significant variations in latency for D2D communication. In total, we achieve $17.24\,ms$ ($\pm 3.8\,ms$) and $26.45\,ms$ ($\pm 5.73\,ms$) for wired and wireless mode respectively, using IoT-KEEPER.

Our prototype implementation is not fully optimized as every new traffic flow has to be handled by SDN controller and the security policies are cached only on usage basis (not on forecast for future traffic patterns). However, this reference implementation gives us an understanding that IoT-KEEPER does not introduce substantial overheads in terms of latencies.

TABLE VIII
LATENCY (MS) EXPERIENCED FOR D2D COMMUNICATION.

Latency	D5	D6	D7	D8
D1	26.1 (±4.7)	17.7 (±3.1)	23.1 (±5.6)	15.8 (±4.1)
D2	25.3 (±6.3)	18.3 (±4.1)	24.1 (±5.1)	16.6 (±3.7)
D3	27.3 (±5.9)	18.5 (±3.4)	28.9 (±6.4)	16.2 (±1.4)
D4	25.6 (±6.1)	19.1 (±3.6)	31.4 (±5.7)	15.7 (±3.9)

TABLE IX
THROUGHPUT (MBPS) ACHIEVED BY USING IOT-KEEPER.

Server	D1	D2	D3	D4
S1	18.4 (±1.5)	16.4 (±1.1)	17.3 (±1.4)	37.3 (±0.9)
S2	7.7 (±9.7)	7.9 (±12.1)	8.5 (±11.1)	13.3 (±9.1)
S3	15.6 (±6.8)	15.8 (±5.6)	19.1 (±7.1)	21.1 (±7.7)

Table IX shows the throughput performance achieved by using four servers deployed remotely where S1: iperf.funet.fi, S2: iperf.scottlinux.com, S3:bouygues.testdebit.info. In order to avoid bandwidth variations, we conduct throughput testing using device connected over Ethernet interface.

We tested the network throughput by deploying an HTTP server with file of size 1 GB, 5 GB, and 10 GB on a local server. We limit the network bandwidth in five different settings and test the bandwidth as user downloads the files from HTTP server. The results (see Fig. 14) show that we achieve $93.37\%(\pm 2.3\%)$ of the total bandwidth using IoT-KEEPER.

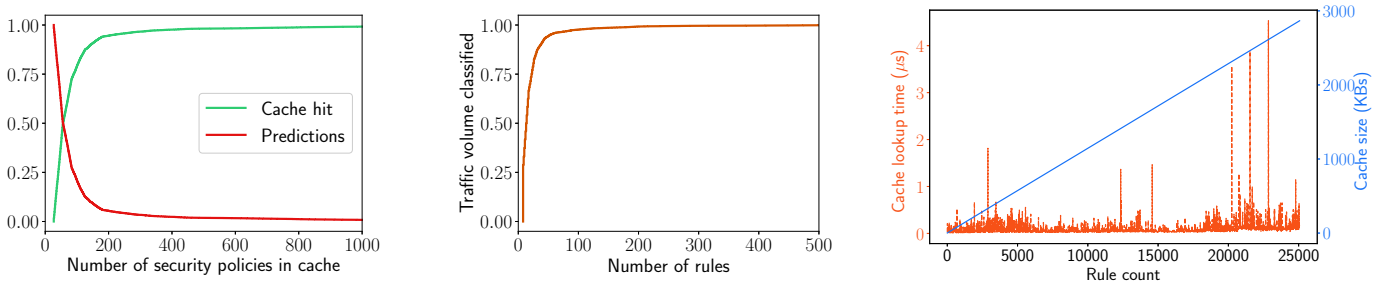
We also studied the variation in latencies experienced for D2D communications as the network activity increases. Fig. 15a shows how the D2D latency varies as the number of security policies used for handling network traffic flows increases. We observe that there is no significant effect on D2D latency as the cache size increases because we get a (nearly) constant lookup time for our cache, as shown in Fig. 13c.

Figure 15b shows that latency for D2D communications varies directly with the number of simultaneous flows in the network. However, we do not see any abnormal increase in latency introduced by IoT-KEEPER. The behavior observed in Fig. 15b is commonly experienced in networks when latency becomes high as more users are connected to the same network and we attribute this behavior to the limitations of underlying hardware.

VII. RELATED WORK

A number of researches have used machine learning (ML) algorithms to detect malicious network activities [32], [33]. Researchers have used data mining (e.g [34], [35]), supervised ML techniques (e.g [36], [37]), and unsupervised ML techniques (e.g [15], [38]) to build network intrusion detection systems (NIDS). Bekerman et al. [33] used 942 features to identify malware by analyzing network traffic. Strayer et al. [39] and Lu et al. [40] studied network behavior and application classification to identify bots in networks.

BotMiner [41] used clustering technique for detecting botnets independent of underlying command-and-control protocol



(a) (b) (c)

Fig. 13. Relationship between the number of rules, number of security policies, its effects on cache hit rate and performance of cache lookup. (a) CDF plot for cache hit rate with respect to number of policies in cache, resulting in lower number of predictions required, lowering the overall latency experienced by the user. (b) CDF plot for the number of classification rules required to predict user traffic class. (c) Behavior of cache lookup time and (deep memory) cache size relative to the number of security policies stored locally.

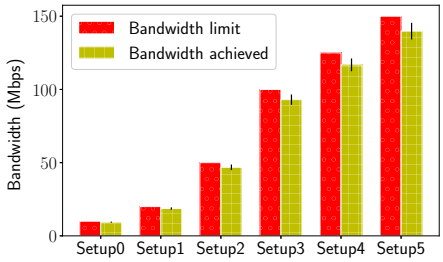
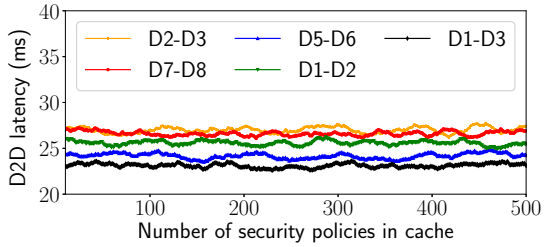
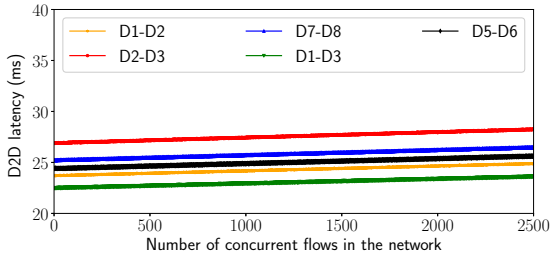


Fig. 14. Network bandwidth performance test.



(a) Variations in D2D latency with respect to cache size



(b) Variation in D2D latency with respect to number of concurrent flows in the network

Fig. 15. Effects of network congestion over the latency experienced in D2D communications within local network.

and strategy. Bohara et al. [15] used unsupervised learning to predict class labels for unlabeled network. These techniques focus on assisting in the classification of data that is obtained from logs of various network security appliances. These solutions are mainly designed for enterprise and data-center environments with enough resources to process and analyze huge amount of training data and classify new instances, and enough expertise to manually translate the results to security policies enforced in networks [34], [42].

On the other hand, there has not been many proposals that would use ML for securing IoT deployments in edge networks, which are central pieces in the network security puzzle [11]. The growth of IoT has unearthed various security issues related to resource-constrained devices that collect and process a lot of sensitive information [43], [44]. Recent proposals have focused on how D2D communication within IoT ecosystem raise threats for securing IoT devices from malicious actors connected to same network [6], [45], [46]. Recent studies have focused on how D2D communication in edge networks has become a threat as malicious actors may have compromised some of the devices [6], [45], [46]. Device-type identification using ML is nascent domain for IoT security [47], [48]. PARADIS [47] uses transmitter-specific imperfections in devices's NIC card to identify device types. IoT Sentinel [49] uses device type information to limit the network access of vulnerable devices. PorfilIoT [50] uses a multistage classification technique for differentiating IoT from non-IoT devices and then finding the actual type of IoT device. IoTSentinel and ProfilIoT rely on fingerprints generated from devices' network activity to train classification models, and thus fail to detect impersonation attacks.

Table X gives a qualitative comparison of IoT-KEEPER with the state of the art. Although traditional NIDS and FW solutions provide low latency and support some level of automation in updating signature database used for traffic filtering and anomaly detection, the process is typically slow and limits the overall efficiency of the system. Similarly, they can secure network communications from IoT devices but they are not designed for securing heterogeneous IoT-dominant networked environments where unknown (potentially untrusted) devices are continuously joining and leaving the network. Additionally,

TABLE X
QUALITATIVE COMPARISON OF NETWORK SECURITY SOLUTIONS (✓: FULL SUPPORT, ○ : PARTIAL SUPPORT, ✗: NO SUPPORT)

System	IoT, D2D	Realtime updates	Automation	Behavior analysis
Firewall	✗	○	✗	✗
NIDS	✗	○	✗	✗
Securebox [51]	✓	✗	✓	✗
IoT Sentinel [12]	✓	✗	✓	✗
IOT-KEEPER	✓	✓	✓	✓

these solutions do not support realtime behavior analysis of the devices' activity.

IoT-Sentinel [12] is also unable to monitor and react to (previously benign) device which starts behaving maliciously after it is connected to the network. It requires manual effort to maintain up-to-date database of device vulnerability profiles. On the other hand, Securebox [51] supports real time analysis of network traffic using a remote service at the cost of high latency and privacy concerns for the users whose data is analyzed by a remote service.

The use of ML techniques for analyzing network traces to detect anomalous activities has shown postive results but all such techniques perform the analysis *offline* and require additional effort to translate it into set of traffic signatures used by NIDS solution deployed in the network. With IOT-KEEPER, we remove this additional step to build a system where latest analysis results are directly (automatically) translated into set of rules used for traffic management in edge networks. This design choice saves valueable resources and improves the reaction time to any anomalous activity in the network.

VIII. DISCUSSION

Our evaluation shows that IOT-KEEPER can secure device communications without requiring explicit signatures or sophisticated hardware. It uses simple features extracted from network data, without requiring explicit labeling of each training data sample, and is able to identify subtle differences between feature value distribution to differentiate between various types of *malicious* activity. We can expand the types of activity identifiable by IOT-KEEPER by using additional features and data sources for classification model training.

We can also use a multi-layer prediction mechanism where we predict activity type as *malicious* or *benign* and then predict the subclass for *malicious* activities. While this increases the prediction accuracy to 0.995, it also increases the latency overhead by 60%. In addition, considering that our goal is to isolate the suspiciously behaving devices (instead of quarantining them completely), the accuracy of predicting different attacks from each other is not our primary goal.

It is possible to perform clustering locally on KEEPER GATEWAY (instead of at KEEPER SERVICE) in order to improve user privacy. However, doing so requires more computation resources at KEEPER GATEWAY and would affect classification model generality due to limited diversity in training data. Using KEEPER SERVICE for classification also allows scalability for dissemination of classification rules to all connected networks, resulting in coherent security. In order to prevent KEEPER SERVICE from becoming single point of failure as a victim of DDoS attack, it is possible to use peer-to-peer protocols for disseminating the classification rules. In this case, checksums and public key encryption can be used to guarantee the integrity of the transmitted rules.

IOT-KEEPER is able to identify major software updates e.g. firmware upgrade, for IoT devices by observing the change in network activity of the device. This change can result in occasional misclassification of device's activity as *malicious*. In such cases, our solution does not block network access completely but only restricts it (based on user preference) until the KEEPER SERVICE updates the classification model to incorporate updated network behavior as *benign* for given type of devices. This feature allows us to track the software update history for given device, useful to identify (and block) possible set of vulnerabilities identified for given firmware version. However, IOT-KEEPER cannot identify any minor upgrades e.g. software patches, which may not result in change in network activity.

IX. CONCLUSION

This paper presents IOT-KEEPER, a two-tier platform that is able to detect suspicious device-to-device communications in IoT-device-dominant edge networks. IOT-KEEPER predicts whether given network activity is suspicious or not, using a sparse rule base generated by clustering performed on network monitoring data. Devices that exhibit suspicious behavior are isolated from communicating with other device using AONs. IOT-KEEPER platform adopts lightweight design and can be deployed using low-cost programmable devices (KEEPER GATEWAY) such that the hub performs real-time traffic classification and security policy enforcement locally, while the model training is done using a remote service (KEEPER SERVICE). Our evaluation using live testbed shows that IOT-KEEPER achieves a high prediction accuracy (>0.985) without incurring significant overhead on latency experienced by user and it does not require sophisticated hardware or modifications on existing IoT and other devices.

REFERENCES

[1] H. Ning *et al.*, "Cyberentity security in the internet of things," *Computer*, vol. 46, no. 4, pp. 46–53, April 2013.

[2] L. Constantin, "Hackers found 47 new vulnerabilities in 23 iot devices at def con," http://www.csoonline.com/article/3119765/security/hackers-found-47-new-vulnerabilities-in-23-iot-devices-at-def-con.html, IDG, [Accessed: 2017-05-07].

[3] M. Stahlberg, "Smart homes: Oppourtunities and risks," F-Secure Corporation, Helsinki, Finland, Tech. Rep. MSU-CSE-06-2, January 2015. [Online]. Available: http://www.internetofthings.fi/extras/IoTMagazine2015.pdf

[4] "Senrio. 400,000 publicly available iot devices vulnerable to single flaw," http://blog.senr.io/blog/400000-publicly-available-iot-devices-vulnerable-to-single-flaw/, Senrio, [Accessed: 2017-05-05].

[5] D. Pauli, "414,949 d-link cameras, iot devices can be hijacked over the net," https://www.theregister.co.uk/2016/07/08/414949_dlink_cameras_iot_devices_can_be_hijacked_over_the_net/, The Register, [Accessed: 2017-05-07].

[6] M. Patton *et al.*, "Uninvited connections: A study of vulnerable devices on the internet of things (iot)," in *2014 IEEE Joint Intelligence and Security Informatics Conference*, Sept 2014, pp. 232–235.

[7] M. Wall, "How 'the invisible network' poses a major security threat," http://www.bbc.com/news/business-41252203, BBC News, [Accessed: 2017-09-24].

[8] M. B. Barcena and C. Wueest, "Insecurity in the internet of things," *Security Response, Symantec*, 2015.

[9] L. Zeltser, "Common vulnerabilities and exposures," https://cve.mitre.org/, [Accessed: 2017-05-07].

[10] Mitre, "Common weakness enumeration," https://cwe.mitre.org/, [Accessed: 2017-06-24].

[11] N. Feamster, "Outsourcing home network security," in *Proceedings of the 2010 ACM SIGCOMM Workshop on Home Networks*, ser. HomeNets '10. New York, NY, USA: ACM, 2010, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/1851307.1851317

[12] M. Miettinen *et al.*, "IoT Sentinel Demo: Automated device-type identification for security enforcement in IoT," in *Proc. 37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017)*. IEEE, Jun. 2017.

[13] "Malware sample sources for researchers," https://zeltser.com/malware-sample-sources/, [Accessed: 2017-05-07].

[14] "Kdd cup 1999 data," http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, University of California, Irvine, [Accessed: 2016-07-18].

[15] A. Bohara *et al.*, "Intrusion detection in enterprise systems by combining and clustering diverse monitor data," in *Proceedings of the Symposium and Bootcamp on the Science of Security*, ser. HotSos '16. New York, NY, USA: ACM, 2016, pp. 7–16. [Online]. Available: http://doi.acm.org/10.1145/2898375.2898400

[16] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94. San Francisco, CA, USA: Morgan Kaufmann publishers Inc., 1994, pp. 487–499. [Online]. Available: http://dl.acm.org/citation.cfm?id=645920.672836

[17] C. Borgelt, "An implementation of the fp-growth algorithm," in *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, ser. OSDM '05. New York, NY, USA: ACM, 2005, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/1133905.1133907

[18] K. Zhou *et al.*, "Fuzziness parameter selection in fuzzy c-means: The perspective of cluster validation," *Science China Information Sciences*, vol. 57, no. 11, pp. 1–8, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11432-014-5146-0

[19] E. Trauwaert, "On the meaning of dunn's partition coefficient for fuzzy clusters," *Fuzzy Sets and Systems*, vol. 25, no. 2, pp. 217 – 242, 1988. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0165011488901893

[20] S. Aranganayagi and K. Thangavel, "Clustering categorical data using silhouette coefficient as a relocating measure," in *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*, vol. 2, Dec 2007, pp. 13–17.

[21] D. M. Mendez *et al.*, "Internet of things: Survey on security and privacy," *CoRR*, vol. abs/1707.01879, 2017. [Online]. Available: http://arxiv.org/abs/1707.01879

[22] N. Apthorpe *et al.*, "Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic," *CoRR*, vol. abs/1708.05044, 2017. [Online]. Available: http://arxiv.org/abs/1708.05044

[23] M. Haus *et al.*, "Security and privacy in device-to-device (d2d) communication: A review," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 1054–1079, Secondquarter 2017.

[24] Symantec, "Smart home security and the internet of things: The future is here," https://us.norton.com/internetsecurity-iot-smart-home-security-core.html, Symantec, [Accessed: 2017-12-10].

[25] K. Lab, "Kaspersky iot scanner: How to keep your home network and its smart devices safe," https://www.kaspersky.com/blog/kaspersky-iot-scanner/18449/, Kaspersky, [Accessed: 2017-12-10].

[26] M. Tavallaee *et al.*, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, ser. CISDA'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 53–58. [Online]. Available: http://dl.acm.org/citation.cfm?id=1736481.1736489

[27] N. V. Chawla *et al.*, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622407.1622416

[28] P. Martin, "Using your new raspberry pi 3 as a wifi access point with hostapd," https://frillip.com/using-your-raspberry-pi-3-as-a-wifi-access-point-with-hostapd/, Firllip's Blog, [Accessed: 2017-07-22].

[29] A. M. Bronstein *et al.*, "Generalized multidimensional scaling: A framework for isometry-invariant partial surface matching," *Proceedings of the National Academy of Sciences*, vol. 103, no. 5, pp. 1168–1172, 2006. [Online]. Available: http://www.pnas.org/content/103/5/1168.abstract

[30] Majestic, "The majestic million," https://majestic.com/reports/majestic-million, Majestic, [Accessed: 2016-07-28].

[31] S. Press, "Western digital® breaks boundaries with world's highest-capacity microsd™ card," https://www.sandisk.com/about/media-center/press-releases/2017/western-digital-breaks-boundaries-with-worlds-highest-capacity-microsd-card, SanDisk, [Accessed: 2016-09-14].

[32] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 56–76, Fourth 2008.

[33] D. Bekerman *et al.*, "Unknown malware detection using network traffic classification," in *2015 IEEE Conference on Communications and Network Security (CNS)*, Sept 2015, pp. 134–142.

[34] V. Jeyakumar *et al.*, "Data driven data center network security," in *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, ser. IWSPA '16. New York, NY, USA: ACM, 2016, pp. 48–48. [Online]. Available: http://doi.acm.org/10.1145/2875475.2875490

[35] W. Lee *et al.*, "Adaptive intrusion detection: A data mining approach," *Artificial Intelligence Review*, vol. 14, no. 6, pp. 533–567, 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1006624031083

[36] S. Akbar *et al.*, "Improving network security using machine learning techniques," in *2012 IEEE International Conference on Computational Intelligence and Computing Research*, Dec 2012, pp. 1–5.

[37] T. Shon *et al.*, *SVM Approach with a Genetic Algorithm for Network Intrusion Detection*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 224–233. [Online]. Available: http://dx.doi.org/10.1007/11569596_25

[38] R. Shanmugavadivu and N. Nagarajan, "Network intrusion detection system using fuzzy logic," *Indian Journal of Computer Science and Engineering (IJCSE)*, vol. 2, no. 1, pp. 101–111, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.300.7185&rep=rep1&type=pdf

[39] W. T. Strayer *et al.*, *Botnet Detection Based on Network Behavior*. Boston, MA: Springer US, 2008, pp. 1–24. [Online]. Available: https://doi.org/10.1007/978-0-387-68768-1_1

[40] W. Lu *et al.*, "Automatic discovery of botnet communities on large-scale communication networks," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ser. ASIACCS '09. New York, NY, USA: ACM, 2009, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/1533057.1533062

[41] G. Gu *et al.*, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th Conference on Security Symposium*, ser. SS'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 139–154. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496721

[42] J. Sherry *et al.*, "Making middleboxes someone else's problem: Network processing as a cloud service," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 13–24, Aug. 2012. [Online]. Available: http://doi.acm.org/10.1145/2377677.2377680

[43] J. Granjal *et al.*, "Security for the internet of things: A survey of existing protocols and open research issues," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1294–1312, thirdquarter 2015.

[44] S. Raza *et al.*, "SVELTE: Real-time intrusion detection in the internet of things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2661 – 2674, 2013.

[45] T. Yu *et al.*, "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*, ser. HotNets-XIV. New York, NY, USA: ACM, 2015, pp. 5:1–5:7. [Online]. Available: http://doi.acm.org/10.1145/2834050.2834095

[46] I. Hafeez *et al.*, "Poster – ioturva: Securing device-to-device communications for iot," in *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, ser. MobiCom

'17. New York, NY, USA: ACM, 2017. [Online]. Available: https://doi.org/10.1145/3117811.3131262

[47] V. Brik *et al.*, "Wireless device identification with radiometric signatures," in *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, ser. MobiCom '08. New York, NY, USA: ACM, 2008, pp. 116–127. [Online]. Available: http://doi.acm.org/10.1145/1409944.1409959

[48] P. N. Mahalle *et al.*, "Object classification based context management for identity management in internet of things," *International Journal of Computer Applications*, vol. 63, no. 12, pp. 1–6, February 2013, full text available.

[49] M. Miettinen *et al.*, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2177–2184.

[50] Y. Meidan *et al.*, "Profiliot: A machine learning approach for iot device identification based on network traffic analysis," in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17. New York, NY, USA: ACM, 2017, pp. 506–509. [Online]. Available: http://doi.acm.org/10.1145/3019612.3019878

[51] I. Hafeez *et al.*, "Securebox: Toward safer and smarter iot networks," in *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, ser. CAN '16. New York, NY, USA: ACM, 2016, pp. 55–60. [Online]. Available: http://doi.acm.org/10.1145/3010079.3012014

[52] ACRIS, "IoT devices setup captures (IoT Sentinel experiments)," https://research.aalto.fi/files/11504588/captures_IoT_Sentinel.zip.

TABLE XI

LIST OF DEVICES USED IN SETTING UP THE TESTBED FOR DATA COLLECTION AND SYSTEM EVALUATION PURPOSES. CORE-I7 MACHINES WITH 32GB MEMORY WERE USED FOR HOSTING VIRTUAL MACHINES (SERVING AS ATTACK AND TARGET NODES IN LOCAL AND REMOTE NETWORKS) AND DEPLOYING KEEPER SERVICE. NEARLY ALL OF THESE DEVICES SUPPORT WI-FI CONNECTIVITY. IN ADDITION TO THESE DEVICES, WE ALSO USE TRACES FROM IN IOT-SENTINEL DATASET [49], [52]

Device Name	OS	Connectivity			
		WiFi	Ethernet	Bluetooth	NFC
Nexus 5	Android 6.0	✓	✗	✓	✓
Nexus 6p	Android 7.1	✓	✗	✓	✓
Nexus 6p	Android 7.1	✓	✗	✓	✓
Samsung Galaxy S4	Android 4.4	✓	✗	✓	✓
Samsung Galaxy S4	Android 4.4	✓	✗	✓	✓
Samsung Galaxy S5	Android 5.1.1	✓	✗	✓	✓
Samsung Galaxy S5	Android 5.1.1	✓	✗	✓	✓
Samsung Galaxy S6	Android 6.0	✓	✗	✓	✓
OnePlus3T	Android 7.0	✓	✗	✓	✓
Samsung Galaxy S3	Android 4.4	✓	✗	✓	✓
Lenovo Tab3	Android 4.4	✓	✗	✓	✗
Google Nexus 7	Android 6.0	✓	✗	✓	✓
Samsung Tab 8	Android 5.1.1	✓	✗	✓	✗
Samsung Tab 10	Android 4.4	✓	✗	✓	✗
IPhone 6 plus	iOS 10	✓	✗	✓	✗
IPhone 6s	iOS 10	✓	✗	✓	✗
IPhone 4s	iOS 9	✓	✗	✓	✗
IPad Air 2	iOS 10	✓	✗	✓	✗
IPad Air	iOS 10	✓	✗	✓	✗
IPad 4	iOS 10	✓	✗	✓	✗
MacBook Pro 15”	MacOS 10.12	✓	✓	✓	✗
MacBook Pro 15”	MacOS 10.12	✓	✓	✓	✗
Lenovo ThinkPad	Windows 10	✓	✓	✓	✗
HP Pavilion	Windows 10	✓	✓	✓	✗
Lenovo ThinkPad	Ubuntu 16.04	✓	✓	✓	✗
HP Pavilion	Ubuntu 16.04	✓	✓	✓	✗
HP ultrabook	Ubuntu 16.04	✓	✓	✓	✗
Dell	Ubuntu 14.04	✓	✓	✓	✗
LG UH58	WebOS 3.0	✓	✓	✗	✗
Samsung 55UE6000	Tizen OS	✓	✓	✗	✗
Samsung Gear S2	Tizen OS	✓	✗	✓	✓
LG Watch	Android wear 2	✓	✗	✓	✓
2× Raspberry PI 2	Kali Linux	✓	✓	✗	✗
	Ubuntu 16.04	✓	✓	✗	✗
2× Core i7 machines	Ubuntu 14.04	✓	✓	✗	✗
	Windows 10	✓	✓	✗	✗