IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 8 June 2024, accepted 25 June 2024, date of publication 27 June 2024, date of current version 5 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3420238

RESEARCH ARTICLE

Identifying Anomaly in IoT Traffic Flow With Locality Sensitive Hashes

BATYR CHARYYEV[1] **AND MEHMET HADI GUNES**[2]
[1]Computer Science and Engineering Department, University of Nevada, Reno, NV 89557, USA
[2]Akamai, Boston, MA 02142, USA

Corresponding author: Batyr Charyyev (bcharyyev@unr.edu)

ABSTRACT Internet of Things (IoT) devices introduce new vulnerabilities to the network. These devices are relatively cheap, have simple design yet they can collect private user data, and be employed as botnets to conduct large-scale attacks. In general, IoT devices have a limited set of functionalities. Thus, the network administrator can formulate the expected traffic patterns of the devices and employ the network traffic to detect malicious activities. Existing systems to detect anomaly in IoT traffic mainly use machine learning. Thus, they require tuning the parameters of models and selecting/extracting a representative set of features from the network traffic data. In this paper, we introduce a novel approach *Locality Sensitive Anomaly Detection and Identification* (LSADI) to detect anomaly in IoT network traffic based on the locality-sensitive hash of the traffic flow. The proposed approach does not require feature selection/extraction from the data and does not have complex set of parameters that need to be tuned. Evaluation with three datasets containing 25 attacks shows that LSADI can detect and identify the type of anomalous flows with an accuracy above 90% on average and performs equally well compared to the state-of-the-art machine learning-based methods.

INDEX TERMS Internet of Things, networking, traffic fingerprinting.

I. INTRODUCTION

IoT devices introduce new vulnerabilities in a network as there is a myriad of IoT devices with different communication patterns. These devices have limited functionality and often are unpatched even when software updates are available. Limited resources also prevent IoT devices from having a sophisticated security analysis on the device. Attackers can launch various attacks against vulnerable IoT devices in the form of brute force, and TCP SYN/UDP flooding [1], or use these devices as botnets to launch DoS attacks by infecting them with malware [2].

Several studies detect anomalous patterns in network activity of the IoT devices using one-class machine learning (ML) algorithms [1], [3] and autoencoders [2], [4]. Since ML algorithms detect outliers in the network traffic, they need to be trained on benign traffic of the IoT device and then use the trained model to detect any malicious activity as an outlier. Similarly, autoencoders can be trained on benign network traffic to learn IoT communication behavior through neural networks [2]. In the compression process of the autoencoder, the neural network learns the relation among provided features. Even though autoencoders can reconstruct normal traffic based on benign network traffic, they cannot reconstruct abnormal traffic. While ML-based approaches are promising, they require feature selection and extraction from the data, which is resource-intensive and sometimes undesirable due to privacy concerns [5].

In general, machine learning-based anomaly detection systems have two phases, training and online detection. The training phase consists of data pre-processing, ML model selection, and training of the model. The data pre-processing can further be split into data collection, feature selection, and extraction of chosen features. The features for traffic analysis are the properties of individual network traffic packets such as packet size, arrival time, and direction, as well as overall statistical values of traffic such as mean and standard deviation of packet characteristics. After pre-processing the data, the machine learning model is selected and trained on the pre-labeled data. A crucial step in training the model

The associate editor coordinating the review of this manuscript and approving it for publication was Nurul I. Sarkar.

is tuning the parameters to obtain optimum performance. The model can be retrained if there is a need for an update due to environmental changes, firmware updates, availability of broader labeled data, etc. Finally, the trained model is used for online detection of anomaly in network traffic. The representativeness of the training data, selected features, machine learning model, and its parameters are some of the key factors impacting the performance of ML-based anomaly detection systems.

Several studies focus on determining the relevant set of features in the IoT network traffic for different machine learning problems such as device identification [6], [7] and anomaly detection [3], [8]. Identifying the representative set of features enables the reduction of computing resources, mitigates the curse of dimensionality, prevents the model from overfitting the training data, and ignores noisy features. Researchers have developed machine learning-based traffic fingerprinting for IoT device identification [9], [10], device event identification [11], [12], and anomaly detection and identification [4], [7], [13], [14].

IoT devices differ from personal computing devices such as laptops and tablets as IoTs are designed for performing a particular task and often generate repetitive network traffic patterns. This enables researchers to obtain the signature of IoT device traffic for device identification [15], device event identification [16], and anomaly detection in network traffic [17]. In this paper, we present a novel approach called *Locality Sensitive Anomaly Detection and Identification* (LSADI) to detect/identify the anomaly in IoT network traffic. LSADI generates the hash of benign network traffic of an IoT device using a Locality-Sensitive Hash (LSH) function. Note that the locality-sensitive hash differs from cryptographic hashes. By design, LSH is less sensitive to small changes in input in order to produce similar hash values with similar inputs. This enables LSADI to detect the anomaly in network traffic of the device by comparing the hash of the traffic flow with the hash of the benign traffic stored in the database. Furthermore, it enables the identification of the type of malicious flow (TCP SYN, UDP flooding, device infected with Mirai, etc.) by comparing the hash of the traffic flow with the hash of known malicious flows.

We evaluated LSADI on three different datasets. The first dataset contains traffic from IoT devices under different volumetric attacks such as ARP Spoofing, Ping of Death, Smurf, Fraggle, and TCP Syn/SNMP/SSDP flooding. The second and third dataset contains traffic from different IoT malware such as Mirai, IRCBot, Hajime, Kenjior, etc. We compared our approach with a one-class classifier-based approach MUD-Engine, neural network-based approach Kitsune, and with other one-class classifiers such as Local Outlier Factor, One-Class SVM, Elliptic Envelope, and Isolation Forest. Compared to previous studies [1], [2], [3], [4] that rely on one-class classifiers and autoencoders, our approach does not require feature extraction from the data and tuning the parameters of the algorithm. Furthermore, it is relatively easy to update the model if the behavior of the benign device changes due to firmware updates, and environmental/user behavior changes.

In a preliminary study [17], we explored the use of locality-sensitive hashing to detect volumetric attacks. This study differs from [17] in several aspects. First, we propose an approach to identify the attacks and design the system by integrating the attack identification feature. Second, we incorporate multiple hashing algorithms to increase the overall accuracy without significantly increasing the computational overhead. Third, we systematically evaluate and show that the best representative set of network traffic features change from device to device. Fourth, we considerably extend the evaluation of our new system with two more datasets. Finally, we extended comparison of our system with other intrusion detection systems including Kitsune [4], Gaussian Mixture Models [18], pcStream [19] and Suricata [20].

In the rest of the paper; Section II provides our motivation by discussing the importance of feature selection/parameter tuning in network traffic fingerprinting; and provides background information on locality-sensitive hashes. Section III introduces the design principles of the proposed LSADI approach; and describes the datasets used for evaluation. Section IV shows the experimental evaluation results. Section V presents related studies on anomaly detection in network traffic with one-class classifiers, autoencoders, and anomalous traffic identification. Finally, Section VI concludes the paper.

II. BACKGROUND AND MOTIVATION

In this section, we discuss the importance of feature selection and parameter tuning in traffic fingerprinting and provide background information on locality-sensitive hashing.

A. FEATURE SELECTION AND PARAMETER TUNING

Feature selection is the process of selecting the best representative features of the original data and remove the rest that could be potentially irrelevant or redundant for the problem. It mitigates the curse of dimensionality, prevents the model from overfitting or underfitting the training data, and reduces the required storage and computation resources. There exist studies focusing on feature selection for various network traffic fingerprinting problems. For instance, optimization-based approaches such as Genetic Algorithm used to select features for IoT device identification [6], and the Grey Wolf Optimization algorithm for IoT botnet problems [7]. Statistical properties are also used to select features such as Fisher's score [21], variance and Hopkins statistics [3], and selection based information gain resulting from the inclusion of the feature [8].

To show the importance of feature selection we used the IoT dataset from [1] that contains network traffic of nine IoT devices exposed to fifteen different volumetric attacks. For each device, we selected the best representative set of features for detecting the attacks on devices. To do that we split traffic of devices into flows of size one minute and collected

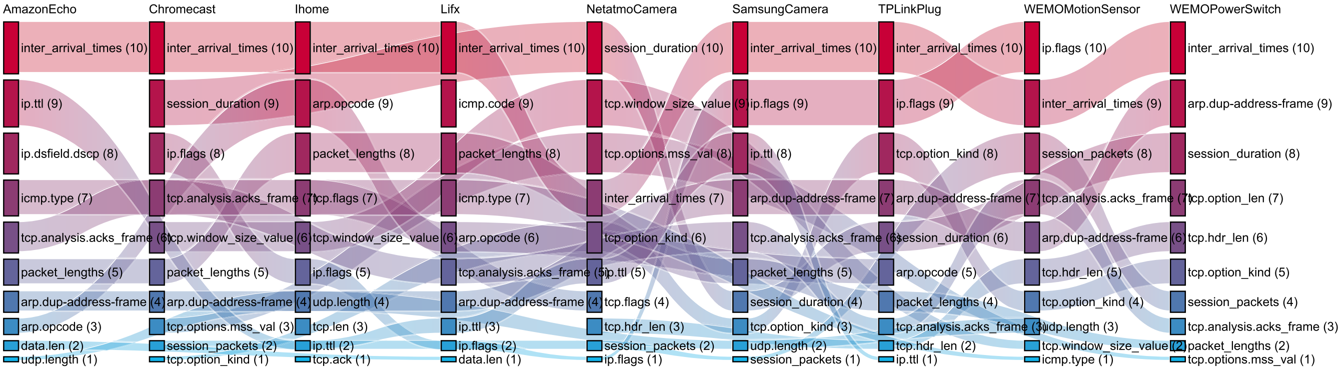


FIGURE 1. Ranking of network traffic features of sample IoT devices based on discriminative power.

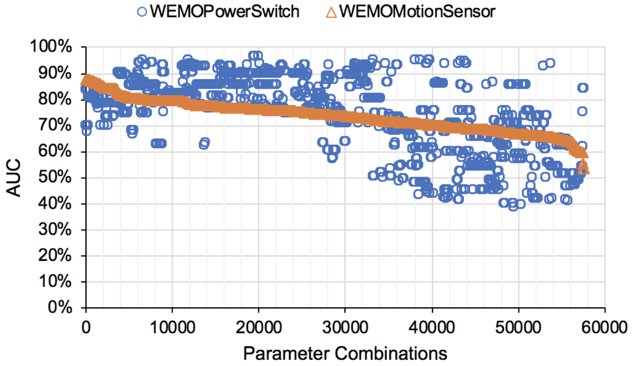


FIGURE 2. Accuracy (in terms of AUC) for local outlier factor with different parameter set in detecting anomaly in IoT traffic.

all numeric attributes (features) of each observed protocol based on the Wireshark documentation. We normalized the features and removed the ones with the same value in 99% of the samples using the variance thresholding. We also removed features that have a high correlation with 99%. Then we ranked the features for each device using the univariate feature selection and importance coefficient of features when they are fitted to classifiers.

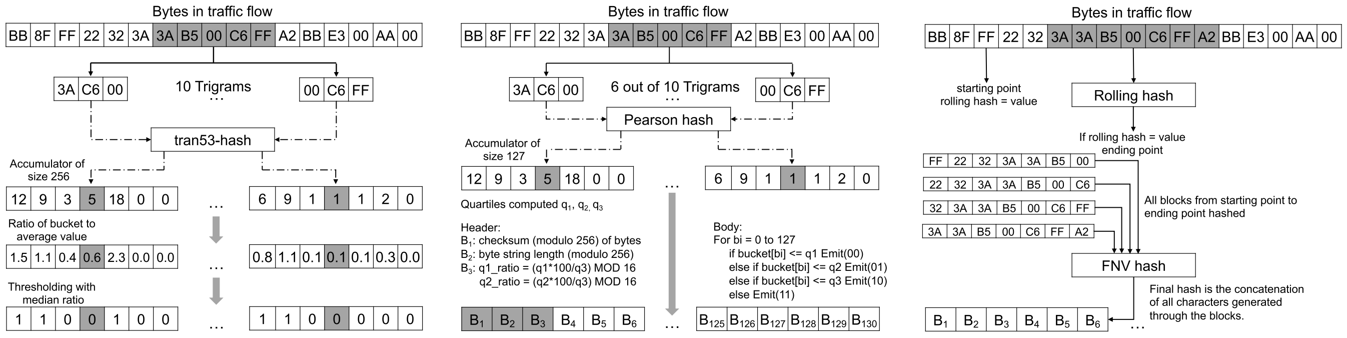
Figure 1 presents how the rank of the top ten features with the highest discriminative power changes from device to device for a sample of devices. We can observe that top features are in general related to packet timing and data size, such as inter-arrival timing of packets, average session duration, average packet length, tcp header length, etc. However, except for the inter-arrival timing of packets, there does not exist a feature that performs well for all devices. We observe that representative features differ even if devices are from the same manufacturer (Wemo Motion Sensor and Power Switch) or have a similar type (Netatmo and Samsung cameras). For these reasons, to get the best out of data, it is crucial to carefully select features. However, analyzing features for each device and training separate models for each device might be a time-consuming task requiring huge manual labor.

To achieve high performance in network traffic fingerprinting it is critical to select the right ML model and tune its parameters to the optimum. The preference for one model over another depends on the nature of the problem, the amount of available data, and the trade-off between accuracy, speed, and resource requirement. For instance, neural networks are preferred when there is abundant data, whereas simple classifiers are preferred when the data is limited. However, in both cases, the parameters of the models need to be tuned to obtain the best performance. Figure 2 presents the accuracy of Local Outlier Factor based malicious IoT traffic detection. Accuracy is in terms of area under curve (AUC) and computed for different parameter combinations. Even though both IoT devices belong to the same brand WEMO the optimum parameters of the model differ. This means that models need to be tuned to the optimum for each device. Also, changes to the configuration, environment, firmware update, user behavior, and even deployment region may require retraining of the ML model. For instance, [22] observed that the performance of the IoT device identification system slightly changes when the system is trained on devices bought in the US and tested on devices bought in the UK, even when the devices belong to the same brand and manufacturer. Similar behavior was observed in [16] regarding device event identification. To this end, researchers have compared different machine learning models for anomaly detection in IoT network traffic [4], [7], [13] and identification of the type of the IoT attack [13], [14].

B. LOCALITY-SENSITIVE HASHING

Cryptographic hash functions such as MD5 [23] or SHA [24] generate completely different hashes if there exists a small difference in input values. Locality-sensitive hashes are different and produce similar hash values for similar inputs. In this study, we use three different locality-sensitive hash functions, namely nilsimsa [25], tlsh [26], and ssdeep [27], for anomaly detection and identification in IoT traffic.

Nilsimsa: operates by using a window size of 5 characters that slide along the input file, one character a time. (shown in Figure 3(a)). Trigrams are generated when a new character


Bytes in traffic flow
BB 8F FF 22 32 3A 3A B5 00 C6 FF A2 BB E3 00 AA 00
3A C6 00
10 Trigrams
00 C6 FF
tran53-hash
Accumulator of size 256
12 9 3 5 18 0 0
6 9 1 1 1 2 0
Ratio of bucket to average value
1.5 1.1 0.4 0.6 2.3 0.0 0.0
0.8 1.1 0.1 0.1 0.1 0.3 0.0
Thresholding with median ratio
1 1 0 0 1 0 0
1 1 0 0 0 0 0
6 out of 10 Trigrams
Pearson hash
Accumulator of size 127
12 9 3 5 18 0 0
6 9 1 1 1 2 0
Quartiles computed q1, q2, q3
Header:
B1: checksum (modulo 256) of bytes
B2: byte string length (modulo 256)
B3: q1_ratio = (q1*100/q3) MOD 16
q2_ratio = (q2*100/q3) MOD 16
Body:
For bi = 0 to 127
if bucket[bi] <= q1 Emit(00)
else if bucket[bi] <= q2 Emit(01)
else if bucket[bi] <= q3 Emit(10)
else Emit(11)
B1 B2 B3 B4 B5 B6
B125 B126 B127 B128 B129 B130
starting point rolling hash = value
Rolling hash
If rolling hash = value ending point
FF 22 32 3A 3A B5 00
22 32 3A 3A B5 00 C6
32 3A 3A B5 00 C6 FF
3A 3A B5 00 C6 FF A2
All blocks from starting point to ending point hashed
FNV hash
Final hash is the concatenation of all characters generated through the blocks.
B1 B2 B3 B4 B5 B6

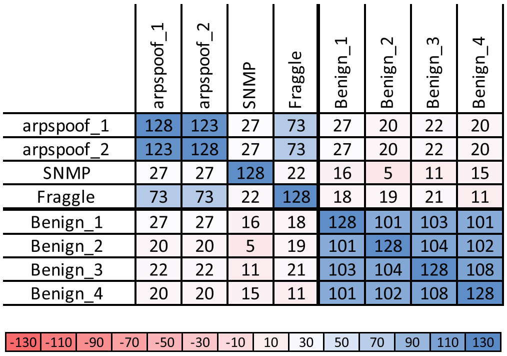
(a) Nilsimsa (b) TLSH (c) SSDEEP

FIGURE 3. Operation of locality-sensitive hash functions.

enters the window and is passed to the *tran53* hash function which generates a value *i* between 0 and 255. Then i^{th} counter of the *accumulator* (i.e. array of integers of size 256) is increased by 1. After iterating over the entire input file, the accumulator will hold frequencies of the trigrams that have been observed in the input. Then, the ratio of the value in a bucket to the average values of the buckets is assigned for each bucket in the accumulator. Finally, if the i^{th} ratio is greater than the median of the ratios, then the i^{th} bit of the Nilsimsa code will be set to 1, and otherwise set to 0. The output of the Nilsimsa is a 32-byte hash value. The similarity score of two hashes ranges from -128 (i.e., completely different) to 128 (i.e., completely similar) and is calculated by subtracting 128 from the number of similar bits in two hashes.

TLSH: processes input by a sliding window of size of 5 bytes, moving one byte at a time. (as shown in Figure 3(b)). In each step, TLSH selects 6 out of 10 possible trigrams generated from the 5 bytes. For each trigram, a Pearson hash is computed to populate a 128-bit array of bucket counts. After iterating through the input, the quartile points of the array are calculated to generate an output of 35 bytes. The first 3 bytes of the output are referred to as the header and constructed based on the quartile points, the object size, and a checksum. The remaining 32 bytes are referred to as the body and generated by comparing each array position to the quartile points. The distance for the header is computed from input size and quartile ratios, and the distance for the body is calculated with the hamming distance between the digest bodies. If the sum of two distances is low, then there is a high similarity between two TLSH hashes.

SSDEEP: creates variable size blocks using a rolling hash algorithm to set block boundaries. (as shown in Figure 3(c)). It uses a sliding window of 7 bytes that moves one byte at a time and identifies starting and ending points whenever the rolling hash produces a specific output based on the current bytes in the window. Generated blocks are hashed using a cryptographic hash function FNV, and the 6 least significant bits of each hash is encoded using the Base64 encoding. SSDEEP returns the hash as a concatenation of all characters generated through the blocks. The similarity score of two SSDEEP digests ranges from 0 (i.e., completely different) to 100 (i.e., same) and computed by counting the minimum number of operations required to transform one digest into another using weighted operations such as insertion, deletion, substitution of a single character, and transpositions of two adjacent characters.



	arpspoof_1	arpspoof_2	SNMP	Fraggle	Benign_1	Benign_2	Benign_3	Benign_4
arpspoof_1	128	123	27	73	27	20	22	20
arpspoof_2	123	128	27	73	27	20	22	20
SNMP	27	27	128	22	16	5	11	15
Fraggle	73	73	22	128	18	19	21	11
Benign_1	27	27	16	18	128	101	103	101
Benign_2	20	20	5	19	101	128	104	102
Benign_3	22	22	11	21	103	104	128	108
Benign_4	20	20	15	11	101	102	108	128

-130 -110 -90 -70 -50 -30 -10 10 30 50 70 90 110 130

FIGURE 4. Nilsimsa hash similarity score for sample traffic flows.

C. MOTIVATION

Our work was built on our previous studies on locality-sensitive hashing [17], [28]. The similarity of locality-sensitive hashes generated from two benign traffic flows of a device is expected to be higher compared to the similarity score of the benign and malicious traffic from that device. Figure 4 shows the Nilsimsa similarity score as a color-coded matrix for eight different traffic flows containing both benign and malicious flows such as ARP spoofing, SNMP flooding, and Fraggle (UDP flooding). The similarity score is mostly higher between two benign traffic and between malicious traffic of the same kind. This enables LSADI to detect anomalies in IoT network traffic given signatures of both benign and malicious traffic. Furthermore, it allows differentiating signatures of anomalous flows from each other enabling the identification of the malware if the system already has the signature of malicious flow of the same kind.

III. NETWORK TRAFFIC ANOMALY DETECTION AND IDENTIFICATION

In this section, we present the LSADI (Locality Sensitive Anomaly Detection and Identification) system to detect and

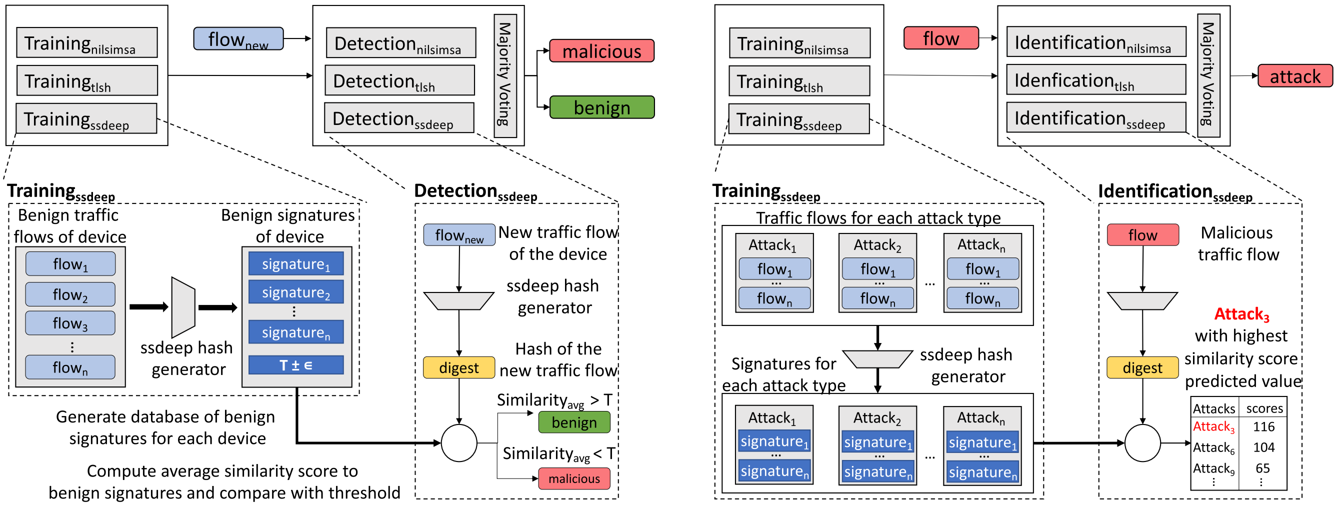


FIGURE 5. **Locality sensitive anomaly detection and identification system.**

identify the type of anomaly observed in network traffic and provide details of the datasets used in our evaluation.

A. LOCALITY SENSITIVE ANOMALY DETECTION AND IDENTIFICATION

Most of the IoT devices perform simple tasks, with well-defined traffic patterns [29]. Thus, it is possible to create a set of signatures from the benign traffic flow of the device and use those signatures to detect anomalies in the traffic. However, network traffic of the device can vary based on factors such as firmware version, and configuration parameters of the device. This should be reflected in the signature mechanism and the signature of the different benign traffic flows must differ a little. Since locality-sensitive hash functions generate similar hash values for similar inputs, they can be used to generate signatures of IoT traffic flows.

LSADI utilizes three locality-sensitive hash algorithms namely Nilsimsa [25], TLSH [26], and SSDEEP [27] to generate a set of hashes (i.e. signatures) from the benign traffic flows of the IoT device and computes the threshold value T for each hash function. Note that as traffic flow, we employ raw packet data generated by the devices. The T threshold value is computed by simply measuring the average similarity of the hashes generated from the benign traffic flows. Signatures from benign traffic and the computed threshold value are stored in the database. Similarly, signatures of previously known malicious flows are stored in the database. Known malicious flows may include the traffic generated from flooding attacks (e.g., SSDP, SNMP, and UDP) or botnet attacks (e.g., Mirai, Bashlite). Note that this operation is performed only once, thus, it does not incur future overheads.

The overall functionality of LSADI is summarized in Figure 5. It first generates the signatures of the device from its benign traffic, computes the threshold value T from the average similarity of the benign signatures, and stores them in a database. The signatures of the known attacks are also stored in the database. When a monitored device generates a new traffic flow, LSADI computes the locality-sensitive hash (i.e. digest) of the flow, compares it with the benign signatures of that device, and computes the average similarity score. If the average similarity score is below the T threshold value, the flow is labeled as malicious. This is repeated for all three locality-sensitive hash functions (i.e., nilsimsa, tlsh, and ssdeep) and a final label is determined by majority voting. If the traffic is labeled as malicious, the digest of the flow is compared to signatures of the already known attacks, and the label of the attack is determined from the highest average similarity value. The final identification result is based on the majority voting of *nilsimsa*, *tlsh*, and *ssdeep*. Detecting and identifying abnormal traffic patterns allows network administrators to take necessary measures, such as blocking specific network traffic, throttling the traffic rate, or rejecting flows originating from particular sources (e.g., internal or external networks).

While *nilsimsa* performs best individually, we utilize multiple hash functions as some attacks are missed by *nilsimsa* but detected by other hashing approaches. For instance, *nilsimsa* misses low rate *TCP SYN* reflection and *Ping of Death* attacks whereas *tlsh* and *ssdeep* achieve high detection accuracy. Similarly, for *SMURF* and *Fraggle* attacks, *nilsimsa* and *tlsh* achieve high accuracy while *ssdeep* performs poorly. We observed that employing all three hashing methods with a majority voting increases accuracy by 5% on average, and up to 27% for *Fraggle*. While *nilsimsa* is the best-performing hash function, it has the highest computation overhead. Hence, integrating *tlsh* and *ssdeep* improves the overall accuracy while incurring considerably less overhead than *nilsimsa*.

B. DATASETS

The summary of the datasets used in our evaluation is provided in Table 1. The first dataset used in our evaluation collects network traffic of real IoT malware running on

TABLE 1. Summary of evaluated datasets.

Dataset	*Source*	*Attacks*	*Description*
Aposemat data	Parmisano et al. [30]	5	Traffic from Raspberry Pi infected with IoT malware
Kitsune data	Yisroel et al. [4]	5	Nine IoT devices infected with Mirai
Volumetric attacks	Hamza et al. [1]	15	Volumetric attacks launched to nine IoT devices as direct and reflection attacks

Raspberry Pi. IoT malware includes Mirai, IRCBot, Hajime, Kenjior, and Okiru [30]. Hajime (Japanese for "beginning") targets devices with open Telnet ports and default usernames and passwords. It uses hardcoded addresses for its command and control (C&C) server, and is built on a peer-to-peer network. Hajime blocks access to ports 23, 7547, 5555, and 5358. Attacker can open a shell script to any infected machine in the network at any time. Okiru is a variant of Mirai malware that can utilize IoT devices as botnets. It is known as the first malicious code to specifically target Argonaut RISC Core (ARC) processors. While ARC processors are not as common as Intel or ARM, they are employed in a wide area of systems such as smart energy hubs, intelligent appliances, wearable fitness, and medical devices. Kenjiro is a variant of a Hakai malware designed similar to Izuku. The difference between Kenjiro and Izuku is that Kenjiro changes its UDP flood algorithm to add some randomization to the buffer. Kenjiro mainly aims D-Link, Huawei, and Realtek routers. IRCBot is built with the help of a Shellbot targeting IoT devices and Linux servers. It also exploits authentication bypass vulnerability in D-Link routers. IRCBot modifies the DNS server setting in the configuration of the IoT devices to redirect the traffic from the infected device to malicious servers controlled by the attacker. In the rest of the paper, we refer to this dataset as *Aposemat data*.

The second dataset contains network traffic from nine IoT devices infected with real Mirai malware, along with SSDP_Flood, SSL_Renegotiation, SYN_DoS, and Video _Injection attacks [4]. Mirai malware is notoriously known for utilizing IoT devices as botnets in DDoS attacks [31]. Devices infected with this malware continuously scan the Internet for vulnerable devices and try to infect them using factory default credentials. The dataset also contains traffic from IP cameras exposed to attacks such as Video_Injection, SSDP_Flood, SYN_DoS, and SSL_Renegotiation. In the Video_Injection attack, an adversary injects a recorded video clip into the live video stream of IP cameras. Similarly, SSDP_Flood, SYN_DOS, and SSL_Renegotiation attacks are conducted on a network of IP cameras. An attacker tries to disable video streams by causing cameras to spam DVR with UPnP advertisements as well as SSL renegotiations. To conduct these attacks, authors use tools such as Saddam, THC, Hping3, and Video Jack. In the rest of the paper, we refer to this dataset as *Kitsune data*.

The final dataset contains 15 different volumetric attacks launched towards 9 IoT devices [1]. Monitoring volumetric attacks is essential because they can be overlooked by access control lists based on the Manufacturer Usage Description (MUD) profile of the device. The dataset contains various attacks, including ARP Spoofing, TCP SYN/SNMP/SSDP Flooding, Fraggle (UDP Flooding), Smurf, and Ping of Death. Both attackers and targets can be located on the Internet (denoted as W) or the local network (denoted as L).

Additionally, attacks can be categorized as direct or reflection. Direct attacks are the ones directly launched to IoT devices (i.e., L-D or W-D). For instance, "*ARP Spoofing (L-D)*" is an attack launched to the IoT device (i.e., D) from the local network (i.e., L). Reflection attacks (i.e., L-D-L, W-D-W, and L-D-W) are the ones that reflect off of the IoT device and target a destination. For instance, "*SNMP (W-D-W)*" is an attack coming from the Internet (i.e., W) to the IoT device (i.e., D), then the device reflects it and targets a system on the Internet (i.e., W). In the reflection attacks, the adversary sends fake requests by sticking the victim's IP address in the source IP address field. After the reflector IoT device, in this case, receives the packet the device responds to the packet using the source IP address of the packet. The victim system ends up receiving a large volume of response packets it never had requested. While an IoT device is a victim in the direct attacks it is utilized as an intermediate tool (reflector) in reflection attacks to target the servers. In the rest of the paper, we refer to this dataset as *Volumetric attacks*.

IV. EVALUATION

In the evaluation of LSADI, we divided traffic data into flows of 1 minute and used 100 random benign flows to generate the signatures of each device. These signatures are then used to detect anomalies in IoT network traffic. To identify the type of anomalous traffic flow, we used 100 random signatures for each kind of attack. Anomaly detection in network traffic is evaluated in terms of true-positive rate (i.e., $TPR = TP/(TP + FN)$) and false-positive rate (i.e., $FPR = FP/(FP + TN)$). Identification of the type of anomalous traffic is evaluated in terms of precision (i.e., $TP/(TP+FP)$), recall (i.e., $TP/(TP + FN)$), and f1-score (i.e., $2/(1/precision + 1/recall)$), where TP, TN, FP, and FN stand for true positive, true negative, false positive and false negative, respectively.

The overall performance of the LSADI for each dataset is summarized in Table 2. We present results for one-minute traffic of the device with an average false-positive rate of 5%. We selected these values for direct comparison with the [1]. Overall we observe that LSADI can achieve an accuracy above 90% in both detecting and identifying the type of malicious traffic flows, except for the Aposemat data. As detailed below, we observed that in the Aposemat data, LSADI performs poorly in detecting command & control (C&C) communication of some malware.

A. EVALUATION ON ANOMALY DETECTION AND IDENTIFICATION

The overall performance of the LSADI for each malware in the *Aposemat dataset* is summarized in Table 3. LSADI

TABLE 2. Average performance of LSADI on all three datasets.

Dataset	*Detection*		*Identification*		
	TPR	FPR	Precision	Recall	F1-score
Aposemat data	88%	5%	99%	99%	99%
Kitsune data	96%	5%	94%	98%	95%
Volumetric attacks	98%	5%	99%	99%	99%

TABLE 3. Results for individual attacks on Aposemat data.

Attack	*Detection*	*Identification*	
	TPR	Precision	Recall
Mirai	*70%*	99%	99%
IRCBot	90%	100%	100%
Hajime	98%	99%	98%
Kenjiro	90%	100%	100%
Okiru	91%	100%	100%

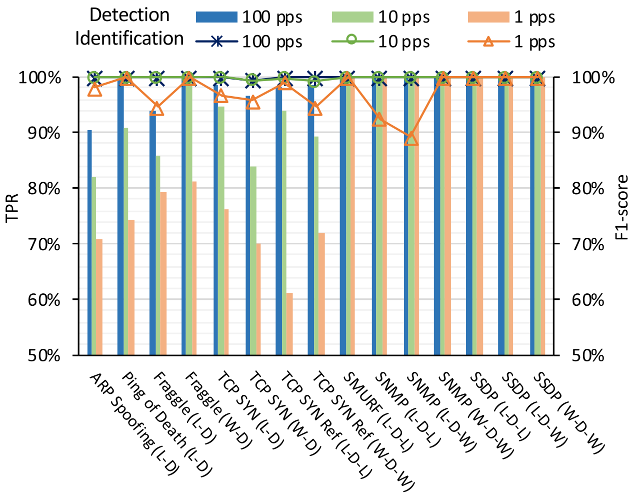


FIGURE 6. Detection and identification accuracy of LSADI for different attack rates.

achieves high detection and identification accuracy except for the cases noted with *italic* font. We observed that LSADI performs poorly in detecting traffic flow generated by C&C communications. LSADI detects malicious flows with a true-positive rate of 100% when Mirai makes a port scan or conducts a denial of service attack, however, for C&C it achieves 10% of true positive value. In terms of attack identification, some of the Hajime traffic is predicted as belonging to Mirai and some of the Mirai traffic is predicted as IRCBot and Kenjiro. The reason for this misclassification could be due to similarities in the traffic patterns and behaviors of Hajime, Mirai, IRCBot, and Kenjiro malware, leading to overlapping features that confuse the detection algorithm. Overall, on average, LSADI achieves both precision and recall above 90% for all attacks in the *Aposemat dataset*.

For the second dataset (*Kitsune data*), we observed that our system detects and identifies attacks with an accuracy of around 94-95% for all cases. The only lower results are observed in the detection of Video_Injection attacks and identification of SSL_Renegotiation and SYN_DoS attacks. In volumetric attacks, we observed that LSADI can detect and identify most of the attacks with an average accuracy of 98-99%. However, in some cases when devices generate a high volume of traffic in the benign state and the ratio of malicious traffic is lower compared to benign flow, there is a little drop in the performance of LSADI.

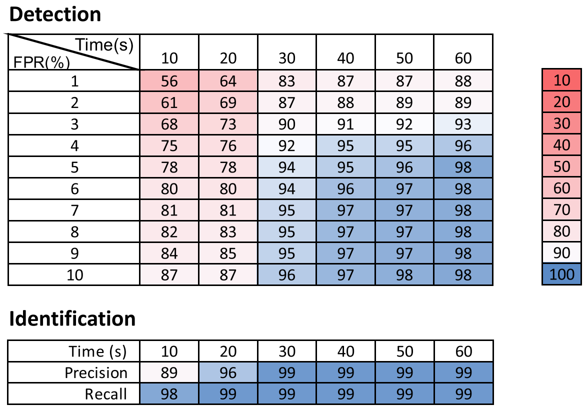
B. IMPACT OF THE ATTACK RATE

In general, devices are impacted differently by the attack rate. Devices generating high volumes of traffic are less affected by low-rate attacks, whereas simpler devices like smart plugs are more susceptible and may become even non-functional as the attack rate increases. Consequently, the performance of LSADI may vary depending on the attack rate and the traffic rate generated by the device. For instance, in *Volumetric attacks* for devices similar to Chromecast Ultra that have a high traffic rate, it becomes difficult to detect low-rate attacks (e.g., ARP Spoofing (L-D)) because there is a slight difference in the signatures of the benign and malicious flows as the volume of malicious traffic is low.

We evaluated LSADI with different attack rates, low: 1 packet-per-second (pps), medium: 10 pps, and high: 100 pps. The performance of LSADI in detecting attacks with different traffic rates is shown with a bar graph in the primary y-axis (i.e., TPR score) of Figure 6. The performance of LSADI in identifying the type of attack is shown as a line graph in the secondary y-axis (i.e., f1-score). For high-rate attacks (100 pps), LSADI mostly achieves a TPR score of around 99%, the lowest TPR (90%) observed in ARP Spoofing (L-D) attacks. For medium-rate (10 pps) attacks LSADI mostly achieves a TPR score of around 95%, the lowest TPR (80%) again observed in ARP Spoofing (L-D) attacks, and similar observation is also observed in low-rate attacks (1 pps) as well. Detailed analyses showed that LSADI performance drops for low-rate attacks on devices with high traffic rates such as the WeMo Motion Sensor, WeMo Power Switch, and Chromecast Ultra. Note that, for other devices, LSADI still can achieve a high detection accuracy of around 90% even in low-rate attacks. In terms of attack type identification, we observed that LSADI can identify high and medium-rate attacks with accuracy nearly equal to 100%. For low-rate attacks, the average identification accuracy is 95%.

C. IMPACT OF THE FPR AND WINDOW-SIZE

Tolerance to the false-positive rate (FPR) can vary across different networks. LSADI controls the FPR through the threshold value T (i.e., the average similarity of the benign signatures for the device). The similarity score of the flows lower than the T value is considered as an anomalous flow. Thus, decreasing the T will increase the FPR by being



Detection

FPR(%) \ Time(s)	10	20	30	40	50	60
1	56	64	83	87	87	88
2	61	69	87	88	89	89
3	68	73	90	91	92	93
4	75	76	92	95	95	96
5	78	78	94	95	96	98
6	80	80	94	96	97	98
7	81	81	95	97	97	98
8	82	83	95	97	97	98
9	84	85	95	97	97	98
10	87	87	96	97	98	98

10
20
30
40
50
60
70
80
90
100

Identification

Time (s)	10	20	30	40	50	60
Precision	89	96	99	99	99	99
Recall	98	99	99	99	99	99

FIGURE 7. Average detection (TPR(%)) and identification accuracy of LSADI for different time-window and FPR(%) values.

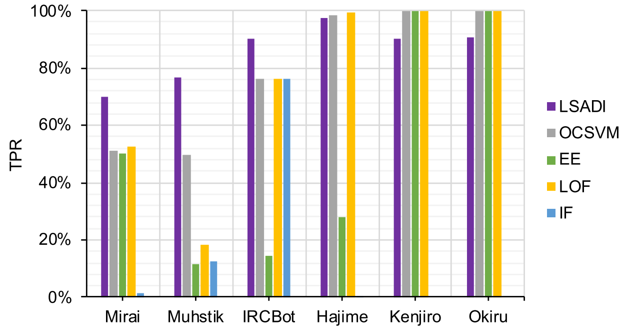


FIGURE 8. Comparison of LSADI with one-class classifiers EE, LOF, IF, OCSVM on the Aposemat dataset.

less restrictive on the similarity of the benign signatures. However, it will also decrease the TPR by labeling malicious flows as benign. Another important parameter that also impacts the TPR is the time *window size* used to detect and identify anomalies. Here *window size* represents the delta time during which traffic data is captured to detect malicious activities.

Figure 7 presents the average TPR for attack detection and average precision and recall for attack identification under different *window size* and FPR values. We can observe that, it is possible to reduce the *window size* to 30 seconds and still achieve an acceptable detection accuracy. In terms of attack identification, we can achieve both average precision and recall of around 90% with 10 seconds of network traffic. Overall, both attack detection and identification accuracy increase with increasing the *window size*. While a high FPR value also increases the detection accuracy (i.e., TPR) it may not be acceptable for some networks. Thus, to decrease the FPR value while achieving high detection accuracy, network administrators can increase the *window size* and decrease the threshold value of T. Note that increasing the *window size* has a negligible impact on the time required to generate the hash of the traffic flow, as generating the digest is a lightweight operation (discussed in overhead analysis section IV-E).

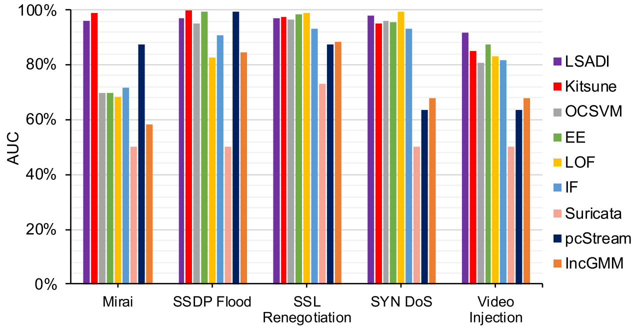


FIGURE 9. Comparison with Kitsune, EE, LOF, IF, OCSVM, and other anomaly detection systems on the Kitsune dataset.

D. COMPARISON WITH PREVIOUS STUDIES

In this section, we compare LSADI with one-class classifiers as previous studies [2], [3], [7] use these classifiers to detect anomalies in IoT traffic. We also compare LSADI with other intrusion detection systems Kitsune [4] and MUD-Engine [1].

1) COMPARISON WITH ONE-CLASS CLASSIFIERS

One-class classifiers such as one-class SVM (OCSVM), Local Outlier Factor (LOF), Isolation Forest (IF), and Elliptic Envelope (EE) are widely utilized to detect anomalies in network traffic [2], [3], [7]. These classifiers are trained on benign flows of the device and suspicious flows are recognized as an outlier. We used the implementation of the one-class classifier from the scikit learn package (`https://scikit-learn.org`) in Python. We tuned the parameters by trying all possible combinations of the parameters obtained from the scikit learn documentation. As input features to the machine learning classifiers, we used the number of packets in a flow (i.e., one minute traffic), sum, mean, and variance of the packet length, and inter-arrival time of packets – features suggested in [8] and [21] to detect anomaly in IoT network traffic.

Figure 8 shows the comparison of LSADI with one-class classifiers in detecting the malicious flows in the Aposemat data. LSADI achieves equal or better true positive value in detecting three attacks out of five. While LSADI achieves better performance on Mirai and IRCBot attacks, one-class classifiers outperform on Kenjiro and Okiru attacks. Among the machine learning models, Isolation Forest (IF) performs the worst, achieving a TPR value nearly equal to zero in four of the attacks.

2) COMPARISON WITH AUTOENCODER BASED KITSUNE

Autoencoders are also employed to detect anomalies in IoT network traffic [2], [4], [32]. Autoencoder is an artificial neural network that can be used for purposes such as dimensionality reduction, information retrieval, and image processing.

To detect anomalies in network traffic, autoencoders are first trained on the benign network traffic and then reconstruction error is used to detect anomalies in live

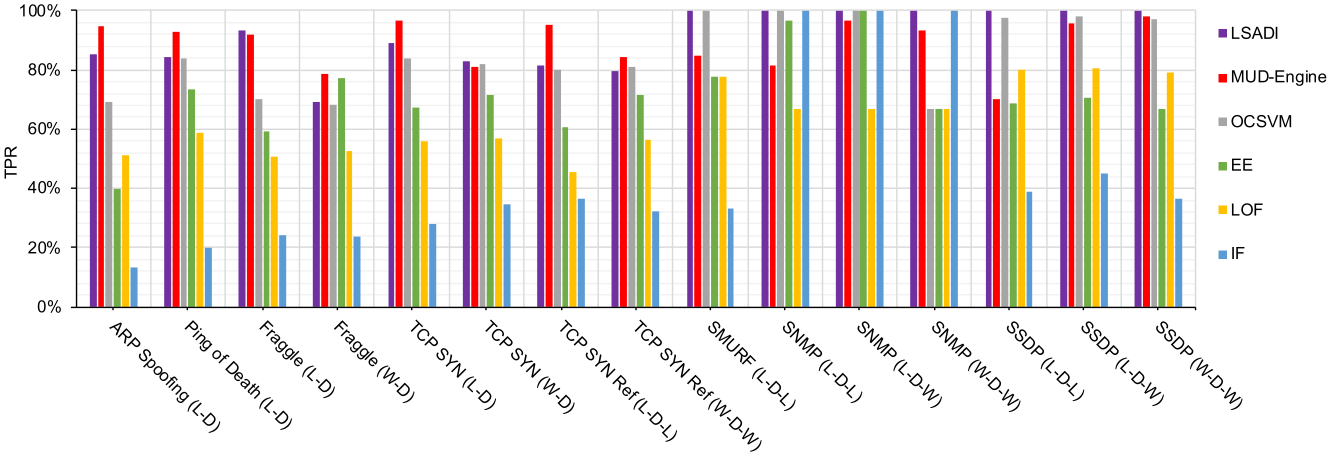


FIGURE 10. Comparison with MUD-Engine and one-class classifiers (OCSVM, EE, LOF, IF) on volumetric attacks.

flows. If a new instance of data is benign, autoencoders will give a low reconstruction error. On the other end, if traffic data belongs to malicious activity, it will have a high reconstruction error.

Figure 9 shows the comparison of the LSADI with the autoencoder based *Kitsune* intrusion detection system [4], one-class classifiers, and other intrusion/anomaly detection methods that include incremental Gaussian Mixture Models (IncGMM) [18], pcStream [19] and Suricata [20]. Incremental GMM is a statistical method based on the expectation-maximization algorithm, pcStream is a stream clustering algorithm that uses Mahalanobis distance of new instances to known clusters to determine whether the instance is an outlier or not. Suricata is an open-source signature-based network intrusion detection system similar to Snort and configured to use 13,465 rules from the Emerging Threats repository [33]. Since [4] provides results in terms of area under the curve (where $AUC = (1/2)\{TP/(TP + FN) + TN/(TN + FP)\}$), we use AUC for performance comparison. We observe that LSADI outperforms Kitsune on SSL_Renegotiation, SYN_DoS, and Video_Injection attacks. Compared to other methods, LSADI performs better at least three out of five attacks. Overall, LSADI achieves an AUC above 95% on all attacks except the Video_Injection for which it achieves 92%.

3) COMPARISON WITH MUD-ENGINE

MUD-Engine utilizes a one-class SVM with the Markov Chain model to detect volumetric attacks on IoT devices [1]. Figure 10 presents a comparison of the LSADI with the MUD-Engine and one-class classifiers on Volumetric attacks with low, medium, and high rate attacks. Overall, LSADI performs better or equal on ten out of fifteen attacks. MUD-Engine has higher accuracy on five attacks, namely, ARP Spoofing (L-D), Ping of Death (L-D), Fraggle (L-D), TCP SYN (L-D), and TCP SYN Ref (L-D-L). The reason might be due to the small size of ARP, ICMP, TCP, and UDP packets used in these attacks compared to larger packets used for attacks such as SNMP and SSDP. For example, in low-rate ARP Spoofing attacks, the size of the benign traffic increases by 10%, whereas, for SNMP flooding attacks, it increases by 85% to 125%. Since the ratio of malicious traffic is smaller in low-rate attacks, the performance of the LSADI deteriorates. MUD-Engine utilizes traffic features extracted from packets through a learning process. Thus, it can detect unique characteristics of the low rate attacks. Similarly, with TCP SYN and TCP SYN Ref attacks, LSADI achieves an accuracy of almost 100% in high-rate attacks and 95% in medium-rate attacks, but its performance degrades with low-rate attacks. These results indicate that LSADI performs better with high-volume attacks as it relies on locality-sensitive hashing and the impact of an attack to benign traffic is smaller with low-volume attacks. Overall, we observe that both LSADI and MUD-Engine generally outperform other machine learning classifiers. Isolation Forest (IF), Elliptic Envelope (EE), and one-class SVM (OCSVM) achieve higher accuracy than MUD-Engine in SNMP attacks while LSADI performs similarly to IF, EE, and OCSVM.

E. PROCESSING OVERHEAD

We analyzed the processing overhead (shown on Table 4) of the LSADI in terms of the model size (i.e., size of the signature database), feature size (i.e., size of the hash generated from traffic flow), the response time (i. e., the time required to identify the flow). We run experiments on an Intel Core i5-7200U @2.50 GHz CPU with 15.2 GB of available memory and compare results with MUD-Engine on Volumetric attacks. The signature database contains 100 signatures for each device and attack type. As there exist nine devices and fifteen different attack types we generate a total of 900 (i. e., 9 × 100) signatures for attack detection and 1,500 (i. e., 15 × 100) signatures for attack identification.

The size of one Nilsimsa signature is 0.10 KB, TLSH signature is 0.11 KB, and SSDEEP signature is 0.14 KB. Note that the size of a signature is fixed in each of the hashing algorithms. Thus, for Nilsimsa the size of the signature

TABLE 4. Processing cost.

	Feature Size	Detection		Identification	
		Model Size	Response t	Model Size	Response t
Nilsimsa	0.10 KB	0.09 MB	4.4 ms	0.15 MB	62.30 ms
TLSH	0.11 KB	0.10 MB	0.1 ms	0.17 MB	1.17 ms
SSDEEP	0.14 KB	0.13 MB	0.2 ms	0.21 MB	3.62 ms
LSADI	**0.35 KB**	**0.32 MB**	**4.4 ms**	**0.53 MB**	**62.30 ms**
Kitsune	**4.83 KB**	**0.33 MB**	**7.3 ms**	**-**	**-**
MUD Engine	**3.56 KB**	**14.9 MB**	**13.0 ms**	**-**	**-**

database for attack detection is 0.09 MB and for attack identification is 0.15 MB. Similarly, for TLSH, these values will be 0.10 MB and 0.17 MB. For SSDEEP, the size of the signature database for attack detection is 0.13 MB and attack identification is 0.21 MB. Since LSADI utilizes all three hashing algorithms, it will have a total size of 0.32 MB for the attack detection database and 0.53 MB for the attack identification database. These values are lower than the Kitsune and the MUD-Engine.

Nilsimsa processes input at a rate of 3.3×10^5 bytes/second. The processing speed for TLSH is 1.1×10^8 bytes/second and SSDEEP is 1.25×10^8 bytes/second. The processing speed of all locality-sensitive hashes is considerably higher than the IoT traffic rate. For instance, from the dataset, we observe that Chromecast Ultra with a high rate of SSDP (W-D-W) attack has the highest traffic rate of 4.5×10^4 bytes/second. The high processing speed of all locality-sensitive hashes enables LSADI to generate the hash of traffic flow with negligible overhead.

To detect malicious flows, LSADI compares the hash of traffic with the benign signature of the device. The time required for this operation is 4.4 ms for Nilsimsa, 0.1 ms for TLSH, and 0.2 ms for SSDEEP. Since the comparison of hashes can be performed in parallel, LSADI response time depends on the slowest hash function, i.e., Nilsimsa. Thus, the response time of LSADI in detecting the malicious flow is 4.4 ms whereas the response time of Kitsune and MUD-Engine are 7.3 ms and 13.0 ms respectively. Similarly, to identify the malicious traffic, LSADI compares the hash of the traffic flow with signatures of the known attacks in a database. The time required to compare signatures of the known attacks and select the attack type with the highest similarity is 62.30 ms for Nilsimsa, 1.17 ms for TLSH, and 3.62 ms for SSDEEP. Since Nilsimsa is the slowest hash function, the time requirement for LSADI to identify the malicious flow is 62.30 ms.

V. RELATED WORK

In this section, we present prior works on machine learning-based network traffic fingerprinting for anomaly detection. We also provide previous studies that use locality-sensitive hashing for various problems. Finally, we present how our study differs from previous studies.

A. ONE-CLASS CLASSIFIER BASED ANOMALY DETECTION

One-class classifiers are widely utilized to detect anomaly in network traffic. Sven et al. employ the Local Outlier Factor, One-Class SVM, and Isolation Forest to detect anomalous traffic and IoT botnets [3]. Amaal et al. use the Grey Wolf Optimization swarm intelligence algorithm to optimize the hyperparameters of the One-Class SVM and find features that best describe the IoT anomalous traffic [7]. They use One-Class SVM, Isolation Forest, and Local Outlier Factor classifiers for anomaly detection. Hamza et al. analyze the compliance of IoT devices with the MUD behavioral profile and detect volumetric attacks such as DoS, direct and reflective TCP/UDP/ICMP Flooding, and ARP spoofing attacks using machine learning algorithms [1]. The MUD-engine system first learns expected MUD compliant behavior for each device with One-Class SVM and Markov Chain, then measures network traffic of devices to detect suspicious behavior.

B. AUTOENCODER BASED ANOMALY DETECTION

N-BaIoT uses deep autoencoders to construct the signature of the devices from the device's benign network traffic and detect anomalous traffic [2]. The authors evaluate the N-BaIoT on nine devices infected with Mirai and Bashlite and compare results with machine learning algorithms such as Local Outlier Factor, One-Class SVM, and Isolation Forest. Similarly, Kitsune uses autoencoders to detect compromised devices [4]. Kitsune is evaluated with OS scan, man in the middle, Mirai, and denial of service attacks. Additionally, orthogonal matching pursuit uses a greedy sparse recovery algorithm for the autoencoding the network traffic and detect anomaly in it [34]. Deris et al. evaluate the Snort open-source network intrusion detection system to detect TCP FIN flood attacks on IoT devices [35]. However, evaluation by Hamza et al. [1] showed that Snort misses most of the volumetric attacks on IoT devices. DIOT utilizes a federated learning approach for anomaly and intrusion detection [36].

C. ATTACK IDENTIFICATION

Anthi et al. [13] employs a J48 decision tree with 121 traffic features to classify the type of network attack. They evaluate the proposed system on a testbed consisting of 8 devices with 12 attacks. Shaikh et al. [14] use the Center for Applied Internet Data Analysis (CAIDA) dataset to classify malicious IoT traffic. They collect IP addresses of about 3 million IoT devices from Censys and Shodan databases and then compare them to the source IP addresses collected by the network telescope at CAIDA. Authors train Gradient Boosting classifier with nine network traffic features and observe that the classifier can achieve precision and recall of 99.88% in classifying the type of malicious IoT traffic such as port scanning, network scanning, stealth scanning, backscatter or misconfigured.

D. LOCALITY-SENSITIVE HASHING

Previous studies showed the effectiveness of locality-sensitive hashing for spam detection [25], [37], malware classification [38], genome assembling [39], video content analysis [40], [41], electronic identification [42], and for biometric information identification [43]. In a preliminary study [15], we showed that locality-sensitive hashing can also be employed to identify IoT devices.

Compared to previous studies on anomaly detection in IoT traffic discussed in previous subsections V-A and V-B of related works, our study proposes a new approach to detect anomaly in IoT traffic using locality-sensitive hashing, rather than relying on machine learning algorithms such as one-class classifiers and autoencoders. Our approach also enables identifying the type (ARP Spoofing, Fraggle, etc.) of anomaly observed in the network traffic. Compared to previous studies on attack identification (discussed in subsection V-C) our approach does not require feature selection and extraction from network traffic data as it relies on locality-sensitive hash of the data. Compared to previous studies on locality-sensitive hashing provided in subsection V-D, our article focuses on completely different domains, which are anomaly detection and identification (classification of anomaly) in the network traffic of IoT devices.

VI. CONCLUSION

In this paper, we present *Locality Sensitive Anomaly Detection and Identification* (LSADI), a method to detect and identify anomaly in IoT network traffic flows. The proposed approach employs a locality-sensitive hash of the traffic flow. Thus, it does not require feature selection and extraction from the data, a process needing considerable computations and fine-tuning by an expert. The evaluation was performed on three different datasets that contain 25 attacks ranging from volumetric attacks to different IoT malware. LSADI achieved an average detection accuracy of 98% on volumetric attacks and an accuracy above 90% on most of the IoT malware, using only 1 minute of IoT traffic. In terms of attack identification, LSADI, on average, achieves both precision and recall above 94%. Overall, LSADI has negligible processing overhead and performs the same or better than the state-of-the-art machine learning approaches (including one class classifier-based MUD-Engine and neural network-based Kitsune) in detecting malicious traffic.

REFERENCES

[1] A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman, "Detecting volumetric attacks on loT devices via SDN-based monitoring of MUD activity," in *Proc. ACM Symp. SDN Res.* New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 36–48.

[2] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, Jul. 2018.

[3] S. Nõmm and H. Bahsi, "Unsupervised anomaly based botnet detection in IoT networks," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2018, pp. 1048–1053.

[4] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," 2018, *arXiv:1802.09089*.

[5] E. Valdez, D. Pendarakis, and H. Jamjoom, "How to discover IoT devices when network traffic is encrypted," in *Proc. IEEE Int. Congr. Internet Things (ICIOT)*, Jul. 2019, pp. 17–24.

[6] A. Aksoy and M. H. Gunes, "Automated IoT device identification using network traffic," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[7] A. Al Shorman, H. Faris, and I. Aljarah, "Unsupervised intelligent system based on one class support vector machine and grey wolf optimization for IoT botnet detection," *J. Ambient Intell. Hum. Comput.*, vol. 11, no. 7, pp. 2809–2825, Jul. 2020.

[8] S. S. Chawathe, "Monitoring IoT networks for botnet activity," in *Proc. IEEE 17th Int. Symp. Netw. Comput. Appl. (NCA)*, Nov. 2018, pp. 1–8.

[9] A. J. Pinheiro, J. de M. Bezerra, C. A. P. Burgardt, and D. R. Campelo, "Identifying IoT devices and events based on packet length from encrypted traffic," *Comput. Commun.*, vol. 144, pp. 8–17, Aug. 2019.

[10] N. Msadek, R. Soua, and T. Engel, "IoT device fingerprinting: Machine learning based encrypted traffic analysis," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2019, pp. 1–8.

[11] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proc. 13th ACM Conf. Secur. Privacy Wireless Mobile Netw.* New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 207–218, doi: 10.1145/3395351.3399421.

[12] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "HomeSnitch: Behavior transparency and control for smart home IoT devices," in *Proc. 12th Conf. Secur. Privacy Wireless Mobile Netw.* New York, NY, USA: Association for Computing Machinery, May 2019, pp. 128–138, doi: 10.1145/3317549.3323409.

[13] E. Anthi, L. Williams, M. Slowinska, G. Theodorakopoulos, and P. Burnap, "A supervised intrusion detection system for smart home IoT devices," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 9042–9053, Oct. 2019.

[14] F. Shaikh, E. Bou-Harb, J. Crichigno, and N. Ghani, "A machine learning model for classifying unsolicited IoT devices by observing network telescopes," in *Proc. 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2018, pp. 938–943.

[15] B. Charyyev and M. H. Gunes, "IoT traffic flow identification using locality sensitive hashes," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6.

[16] B. Charyyev and M. H. Gunes, "IoT event classification based on network traffic," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 854–859.

[17] B. Charyyev and M. H. Gunes, "Detecting anomalous IoT traffic flow with locality sensitive hashes," in *Proc. IEEE Global Commun. Conf.*, Dec. 2020, pp. 1–6.

[18] S. Calinon and A. Billard, "Incremental learning of gestures by imitation in a humanoid robot," in *Proc. HRI*. New York, NY, USA: Association for Computing Machinery, Mar. 2007, pp. 255–262, doi: 10.1145/1228716.1228751.

[19] Y. Mirsky, T. Halpern, R. Upadhyay, S. Toledo, and Y. Elovici, "Enhanced situation space mining for data streams," in *Proc. SAC*. New York, NY, USA: Association for Computing Machinery, Apr. 2017, pp. 842–849, doi: 10.1145/3019612.3019671.

[20] *Suricata: Open-source IDS, IPS NSM Engine*. Accessed: Feb. 29, 2020. [Online]. Available: https://suricata-ids.org/

[21] H. Bahsi, S. Nõmm, and F. B. La Torre, "Dimensionality reduction for machine learning based IoT botnet detection," in *Proc. 15th Int. Conf. Control, Autom., Robot. Vis. (ICARCV)*, Nov. 2018, pp. 1857–1862.

[22] B. Charyyev and M. H. Gunes, "Locality-sensitive IoT network traffic fingerprinting for device identification," *IEEE Internet Things J.*, vol. 8, no. 3, pp. 1272–1281, Feb. 2021.

[23] R. Rivest and S. Dusse, "The MD5 message-digest algorithm," Tech. Rep., 1992.

[24] S. Gueron, S. Johnson, and J. Walker, "SHA-512/256," in *Proc. 8th Int. Conf. Inf. Technol., New Generat.*, Apr. 2011, pp. 354–358.

[25] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "An open digest-based technique for spam detection," in *Proc. Int. Workshop Secur. Parallel Distrib. Syst.*, 2004, vol. 41, no. 8, pp. 74–83.

[26] J. Oliver, C. Cheng, and Y. Chen, "TLSH—A locality sensitive hash," in *Proc. 4th Cybercrime Trustworthy Comput. Workshop*, Nov. 2013, pp. 7–13.

[27] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digit. Invest.*, vol. 3, pp. 91–97, Sep. 2006.

[28] B. Charyyev, "Security of Internet of Things with network traffic fingerprinting," Ph.D. dissertation, Stevens Inst. Technol., Hoboken, NJ, USA, 2022.

[29] V. H. Bezerra, V. G. T. da Costa, S. Barbon Junior, R. S. Miani, and B. B. Zarpelão, "IoTDS: A one-class classification approach to detect botnets in Internet of Things devices," *Sensors*, vol. 19, no. 14, p. 3188, Jul. 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/14/3188

[30] M. J. E. A. Parmisano and S. Garcia. (2020). *A Labeled Dataset With Malicious Beign IoT Network Traffic*. Stratosphere Laboratory. Accessed: Jan. 22, 2020. [Online]. Available: https://www.stratosphereips.org/datasets-iot23

[31] Cloudflare. (2020). *What is Mirai Botnet?* Accessed: Jan. 11, 2020. [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/

[32] R. Sridharan, R. R. Maiti, and N. O. Tippenhauer, "WADAC: Privacy-preserving anomaly detection and attack classification on wireless traffic," in *Proc. 11th ACM Conf. Secur. Privacy Wireless Mobile Netw.* New York, NY, USA: Association for Computing Machinery, Jun. 2018, pp. 51–62, doi: 10.1145/3212480.3212495.

[33] *Suricata-Rules: Index/Open/Suricata-2.0*. Accessed: Feb. 29, 2020. [Online]. Available: https://rules.emergingthreats.net/open/

[34] C. Tzagkarakis, N. Petroulakis, and S. Ioannidis, "Botnet attack detection at the IoT edge based on sparse representation," in *Proc. Global IoT Summit (GIoTS)*, Jun. 2019, pp. 1–6.

[35] D. Stiawan, D. Wahyudi, A. Heryanto, S. Samsuryadi, M. Y. Idris, F. Muchtar, M. A. Alzahrani, and R. Budiarto, "TCP FIN flood attack pattern recognition on Internet of Things with rule based signature analysis," *Int. J. Online Biomed. Eng.*, vol. 15, no. 7, p. 124, Apr. 2019.

[36] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "DÏoT: A federated self-learning anomaly detection system for IoT," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 756–767.

[37] M. N. Marsono, "Packet-level open-digest fingerprinting for spam detection on middleboxes," *Int. J. Netw. Manage.*, vol. 22, no. 1, pp. 12–26, Jan. 2012.

[38] H. M. Kim, H. M. Song, J. W. Seo, and H. K. Kim, "Andro-simnet: Android malware family classification using social network analysis," in *Proc. 16th Annu. Conf. Privacy, Secur. Trust (PST)*, Aug. 2018, pp. 1–8.

[39] K. Berlin, S. Koren, C.-S. Chin, J. P. Drake, J. M. Landolin, and A. M. Phillippy, "Assembling large genomes with single-molecule sequencing and locality-sensitive hashing," *Nature Biotechnol.*, vol. 33, no. 6, pp. 623–630, Jun. 2015.

[40] S. Hu, "Efficient video retrieval by locality sensitive hashing," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 2, Mar. 2005, pp. II/449–II/452.

[41] Y. Zhang, H. Lu, L. Zhang, X. Ruan, and S. Sakai, "Video anomaly detection based on locality sensitive hashing filters," *Pattern Recognit.*, vol. 59, pp. 302–311, Nov. 2016.

[42] W. P. Filho, C. Ribeiro, and T. Zefferer, "An ontology-based interoperability solution for electronic-identity systems," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2016, pp. 17–24.

[43] A. Alruban, N. Clarke, F. Li, and S. Furnell, "Biometrically linking document leakage to the individuals responsible," in *Trust, Privacy and Security in Digital Business*, S. Furnell, H. Mouratidis, and G. Pernul, Eds. Cham, Switzerland: Springer, 2018, pp. 135–149.

BATYR CHARYYEV received the B.S. degree in computer engineering from Middle East Technical University, Ankara, Turkey, the M.S. degree in computer science and engineering from the University of Nevada, Reno, and the Ph.D. degree in systems engineering from the Stevens Institute of Technology, Hoboken, NJ, USA. He is currently an Assistant Professor with the Computer Science and Engineering Department, University of Nevada. His research interests include network systems, the Internet of Things, traffic fingerprinting, cyber security and privacy, network science, and edge computing.

MEHMET HADI GUNES received the M.S. degree in computer science and engineering from Southern Methodist University and the Ph.D. degree in computer science from The University of Texas at Dallas. He is currently a Senior Software Engineer with Akamai, working on networking and cloud and edge computing. Prior to joining Akamai, he was an Associate Professor with the Stevens Institute of Technology. His research has been funded by the National Science Foundation, the National Institute of Justice, the Department of Defense, Amazon AWS, the Cincinnati Children's Center for Pediatric Genomics, Argonne ALCF, and the Nevada Homeland Security Grant Program. His research interests include network systems, complex networks, data science, and cyber security.