WADAC: Privacy-Preserving Anomaly Detection and Attack Classification on Wireless Traffic

Ragav Sridharan
Singapore University of Technology and Design (SUTD)
Somapah Road 8
Singapore 487272
ragav_sridharan@sutd.edu.sg

Rajib Ranjan Maiti
Singapore University of Technology and Design (SUTD)
Somapah Road 8
Singapore 487272
rajib_maiti@sutd.edu.sg

Nils Ole Tippenhauer
Singapore University of Technology and Design (SUTD)
Somapah Road 8
Singapore 487272
nils_tippenhauer@sutd.edu.sg

ABSTRACT

In this work, we address the problem of detecting application-layer attacks on nearby wireless devices. In particular, we assume that the detection scheme is limited to link-layer traffic (either because schemes such as WPA2 are used, and the key is unknown, or to preserve user privacy). Such a setting allows us to detect attacks in nearby third party networks that we are not associated with, unlike related work that relies on wireline taps to observe traffic. We propose and implement a framework consisting of an anomaly detection module (unsupervised), and an attack classification module that identifies a known set of attacks (supervised). We evaluate our prototype with experiments including a range of attacks. For example, we demonstrate that the anomaly detector detects Mirai C&C traffic by an IoT device (without training with Mirai). In addition, we detect that the Mirai infected device is attacking other devices with 96.1% accuracy. We show that our prototype can be applied to different wireless standards (such as 802.11 (WiFi) and 802.15 (Zigbee)) and detect attacks with an accuracy of 96%-99%.

1 INTRODUCTION

In recent years, the number of IoT devices has increased dramatically, thereby increasing threats to security and user's privacy. With the extensive adoption of IoT devices in various areas such as health monitoring, smart homes, industrial and public control, the number of IoT-related attacks has increased [17, 33]. Recently, large-scale campaigns by botnets such as Mirai [5] have targeted IoT devices. Compromise of devices often goes undetected, as end users are ignorant of such threats [14, 15]. Thus, compromised IoT devices can be used to launch attacks such as a large-scale distributed denial of service (DDoS) attack [1].

Intrusion Detection Systems (IDS) can help users to detect compromised devices in their network. IDS can observe network traffic at different layers of the OSI model. Traffic collected at the network-layer (or above) provides information about the application and user, which could be considered to violate the user's privacy [16]. Such traffic is typically collected using a network tap (or by placing the IDS in the path of the traffic), only possible for network operators.

In this paper, we show that it is possible to perform anomaly detection and attack classification using wireless link-layer traffic only, without access to network-layer (and above) payload. In particular, this would allow a passive IDS to perform anomaly and attack detection on a third party network in the neighborhood, even if network-layer (and higher) data is encrypted. Using link-layer data, we propose and implement a framework consisting of an anomaly detection module (unsupervised), and an attack classification module that identifies a known set of attacks (supervised). Since the system does not access network-layer (and higher) data, we argue that our proposed system poses only a minimal privacy threat to the monitored devices or users. The system passively captures (publicly available) wireless link-layer traffic and extracts a set of features. While such features do not include data from the network layer (such as IP addresses), additional features related to control and management frames are available (which are not present in wireline traffic). The features are then used by an unsupervised classifier (trained with benign traffic) to detect anomalous traffic patterns. If an anomaly is detected, we use a supervised classifier to determine the type of attack (based on a set of learned attacks). Identifying a known attack will help in proceeding with a particular mitigation strategy. We use cheap commercial-of-the-shelf radios (COTS), and do not require traffic captures from mirroring ports, or Man-in-the-Middle setups. We test our proposed system on WiFi-enabled IP cameras and smart bulbs, and Zigbee enabled smart bulbs. By testing our system on various types of devices using different wireless protocols, we claim that our framework can easily be applied to other wireless devices.

Our main contributions are summarized as follows:

- We design and implement a framework to passively detect and identify application layer attacks by analyzing link-layer features of encrypted wireless traffic. A total of 632MB of data (3.25 million WiFi and Zigbee frames) is collected from different IoT devices to build and test our framework.
- Our experiments show that the detection mechanism reliably detects 96.2% of attacks on IP cameras (e.g., brute-force, firmware, DoS), and classifies them with 97% accuracy.
- We use our framework to detect IP camera infection by Mirai. We detect a Mirai bot (an active IP camera) launching attacks on other devices with 96.1% accuracy. Additionally, our framework detects DDoS attacks on IoT devices,

WiSec '18, Stockholm, Sweden
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
978-1-4503-5731-9/18/06…$15.00
DOI: 10.1145/3212480.3212495

Frame Control	Duration	Dest. Address	Sender Address	Receiver AP Address	Seq. Control	Transmitter AP Address	Encrypted Payload	CRC32 Checksum
16bits	16bits	48bits	48bits	48bits	16bits	48bits	0-2312Bytes	32bits

Figure 1: IEEE 802.11 link-layer frame format with encrypted network-layer payload.

launched by Mirai botnet (formed using 3 D-Link cameras) with 98.3% accuracy.

- We apply our proposed framework to a different type of device (TP-Link bulb using WiFi) and show that we detect anomalies with more than 98.8% accuracy.
- Finally, we test our proposed framework on Zigbee based smart bulb, and detect association flooding attacks on the Zigbee controller with 99% accuracy.

The paper is structured as follows. Section 2 presents specific background information to align readers with our work. Section 3 describes the architecture of the proposed framework and the system model. The implementation of our framework is elaborated in Section 4. In Section 5, we discuss the experimental setup, and different datasets we build from the collected traffic. We discuss the results of applying our framework in different scenarios in Section 6. Section 7 draws comparisons of our work with related works. Finally, we conclude our work in Section 8.

2 BACKGROUND

In this section, we review concepts essential for our framework.

2.1 Feature Extraction and Feature Selection

To ensure that privacy of user data is preserved, we extract a set of features from frame headers of the link-layer, without decrypting the encrypted payload (see Figure 1). We then select a subset of the extracted features using *information value* (IV) of each feature. IV quantifies the influence a feature has in detecting an anomaly. IV for a feature f_1 is computed by first grouping the values of f_1 into n groups based on the distribution of anomalous and non-anomalous traffic. Then,

$$IV = \sum_{i=1}^{n} (f_{\mathrm{i}}^{\mathrm{na}} - f_{\mathrm{i}}^{\mathrm{am}}) \ln(\frac{f_{\mathrm{i}}^{\mathrm{na}}}{f_{\mathrm{i}}^{\mathrm{am}}}) \tag{1}$$

where f^{na} and f^{am} indicates the percentage of values that accounts for non-anomaly and anomaly data respectively in each of n groups. In this work, features having IV in the range of 0.5 to 1 are considered as reliable predictors.

2.2 Autoencoders

Anomaly detection can be done using either supervised or unsupervised machine learning algorithms. It is difficult to detect unknown anomalies (not present in the training data) using supervised learning. Hence, in this study, we use *autoencoder* [9] which is an unsupervised learning model.

An autoencoder tries to learn an approximation to the identity function, to generate an output vector that is similar to its input vector. In our case, the input vectors are features extracted from non-anomalous traffic. It takes a $d-$dimensional feature vector $\vec{x}$ as input, represents it in a lower dimensions, and then reconstructs a $d-$dimensional output vector $\vec{x}'$ with minimal information. The most basic autoencoder (called vanilla-autoencoder) has one input layer, one hidden layer, and an output layer. In the encoder phase, the autoencoder reduces the input vector $\vec{x}$ to a lower dimension vector $\vec{h_1}$ (also called as hidden layer):

$$\vec{h_1} = F(W_{\mathrm{e}}\vec{x} + b_{\mathrm{e}}) \tag{2}$$

In the decoder phase, the autoencoder reconstructs the output from the hidden layers. In a vanilla autoencoder, the output vector $\vec{x}'$ is constructed from the output of the hidden layer $\vec{h_1}$:

$$\vec{x}' = F(W_{\mathrm{d}}\vec{h_1} + b_{\mathrm{d}}) \tag{3}$$

Each layer has Weight and bias vectors (W_{e} and b_{e} in encoder phase, and W_{d} and b_{d} in decoder phase) associated with it. The Weight and the bias vectors are initialized randomly. Autoencoders are trained to minimize the reconstruction error (e) between $\vec{x}$ and $\vec{x}'$:

$$e = \vec{x}' - \vec{x} \tag{4}$$

Weight and bias vectors at each layer are recalculated using back-propagation to minimize the reconstruction error during training.

In our work, we use *deep autoencoders* with *tanh* as the activation function *F*, and stochastic gradient descent as the optimizer.

3 PRIVACY-PRESERVING ATTACK DETECTION AND CLASSIFICATION

In this section, we discuss our system and adversary models. We also describe our framework for anomaly detection and attack classification.

3.1 System Model and Attacker Model

Our system model consists of three actors. An IoT device, an attacker, and our proposed detection system WADAC (Wireless Anomaly Detection and Attack Classification). The IoT device is set up in a neighborhood connected using wireless technology (e.g., 802.11 or 802.15), and the attacker is interacting with the IoT device through the wireless medium.

The attacker's goal is to use application-layer attacks to try and compromise the IoT device. Once it is compromised, the attacker would be able to perform different attacks. WADAC is set up in the proximity of both the IoT device and the access point (AP). It monitors wireless traffic and detects attacks without violating the user's privacy. We assume the IoT device uses a static channel to communicate with the infrastructure. WADAC continuously monitors that channel from a distance to observe traffic exchanged between the IoT device and the AP. A third-party can operate WADAC without access to network keys, and thus, it cannot associate to APs, or decrypt intercepted traffic payload.

3.2 Problem Statement

Our goal is to detect attacks that occur on layer 3 and above over wireless links, without decrypting information at network-layer (or above), thereby ensuring the privacy of users. As the likelihood of attacks is not known apriori, we aim to develop an anomaly detector that maximizes accuracy and raises minimum false alarms. After detecting an anomaly, we want to identify the type of attack based on a set of trained attack types.

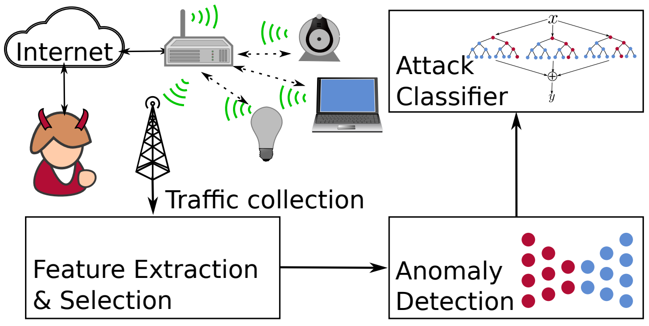


Figure 2: WADAC Framework

3.3 WADAC: Wireless Anomaly Detection and Attack Classification

WADAC has four modules: traffic collector, feature extractor & selector, anomaly detector, and attack classifier. Figure 2 shows a block diagram of our framework.

The traffic collector module passively sniffs link-layer wireless traffic. It identifies traffic belonging to a particular device based on the MAC address of the device. The feature extractor & selector module first extracts features from the traffic collected, and then selects relevant features. The selected features are used by the anomaly detector module to detect anomalies. If an anomaly is detected, the attack classifier module classifies the anomaly into one of the learnt sets of attacks. To test the performance of WADAC, we develop a separate module called attack generator to perform various attacks on different IoT devices under observation.

Each wireless communication technology has its encryption technique ensuring security and privacy of the communication. In our work, we do not attempt to decrypt this encryption. Thus, we claim that our system minimizes the privacy threat to the monitored users. Our framework can be integrated into an existing network structure without additional changes required.

3.4 Anomaly Detector and Attack Classifier

The anomaly detector and the attack classifier are the core modules of WADAC. In our problem, every attack is considered as an anomaly. Once the anomaly detector module reports an anomaly, the attack classifier module identifies the type of attack based on the learnt set of attacks. The underlying intuition is that normal IoT traffic is highly regular, and thus autoencoders will perform well to discriminate attacks.

Anomaly Detector. This module uses autoencoders to detect anomalous traffic patterns. By (unsupervised) training on normal traffic, autoencoders learn a pair of non-linear transformations: A mapping from the original space to hidden space (encoder) and a mapping back from hidden space to the original one (decoder). These transformations help to reconstruct normal traffic with minimal reconstruction errors. Since attacks are rare and different, any data point with a reconstruction error larger than a given threshold is considered an anomaly. Choosing an optimal threshold is vital for anomaly detection. We may not detect any attack if the threshold is very large. Likewise, a small threshold would raise many false alarms. To select an optimal threshold, we analyze the accuracy, sensitivity and specificity achieved by different threshold values. In this work, we experiment with multiple threshold values (E^{th}) calculated using the following formula:

$$E^{\text{th}} = P + x * IQR \tag{5}$$

where P varies from 0th percentile to 100th percentile of reconstruction errors and we take x from the set of values $\{0,1,1.5\}$. IQR is the Inter Quantile Range of reconstruction errors. IQR is calculated as:

$$IQR = P^{75\text{th}} - P^{25\text{th}} \tag{6}$$

Attack Classifier. In the attack classifier module, we perform supervised training of a *random forest* model [12], using a known set of attacks. We then deploy this model to check if the anomaly detected belongs to a known set of attacks.

4 SYSTEM IMPLEMENTATION

All modules in WADAC are developed and implemented using open-source tools based on Python and R[1]. Each module has minimal dependency with other modules. This gives the user an option to either use the complete framework as a single unit or to use any module independently.

4.1 Attack Generator

We test the performance of WADAC by performing various attacks on different IoT devices using the attack generator module. This module finds target IoT devices in the network using *arp-scan* and launches various attacks on them. In particular, we do port scans, vulnerability checks, brute force Telnet logins, Denial of Service (DoS), Distributed Denial of Service (DDoS), Mirai and Zigbee flooding attacks. Table 7 in Appendix A shows the list of attacks implemented on different devices. Three different devices (a laptop, a smartphone, and a tablet) connected to the AP are used to implement these attacks.

Mirai. Launching a Mirai attack involves two stages. In the first stage, a control and command (C&C) server is set up and then the Mirai bot is installed on the target IoT device. In our experiment we install the Mirai bot on a D-Link camera (model DCS-942L B). Both the C&C server and the D-Link camera are connected to the same network via a wireless AP over WiFi. We install Mirai on the camera via a Telnet session similar to the original process.In the second stage, attacks (such as DDoS) are launched on other IoT devices using the D-Link camera as a Mirai bot.

Vulnerability checks and Telnet brute-force. We use *nmap* [30] to check for open ports and vulnerabilities. We also use nmap to perform Telnet brute-force attacks.

DoS. DoS attacks can either be launched by using existing tools or by using custom-built Python scripts. To perform DoS using an existing tool, we use *Slowloris* [40] which is a nmap based HTTP DoS attack tool. Slowloris performs a large number of partial HTTP requests without allowing the TCP session to close. This exhausts server resources and prohibits the server from establishing new legitimate connections. To get more control over the rate and the type of traffic causing DoS, we develop a custom-built Python script to launch specific DoS attacks such as UDP flooding.

[1]The source code and sample data is available at https://github.com/scy-phy/wadac.

DDoS. We use customized DDoS attacks using four malicious applications on an android smartphone and two android tablets. The applications include DDOS app [7], Terminal Emulator for Android [32], Packets Generator [20] and AnDOSid [19].

Firmware. We included remote firmware upgrades as an attack example, as it can be part of a more complex attack. In any case, we argue that firmware changes are noteworthy events to detect.

Zigbee flooding attacks. Unlike WiFi-based devices, IoT devices using wireless technologies such as Zigbee are not connected to the Internet directly. These devices are connected to a *mother controller* which is connected to the Internet via wired or wireless medium. To launch Zigbee flooding attacks, the attacker needs to be in the physical neighborhood of the device. The attacker can target either the mother controller or actual Zigbee devices. We use KillerBee tool [39] to launch Zigbee flooding attacks.

4.2 Traffic Collector

Our traffic collector module sniffs wireless frames using COTS radios, extended from a prior work setup [27]. Specifically, a TP-Link 150Mbps high gain adapter is used to capture WiFi traffic, and an Atmel RZ Raven USB Stick is used to collect Zigbee traffic (compatible with KillerBee). The WiFi adapter is capable of performing channel hopping through 14 WiFi channels. Sniffing traffic from the same channel as the AP helps to collect more data frames. In our experiments, we collect traffic in two scenarios. In the first case, we allow the adapter to hop on different channels. In the second situation, we align our adapter and the access point to work on a same channel. The range and the physical location of the adapter also plays a significant role in controlling traffic capture [6]. We collect traffic by placing our adapter within a distance ensuring minimal packet loss.

The traffic collector module uses *scapy* [11] to collect WiFi frames sniffed by the TP-link adapter. The collected traffic is then grouped based on their MAC address. This grouping helps us to filter traffic belonging to a particular IoT device and manually label them according to the activities performed on that device. For example, the traffic collected while allowing an IP camera to stream and send video is annotated as *camera-streaming*. Similarly, traffic collected for IP camera during a brute force attack on Telnet is annotated as *camera-brute forced*. This labeling helps to validate the results of attack detection and classification. We use two Atmel RZ Raven USB Sticks with KillerBee to sniff Zigbee frames and to inject frames.

The wireless traffic is collected from a) IP cameras, b) WiFi enabled smart bulbs, and c) Zigbee enabled smart bulbs. We first collect normal traffic from different devices, and then collect malicious traffic while performing different attacks on them. Table 8 in Appendix A shows the total amount of traffic collected from different devices in different scenarios.

4.3 Feature Extractor & Selector

This module takes pcap files containing frames associated with a particular device as input and extracts a set of features. The process of extracting features from the traffic collected does not depend on device type, or the wireless standard followed.

Extracting Packet Features. The feature extractor extracts a set of features based on packet header information of each frame. This set of features are termed *packet features*. Because of different frame formats, we get different set of packet features from WiFi and Zigbee traffic. Table 1 lists packet features that we extract.

Field Name	Value(s)
Dest. MAC	WiFi: 48 bits, Zigbee: 16 bits
Src. MAC	WiFi: 48 bits, Zigbee: 16 bits
Trans. MAC	WiFi: 48 bits/ Null
Recv. MAC	WiFi: 48 bits/ Null
Src. PAN Id	Zigbee: 16 bits
Dest. PAN Id	Zigbee: 16 bits
frame type	WiFi : {data/management/control}, Zigbee: {command/data/beacon}
frame subtype	WiFi: {probe req, RTS, QoS data, etc.}
more frag. flag	WiFi: {0/1}
more data flag	WiFi: {0/1}
frame size	WiFi: max. 2324B, Zigbee: max. 121B
frame time	system time of frame capture

Table 1: Packet features for WiFi and Zigbee frames with possible values.

Extracting Traffic Features. Since frames of similar size and type arrive in bursts, the feature extractor first groups them into blocks of size s_{blk}. Blocks with less than s_{blk} frames are discarded. After grouping, each block is processed to create one signature for the corresponding MAC address. Thus, we have $k = n_f / s_{\text{blk}}$ signatures from a pcap file containing n_f frames for a given MAC address.

Each block of packets is processed to compute a set of statistical features using packet features. We call this set of statistical features as *traffic features*. Overall, we categorize the traffic features into four types: census (count of different types of frames in a block), ratio (ratio of numbers of frames in a block), load (mean and standard deviation of sizes of frames in a block), and gap (mean and standard deviation of time gaps of frame arrival times in a block).

In case of WiFi, the frame size of different types of frames varies significantly during an attack. This difference in frame size of different frame types, is not observed in Zigbee. Based on this observation, WiFi frames in each block are grouped according to their *frame type* into three types; data frames (d-frames), control frames (c-frames), and management frames (m-frames). As the size of d-frames vary the most during an attack, they are divided into bins of equal length l_{bin}. Thus, d-frames are placed into $c_{\text{b}} = (L_{\max} - L_{\min})/l_{\text{bin}}$ bins, where l_{bin} is bin length, and $L_{\max}$ and $L_{\min}$ are the maximum and minimum length of the WiFi frames.

Table 2 shows the number of traffic features of each type in each block for WiFi frames. The feature extractor extracts a total of 45 traffic features from each block. Table 3 shows the number of traffic features in each bin of data frames. Overall, we obtain 18 features from each bin. For $c_{\text{b}} = 2$ bins in a block, $18 * 2 = 36$ features are extracted. Additionally, 45 traffic features are extracted in total from each block. Thus, overall, $45 + 36 = 81$ traffic features are extracted from WiFi frames. Similarly, a total of 72 features are extracted from Zigbee frames. These features include all four categories, i.e. census, load, ratio and gap. Table 4 shows the description and the type of features extracted from both WiFi and Zigbee frames.

Type	Control	Manag.	Data	Total
census	6	6	6	18
ratio	3	3	3	9
load	6	6	6	18
total	15	15	15	45

Table 2: Traffic features extracted from each block.

Type	Census	Ratio	Load	Gap	Total
#Features	3	3	6	6	18

Table 3: Traffic features extracted from each bin.

The feature extractor uses *scapy* and *killerbee.scapy_extensions* packages to extract packet features from WiFi and Zigbee frames respectively. Traffic features for each signature are then extracted using a Python script. Both packet and traffic features are stored separately for WiFi and Zigbee in a *SQLite* database table.

Table 4: Features description. Type t of frame in WiFi is management frame/data frame/control frame and in Zigbee is data frame/command frame/beacon frame. Direction of frame is either sent (s), received (r) or any (x).

Type	Notation	Description
	t	type of frame
	dir	direction of a frame; x/s/r
	s_{blk}	number of frames in a block
	l_{bin}	bin length
	$f(t, \text{dir})$	set of frames (type t) flowing in dir
	$u(t, \text{dir})$	set of unique t frame sizes in dir
	$f(i, t, \text{dir})$	set of frames (type t) in bin b_i in dir
	$g(i, t, \text{dir})$	time gaps of frames (type t) in bin b_i in dir
Census	$\kappa(t, \text{dir})$	$\lvert f(t, \text{dir})\rvert$
	$\kappa(i, t, \text{dir})$	$\lvert f(i, t, \text{dir})\rvert$ in a bin b_i
	$\gamma(t, \text{dir})$	$\lvert u(t, \text{dir})\rvert$
Ratio	$r(t, \text{dir}, s_{\text{blk}})$	ratio of $\kappa(t, \text{dir})$ and s_{blk}
	$r(i, t, \text{dir}, s_{\text{blk}})$	ratio of $\kappa(i, t, \text{dir})$ and s_{blk}
Load	$l_\mu(t, \text{dir})$	mean of the sizes of frames in $f(t, \text{dir})$
	$l_\mu(i, t, \text{dir})$	mean of the sizes of frames in $f(i, t, \text{dir})$
	$l_\sigma(t, \text{dir})$	std. of the sizes of frames in $f(t, \text{dir})$
	$l_\sigma(i, t, \text{dir})$	std. of the sizes of frames in $f(i, t, \text{dir})$
Gap	$g_\mu(i, t, \text{dir})$	mean of time gaps in $g(i, t, \text{dir})$
	$g_\sigma(i, t, \text{dir})$	std. of time gaps in $g(i, t, \text{dir})$

Feature Selector. Feature Selector selects features important for anomaly detection or attack classification. We implement this module using the *Information* package in R. The *createinfo_tables* function computes the information value for each feature.

4.4 Anomaly Detector and Attack Classifier

Anomaly Detector. Autoencoders in the anomaly detector are built in *R* using the *h2o*[23] package. We use *IQR* function to calculate the inter-quartile range to compute thresholds for classification.

The anomaly detector module is heavily reliant on the functionality of a device. Since the traffic patterns of a device can vary significantly with the change in its functionality, the number of autoencoders required to learn bening traffic patterns (corresponding to its functionality) varies accordingly.

For instance, the behavior of an IP camera is different when it is *idle*[2] in comparison to when it is *active*[3]. Because of this difference in traffic patterns, we train two separate autoencoders (called enc_{i} and enc_{l}) for the two different states (idle and active) of IP cameras. If both enc_{i} and enc_{l} detect an anomaly, an anomaly is reported. However, in the case of TP-Link bulb and Phillips Hue bulb, a single autoencoder is sufficient to detect anomalies. This is because both these devices have limited functionalities (i.e. they are of similar device type), that do not overlap with a camera's functionality. Anomaly detectors for a specific device's traffic would be selected based on the device type (e.g., using the MAC address). We leave investigations of generalization of our anomaly detectors for generic devices types to future work.

Attack Classifier. In the Attack Classifier module, we train a random forest model with signatures of different attacks. Each anomaly traffic signature is labeled manually with the corresponding attack (e.g., slowloris or UDP flood). The set of features used for classification is the same set of features used in the anomaly detector module. We use the *randomforest* package in R to build a random forest model and the *caret* package in R for 10-fold cross-validation, model tuning, and to evaluate the performance of the model. This module is activated only when an anomaly is detected by the anomaly detector module.

5 EXPERIMENTAL SETUP

In this section, we give an overview of our testbed, the traffic collection process and the signature extraction process.

5.1 Testbed

All our experiments are conducted inside a shielded room to isolate devices under observation and avoid any external wireless interference. Our testbed consists of six IP cameras, two smart lights and three Zigbee enabled smart lights. The IP cameras are from five different vendors. The WiFi enabled smart light are from TP-Link and the Zigbee enabled smart bulbs are from Philips Hue. Each IoT device is controlled using their respective Android applications installed on a OnePlus smartphone or a Lenovo Tablet. The Zigbee

[2]Idle state of the device implies that the device is not accessed through their apps, but remains switched on.
[3]The active state implies that the device is actively accessed through their app, e.g., stream live video,capture images, sense motion etc.

enabled smart lights are connected to a base controller which is connected to Internet via Ethernet.

5.2 Wireless Traffic Collection

IP cameras. We collect benign traffic from IP cameras while performing different functions on them. To analyze traffic patterns during different attacks, we collect malicious/attack traffic when different attacks are launched on targeted IP Cameras. The collected traffic is annotated based on timestamps at which a particular function is performed. We divide the collected traffic into 5 subsets:

(1) D_a^{cam} – the traffic contains only malicious traffic. The malicious traffic comprises of all attacks other than Mirai.
(2) D_i^{cam} – this subset of traffic consists of only idle behavior (traffic collected when the device is not accessed using its corresponding app),
(3) D_l^{cam} – this subset represents traffic for only active behavior (traffic collected while performing different functions on the IP camera using its corresponding app),
(4) $D_{\text{ail}}^{\text{cam}}$ – this subset is a mix of the anomalous, idle and active behavior of the cameras.
(5) $D_{\text{Mirai}}^{\text{cam}}$ – this subset contains traffic collected when a Mirai attack is launched on D-Link camera (i.e., infecting the camera and launching Mirai attack from it on other devices).
(6) $D_{\text{m-DDoS}}^{\text{cam}}$ – this subset of traffic is collected from NestCam and TP-Link IP cameras when 3 Mirai infected D-link cameras form a botnet, and launch DDoS attacks on them.

We use D_i^{cam} and D_l^{cam} for training the autoencoders and the rest for testing trained models.

WiFi enabled smart bulbs. We collect two sets of traffic from WiFi enabled smart bulbs.

(1) D_l^{bulb} – contains traffic collected when the device performs normal activities such as changing color or changing lighting themes.
(2) $D_{\text{ld}}^{\text{bulb}}$ – contains mixed traffic collected when the bulbs perform normal activities and when DDoS attacks are launched on the bulbs.

D_l^{bulb} is used to train autoencoders and the other traffic set is used to test the performance of the trained model.

Zigbee enabled smart bulbs. The traffic collection process in Zigbee enabled smart bulbs is similar to that in TP-Link bulbs. We collect two sets of traffic from this set of bulbs:

(1) D_l^{zb} – contains non-anomalous traffic used for training autoencoders, and
(2) $D_{\text{al}}^{\text{zb}}$ – consists of a mixture of malicious (i.e., only Zigbee flooding attack) and normal traffic.

5.3 Signature Formation

We experiment with different s_{blk} for WiFi and Zigbee frames to improve the performance of the anomaly detection module (see Section 4.3). For WiFi frames, we select block size s_{blk} from {100, 200, 300, 400, 500}, and for Zigbee, we choose s_{blk} from {40, 60, 80, 100} to form signatures. Every signature is labeled according to the corresponding activities performed at the time of traffic collection.

6 RESULTS

In this section, we discuss the performance metric used to test our framework and present the results of our analysis.

6.1 Performance Metric

Our primary objective is to detect anomalies with a high accuracy and minimal false alarms. Once an anomaly is detected, we aim to identify the attack with a high accuracy.

We use three known metrics to evaluate the performance of trained models for anomaly detection.

- Accuracy (Acc) - ratio between the number of correct detection and the total number of test samples, i.e.,

$$Acc = \frac{N_{TP} + N_{TN}}{N} * 100 \tag{7}$$

where N_{TP} is the number of true positives, N_{TN} is the number of true negatives and N is the total number of observations.
- Sensitivity (Sens) - ratio between the number of *correctly detected normal* samples and the total number of normal samples, i.e.,

$$Sens = \frac{N_{TP}}{N_{TP} + N_{FN}}. \tag{8}$$

N_{FN} is the number of false negatives. Higher sensitivity indicates lower number of false alarms.
- Specificity (Spec) - ratio between the number of *correctly detected anomaly* samples and the number of anomaly samples, i.e.,

$$Spec = \frac{N_{TN}}{N_{TN} + N_{FP}}. \tag{9}$$

N_{FP} is the number of false positives.

6.2 Anomaly Detection: IP Camera

We develop autoencoders enc_{i} and enc_{l} in the anomaly detector module, to detect anomalies in IP cameras. The two autoencoders are trained using normal traffic in D_i^{cam} and D_l^{cam} datasets. enc_{i} detects anomalies in the idle state of the camera, and enc_{l} detects anomalies in the active state of the camera. We detect an attack when both autoencoders raise an alarm.

Changing the block size and bin length effects the distribution of each feature (see Section 4.3). Hence, choosing optimal block size and bin length plays a vital role in improving the performance of our framework. To find the optimal combination, we train autoencoders to achieve high accuracy with different combinations of bin length l_{bin} and block size s_{blk} by varying its configuration (i.e., number of hidden layers and number of nodes in those layers). Table 9 in Appendix B.1 shows the configurations of autoencoders used when $l_{\text{bin}} = 800$ and s_{blk} is varying.

ROC Curve Analysis. Choosing an optimal threshold of reconstruction errors is essential for anomaly detection. To select an optimal threshold for the anomaly detector module, a set of threshold values (E^{th}) are calculated by varying P and x in Eq. 5 during training. We observe that $x = 1$ gives the least misclassification rate in the case of IP cameras. Thus, we fix $x = 1$ in this case. P takes percentile values from {0, 5, 10, ..., 100}. Plotting the ROC plots for different l_{bin} and s_{blk} by varying the threshold helps to

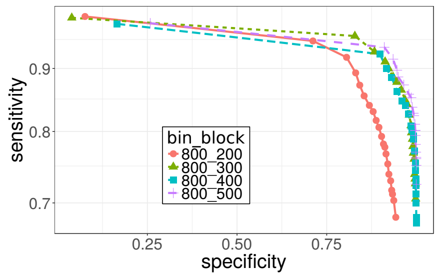


Figure 3: ROC Plots using different thresholds for different l_{bin} and s_{blk}. Though, $l_{\text{bin}} = 800$ and $s_{\text{blk}} = 500$ covers slightly more area, $l_{\text{bin}} = 800$ and $s_{\text{blk}} = 300$ shows higher sensitivity.

identify the optimal threshold, l_{bin} and s_{blk} for anomaly detection. Figure 3 shows the ROC plot built using E^{th} for a fixed $l_{\text{bin}} = 800$ and varying s_{blk} using the dataset $D^{\text{cam}}_{\text{ail}}$. We observe that the upper threshold of $P = 95^{\text{th}}$ percentile of reconstruction errors in Eq. 5 gives best results. Thus we fix $P = 95^{\text{th}}$ percentile for anomaly detection in IP cameras. Further, the Area under the Curve (AUC) is similar for $s_{\text{blk}} = 300, 400$ and 500. However, the sensitivity is the highest when $s_{\text{blk}} = 300$. This means that there are lesser number of false alarms with $s_{\text{blk}} = 300$. Additionally, extracting features from a block with $s_{\text{blk}} = 300$ takes lesser time as compared to a block with $s_{\text{blk}} = 400$ or 500.

Optimal Block Size and Bin Length. Figure 4a shows the variation of accuracy (Acc) with different combinations of l_{bin} and s_{blk} in D^{cam}_{a} dataset. This dataset only contains attack signatures. Hence, there are no true positives in this dataset which makes sensitivity = 0. The highest accuracy (96.9%) is obtained with $l_{\text{bin}} = 800$ and $s_{\text{blk}} = 400$. $l_{\text{bin}} = 800$ signifies that we extract features from two bins of *data* frames. Reducing s_{blk} from 400 to 300 marginally reduces the accuracy to 96.2%. Figures 4b , 4d and 4c show the accuracy, specificity and sensitivity observed for different combinations of l_{bin} and s_{blk} in $D^{\text{cam}}_{\text{ail}}$ dataset. Even for this dataset, choosing $l_{\text{bin}} = 800$ and $s_{\text{blk}} = 400$ gives the highest accuracy (95.2%) and specificity(0.96). However, the highest sensitivity (0.89) is observed when $l_{\text{bin}} = 800$ and $s_{\text{blk}} = 300$. The accuracy and specificity only reduces marginally to 93.2% and 0.94 respectively with $s_{\text{blk}} = 300$. This analysis matches the ROC curve analysis and confirms $l_{\text{bin}} = 800$ and $s_{\text{blk}} = 300$ (for IP cameras).

Feature Importance. Table 10 in Appendix B.2 lists the importance values of different features used in enc_{i} and enc_{l} for $l_{\text{bin}} = 800$ and $s_{\text{blk}} = 300$. We see that the features of type *Gap* (i.e., time gaps between sent or received, or only sent, or only received data frames) have higher significance in general as compared to other types of features. Our experiments highlight that $g_{\mu}(1, \text{data}, s)$, i.e., mean time gap between sent data frames in bin b_1, and $g_{\mu}(1, \text{data}, r)$, i.e., mean time gap between received data frames in bin b_1, are the most significant features in enc_{l} and enc_{i} respectively. Similar features in bin b_0 are also important. On tracing back to pcap files, we observed that a large number of both small (less than 300B) and large (more than 1400B) sized data frames (subtype as "reserved") are sent quite frequently when the camera (e.g. D-Link camera) is streaming. This analysis confirms that the behavior of non-anomalous traffic of IP cameras is well captured by the features we extract in this work.

6.3 Detecting Botnet Attacks

We use our pre-trained autoencoders enc_{l} and enc_{i} to detect the communication between the C&C server and a Mirai bot. We also detect attacks launched by a Mirai botnet on IP cameras.

Detecting Mirai C&C Activity. Our system analyzes traffic patterns between C&C and camera. The anomaly detector module detects the communication of the the C&C and the camera with 98% accuracy. When the camera communicates with the C&C server, the number of frames exchanged is relatively higher. Thus, we can detect this communication with high accuracy.

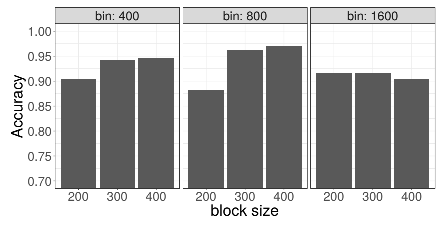
Detecting Mirai Bot launching attacks. We use $D^{\text{cam}}_{\text{Mirai}}$ dataset to test for Mirai bots launching attacks. WADAC detects an active IP camera infected with Mirai, launching attacks on other devices, with 96.12% accuracy, 0.85 sensitivity and 0.98 specificity. This shows that the traffic patterns of an active camera infected with Mirai, launching attacks, is significantly different from its normal traffic behaviour. While launching the attack, the data frame size in upstream is relatively small (typically $\leq 200B$).

Detecting Mirai DDoS. We use $D^{\text{cam}}_{\text{m-DDoS}}$ dataset to detect DDoS attacks launched by Mirai botnets on NestCam and TP-Link IP cameras. This dataset contains signatures built from a mixture of DDoS and non-anomalous traffic from 2 different IP cameras. In this case, the anomaly detector module detects anomalies (i.e., DDoS attack by Mirai bots) with an accuracy of 98.3%, specificity = 1.0 and sensitivity = 0.90. The anomaly detector traffic observed at the targeted IP cameras is downstream and numerous (unlike that in Mirai bot). The anomaly detector recognizes that this traffic contains a large number of management frames. This is because the camera fails to respond to AP's beacons when it is busy handling attack data frames. Thus, the camera sends probe requests explicitly asking for the AP's parameter.

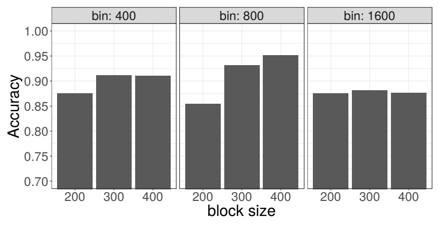
6.4 Attack Classification: WiFi Devices

The attack classification module, described in Section 3.4, is required only when an anomaly has been detected by the anomaly detector module. This module identifies attacks based on a known set of attacks. Correctly identifying the type of attack may help in developing appropriate mitigation strategies.

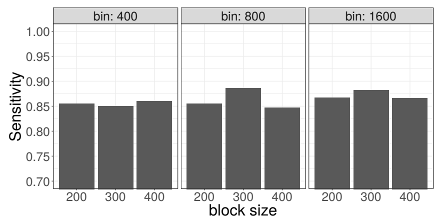
We use D^{cam}_{a}, $D^{\text{cam}}_{m-\text{DDoS}}$ and $D^{\text{cam}}_{\text{Mirai}}$ datasets to train and test a random forest classifier. The dataset is randomly segregated into training (65%) and validation (35%) set. Since $s_{\text{blk}} = 300$ and $l_{\text{bin}} = 800$ is used to build signatures for anomaly detection, the same configuration is used for attack classification. The resulting classifier identifies attacks on IP cameras with an overall accuracy of 97.4%. The confusion matrix shown in Table 5 indicates that a few slowloris attack signatures are misclassified as vulnerability check (10 signatures), and a few vulnerability check are categorized as slowloris (27 signatures). On analyzing the frame formats in detail, we found that both vulnerability check and slowloris have similar sized data frames. This is because vulnerability checks send small sized packets to different ports, and slowloris attack also sends small sized packet to form HTTP connections.



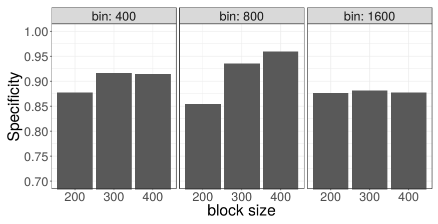
(a)



(b)



(c)



(d)

Figure 4: (a) Change in accuracy with change in $l_{\mathbf{bin}}$ and $s_{\mathbf{blk}}$ in the anomaly dataset (D_a). (b,c,d) variance in accuracy, sensitivity and specificity with change in $l_{\mathbf{bin}}$ and $s_{\mathbf{blk}}$ in the mixed dataset ($D_{\mathbf{ail}}$)

I ∖ T	t-net	f-ware	s-loris	u-flood	v-check	ddos	Mirai
t-net	14	0	0	0	0	0	0
f-ware	0	167	0	0	2	0	0
s-loris	1	0	506	0	27	0	0
u-flood	0	0	2	525	0	0	0
v-check	1	0	10	0	196	0	0
ddoS	0	0	0	0	0	36	0
Mirai	0	0	0	0	0	0	528

Table 5: Confusion matrix for identifying attack using random forest supervised learning model, T and I refer to true and identified attacks, respectively. t-net, f-ware, s-loris, u-flood, v-check indicate Telnet brute-force, firmware update, slowloris, UDP flood, and vulnerability check attacks.

The top four important features for attack classification along with their importance values are listed in Table 11 in Appendix B.3. Features belonging to the load category are more significant.

6.5 Detecting DDoS in TP-Link Bulb

Since the number of WiFi frames generated by a TP-Link bulb is lower than that in IP Cameras, we build signatures with block sizes s_{blk} of 100, 200, and 300 to detect attacks. As discussed in Section 4.4, one autoencoder is trained for every combination of l_{bin} and s_{blk}. Figure 5 shows the accuracy, sensitivity and specificity observed for different combinations of l_{bin} and s_{blk}. Results are obtained using $D_{\text{ld}}^{\text{bulb}}$ test set. A maximum accuracy of 98.8%, sensitivity of 0.78 and specificity = 1.0 is observed when $l_{\text{bin}} = 1600$ and $s_{\text{blk}} = 100$. This means that despite a few false alarms, we successfully detect most DDoS attacks. On analyzing the ROC plot, we observed that $l_{\text{bin}} = 1600$ and $s_{\text{blk}} = 100$ gives the maximum AUC of about 0.99. The error threshold E^{th} calculated with $P = 75^{\text{th}}$ percentile and $x = 0$ gives optimal sensitivity and specificity. Hence we choose $l_{\text{bin}} = 1600$ and $s_{\text{blk}} = 100$ and the corresponding autoencoder to detect attacks in TP-Link bulbs.

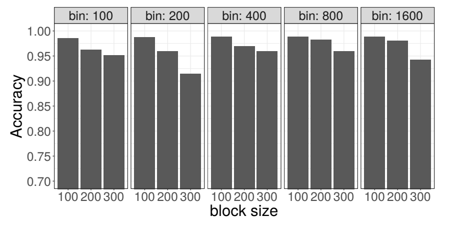
Table 10 in Appendix B.2 shows that $l_\mu(i = 0, t = \text{d-frame}, \text{dir} = s)$, i.e, mean size of sent data frames in bin b_0, which is of type Load is the most important feature having about 3.6% importance. We do not develop the attack classifier for TP-Link bulb or Phillips Hue Bulb in our current work (this is one of our potential future work where we would incorporate additional attacks to the bulbs).

6.6 Execution Time Analysis

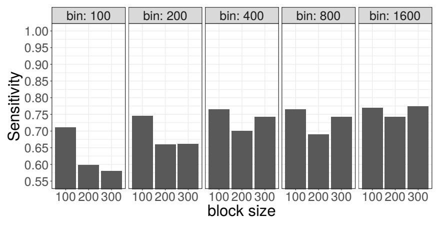
Once the attack detector and attack classifier modules are trained, the execution time of our framework is dominated by total time taken for traffic collection and feature extraction.

Traffic collection. The execution time of the traffic collector module depends on the type of IoT device being monitored. It takes about 49.6s, 2.7s and 14.22s to collect traffic for D_{ail} with $s_{\text{bk}} = 300$ from D-link IP camera, TP-link IP camera and TP-link bulb respectively. D-link camera has a greater traffic collection time as compared to others. The state of the device under observation also plays a vital role. For instance, it takes about 10-12 minutes more to sniff traffic as opposed to its active state, as D-link camera does not communicate with its server in the idle state. Hence, the traffic collection module can be a potential bottleneck.

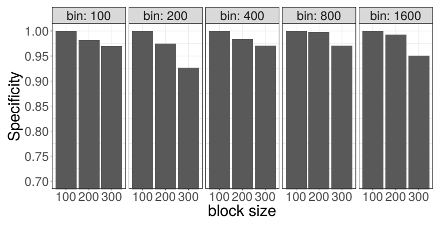
Feature Extraction. The time required to extract features primarily depends on l_{bin} as the number of features to extract can vary substantially depending on l_{bin}. The time taken for feature extraction is negligible (max of 17.5 ms) as compared to traffic collection. Figure 7 in Appendix B.4 shows the boxplot of execution time on a Linux based laptop with i7 Core processor for varying bin lengths. Since we collect a block of frames to extract features, the runtime of the framework cannot be real-time. It is primarily dependent on the traffic collection time.



(a)



(b)



(c)

Figure 5: Variance in accuracy (a), sensitivity (b) and specificity (c) with change in $l_{\mathbf{bin}}$ and $s_{\mathbf{blk}}$ in TP-Link mixed dataset ($D^{\mathbf{bulb}}_{\mathbf{ld}}$).

6.7 Anomaly Detection: Zigbee Devices

We apply our proposed framework to detect attacks on Phillips Hue bulbs that use Zigbee protocol. As mentioned in Section 5.3, signatures from Zigbee frames are built with small (<100) block sizes. Further, the data frames are not binned to extract traffic features. As in the case of TP-Link bulbs, the anomaly detector module uses a single autoencoder to detect attacks on Phillips Hue bulbs. The autoencoder is trained and tested with D^{zb}_{l} and $D^{\mathrm{zb}}_{\mathrm{al}}$.

Figure 6 shows the variation of the detection accuracy, sensitivity and specificity with different block sizes. A maximum accuracy of 99% ,sensitivity = 0.94 and specificity = 1.0 is obtained when signatures are built with $s_{\mathrm{blk}} = 40$. Increasing the block size reduces the accuracy and the sensitivity marginally. However, the specificity remains same irrespective of the change in block size. ROC curve analysis shows a maximum AUC of about 0.99 when $s_{\mathrm{blk}} = 40$. The reconstruction error threshold with $P = 95^{\mathrm{th}}$ percentile and $x = 1.5$ in Eq. 5 gives optimal sensitivity and specificity.

Table 10 in Appendix B.2 lists significant features for anomaly detection in Zigbee devices. The table highlights that the mean size of received command frames (belongs to Load type) is the most significant feature for detecting anomalies.

7 RELATED WORK

We broadly categorize related work into three categories, based on the layer at which traffic is collected: i) L1 considers features of PHY-layer traffic such as signal strength and noise ii) L2 (link-layer) which uses features such as MAC address, frame type, length and payload iii) L3+ (network-layer and higher) which considers network and transport layer features such as IP address, packet length, round trip delay, etc. We mostly focus on IDS-related work, summarized in Table 6. Our framework belongs to the L2 category.

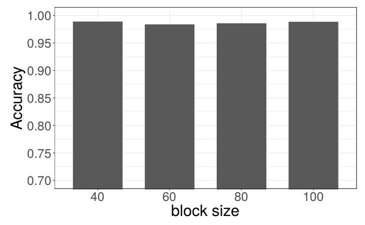
Existing works in **L1 category** analyze channel state information (CSI), frame header, transmission times and similar. For example, A. Sheth et al. [34] correlated the effect of a hidden terminal on round-trip time at the network-layer. Based on this correlation, they proposed a diagnostic tool to mitigate network performance that is caused by different attacks on PHY layer. I. Bagci *et al.* [8] used CSI to detect physical tampering on transmitting antenna or relocation of the transmitting device with more than 53% accuracy.

L2 schemes detect attacks (such as fake AP or evil twin attack) at link layer. These attacks normally affect the network infrastructure. Yu Liu *et al.* [26] investigate the use of link layer traffic in ad-hoc networks to detect routing attacks. Using NS-2 simulations, the authors evaluate and select a set of features from link layer to profile normal behaviors of mobile nodes, and then apply cross-feature analysis on feature vectors constructed from training data according to the proposed feature set. As routing schemes and ad-hoc networks are considered, most features are related to such protocols, and are not used in our setting. R. Carmo et al. [18] proposed an IDS that leverages active probing to detect malicious behavior of the nodes in wireless infrastructure. The devices are assumed to show their malicious behavior by transmitting unexpected frames without creating PHY layer attacks such as jamming attacks. In [2], the authors proposed techniques to detect *deauth* and *evil twin* attacks at MAC layer by utilizing control and management frames. The features such as time gap between management or control frames, RSSI from PHY headers are used to detect such attacks. Wireless Snort [38] is an open source tool that devises rules (such as the rule based snort [13]) to detect attacks, such as fake AP, by passively monitoring wireless traffic. It considers MAC header fields of management, control and data frames as a basic building block to generate rules to detect attacks. Since management and control frames are not encrypted, the tool can consider these frames to detect attacks.

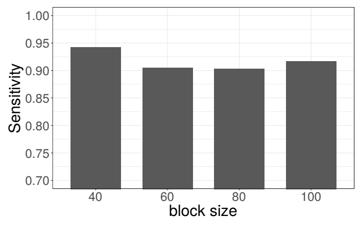
L3+ and wireline schemes analyze network or transport layer traffic [16, 21, 22]. For example, the authors of [21, 22] discuss detection of bots in a botnet based on network and transport layer traffic features (extracted with Snort IDS), application layer data is ignored. The (unencrypted) L3/L4 traffic is obtained leveraging access to intermediate routers.

The scheme proposed in [10] detects malicious electric smart meters using a passive traffic tap in the infrastructure. The scheme learns the periodicity of communication between the smart meters and its smart grid server to detect malicious meters. In [28], the authors detect attacks (such as flooding, packet drop, etc.) at the network-layer in a MANET [29] simulation environment using different supervised models. In [4], the authors present a scheme to detect malware traffic (which is using TLS). They extract flow based features (such as number of sent and received bytes and packets) from TLS traffic, and use supervised classifiers to detect attacks.

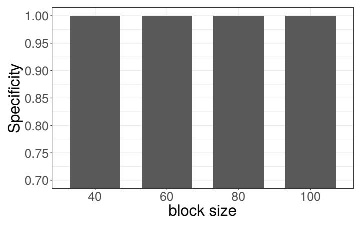
L3+/wireline using ML. In [3], the authors use neural network to recognize and identify different port scan and host sweep attacks, leveraging features from network layer and above. The scheme works in real-time: it first captures packets, extracts features using a time-delayed neural network, then recognizes host sweep or port scan attacks using a set of 2 pattern recognition neural networks, and finally classifies the attacks using a set of neural networks. This approach helped them recognize attacks such as TCP ACK port



(a)



(b)



(c)

Figure 6: (a) Anomaly detection accuracy, (b) sensitivity and (c) specificity on Philips Hue Smart Bulb (Zigbee) with different block sizes for for $D_{\mathbf{al}}^{\mathbf{zb}}$ dataset.

Related Work	Passive?	Traffic	Traffic Encrypted?	Target Attack/Anomaly
[34]	No	L1 ●	N/A	Hidden terminal, capture effect, noise, signal strength
[18]	No	L2 ●	WEP/WPA	Compromised or malicious nodes with byzantine behavior on MAC layer
[3, 24, 31]	No	L3+ ○	No	Malicious TCP connections, host sweep, port scan, DoS, probing, unsafe local or remote access,DDoS
[8]	Yes	L1 ●	N/A	Physical tampering or relocating antenna
[2, 38]	Yes	L2 ●	WEP/WPA	Deauth and evil twin attack at MAC layer, Fake access points
[10, 28]	Yes	L3+ ○	TLS	Malicious devices, flooding, packet dropping, black hole
[4]	Yes	L3+ ○	TLS	Malware traffic
[26]	Yes	L2, L3+ ●	No	Routing attacks (black hole, packet drop, etc.) in Ad-hoc networks
[21, 22]	Yes	L3+ ○	N/A	Botnet attacks (characterized by, e.g., C&C traffic and then attack others) on internal hosts using flow based features along with Snort [13] log
This work	Yes	L2 ●	WEP/WPA/WPA2	Vulnerability check, firmware update, DoS (TCP/UDP), DDoS, brute force

Table 6: Related work on anomaly or intrusion detection based on network traffic analysis. ●= wireless, ○=wireline traffic.

scan attacks, which could not be detected using snort. The authors applied their framework to DARPA 2000[25] dataset and detected all host sweep and port scan attacks with 100% recognition rate.

In [24], the authors analyze detection of DDoS attacks in the KDD cup dataset, DARPA 99 and 2000 datasets, and Conficker dataset [37] using different machine learning techniques. They found that RPB boost algorithm performed better than other techniques. RPB boost gave a detection accuracy of 96.4% (in KDD dataset), 99.4% (in DARPA 99 and 2000 datasets) and 97.2% (in confiker dataset). The FPR was 3.1, 3.7 and 3.6 in KDD, DARPA and Conficker datasets.

G. Osada *et al.* [31] compare random forest classifiers to semi-supervised variational autoencoders to detect attack traffic, leveraging features from network layer and above. The datasets used are Kyoto2006 [35] and NSL-KDD [36]. Due to the age of the datasets, most traffic is unencrypted. The authors report FPR of 0.0343, Recall of 0.953 and an AUC of 0.984 (Kyoto2006 dataset), and FPR of 0.121, Recall of 0.859 and AUC of 0.957 (NSL-KDD dataset).

While the settings of the discussed works differ from our setting in this work, we note that our overall AUC, False Positive Rate and Recall are comparable, even though we are restricted to link-layer traffic. We have shown that for IP cameras, we detect DoS, vulnerability scan, brute-force attacks UDP-flood with an AUC of 0.932, FPR of 0.06 and Recall of 0.89. We even detected DDoS on IP cameras with an AUC of 0.983, FPR of 0 and Recall of 0.9. In the case of TP-Link bulbs, we detected DDoS with an AUC of 0.988, FPR of 0 and Recall of 0.78.

Summarizing, we show that it is possible to detect such attacks without control over the network, and without requiring to have access to private application layer information of other users.

8 CONCLUSION

In this paper, we proposed a framework to detect anomalies in wireless traffic of IoT devices, and then classified the attack from a known set of attacks. We analyzed link-layer information only, and we thus argue that our system mitigates the impacts on privacy of supervised users. We proposed a unsupervised machine learning based approach (autoencoders) for anomaly detection, and a supervised based approach (random forest classifiers) for attack classification. Our experiments show that our framework detects attacks on IP Cameras with 96.2% accuracy and classified 97% of the attacks correctly. To test our system against botnet attacks, we injected Mirai on D-Link cameras. We detected attacks launched by the Mirai bot with 96.1% accuracy. In addition, we detect DDoS attacks (carried out by Mirai bots or targeting IoT devices) with 98% accuracy. We show that our framework can also detect flooding attack on Zigbee based smart bulbs with an accuracy of about 99%, which demonstrates that our method can be applied to detect various application layer attacks on different wireless technologies (and device types). A survey of related work on attack detection shows that our performance is comparable to other schemes that leverage network layer and above information from wireline communications.

REFERENCES

[1] 2016. Internet of Things and the Rise of 300 Gbps DDoS Attacks. https://www.akamai.com/us/en/multimedia/documents/ social/q4-state-of-the-internet-security-spotlight-iot-rise-of-300-gbp-ddos-attacks.pdf. Akamai Fast Forward.

[2] Z. Afzal, J. Rossebo, B. Talha, and M. Chowdhury. 2016. A Wireless Intrusion Detection System for 802.11 networks. In *Proceedings of International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. 828–834.

[3] O. Al-Jarrah and A. Arafat. 2014. Network Intrusion Detection System using Attack Behavior Classification. In *Proceedings of International Conference on Information and Communication Systems (ICICS)*. 1–6.

[4] Blake Anderson, Subharthi Paul, and David A. McGrew. 2016. Deciphering Malware's use of TLS (without Decryption). *CoRR* abs/1607.01639 (2016).

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the Mirai botnet. In *Proceedings of USENIX Security Symposium.*

[6] Daniele Antonioli, Sandra Siby, and Nils Ole Tippenhauer. 2017. Practical Evaluation of Passive COTS Eavesdropping in 802.11b/n/ac WLAN. In *Proceedings of Conference on Cryptology And Network Security (CANS).*

[7] Avalium Systems. 2017. DDOS App. https://play.google.com/store/apps/details?id=com.ddos.avaliumsystems.ddos.

[8] Ibrahim Ethem Bagci, Utz Roedig, Ivan, Matthias Schulz, and Matthias Hollick. 2015. Using Channel State Information for Tamper Detection in the Internet of Things. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*. ACM, New York, NY, USA, 131–140. http://doi.acm.org/10.1145/2818000.2818028

[9] Pierre Baldi. 2012. Autoencoders, Unsupervised Learning, and Deep Architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning (Proceedings of Machine Learning Research)*, Vol. 27. PMLR, 37–49.

[10] Robin Berthier, David Urbina, Alvaro Cárdenas, M Guerrero, U Herberg, J G Jetcheva, Daisuke Mashima, Jih-Chun Huh, and Rakesh Bobba. 2014. On the practicality of detecting anomalies with encrypted traffic in AMI. In *Proceedings of Conference on Smart Grid Communications (SmartGridComm)*. 890–895.

[11] Biondi, Philippe. [n. d.]. Scapy. http://www.secdev.org/projects/scapy.

[12] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (Oct 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[13] Brian Caswell, James C. Foster, Ryan Russell, Jay Beale, and Jeffrey Posluns. 2003. *Snort 2.0 Intrusion Detection.* Syngress Publishing.

[14] Daniel B. Cid. 2016. IoT Home Router Botnet Leveraged in Large DDoS Attack. https://blog.sucuri.net/2016/09/iot-home-router-botnet-leveraged-in-large-ddos-attack.html.

[15] Daniel B. Cid. 2016. Large CCTV Botnet Leveraged in DDoS Attacks. https://blog.sucuri.net/2016/06/large-cctv-botnet-leveraged-ddos-attacks.html.

[16] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. 2016. Analyzing Android Encrypted Network Traffic to Identify User Actions. *IEEE Transactions on Information Forensics and Security* 11, 1 (Jan 2016), 114–125. https://doi.org/10.1109/TIFS.2015.2478741

[17] Alison DeNisco. 2017. Report: 2016 saw 8.5 million mobile malware attacks, ransomware and IoT threats on the rise. http://www.techrepublic.com/article/report-2016-saw-8-5-million-mobile-malware-attacks-ransomware-and-iot-threats-on-the-rise/.

[18] Rodrigo do Carmo and Matthias Hollick. 2013. DogoIDS: A Mobile and Active Intrusion Detection System for IEEE 802.11s Wireless Mesh Networks. In *Proceedings of ACM Workshop on Hot Topics on Wireless Network Security and Privacy (HotWiSec)*. 13–18.

[19] Media Fire. 2013. AnDOSid. http://www.mediafire.com/file/1xcdnycdxuaqdiq/AnDOSid-com.scott.herbert. AnDOSid-3-v1.1.apk.

[20] MS GROUP. 2016. Packet Generator. https://play.google.com/store/apps/details?id=packetGenrator.edu.ae.

[21] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection. In *Proceedings of USENIX Security Symposium.* USENIX Association, Berkeley, CA, USA, 139–154.

[22] Guofei Gu, Phillip Porras, Vinod Yegneswaran, and Martin Fong. 2007. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proceedings of USENIX Security Symposium.* USENIX Association, Boston, MA.

[23] H2O.ai. 2016. R Interface for H2O package. https://github.com/h2oai/h2o-3.

[24] P. Arun Raj Kumar and S. Selvakumar. 2011. Distributed denial of service attack detection using an ensemble of neural classifier. *Computer Communications* 34, 11 (2011), 1328 – 1341. https://doi.org/10.1016/j.comcom.2011.01.012

[25] MIT Lincoln Laboratory. 2000. DARPA datasets. https://www.ll.mit.edu/ideval/data/

[26] Yu Liu, Yang Li, and Hong Man. 2005. MAC layer anomaly detection in ad hoc networks. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop.* 402–409. https://doi.org/10.1109/IAW.2005.1495980

[27] Rajib Ranjan Maiti, Sandra Siby, Ragav Sridharan, and Nils Ole Tippenhauer. 2017. Link-Layer Device Type Classification on Encrypted Wireless Traffic with COTS Radios. In *Proceedings of European Conference on Computer Security (ESORICS).* 247–264.

[28] Aikaterini Mitrokotsa and Christos Dimitrakakis. 2012. Intrusion Detection in MANET using classification Algorithms: The Effects of Cost and Model Selection. *Ad-Hoc Networks* (2012).

[29] A Nadeem and M P Howarth. 2013. A Survey of MANET Intrusion Detection amp; Prevention Approaches for Network Layer Attacks. *IEEE Communications Surveys Tutorials* 15, 4 (2013), 2027–2045.

[30] Nmap.org. 2017. Nmap - the Network Mapper. https://github.com/nmap/nmap.

[31] Genki Osada, Kazumasa Omote, and Takashi Nishide. 2017. Network Intrusion Detection Based on Semi-supervised Variational Auto-Encoder. In *Proceedings of European Symposium on Research in Computer Security (ESORICS).* 344–361.

[32] Jack Palevich. 2015. Terminal Emulator for Android. https://play.google.com/store/apps/details?id=jackpal.androidterm&feature=search_result.

[33] Justin Shattuck Sara Boddy. 2017. The hunt for IoT: The rise of THINGBOTS. https://f5.com/labs/articles/threat-intelligence/ddos/the-hunt-for-iot-the-rise-of-thingbots

[34] Anmol Sheth, Christian Doerr, Dirk Grunwald, Richard Han, and Douglas Sicker. 2006. MOJO: A distributed physical layer anomaly detection system for 802.11 WLANs. In *Proceedings of the international conference on Mobile systems, applications and services (MobiSys).* ACM, 191–204.

[35] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. 2011. Statistical Analysis of Honeypot Data and Building of Kyoto 2006+ Dataset for NIDS Evaluation. In *Proceedings of the Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS).* ACM, New York, NY, USA, 29–36. https://doi.org/10.1145/1978672.1978676

[36] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. 2009. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications.* 1–6. https://doi.org/10.1109/CISDA.2009.5356528

[37] UCSD. 2000. Conficker dataset. https://www.caida.org/data/passive/telescope-3days-conficker_dataset.xml

[38] Craig Valli. 2004. Wireless Snort: A WIDS in progress. In *Proceedings of Australian Computer, Network and Information Forensics Conference.* Edith Cowan University.

[39] Joshua Wright, Ryan Speers, and Ricky Melgares. 2011. ZigBee Security Research Toolkit. https://github.com/riverloopsec/killerbee.

[40] Gokberk Yaltiraklі. 2016. Low bandwidth DoS tool: Slowloris 0.1.4 rewrite in Python. https://github.com/gkbrk/slowloris.

A APPENDIX: SYSTEM IMPLEMENTATION

Table 7 lists the attacks performed on different IoT devices. Table 8 shows the total amount of traffic collected from different devices.

Device	vul. scan	brute-force	UDP-flood	slow-loris	firm-ware	DDoS	flood-ing	Mirai
Dlink Cam.	●	●	○	●	●	●	○	●
TP Link Cam.	●	○	●	●	○	●	○	○
Nest Cam.	●	○	●	○	○	●	○	○
Natat. Cam.	○	○	●	●	○	●	○	○
With. Cam.	●	○	●	○	○	○	○	○
TP Link Bulb	○	○	○	○	○	●	○	○
Phillip Bulb	○	○	○	○	○	○	●	○

Table 7: Type of attacks performed on various IoT devices. ●= attack was performed, ○=attack was not performed.

Device	frame_count(K)	size(MB)	time(hrs)
IP Camera	2060	508.40	7.63
WiFi Bulb	459	58	2.31
Zigbee Bulb	18.5	0.96	0.50

Table 8: Total amount of traffic collected.

B APPENDIX: RESULTS

B.1 Anomaly Detection: IP Camera

Table 9 shows the configuration of autoencoders corresponding to $l_{\text{bin}} = 800$ while varying s_{blk}

$\downarrow s_{\text{blk}} \setminus type \rightarrow$	Idle	Active
100	(58,29,58)	(58,29,58)
200	(63,28,63)	(63,29,63)
300	(59,29,59)	(59,29,59)
400	(59,29,59)	(59,29,59)
500	(59,28,59)	(59,29,59)

Table 9: Configurations of idle and active autoencoders corresponding to bin length ($l_{\text{bin}} = 800$) and varying block sizes (s_{blk}).

B.2 Anomaly Detection: Feature Importance

Table 10 lists features significant for anomaly detection in different device types, and their importance values.

AE	Feature	Type	Importance
IP Camera Active	$g_\mu(1, \text{data}, s)$	Gap	0.02572
	$g_\sigma(1, data, s)$	Gap	0.02345
	$g_\mu(0, data, r)$	Gap	0.02312
	$g_\sigma(1, data, r)$	Gap	0.02249
IP Camera Idle	$g_\mu(1, \text{data}, r)$	Gap	0.02478
	$g_\mu(0, data, r)$	Gap	0.02218
	$l_\mu(0, data, s)$	Load	0.02209
	$g_\sigma(1, data, r)$	gap	0.02189
TP-Link bulb	$l_\mu(ctrl, r)$	Load	0.03667
	$\kappa(data, s)$	Census	0.03524
	$l_\sigma(data, s)$	Load	0.03512
	$\kappa(ctrl, x)$	Census	0.03496
Phillips Hue bulb	$l_\mu(data, r)$	Load	0.05054
	$l_\mu(data, x)$	Load	0.04790
	$l_\mu(any, x)$	Load	0.04711
	$\kappa(cmd, r)$	Census	0.04520

Table 10: Feature importance for anomaly detection. Column *AE* indicates autoencoders for IP Camera (idle and active with $l_{\text{bin}} = 800$ and $s_{\text{blk}} = 300$), TP-Link bulbs ($l_{\text{bin}} = 1600$ and $s_{\text{blk}} = 100$) and Phillips Hue bulbs($s_{\text{blk}} = 40$).

B.3 Attack Classification: Feature Importance

Table 11 lists important features for attack classification along with their importance values.

B.4 Execution Time Analysis

Figure 7 gives information about the time taken by the feature extractor to build one signature from 500 frames.

Feature	$l_\mu(0, data, x)$	$l_\mu(0, data, r)$	$\kappa(ctrl, s)$	$r(ctrl, s, s_{\text{blk}})$
Type	Load	Load	Census	Ratio
Imp.	17.78	10.63	6.224	5.732

Table 11: Feature importance for attack classification.

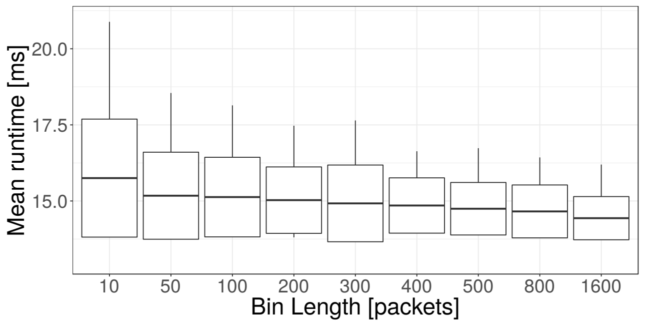


Figure 7: Average and standard deviation of runtime taken for feature extractor to extract a single signature from 500 frames.