IoT Botnet Detection via Power Consumption Modeling

Woosub Jung, Hongyang Zhao, Minglong Sun, Gang Zhou*

Computer Science Department, College of William and Mary, United States

Abstract

Many IoT botnets that exploit vulnerabilities of IoT devices have emerged recently. After taking over control of IoT devices, the botnets generate tremendous traffic to attack target nodes. It is also a threat to the smart health area since they have used IoT devices more and more. To detect the malicious IoT botnets, many researchers have proposed botnet detection systems; however, these are not easily applicable to resource-constrained IoT devices. Moreover, since the botnet's early stage makes marginal differences in terms of traffic, it is hard to detect when they first attack the victim nodes. However, we observe that the IoT botnets generate distinguishable power consumption patterns. Thus, we aim to classify whether the IoT device is affected by malign behaviors through power consumption patterns so that we can protect the healthcare ecosystem from the malicious IoT botnets.

We propose a CNN-based deep learning model that consists of a data processing module as well as an 8-layer CNN. Prior to applying the CNN model, we segment and normalize the collected power consumption data to help our CNN model to achieve higher accuracy. The 8-layer CNN classifies the processed data into four classes including a botnet class, which is our primary target. To demonstrate the performance, we run self-evaluation, cross-device-evaluation, leave-one-device-out, and leave-one-botnet-out tests on three common types of IoT devices, which are Security Camera, Router, and Voice Assistant devices. The self-tests achieve up to 96.5% classification accuracy whereas the cross-evaluation tests perform about 90% accuracy. Leave-one-out tests also introduce higher than 90% accuracy for botnet detection.

Keywords: IoT Botnet Mitigation, Power-based Botnet Detection Method, Convolutional Neural Network Modeling

1. Introduction

Internet of Things (IoT) has grown rapidly over the past several years. Recently, anything connected to the Internet is considered an IoT device. According to [1], the number of IoT devices will reach about 30 billion by 2020. As the market grows, a number of IoT devices have been applied to the medical environment [2]. Accordingly, the number of attacks that exploit the vulnerabilities of IoT devices has increased as well, which could be critical in keeping patients' data secure. However, since an IoT device is designed for a simple purpose with less processing power and memory like a thermostat and a light bulb, manufacturers and researchers have struggled with satisfying its needs of security [3, 4]. In the meantime, adversaries have taken advantage of its vulnerabilities.

In fall 2016, a botnet named Mirai exploited a number of vulnerable IoT devices to build a huge attack vector. After the botnet accumulated the immense network, it generated 600GB of traffic per second only by using the infected IoT bots like security cameras and routers [5]. Since then, many variants of Mirai have emerged to target the vulnerabilities of IoT devices. Many researchers have also proposed to detect those botnets. Despite such efforts, the problem is that detecting these IoT botnets when they first get into a victim node is not easy. This situation eventually causes a huge volume of botnet attacks, which could lead to a serious issue in a medical field because IoT devices have also been used in a hospital. First of all, a router is now one of the most common devices in any building system. It should be

*Corresponding author

Email addresses: wsjung@cs.wm.edu (Woosub Jung), hyzhao@cs.wm.edu (Hongyang Zhao), msun@cs.wm.edu (Minglong Sun), gzhou@cs.wm.edu (Gang Zhou)

secure against any malicious attack to keep patients' personal information. A security camera is also considered an important device in a hospital. For some hospitals, they monitor patients at risk for any type of self-injury such as falls. Moreover, in the near future, a voice assistant could assist patients to get help easily from nurses or doctors. In other words, all the IoT devices we address in this paper are needed to be more secure against botnet attacks. This motivates us to propose a method that can detect malicious behaviors at the very beginning stage at IoT devices.

To tackle this situation, many researchers have proposed IoT Botnet Detection System (BDS) or Network Intrusion Detection System (IDS) [5, 6, 7, 8, 9]. Some systems have applied Machine Learning (ML) techniques to detect unknown botnet attacks [10, 11, 12]. Other researchers have also proposed malware detection systems based on power measurement on mobile systems [13, 14]. However, it is hard to detect when they first get into the victim nodes during the propagation period. This situation inspires us to examine the characteristics of Mirai and its variants and detect them at an IoT device at the very early stage. We figure out that when the well-known IoT botnets attack, the power usage of the target device is distinguishably changed even though the traffic is marginal. Therefore, we aim to classify IoT device behaviors including the botnet attacks based on power consumption data. Furthermore, since our model is designed to measure power consumption outside an IoT device, it is also able to detect the malicious botnets even when the IoT device is being compromised, which is the first attempt to the best of our knowledge.

In this paper, we build our model to learn the characteristics of the well-known IoT botnets and detect malicious behaviors via power consumption data. Our proposed method consists of a data processing module as well as an 8-layer Convolutional Neural Network (CNN) model. During the data processing, we first segment power consumption data with a sliding window and normalize it before we feed it into the CNN. By modeling the 8-layer CNN, our method is able to sense subtle differences in power consumption patterns when the IoT botnets attack.

We evaluate our classifier on three common types of IoT devices, which are Security Camera, Router, and Voice Assistant devices. To evaluate the performance, we perform self-evaluation tests on our datasets and achieve up to 96.5% classification accuracy whereas we reach about 90% accuracy on a cross-evaluation test. Leave-one-out tests confirm the robustness of our system. In these tests, the two-class classification results are around 90%, and *F1-Measure* values are around 85%. As a result, we achieve the following contributions:

- We exploit power consumption data to monitor and diagnose malicious botnet behaviors on IoT devices.
- We model a deep learning-based method to sense subtle differences of power consumption data.
- We evaluate our model on IoT devices achieving up to 96.5% accuracy of detecting the IoT botnets.

The rest of the paper is organized as follows: Section 2 provides motivation and a threat model. In Section 3, we introduce our datasets. In Sections 4 and 5, we propose our model that includes data processing and an 8-layer CNN model. The performance evaluation is presented in Section 6. Section 7 gives some discussions and future works. Section 8 addresses related work. Finally, we conclude the paper in Section 9.

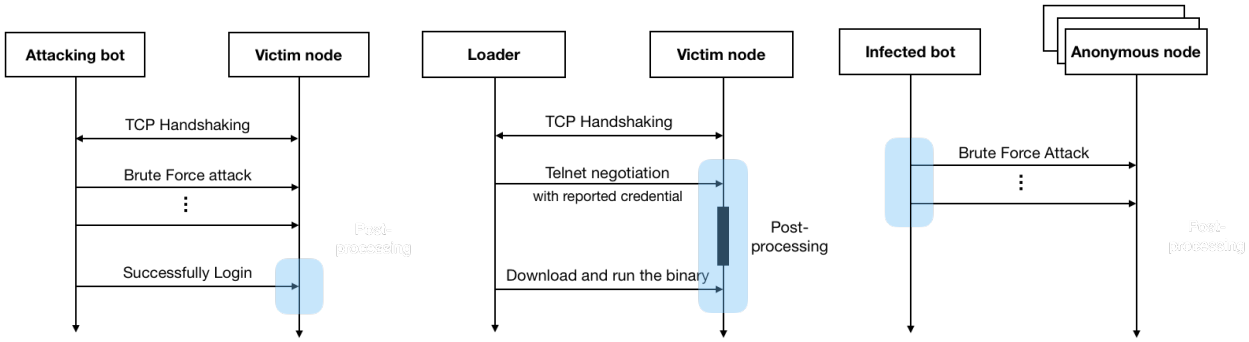
2. Motivation and Threat Model

In this section, we introduce some background knowledge and our motivation as well as a threat model.

2.1. Mirai and its Variants

In fall 2016, a popular security website was hit by a Distributed Denial of Service (DDoS) attack that broke all previous records in this field. The website received 600GB of traffic per second only by IoT bots. This famous attack is called Mirai. Since then, a number of variants of Mirai against IoT devices have emerged. Interestingly, many attackers have used IoT devices for their attacks despite decreased processing power and memory. This is because they are more vulnerable, and the number of IoT devices has rapidly increased.

Mirai uses an insecure connection like telnet to get into a victim device with well-known account/password combinations. In fact, this kind of scanning occurs every day, and the amount of scanning traffic is not significant; thus, it is hard to perceive the attack at the beginning stage. However, after an incubation period, a huge number of IoT bots receive commands from the Control and Command server (C&C server) and attack target nodes simultaneously. By then, it is too late to defend against the attack. Although many researchers have proposed IoT botnet detection systems, most focus on analyzing botnet traffic patterns. However, since this kind of approach requires more computing



(a) Intrusion Case (Scanning) (b) Intrusion Case (Loading) (c) Extrusion Case (Scanning again)

Figure 1: Target Procedures of Our Classifier

The blue rectangle areas represent the target procedures that we need to detect

resources and is commonly deployed at core network servers, it is not applicable to resource-constrained IoT devices. Therefore, we need to detect IoT botnet behaviors on IoT devices.

As shown in Figure 1, Mirai and its variants have common procedures when they try to access into IoT devices. Those cases are the target scenarios that we want to detect. First, a victim node is scanned and invaded by the attacking bot as illustrated in Figure 1a. Even though IoT devices are being scanned every day, we figure out that the botnet power consumption patterns are distinguishable from scanning or manual access. In Figure 1b, after the attacking bot logs in successfully, the botnet loader accesses the victim device with the obtained credential information from the scanning procedure and performs post-processing jobs. For example, in Mirai, once the loader gets access, it kills suspicious processes, finds a writeable directory, checks its architecture type, and uploads the architecture-specific Mirai binary to keep scanning other devices. Table 1 summarizes the post-processing jobs of Mirai and its variant botnets. Lastly, Figure 1c describes a third case in which an infected bot runs the Mirai binary to start scanning as the original attacking bot does.

2.2. Threat Model

In our threat model, we consider that an adversary is capable of conducting large-scale IoT attacks, such as Mirai and its variants, to compromise IoT devices and launch DDoS attacks. We do not assume specific types of IoT devices the adversary targets. There are two possible ways that the adversary can attack. 1) Vulnerabilities of the device that would measure power consumption: Since our model could be deployed at another type of IoT device like Smart Plug, attackers could first try to compromise our device. 2) Complicated post-processing jobs. Adversaries might

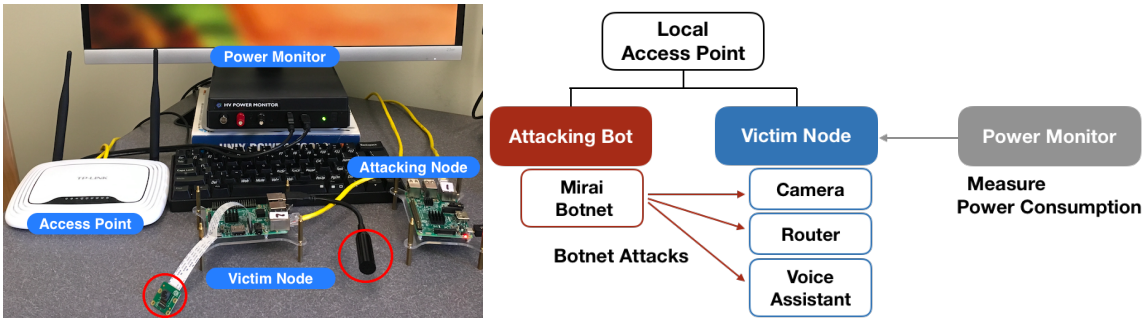
Table 1: Summary of Mirai and its Variants

IoT Botnet	**Reported Day**	**Target Devices**	**Intrusion Method**	**Post-processing Jobs**
Mirai	Sep. 2016	IP Cameras	Brute Force Attack	Upload a binary through Internet
Hajime	Oct. 2016	Any device	Brute Force Attack	Upload a binary through Internet
BrickerBot	Apr. 2017	Any device	Brute Force Attack	Corrupting a device using Linux commands
IoT Reaper	Oct. 2017	Routers, IP Cameras, Network Storage	Exploit known vulnerabilities	
Masuta	Jan. 2018	Routers	Brute Force Attack	Upload a binary through Internet
Sora	May 2018	Routers, CCTV-DVRs	Brute Force Attack	Upload a binary through Internet

perform complicated jobs that have different power patterns. For instance, downloading multiple binaries, connecting to multiple servers or rebooting the device can generate longer and more complicated power traces. To address the adversary models above, we assume that the device that would monitor power consumption does not allow inbound traffic of remote access. Our model could be directly connected to an IoT device by a power socket to measure power consumption data. For the second case, since we use segmented datasets and feed them to a CNN model, those segmented data from the different patterns can also be trained as botnet instances. As long as power signal data is noticeable enough to label, our system is able to learn and detect even more complicated patterns.

3. Data Collection

In this section, we demonstrate an experimental setup and show our data collection procedure and the datasets we collect.



(a) Hardware Deployment (b) Overview of Logical Deployment

Figure 2: Experimental Setup

3.1. Experimental Setup

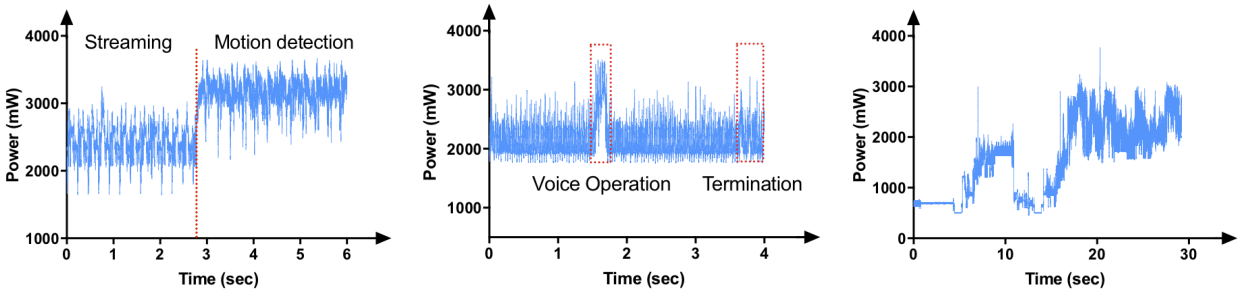
To build our environment, we first configure a local network with two identical Raspberry Pi 3 devices as IoT devices. Raspberry Pi 3 is one of the most popular devices for IoT prototyping; thus, we believe that building different types of IoT service on the Raspberry Pi 3 provides characteristics of heterogeneous IoT devices. As shown in Figure 2a, the two devices are connected to a local access point, which is not connected to the Internet. We generate malicious traffic only within the local network. Among the two devices, one represents an attacking node, which runs the Mirai binary. The other one acts like a victim node to run each IoT service application. There are two extra hardware modules, a microphone and a camera, attached to the victim node for the Voice Assistant and Camera service, respectively. For the Router service, we do not need any hardware module. Lastly, we use an off-the-shelf power monitor device connected to the victim node to measure its power consumption. For the Mirai binary, we download the Mirai open source from Github and build it on the attacking bot.

After configuring the local network, we modify the scanner and loader modules of the Mirai botnet on the attacking bot; we manually assign the destination IP to our victim's address. Since we keep the existing procedures and only change the target address right before it scans, we assure that the behaviors of Mirai do not change. In fact, in view of the victim node, it is the same as the Mirai botnet tries to get into it using a brute force attack as shown in Figures 1a and 1b.

For the victim node, we choose three common types of IoT services as follows: Security Camera, Router, and Voice Assistant. As we observe in Table 1, IoT cameras and routers have been major targets of the IoT botnets. Besides those two types, we also include a Voice Assistant as a third type of our IoT devices. It can be a potential target of IoT Botnets in the future since it has become one of the most popular IoT services. We describe how we prepare each IoT service in the following subsections.

3.1.1. *Security Camera*

The majority of target devices in the Mirai botnet is an IoT security camera. Since it tends to connect to public networks directly, it is more vulnerable than other types of devices. Furthermore, after the Mirai botnet emerged, many variants of Mirai have exploited the vulnerabilities of IoT cameras as described in Table 1. Therefore, in our experiments, we set up web camera modules on the victim node. We download a popular camera open-source, MotionEye. This open-source includes a motion detection module as well as a basic camera streaming module. Once it detects an object's motion, it captures live photos and stores them into a local file system. Figure 3a illustrates one power reading sample from the IoT camera service.



(a) Camera streaming with Motion Detection (b) Operation of Voice Assistant (c) Device Rebooting

Figure 3: Power Consumption Patterns of IoT services

3.1.2. *Router*

Another common type of target device is a router. Although it is not considered a standard type of IoT device, it is reachable from anywhere through remote access protocols, i.e., ssh, telnet. As shown in Table 1, attackers have exploited vulnerabilities of old routers as their bots. Therefore, we build a router environment on the victim node by enabling existing Linux daemons, which are hostapd [15] and dnsmasq [16]. While it is connected to the Internet, it provides the functionalities of a WiFi access point.

In terms of traffic patterns, according to the authors in [17], the dominant traffic of the Internet comes from HTTP followed by FTP and DNS. By then, we generate HTTP, FTP, and DNS traffic following the common ratio of Internet traffic. By doing this, we simulate the router behaviors while Mirai botnet attacks the victim node. We do not describe any figures about power consumption patterns since the traffic varies.

3.1.3. *Voice Assistant*

Thus far, voice assistant devices have not been compromised by malicious IoT botnets. However, it has been more common for the last several years, and a number of manufacturers have released their products. As long as it is connected to the Internet, it can always be jeopardized by botnet attacks. Due to this, we also include a Voice Assistant to our test types.

We also use one common open-source for the voice assistant service, Google AIY project, which is installed on the victim node. To initiate the service, a user needs to say "Okay, google". Its power consumption mainly comes from voice-processing computing followed by HTTP traffic. Figure 3b demonstrates one sample power consumption pattern when it is processing voice operations. In this figure, we notice that there is no significant power consumption pattern between voice operation and termination. This is when voice responses play, which might be played locally.

3.2. *Datasets*

In this section, we describe how we collect our datasets and explain the datasets we collect. We collect a huge amount of power consumption data from the selected IoT services, which we will train and test later in Section 6.

Table 2: Summary of Our Datasets

Class		Description	Number of Instances	Total number of Instances
IoT Service	Camera	Streaming Service	1,000	2,000
		Motion Detection	1,000	
	Router	HTTP Traffic	1,800	2,000
		FTP Traffic	100	
		DNS	100	
	Voice Assistant	When operating Voice Operations	2,000	2,000
Reboot		When system is Rebooting	2,000	2,000
Idle		When IoT service is not Running	2,000	2,000
Botnet		When system is Idle	1,000	2,000
		When IoT service is Running	1,000	

3.2.1. Methodology

In order to collect power consumption data from the IoT devices, we use Monsoon power monitor [18], as shown in Figure 2a. This allows us to measure power consumption data with $5kHz$ sampling rate. In our experiments, it is connected to one of the Raspberry Pi devices so that it measures power consumption when the IoT device is being compromised or running specific applications. However, note that it cannot be widely deployed because it is expensive and has a large form factor; thus, we discuss our future plan to implement a power monitoring module in Section 7.

With the power monitor hardware, we need adequate datasets for the better performance of our CNN model. First of all, we collect 2,000 power consumption instances while the Security Camera, Router, or Voice Assistant service is running. Those data make up our first class, an *IoT* service. The second class is *Reboot*. After we manually reboot the victim device, we measure power consumption data until the device becomes stable. The third class is an *Idle* class, which is collected when the victim device is not running any service. Lastly, we collect a *Mirai* class under the two cases: 1) The victim device is being attacked when it is *Idle*. 2) The victim device is being attacked when it is running one of the three IoT services. In the *Botnet* class, one instance represents one single attack by the Mirai botnet. We also collect *Botnet* instances from *Sora* and *Masuta* botnets and compare the results in Figure 11. For the four classes, the length of each instance is 1.5 seconds. Thus, the dataset for each class has 3,000 seconds of power consumption data, which consists of 15 million data points each. We will discuss the length of each instance in Section 4.1.

3.2.2. Collected Datasets

Table 2 summarizes our datasets. In detail, for the camera service dataset, we collect half of the data when it is streaming videos, which mostly consists of HTTP traffic. Another half of the data is collected when the motion detection service is running. In this case, many file system accesses occur and impact power consumption patterns as shown in Figure 3a. Next, our router dataset consists of 90% HTTP, 5% FTP, and 5% DNS traffic based on the real-world statistics [17]. We also perform voice operations to measure power consumption data from the Voice Assistant service. We use a common questionary [19], which consists of one-hundred trivial questions. With the questionary, we initiate voice transactions and collect the responses over again. The length of the queries and responses varies, and they generate different power traces; thus, the collected dataset mimics real-world usage. Overall, we have three IoT service datasets. We also collect the same number of instances for the *Idle* class and the *Reboot* class. We choose the *Reboot* class because it fluctuates excessively while the device is rebooting as shown in Figure 3c, which makes our CNN more durable. For the *Mirai* class, we first collect a dataset and apply a threshold-based segmentation method to extract Mirai instances, which we will describe in Section 4.1.

Consequently, we collect 8,000 instances from the four classes, which are *IoT service*, *Reboot*, *Idle*, and *Botnet* classes, before we feed them into the CNN. Since our datasets are collected on different days, and especially the router dataset includes wide-area traffic, our results show its tolerance against sudden communication patterns. All those instances and labels for the CNN training will be open to the public for download.

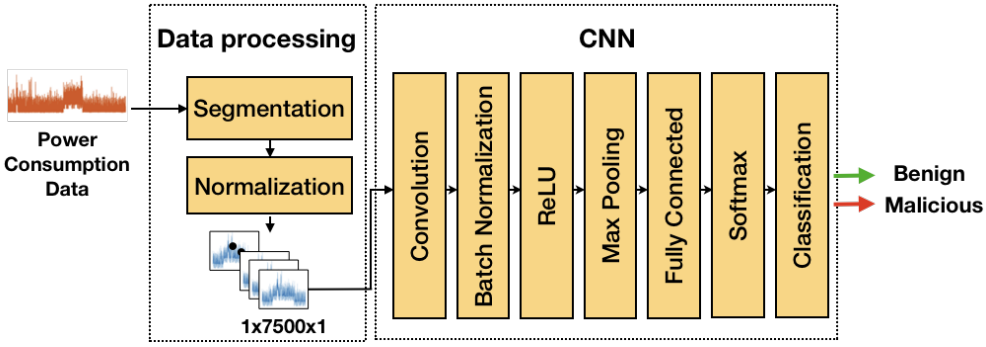


Figure 4: Data Flow of Our Proposed Model

4. Data Processing

In this section, we explain data processing and discuss its impact on our design before we feed data into the CNN model. The data flow of our model is illustrated in Figure 4. It consists of two main modules: 1) *Data Processing*. This module segments and normalizes the collected sensing data before it feeds them into the CNN model. 2) *CNN model*. The processed data is fed into the CNN-based classifier, which we will describe in the next section. Our CNN model classifies those input data as either a benign behavior or a malicious attempt.

4.1. Segmentation

We segment power sensing readings to help our CNN model train well. In Figure 5, we plot the time distribution of Mirai and its variants when they successfully get into victim devices and perform the post-processing tasks as we explained in Section 2. When they get in and perform the post-processing jobs, it usually lasts 1.4 seconds and 90% of the attacks are less than 1.5 seconds. Like the Mirai botnet, its variants have similar distribution patterns since they use similar modules. Thus, we use a sliding window of 1.5 seconds to capture the power readings. Since the sampling rate of the power monitor we use is $5kHz$, there are 7,500 data points within a single sliding window, which represents 1.5 seconds of power consumption data. Recall that the length of the instance could vary. However, since we use segmented datasets and feed them to a CNN model, those various segmented data are also trained as botnet instances. Thus, as long as power signal data is noticeable enough to label, our model is able to learn and detect even more complicated patterns.

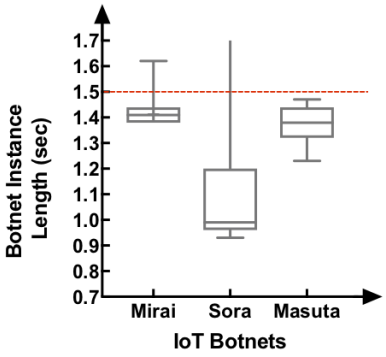
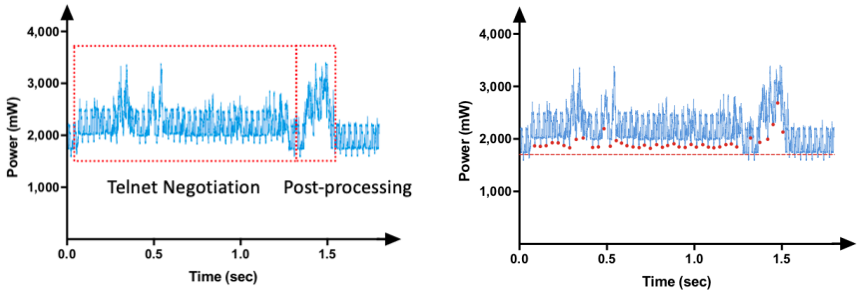


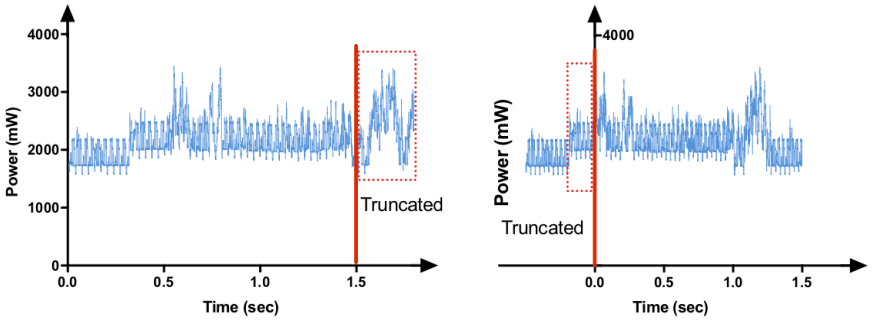
Figure 5: Distribution of Mirai and its Variants

In Figure 6a, we observe that the power consumption data is divided into two parts when Mirai attacks. One comes from telnet negotiations, which is the majority part within one sliding window, and the other part is due to post-processing jobs. After segmenting the power readings into each instance of 7,500 data points, we need to label them correspondingly. We use a threshold-based method to extract Mirai instances and label them. As shown in Figure 6b, we compute valley points with minimum-distance of 150 data points from a segmented window. Since the size of the sliding window is 7,500, there could be 50 valley points at the most within one sliding window; there are 45 valley points on average from our dataset. In the Mirai attacks, the number of valley points that are above $1,700mW$ is around 42. It confirms that most of Mirai attack instances lay around 1.4 seconds as shown in Figure 5. However, when we segment data using the sliding window method, it is less likely to have an intact Mirai instance within one sliding window in a real-time scenario. In Figure 7, there are two possible shapes of incomplete power consumption patterns for the Mirai attacks. As shown in Figures 7a and 7b, sometimes you only observe a part of the Mirai instances; however, it is still noticeable that it is a part of the Mirai attack case like the one in Figure 6a. Therefore, we use the sliding window with 0.5 seconds overlap. If the number of minimum-distance valley points



(a) Sample Mirai Attack Instance (b) Threshold-based Labeling

Figure 6: Data Segmentation on Mirai Attack Power Traces



(a) Truncated Instance at the back side (b) Truncated Instance at the front side

Figure 7: Various Mirai Instances after Segmentation

within one sliding window is more than 35, which is longer than 1.2 seconds, we label it as a Mirai instance. By doing this, we successfully collect all the Mirai instances. For all other normal behaviors, we just segment them with the 1.5 seconds sliding window and label them as they are.

4.2. Normalization

Before we feed the segmented data into our CNN, we take a normalization that makes our dataset have mean value zero and standard deviation one. It simply means that we limit the range of our input data. Considering that the range of our power consumption dataset distributes variously depending on services, the range of the features that the CNN learns are also widely spread, which is bad for our CNN to train data well and fast.

More specifically, we use Stochastic Gradient Descent with Momentum (SGDM) to update weights and biases in our CNN. During the training, we locate an optimal parameter vector that minimizes the cost function. In other words, the CNN updates the weights and biases at each layer so that our model reaches the optimal point. Eq. (1) shows how our CNN updates the parameter vector. The learning rate affects the step size of gradient descent in an optimization problem. Without normalization, our CNN either overshoots an optimal point or stops at a local minimum point, which would eventually lead to poor performance. This is because the optimal learning rate is likely to be different for each dataset. However, when we normalize the datasets first, our model performs much better. We will discuss its impact in Section 6.

$$\theta_{l+1} = \theta_l - \alpha \nabla E(\theta_l) + \gamma(\theta_{l+1} - \theta_l), \tag{1}$$

where θ is the parameter vector, l is the iteration index, α is the learning rate, $E(\theta)$ is the loss function, and γ is the momentum term.

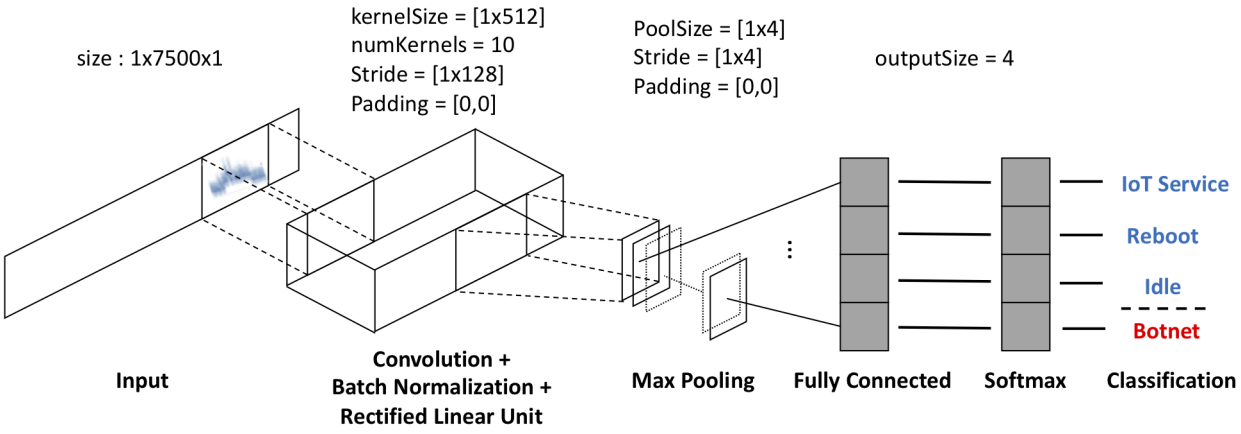


Figure 8: CNN Design of Our model

5. 8-Layer CNN Design

In this section, we design an 8-layer CNN for classification of malicious behaviors. We choose a CNN to learn power patterns because it takes a number of features to learn power patterns, which makes our model tolerant of various cases. Recurrent Neural Network (RNN) could help streaming data but is mostly popular in natural language applications. Instead, CNN is more widely used in sensor streaming applications. Furthermore, other shallow learning techniques such as SVN or random forests have fewer parameters so that it is simpler than deep learning approaches and could be overfitted. Our evaluation results demonstrate the effectiveness and robustness of the CNN approach.

Note that, in our system, each instance is a one-dimensional consecutive subsequence of power consumption data. Thus, our CNN model takes one-dimensional input data. We tune the hyper-parameters of the 8-layer CNN to emulate power consumption in various situations as discussed in Section 6. We describe the design factors of each layer in the following subsections. For Rectified Linear Unit (ReLU), Fully Connected and Softmax layer, we do not have any special requirements; hence, we adopt a widely used design choice in CNNs [20, 21].

5.1. Input Layer

The Input layer takes segmented and normalized data as described in the previous section. Then, it prepares one-dimensional input for the convolution layer. Recall that each input instance represents 1.5 seconds of power consumption data, which consists of 7,500 sequential data points. Thus, the input size of our CNN model is (1 x 7500 x 1). Consequently, the final goal of our CNN is to classify whether each instance contains malicious power consumption patterns generated by IoT botnets or not.

5.2. Convolution Layer

Our CNN has only one Convolution layer. Considering the tradeoff between computing resource and performance, one convolution layer works well for our datasets without consuming many resources. The proposed CNN uses ten one-dimensional (1 x 512) kernels, and the stride size is 128. For image classifications, it is common to take small kernel size to extract features since every single pixel could be considered important. In our design, however, we do not operate any downsampling, and the sampling rate is pretty high. Thus, small kernel size is not helpful to learn features. Instead, we choose bigger kernel size so that it helps to learn time-related features. Furthermore, the stride size means the gap between two neighboring convolution operations; hence, we also choose bigger stride size. In our study, the 512 kernel size represents about 0.1 seconds of power consumption data since the sampling rate is $5kHz$. Therefore, the convolution layer computes the dot product between the 0.1 seconds of power consumption data and the kernel. When we use smaller size of kernel or stride, it learns our dataset very slowly, and the results are not good enough. It is because less than 0.1 seconds of data do not contain much meaningful information in terms of power changes. We will discuss the impact of the kernel and stride size in Section 6.

5.3. Batch Normalization Layer

We put the Batch Normalization layer before we apply feature maps to an activation function in the ReLU layer. Although we have already normalized our dataset during the data processing before, the batch normalization layer is still able to improve the performance of our design. The limitation of the normalization as preprocessing is that internal covariate shift could slow down our training while a CNN is going through the layers [22]. However, this layer normalizes a dataset at each mini-batch; thus, the effect of normalization still remains during the training procedures, which also helps avoid overfitting. Eq. (2) represents the normalized activation. Using this function, the batch layer normalizes the input x_i over a mini-batch. The output of the batch normalization is shown as Eq. (3).

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{{\sigma_B}^2 + \epsilon}}, \tag{2}$$

where μ_B and σ_B are the mean and variance of the mini-batch [22].

$$y_i = \kappa \hat{x_i} + \rho, \tag{3}$$

where κ is the scale factor, ρ is the offset, and $\hat{x_i}$ is the normalized activation in Eq. (2) [22].

5.4. Max Pooling Layer

The purpose of the Pooling layer is to avoid overfitting by downsampling the extracted features. Since the size of our dataset after the convolution layer is still huge, we need to reduce the number of features to classify them properly. Our model uses max pooling method with a (1x4) filter, and the stride size is four. By reducing the size of our dataset, we decrease training time and also avoid overfitting since we drop out three-quarter of data features. This is why we do not use a dropout layer to reduce overfitting separately.

5.5. Output Layer

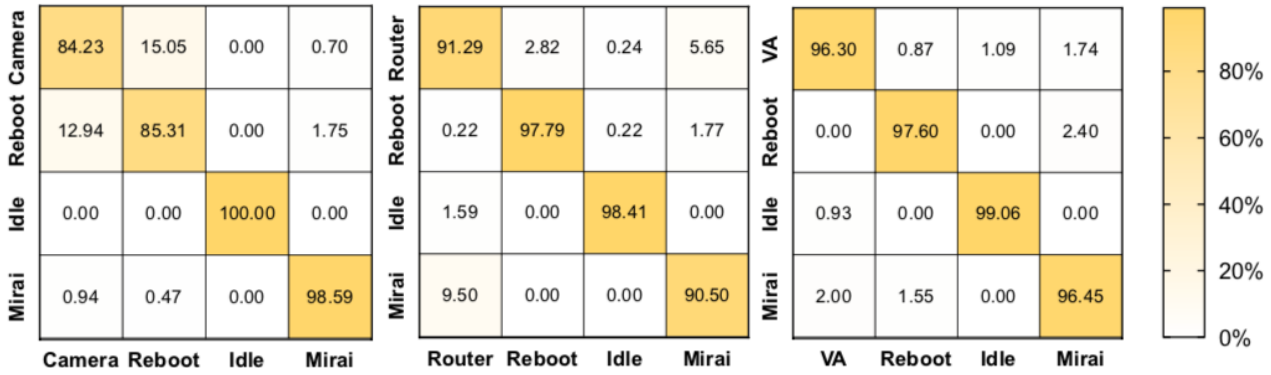
The Output layer has four classes, which are *IoT service*, *Reboot*, *Idle*, and *Botnet* classes, to represent the behaviors of each IoT device. As a matter of fact, we only need two classes to detect malicious behaviors; one for normal behaviors, and the other one for malicious behaviors, which would be enough for a classifier to make a right decision against IoT botnets. However, instead of modeling two-classes CNN, we propose four-classes CNN to show its potential uses and validate the performance of our model. Our evaluation results demonstrate that it achieves pretty high accuracy for the four-classes classification. We will discuss it in detail later in Section 6.

6. Evaluation

In this section, we perform several evaluations to demonstrate the performance of our model. With the datasets on the three devices as described in Table 2, we perform data processing and classification in MATLAB on a MacBook laptop with 1.6GHz dual-core and 8GB RAM. Most of our tests are performed under the intrusion attacks as described in Figures 1a and 1b, which we explain the results in Section 6.1. For the extrusion cases as shown in Figure 1c, we discuss the results in Section 6.2. Mostly we focus on classification accuracy, but we also measure testing time and other metrics and discuss them in Section 6.3. The impact of design factors are considered in Section 6.4.

6.1. Intrusion Detection Accuracy

We conduct three different tests for intrusion detection. All of them have four classes that we need to categorize correctly.



(a) Camera (b) Router (c) Voice Assistant

Figure 9: Confusion Matrix of the Self Evaluation Tests

6.1.1. Self-Evaluation Test

We run 5-fold cross-validation for each type of IoT device. For each test, we use the datasets as described in Table 2. For example, in the Camera self-test, we use 2,000 instances of Camera service for the first class and 2,000 instances attacked by Mirai for the Botnet class in addition to the same number of Reboot and Idle instances. With the given dataset, we randomly partition each class into five subsets; then, we train on four subsets and test on the remaining one. We repeat this procedure five times so that each subset is used for validation. In terms of accuracy, which we define as the portion of correctly classified instances among all the testing instances, each of the tests on the three devices performs well. Since, in this test, the CNN trains and tests on the same datasets, it is supposed to achieve the best performance among all the tests for intrusion detection. As expected, the overall accuracy of the classification results for each device is 90.6%, 95.0%, and 96.5% for the *Camera*, *Router*, and *Voice Assistant* devices, respectively. Considering that our systems targets to detect IoT botnets, we can simplify our four-class classifier to two-class classifier that determines whether it is a *Botnet* class or not. Then, this two-class classification accuracy become 97.7%, 92.1%, and 96.0%.

In Figure 9, we plot confusion matrices for the three device types. Figure 9a shows the classification results on the camera dataset. The overall accuracy is 90.6% whearas the accuracy of the *Mirai* class is about 99%, which means our model classifies whether the camera is attacked by the Mirai botnet very well. We also compare the accuracy of the *Mirai* class with the overall classes for all the tests in Figure 10. In general, the results for the *Mirai* class outperform the other classes. Moreover, only the *Reboot* and *Camera* classes exhibit little lower accuracy, which is about 84% and 85%, respectively. This is because those two classes generate more fluctuating power signals than the other two classes as shown in Figures 3a and 3c. In Figure 9b, all the classes achieve higher than 90% accuracy. About 10% of the Mirai instances are misclassified as the *Router* class. This is because the randomness of Router traffic might impact the classification performance as we mentioned in Section 3.1.2. We observe the same phenomenon during the leave-one-out tests in Section 6.1.4. For the Voice Assistant self-test, we achieve the best performance: the accuracy of all the cases is higher than 95% as shown in Figure 9c.

In order to demonstrate the validity of our classifier against different IoT botnets, several other self-evaluation tests are performed with variants of Mirai, which are Sora and Masuta botnets. They emerged after Mirai had occurred and used similar approaches as we illustrated in Table 1. Accordingly, we also observe that they have similar time distribution with the Mirai botnet as illustrated in Figure 5. We only conduct self-tests on the Security Camera device to see the robustness of our detection design against other IoT botnets. Doing so demonstrates that our system is not overfitted to a specific botnet. Despite Sora and Masuta's slightly different post-processing jobs and the length of its instances, the accuracy of the tests is still quite good, achieving 95.3% and 93.6%, respectively. These results are shown in Figure 11.

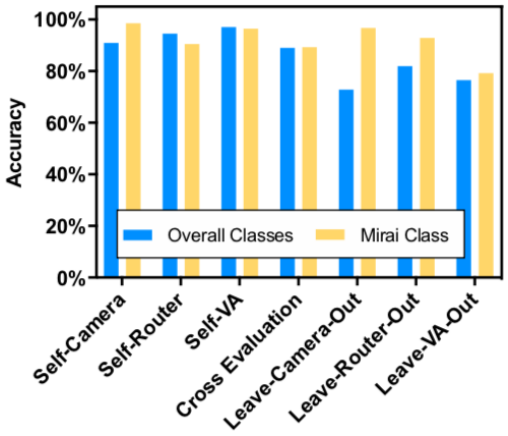


Figure 10: Accuracy of the Mirai and Overall Classes

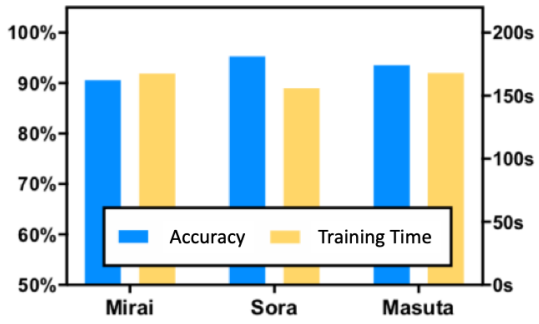


Figure 11: Comparisons between Mirai and its Variants

6.1.2. Cross-Evaluation Test

In this section, we perform another self-tests. Instead of training and testing on each device dataset, we first merge data from the three device datasets into one bigger dataset as an *IoT service* class, which is our first class in this test. This master class covers all kinds of power consumption patterns generated by network traffic, filesystem access, voice processing, etc. For the *Mirai* class, we also combine all the instances compromised by the Mirai botnet while the victim node is running different types of IoT applications. For the *Reboot* and *Idle* classes, we use the same datasets.

In the cross-evaluation test, both the accuracy of the overall classes and of the Mirai class are about 90% as shown in Figure 10. It is a little bit lower than the self-evaluation tests but still durable against the Mirai botnet. Considering that our model could be potentially integrated into a random IoT device, if we train all the power consumption data from many types of IoT devices, our design will still perform well in terms of classification accuracy.

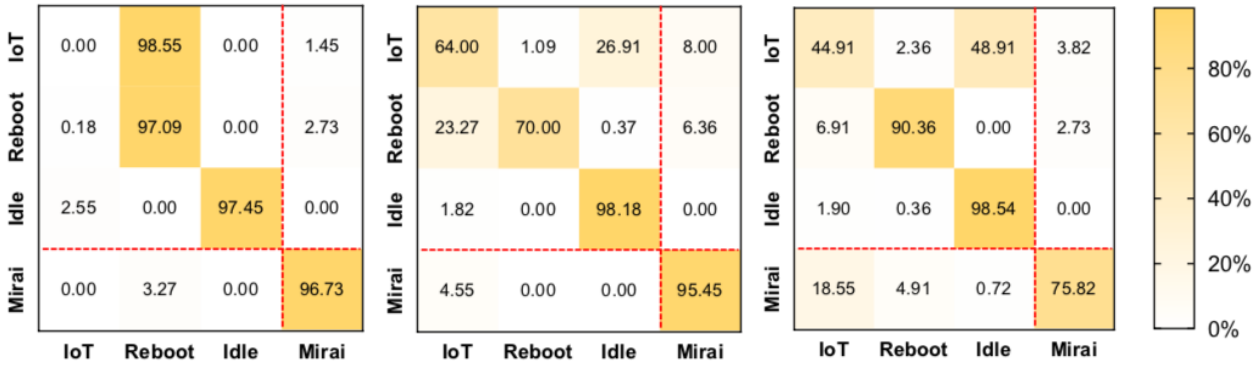
6.1.3. Leave-One-Botnet-Out Test

The basic procedure of a leave-one-out test is that test and training datasets are different. Thus, it demonstrates that our system is independent of various environments since it can tell how well our system detects unknown data. First of all, we conduct leave-one-botnet-out tests. It is important to show the robustness against other new botnets that have different power consumption patterns. We train the datasets from two botnets and test on the other botnet's dataset.

Table 3 shows the two-class (*Botnet* and *Non-Botnet*) classification results. In most cases, the accuracy of the classification is above 90%. This infers that our model is still able to distinguish unknown power patterns well based on the trained datasets. When we leave the Masuta dataset out, the result is worse than the other two tests. This is because the trained datasets from the Mirai and Sora datasets are more similar to each other than the Masuta dataset. Even in this case, the accuracy is around 90% whereas the leave-Mirai-out and leave-Sora-out tests achieve about 94%. Table 3 also exhibits the four-class classification accuracy. Even though results are about 10% worse than two-class classifications, they are still around 80%. This is durable considering the fact that we classify untrained four-class datasets.

Table 3: Classification Accuracy of the Leave-One-Botnet-Out tests

	Leave-Mirai-Out	**Leave-Masuta-Out**	**Leave-Sora-Out**
Two-Class Classification Accuracy	94.3%	89.5%	94.5%
Four-Class Classification Accuracy	84.3%	79.6%	83.6%



(a) Camera (b) Router (c) Voice Assistant

Figure 12: Confusion Matrix of the Leave-One-Device-Out Tests

6.1.4. Leave-One-Device-Out Test

We also run leave-one-device-out tests; we train the datasets from two devices and apply the result to diagnose the dataset of the remaining subject. This scenario happens rarely in real situations. However, if a new type of IoT device or a new service generates untrained power consumption patterns, then we need to validate the performance of our model in these cases. The evaluation results show the robustness of our design; it also shows that our classifier is not overfitted to a specific type of IoT device. Therefore, we verify that our model can still detect untrained malicious botnet behaviors properly.

According to the results of the leave-one-out tests, as shown in Table 4, the two-class classification still performs well, which results are 96%, 88%, and 89% for each test. For example, when we train the datasets from the *Camera* and *Voice Assistant* devices for testing on *Router* dataset, we achieve 87.9% accuracy overall. Although the results are not as good as the self-tests or cross-evaluation, it is justifiable since they train on different instances from the dataset we classify.

Table 4: Classification Accuracy of the Leave-One-Device-Out tests

	Leave-Camera-Out	**Leave-Router-Out**	**Leave-Voice Assistant-Out**
Two-Class Classification Accuracy	96.2%	87.9%	89.1%
Four-Class Classification Accuracy	74.4%	77.3%	82.4%

As shown in Table 4, when we compute the four-class classification accuracy, the results are around 80%, which is not as good as two-class classifications. However, even in these tests, we achieve higher than 95% accuracy on the *Mirai* class for the Camera and Router devices as illustrated in Figures 12a and 12b. In Figure 12c, when we leave out the Voice Assistant instances during the testing procedures, we get about 75% accuracy for the *Mirai* class. We see a misclassification of 19% as the *IoT service*, which is power consumption instances generated by the Camera and Router devices in this case. This is because the Router instances might impact the accuracy since the self-test on the router achieves 90% accuracy on the *Mirai* class as you see in Figure 9b. This is the lowest accuracy among the self-evaluations.

Another thing we observe is that none of the Camera instances are classified as the *IoT service*, which consists of Router and Voice Assistant instances, as shown in Figure 12a. Instead, most of the Camera instances are classified as the *Reboot* class. This is consistent with the fact that 15% of Camera instances are misclassified as the *Reboot* class in the self-test as shown in Figure 9a. We have already discussed in Section 6.1.1 that the traffic from the Camera services is distinct from the power consumption data of the other devices. In Figure 12c, we also see that about half of the Voice Assistant traffic is misclassified as the *Idle* class. This is also consistent with the finding that the power consumption patterns between Voice Operation procedures look like an idle status as described in Figure 3b.

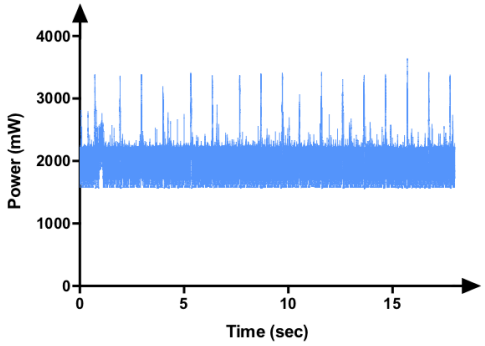


Figure 13: Power Consumption Patterns of Extrusion

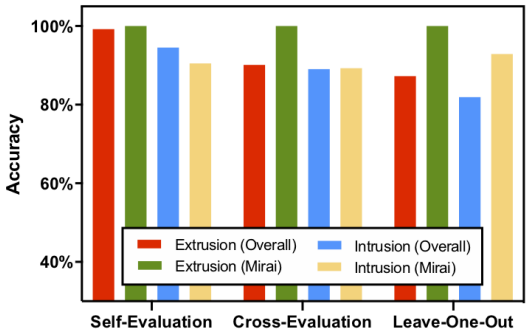


Figure 14: Accuracy of Extrusion Detection

Overall, we still achieve good performance in terms of detecting botnet behaviors. The total accuracy of the leave-one-out tests is reasonably good considering that we test on untrained datasets. Moreover, the leave-one-out tests reveal the high potential of our model to observe the characteristics of IoT applications as well as to detect IoT botnets.

6.1.5. Comparison with State-of-the-Art

There are no previous botnet detection systems to target emerging IoT botnets by measuring power consumption. However, there were some pioneered works that target malicious behaviors by analyzing power consumption. Thus, we choose one of the most influential pioneered work, *MODELZ* [14], and compare it with our system.

Table 5: The results of the two-class classification accuracy compared with *MODELZ*

	Self-Evaluation (Camera)	**Self-Evaluation (Router)**	**Self-Evaluation (Voice Assistant)**	**Cross Evaluation**	**Leave - Camera -Out**	**Leave - Router -Out**	**Leave - Voice Assistant -Out**
MODELZ	74%	63%	65%	61%	59%	61%	51%
Our Model	97%	92%	96%	90%	96%	88%	89%

We implement their method on our datasets to compare with our method. In [14], they first apply a moving average filter to remove high-frequency noises of power consumption data, then calculate the χ^2-distance between the power reading and templates to find the most similar one. Table 5 displays the classification accuracy of our method and *MODELZ*. The results show that our method outperforms their method. Our deep learning-based system brings accuracy improvement up to 30%. This is because our CNN design has much more parameters; thus, it is sophisticated enough to learn power consumption patterns. *MODELZ*, however, performs poorly since it depends on only a few number of predefined power signatures, which are their templates, to characterize power consumption data.

6.2. Extrusion Detection Accuracy

We evaluate the performance of our botnet detection model after an IoT device has already been compromised by botnets, as illustrated in Figure 1c. Even if the device is compromised, our botnet detection design can still prevent IoT devices from rapid propagation. Figure 13 shows how power consumption patterns look when the infected device starts scanning. Specifically, it scans randomly generated IP addresses every second, making the power consumption patterns very spiky.

With the extrusion dataset for the *Mirai* class, we run three other tests —Self-Evaluation on the Router device, Cross-Evaluation, and Leave-Router-Out test —and compare the results with the intrusion cases. As shown in Figure 14, the accuracy for the extrusion cases is always higher than the results in the intrusion cases. Because the extrusion power data is easily distinguishable from the other classes, we achieve 100% accuracy for the *Mirai* class for all the three tests. This observation supports that our model is able to detect IoT botnets even when an IoT device is being compromised. Moreover, it infers that our botnet detection method can prevent IoT devices from rapid propagation at an individual device level.

6.3. Other Performance Metrics

In addition to *Accuracy*, we also provide *Precision*, *Recall*, and *F1-Measure* metrics to validate the performance of our model. *Precision* tells the exactness of the system, where low *Precision* means a large number of False Positives. *Recall* infers the completeness of the classification results, with low *Recall* represents more False Negatives. Lastly, *F1-Measure* considers the balance between *Precision* and *Recall* as described in Eq. (4).

Our self-evaluation and cross-evaluation tests introduce decent results for the above metrics, as illustrated in Figure 15. Overall, most of the metrics are higher than 90% accuracy. For the-leave-one-subject-out tests, they have lower *Precision* results than *Recall*, which infers that our model tends to have more False Positives. In other words, our model classifies IoT botnets conservatively. In terms of *F1-Measure*, most of the tests, including the leave-one-out tests, have higher than 85% *F1-Measure* values. This shows the robustness of our botnet detection model.

$$F1\text{-}Measure = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (4)$$

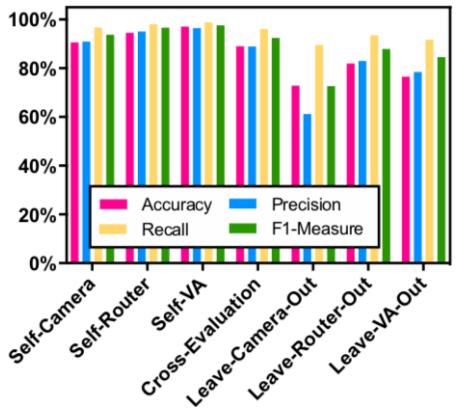


Figure 15: Comparisons of Different Metrics

6.4. Impact of Design Factors

In this section, we discuss some design factors that impact the performance of our model.

6.4.1. Sampling Rate

For our evaluation, we use an off-the-shelf power monitor that offers a high sampling rate. Obviously, this device introduces accurate results, but it is very expensive, making it difficult to be widely deployed. Thus, we evaluate the performance of our model with lower sampling rates on the Camera self-test. If our model does not require high frequency in power monitoring, then we will build hardware with a cheaper sensor in the future.

Specifically, to extract lower sampled data, we take every nth element from given power consumption data. For instance, we take power consumption data at every five elements to get $1kHz$ sampled data. After that, we feed them to our model. By doing so, we provide the evaluation results for four different sampling rates. The results are illustrated in Table 6. As the sampling rate decreases, we get the lower accuracy for both the overall classes and Mirai class. However, even with the $512Hz$ sampling rate, the accuracy is still above 95% for the *Mirai* class whereas the $128Hz$ sampling rate does not introduce good accuracy for all the classes. We also observe that the low sampling rate reduces training and testing time of classification. This infers that our model performs best, achieving both good accuracy and fast training time with the $512Hz$ sampling rate. Therefore, we will take the $512Hz$ sampling rate when we build hardware for future work.

Table 6: Impact of Sampling Rate

Sampling Rate	**5kHz**	**1kHz**	**512Hz**	**128Hz**
Overall Classes Accuracy	90.9%	90.8%	90.4%	81%
Mirai Class Accuracy	99%	98.1%	95.3%	85%

6.4.2. Normalization

Figure 16 shows the impact of normalization as data processing. To test this, we perform cross-evaluation tests as described in Section 6.1.2. As it is shown, the accuracy is below 80% without normalization, whereas our model achieves almost 90% accuracy. We also compute some other common metrics including *Precision*, *Recall* and *F1-Measure*, these metrics are increased by conducting normalization from 2.7% up to 9.4%. Thus, these results justify our explanations in Section 4.2.

6.4.3. *Input Size*

The amount of input is important for CNNs to train datasets. If it is not enough, CNNs cannot fully learn features from the datasets. However, a larger amount of input also makes training time much longer; thus we need to consider both performance and time consumption.

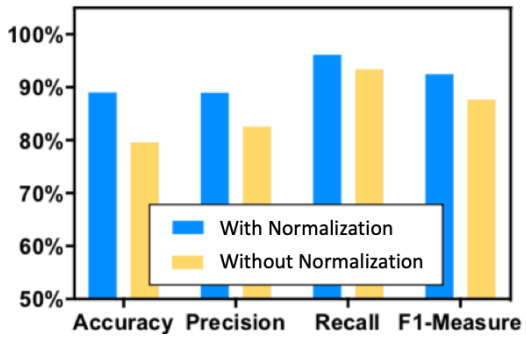


Figure 16: Impact of Normalization

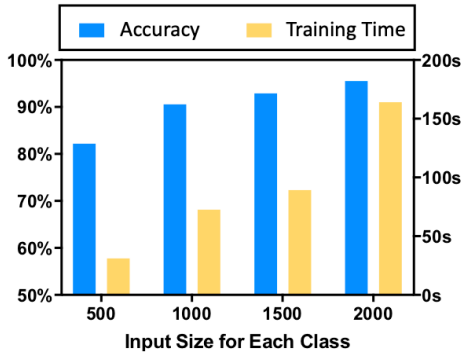
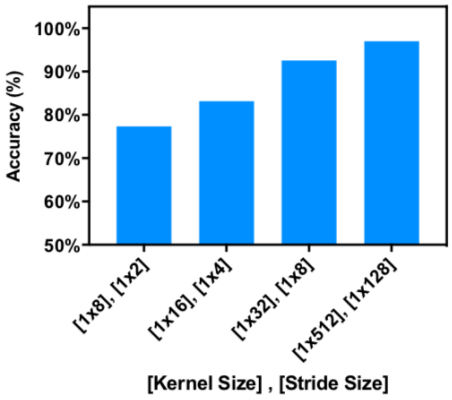


Figure 17: Impact of Input Size

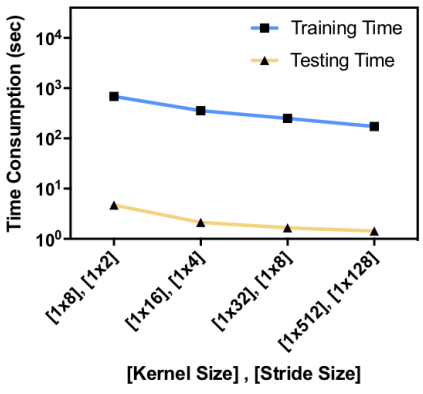
We run self-evaluation tests with the Router datasets. In Figure 17, we observe that our system cannot converge to a global optimal point with 500 instances for each class. However, when we have more than 1,000 instances for each class, the accuracy is higher than 90%, and the accuracy increases as the number of input increases. Considering that the training time is less sensitive because our data is trained off-line, we could have more data until the accuracy converges. In our experiments, we use 2,000 instances for each class because it is the minimum size to get stable and produce more accurate results.

6.4.4. *Kernel Size*

As explained in Section 5.2, the size of the kernel and stride is important to our botnet detection design. To demonstrate this, we perform self-evaluation tests using the Voice Assistant datasets. Figure 18 illustrates the impact of them. As the kernel size increases, we keep the ratio of the stride as one-quarter of the kernel size. It allows us to learn features faster so that our classifier reduces its training time. Since our sampling rate is $5kHz$, we do not need smaller kernel size, which might result in a huge volume of redundant convolution operations.



(a) Accuracy



(b) Time

Figure 18: Impact of Kernel Size

As you observe in Figure 18a, we achieve much higher accuracy as the kernel size goes up. Moreover, we measure the training time and testing time from the self-tests as shown in Figure 18b. Both of them also decrease exponentially

as the kernel size increases. Since both the accuracy and training time begin to converge when the kernel size is larger than 512, we take 512 as our kernel size. Accordingly, the evaluation results validate that the kernel size of our CNN model is appropriate.

7. Discussion

In this section, we address the potential usage of the botnet detection model and discuss our future works.

Our model focuses on learning power consumption patterns to detect malicious behaviors. However, because our model requires measuring power consumption data, it does not apply to be integrated into the existing IoT devices such as a router, camera, or voice assistant. Instead, it is potentially deployed outside a primary IoT device in a secondary device. One possible solution is for our model to be integrated into a smart plug device. Specifically, the smart plug is an IoT device that monitors the power consumption of a connected device; thus, it will be much easier for it to adapt our model. By doing this, we can significantly reduce the introduction cost of our model.

One assumption we have in this paper is that different power consumption due to various WiFi-signal condition is negligible. Although it may influence power consumption, we do not vary our experimental setups but focus on characteristics of the IoT botnets. Moreover, even though most current off-the-shelf Smart Plug devices monitor a single device, there have been some Smart Plug devices that can monitor multiple devices at the same time. We believe that the deep learning classifier can also learn mixed power consumption data monitored by one power monitor, but we leave it for our future work.

We propose a botnet detection method via power consumption modeling. By conducting offline training and testing, we demonstrate the potential of our model. Then, our next step is to make an online classifier. We plan to build our server to learn power consumption data for offline training. For online testing on IoT devices, it does not require much testing time as we observed in the evaluation section. Therefore, we will design a deep learning classifier on resource-constrained IoT devices. Moreover, we also need to design a small form factor sensor module instead of using expensive power monitor hardware. By fully implementing our model and achieving still good performance, we will support that our system is still durable against real-world botnet attacks.

8. Related Work

In this section, we summarize the related work concerning about IoT botnets.

Network Intrusion Detection Systems (IDS) has been scrutinized in the literature for many years [23, 24, 25, 26, 27, 28, 29, 6, 7, 30, 31, 32, 33, 34]. In general, these methods can be classified as either signature-based or anomaly-based systems. The signature-based methods can detect well-known attacks, but those are still vulnerable to unknown patterns. The anomaly-based methods usually rely on ML techniques to detect unusual traffic. Those approaches commonly take advantage of a specific protocol like SMTP or DNS to detect anomaly traffic. Among them, the authors in [6] and [7] had discussed the possibility of huge botnets by IoT devices before Mirai emerged.

After Mirai occurred in 2016, many researchers have either reviewed the IoT botnets thoroughly [35, 36, 37, 5, 38] or proposed a system to defend them [8, 39, 40, 41, 10, 11, 12]. In [8], the authors propose a system to expel the Mirai botnet while its propagation, but they require a user to get involved to do that. Some researchers also conduct a thorough survey on the IoT security concerns including botnet detection [42, 43, 9, 44], which have already covered a huge number of papers. This infers that detecting IoT botnets has been actively investigated. Some of them have applied ML or Neural Networks [10, 11, 12, 40]. However, none of them can detect the early propagation stage of IoT botnets at a device level.

Another type of botnet detection approach is to measure power consumption data [13, 14]. As observed in our evaluation, they also exploit the fact that power signals could change noticeably by malicious behaviors. The authors in [14] pioneered to exploit power consumption data for malware detection, achieving good classification results. However, this work was conducted on old mobile platforms several years ago, and we show the performance comparison in Section 6.1.5. In [13], they review power consumption-based malware detection systems in the mobile environment. Even though power-based methods have also been investigated, those research efforts usually target malware in mobile devices. Moreover, since we design a deep learning-based detection model, our model has more modeling capacity against various botnets. To the best of our knowledge, none of the existing work covers the power

characteristics of the emerging IoT botnets to detect them. Therefore, our deep learning-based model is proposed to diagnose IoT botnets at an IoT device through power consumption modeling.

9. Conclusion

In this paper, we aim to model IoT botnet detection for heterogeneous devices with a deep learning-based method. By exploiting power consumption data, our system achieves high accuracy for botnet detection. Moreover, we detect the IoT botnets even when an IoT device is being compromised so that we slow down the propagation of the malicious botnets earlier. To meet our requirements, we first process power consumption data with segmentation and normalization techniques. Then, we classify whether the device is attacked by the malicious botnets or not using an 8-layer CNN. Our CNN model has a number of parameters to learn power patterns, which makes our model tolerant to various cases. As a result, we achieve up to 98.6% botnet detection accuracy on the self-tests and about 90% on the cross-evaluation test. Even in the most leave-one-out tests, our model detects the botnet with higher than 95%. Moreover, we achieve higher than 90% accuracy during the self-tests and the cross-evaluation test and get about 85% *F1-Measure* during the leave-one-out tests for the overall four classes, — *IoT service, Reboot, Idle*, and *Botnet*. For the two-class classifications, the results of the leave-one-out tests are 96%, 88%, and 89% for the Leave-Camera-Out, Leave-Router-Out, and Leave-Voice-Assistant-Out tests, respectively. These results validate that our model is durable against untrained malicious behaviors as well as observe the characteristics of IoT services.

References

[1] D. Lund, C. MacGillivray, V Turner… - … Data Corporation (IDC) …, 2014, Worldwide and regional internet of things (iot) 2014–2020 forecast: A virtuous circle of proven value and demand, business.att.com.
URL https://www.business.att.com/content/article/IoT-worldwide_regional_2014-2020-forecast.pdf

[2] D. V. Dimitrov, Medical Internet of Things and Big Data in Healthcare, Healthcare Informatics Research 22 (3) (2016) 156.

[3] W. Xue, C. Luo, G. Lan, R. K. Rana, W. Hu, A. Seneviratne, Kryptein - a compressive-sensing-based encryption scheme for the internet of things., IPSN (2017) 169–180.

[4] F. A. Teixeira, G. V. Machado, F. M. Q. Pereira, H. C. Wong, J. M. S. Nogueira, L. B. Oliveira, SIoT - securing the internet of things through distributed system analysis., IPSN (2015) 310–321.

[5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, Y. Zhou, Understanding the Mirai Botnet., USENIX Security Symposium.
URL https://dblp.org/rec/conf/uss/AntonakakisABBB17

[6] S. Asri, B. Pranggono, Impact of Distributed Denial-of-Service Attack on Advanced Metering Infrastructure, Wireless Personal Communications 83 (3) (2015) 2211–2223. doi:10.1007/s11277-015-2510-3.
URL https://link.springer.com/article/10.1007/s11277-015-2510-3

[7] D. Peraković, M. Periša, I. C. o. N. T. i. P. and, 2015, Analysis of the IoT impact on volume of DDoS attacks, academia.edu.
URL http://www.academia.edu/download/41030230/Postel2015-Perakovic_Perisa_Cvitic_eng_02112015.pdf

[8] P. L. N. G. J. L. J. X. Chen Cao, Le Guan, Hey, you, keep away from my device: remotely implanting a virus expeller to defeat Mirai on IoT devices, UbiComp (2017) 1–15.

[9] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, S. C. de Alvarenga, A survey of intrusion detection in Internet of Things., J. Network and Computer Applications 84 (2017) 25–37. doi:10.1016/j.jnca.2017.02.009.
URL http://linkinghub.elsevier.com/retrieve/pii/S1084804517300802

[10] E. Anthi, L. Williams, P. Burnap, Pulse: an adaptive intrusion detection for the internet of things, in: Living in the Internet of Things: Cybersecurity of the IoT - 2018, Institution of Engineering and Technology, 2018, pp. 1–5. doi:10.1049/cp.2018.0035.
URL http://digital-library.theiet.org/content/conferences/10.1049/cp.2018.0035

[11] I. Letteri, M. Del Rosso, P. Caianiello, D. Cassioli, Performance of Botnet Detection by Neural Networks in Software-Defined Networks., ITASEC.
URL https://dblp.org/rec/conf/itasec/LetteriRCC18

[12] M. Lotfollahi, R. S. H. Zade, M. J. Siavoshani, M. Saberian, Deep Packet - A Novel Approach For Encrypted Traffic Classification Using Deep Learning., CoRR cs.LG.
URL http://arxiv.org/abs/1709.02656v3

[13] J. e. Q. I. J. o. C. S. Awareness, 2016, A review of significance of energy-consumption anomaly in malware detection in mobile devices, researchgate.net.

[14] H. Kim, J. Smith, K. G. Shin, Detecting energy-greedy anomalies and mobile malware variants., MobiSys (2008) 239doi:10.1145/1378600.1378627.
URL http://portal.acm.org/citation.cfm?doid=1378600.1378627

[15] R. Floeter, Hostapd, [Online; accessed 2004] (2004).
URL https://man.openbsd.org/hostapd.8
[16] S. Kelley, Dnsmasq, [Online; accessed 2001] (2001).
URL http://www.thekelleys.org.uk/dnsmasq/doc.html
[17] A. Shiravi, H. Shiravi, M. Tavallaee, A. A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection., Computers & Security.
URL https://dblp.org/rec/journals/compsec/ShiraviSTG12
[18] M. S. Inc., Monsoon power monitor, [Online; accessed 2016] (2016).
URL https://www.msoon.com
[19] A. C. Madrigal, 20 questions with google's assistant and apple's siri (2017).
URL https://www.theatlantic.com/technology/archive/2017/05/20-questions-with-googles_assistant-and-apples-siri/527091
[20] Y. B. Ian Goodfellow, A. Courville, Deep learning (2016).
URL http://www.deeplearningbook.org.
[21] Y. Ma, G. Zhou, S. Wang, H. Zhao, W. Jung, SignFi, Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2 (1) (2018) 1–21. doi:10.1145/3191755.
URL http://dl.acm.org/citation.cfm?doid=3200905.3191755
[22] S. Ioffe, C. Szegedy, Batch Normalization - Accelerating Deep Network Training by Reducing Internal Covariate Shift., CoRR.
URL https://dblp.org/rec/journals/corr/IoffeS15
[23] M. H. Bhuyan, D. K. Bhattacharyya, J. K. Kalita, Network Anomaly Detection - Methods, Systems and Tools., IEEE Communications Surveys & Tutorials 16 (1) (2014) 303–336.
URL http://ieeexplore.ieee.org/document/6524462/
[24] C. Catania, C. G. Garino, Automatic network intrusion detection - Current techniques and open issues., Computers & Electrical Engineering 38 (5) (2012) 1062–1072. doi:10.1016/j.compeleceng.2012.05.013.
URL http://linkinghub.elsevier.com/retrieve/pii/S0045790612001073
[25] S. C. Guntuku, P. Narang, C. Hota, Real-time Peer-to-Peer Botnet Detection Framework based on Bayesian Regularized Neural Network, arXiv.orgarXiv:1307.7464v1.
URL http://arxiv.org/abs/1307.7464v1
[26] X. D. Hoang, Q. C. Nguyen, Botnet Detection Based On Machine Learning Techniques Using DNS Query Data., Future Internet 10 (5) (2018) 43. doi:10.3390/fi10050043.
URL http://www.mdpi.com/1999-5903/10/5/43
[27] E. B. B. Samani, H. H. Jazi, N. Stakhanova, A. A. Ghorbani, Towards effective feature selection in machine learning-based botnet detection approaches., CNS (2014) 247–255doi:10.1109/CNS.2014.6997492.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6997492
[28] J. van Roosmalen, H. P. E. Vranken, M. C. J. D. van Eekelen, Applying deep learning on packet flows for botnet detection., SAC (2018) 1629–1636doi:10.1145/3167132.3167306.
URL http://dl.acm.org/citation.cfm?doid=3167132.3167306
[29] K.-C. Lin, S.-Y. Chen, J. C. Hung, Botnet Detection Using Support Vector Machines with Artificial Fish Swarm Algorithm., J. Applied Mathematics 2014 (4) (2014) 1–9. doi:10.1155/2014/986428.
URL http://www.hindawi.com/journals/jam/2014/986428/
[30] J. He, Y. Yang, X. Wang, Z. Tan, Adaptive traffic sampling for P2P botnet detection., Int. Journal of Network Management 27 (5) (2017) e1992. doi:10.1002/nem.1992.
URL http://doi.wiley.com/10.1002/nem.1992
[31] P. Bazydlo, K. Lasota, A. Kozakiewicz, Botnet Fingerprinting - Anomaly Detection in SMTP Conversations., IEEE Security & Privacy 15 (6) (2017) 25–32. doi:10.1109/MSP.2017.4251116.
URL http://ieeexplore.ieee.org/document/8123472/
[32] R. Kozik, M. Choras, Pattern Extraction Algorithm for NetFlow-Based Botnet Activities Detection., Security and Communication Networks 2017 (2017) 1–10. doi:10.1155/2017/6047053.
URL https://www.hindawi.com/journals/scn/2017/6047053/
[33] X. Li, J. Wang, X. Zhang, Botnet Detection Technology Based on DNS., Future Internet 9 (4) (2017) 55. doi:10.3390/fi9040055.
URL http://www.mdpi.com/1999-5903/9/4/55
[34] M. Thangapandiyan, P. M. R. Anand, An efficient botnet detection system for P2P botnet, in: 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET, IEEE, 2016, pp. 1217–1221. doi:10.1109/WiSPNET.2016.7566330.
URL http://ieeexplore.ieee.org/document/7566330/
[35] G. Kambourakis, C. Kolias, A. Stavrou, The Mirai botnet and the IoT Zombie Armies, in: 2017 IEEE Military Communications Conference (MILCOM), IEEE, 2017, pp. 267–272. doi:10.1109/MILCOM.2017.8170867.
URL http://ieeexplore.ieee.org/document/8170867/
[36] M. De Donno, N. Dragoni, A. Giaretta, A. Spognardi, DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation, Security and Communication Networks 2018 (4) (2018) 1–30. doi:10.1155/2018/7178164.
URL https://www.hindawi.com/journals/scn/2018/7178164/
[37] C. Kolias, G. Kambourakis, A. Stavrou, J. M. Voas, DDoS in the IoT - Mirai and Other Botnets., IEEE Computer 50 (7) (2017) 80–84. doi:10.1109/MC.2017.201.
URL http://ieeexplore.ieee.org/document/7971869/
[38] R. Hallman, J. Bryan, G. Palavicini, J. D. o. I. o. Things, 2017, IoDDoS the internet of distributed denial of sevice attacks, researchgate.net.
URL https://www.researchgate.net/profile/Roger_Hallman2/publication/316455478_IoDDoS_-_The_

Internet_of_Distributed_Denial_of_Service_Attacks_A_Case_Study_of_the_Mirai_Malware_and_IoT-Based_Botnets/

[39] M. Lyu, D. Sherratt, A. Sivanathan, H. H. Gharakheili, A. Radford, V. Sivaraman, Quantifying the reflective DDoS attack capability of household IoT devices, ACM, New York, New York, USA, 2017. doi:10.1145/3098243.3098264.
URL http://dl.acm.org/citation.cfm?doid=3098243.3098264

[40] N. F. Rohan Doshi, Noah Apthorpe, Machine Learning DDoS Detection for Consumer Internet of Things Devices (2018) 1–7.

[41] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, Y. Elovici, N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders, arXiv.orgarXiv:1805.03409v1.
URL http://arxiv.org/abs/1805.03409v1

[42] T. Tyagi, Botnet of Things: Menace to Internet of Things, ijfrcsce.org.
URL http://www.ijfrcsce.org/download/conferences/IC3NS_2018/IC3NS_Track/1522230382_28-03-2018.pdf

[43] Y. Yang, L. Wu, G. Yin, L. Li, H. Zhao, A Survey on Security and Privacy Issues in Internet-of-Things, IEEE Internet of Things Journal 4 (5) (2017) 1250–1258. doi:10.1109/JIOT.2017.2694844.
URL http://ieeexplore.ieee.org/document/7902207/

[44] N. Kumar, J. Madhuri, M. ChanneGowda, Review on security and privacy concerns in Internet of Things, in: 2017 International Conference on IoT and Application (ICIOT), IEEE, 2017, pp. 1–5. doi:10.1109/ICIOTA.2017.8073640.
URL http://ieeexplore.ieee.org/document/8073640/