Deep Learning based Malware Detection in IoT

Ben Salem Amina
Chair of Distributed and Multimedia Information Systems
University of Passau
Passau, Germany
bensal02@ads.uni-passau.de

***Abstract*—The Internet of Things (IoT) is a fast-growing technology that is finding its way into almost all aspects of our lives, be it in the public sector such as smart grids and smart cities or more private domains like wearable devices and smart homes. However, the rate at which IoT is evolving comes with huge risks. Given the number of vulnerabilities IoT devices have due to their limited resources and poor design, they have become a favorite target for attackers. In this paper, we propose a deep learning model for malicious traffic detection within IoT environments. We work with the IoT-23 dataset containing network flows from real gadgets and implement a deep neural network which yields an accuracy of 86.26%.**

***Index Terms*—IoT Security, Malware Detection, Deep Learning**

I. Introduction

Adapting security solutions to keep up with the ever-expanding threat of IoT has motivated researchers to investigate novel approaches. Machine Learning (ML) has generally received a lot of attention, however recently, we have seen an emphasis on Deep Learning (DL) techniques in IoT malware detection. Using traditional machine learning algorithms can be inefficient and resource-intensive since they heavily rely on human intervention to craft learning features which also limits their applicability. As for DL models, not only are they capable of learning from raw data, they also scale very well with the sizes of the datasets [1]. This makes them suited for the IoT context where an enormous amount of data is being constantly generated by the devices. Malware Detection techniques can be based on static features which rely on examining the code structure to extract characteristics of malicious files. Another approach, however, would be dynamic analysis, where the malware is executed in a safe environment to study its behavior. Unlike signature-based technique which is specific for each malware variant, the latter is capable of detecting both known and unknown malicious software. This is crucial in IoT for which malware is considered to be one of the top 5 biggest security challenges [2]. Applying the latest advances in security has become a necessity and in this work, we join efforts with the existing studies of the potential application of deep learning in IoT. Specifically, we focus on network analysis of the IoT-23 dataset, and we design and implement a neural network to identify malicious behavior within an IoT environment.

The remainder of the paper is structured as follows: In section II we review related work in the literature and discuss their methods in tackling the issue. In Section III we begin by presenting the dataset after which we outlay and explain the pre-processing steps we have taken. We discuss the proposed model and obtained results in section IV and finally conclude by present our thoughts regarding future work.

II. Related Work

Leveraging Machine Learning (ML) to create powerful models for security applications has been the focus area of many research projects. More specifically, models based on deep learning techniques have been the topic of study for many academics. In this section, we discuss the latest works in this field.

Dutta, V et al. (2020) [3] directed their efforts towards improving the efficiency of Intrusion Detection Systems (IDSs) by combining Deep Ensemble Learning and Stacked Generalization. Their method was based on two algorithms: Deep Neural Network (DNN) and Long Short-Term Memory (LSTM). Results obtained from this ensemble which represents the base model were then passed to a meta-classifier. In the first step, level-0 models decide on the nature of the flows while the level-1 classifier uses Linear Regression to decide on how the best combination of the output. To evaluate their approach, they used three of the latest benchmark datasets. Namely IoT-23, LITNET-2020, and NetML-2020 comprised of network flows. Their findings showed that the proposed technique gave better results compared to the individual algorithms across all datasets. They achieved an accuracy of 99,7% on the IoT-23 dataset and a 100% for the remaining two.

In [4], J. Jeon et al. (2020) proposed a dynamic analysis for IoT malware detection (DAIMD), which utilized Convolution Neural Network (CNN) trained on images of behavior features. After running malicious and benign files in a virtual environment, the authors extracted the signatures of these features, which engulfed memory, network, system calls, process, and virtual file system (VFS). Then, they converted the data into images by transforming the characters into integers mapped to the RGB value range. The resulting images were used to train the CNN ZFNet model, which selects representative features and classifies the files accordingly. The model was trained on a dataset of 840 images and yielded an accuracy of 99.87% in detecting malware. However, it is worth mentioning that this study neglects the malware's ability to change its behavior when it recognizes that it's being executed in a sandbox.

Bendiab et al. [5] (2020) suggested an approach based on the CNN Residual Neural Network (ResNet-50) model to analyze and classify IoT network traffic. They collected pcap files of both benign and malicious flows and converted the obtained data into images using the Binvis tool by comparing bytes with their respective ASCII values. The authors provided a comparison with a ResNet model using 34 layers (Resnet-34) and previous works they conducted using Self-Organizing Incremental Neural Networks (SOINN) as well as Google's MobileNet CNN model [6], [7] (2019). Results showed that the ResNet-50 algorithm achieved the best performance with an accuracy of 94.5%. This approach was not, however, tested in a real environment.

Although CNN and LSTM are very popular among the deep learning approaches for network security applications, their use in real-time context faces some limitations due to the detection delay caused by the flows' pre-processing and features extraction. In [8], Hwang et al. (2019) address this challenge by resorting to a packet-level classification. Their framework consists of three steps: first, data pre-processing, which includes mapping packets to a sentence comprised of words, each representing a field in the header. Second, word embedding was performed to extract relevant features. Finally, the output was passed to the LSTM model for classification. The authors tested their method on a selection of IoT datasets as well as synthetic data that they generated. The model's ability to classify malware with the highest accuracy being 99.98%. This method proved to be more efficient in terms of detection time which is less than 2s. However, given the size of the dataset and the high number of epochs (200) used to achieve these results, the training time was significantly higher than flow-based approaches.

Some other studies focused on creating a lightweight IoT malware detection framework based on CNN [9], [10]. In [9] (2019), the model was tested on the IoTPOT dataset of Mirai and Gafgyt attacks. Malware images were constructed and then passed to a shallow network of two layers which detected malicious activity with an accuracy of 94% and could correctly identify the attack with an accuracy of 93.33%. This work, however, did not include obfuscation techniques. Researchers in [11] (2020) used the same dataset with a different approach which consisted of installing a monitoring node to detect the misbehavior of IoT devices within the network. In addition to representing the data as grayscale images, they also constructed images based on the Hilbert Curve to reap the benefits of its clustering properties. According to the authors, using this data representation with CNN provides better results. Their model adds a third layer to the previous model in [9] which gave a higher accuracy (95.6%). In addition, this paper tested the model's performance under code obfuscation and achieved an accuracy of 92%.

Meidan et al. (2018) [12] proposed a network-based approach using deep auto-encoders focusing on IoT Botnets. They built a network of nine commercial IoT gadgets from which they collected benign traffic and used it to train the model. Their theory was that the auto-encoder would be able to reconstruct normal flows and fail to do so with unknown observations which consist of the malicious traffic. Although the created model was 100% successful in detecting the attacks, mention must be made that the scalability of this approach requires further testing given that the authors used a small number of devices and maintained a model for each one. Additionally, only two botnet families were used in this paper, namely Mirai and BASHLITE from the N-BaIoT dataset.

The same dataset was employed by Kim, J. et al. (2020) in [13] in a more comprehensive study where they tested five ML algorithms and three DL models, CNN, Recurrent Neural Network (RNN), and LSTM. Unlike the previous work, which classified the attack types, this paper also focused on identifying the nature of traffic, turning the problem into a binary classification task. In both the multi-class and binary classification, the CNN model outperformed RNN and LSTM.

In [14] Brun, O. et al. (2018) focused their efforts on the most common network attacks against IoT gateways which are Denial of Service (DoS) and Denial of Sleep attacks. After analyzing how the attacks work, the researchers defined a set of metrics on which they based the detection. They used the number of outgoing ICMP packets with "destination unreachable" for UDP flood attacks, the number of half-opened TCP connections for TCP SYN, the number of packets over a long and short time for Sleep Deprivation attack and Barrage attack, and the number of broadcast message for the Broadcast Attack. Dense Random Neural Network (RNN) was trained on the pcap files however, using a simple threshold detector gave similar results.

Reddy et al. (2020) [15] presented an anomaly detection model in the context of smart cities based on DNN, an Artificial Neural Network (ANN) with more than a hidden layer which makes them capable of tackling complex non-linear problems. They tested a model with a different number of neurons for each hidden layer and used Relu and softmax activation functions along with the Adam optimizer. This classifier was trained on the DS2OS dataset and achieved above 98% accuracy surpassing traditional ML algorithms. However, the number of addressed attacks was limited to seven.

Having evaluated related work, the next part is dedicate to explaining our approach.

III. Methodology

In this section, we present the dataset and detail our pre-processing and feature engineering steps.

A. Dataset

The dataset we are basing this work on is the IoT-23 dataset [16] of real and labeled IoT traffic. It was collected for a year in the Stratosphere Laboratory at the Czech Technical University (CTU) as part of a project funded by Avast Software. To the best of our knowledge, this is the latest and most comprehensive dataset containing over 20 different malicious scenarios from various malware families in addition to benign traffic from three devices: *(i)* a Philips HUE smart

LED lamp, *(ii)* an Amazon Echo home intelligent personal assistant, and *(iii)* a Somfy smart door-lock. The dataset is divided into 23 folders, each containing a labeled file for every studied scenario. These files were obtained by running the Zeek network analysis tool on the pcap captures. Tables I and II summarize the information about the different files in the dataset and the features respectively.

TABLE I
DATASET DESCRIPTION

Dataset Name	Zeek Flows	Scenario
CTU-Honeypot-Capture-7-1	139	Somfy Door Lock
CTU-Honeypot-Capture-4-1	461	Philips HUE
CTU-Honeypot-Capture-5-1	1,383	Amazon Echo
CTU-IoT-Malware-Capture-34-1	23,146	Mirai
CTU-IoT-Malware-Capture-43-1	67,321,810	Mirai
CTU-IoT-Malware-Capture-44-1	238	Mirai
CTU-IoT-Malware-Capture-49-1	5,410,562	Mirai
CTU-IoT-Malware-Capture-52-1	19,781,379	Mirai
CTU-IoT-Malware-Capture-20-1	3,210	Torii
CTU-IoT-Malware-Capture-21-1	3,287	Torii
CTU-IoT-Malware-Capture-42-1	4,427	Trojan
CTU-IoT-Malware-Capture-60-1	3,581,029	Gagfyt
CTU-IoT-Malware-Capture-17-1	54,659,864	Kenjiro
CTU-IoT-Malware-Capture-36-1	13,645,107	Okiru
CTU-IoT-Malware-Capture-33-1	54,454,592	Kenjiro
CTU-IoT-Malware-Capture-8-1	10,404	Hakai
CTU-IoT-Malware-Capture-35-1	10,447,796	Mirai
CTU-IoT-Malware-Capture-48-1	3,394,347	Mirai
CTU-IoT-Malware-Capture-39-1	73,568,982	IRCBot
CTU-IoT-Malware-Capture-7-1	11,454,723	Linux,Mirai
CTU-IoT-Malware-Capture-9-1	6,378,294	Linux,Hajime
CTU-IoT-Malware-Capture-3-1	156,104	Muhstik
CTU-IoT-Malware-Capture-1-1	1,008,749	Hide and Seek

Developing detection models based on this dataset can be done using the **Zeek log** files containing the described flows in I, or by using the **pcap** files instead. We decided to exclude the latter and our choice can be simply justified by the time and technical constraints the first option imposes. Working with the pcap files would imply redoing the work done by the analysts of this research team, a task that is not only time-consuming beyond the allocated period for our project but also unnecessary. Our same approach is also embraced by the creators of the dataset themselves who offer a download option containing only the files produced by Zeek.

B. Pre-processing

Data pre-processing is a necessary phase that allows us to transform the data into an understandable format that can be exploited by our model. As mentioned in paragraph III-A, the data in our case was spread across 23 files. We used the Pandas library to facilitate the sanitization process of deleting irrelevant metadata and comments. We then concatenate all frames to obtain a single data frame on which we conduct the remaining operations. The first step we took was to adapt our target label values to our classification problem. The label in IoT-23 was a composition of multiple columns containing information about the nature of the network flow as well as the type of the attack, hence we decided to maintain the column

TABLE II
DATASET FEATURE OVERVIEW

#	***Name***	***Description***
1	ts	Flow start time
2	uid	Unique connection ID
3	id.orig-h	Source IP address
4	d.orig-p	Source port
5	id.resp-h	Destination IP address
6	id.resp-p	Destination port
7	proto	Connection transport layer protocol
8	service	An identification of an application protocol being sent over the connection
9	duration	How long the connection lasted in seconds
10	orig-bytes	Number of bytes in the originator's payload
11	resp-bytes	Number of bytes in the responder's payload
12	conn-state	Connection state
13	local-orig	T if connection originated locally else F
14	local-resp	T if connection responded to locally else F
15	missed-bytes	Missing bytes during connection
16	history	Connection history
17	orig-ip-bytes	Flow of source bytes
18	orig-pkts	Number of originator's packets
19	resp-pkts	Number of destination's packets
20	resp-ip-bytes	Flow of destination bytes
21	tunnel_parents label detailed label	Traffic tunnel, attack label, and attack type

holding the data relevant to our detection problem, which is a label describing whether a flow is malicious or benign. We then directed our focus towards the remaining attributes and examined the data for missing values. We chose to eliminate columns having over 90% of unset values as they provide no meaningful information and would negatively impact the performance of our model. These columns were: *"id.orig-h"*, *"id.resp-h"*, *"local-orig"* and *"local-resp"*. Additionally, we eliminated the columns *"ts"* and *"uid"* as their content was irrelevant to our problem.

To deal with the missing values in the remaining columns and reduce the sparseness in our data, we resorted to imputation techniques (replacing missing values). The concerned numeric columns were *"duration"*, *"orig_bytes"*, and *"resp_bytes"*. When we examined these columns individually, assigning them the proper values seemed ambiguous since there was no straightforward advance towards predicting the duration nor the number of exchanged bytes of a given connection. However, when looking at them together, and analyzing the pattern of empty fields, we found that the values were not missing at random.
We noticed that in almost all cases, the three fields would be empty at the same time. In addition, this event was mainly associated with a value of *"conn_state"* equal to *"S0"*. In fact, S0 represents attempted connection without a reply. This means we could give the field the value *zero* to represent the absence of connection establishment and data exchange.

In the case of categorical columns, the only columns having incomplete data were *"service"* and *"proto"*. In our context, the missing values within these columns were actually meaningful since they represent undetected services and protocols.

The absence of the value itself could be an indicator of abnormality and contribute to our behaviour profiling. For this reason, we opted for substituting the *"nan"* values by an *"unknown"* label.
We also noticed inconsistency in the target value labeling. We found that there were two different labels "benign" and *"Benign"* both designating normal traffic, which we merged into a single value to maintain our binary classification.

Since our model requires all values to be numerical, we needed to ensure that we properly handled and encoded categorical data. usually, the most common approaches to achieve this are: *(i)* Integer Encoding, *(ii)* One-Hot Encoding, and *(iii)* Learned Embedding.

We discarded the option of Learned Embedding, which consists of training a model to learn a distributed representation of the categories since it would add more complexity to our approach. Given the high number of features present in our dataset, we also avoided using One-Hot Encoding, as it would lead to an even higher cardinality. Instead, we opted for Integer Encoding, specifically Label Encoding, which maps the different categories into unique numbers. This method ensures that no ranking or order is established between the different labels which might affect our model's performance.

The final step we took in this phase was to normalize our data to decrease the learning time of our algorithm. We achieve this by rescaling the original values to fall into the range of 0 and 1.

IV. Results and Discussion

A. Proposed Model

In this paper, we propose a deep neural network anomaly detector capable of classifying network flows and detecting malicious traffic within IoT environments. During the design phase, we conducted a number of trials with different architectures that led us to make the following choices:

- **Activation function:** the choice is made separately for both the hidden and output layers. While the activation function in deep layers determines the model's ability to learn from our training data, the choice for the output layer controls the type of predictions made.
  - Hidden layers: we observed the performance with different activation functions and saw that the worst output was associated with the the use of the classical Sigmoid function. Also called logistic function, it is identical to that utilized in Logistic Regression algorithm and is defined as follows:

$$S(x) = \frac{1}{1 + e^{-x}}$$

    where *x* is the input and *e* represents Euler's constant. The main issue with this function is the vanishing gradient that makes it unsuitable for a deep network with many layers. The solution was to use the Rectified Linear Unit (ReLU) activation, a linear function that returns the output if it's positive and zero otherwise, and which overcomes this problem.
  - Output layer: depending on the type of predicted variable, meaning the problem we are solving. It can be categorical (a classification problem) or numerical (a regression problem). In the previous version of this work, we chose the *Softmax* which gave us the *accuracy* and *loss* values presented in tables V and VII. However, when preserving the same activation function for the hidden layers and changing that of the output layer to a *Sigmoid* activation we not only managed to enhance the accuracy of our model, but also reduce the training time cost as seen in tables VI and VIII.
- **Loss function**: or error function, is necessary for the model to estimate its current state during each iteration and readjust the weights to minimize the loss during the following step. We opt for *Binary Cross-Entropy Loss* which is the default function for binary classification with a target set $\{0,1\}$ the most suitable for our use case.
- **Optimizer:** in this case *Adam* optimizer has been proven to be the best option since it is optimized for large data and parameters but also for its computational efficiency, and little memory requirements.
- **Number of layers:** our model consists of a six-layer network where we gradually decrease the number of node for each hidden layer till we have a single node for the output layer whose activation indicates the nature of the predicted traffic. Nevertheless, we also considered the time as amongst our criteria. Our goal was to optimize the time cost while still making sure our model could learn the complexity of our problem. We decided to adjust the chosen model by decreasing the number of layers and that of the nodes to test the effect on accuracy, to minimize the training time by reducing the number parameters while maintaining a roughly similar performance. Taking into consideration this trade-off we managed to obtain close results with a three-layer architecture:
  - The input layer takes in 14 variables representing the number of features in our training set.
  - The first layer has 1200 nodes with ReLU activation.
  - The second layer uses ReLU activation with 800 nodes.
  - Finally, the output layer uses the Softmax activation function and has a single node.

  We summarize both approaches in Tables III and IV.

TABLE III
INITIAL SIX-LAYER MODEL SUMMARY

Layer (type)	*Output Shape*	*Parameters Total*
dense (Dense)	(None, 2000)	30000
dense_1 (Dense)	(None, 1500)	3001500
dense_2 (Dense)	(None, 800)	1200800
dense_3 (Dense)	(None, 400)	320400
dense_4 (Dense)	(None, 150)	60150
dense_5 (Dense)	(None, 1)	151

TABLE IV
THREE-LAYER MODEL SUMMARY

Layer (type)	*Output Shape*	*Parameters Total*
dense_1 (Dense)	(None, 1000)	15000
dense_2 (Dense)	(None, 1000)	1001000
dense_3 (Dense)	(None, 1)	1001

The total number of parameters for the first setup was 4,613,001 whereas it was only 1,017,001 for the second network. All the parameters were trainable in both cases.

B. Results

Before training our model, we had to define a number of hyperparameters:

- **Batch:** defines the number of rows from the dataset the model includes in the prediction process before it updates its parameters.
- **Epoch:** defines the number of iterations over the training set.

We chose to work with 10 epochs and a batch size of 300. These values were conserved for both architectures.

To evaluate our model, we rely on accuracy which is defined as the number of correctly predicted labels, meaning the number of identified malicious flows and we also compare the values for the loss function. In our previous version of this paper, our deep model used a Softmax activation in the output layer and achieved an accuracy of 86.26% for the six-layer architecture and 86.18% for the three-layer approach, while no change was observed for the validation accuracy. However, it is important to note the difference in time taken by each network as the duration was significantly reduced to more than a third of the initial value.

When changing the activation function of the output layer, we noticed a significant improvement in accuracy. The six-layer network achieved 95.38% and had a slightly better time cost. The three-layer model had the best accuracy overall where it reached 95.43% during the training phase and 95.56% for the validation set. Although its time cost was naturally much better than the deeper six-layer model, the previous architecture using *Softmax* gave a marginally better time count.

Nevertheless, we deem this small difference to be acceptable given the scale of enhanced performance. These improvements extend to the loss values where the three-layer model had the smallest loss of 0.0803 compared to all other approaches. Detailed results are given in Tables V-VIII.

TABLE V
RESULT COMPARISON USING SOFTMAX ACTIVATION FUNCTION

Model	*Training Acc*	*Validation Acc*	*Time Count*
Six-layer Model	0.8622	0.8623	1552.99s
Three-layer Model	0.8618	0.8623	415.75s

Examining the accuracy of both the training and validation sets, we observe that they are very close in values which indicates that we probably do not have an overfitting problem.

TABLE VI
RESULT COMPARISON USING SIGMOID ACTIVATION FUNCTION

Model	*Training Acc*	*Validation Acc*	*Time Count*
Six-layer Model	0.9538	0.9533	1550.91s
Three-layer Model	0.9543	0.9556	428.48s

TABLE VII
LOSS FUNCTION COMPARISON USING THE SOFTMAX ACTIVATION

Model	*Training Loss*	*Validation Loss*
Six-layer Model	0.0844	0.0835
Three-layer Model	0.0841	0.0835

TABLE VIII
LOSS FUNCTION COMPARISON USING THE SIGMOID ACTIVATION

Model	*Training Loss*	*Validation Loss*
Six-layer Model	0.0858	0.0827
Three-layer Model	0.0830	0.0803

This can also be confirmed using the loss function as a reference where we found the validation loss value to be smaller that that of the test data.

V. CONCLUSION AND FUTURE WORK

In summary, this paper discussed the different deep learning approaches utilized in detecting malicious traffic within IoT environments. We used a recent real IoT dataset of network flows and built our own deep learning algorithm based on fully connected layers. Our model achieved a reasonable result despite the data limitation and the the time constraints we imposed. We also went as far as remodelling the architecture to reduce its training time then optimizing to obtain higher performance that reached 95.56% in terms of accuracy. The overall performance was very promising and we believe there can still be room for further adjustments to the model as well as the dataset in terms of size to help in enhancing the classification ability. Additionally, the big number of malicious flows compared to benign ones should be taken into consideration. Finally, given the favorable results, we think it would be interesting to create a benchmark of various ML algorithms including DL on the IoT-23 dataset not only for binary but also multi-class classification.

REFERENCES

[1] Jason Brownlee. What is Deep Learning? . In https://machinelearningmastery.com/what-is-deep-learning, 2019.

[2] N. Limbachiya, "Top 10 IoT applications in 2020", DZone, August 18, 2020. [Online Document], Available: DZone Online https://dzone.com/articles/top-5-iot-security-challenges-to-expect-in-2020 [Accessed Dec. 16, 2020]. Onlinehttps://dzone.com/articles/top-5-iot-security-challenges-to-expect-in-2020 [Accessed Dec. 16, 2020].

[3] Dutta, V.; Choraś, M.; Pawlicki, M.; Kozik, R. A Deep Learning Ensemble for Network Anomaly and Cyber-Attack Detection. Sensors 2020, 20, 4583. https://doi.org/10.3390/s20164583

[4] J. Jeon, J. H. Park and Y. Jeong, "Dynamic Analysis for IoT Malware Detection With Convolution Neural Network Model," in IEEE Access, vol. 8, pp. 96899-96911, 2020, doi: 10.1109/ACCESS.2020.2995887.

[5] Bendiab, Gueltoum & Shiaeles, Stavros & Alruban, Abdulrahman & Kolokotronis, Nicholas. (2020). IoT Malware Network Traffic Classification using Visual Representation and Deep Learning.

[6] Baptista, Irina & Shiaeles, Stavros & Kolokotronis, Nicholas. (2019). A Novel Malware Detection System Based on Machine Learning and Binary Visualization. 1-6. 10.1109/ICCW.2019.8757060.

[7] Shire, Robert & Shiaeles, Stavros & Bendiab, Gueltoum & Ghita, B.V. & Kolokotronis, Nicholas. (2019). Malware Squid: A Novel IoT Malware Traffic Analysis Framework Using Convolutional Neural Network and Binary Visualisation. 10.1007/978-3-030-30859-9_6.

[8] Hwang, Ren-Hung & Peng, Min-Chun & Nguyen, Van-Linh & Chang, Yu-Lun. (2019). An LSTM-Based Deep Learning Approach for Classifying Malicious Traffic at the Packet Level. Applied Sciences. 9. 3414. 10.3390/app9163414.

[9] P D, Sai Manoj & Sayadi, Hossein & Mohammadi Makrani, Hosein & Nowzari, Cameron & Rafatirad, Setareh & Homayoun, Houman. (2019). Lightweight Node-level Malware Detection and Network-level Malware Confinement in IoT Networks. 776-781. 10.23919/DATE.2019.8715057.

[10] Su, Jiawei & Vargas, Danilo & Prasad, Sanjiva & Daniele, Sgandurra & Feng, Yaokai & Sakurai, Kouichi. (2018). Lightweight Classification of IoT Malware Based on Image Recognition. 664-669. 10.1109/COMPSAC.2018.10315.

[11] Zaza, Ahmad & Kharroub, Suleiman & Guizani, Mohsen & Abualsaud, Khalid. (2020). Lightweight IoT Malware Detection Solution Using CNN Classification.

[12] Meidan, Yair & Bohadana, Michael & Mathov, Yael & Mirsky, Yisroel & Shabtai, Asaf & Breitenbacher, Dominik & Elovici, Yuval. (2018). N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders. IEEE Pervasive Computing. 17. 12-22. 10.1109/MPRV.2018.03367731.

[13] Kim, Jiyeon & Shin, Yulim. (2020). Intelligent Detection of IoT Botnets Using Machine Learning and Deep Learning. Applied Sciences. 10.3390/app10197009.

[14] Brun, O. & Yin, Yonghua & Gelenbe, Erol. (2018). Deep Learning with Dense Random Neural Network for Detecting Attacks against IoT-connected Home Environments. Procedia Computer Science. 134. 458-463. 10.1016/j.procs.2018.07.183.

[15] Reddy, DKK, Behera, HS, Nayak, J, Vijayakumar, P, Naik, B, Singh, PK. Deep neural network based anomaly detection in Internet of Things network traffic tracking for the applications of future smart cities. Trans Emerging Tel Tech. 2020;e4121. https://doi.org/10.1002/ett.4121

[16] Parmisano Agustin, Garcia Sebastian, and Erquiaga. Maria Jose. A labeled dataset with malicious and benign iot network traffic. In https://www.stratosphereips.org/datasets-iot23. Stratosphere Laboratory, 2020