Detection of IoT Botnet Attacks

Chinmayi C. R.
Dept. of Information Technology
NITK Surathkal
Mangalore, India
chinmayicr27@gmail.com

Priyanka B. G.
Dept. of Information Technology
NITK Surathkal
Mangalore, India
priyahem7@gmail.com

K. Keerthana
Dept. of Information Technology
NITK Surathkal
Mangalore, India
keerthanakanapuram@gmail.com

***Abstract*—As the adoption of Internet-of-Things devices grows, so does their involvement in botnet attacks, necessitating the development of new methods for detecting IoT botnet attacks. The paper proposes a network-based anomaly detection method for the IoT called N-BaIoT. The dataset consists of snapshots of the behaviour of the network. For anamoly detection, deep autoencoders are used to detect anomalous network traffic from compromised IoT devices. The autoencoders have more than one hidden layer. To evaluate the method, the paper has taken dataset of nine infected commercial IoT devices with two widely known IoT-based botnets, Mirai and BASHLITE. The results prove an excellent accuracy of the model. An anomaly detection and attack classification pipeline has been implemented. LIME has been used to explain the features that contribute to the prediction.**

***Index Terms*—botnet, IoT, autoencoders, LIME**

I. Introduction

A ‘bot’ is a software computer program which enables the operator to remotely access and control the infected system wherever it is installed. A network that is compromised with the attack by such bots is called a botnet. It is essential to detect such bots in the network to ensure safety of a system. IoT devices are being easily hacked nowadays. Hence, there is an increase in the Botnet attacks on IoT devices. There is a need to differentiate between hour and millisecond long IoT based attacks. For detecting attacks launched from IoT bots we propose a network-based approach, N-BaIoT for the IoT that uses deep learning techniques to perform anomaly detection. Specifically, we extract statistical features that capture behavioral snapshots of benign IoT traffic, and train a deep autoencoder (one for every device) to find out the IoT device’s normal behaviors. The auto encoders attempt to compress snapshots. When an auto encoder fails to reconstruct a snapshot, it is a strong indication that the observed behavior is anomalous (the IoT device has been compromised and is exhibiting an unknown behavior). An advantage of using deep autoencoders is their ability to find out complex patterns—for example, of varied device functionalities. This leads to an anomaly detector with hardly any false alarms. An attack classification is done with the help of deep neural networks.

II. Literature Survey

Previous IoT-related botnet detection studies focused mainly on the early steps of propagation and communication with the C&C server [1] [3]. Botnet detection approaches are either host-based [3] or network-based [1] [2]. Since not all IoT manufacturers can be counted on to mount designated host-based anomaly detectors on their devices, we believe host-based techniques are less practical. A hierarchical taxonomy of network-based botnet detection approaches, not limited to the IoT domain, was proposed by Sebastián García, Alejandro Zunino, and Marcelo Campo [4]. One of the detection sources they surveyed was honeypots, which have commonly been used for collecting, understanding, characterizing, and tracking botnets14 but are not necessarily useful for detecting compromised endpoints or the attacks emanating from them. Additionally, honeypots usually necessitate a significant investment in the acquisition or emulation of real computers, data inspection, signature extraction, and mutation maintenance. Kalyan Veeramachaneni and his colleagues [5] extended autoencoders for outlier detection but they had a limitation. For their approach, security analysts were also expected to actively mark data for subsequent supervised learning. Closer to our approach, Aaron Tuor and his co authors [6] apply deep learning to system logs for detecting insider threats. They use a combination of DNNs and recurrent neural networks (RNNs), and show dependency on further manual inspection.

III. Methodology

A. Feature Extraction

The data [12] is obtained by extracting a total of 115 traffic statistics. Data set is split into train and test subsets as 80:20. From the data, features are taken over 5 temporal windows: the most recent ones being 100 ms, 500 ms, 1.5 sec, 10 sec, and 1min.

B. Training an Anomaly Detector

We use two separate data sets for training and optimization that only contain benign data.
The first data set is the training set (DS_{trn}):

- Used for training the autoencoder
- Given the parameters for input such as the learning rate (η , the gradient size descent step) and the total number of epochs (complete passes through the entire DS_{trn}).

The second dataset is the optimization set (DS_{opt}):

- Used to optimize these two hyper parameters (η and epochs) iteratively until the mean square error (MSE) between a model’s input (original feature vector) and output (reconstructed feature vector) stops decreasing.

- Terminating at this point prevents overfitting DS_{trn}, thus promoting better detection results with future data.

DS_{opt} is used later to optimize a threshold (tr) that discriminates between the benign and the malicious observations and, finally, the window size (ws), by which the FPR is minimized.

$$tr* = \overline{MSE}_{DS_{opt}} + s(MSE_{DS_{opt}})$$

Fig. 1. Threshold value

The decision of the abnormality is predicted based on a sequence of instances by implementing a majority vote on a moving window. We determine the minimal window size ws* to represent the shortest sequence of instances, a majority vote that produces 0 percent FPR on DS_{opt}:

$$ws^* = \arg\min_{|ws|}(|\{packet \in ws \mid MSE(packet) > tr^*\}| > \frac{ws}{2})$$

Fig. 2. Minimal Time Window size

Autoencoder [9] uses 5 hidden layers of sizes 0.75, 0.5, 0.25, 0.5, 0.75 of the input feature vector size.
Hyperbolic tangent is used as an activation function [9] for our hidden unit neuron. A nonlinear function is used to retain the power of nonlinear models.

C. Attack Classification

The problem of classification is solved with a deep neural network with two hidden layers each with 8 neurons. The proposed neural network consists of:

- Hyperbolic tangent activation function for hidden neurons

$$g(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Fig. 3. Hyperbolic Tangent Activation Function

- Softmax function applied to the last layer

$$softmax(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{N} e^{x_i}}$$

Fig. 4. Softmax Function

- Categorical cross-entropy as a loss function

$$P(y = 1 \mid \boldsymbol{x}) = \max\left\{0, \min\left\{1, \boldsymbol{w}^\top \boldsymbol{h} + b\right\}\right\}$$

Fig. 5. Cross Entropy as Loss Function

	Predicted normal	**Predicted attack**
Actual normal	True Negative (TN)	False Positive (FP)
Actual attack	False Negative (FN)	True Positive (TP)

TABLE I
CONFUSION MATRIX

D. Evaluation metrics

A two-class confusion matrix is used to evaluate the results as in Table 1.

E. Local Interpretable Model-Agnostic Explanations

LIME is used to explain the predictions of complex model in a human interpretable form. This technique works by perturbing values of a particular instance that we want to explain and learning a local linear approximation for classification.This method provides features that resulted in the prediction by the model. Feature weights from a simple model make explanations for the complex model's local behaviour.

IV. TOOLS AND TECHNOLOGIES

Python2 3.6 is the choice made for programming language. Python has recently gained a lot of traction in the machine learning world due to a variety of reasons. The abundance of data processing and data analysis libraries and systems. The majority of the experiments for this thesis were conducted in a Jupyter3 notebook, which allows for better visualization and rerunning of various parts of an experiment. For this project, the Pandas4 library is used to handle the csv info. Pandas data frames with Numpy5 support The scaling and evaluation metrics functions in this work are also based on the scikit-learn6 library. The Keras7 2.2 library is used to create neural network models, which in turn relies on the Tensorflow8 system for all computations.

V. RESULTS

A. Feature Scoring

Fisher's score was calculated on training sets for each data subset used respectively for attack detection, botnet type classification and Mirai attack type classification.

B. Attack detection

The auto encoder was created and trained on all 115 features. Default Keras optimization hyper parameters were used. Using Keras functionality, training was restricted to 100 epochs with the additional condition of early stopping (Figure 6). The model was trained for 23 epochs in total, each taking about 25 seconds, for a total of 584 seconds of training time. Loss is defined as MSE between original and predicted values. There were 185310 samples in the training set and 185311 samples in the optimization (validation) set.

```
es = EarlyStopping(monitor='val_loss', patience=5)
```

Fig. 6. Keras early stopping

2 class (normal, attack)		**3 class (2 botnets + 1 normal)**		**5 classes of Mirai attacks**	
Feature	**F Sc**	**Feature**	**F Sc**	**Feature**	**F Sc**
MI_dir_L0.1_weight	3.34	MI_dir_L3_weight	1.96	MI_dir_L0.01_var	43.75
H_L0.1_weight	3.34	H_L3_weight	1.96	H_L0.01_var	43.75
MI_dir_L1_weight	3.18	MI_dir_L5_weight	1.93	MI_dir_L0.1_var	41.43
H_L1_weight	3.18	H_L5_weight	1.93	H_L0.1_var	41.43
MI_dir_L3_weight	3.01	MI_dir_L1_weight	1.87	MI_dir_L0.01_mean	30.05
H_L3_weight	3.01	H_L1_weight	1.87	H_L0.01_mean	30.05
MI_dir_L5_weight	2.86	MI_dir_L0.1_weight	1.70	MI_dir_L0.1_mean	27.03
H_L5_weight	2.86	H_L0.1_weight	1.70	H_L0.1_mean	27.03
MI_dir_L0.01_weight	1.65	MI_dir_L0.01_weight	1.43	MI_dir_L1_var	19.62
H_L0.01_weight	1.65	H_L0.01_weight	1.43	H_L1_variance	19.62

TABLE II
FISHER'S SCORES

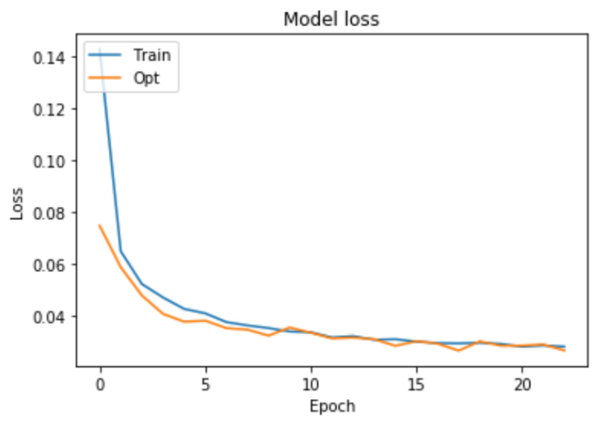


Fig. 7. Autoencoder Training Curve

Values of integers from 1 to 10 were tried for the threshold equation (Figure 1) parameter N on the optimization dataset. It is now possible to evaluate the test set after the model

N	**Accuracy**	**False positives**	**False negatives**
1	0.9894	3911	23
2	0.9934	2203	25
3	0.9961	1396	28
4	0.9973	978	35
5	0.9979	744	38
6	0.9983	587	41
7	0.9986	467	43
8	0.9988	387	45
9	0.9990	292	51
10	0.9992	245	52

TABLE III
RESULTS WITH DIFFERENT N VALUES FOR THRESHOLD

has been trained and the threshold has been set. The precision and accuracy achieved were 0.9991 and 0.9985, respectively. A more detailed breakdown is given by a confusion matrix (Figure 8). False positive rate is thus 0.0015.

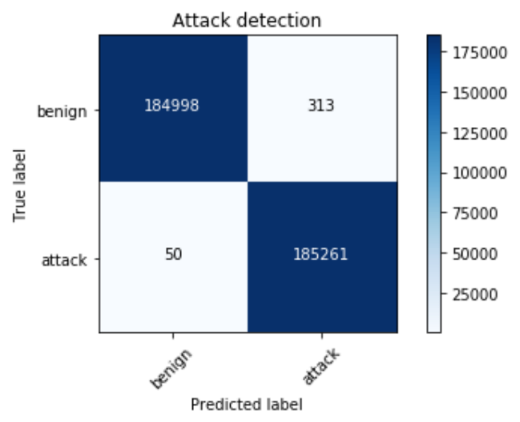


Fig. 8. Attack detection on test data set

C. Attack Classification

Two types of deep learning classifier models were used to distinguish between different botnet types (Mirai, BASHLITE and also non-botnet) and between different Mirai botnet attack types (SCAN, ACK, SYN, UDP, UDPPlain).

1) Botnet type classification: -

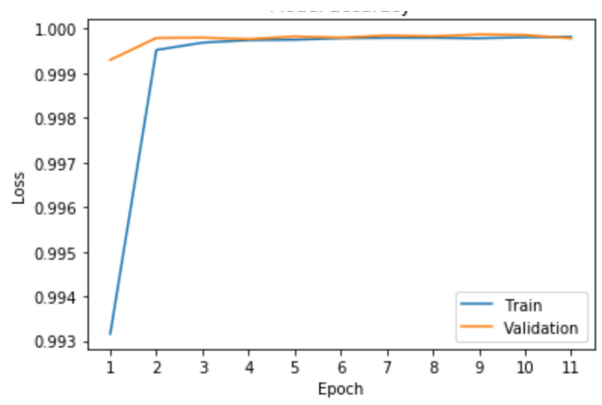


Fig. 9. Botnet classification learning curve

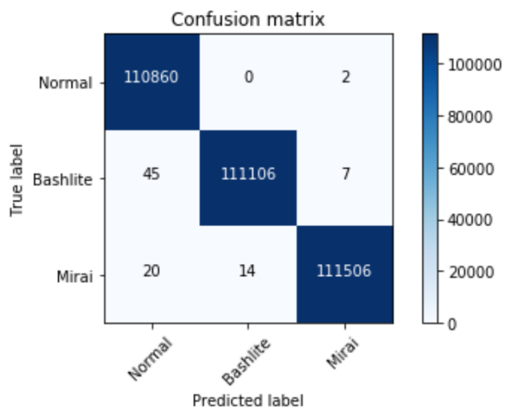


Fig. 10. Botnet classification confusion matrix

2) Mirai attack type classification:

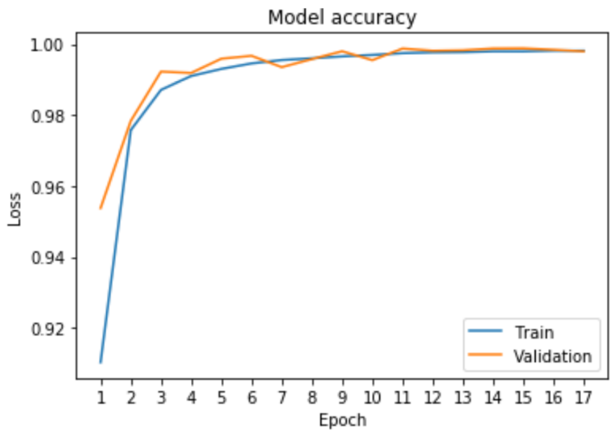


Fig. 11. Mirai attack classification learning curve

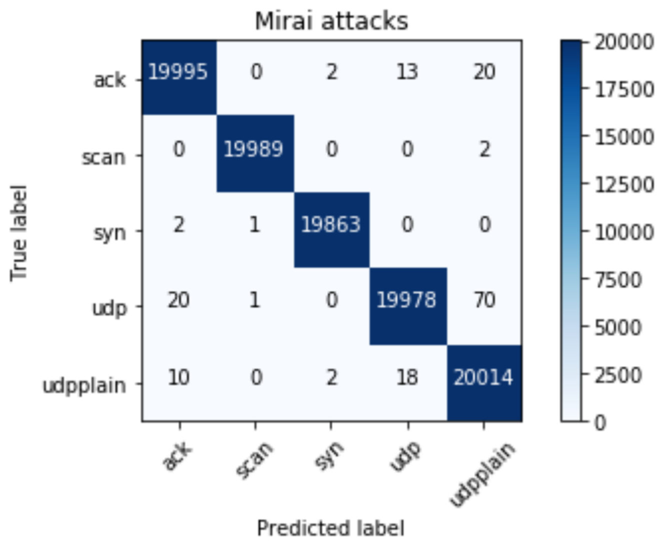
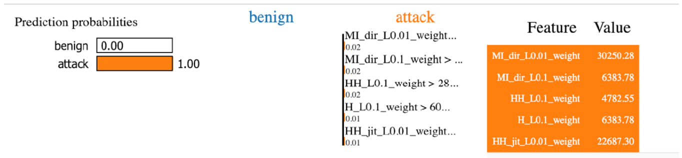


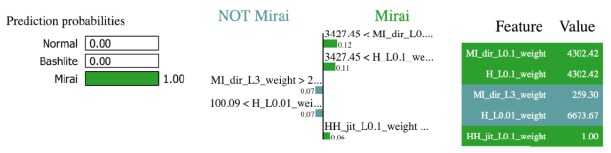
Fig. 12. Mirai attack classification confusion matrix

D. Local Interpretable Model-Agnostic Explanations



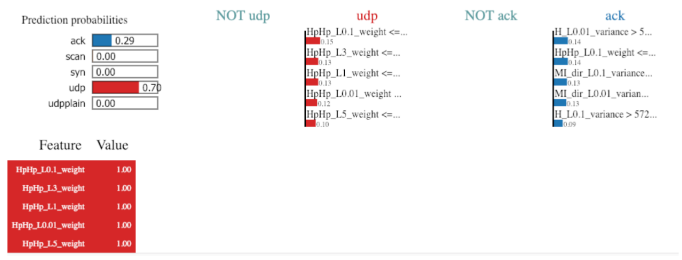
Feature	Value
MI_dir_L0.01_weight	30250.28
MI_dir_L0.1_weight	6383.78
HH_L0.1_weight	4782.55
H_L0.1_weight	6383.78
HH_jit_L0.01_weight	22687.30

Fig. 13. LIME explanation for attack detection



Feature	Value
MI_dir_L0.1_weight	4302.42
H_L0.1_weight	4302.42
MI_dir_L3_weight	259.30
H_L0.01_weight	6673.67
HH_jit_L0.1_weight	1.00

Fig. 14. LIME explanation for botnet classification



Feature	Value
HpHp_L0.1_weight	1.00
HpHp_L3_weight	1.00
HpHp_L1_weight	1.00
HpHp_L0.01_weight	1.00
HpHp_L5_weight	1.00

Fig. 15. LIME explanation for Mirai attack type UDP

VI. Implementation in Real Environment

There would be however, many practical considerations. The anomaly detection part of the pipeline should be easier to implement in practice, as there would be no need to obtain and maintain a data set for different botnets and their attacks. Of course, sampling traffic from an uninfected IoT network will still be needed, which could be a problem in and of itself. Further, it is not clear, how the method performance would change if the network itself is changed, for example: new devices are added. If it's the same sort of device system as the one we already have in the network. As data set actually contains multiple devices from similar categories. In fact, even with a theoretically low false positive rate of 0.0015, more than 1 in 1000 regular data points will be incorrectly classified as an attack. If we produce a data point every 100 milliseconds, we'll get a false alarm every 100 seconds, which isn't realistic. However, in an environment where a continuous stream of packets is normally sent, one data point by itself can hardly be considered an attack.To that regard, an approach suggested by [2]could be implemented, where an alarm only goes off if there is a majority of attack data points within a window of some specified size. While the autoencoder-based attack detection mechanism could be built locally, maybe by an IoT network engineer, the botnet classifier poses a challenge in the form of data set requirements. This is ideally a job for a cyber security company or a research university team that specializes in IoT botnets and gathers and manages the data sets required to train an efficient classifier that can identify a variety of botnets and attack forms.

VII. Conclusion

Because of the rapid growth of IoT technologies, cyber-attacks are increasingly focusing on these computers. In these conditions, botnet attacks, in particular, are extremely difficult. After infecting the botnets, the attackers will use the C&C server to monitor the devices and launch attacks against the victim hosts. There is some difficulty in capturing normal traffic behavior since it differs across IoT devices, and this difficulty may be related to the device's capabilities [1], and [2] the network communications it normally produces. A similar notion was raised by H. Bostani and M. Sheikhan [2], who argued that the specific functionality of today's IoT devices leads to predictable and obvious behaviors. In turn, the ease of establishing baseline behaviors for IoT devices facilitates anomaly detection to detect attacks.

Deep learning approaches demonstrated high accuracy in detecting and classifying IoT botnet attacks. Furthermore, these approaches can operate with a variety of features and, in general, their output is unaffected by additional features, implying that in a real-world setting, they have the ability to use all available data features.

Our proposed attack detection showed excellent results with an accuracy of 0.9991 and false positive rate of 0.0015. Autoencoders can be used to detect anomalies in data coming from a variety of different IoT devices. The LIME technique was applied to an attack data point that gave an intuitive interpretation of the features and predicted output. The two attack classifications: botnet type classification and Mirai attack type classification proved good results with accuracies over 0.99. The aim to increase predictability of traffic behaviour as anomalous was achieved.

Finally, large organisations should use a high predictability score to ensure network functionality and minimize the effect that infected devices can have on the network. That is, security policies can not enable IoT devices with low predictability scores to link to their networks because they complicate attack detection. For future work, more devices can be included and some other unconventional attack types can be studied.

References

[1] M. Özçelik, N. Chalabianloo, and G. Gür, “Software-Defined Edge Defense against IoTBased DDoS,” Proc. 2017 IEEE Int’Conf. Computer and Information Technology (CIT17), 2017; doi.org/10.1109/CIT.2017.61.

[2] H. Bostani and M. Sheikhan, “Hybrid of Anomaly-Based and Specification-Based IDS for Internet of Things Using Unsupervised OPF Based on MapReduce Approach,” Computer Comm., vol. 98, 2017, pp. 52–71.

[3] D.H. Summerville, K.M. Zach, and Y. Chen, “Ultra-Lightweight Deep Packet Anomaly Detection for Internet of Things Devices,” Proc. 2015 IEEE 34th Int’l Performance Computing and Comm. Conf. (IPCCC 15), 2015; doi.org/10.1109/PCCC.2015.7410342.

[4] S. García, A. Zunino, and M. Campo, “Survey on Network-Based Botnet Detection Methods,” Security and Communication Networks, vol. 7, no. 5, 2014, pp. 878–903.

[5] K. Veeramachaneni et al., “AI2̂: Training a Big Data Machine to Defend,” Proc. IEEE 2nd Int’l Conf. Big Data Security on Cloud, IEEE Int’l Conf. High Performance and Smart Computing, and IEEE Int’l Conf. Intelligent Data and Security (BigDataSecurity-HPSC-IDS 16), 2016; doi.org/10.1109/BigDataSecurity-HPSC-IDS.2016.79.

[6] A. Tuor et al., “Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams,” Proc. AAAI 2017 Workshop on Artificial Intelligence for Cybersecurity, 2017; https://arxiv.org/pdf/1710.00811.pdf.

[7] M. Stevanovic and J. M. Pedersen, ”On the Use of Machine Learning for Identifying Botnet Network Traffic,” Journal of Cyber Security, vol. 4, pp. 1-32, 2016.

[8] A. A. Pol, ”Anomaly detection using Deep Autoencoders for the assessment of the quality of the data acquired by the CMS experiment,” in 23rd International Conference on Computing in High Energy and Nuclear Physics, Sofia, 2018

[9] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[10] C. Seaman, ”Threat Advisory: Mirai Botnet,” [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-theinternet/ akamai-mirai-botnet-threat-advisory.pdf. [Accessed 6 January 2019].

[11] Y. Meidan, et al, ”N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders,” IEEE PERVASIVE COMPUTING, vol. 18, no. 9, 2018.

[12] Y. Meidan, et al , ”iot botnet dataset,” [Online]. Available: https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT. [Accessed 27 December 2018].