AI For New Generation of IoT Intrusion Detection Systems

Ulises Almaguer Guzmán*
*School of Computer Science, Tecnológico de Monterrey, Campus Querétaro

ABSTRACT During the last decades, there has been a rapid development of the IoT technology, therefore it has been targeted by malicious third parties. Although IoT can heavily increase productivity and efficiency through intelligent and remote management, it also increases the risk of several cyber-attacks, mainly caused to the lack of robust security measures in the IoT ecosystems and devices [1]. This paper aims to provide an effective way to detect several types of attacks by applying Machine Learning (ML) algorithms to detect annormal behavior in the IoT network traffic, thus, the research is based on a new dataset, Bot-IoT, which incorporates legitimate and simulated IoT network traffic and various types of attacks [2]. The Bot-IoT dataset was developed by N. Koroniotis *et al*; and can be accessed at [3].

I. INTRODUCTION

The development of the Internet and an interconnected world has increased the development rate of the Internet of Things (IoT). Nevertheless, this has also increased the risk to cyber-attacks, mainly caused because the IoT devices consist of lightweight communication protocols, resource constrained devices (having limited computations power and storage), and there is a lack of standardization in IoT systems [1] [2].

In order to identify different cyber threads against IoT devices and ecosystems, it is of vital importance to develop cyber-security applications such as intrusion prevention system (IPS) that are tailored made to this type of devices and that meet their specific requirements [1].

To set a little background, an intrusion detection systems or IDS is considered a second-line defense that can be device or software based, whose primary function is to monitor systems and network events to detect possible malicious activities, abnormal network traffic, or policy violations that successfully evaded security perimeters, such as firewalls [1] [4].

Therefore, the development and evaluation of intrusion detection methods is essential in the design and development of IoT-based intrusion detection methods. However, the lack of availability of state-of-the-art datasets of IoT devices is mainly due to privacy issues, in which companies who created IoT datasets have not shown any interest in sharing the data with the community [1].

Nevertheless, over the years several datasets have been developed, each one having its own advantages and disadvantages. Some lack of reliably labeled data, poor attack diversity, redundancy traffic or missing ground truth [2]. However, N. Koroniotis, *et al*, in their work *Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset* addresses such challenges by having a realistic testbed, multiple tools used to carry out several botnet scenarios and organized packet capture files in directories, based on attack types [2]. Thus, the

Bot-IoT dataset is the one in which this study will be performed.

This study is structured as follows. The review of how the dataset Bot-IoT is composed is describe in Section 2. In the section 3 the logistic regression model applied to the analysis of the dataset is described. Section 4 discusses the implementation of the deep learning algorithm LSTM (Long Short-Term Memory) applied for anomaly detection and mitigation in software-defined network environments. Finally, in section 5, experimental results are presented and discussed, followed by the future work suggested by this paper.

II. BOT-IOT DATASET & BACKGROUND & RELATED WORK

IoT networks are comprised of both typical network elements such as workstations, laptops, and router, but also of IoT devices [2, 5, 6]. The vulnerabilities of IoT networks significantly increases with complex cyber-attacks, such as Botnets. Botnets are synchronized networks of compromised machines, called bots. In a Botnet, the attacker, called *botmaster*, maintains a bi-directional channel of communication towards the compromised network, through a Command and Control (C&C) infrastructure., By using the C&C a botmaster issues commands to the rest of the botnet and can receive feedback regarding attacks in progress, such as *Distributed Denial of Service* attacks (DDoS), *Keylogging*, *Phishing*, *Click fraud*, *Identity theft* and even the proliferation of other Bot malware. The compromised bots are under the complete control of the botmaster [7].

Digital Forensics oversees the identification, preservation, collection, examination, analysis and presentation of the digital evidence, whose forensic methods are applied to data, such techniques heavily rely on the usage of Machine Learning (ML) algorithms for designing reliable models that can efficiently determine cybercrimes [2, 8]. Thus, the most common cyber applications of network forensic-based ML are Intrusion Detection Systems (IDS).

An intrusion detection system may be classified in two main categories, network or host intrusion detection systems. Network IDS (NIDS) are commonly placed in strategic positions of the network (usually in a spot connecting the internal network with the Internet) and are issued to scan the inbound and outbound traffic of a network for well-known patterns of attack. When an attack is detected, notifications are set off and subsequent mechanisms can be deployed. Such NIDS can incorporate ML algorithms in their process of anomaly detection, such as classification, clustering, LSTM, among others [9, 10, 11].

Moreover, several testbeds have been developed to tackle the development of NIPS based on ML algorithms. However, most of the current datasets lack of a balance between attack and normal traffic. That is the main reason that for the purpose of the study, the Bot-IoT dataset is the one chosen to tackle this issue, this mainly because this dataset includes a variety of Botnet activities, including but not limited to DDoS, DoS and Port Scanning [2]. The test bet was developed using four Kali Linux machines to simulate both DoS and DDoS attacks, along with other botnet actions. The environment includes the introduction of IoT-elements though the middleware, *Node-red*, to simulate the existence of IoT devices in the virtual network. The IoT simulated traffic was in the form of *MQTT (Message Queuing Telemetry Transport)*, a communication protocol implemented over *TCP/IP*, which is often used in light weighted network communications, such as IoT. Regarding the generation of realistic network traffic, the *Ostinato Software* was used, allowing to make

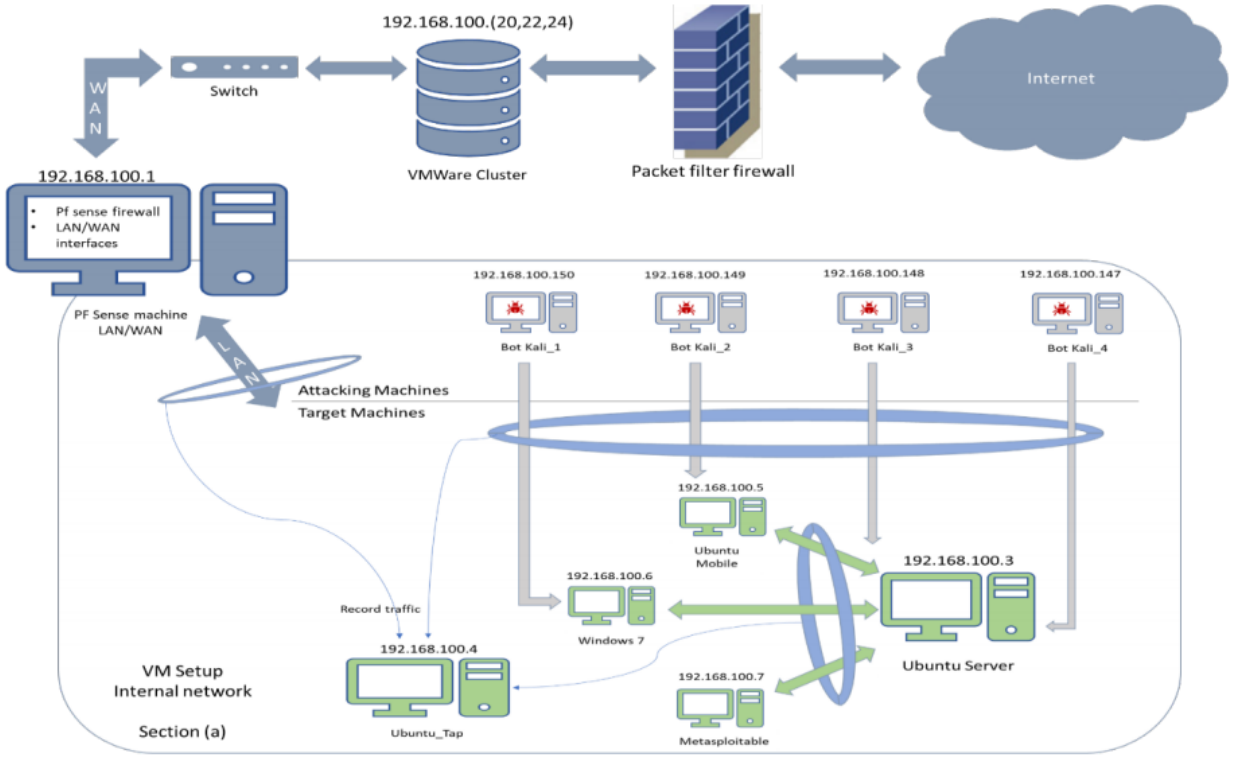

Figure 1 Testbed environment of the Bot-IoT dataset

the dataset appear as if the data was collected from a real-world network [2, 12]. Figure 1 depicts the testbed of the Bot-IoT dataset, where several *VM's* are connected to *LAN* and *WAN* interfaces in the cluster and linked to the Internet through *PFSense* machine. The *Node-red* tool was used for simulating various IoT sensors which were connected with the public IoT hub, AWS.

Node-red is pa popular middleware used to connect IoT physical devices with their backend cloud server and applications, improving and speeding communications between the various parts of an IoT deployment [2]. The MQTT protocol was used as a lightweight communication protocol that links machine-to-machine communications. Thus, the IoT scenarios in the testbed include:

1. A weather station.
2. A smart fridge.
3. Motion activated lights.
4. A remotely activated garage door.
5. A smart thermostat

The generated dataset includes the following features:

1. *pkSeq*: row identifier.
2. *Stime*: record start time.
3. *flgs*: flow state flags seen un transactions.
4. *flgs_number*: numerical representation of feature flags.
5. *proto*: textual representation of transaction protocols present in network flows.
6. *proto_number*: numerical representation of feature proto.
7. *saddr*: source IP address.
8. *sport*: source port number.
9. *daddr*: destination IP address.
10. *dport*: destination port number.
11. *pkts*: total count if packets in transaction.
12. *bytes*: total number of bytes in transaction.
13. *state*: transaction state.
14. *state_number*: numerical representation of feature state.

15. *ltime*: record last time.
16. *seq*: argus sequence number.
17. *dur*: record total duration.
18. *mean*: average duration of aggregated records.
19. *stddev*: standard deviation of aggregated records.
20. *sum*: total duration of aggregated records.
21. *min*: minimum duration of aggregated records.
22. *max*: maximum duration of aggregated records.
23. *spkts*: source-to-destination packet count.
24. *dpkts*: destination-to-source packet count.
25. *sbytes*: source-to-destination byte count.
26. *dbytes*: destination-to-source byte count.
27. *rate*: total packets per second in transaction.
28. *srate*: source-to-destination packets per second.
29. *drate*: destination-to-source packets per second.
30. *attack*: class label (0 for normal traffic, 1 for attack traffic)
31. *category*: traffic category.
32. *subcategory*: traffic subcategory.

In the dataset, attack instances are labeled with 1 and normal instances are labeled with 0 for training and validating machine learning models. However, further new features were added to help improving the capabilities of prediction of the classifiers, based on the previously mentioned features:

1. TnBPSrcIP: total number of bytes per source IP.
2. TnBPDstIP: total number of bytes per destination IP.
3. TnP_PSrcIP: total number of packets per source IP.
4. TnP_PDstIP: total number of packets per destination IP.
5. TnP_PerProto: total number of packets per protocol.
6. TnP_Per_DPort: total number of packets per dport.
7. AR_P_Proto_P_SrcIP: average rate per protocol per source IP (pkts / dur).
8. AR_P_Proto_P_DstIP: average rate per protocol per destination IP.
9. N_IN_Conn_P_SrcIP: number of inbound connections per source IP.
10. N_IN_Conn_P_SrcIP: number of inbound connections per destination IP.
11. AR_P_Proto_P_Sport: average rate per protocol per sport.
12. AR_P_Proto_P_Dport: average rate per protocol per dport.
13. Pkts_P_State_P_Protocol_P_DestIP: number of packets grouped by stated of flows and protocols per destination IP.
14. Pkts_P_State_P_Protocol_P_SrcIP: number of packets grouped by stated of flows and protocols per source IP.

All the scenarios represented in the data set are one of both, benign or botnet scenarios.

The benign scenarios include huge amount of traffic simulating normal traffic between the VM's. Also, there is traffic representations for:

1. Smart fridge.
2. Smart garage door.
3. Weather monitoring system.
4. Smart lights.
5. Smart thermostat.

The botnet scenarios include:

1. Probing attacks: malicious activities that gather information about victims through scanning remote systems, also called fingerprinting. It is splitted into

active and passive probing. During passive probing, an attacker simply captures any and all available packets in the network, thus operating stealthy manner [2, 13, 14, 15, 16]. On the other hand, during active probing, the attacker generates network traffic, targeting the system, and records the responses, comparing them with known responses which allow them to make inferences about services and OS. Two major subcategories are founded:

   a. Port scanning: a scanner identifies the services which run behind the system ports (0-65535) by sending request packets [15].
   b. OS fingerprinting: a scanner gathers information about the remote systems OS by comparing its responses to pre-existing one or based on difference in TCP/IP stack implementations [2, 15].

2. Denial of Service: are malicious activities that attempt to disrupt a service, thus making it unavailable to legitimate users. The DDoS and DoS attacks are performed by a group pf compromised machines called Bots and target a remote machine, usually a server. The purpose is the disruption of services accessible by legitimate users [17].
3. Information Theft: is a group of attacks where an adversary seeks to compromise the security of a machine in order to obtain sensitive data. The information theft attack types can be splitted in data theft and keylogging. The former refers when an adversary targets a remote machine and attempts to compromise it, thus gaining unauthorized access to data, which can be downloaded to the remote attacking machine. The latter is when an adversary compromises a remote host in order to record a user's keystrokes, potentially stealing sensitive credentials [2, 18, 19].

II. LOGISTIC REGRESSION IMPLEMENTATION

For experimental purposes, the logistic implementation is implemented as a first approach to predict the different type of attack.

Thus, the types of attacks that are implemented using the logistic regression approach are the DDoS and DoS attacks.

It is worth mentioning that due to the huge amount of data, the implementation consists of a vectorized version of the algorithm, in order to improve the runtime, using the NumPy capabilities.

The explanation of the vectorized implementation of a logistic regression is mentioned below.

In order to vectorize the logistic regression it is required to vectorize several parts of the algorithm such as:

1. The sigmoid function
2. The hypothesis function
3. The gradient descent function

Vectorizing Sigmoid Function

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

Figure 2 Sigmoid Function

Figure 2 shows how the Sigmoid function works for one value of *z*. As it can be seen, the value of *z* corresponds to the value given by the hypothesis function when multiplying the parameters by the values of the features. However, in order to vectorize this function, the use of NumPy function **exp()** comes into mind [20]. Thus, the value of *z* is transformed in order

to become not only the value of the hypothesis function for a single instance, but rather a vector storing the values of the hypothesis function applied for each instance. Therefore, the result of the vectorized Sigmoid function ***g(z)*** is in fact a vector that matches the number of evaluations of the Sigmoid function to each one of the hypotheses of each one of the instances [20].

Vectorizing the Hypothesis Function

$$h_\theta(x^{(i)}) = \theta^T x^{(i)} = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \dots \\ x_n^{(i)} \end{bmatrix}$$

Figure 3 Vectorized Hypothesis Function

Figure 3 shows the implementation of the vectorized hypothesis function for a single instance of the data set. As it can be seen the inner values (dimensions) of the matrixes to multiply must match, while the resulting matrix will be in the form of the outer values (dimensions) of the matrixes to multiply [20]. As it can be seen this operation is the dot product of two vectors. This vectorization is possible due to the fact that each feature is only multiplied by its corresponding parameter, allowing it highly parallelizable and thus allowing runtime optimizations. However, the implementation for multiple instances is shown below [20]:

$$h_\theta(x) = X\theta = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & & x_n^{(2)} \\ \vdots & \vdots & \dots & \vdots \\ x_0^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Figure 4 Vectorized Hypothesis Function For Dataset

Vectorizing the Gradient Descent

When following an iterative approach, in order to achieve the optimization results, it is required at least 4 nested loops to achieve the gradient descent algorithm, resulting in a very inefficient code when managing large datasets [20]. Therefore, the implementation of the vectorized gradient descent is done following the next equation [20]:

$$\nabla J(\theta) = \frac{1}{m} \cdot X^T \cdot \left(g(X \cdot \theta) - \vec{y}\right)$$

Figure 5 Gradient Descent

Figure 5 shows the vectorized version of the gradient descent, which consists of a dot product of the values of the instances and the difference between the prediction of the model and the true expected value of *y* [20].

III. LSTM IMPLEMENTATION

The deep learning algorithm to be implemented for IDS is based on LSTM (Long-Short Term Memory) and Recurrent Neural Networks. The main reason why LSTM was chosen is due to the fact that LSTM networks are capable of solving the vanishing gradients problem (gradients becoming smaller and smaller, therefore, the parameter updates become very small, and the learning process takes more time with no beneficial learning effectiveness). The deep learning algorithm was implemented using Keras framework [21].

For the purpose of this paper, 4 different architecture models have been tested and modified to compare the training and test accuracy [21].

The first model is composed of a LSTM layer with 64 unit, tanh as activation function and a kernel regularizer of 12, and two dense layers. The first dense layer is 128 unit with relu as activation function and a kernel regularizer of 12. The second dense layer is 1 unit with sigmoid as activation function and a kernel regularizer of 12 [21].

The second model is composed of a Bidirectional LSTM layer with 128 unit, tanh as activation function and a kernel regularizer of 12, and two

dense layers. The first dense layer is 128 unit with relu as activation function and a kernel regularizer of 12. The second dense layer is 1 unit with sigmoid as activation function and a kernel regularizer of 12 [21].

The third model is composed of two LSTM layers with 128 unit, tanh as activation function and a kernel regularizer of 12 each, and two dense layers. The first dense layer is 128 unit with relu as activation function and a kernel regularizer of 12. The second dense layer is 1 unit with sigmoid as activation function and a kernel regularizer of 12 [21].

The fourth model is composed of three LSTM layers with 128 unit, tanh as activation function and a kernel regularizer of 12 each, and two dense layers. The first dense layer is 128 unit with relu as activation function and a kernel regularizer of 12. The second dense layer is 1 unit with sigmoid as activation function and a kernel regularizer of 12 [21].

All models were compiled with the binary_crossentropy as loss function and Adam as an optimizer and ran for a total of 50 epochs [21].

IV. EXPERIMENTAL RESULTS

Experimental Results for Logistic Regression

In order to show the experimental results, it is required to establish the testing parameter on which the logistic regression is going to be tested.

First, the initial parameters for each test are randomly generated, allowing to verify that the algorithm is finding the minima using the gradient descent. Second, the number of epoch that the algorithm is going to be tested begins for a 100 epochs scaling up to 2000 epochs. Third, the value that is going to be changing is the learning rate, decreasing in order to avoid looping in the same gradient descent.

The results show two parameter configurations per each run, the parameters that show the best accuracy and the lowest loss and the parameters generated after lopping for a thousand epochs.

The results shows the accuracy and the loss computed and graphed. However, the confusion matrix is only generated for the resulting parameters for the testing.

The results are shown in this report.

Analysis of the Report for Logistic Regression

As it is shown in the report when training the algorithm for DDoS and DoS attacks, with a highest learning rate (0.003) and a few number of epochs (100) the value of the loss and the accuracy of the model started oscillating between two values. Therefore, that indicates that we are looping trying to find the local minima, but the gap created by the gradient descent to optimize the parameters was huge enough to avoid finding the local minima and just loop between the previously computed parameters. Thus, the generated accuracy in the DDoS model was of 54.6% while the DoS model's accuracy was of 61.58%.

Therefore, for the subsequent trainings of the algorithm the learning rate was reduced, and the number of epochs was increased. As we can see in the training of the DDoS model, with a learning rate set to 0.0001 and the number of epochs set to 2000, the model achieved an accuracy of 64.92%, achieving an improvement of 10.32%. On the other hand, when training the DoS model with a learning rate set to 0.0001 and the number of epochs set to 2000, the model achieved 72.16% of accuracy, achieving an improvement of 10.58%.

In general, reducing the learning rate and increasing the number of epochs achieved an improvement of 10% in the accuracy when comparing the first trained model (with learning rate set to 0.003 and looping for 100 epochs) and

the last trained model (with a learning rate set to 0.00001 and looping for 2000 epochs).

However, the interesting part of the analysis comes when analyzing the test results in which no matter wat type of model is being tested, the result always show a 50% of accuracy while having a confusion matrix evenly distributed having the same number of true positives and false positive and the same number of true negatives and false negatives.

There may be several reasons but one of the most common is that the algorithm is underfitting, this may be cause because the complexity of the model does not satisfy the complexity of the data, so the predictions are not the best ones that can be computed. This may involve selecting a more powerful model, selecting more parameters, finding better features to learn or reduce the constraints of the model. Another reason may be that the dataset has several number of instances that are the opposite, cancelling each other when testing and resulting in a 50% of accuracy.

In order to assess this issues, the next step of this research is the implementation of a deep learning algorithm, more specific, the LSTM model, which come in handy due to the fact that it is a recurrent neural network, allowing to detect different types of attack taking into consideration the source IP address the source port and the destination IP address and destination port, thus, identifying flow of packets corresponding to different types of attacks.

Experimental Results for LSTM

The first model shows a training accuracy of 98.31% and a test accuracy of 98.0145%.

The second model shows a training accuracy of 98.32% and a test accuracy of 98.249%.

The third model shows a training accuracy of 98.41% and a test accuracy of 98.384%.

The fourth model shows a training accuracy of 98.34% and a test accuracy of 98.5646%.

$$accuracy = \left(\frac{TP + TN}{TP + TN + FP + FN}\right)$$

Figure 6 Equation for accuracy

For each of the model the confusion matrix where computed, where:

- True Positive (TP): Attack record classified correctly as attack [21].
- True Negative (TN): Benign record classified correctly as benign [21].
- False Positive (FP): Benign record classified incorrectly as attack. [21]
- False Negative (FN): Attack record classified incorrectly as benign [21].

The results are shown in this report.

V. FUTURE WORK

When DoS and DDoS attack are launched, it floods the network traffic making all services unavailable. Thus, the existence of an Intrusion detection and Preventing Systems (IDPS) have little chance to stand-up for DoS or DDoS attacks, although must IDPS use more than one detection methodologies to classify the traffic into two categories, signature-based or anomaly-based [21].

Both DoS and DDoS attacks seek to exploit the weaknesses of such detection methodologies [21].

Signature-based methodologies are also known as ruled-based. The attack is detected by comparing well-known attack signatures, patterns or malicious intrusion sequences used by malware such as byte sequences with the monitored network traffic [21]. A match generates an alarm for a potential attack. This type of mechanism has a fast detection time and

detects most known attacks. Generally, signature-based IDS have low false positive rate. Signature-based detection systems are based on well-known attack patters that are mostly malformed packets and protocol attacks [21].

On the other hand, anomaly-based methodologies are also known as behavior-based detection and operate by comparing the network traffic behavior against previous normal traffic behavior, any deviation in the comparison is a sign of an attack [21]. The system acquires a normal traffic profile, usually through training and monitors the traffic for any differences from the normal profile [21]. Anomaly-based detection can detect unknown attacks; however, it generally produces higher false positive rates than signature-based systems [21].

For future work, this paper recommends a hybrid approach, combining a network system that has both, signature-based systems, and anomaly-based systems [21].

This approach allows to benefit from the strengths, while decreasing the weaknesses of each type of systems [21]. If an attacks passed IDS network sensors without detection (new IP address) and reached a signature-based detector without detection, then it does not match any signature based attacks [21]. However, the behavior of the attack is already traced and carefully monitored by the anomaly-based detector [21].

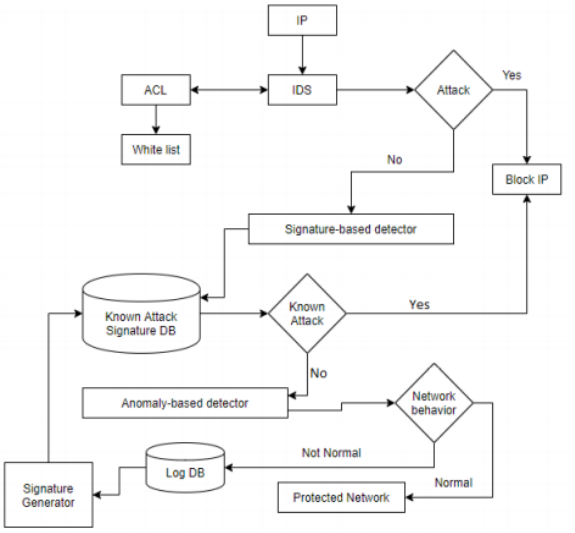

Figure 7 Hybrid network architecture for detecting attacks

Figure 7 shows a hybrid approach combining signature-based and anomaly-based detection systems [21].

If the network behavior is normal during IP request time, it will be announced as a legitimate IP and approved to get into the secure network [21]. On the other hand, if an abnormal behavior is detected at any stage, the IP will be blocked [21].

The IDS sensors will detect an attack if the incoming IP request is a known attack or part of an attack segment based on a stored signature attacks [21]. It will generate an alarm to the router and blocks the IP address source, then dynamically update the Access Control List (ACL) on the router to add the IP address to the blacklist Ips, while if not detected as a thread, it proceeds to the next detection border [21].

Secondly, the signature-based detector will compare the signature of the incoming traffic with well-known attacks signature in its data base [21]. If a match is found, an alarm for a potential attack is triggered and the IP is blocked form accessing the secure network. If no match is found, the packer proceeds to the next detection stage [21].

Thirdly, the anomaly-based detector operates by observing the byte sequence of the incoming network traffic behavior, analyzing the previous and the new bytes sequences for a define time interval and comparing the analysis against what is consider normal traffic [21]. If any deviation is detected in the comparison, it is flagged as an attack. Therefore, it will update the blacklist IP database to store the newly detected attack record and generate the signature of this attack and update the signature-based database [21]. Hereafter, the attack will be known, resulting in blocking the IP address source using signature-based detection quickly [21].

VI. REFERENCES

[1] TON_IoT Telemetry Dataset: A New Generation Dataset of IoT and IIoT for Data-Driven Intrusion Detection Systems

[2] Towards the Development of realistic Botnet dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset

[3] Bot-iot (2018). URLhttps://www.unsw.adfa.edu.au/unsw-canberra-cyber/ cybersecurity/ADFA-NB15-Datasets/bot_iot.php

[4] https://www.barracuda.com/glossary/intrusion -detection-system

[5] S. S. Silva, R. M. Silva, R. C. Pinto, R. M. Salles, Botnets: A survey, Computer Networks 57 (2) (2013) 378–403.

[6] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, S. A. Khayam, A taxonomy of botnet behavior, detection, and defense, IEEE communications surveys & tutorials 16 (2) (2014) 898–924.

[7] P. Amini, M. A. Araghizadeh, R. Azmi, A survey on botnet: classification, detection and defense, in: Electronics Symposium (IES), 2015 International, IEEE, 2015, pp. 233–238.

[8] G. Palmer, A road map for digital forensic research: Report from the first digital forensic workshop, 7–8 august 2001, DFRWS Technical Report DTR-T001-01.

[9] E. Hodo, X. Bellekens, A. Hamilton, P.-L. Dubouilh, E. Iorkyase, C. Tachtatzis, R. Atkinson, Threat analysis of iot networks using artificial neural network intrusion detection system, in: Networks, Computers and Communications (ISNCC), 2016 International Symposium on, IEEE, 2016, pp. 1–6.

[10] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, E. Vázquez, Anomaly-based network intrusion detection: Techniques, systems and challenges, computers & security 28 (1-2) (2009) 18–28. 36

[11] K. Wang, M. Du, Y. Sun, A. Vinel, Y. Zhang, Attack detection and distributed forensics in machine-to-machine networks, IEEE Network 30 (6) (2016) 49–55.

[12] Ostinato. URLhttps://ostinato.org/

[13] S. Paliwal, R. Gupta, Denial-of-service, probing & remote to user (r2l) attack detection using genetic algorithm, International Journal of Computer Applications 60 (19) (2012) 57–62.

[14] G. Bartlett, J. Heidemann, C. Papadopoulos, Understanding passive and active service discovery (extended), Tech. rep., Technical Report ISI-TR2007-642, USC/Information Sciences Institute (2007).

[15] N. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, J. K. Kalita, Network attacks: Taxonomy, tools and systems, Journal of Network and Computer Applications 40 (2014) 307–324.

[16] G. F. Lyon, Nmap network scanning: The official Nmap project guide to network discovery and security scanning, Insecure, 2009.

[17] S. T. Zargar, J. Joshi, D. Tipper, A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks, IEEE communications surveys & tutorials 15 (4) (2013) 2046–2069.

[18] C. Tankard, Advanced persistent threats and how to monitor and deter them, Network security 2011 (8) (2011) 16–19.

[19] A. Jesudoss, N. Subramaniam, A survey on authentication attacks and countermeasures in a distributed environment, Indian J Comput Sci Eng IJCSE 5 (2014) 71–77.

[20] Y, Liu. (2018, Oct 13). Vectorization Implementation in Machine Learning. [Online]. Available: https://towardsdatascience.com/vectorization-implementation-in-machine-learning-ca652920c55d

[21] Shurman, Mohammad & Khrais, Rami & Yateem, A.Rahman. (2020). DoS and DDoS Attack Detection Using Deep Learning and IDS. International Arab Journal of Information Technology. 17. 655-661. 10.34028/iajit/17/4A/10.