Received 1 July 2023, accepted 19 July 2023, date of publication 21 July 2023, date of current version 16 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3297977

RESEARCH ARTICLE

IoT Network Cybersecurity Assessment With the Associated Random Neural Network

EROL GELENBE[1,2], (Life Fellow, IEEE), AND MERT NAKIP[1], (Student Member, IEEE)
[1]Institute of Theoretical and Applied Informatics, Polish Academy of Sciences (PAN), 44100 Gliwice, Poland
[2]Université Côte d'Azur, CNRS I3S, 06103 Nice, France

Corresponding authors: Erol Gelenbe (seg@iitis.pl) and Mert Nakıp (mnakip@iitis.pl)

This work was supported by the European Commission (EC) Horizon 2020 Project IoTAC (Security By Design IoT Development and Certificate Framework with Front-end Access Control) under Grant 952684.

ABSTRACT This paper proposes a method to assess the security of an n device, or IP address, IoT network by simultaneously identifying all the compromised IoT devices and IP addresses. It uses a specific Random Neural Network (RNN) architecture composed of two mutually interconnected sub-networks that complement each other in a recurrent structure, called the Associated RNN (ARNN). For each of the n devices or IP addresses in the IoT network, two distinct neurons of the ARNN advocate opposite views: compromised or not compromised. The fully interconnected $2n$ neuron ARNN structure of paired neurons learns offline from ground truth data. Thus rather than requiring a separate attack detector at each network node, the ARNN offers a single overall attack detector that observes the incoming traffic at each node, learns about the interdependencies between network nodes, and formulates a recommendation for each device or IP address in an IoT network. The ARNN weight initialization and learning algorithm are discussed, and the ARNN performance is evaluated using real attack data, and compared against several learning and testing techniques. Results are obtained both for off-line learning with ground truth data, and for on-line incremental learning using a simplified average metric measured from incoming packet traffic. Comparisons with the best state-of-the-art techniques show that the ARNN significantly outperforms previously known approaches.

INDEX TERMS Internet of Things (IoT), cybersecurity, botnets, machine learning, associated random neural network, MIRAI attacks.

I. INTRODUCTION

A "Botnet" is a cyberattack that can spread Distributed Denial of Service (DDoS) attacks and malware [1], [2] over thousands of devices [3], by targeting IoT devices or IP addresses, and installing malware on its victims, which in turn may become "bots" which generate malicious traffic and spread the malware further to yet other devices [4]. As an example, in 2016, the massive MIRAI DDoS Botnet targeted Domain Name System (DNS) provider Dyn [5] and gained access to servers of several leading cybersecurity companies [6].

Botnets increase network congestion through additional traffic that overwhelms the communication ports of the devices they attack, but they also exploit the physical and logical resources of victim devices, including their batteries, processors, memory, operating systems and network software [7]. Therefore, it is crucial to rapidly identify both compromised IoT devices and malicious packets during a Botnet attack to prevent its propagation and stop it before it can do a huge amount of harm. However, other forms of attacks can target simple IoT systems as well, causing a lot of harm.

The associate editor coordinating the review of this manuscript and approving it for publication was Amjad Ali.

A. RELATED WORK

Botnet Attack Detection. In early work on Botnet attacks, their source code [8] and capabilities [5], as well as other characteristics of these attacks have been examined [9]. In order to detect Botnet attacks, recent research has used ML and deep learning approaches, such as Logistic Regression [10], the Multi-Layer Perceptron (MLP) [11], [12], [13], [14], [15], Classification and Regression Trees (CART) [16], Gradient Boosting [17],Long-Short Term Memory (LSTM)-based techniques [18], [19], and sparse representation [20]. In

[21], the MLP and the Convolutional Neural Network (CNN) were employed with learning on the focal loss to detect IoT intrusions, while in [22] Naive Bayes was combined with an evolutionary feature selection method to develop a signature-based system to detect Botnet, DDoS, and port scan attacks. In [23], a Botnet attack detection system that classifies the network traffic using beta mixture model based on a set of statistically extracted traffic features is discussed.

Recent research on cyberattack detection has also used self-supervised learning systems. For example, in [24], a self-supervised learning algorithm combining LSTM with CNN was developed for anomaly-based attack detection for networks inside vehicles. The bidirectional Generative Adversarial Network (GAN) was used for anomaly detection [25], and a Graph Neural Network (GNN) network-level IDS was developed in [26].

Accurate results have also been obtained for the Deep Random Neural Network (DRNN) with offline [27] and incremental [28] learning to detect MIRAI attacks, while the DRNN was shown to achieve high performance for detecting different types of unknown attacks simultaneously [29]. Earlier work [30] examined the performance of the classical Random Neural Network (RNN) [31] with offline gradient-descent learning [32] to detect SYN denial of service (DoS) attacks. On the other hand, whereas the work reviewed above focused on detecting cyberattacks and malicious traffic, in this paper, we develop an ARNN-based decision system that identifies the compromised IoT nodes. Successful identification of compromised nodes along with malicious traffic paves the way to fend off distributed attacks (e.g. Botnet and DDoS attacks) in their early stages.

Compromised Device Identification. Whereas the majority of related papers identify compromised devices by detecting malware conveyed by Botnets, some work focuses on identifying compromised devices directly using a variety of techniques including: optimization [33], analyzing communication features [34], using language analysis [35], tracking device location [36], or monitoring the downlink channels of a gateway [37]. Moreover, Reference [38] proposed an ML-based system that analyzes traffic flows and packet features in network layer to identify intrusions in an IoT system. In [39], a Botnet detection system, called BotStop, was developed based on extreme gradient boosting model that analyzes packet traffic. In [40], a Compromised Device Identification System (CDIS-DRNN) is developed based on the DRNN model [41] that analyzes the network nodes' incoming and outgoing traffic. The performance of different attack detection techniques can depend on which datasets are used for learning and testing, and prior to the current paper, the CDIS-DRNN offered the best available state-of-the-art performance for compromised device identification when the publicly available Kitsune Botnet dataset [42], [43] is used. However, none of these works considered the interrelationships between IoT nodes and the propagation of a Botnet attack through these nodes.

In recent work a method was proposed to evaluate a set of network or IoT nodes simultaneously in a single recurrent RNN architecture composed of two interconnected and associated neural networks [44], trained and tested with ground truth data. Therefore, in the sequel we will also compare the ARNN technique developed in this paper using the Kitsune dataset, against the performance offered by CDIS-DRNN.

B. CONTRIBUTIONS OF THIS PAPER

This paper develops an *Associated Random Neural Network (ARNN)* decision system, designed to assess the overall security of an IoT network by identifying compromised devices using aggregated multi-node traffic information. The ARNN utilizes two associated RNN neurons for each IoT device (or IP address) in the network that is being assessed for security. These neurons assess the security level of specific devices, and advocate that the device is compromised or not based on the information provided by traffic metrics measured at the device, and inter-neuron weights with other neurons that assess neighbouring network nodes. ARNN based attack detection was previously introduced in [44], and initially evaluated for a system that learns from ground truth data and is then tested for the same ground truth data.

Here, it is hypothesized that the ARNN successfully identifies compromised IoT nodes based on its ability to learn both the interrelationships between those nodes and the propagation of a cyberattack. Therefore, after detailing the ARNN learning algorithm's components: the error metric, the weight restrictions, the ARNN initialization, and its learning algorithm are discussed in the Appendix, we thoroughly evaluate its performance for Mirai Botnet attacks on an IoT network with 107 nodes using the Kitsune dataset [42].

First, we train the ARNN on ground truth data from the initial part of the Kitsune dataset, and test its prediction capabilities with ground truth data from the disjoint latter part of the dataset.

Next, we discuss an average normalized metric based on six relevant metrics [40] extracted from traffic data. We then test the ARNN trained with ground truth data, using the average input metric over the testing period which is disjoint from the training period and is subsequent to it. In all cases we evaluate the Accuracy, True Positive Rate and True Negative Rate of the ARNN and observe very accurate attack detection for most of the 107 nodes that are contained in the Kitsune dataset.

Finally, we compare the performance of ARNN against the state-of-the-art best-in-class CDIS-DRNN and four well-known Machine Learning (ML) models for the same problem [40]. In this case, we also train the ARNN without the ground truth but using the ARNN incrementally on successive short training cycles, followed by testing, and pursued for all of the available Kistune data set. The experimental results again indicate that the ARNN offers superior performance – achieving 100% median accuracy and above 92% accuracy for more than 75% of the network nodes in the dataset – with about 3.5 *ms* of detection time.

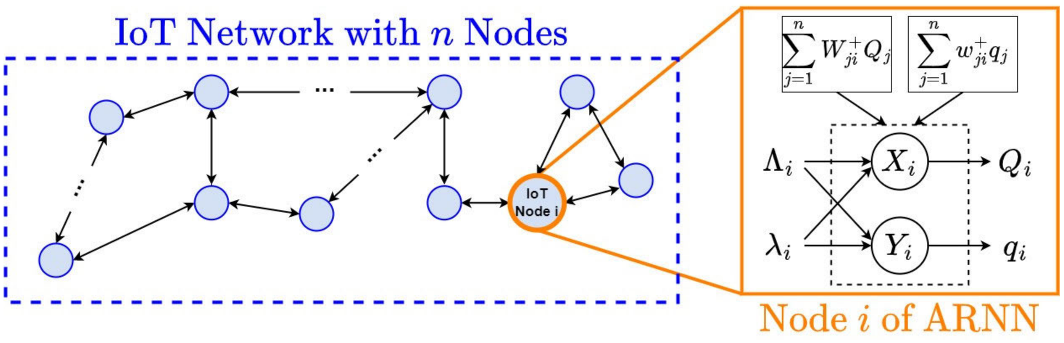

IoT Network with n Nodes
IoT Node i
$\sum_{j=1}^{n} W_{ji}^{+} Q_j$
$\sum_{j=1}^{n} w_{ji}^{+} q_j$
Λ_i
λ_i
X_i
Y_i
Q_i
q_i
Node i of ARNN

FIGURE 1. ARNN based Network-Wide Cyberattack Assessment for an n node network.

The proposed ARNN decision system has the following characteristics:

1) It is an architecture based on the RNN [31] as shown in Figure 1, which associates a pair X_i, Y_i of neurons to assess the security level of each of the i-the node in an n-node IoT network, to determine which IoT devices are compromised. Note that while we consider IoT networks as the object of our research, this approach can also be used for other collections of interconnected IP addresses. While X_i's role is to defend the thesis that i is compromised, Y_i defends the opposite thesis. Thus the ARNN has a total of only $2n$ neurons for evaluating an n-node network.
2) Using ground truth data, the ARNN is trained with a specific gradient descent learning algorithm.
3) During usage or testing, the ARNN receives as input the average value of the traffic characteristics that are used to test the CDIS-DRNN and other ML models. This results in substantial computational savings since a scalar input replaces a vector of six elements.
4) Due to its associated and interconnected architecture with a simplified ignorant weight initialization, the ARNN provides accurate assessment on the security of all devices or IP addresses in a network.

II. CONSTRUCTING THE ARNN FOR NETWORK-WIDE CYBERSECURITY ASSESSMENT

We now detail the Network-Wide Cybersecurity Assessment method based on a novel ARNN decision system. This method provides an assessment of the overall security of an IoT network, taking into account the interconnections of devices and the local information provided by these devices. In this method, the ARNN decision system learns direct and indirect relationships between devices in a single network, and estimates the spread of an attack among devices in the IoT network.

The ARNN is composed of n pairs of neurons which are all interconnected in a recurrent structure, where each pair corresponds to an IoT device (or node or IP address) in the network as shown in Figure 1. X_i and Y_i act as "adversaries" indicating whether the node i is compromised or not. Accordingly, the internal state of X_i, denoted by $K_i(t) \geq 0$ indicates that node i is compromised, and that of Y_i, denoted by $k_i(t) \geq 0$, denotes the opposite. As one of the main properties of an RNN neuron, if $K_i(t)$ at any time t is strictly positive, then X_i sends excitatory and/or inhibitory spikes to the neurons of node $j \neq i$ respectively at rates W_{ij}^{+}, $W_{ij}^{-} \geq 0$. Similarly, if $k_i(t)$ is strictly positive, Y_i sends excitatory and/or inhibitory spikes to $j \neq i$ respectively at rates w_{ij}^{+}, $w_{ij}^{-} \geq 0$. We define the probability that these $2n$ neurons are firing as

$$For\ X_i: \ Q_i = \lim_{t\to\infty} Prob[K_i(t) > 0], \tag{1}$$

$$For\ Y_i: \ q_i = \lim_{t\to\infty} Prob[k_i(t) > 0]. \tag{2}$$

In this decision system, when any neuron of node i (X_i or Y_i) fires, the internal state of this neuron drops by 1 as $K_i(t^+) = K_i(t) - 1$ or $k_i(t^+) = k_i(t) - 1$. When any neuron of node i receives an excitatory spike, its internal state increases by 1, i.e. $K_i(t^+) = K_i(t) + 1$ or $k_i(t^+) = k_i(t) + 1$. Similarly, when it receives an inhibitory spike, its internal state drops by 1 if the current state is not zero, i.e. $K_i(t^+) = max[0, K_i(t) - 1]$ or $k_i(t^+) = max[0, k_i(t) - 1]$.

The ARNN equations are a special case of the RNN equations [31], so that:

$$Q_i = \frac{\Lambda_i + \sum_{j=1}^{n} W_{ji}^{+} Q_j}{\lambda_i + \sum_{j=1}^{n}[W_{ij}^{+} + W_{ij}^{-}] + \sum_{j=1}^{n} w_{ji}^{-} q_j},$$
$$q_i = \frac{\lambda_i + \sum_{j=1}^{n} w_{ji}^{+} q_j}{\Lambda_i + \sum_{j=1}^{n}[w_{ij}^{+} + w_{ij}^{-}] + \sum_{j=1}^{n} W_{ji}^{-} Q_j}, \tag{3}$$

where Λ_i is the rate of external excitatory spikes arriving to X_i, while it is the rate of external inhibitory spikes arriving to Y_i. On the other hand, λ_i has exactly the opposite effect. We will choose these two quantities to lie between zero and one: $\Lambda_i \in [0, 1]$, $\lambda_i \in [0, 1]$.

A. RESTRICTING THE WEIGHTS AND INITIALIZING THE ARNN

The ARNN weights are restricted to reduce the number of gradient descent computations, namely:

- Throughout the network we set the "self-weights" to zero: $W_{ii}^{+} = W_{ii}^{-} = w_{ii}^{+} = w_{ii}^{-} = 0$.
- For all $i \neq j$ we fix:

$$W = W_{ij}^{+} + W_{ij}^{-} = w_{ij}^{+} + w_{ij}^{-}, \tag{4}$$

for a given value of $W > 0$ that is detailed below, so that the gradient descent computation only computes W^+_{ij}, w^+_{ij} $\forall i, j$. Note that we are dealing with a fully recurrent network so that all distinct nodes are interconnected, since each neuron is connected to all other neurons when $i \neq j$, while for the paired "opposing neurons" which are not directly connected in one step, they are connected indirectly to each other via other neurons. The ARNN equations then become:

$$Q_i = \frac{\Lambda_i + \sum_{j=1}^n W^+_{ji} Q_j}{\lambda_i + (n-1)W \sum_{j=1}^n q_j (W - w^+_{ji})},$$
$$q_i = \frac{\lambda_i + \sum_{j=1}^n w^+_{ji} q_j}{\Lambda_i + (n-1)W + \sum_{j=1}^n Q_j (W - W^+_{ji})}. \quad (5)$$

- During learning, a total of $2n(n-1)$ weights are computed for an ARNN that is assessing the security of an n-node IoT network. The inhibitory weights are obtained directly from the value of W minus the excitatory weight, since W remains constant. Because of the specific mathematics of the RNN learning algorithm [32] *only one inversion of a* $2n \times 2n$ *matrix is needed at each gradient descent step* to update *all* of the weights for the fully connected ARNN.

The ARNN is first initialized so that it does not know initially whether any of the devices (or IP addresses) are compromised. To this effect:

- To represent perfect ignorance for all neurons we select the network input rates and weights that will result in $Q_i = q_i = 0.5$ for all the neurons, with $\Lambda_i = \lambda_i = \Lambda$, where Λ is chosen below.
- Similarly for $i \neq j$ the weights are set to $W^+_{ij} = W^-_{ij} = w^+_{ij} = w^-_{ij} = 0.5W$, where $W > 0$.

As a result we write:

$$0.5 = q_i = Q_i = \frac{\Lambda + 0.5 Q_i W(n-1)}{W(n-1) + \Lambda + 0.5 q_i W(n-1)},$$
$$\textit{yielding } \Lambda = 0.75W(n-1). \quad (6)$$

Thus if we take $W = 1$, we have $\Lambda = 0.75(n-1)$, and obtain $Q_i = q_i = 0.5$, $1 \leq i \leq n$, so that the ARNN is "ignorant" before the learning algorithm is used.

B. THE ARNN EXTERNAL INPUTS Λ_i AND λ_i

The external inputs are obtained from data from packet statistics in the network, or from ground truth that is used for training the ARNN, regarding whether given packets are attack packets or normal packets, or other data used for training, or real operational data for testing.

We therefore consider that $Q_i \in [0,1]$ and $q_i \in [0,1]$ are functions $Q_i(\Lambda_i, \lambda_i)$ and $q_i(\lambda_i, \Lambda_i)$. Noting that:

$$\frac{Q_i(1-q_i)}{q_i(1-Q_i)} = \lim_{t\to\infty} Prob[K_i(t) > k_i(t)], \quad (7)$$

we define the outputs of the ARNN for each network node i, as being the binary Z_i variables:

$$Z_i = 1 \textit{ if } \frac{Q_i(1-q_i)}{q_i(1-Q_i)} > \gamma > 0, \quad Z_i = 0 \textit{ otherwise}, \quad (8)$$

where $Z_i = 1$ stands for node i being compromised, while $Z_i = 0$ has the opposite meaning, and $0 < \gamma < 1$ is a threshold.

C. THE LEARNING DATASET LD

The Learning Dataset is a set of packets LD where, for each packet, we know in advance whether it is an attack or a benign, i.e. "normal", packet. Thus the LD is used to train the ARNN.

The set of packets LD that we use to train the ARNN, as well as the dataset used for testing, contain the ground truth for each packet denoted $pk(t,s,d,a)$, where:

- t is the transmission instant of the packet from the source node s, and d is the packet's destination node,
- a is a binary label so that $a = 1$ indicates that it is an "attack" packet and $a = 0$ that it is a "benign" packet,
- The length of the packet in bytes, including the header, is denoted by $|pk(t,s,d,a)|$.
- Packets are grouped into "slots" lasting $\tau = 10$ seconds, so that the packet's slot number is $l = \lfloor * \rfloor \frac{t}{\tau}$, i.e. when $(l-1)\tau \leq t < l\tau$, and M is the total number of slots in the dataset: $1 \leq I \leq M$,

In the dataset that we use, we observe that on average roughly 100 packets are contained in a 10 sec time slot.

We now determine the successive ARNN inputs from the dataset LD, namely: $\Lambda^l_{Gi} \in [0,1]$ and $\lambda^l_{Gi} = 1 - \Lambda^l_{Gi}$, the corresponding output K^l_i, and the decision output Z^l_{Gi} which is a binary variable related to K^l_i.

Let $S^l(i)$ and $R^l(i)$ be the set of packets that have been transmitted or received by node i from the first slot until the end of the $l = \lfloor \frac{t}{\tau} \rfloor$-th time slot:

$$S^l(i) = \{pk(t,s,d,a) : \ 0 < t \leq l\tau, \ \forall d, \ a = 0, 1\},$$
$$R^l(i) = \{pk(t,s,d,a) : \ 0 < t \leq l\tau, \ \forall s, \ a = 0, 1\},$$

and:

$$\text{If } |R^l(i)| > 0, \ \text{ then } \Lambda^l_{Gi} = \frac{|\{pk(t,s,i,1) : \ \forall s\}|}{|R^l(i)|},$$
$$\text{else } \Lambda^l_{Gi} = 0. \quad (9)$$

Furthermore

$$\lambda^l_{Gi} = 1 - \Lambda^l_{Gi}. \quad (10)$$

When a node receives a significant number of attack packets, one expects that it may be compromised, and in turn send out attack packets. Therefore the l-th desired output for node i as K^l_i is the ratio of attack packets **sent by node** i to all other nodes until the end of the $l = \lfloor \frac{t}{\tau} \rfloor$-th time window:

$$\text{If } |S^l(i)| > 0 \text{ , then } K^l_i = \frac{|\{pk(t,i,d,1), \ \forall d\}|}{|S^l(i)|},$$
$$\text{else if } |S^l(i)| = 0 \text{ , then } K^l_i = 0. \quad (11)$$

We also define the i-th binary decision variable as D_i^l for some threshold $1 > \theta > 0$ regarding the ground truth:

$$D_i^l = \begin{cases} 1 & \text{if } K_i^l > \theta, \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

so that $D_i^l = 1$ indicates that i is a compromised node in the l-th slot, while D_i^l indicates the opposite.

On the other hand, since the ARNN is trained directly with the values of K_i^l as output, we use the metric defined in (8) to evaluate the output decision from the ARNN, namely:

$$Z_i^l = \begin{cases} 1 & \text{if } \dfrac{Q_i(\Lambda_i^l, \lambda_i^l)[1 - q_i(\lambda_i^l, \Lambda_i^l)]}{q_i(\lambda_i^l, \Lambda_i^l)[1 - Q_i(\Lambda_i^l, \lambda_i^l)]} > \gamma > 0, \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where(Λ^l, λ^l) are the corresponding n-vectors obtained from ARNN input data at the l-th slot.

D. LEARNING THE ARNN WEIGHTS FROM THE LD

To construct a balanced training dataset LD, the sequence of slots in the MIRAI dataset [42] was scanned from the first slot $l = 1$ up to and including the first slot where some node sends attack packets, which turns out to be slot $l = 445$, and the LD then included slot 433 up to and including slot 457 (a total of 25 slots). On the other hand, the test dataset TD contains all the subsequent slots, starting with slot 458.

The ARNN is trained with the LD that uses the slots l of the dataset which are being used for training using the Gradient Descent Algorithm detailed in the Appendix, with the learning rate $\eta = 0.1$. It adjusts the ARNN weights so as to minimize the following **error function** (14) for each successive bucket l within the LD:

$$\mathbf{E}^l = \frac{1}{2}\sum_{i=1}^{n}\Big[\big(Q_i(\Lambda_i^l, 1 - \Lambda_i^l) - K_i^l\big)^2 + \big(q_i^l(1 - \Lambda_i^l, \Lambda_i^l) - (1 - K_i^l)\big)^2\Big], \tag{14}$$

where $Q_i^l(.)$ and $q_i^l(.)$ are obtained from (5).

E. TESTING THE ARNN'S PREDICTION CAPABILITY

We first test the ARNN's ability to act as a **predictor** about whether a node is compromised, based on **training with the** LD composed of the sequence of 25 slots around the first slot that contained some compromised nodes, namely slot 445. The test data stream that is **subsequent** to the LD that is used, namely slot $l = 445 + 13$ up to the last slot $l = 713$.

Testing therefore uses the input values Λ_i^l, λ_i^l for $458 \leq l \leq 713$ in the trained ARNN, and the ARNN then outputs the corresponding Z_i^l values, with $\theta = 0.3$ as the threshold in obtaining the ground truth decision variables D_i^l from expression (12). The threshold to produce the testing output Z_i^l is typically of the form $\gamma = 1 - \epsilon$ where ϵ is often zero and always well under 0.1.

The Accuracy, True Positive Rate (TPR) and True Negative Rate (TPR) of ARNN are detailed in Figures 2, 3, and 4.

On the other hand, Figure 5 shows a box-plot for the statistics related to all the node addresses and indicates that the ARNN offers high performance with a median accuracy of 100%. In addition, although the TPR is almost zero for 9 of the addresses, while TNR is almost 0 for 22 addresses; hence the Accuracy exceeds 95% for 80% of all addresses.

Figure 2 displays the average decision accuracy for each address $i \in \{1, \ldots, 107\}$, showing that the accuracy of ARNN is above 95% for 50% of the IP Addresses while it is between 62% and 80% for only 20% of them without ever being under 62%, while Figure 4 exhibits the average TNR for the addresses. For 59% of the addresses, the TNR lies above 95% while for 15% of them it is in the 62% to 80% range. Finally in Figure 3 the average TPR is shown for the 39 addresses which were at least once compromised according to the ground truth indicator. The TPR exceeds 95% for most (64%) addresses, and exceeds 90% in over 74% of the them.

III. TESTING THE ARNN WITH THE AVERAGE TRAFFIC METRIC

Six representative traffic metrics were introduced in recent work [40] as being indicative of network attacks and were shown to be effective for MIRAI Botnet detection using available real datasets. Rather than using the full metrics, in this section their *average normalized value* will be used to test the ARNN attack detector.

To define these metrics, let $|p|$ be the size in bytes of some packet p, including its header and all the data it contains. Let $P_l^{S,i}$ be the set of all packets sent by all network nodes to node i in slot l, and let the maximum length in bytes of packets sent by node s to i up to the end of slot l, be $L_s^l = max\{|p| : p \in P^l(s)\}$. The six metrics from [40], all normalized to a value between 0 and 1, are as follows:

- Average packet size of packets received by device i in slot l:

$$x_l^{i,1} = \frac{\sum_{p \in P_l^{S,i}} |p|}{\sum_{s \in S} L_s^l \times |P_l^{S,i}|}. \tag{15}$$

- The maximum size of any packet received at node i in slot l:

$$x_l^{i,2} = \frac{\max_{p \in P_l^{S,i}} |p|}{L_S^l}. \tag{16}$$

Denial of Service attacks are not always carried out with large packets; for instance, SYN attack packets can be quite short since their effect is to overload the receiving node with requests to open a connection, rather than with the amount of traffic that is being sent. However, the amount of traffic sent by other types of Denial of Service attacks are often meant to cause link and node congestion, so that the amount of attack traffic can be large, and the length of packets that are sent by attackers can be large too. Thus, the amount of traffic and packet size are often relevant metrics for detecting attacks.

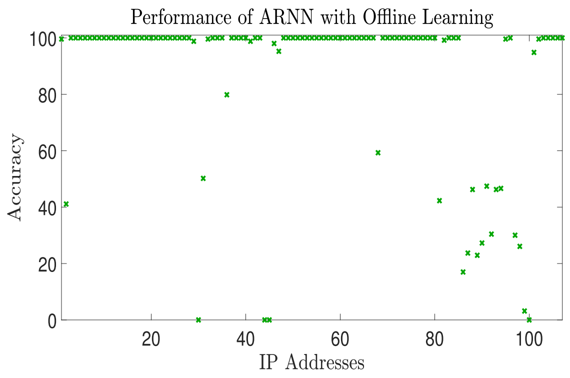


FIGURE 2. The average accuracy of the ARNN predictions for all addresses $1 \leq i \leq 107$ is shown for the test data *TD*, by comparing the ground truth binary value Y_i^l with the ARNN's binary output Z_i^l.

- The average number of packets received at i in slot l from all nodes:

$$x_l^{i,3} = \frac{|P_l^{S,i}|}{\sum_{s \in S} 1[|P_l^{s,i}| > 0]} . \qquad (17)$$

Note that the denominator in the above expression can be computed iteratively in an efficient manner, so that $x_l^{i,3}$ can be obtained directly from $x_{l-1}^{i,3}$.

- The (normalized) maximum number of packets received by node i from any single source in the slot l:

$$x_l^{i,4} = \frac{\max_{s \in S} |P_l^{s,i}|}{\max_{s \in S,\ 1 \leq u \leq l} |P_u^{s,i}|} . \qquad (18)$$

where $P_l^{s,i}$ denotes the set of packets sent from node s to node i during slot l.

- Finally, the last two metrics, both normalized to lie between 0 and 1, describe the total number of bytes sent to all destinations d by node i, and the total number of packets sent by i to all d:

$$x_l^{i,5} = \frac{1}{B_i} \sum_d \sum_{p \in P_l^{i,d}} |p|, \quad x_l^{i,6} = \frac{L_i^m}{B_i} \sum_d |P_l^{i,d}|, \qquad (19)$$

where, L_i^m is the maximum length of any packet that i sends, and B_i is the maximum number of bytes sent out by i in any slot:

$$B_i = max\{\sum_d \sum_{p \in P_l^{i,d}} |p|.|P_l^{i,d}| : 1 \leq l \leq M\} .$$

Since each neuron at any node of ARNN has a single input, i.e. Λ_i or λ_i, for testing purposes we only use the **average value of the normalized metrics** as the input to each neuron of ARNN for slot l:

$$\Lambda_{i,mean}^l = \frac{\sum_{k=1}^6 x_l^{i,k}}{6}, \quad \lambda_{i,mean}^l = 1 - \Lambda_{i,mean}^l. \qquad (20)$$

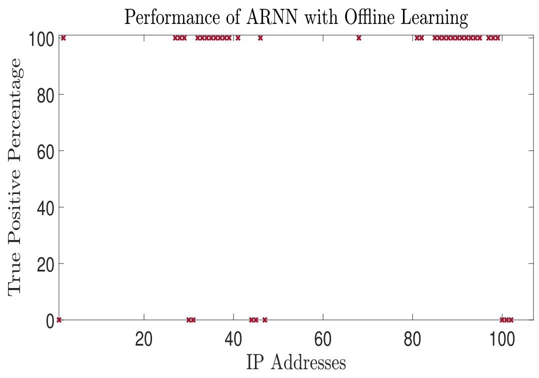


FIGURE 3. The average percentage TPR is shown over all the slots for each of the 107 addresses in the network, obtained by comparing Y_i^l and Z_i^l for those values of l where $Y_i^l = 1$. Note that if $Y_i^l = 0$ for an address i in all the slots l in the *TD*, then the TPR cannot be measured for i. Thus only 39 out of 107 addresses are concerned by the TPR, as shown in the figure.

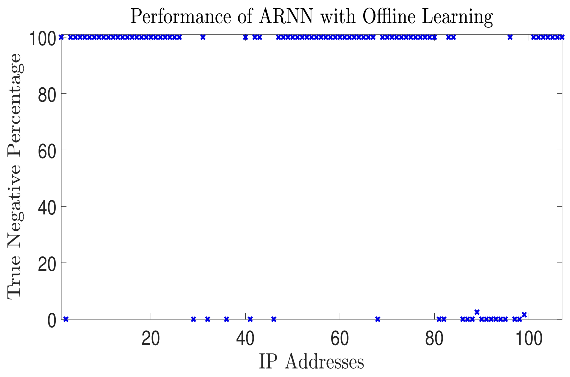


FIGURE 4. The average percentage TNR is shown over all the slots for each of the 107 addresses in the network, obtained by comparing Y_i^l and Z_i^l for those values of l where $Y_i^l = 1$. Note that if $Y_i^l = 1$ for an address i in all the slots l in the *TD*, then the TNR cannot be measured for i.

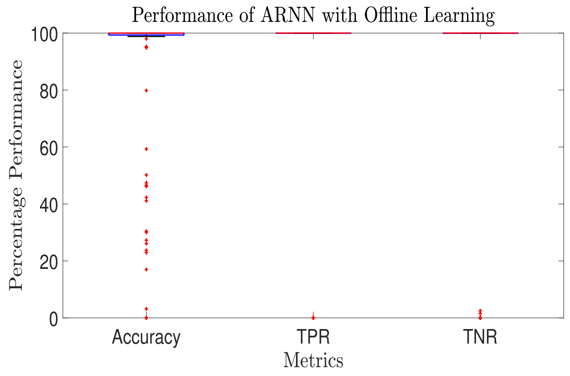


FIGURE 5. The Accuracy, TPR and TNR performance of the ARNN algorithm is illustrated by box-plots for the statistics obtained from the results in Figures 2, 3, 4.

A. THE ARNN TRAINED WITH THE GT AND TESTED WITH AVERAGE METRICS

In the first test using the average metric based input data, we use the ARNN trained with the ground truth *GT* from real attack *LD* sequence of 25 slots starting at $l = 432$ up to 457, as before. Then for each i we use the average metric value to compute $\Lambda_{i,mean}^l$ for $l = 458$ to $l = 713$. We input the corresponding values $\Lambda_{i,mean}^l$, $\lambda_{i,mean}^l = 1 - \Lambda_{i,mean}^l$ into the ARNN for testing.

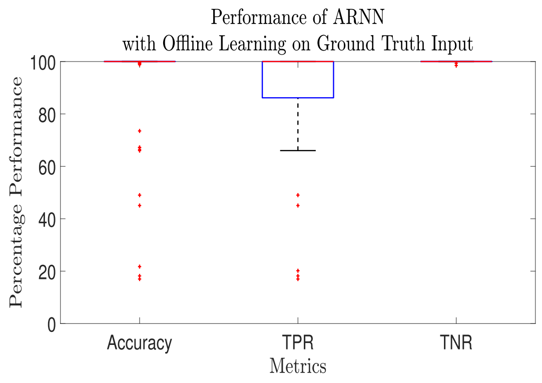


FIGURE 6. The Accuracy, TNR and TPR of the ARNN algorithm are shown as box-plots for the ARNN trained with the GT and tested with the average metric values for $458 \le l \le 713$.

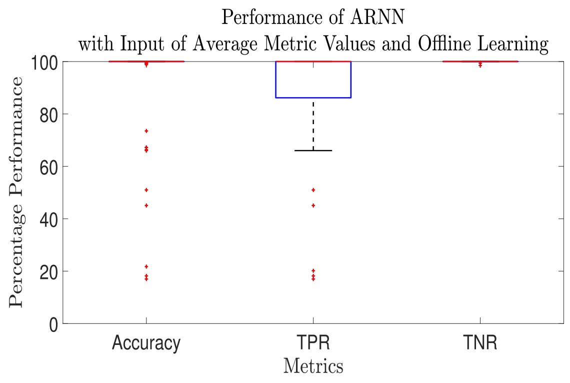


FIGURE 7. The Accuracy, TPR, TNR of the ARNN algorithm are shown as box-plots for the ARNN trained and tested with the average metric values for $458 \le l \le 713$.

The ARNN output is the Z_i^l value for each successive l and for each node i, as given in (13) with a threshold which can differ in the range $0.96 \le \gamma \le 1$ or $0 \le \epsilon \le 0.04$. The threshold $\theta = 0.3$ is used for the output decision variables D_i^l for the known GT. The results are summarized in Figure 6, where we see that the median performance with respect to each of Accuracy, TPR, and TNR is 100%.

ARNN achieves Accuracy above 99% for 97 of 107 IP Addresses, while there are 10 nodes with outlier performance, three with Accuracy below 30%, two between $60\% - 30\%$, and five node addresses with Accuracy between $80\% - 60\%$. In addition, as the lower whisker shows, TPR is above 86% for 75% of all nodes, and while lowest TNR performance is about 98.5%.

B. ARNN TRAINED AND TESTED WITH THE AVERAGE METRICS

We now consider training as well as testing the ARNN using the average metric inputs $\Lambda_{i,mean}^l$ and $\lambda_{i,mean}^l = 1 - \Lambda_{i,mean}^l$. To this effect, we still use the ground truth data represented by K_i^l, in the algorithm detailed in the Appendix. The error function that needs to be minimized during training becomes:

$$\mathbf{E}^l = \frac{1}{2}\sum_{i=1}^{n}\Big[\big(Q_i(\Lambda_{i,mean}^l, 1-\Lambda_{i,mean}^l) - K_i^l\big)^2 + \big(q_i^l(1-\Lambda_{i,mean}^l, \Lambda_{i,mean}^l) - (1-K_i^l)\big)^2\Big], \qquad (21)$$

with $Q_i^l(.)$ and $q_i^l(.)$ are given by equation (5), and the gradient descent parameters is as previously $\eta = 0.1$.

With regard to the previous case where the ARNN was trained with the GT, we see some very very minor variations in Accuracy, True Positive Rate and True Negative Rate. For instance, in our experiments we only observed 3 network nodes out of 107 where Accuracy differed between the previous sub-section and this one. In particular we observed that:

- For $i = 37$, Accuracy using Average Metric based learning is $ACC = 92.68\%$ while using the GT it is 92.54%,
- For $i = 46$ we have $ACC = 95.49\%$ while with GT training it is $ACC = 95.63\%$, and
- For $i = 47$ we have $ACC = 93.1\%$ using the Average Metric for training, while using the GT it is $ACC = 93.94\%$.

In fact, we also observe that using the Average Metric for training results in general in somewhat fewer False Alarms, i.e. a higher True Negative Rate. The corresponding results are summarized in the Box Plot Diagram for Accuracy, True Positive and True Negatives given in Figure 7.

IV. INCREMENTAL TRAINING OF THE ARNN

In recent work [40], the CDIS-DRNN, a compromised device identification method was presented. This attack detection method is trained sequentially on the assumption that off-line ground truth data is not available. CDIS-DRNN is composed of a deep learning feedforward RNN architecture which does not exploit knowledge of the interconnections between network nodes.

Thus, as with CDIS-DRNN, in this section we will assume that offline training data is **not available** in advance of the exploitation of the ARNN for attack detection. In such a case, the ARNN will be trained incrementally in parallel to its online operation. To this end, we update the weights of the ARNN every successive 6 slots, i.e. at the end of slot l such that $mod(l, 6) = 0$, where each training window corresponds to 1 minute, whereas the ARNN provides a decision for each device i at the end of each individual slot l, i.e. every 10 seconds.

Thus using the data for a successive set of 6 slots, the ARNN is trained with the algorithm presented in the Appendix, using the training data TD constructed a follows:

$$TD = \{(A_i^{l'}, K_i^{l'}),\ l' = l-5, \ldots, l\}.$$

We now present the performance of the incrementally trained ARNN decision system for compromised device identification. During the performance evaluation, we set $\Theta = 0.3$ and $0.96 \le \gamma \le 1$. The Accuracy, True Negative Rate (TNR), and True Positive Rate (TPR) of the ARNN with incremental training are presented in Figure 9 as a Box-plot. The results in this figure show that the ARNN achieves a median accuracy of 100% while Accuracy is shown to be greater than 97% for 75% of all network nodes. These results

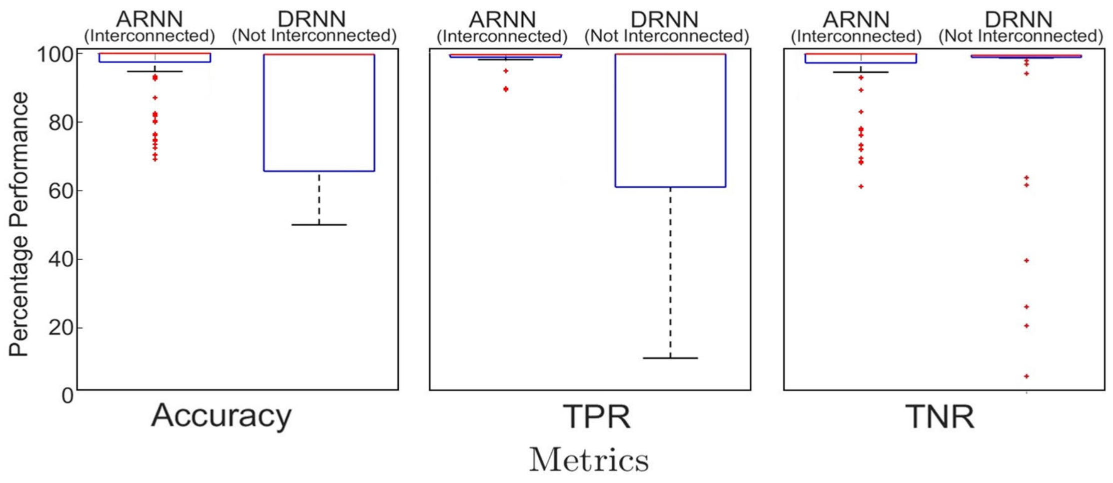


FIGURE 8. Performance comparison of the ARNN with the CDIS-DRNN approach [40] with respect to (left) Accuracy, (middle) TNR, and (right) TPR.

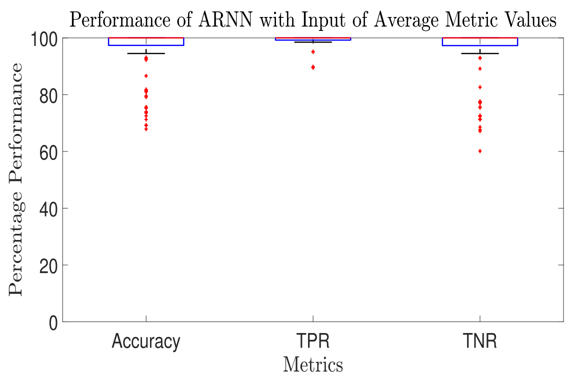


FIGURE 9. The accuracy, TPR, TNR of the ARNN algorithm are shown as box-plots for the ARNN trained and tested with the average metric values.

also show that the TNR is above 99% for approximately 72% of nodes, while 58% of the nodes are 100% accurately identified as being compromised, which provides a median TPR value of 100%.

In Figure 8, we compare the performance of the ARNN with similar results obtained recently with the CDIS, which is a DRNN-based system [40], with respect to Accuracy, TNR and TPR. The results show that the ARNN significantly outperforms the CDIS-DRNN by providing approximately 50% higher Accuracy for all network nodes. This appears to be due to the fact that the ARNN, through its internal neuron connections, is able to simultaneously process information regarding the nodes themselves, and also regarding their connections to other nodes.

We further compare the performance of ARNN against four well-known ML models 1-Dimensional Convolutional Neural Network (1D CNN), Long-Short Term Memory (LSTM), Multi-Layer Perceptron (MLP), and Decision Tree (DT), which are often used for intrusion detection systems in recent research [10], [11], [14], [19], [45]. The MLP is comprised of three fully connected layers each with n neurons. 1D CNN and LSTM respectively consist of convolution and LSTM layers connected to two fully connected layers, where each layer is comprised of n neurons. In addition,

TABLE 1. Comparison of the ARNN with well-known ML models with respect to execution and training times.

	Execution Time (ms)	Training Time (secs)
ARNN	3.446	198.205
1D CNN	18.473	0.027
LSTM	19.851	0.039
MLP	18.252	0.027
DT	0.056	0.195×10^{-3}

all activation functions in 1D CNN, MLP, and LSTM are sigmoids, and we implemented these models using Keras API in Python. For the implementation of DT, we used the Scikit-learn library in Python setting the maximum depth and maximum number of features equal to n.

Figure 10 displays the comparison of ARNN with 1D CNN, LSTM, MLP, and DT with respect to Accuracy, TPR, TNR, F1 Score, Recall, and Precision. These results show the superior overall performance of ARNN against these well-known ML models. It can be seen that the performance of the ARNN is slightly lower than DT in terms of average Accuracy, TNR and Precision but significantly higher than the other models in terms of TPR, F1 Score and Recall. Thus although DT is slightly better than ARNN at detecting negative samples (i.e. DT gives fewer false positives compared to ARNN), ARNN significantly outperforms all models (at least by 19%) in detecting compromised nodes.

Finally, Table 1 compares the same set of ML models with respect to execution and training times, showing that the ARNN is the second fastest in online operation, among the five methods that were tested, with an execution time of $\approx 3.5\ ms$ to rapidly identify nodes compromised by malicious Bots in real time, but it is the slowest one regarding learning. For an n-node IoT or IP network's attack prediction, the Deep Learning Algorithm for the ARNN based on a fully connected "recurrent" RNN with $2n$ neurons [32], requires at each

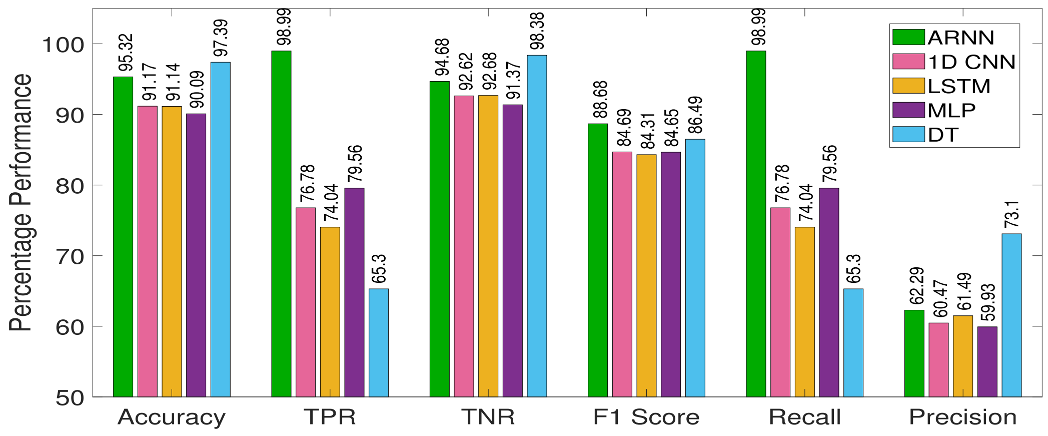


FIGURE 10. **Performance comparison of the ARNN with well-known ML models with respect to Accuracy, TNR, TPR, F1 Score, Recall, and Precision.**

learning step (1) the inversion of a $2n \times 2n$ matrix of time complexity $A(2n)^3 = 8An^3$, (2) the solution of the equation for the state vectors Q, q of time complexity $B(2n)(2n)^2 = 8Bn^3$, and the updates $2n(2n-1)$ individual weights, for a total computation time per learning step:

$$TARNN(n) \approx 8(A+B)n^3 + 4Cn^2, \tag{22}$$

for positive constants A, B, C. On the other hand, the CNN or the MLP are feedforward models, typically with three feedforward layers, whose learning algorithm is of time complexity $O(n^2)$. Thus they will require the update of some most $2n^2$ weights, yielding a Learning computational time:

$$TFF(n) \approx 2bn^2, \tag{23}$$

for a positive constant b which is comparable to C. This simplistic calculation suggests an approximate $\frac{8(A+B)n^3}{2bn^2} = 4n\frac{A+B}{b}$ fold increase in learning times for the ARNN with respect to an MLP or CNN model. For the $n = 107$ network that is evaluated in this paper, this corresponds to a $428 \times \frac{A+B}{b}$ fold increase, and for $A+B \approx 20b$ this analysis is compatible with the results shown in Table 1.

V. DISCUSSION REGARDING THE RESULTS

During our experimental evaluation:

- The ARNN is first trained offline with ground truth data and tested with disjoint ground truth data exhibiting a high level of precision.
- Then, the offline trained ARNN is tested with a simplified average input metric directly extracted from measurements, and again a high level of precision is observed.
- The average metric is then used as input for offline training, while the ground truth is used in the error function, and testing is carried out with the average metric using disjoint data.
- Finally, online incremental training using the testing data output for learning is also tried, without use of the ground truth.

All these experiments use the Kitsune dataset, and confirm the high level of accuracy of the ARNN's predictions.

Experiments are also conducted to compare the ARNN for the identification of compromised IoT nodes using real MIRAI Botnet attack data from the Kitsune dataset, with the recent state-of-the-art CDIS-DRNN technique [40] and the well-known 1D CNN, LSTM, MLP and DT models are also carried out, indicating that the ARNN:

- Provides significant improvement compared to CDIS-DRNN, achieving 92% median accuracy and minimum 60% accuracy per node,
- Outperforms the "best-of-class" CDIS-DRNN by a wide margin with respect to both TNR and TPR, and
- Identifies compromised nodes in just under 3.5 *ms* at least 19% more accurately than 1D CNN, LSTM, MLP and DT models.

Accordingly, as its main advantage, the ARNN successfully captures the interrelationships and communication patterns between devices, thus providing considerably high identification performance that is superior to state-of-the-art techniques. On the other hand, although the ARNN provides a detection in under 3.5 *ms*, it requires significantly longer training time compared to well-known ML models. That is, while the current design and implementation of the ARNN learning algorithm may not be suitable for time-limited online learning applications, it is highly successful and promising in identifying compromised nodes with offline learning and online detection.

VI. CONCLUSION

This paper presents and evaluates the novel ARNN cyberattack decision system that utilizes two interconnected competing RNN neurons for each IoT node, where each network node is related to a neuron pair connected with all the neurons associated with other nodes in the network. The unique structure of the ARNN evaluates the *security* *of each node in a given network* by including locally relevant data from incoming traffic as well as the relationship between all nodes, as part of the decision mechanism. In this way, ARNN learns, as one of its most important features, both the normal communication patterns between nodes and the propagation of a cyberattack over the IoT network.

The ARNN can be particularly useful for private or industrial networks containing a few hundred nodes. It can be used to detect attacks such as Botnets, where distinct node behaviours are correlated due to the propagation of the attack. Its use with average traffic metrics measured directly on incoming traffic at each network node, removes the need for separate computationally costly attack detectors that are placed at each node in the network. Thus, in sharp contrast with traditional attack detectors, the ARNN collectively evaluates a large number of interconnected nodes in a single neural network architecture at a low additional computation cost per node.

We have presented the ARNN architecture, with weight restrictions to simplify learning, and "ignorant initialization" that helps to avoid initial biases of the ARNN. The error function used for ARNN learning is introduced, and the specific gradient learning algorithm is detailed in the Appendix. Most of the paper is then devoted to evaluating the performance of the ARNN using real Botnet attack data, and real benign traffic and comparing that against the state of the art methods based on the commonly used metrics of Accuracy, True Positive Rate, True Negative Rate, F1 Score, Recall, and Precision. The results revealed that ARNN achieved significantly superior performance with highly accurate detection of compromised nodes and low false alarms, but with high training time.

In future work, the computational complexity of ARNN learning which was discussed in this paper from a theoretical perspective, will be analyzed in detail from practical experimental data, and incremental schemes will be considered for on-line learning to reduce the amount of energy that such algorithms consume [46]. Since many IoT devices have limited battery power, this is important for sustainability, and it can also enhance IoT security.

APPENDIX

THE ARNN LEARNING ALGORITHM

The ARNN uses the Random Neural Network (RNN) which is an effective approximator for continuous and bounded functions [47], based on principles in [48], whose gradient descent learning is described in [32]. Other RNN learning algorithms are described in [49] and [50], and the G-Network queueing model [51], [52], [53], [54], [55] is a generalization of the RNN.

The gradient descent algorithm for the ARNN weights seeks local minima of $\mathbf{E}$ in (14), and computes the partial derivatives:

$$Q_i^{U,V} = \frac{\partial Q_i}{\partial W^+_{U,V}}, \quad Q_i^{u,v} = \frac{\partial Q_i}{\partial w^+_{u,v}},$$
$$q_i^{U,V} = \frac{\partial q_i}{\partial W^+_{U,V}}, \quad q_i^{u,v} = \frac{\partial q_i}{\partial w^+_{u,v}},$$

that are needed in the computation (24):

$$\begin{aligned} E^{U,V} &\equiv \frac{\partial \mathbf{E}}{\partial W^+_{U,V}} \\ &= \sum_{i=1}^{n} [\,(Q_i - K_i)Q_i^{U,V} + (q_i - 1 + K_i)q_i^{U,V}\,], \quad (24) \\ E^{u,v} &\equiv \frac{\partial \mathbf{E}}{\partial w^+_{u,v}} \\ &= \sum_{i=1}^{n} [\,(Q_i - K_i)Q_i^{u,v} + (q_i - 1 + K_i)q_i^{u,v}\,]. \quad (25) \end{aligned}$$

Equations (24), (25) are used to update the ARNN weights for steps $k = 1, 2, \ldots$ of the Gradient Descent Rule with $\eta > 0$:

$$\begin{aligned} W^+_{U,V,k+1} &\leftarrow W^+_{U,V,k} - \eta E^{U,V}|_{W^+_{U,V,k}}, \\ w^+_{u,v,k+1} &\leftarrow w^+_{u,v,k} - \eta E^{u,v}|_{w^+_{u,v,k}}. \quad (26) \end{aligned}$$

As indicated earlier, the value $\eta = 0.1$ is used. From the inputs $\Lambda = (\Lambda_1, \ldots \Lambda_n)$, $\lambda = (\lambda_1, \ldots, \lambda_n)$, we derive the derivatives needed in (26) from (5):

$$\begin{aligned} Q_i^{U,V} &= \frac{Q_U}{D_V} 1[i = V] + \sum_{j=1}^{n} \frac{W^+_{ji}}{D_i} Q_j^{U,V} \\ &\quad - \sum_{j=1}^{n} \frac{Q_i[W - w^+_{ji}]}{D_i} q_j^{U,V}, \quad (27) \\ q_i^{U,V} &= \sum_{j=1}^{n} \frac{w^+_{ji}}{d_i} q_j^{U,V} - \sum_{j=1}^{n} \frac{q_i[W - W^+_{ji}]}{d_i} Q_j^{U,V} \\ &\quad + \frac{q_U}{d_V} 1[i = V], \quad (28) \end{aligned}$$

where D_i, d_i are the denominators of Q_i, q_i respectively in the expression (5):

$$\begin{aligned} D_i &= \Lambda_i + \sum_{j=1, j\neq i}^{n} W + \sum_{j=1, j\neq i}^{n} [W - w^+_{ji}]\,.q_j, \\ d_i &= \lambda_i + \sum_{j=1, j\neq i}^{n} W + \sum_{j=1, j\neq i}^{n} [W - W^+_{ji}]\,.Q_j. \quad (29) \end{aligned}$$

We write the *state vectors* $Q = (Q_1, \ldots, Q_n)$ and $q = (q_1, \ldots, q_n)$, the corresponding derivatives $Q^{U,V} = (Q_1^{U,V}, \ldots, Q_n^{U,V})$ and $q^{U,V} = (q_1^{U,V}, \ldots, q_n^{U,V})$, and define the $n \times n$ matrices as:

$$\begin{aligned} B^+ &= \{\frac{W^+_{ij}}{D_j}\}, \quad C = \{\frac{Q_j[W - w^+_{ij}]}{D_j}\}, \\ F^+ &= \{\frac{w^+_{ij}}{d_j}\}, \quad G = \{\frac{q_j[W - W^+_{ij}]}{d_j}\}, \quad (30) \end{aligned}$$

where vector δ_V has zero elements everywhere, except for position V where the value is 1. Then (27) and (28) expressed as vectors yield:

$$\begin{aligned} Q^{U,V} &= B^+ Q^{U,V} - Cq^{U,V} + \delta_V . \frac{Q_U}{D_V}, \\ q^{U,V} &= F^+ q^{U,V} - GQ^{U,V} + \frac{q_U}{d_V}\delta_V, \\ &= [-GQ^{U,V} + \frac{q_U}{d_V}\delta_V][I - F^+]^{-1}, \end{aligned}$$

resulting in:

$$\begin{aligned} Q^{U,V} &= \{-\frac{q_U}{d_V} C\delta_V [I - F^+]^{-1} + \frac{Q_U}{D_V}\delta_V\}. \\ &\quad .\{I - B^+ - CG[I - F^+]^{-1}\}^{-1}. \end{aligned} \tag{31}$$

We can then write the matrices:

$$\begin{aligned} B_*^+ &= \{\frac{w_{ij}^+}{d_j}\},\ C_* = \{\frac{q_j[W - W_{ij}^+]}{d_j}\}, \\ F_*^+ &= \{\frac{W_{ij}^+}{D_j}\},\ G_* = \{\frac{Q_j[W - w_{ij}^+]}{D_j}\}, \end{aligned}$$

and by the symmetry of $Q^{U,V}$ and $q^{u,v}$, and of $Q^{u,v}$ and $q^{U,V}$, we have:

$$\begin{aligned} q^{u,v} &= \{-\frac{Q_u}{D_v} C_* \delta_v [I - F_*^+]^{-1} + \frac{q_u}{d_v}\delta_v\} \\ &\quad \times \{I - B_*^+ - C_* G_* [I - F_*^+]^{-1}\}^{-1}, \\ Q^{u,v} &= \{-G_* q^{u,v} + \frac{Q_u}{D_v}\delta_v\}[I - F_*^+]^{-1}, \end{aligned} \tag{32}$$

which provides us with the derivatives of Q and q.

REFERENCES

[1] E. Gelenbe, P. Campegiani, T. Czachórski, S. K. Katsikas, I. Komnios, L. Romano, and D. Tzovaras, *Security in Computer and Information Sciences*. Cham, Switzerland: Springer, 2018. [Online]. Available: http://library.oapen.org/handle/20.500.12657/23295

[2] E. Gelenbe, M. Jankovic, D. Kehagias, A. Marton, and A. Vilmos, *Security in Computer and Information Sciences*, vol. 1596. Nice, France: Springer, 2021. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-031-09357-9.pdf

[3] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: Classification and state-of-the-art," *Comput. Netw.*, vol. 44, no. 5, pp. 643–666, Apr. 2004.

[4] D. Goodin. (Dec. 2017). *100,000-Strong Botnet Built on Router 0-Day Could Strike at Any Time*. [Online]. Available: https://arstechnica.com/information-technology/2017/12/100000-strong-botnet-built-on-router-0-day-could-strike-at-any-time/

[5] J. Margolis, T. T. Oh, S. Jadhav, Y. H. Kim, and J. N. Kim, "An in-depth analysis of the Mirai botnet," in *Proc. Int. Conf. Softw. Secur. Assurance (ICSSA)*, Jul. 2017, pp. 6–12.

[6] N. Statt. (Oct. 2016). *How an Army of Vulnerable Gadgets Took Down the Web Today*. [Online]. Available: https://www.theverge.com/2016/10/21/13362354/dyn-dns-ddos-attack-cause-outage-status-explained

[7] B. Tushir, H. Sehgal, R. Nair, B. Dezfouli, and Y. Liu, "The impact of DoS attacks on resource-constrained IoT devices: A study on the Mirai attack," 2021, *arXiv:2104.09041*.

[8] H. Sinanovic and S. Mrdovic, "Analysis of Mirai malicious software," in *Proc. 25th Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2017, pp. 1–5.

[9] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, and D. Kumar, "Understanding the Mirai botnet," in *Proc. 26th USENIX Secur. Symp.*, Vancouver, BC, USA, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[10] A. O. Prokofiev, Y. S. Smirnova, and V. A. Surov, "A method to detect Internet of Things botnets," in *Proc. IEEE Conf. Russian Young Researchers Electr. Electron. Eng. (EIConRus)*, Jan. 2018, pp. 105–108.

[11] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer Internet of Things devices," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2018, pp. 29–35.

[12] T. A. Tuan, H. V. Long, R. Kumar, I. Priyadarshini, and N. T. K. Son, "Performance evaluation of botnet DDoS attack detection using machine learning," *Evol. Intell.*, vol. 13, pp. 283–294, Jun. 2019.

[13] I. Letteri, M. Del Rosso, P. Caianiello, and D. Cassioli, "Performance of botnet detection by neural networks in software-defined networks," in *Proc. ITASEC*, 2018, pp. 1–10.

[14] S. Sriram, R. Vinayakumar, M. Alazab, and K. P. Soman, "Network flow based IoT botnet attack detection using deep learning," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 189–194.

[15] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, "Machine learning-based IoT-botnet attack detection with sequential architecture," *Sensors*, vol. 20, no. 16, p. 4372, Aug. 2020.

[16] C. S. Htwe, Y. M. Thant, and M. M. Su Thwin, "Botnets attack detection using machine learning approach for IoT environment," *J. Phys., Conf.*, vol. 1646, no. 1, Sep. 2020, Art. no. 012101.

[17] M. Banerjee and S. Samantaray, "Network traffic analysis based IoT botnet detection using honeynet data applying classification techniques," *Int. J. Comput. Sci. Inf. Secur. (IJCSIS)*, vol. 17, no. 8, pp. 1–6, 2019.

[18] C. D. McDermott, F. Majdani, and A. V. Petrovski, "Botnet detection in the Internet of Things using deep learning approaches," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–8.

[19] J. Liu, S. Liu, and S. Zhang, "Detection of IoT botnet based on deep learning," in *Proc. Chin. Control Conf. (CCC)*, Jul. 2019, pp. 8381–8385.

[20] C. Tzagkarakis, N. Petroulakis, and S. Ioannidis, "Botnet attack detection at the IoT edge based on sparse representation," in *Proc. Global IoT Summit (GIoTS)*, Jun. 2019, pp. 1–6.

[21] A. S. Dina, A. B. Siddique, and D. Manivannan, "A deep learning approach for intrusion detection in Internet of Things using focal loss function," *Internet Things*, vol. 22, Jul. 2023, Art. no. 100699.

[22] R. Panigrahi, S. Borah, M. Pramanik, A. K. Bhoi, P. Barsocchi, S. R. Nayak, and W. Alnumay, "Intrusion detection in cyber-physical environment using hybrid Naïve Bayes-decision table and multi-objective evolutionary feature selection," *Comput. Commun.*, vol. 188, pp. 133–144, Apr. 2022.

[23] J. Ashraf, M. Keshk, N. Moustafa, M. Abdel-Basset, H. Khurshid, A. D. Bakhshi, and R. R. Mostafa, "IoTBoT-IDS: A novel statistical learning-enabled botnet detection framework for protecting networks of smart cities," *Sustain. Cities Soc.*, vol. 72, Sep. 2021, Art. no. 103041.

[24] H. M. Song and H. K. Kim, "Self-supervised anomaly detection for in-vehicle network using noised pseudo normal data," *IEEE Trans. Veh. Technol.*, vol. 70, no. 2, pp. 1098–1108, Feb. 2021.

[25] X. Zhang, J. Mu, X. Zhang, H. Liu, L. Zong, and Y. Li, "Deep anomaly detection with self-supervised learning and adversarial training," *Pattern Recognit.*, vol. 121, Jan. 2022, Art. no. 108234.

[26] E. Caville, W. W. Lo, S. Layeghy, and M. Portmann, "Anomal-E: A self-supervised network intrusion detection system based on graph neural networks," *Knowl.-Based Syst.*, vol. 258, Dec. 2022, Art. no. 110030.

[27] M. Nakip and E. Gelenbe, "MIRAI botnet attack detection with auto-associative dense random neural network," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2021, pp. 01–06.

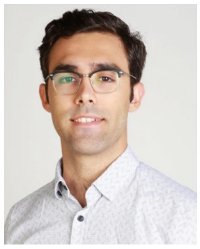
[28] M. Nakip and E. Gelenbe, "Botnet attack detection with incremental online learning," in *Proc. Int. ISCIS Secur. Workshop*. Nice, France: Springer, Oct. 2021, pp. 51–60.

[29] E. Gelenbe and M. Nakip, "G-networks can detect different types of cyberattacks," in *Proc. 30th Int. Symp. Model., Anal., Simul. Comput. Telecommun. Syst. (MASCOTS)*, Oct. 2022, pp. 9–16.

[30] S. Evmorfos, G. Vlachodimitropoulos, N. Bakalos, and E. Gelenbe, "Neural network architectures for the detection of SYN flood attacks in IoT systems," in *Proc. 13th ACM Int. Conf. Pervasive Technol. Rel. Assistive Environments*, Jun. 2020, pp. 1–4.

[31] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," *Neural Comput.*, vol. 1, no. 4, pp. 502–510, Dec. 1989.
[32] E. Gelenbe, "Learning in the recurrent random neural network," *Neural Comput.*, vol. 5, no. 1, pp. 154–164, Jan. 1993.
[33] A. Kumar and T. J. Lim, "Early detection of Mirai-like IoT bots in large-scale networks through sub-sampled packet traffic analysis," in *Proc. Future Inf. Commun. Conf.* Cham, Switzerland: Springer, 2019, pp. 847–867.
[34] M. Chatterjee, A. S. Namin, and P. Datta, "Evidence fusion for malicious bot detection in IoT," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 4545–4548.
[35] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "DÏoT: A federated self-learning anomaly detection system for IoT," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 756–767.
[36] M. Taneja, "An analytics framework to detect compromised IoT devices using mobility behavior," in *Proc. Int. Conf. ICT Converg. (ICTC)*, Oct. 2013, pp. 38–43.
[37] N. V. Abhishek, T. J. Lim, B. Sikdar, and A. Tandon, "An intrusion detection system for detecting compromised gateways in clustered IoT networks," in *Proc. IEEE Int. Workshop Tech. Committee Commun. Quality Rel. (CQR)*, May 2018, pp. 1–6.
[38] M. M. Alani and A. I. Awad, "An intelligent two-layer intrusion detection system for the Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 19, no. 1, pp. 683–692, Jan. 2023.
[39] M. M. Alani, "BotStop: Packet-based efficient and explainable IoT botnet detection using machine learning," *Comput. Commun.*, vol. 193, pp. 53–62, Sep. 2022.
[40] E. Gelenbe and M. Nakip, "Traffic based sequential learning during botnet attacks to identify compromised IoT devices," *IEEE Access*, vol. 10, pp. 126536–126549, 2022.
[41] E. Gelenbe and Y. Yin, "Deep learning with random neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*. Cham, Switzerland: Springer, Jul. 2016, pp. 3–18.
[42] (Aug. 2020). *Kitsune Network Attack Dataset*. [Online]. Available: https://www.kaggle.com/ymirsky/network-attack-dataset-kitsune
[43] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.
[44] E. Gelenbe and M. Nakip, "Associated random neural networks for collective classification of nodes in botnet attacks," 2023, *arXiv:2303.13627*.
[45] G. D. L. T. Parra, P. Rad, K.-K.-R. Choo, and N. Beebe, "Detecting Internet of Things attacks using distributed deep learning," *J. Netw. Comput. Appl.*, vol. 163, Aug. 2020, Art. no. 102662.
[46] B. Pernici, M. Aiello, J. vom Brocke, B. Donnellan, E. Gelenbe, and M. Kretsis, "What IS can do for environmental sustainability: A report from CAiSE'11 panel on green and sustainable IS," *Commun. Assoc. Inf. Syst.*, vol. 30, no. 1, p. 18, 2012.
[47] E. Gelenbe, Z.-H. Mao, and Y.-D. Li, "Function approximation with spiked random networks," *IEEE Trans. Neural Netw.*, vol. 10, no. 1, pp. 3–9, Jan. 1999.
[48] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Syst.*, vol. 2, no. 4, pp. 303–314, Dec. 1989, doi: 10.1007/BF02551274.
[49] S. Basterrech, S. Mohammed, G. Rubino, and M. Soliman, "Levenberg—Marquardt training algorithms for random neural networks," *Comput. J.*, vol. 54, no. 1, pp. 125–135, Jan. 2011, doi: 10.1093/comjnl/bxp101.
[50] S. Timotheou, "Fast non-negative least-squares lerning in the random neural network," *Probab. Eng. Informational Sci.*, vol. 30, no. 3, pp. 379–402, Jul. 2016.
[51] E. Gelenbe, "G-networks with instantaneous customer movement," *J. Appl. Probab.*, vol. 30, no. 3, pp. 742–748, 1993.
[52] P. G. Harrison and E. Pitel, "Sojourn times in single-server queues by negative customers," *J. Appl. Probab.*, vol. 30, no. 4, pp. 943–963, Dec. 1993.
[53] P. G. Harrison and E. Pitel, "The M/G/1 queue with negative customers," *Adv. Appl. Probab.*, vol. 28, no. 2, pp. 540–566, Jun. 1996.
[54] J.-M. Fourneau, L. Kloul, and F. Quessette, "Multiple class G-networks with jumps back to zero," in *Proc. 3rd Int. Workshop Model., Anal., Simul. Comput. Telecommun. Syst.*, 1995, pp. 28–32.
[55] M. U. Caglayan, "G-networks and their applications to machine learning, energy packet networks and routing: Introduction to the special issue," *Probab. Eng. Informational Sci.*, vol. 31, no. 4, pp. 381–395, Oct. 2017.

EROL GELENBE (Life Fellow, IEEE) received the B.S. degree from Middle East Technical University, the M.S. and Ph.D. degrees in electrical engineering from the Tandon School, New York University, and the D.Sc. degree in mathematical sciences from Sorbonne University, Paris. He was a Chair Professor with the University of Liège, the University of Paris-Saclay, Paris-Descartes University, NJIT, Duke University, UCF, and Imperial College London. He is currently a Professor with the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, a Research Professor with Yaşar University, Turkey, and a Researcher with the I3S CNRS Laboratory, University Côte d'Azur. He invented methods to optimize computer and network performance, including diffusion approximations, G-networks, the random neural network and its machine learning algorithms, and AI-based network routing that enables multi-party internet communications, for which he received the 1996 Grand Prix France Télécom of the French Academy of Sciences, the 2008 ACM SIGMETRICS Life-Time Achievement Award, and the 2017 Mustafa Prize. The Mathematics Genealogy Project ranks him among the world's top 25 Ph.D. advisors for graduating 95 Ph.D. students. He also develops methods that enhance cybersecurity, performance and sustainability in computer systems and networks. He is a fellow of ACM, IFIP, RSS, IET, the National Academy of Technologies of France, the Science Academy of Turkey, the Royal Academy of Science, Arts and Letters of Belgium, and the Science Academy of Poland, and an Honorary Fellow of the Hungarian Academy of Sciences and the Islamic Academy of Sciences. He received the honors of Knight of the Légion d'Honneur, the Commander of Merit by both France and Italy, and the Commander in the Order of the Crown of Belgium. He chairs the Informatics Section of Academia Europaea.



MERT NAKIP (Student Member, IEEE) received the B.Sc. (Hons.) and M.Sc. degrees in electrical and electronics engineering from Yaşar University, Izmir, Turkey, in 2018 and 2020, respectively. He is currently pursuing the Ph.D. degree with the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Gliwice, Poland. His thesis focused on the application of machine learning methods to the IoT and was supported by the National Graduate Scholarship Program of TÜBİTAK 2210C in High-Priority Technological Areas. He is a Research Assistant with the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Gliwice, Poland. He is a Researcher with the IoTAC Research and Innovation Action of the European Commission H2020 Program. His design of a multi-sensor fire detector via machine learning methods was ranked #1 nationally at the Industry-Focused Undergraduate Graduation Projects Competition organized by the Turkish Scientific and Technological Research Council (TÜBİTAK).