IoT Network Attack Detection and Mitigation

Erol Gelenbe, Piotr Fröhlich and Mateusz Nowak
IITIS-PAN, Polish Academy of Sciences
ul. Baltycka 5, PL Gliwice 44100, Poland
Stavros Papadopoulos, Aikaterini Protogerou,
Anastasis Drosou and Dimitrios Tzovaras
ITI-CERTH, GR 57001 Thermi
Thessaloniki, Greece

Abstract—Cyberattacks on the Internet of Things (IoT) can cause major economic and physical damage, and disrupt production lines, manufacturing processes, supply chains, impact the physical safety of vehicles, and damage the health of human beings. Thus we describe and evaluate a distributed and robust attack detection and mitigation system for network environments where communicating decision agents use Graph Neural Networks to provide attack alerts. We also present an attack mitigation system that uses a Reinforcement Learning driven Software Defined Network to process the alerts generated by the attack detection system, together with Quality of Service measurements, so as to re-route sensitive traffic away from compromised network paths using. Experimental results illustrate both the detection and re-routing scheme.

Keywords—IoT Security, Graph Neural Nets, Cognitive Packet Network, Random Neural Networks, Software Defined Networks

I. INTRODUCTION

The IoT [1] has the potential to improve the critical processes that are at the heart of our socio-economic systems [2], [3]. However, it creates raises risks that go way beyond the individual technologies such as the Internet, wireless networks and machine to machine systems [4], [5]. In addition to risks related to system malfunctions, quality of service (QoS) failures, and excessive energy consumption, the theft and tampering of data, conventional network attacks and attacks that attempt to deplete the energy of autonomous sensors and actuators also need to be considered [6]–[12].

Since IoT devices can carry out real-time measurements and controls with time constants that are far shorter than human reaction times, we must design IoT networks that both detect and mitigate security risks automatically and adaptively, while preserving Quality of Service (QoS), and being energy efficient [13].

Thus we propose an autonomic [14] scheme that offers (a) a distributed attack detection approach based on deep learning (DL) and graph neural networks to achieve high detection probabilities with low false alarm rates [15], [16], and (b) a mitigation method that exploits network Self-Awareness [17], [18] centered on Software Defined Networks [19] to achieve secure QoS based routing of traffic flows using machine learning and adaptivity [20]–[22].

Section II discusses the multi-agent system (MAS) for network attack detection, and summarizes its performance. The overall system architecture for attack detection and mitigation is presented in Section III. The node attack detection probability estimated by MAS is used to compute safer paths in the network using reinforcement learning as described in Sections III-A and III-B. Experimental results are presented in Section III-C, and Section IV presents conclusions and future work.

II. DISTRIBUTED ATTACK DETECTION

Since IoT systems are distributed and have a heterogeneous structure which is an additional challenge for real-time anomaly detection [23], [24]. Thus the distributed MAS for detecting attacks monitors the network traffic in a distributed manner. The output of this analysis is provided to the novel routing system described in Section III that mitigates attacks with a SDN based routing engine. The MAS’s mutually communicating multiple agents can improve its robustness incorporating redundancy in he detection algorithm [25]. The MAS also offers scalability, since its modularity allows new agents to be added if the IoT network grows, and agents exchange information [26] in a structure that is inspired by Graph Neural Networks [15], [16].

The structure of the IoT network is reflected by the graph $G(V,E)$, where $V$ corresponds to the set of nodes of the IoT network, and $E \subseteq V \times V$ is a set of edges which represent the nodes which communicate (directly or indirectly) with each other through the IoT network. The nodes can represent sensors or actuators, edge nodes, servers or routers in the IoT network.

We associate a real-valued feature vector $x_i \in \mathbb{R}^N$ to each $i \in V$, where $N_V$ is its length. Similarly we associate the feature $N_E$-vector of real numbers $e_{ij} \in \mathbb{R}^{NE}$ with each edge $(i,j) \in E$. An example of the features for the nodes and edges is given in Table I.

Measurements to collect the feature vector parameters are taken in the IoT network during successive disjoint time slots $[(t-1)T,tT]$ where $T$ is the slot length and $t$ is the integer slot index. The slots are long enough to provide representative data, but short enough to reflect time variations in the system. Thus all feature vectors are also associated with individual slots and successive values. Thus $x_{i,k}^t$ the $k$–th successive value of $x_i$ within the $t$–th slot, while $e_{ij,k}^t$ is the $k$–th successive value of $e_{ij}$ in the $t$–th slot. We will denote by $e_{ij,k}^t$ and $X_{i,k}^t$, respectively, the feature vector values at the end of the $t$–th slot, while $e_{ij,0}^t$ and $X_{i,0}^t$ are their values when the measurement system starts to operate and the first slot begins.

The MAS uses four Deep Neural Networks (DNNs):
The EDNN (edge DNN) which undertakes the update:
\[
\epsilon_{ij}^{k+1,t} \leftarrow \text{EDNN}(x_{ij}^{k,t}, \hat{x}_{ij}^{k,t}, \epsilon_{ij}^{k,t}) .
\]
EDNN uses an edge’s current features, and the features of the two nodes at its edges, to update its features.

- The NDNN (node DNN) which undertakes the update:
\[
x_{ij}^{k+1,t} \leftarrow \text{NDNN}(\hat{x}_{ij}^{k,t}),
\]
and updates a given node’s features using the average value of the features of the nodes with which it communicates and of the related edges, with:
\[
\hat{x}_{i}^{t} = \frac{1}{m_{j}} \sum_{i, \text{ s.t.} \ (i,j) \in E} x_{ij}^{k,t},
\]
where \(m_{j} = |\{(i, (i,j) \in E)\}|\) is the number of neighbours of \(j \in V\), and \(s.t.\) stands for “such that”.

- The third DNN, CLN builds \(p_{Ni}^{k,t}\), the probability that node \(i\) is compromised, only using its own feature vector:
\[
p_{Ni}^{k,t} \leftarrow \text{CLN}(x_{i}^{k,t}).
\]
Finally the fourth DNN, CLNEI builds \(p_{Nj}^{k,t}\), the probability that its neighbour \(j\) is compromised:
\[
p_{Nj}^{k,t} \leftarrow \text{CLNEI}(\epsilon_{ij}^{k,t}, \hat{e}_{ij}^{k,t}, x_{i}^{k,t}).
\]

These four networks constitute the node agent, and are duplicated in each node, and can be trained off-line. They operate in each node separately and asynchronously. Starting from feature vectors from data gathered during the previous time slot, they update the decision probabilities and communicate their updated feature values and decision probabilities to their neighbours. These computations are shown schematically in Figures 1 and 2.

<table>
<thead>
<tr>
<th>#</th>
<th>Target</th>
<th>Feature description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>edge/node</td>
<td>Average number of packets sent</td>
</tr>
<tr>
<td>2</td>
<td>node</td>
<td>Average number of packets received</td>
</tr>
<tr>
<td>3</td>
<td>edge/node</td>
<td>Average number of bytes sent</td>
</tr>
<tr>
<td>4</td>
<td>node</td>
<td>Average number of bytes received</td>
</tr>
<tr>
<td>5</td>
<td>edge/node</td>
<td>Average connection duration</td>
</tr>
</tbody>
</table>

The multiple iterations of these operations represented by the integer \(k\), allow nodes and edges to update and exchange information multiple times within each time interval as shown in Figure 2. The entire network of nodes is trained in a supervised manner using the back-propagation algorithm with cross-entropy as the cost function for classification.

In the system that we have described, each node agent can perform anomaly detection not only on itself, but also with regard to its neighbouring nodes. This redundancy improves the algorithm’s robustness in cases where some agents may fail, since the agents at neighbouring nodes may still detect anomalies that occur at their neighbours. Finally, to combine the overlapping decisions of different agents into a single decision for each node in the IoT network, we use a simple aggregation method, where a node is considered anomalous if at least one agent has reported it as being anomalous. More sophisticated aggregation schemes can be considered in future work.

To evaluate the proposed approach, we have used a simulated infiltration attack, where the attacker tries to infiltrate the network by scanning a range of IP addresses in order to run services, and performs a dictionary attack in order to find vulnerable IoT devices. The resulting Receiver Operating Characteristic (ROC) is shown in Figure 3. The overall results are summarized in Table II. The metrics used for the evaluation are the Area Under the Curve (AUC) score, the detection accuracy, the utilized Bandwidth, and the Power consumption [27]. For the first two metrics, which measure detection efficiency, the proposed approach outperforms all other methods we have tested for anomaly detection, achieving Area Under the Curve (AUC) score and accuracy of 0.99, compared to 0.97 for the second (random forest) and third (decision tree) classifiers. With respect to the last two metrics, i.e. Bandwidth...
and Power consumption, the proposed decentralized approach greatly reduces the bandwidth required for monitoring, which in turn reduces the power that is consumed. However, the execution time and the power consumed at each node will determine the energy consumed by our approach, so that a slower low power approach may consume more energy than a very fast method that uses higher power.

TABLE II

<table>
<thead>
<tr>
<th>Method</th>
<th>AUC</th>
<th>Accuracy</th>
<th>Bandwidth</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td>0.99</td>
<td>0.99</td>
<td>452.26 bits/s</td>
<td>353 W</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.96</td>
<td>0.96</td>
<td>33985.7 bits/s</td>
<td>353.01 W</td>
</tr>
<tr>
<td>SVM</td>
<td>0.98</td>
<td>0.98</td>
<td>33985.7 bits/s</td>
<td>353.01 W</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.97</td>
<td>0.97</td>
<td>33985.7 bits/s</td>
<td>353.01 W</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.69</td>
<td>0.70</td>
<td>33985.7 bits/s</td>
<td>353.01 W</td>
</tr>
</tbody>
</table>

Fig. 3. Receiver Operating Curve curve of detection scheme for the simulated infiltration attack scenario, using several attack detection mechanisms. The approach we propose has the highest Area Under the Curve (AUC) score.

III. SYSTEM ARCHITECTURE AND ROUTING ENGINE

The Architecture of the SerIoT system is shown in Figure 4, which shows several interconnected smart forwarding engines (SFE) connected connected to several fixed or mobile IoT devices, IoT gateways, and to Cloud Servers which may also be Fog servers. SFEs may be connected to Honeypots (H) whose role is to attract and interpret attacks. Specific software at IoT devices and gateways may be used to detect attacks [28], but here we use the decisions provided by the distributed attack detector of Section II. QoS, Energy and Security are monitored and forwarded to “smart controllers and routing engines” (SRE) which operate as OpenFlow SDN Controllers to choose paths and download the to the SFEs [29], [30].

The SRE uses the Cognitive Packet Network (CPN) routing algorithm [31], implemented with the Random Neural Network (RNN) and Reinforcement Learning [32] which has also attracted interest from industry [33]. It extends a standard SDN network using the SRE, with SFEs which are extensions of SDN forwarders, and the Monitoring and Anomaly Detection (MAD) module which detects potential threats from data collected by SerCPN, Active Honeypots that attract attacks, deflect them to safe IP locations, and inform the SRE, and local attack detectors at nodes and gateways [28].

Each SFE, shown schematically in Figure 5, switches SDN flows according to the OpenFlow protocol. In addition to payload traffic, SFEs also forward smart packets (SPs) which gather security, QoS and energy usage data from the SFEs, IoT devices and gateways. Each SFE has a Cognitive Packet Agent (CPA) that unpacks the SP, adds its own data to the list stored inside, packs it again and forwards it to the SRE. SPs travel over paths, carrying information provided by the SFEs on the path. When a CPA recognizes that a SP has attained the end of its path, it encapsulates it and forwards it to the corresponding SRE, where its data is unloaded into the local Network State Database (NetStatDB). SFEs can also forward data that is monitored, such as packet counters or byte counters) to the MAD at the SRE.

Each SRE is based on ONOS [34] and its software implemented as an ONOS application with the three main modules shown in Figure 6. The heart of the system is the Cognitive
Routing Module (CRM) that implements decision taken by a RNN [35] with Reinforcement Learning for path selection based on QoS, security or energy consumption in the network. The MAD detects attacks at nodes using MAS of Section II. Other attack detection methods will also be considered in future work [28].

The SRE selects paths based on a Goal Function \( G(f, P) \) which has non-negative real values and which must be minimized, where \( f \) denotes the packet flow to or from an IoT device or end-user software, and \( P \) denotes a path travelled by and the \( d_f \), and the MAS in Section II provides the probability \( p_i \) that node \( i \) is under attack. For some SFE or network node \( i \), the Trust Level \( T(f, i) \) is non-negative number that is high when \( i \) is not deemed secure enough to convey the flow secure \( f \). Also, \( S(f, i) \) is defined as the sensitivity of \( f \) to attacks at node \( i \).

### A. Linking \( T(f, i) \) to the \( p_i \) from MAS

Let \( A > 0 \) be a large positive constant used so that \( T(f, i) \) may take values comparable to QoS values such as the delay of links, and \( p_i \) is the probability that an attack is detected at node \( i \) by MAS. Then \( T(f, i) = A(1 - p_i) \) is the security level of \( f \) related to node \( i \). Let \( S(f, i) \) be the sensitivity of \( f \) to the security of \( e \). The Insecurity Factor \( I(f, i) \) is then used to “separate” \( e \) and \( f \):

\[
I(f, i) = [S(f, i) - T(f, i)]^+, \tag{6}
\]

where use the notation \( [X]^+ = X \) if \( X > 0 \), and \( [X]^+ = 0 \) if \( X < 0 \). If we take \( S(f, i) = A \), then \( I(f, i) = A p_d(i) \), and we see that as \( p_i \) increases, the “security cost” incurred by \( f \) as it travels through \( i \) increases. The “Insecurity Factor” that relates flows to paths, is:

\[
I(f, P) = \sum_{i \in P} I(f, i). \tag{7}
\]

When less attention is paid to security, we may take the smaller value \( I(f, P) = max_{i \in P} I(f, i) \).

Let \( L(f, p) \) be the packet loss ratio, and \( D(f, P) \) be the forwarding delay for a packet of \( f \) on path \( P \), while \( J_i \) is the energy consumption per packet at node \( i \). The packet retransmissions due to packet losses [31], [36] result in:

\[
Q(f, P) = \frac{D(f, P)}{1 - L(f, P)}, \text{ hence} \tag{8}
\]

\[
G(f, P) = \begin{cases} I(f, P), & I(f, P) \geq \theta, \\ Q(f, P), & \text{if } I(f, P) < \theta, \end{cases} \tag{9}
\]

where \( \theta \geq 0 \) is a security threshold that can be chosen based on the importance of security considerations for this system. \( G(f, i) \) or \( G(f, P) \) are quantities to be minimized, but Reinforcement Learning (RL) requires a “reward” \( R(f, i) \) that should be maximized, where:

\[
R(f, i) = \frac{1}{G(f, i)}, \quad R(f, P) = \frac{1}{G(f, P)}. \tag{10}
\]

### B. Reinforcement Learning

The metrics that feed into the quantity \( R(f, i) \) are all collected via measurements, except the ones that are initially fixed, namely \( \theta, S(f, i) \) and the parameters such as \( \alpha, \beta, \gamma \) describing the relative importance of different factors. Therefore the RL based routing scheme to improve network security, QoS and energy consumption, collects at each node \( i \) the quantity \( G(f, i) \) and hence \( R(f, i) \), at successive arrivals of a SP packet to an SDN controller. The SP will collect bring back the relevant data for \( R(f, i) \) concerning each node \( i \) that the SP has visited, to the SDN router that exploits the RL algorithm to compute a “next hop” for SPs. Let the integer \( l \) refer to the \( l-th \) value of the reward \( R_l(e, f) \) computed by the SDN router for the node \( i \) and flow \( f \). The RL algorithm will first compute the quantity:

\[
T_l = aT_{l-1} + (1 - a)R_l, \quad 0 \leq a < 1, \tag{11}
\]

that describes the historical behaviour of the reward, and tells how well the network has been doing. The RL algorithm will then compute a set of RNN [35] weights as follows.

For an \( N \) node RNN, where \( N \) is the number of outgoing links for node \( i \), we associate with each outgoing link \( i \) a neuron whose state is represented by the “excitation probability” \( q_i \) of the RNN. The RNN weights are real numbers \( W^+_ij, \quad W^-ij \geq 0 \) for \( i, j \in \{1, \ldots, N\} \). From RNN theory [35] we know that:

\[
q_j = \frac{\lambda_j^+ + \sum_{i=1}^{N} q_i W^+_ij}{r_j + \lambda_j^+ + \sum_{i=1}^{N} q_i W^-ij}, \tag{12}
\]

where \( r_j = \sum_{i=1}^{N} [W^+_ij + W^-ij] \) is the “total firing rate” of the neuron \( j \). \( \lambda_j^+, \lambda_j^- \) are, respectively, the arrival rate of excitatory and inhibitory spikes to neuron \( j \) from outside the neuron \( i \), which are set so that when all connection weights are equal, then all neurons in the network have an excitation probability of \( q_j = 0.5 \).

Let \( k \) be the index of the neuron for which, after the \( v-1-th \) update of the RNN we have \( q_k = \max\{q_1, \ldots q_N\} \). Also
save the current value \( r_j \leftarrow \sum_{i=1}^{N} [W_{ij}^+ + W_{ij}^-] \). Note that the node from which a SP entered the node where the next-hop decision is being taken will not be used as the next-hop, so that the decision at a given node will select one outgoing link among \( N - 1 \). The RNN’s weights are updated as follows:

\[
\begin{align*}
\text{If } R_l \geq T_{l-1} & : \forall j \neq k, j \neq i, (\text{Previous}), i \neq k, \\
W_{ik}^+ & \leftarrow W_{ik}^+ + R_l, W_{ij}^- & \leftarrow W_{ij}^- + \frac{R_l}{N - 2}, \\
\text{If } R_l < T_{l-1} & : \forall j \neq k, j \neq i, (\text{Previous}), i \neq k, \\
W_{ik}^- & \leftarrow W_{ik}^- + R_l, W_{ij}^+ & \leftarrow W_{ij}^+ + \frac{R_l}{N - 2},
\end{align*}
\]

where we divide by \( N - 2 \) since we are excluding \((i, \text{Previous})\) from which the SP initially arrived, also not increasing the inhibitory weights of the winner node when \( R_l \geq T_{l-1} \), nor increasing the excitatory weights of the loser node when \( R_l < T_{l-1} \). We then also renormalize the weights as follows:

\[
\begin{align*}
q_j^* & \leftarrow \sum_{i=1}^{N} [W_{ij}^+ + W_{ij}^-], \\
W_{ij}^+ & \leftarrow W_{ij}^{+ \times} r_j, W_{ij}^- & \leftarrow W_{ij}^{− \times} r_j.
\end{align*}
\]

Finally we calculate all the \( q_j \) from equation (12), to select the new output link for flow \( f \) at node \( i \) by selecting the new output link \( k^* \) with \( q_{k^*} = \max \{q_j, j \neq i, 1 \leq j \leq N\} \).

C. Experimental Results

Experiments were run on a network with several with SFEs composed of Linux boxes with ARMv8 processors (1.4 GHz, 4 GBit Ethernet, 2.4GHz and 5GHz 802.11b/g/n/ac WiFi interface). They were configured to use Ethernet as the data plane interface shown in Figure 7, and WiFi for management, monitoring and for communications with the SRE. In the figure \( s1, \ldots, s7 \) denote SFEs, \( h1, \ldots, h4 \) are IoT devices each with a 633MHz MIPS processor, 100Mb/s Ethernet port, and 2.4Mhz WiFi connection used as a management port, the SRE is a workstation connected by WiFi to the test-bed, and the MAD is installed on a separate workstation connected by Ethernet port to the SRE, and by WiFi to the test-bed. The type of experiments we run are represented by the measured event trace shown in Figure 8.

In our experiments, every distinct pair of IoT devices in \{\( h1, \ldots, h4 \)\} forward 20 packets/sec or roughly 20 – 40 Kb/sec with 12 ongoing connections so that each packet rate is compatible with IoT connections monitoring temperature or water flow in pipes, etc. SPs are generated by every edge SFE at 10 packets/sec. SRE management traffic includes OpenFlow commands, link and topology discovery packets, and traffic statistics. Management packet traffic through SFEs measured using the Wireshark packet analyzer was four to five times higher than SP traffic.

The experiments illustrate the system’s aptitude to be Self-Aware and adapt, and we measure the SRE’s reaction time to abrupt changes in the security conditions expressed by the trust level for connections, and track changes to parameters \( R_l \) and \( T_{l-1} \) given in (11) for the Reinforcement Learning Algorithm’s successive steps \( l \). The SRE was programmed to change network paths every 5 seconds, so that the experimental results we present are limited by this constraint that has been placed to avoid frequent changes that may increase system overheads. The effect of changing the trust \( TF(f, i) \) is shown in Figure 8. The quantity that is plotted is the proportion of the time it takes the SRE to respond to a large increase of 100 in the value of \( TF(f, i) \) for a node \( i \) on the path that is currently used. We see that the reaction tie is on average around 1 second, with a maximum value around 2 seconds.

IV. Conclusions and Future Work

In this paper we have described a system that detects node attacks in an IoT network using a deep learning based Multi-ple Agent System, and which exploits attack detection in order to automatically mitigate the attacks by re-routing sensitive traffic using Reinforcement Learning, while also taking into consideration the QoS of different network paths. We have also provided a preliminary evaluations of the performance of both the attack detection and mitigation system.

In future research, additional measurements, fine tuning of parameters, and experimental results will be conducted.
to better evaluate the interaction of QoS and security in complex adaptive IoT networks. Using methods from diffusion processes [37], [38], we will also investigate the transients due to SDN based frequent route updates, in response to potential attacks and changes in QoS. We also plan to install locally operating anomaly and attack detection software at nodes to examine the effect of reducing computation times for anomaly detection, improving the response of the system to QoS and security changes, while possibly reducing the accuracy that is offered by the proposed network-wide anomaly detection scheme.

ACKNOWLEDGEMENT

The authors are grateful for the support provided by the EC H2020-IOT-2017 Program for the SerIoT Research and Innovation Action via Grant Agreement No. 780139.

REFERENCES


