A Robust Deep-Learning-Enabled Trust-Boundary Protection for Adversarial Industrial IoT Environment

Mohammad Mehedi Hassan, *Senior Member, IEEE*, Md. Rafiul Hassan, *Member, IEEE*, Shamsul Huda, and Victor Hugo C. de Albuquerque, *Senior Member, IEEE*

***Abstract*—In recent years, trust-boundary protection has become a challenging problem in Industrial Internet of Things (IIoT) environments. Trust boundaries separate IIoT processes and data stores in different groups based on user access privilege. Points where dataflow intersects with the trust boundary are becoming entry points for attackers. Attackers use various model skewing and intelligent techniques to generate adversarial/noisy examples that are indistinguishable from natural data. Many of the existing machine-learning (ML)-based approaches attempt to circumvent this problem. However, owing to an extremely large attack surface in the IIoT network, capturing a true distribution during training is difficult. The standard generative adversarial network (GAN) commonly generates adversarial examples for training using randomly sampled noise. However, the distribution of noisy inputs of GAN largely differs from actual distribution of data in IIoT networks and shows less robustness against adversarial attacks. Therefore, in this article, we propose a downsampler-encoder-based cooperative data generator that is trained using an algorithm to ensure better capture of the actual distribution of attack models for the large IIoT attack surface. The proposed downsampler-based data generator is alternatively updated and verified during training using a deep neural network discriminator to ensure robustness. This guarantees the performance of the generator against input sets with a high noise level at time of training and testing. Various experiments are conducted on a real IIoT testbed data set. Experimental results show that the proposed approach outperforms conventional deep learning and other ML techniques in terms of robustness against adversarial/noisy examples in the IIoT environment.**

***Index Terms*—Adversarial attack, deep learning (DL), Industrial Internet of Things (IIoT), robustness, trust boundary protection.**

Manuscript received May 15, 2020; revised July 25, 2020; accepted August 19, 2020. Date of publication August 24, 2020; date of current version June 7, 2021. This work was supported in part by the Deanship of Scientific Research at King Saud University through the Vice Deanship of Scientific Research Chair of Smart Technologies. *(Corresponding author: Mohammad Mehedi Hassan.)*

Mohammad Mehedi Hassan is with the College of Computer and Information Sciences and the Research Chair of Smart Technologies, King Saud University, Riyadh 11543, Saudi Arabia (e-mail: mmhassan@ksu.edu.sa).

Md. Rafiul Hassan is with the College of Arts and Sciences, University of Maine at Presque Isle, Presque Isle, ME 04769 USA (e-mail: md.hassan@maine.edu).

Shamsul Huda is with the School of Information Technology, Deakin University, Burwood, VIC 3125, Australia (e-mail: shamsul.huda@deakin.edu.au).

Victor Hugo C. de Albuquerque is with the Department of Computer Science, Universidade de Fortaleza, Fortaleza 60811-905, Brazil (e-mail: victor.albuquerque@unifor.br).

Digital Object Identifier 10.1109/JIOT.2020.3019225

I. Introduction

INDUSTRY 4.0 has combined the cyber world and the physical world through the Internet and transformed conventional industrial systems into a revolutionary global network known as the Industrial Internet of Things (IIoT) [1], [2]. IIoT networks connect numerous IoT devices, such as sensors, programmable logic controllers, actuators, intelligent electronic devices, and cyber–physical systems (CPSs) over the Internet. Owing to the integration of the physical and cyber world, a whole range of industrial operations, related subsystems, and business processes for design, infrastructure, monitoring and control, scheduling, and maintaining the value chain are intensively interconnected within the systems and to the external networks using the Internet. This integration and interconnection provides enormous flexibility and agility for precise control of physical processes, autonomous management, and inter/intra-industrial system collaboration through less expensive production data collection, data analytics, and intelligent processing in real time, resulting in much higher revenue. However, the integration highlights the limitations and vulnerabilities of industrial protocols, networks, systems, and services to the open TCP/IP network and exposes the field devices of supervisory control and data acquisition (SCADA) networks to severe security threats [3]–[5]. This risk is boosted by the existing flaws and security holes of conventional IT systems. This has attracted cybercriminals to the IIoT networks as a primary target, and cyberattacks on IIoT networks have been a lucrative business venture for the attackers.

Therefore, current IIoT networks utilize trust boundary techniques as the primary measure to maintain security objectives, such as confidentiality, integrity, and availability of IIoT data and services and to safeguard the core operations of CPS from compromise [3], [6]. For example, large IIoT networks, such as smart grids, as mentioned in Fig. 1, implement trust-boundary techniques to segment the networks into different domains of generation, distribution, transmission, distributed energy resources, and energy management systems. Trust boundaries

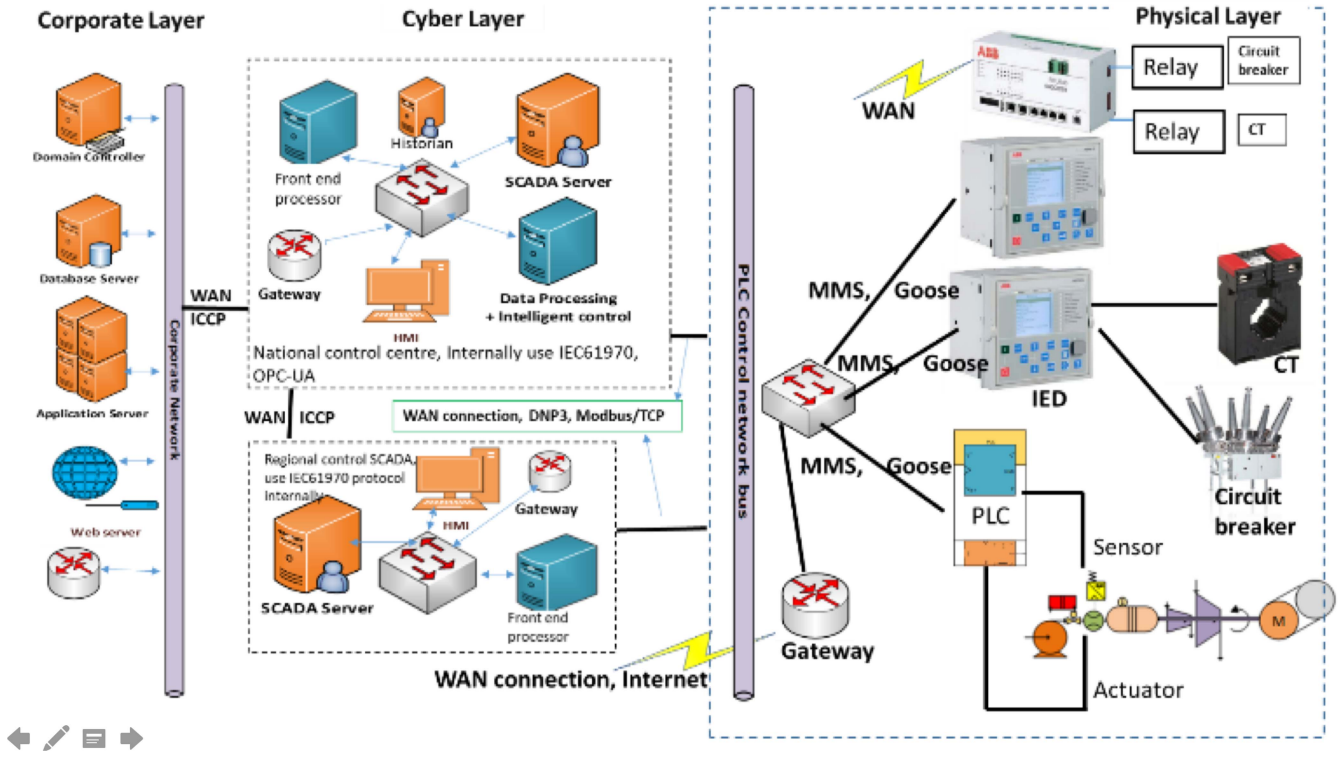

Fig. 1. IIoT networks with different domains and layers.

reduce the scope of the attack surfaces and models and controls the level of access to sensitive information. The core techniques of trust-boundary protection are malicious traffic analysis and intrusion detection, which are used in demilitarized zone architecture and virtualization to enforce security policy and prevent attackers from conducting footprinting, scanning, enumeration, and attack delivery activities.

Machine learning (ML) techniques, such as support vector machines (SVMs), decision trees, and deep learning (DL) approaches (such as convolution networks, CNN, and LSTMs) [3], [4], [7]–[9] have been used in many research articles and industrial implementations for malicious traffic, malicious code-executables analysis and detection and intrusion detection as part of the boundary protection. Detection models based on the ML and DL algorithms constitute a decision hyperplane that can separate normal operational traffic from attack type through a training process. In the training, an optimal decision surface can be obtained if a large amount of labeled data is used where test data come from the same distribution as the training data. As mentioned in Fig. 2, in principle, because the test example is farther away from the decision surface, the detection algorithm has more confidence in its classification, and as it gets closer to the decision surface, the confidence level decreases. If a large amount of training data is not obtained, finding a true decision surface is difficult. This worsens the confidence level of classification and can be incorrect for the test examples, which are extremely close to the decision surface. This is because overly large networks, such as IIoT networks and smart grids (as presented in Fig. 1) have an outsized attack surface, heterogeneous communication infrastructure, and protocols, such as power line carrier, wired

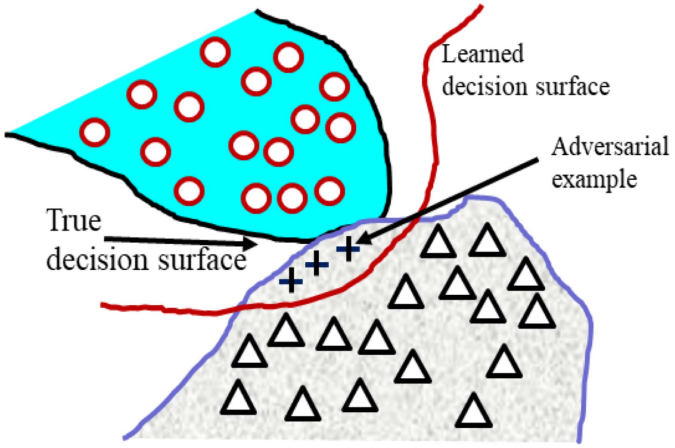

Fig. 2. True decision surface and adversarial examples.

connection, cellular, LAN, WAN, radio communications, wireless, and WiMAX and a wide range of multilevel protocols, such as DNP3.0, Modbus, IEC61850, and ICCP (IEC69870-5). Therefore, attack models become highly dynamic, and data distribution changes, which differs from the existing training database. In this case, the performance of detection engine is reduced as a result of rapidly changing attack models. Different automatic update strategies have been proposed, such as semisupervised learning [7]–[9], online learning [10], [11], and incremental learning [12]. These approaches improve the data distribution problem to a certain extent by adding new data to the system, but attack arising from zero-day vulnerabilities or adversarial noise added to the data set cannot be

completely identified because data are added to the system after post-scanning, enumeration, or attack events. For large networks as in Fig. 1 a large amount of normal operational data can be achieved, but it is challenging to find a large amount of attack data at prescanning, pre-enumeration, or preattack stage from zero-day vulnerabilities or before the vendor patch releases. Therefore, the implemented detection model in boundary protection is left to be trained on a suboptimal knowledge based, which is one of the main reasons for the deviation of the learned decision surface from the true decision surface, as Fig. 1 shows. This leaves most ML and DL approaches vulnerable against zero-day vulnerabilities or noisy data. Attackers perturb the input data using model skewing, creating a well-designed attack model to generate samples that are extremely close to the decision surface and inside the normal operational region (Fig. 1), known as adversarial/noisy examples, to evade these samples as benign types. Adversarial vulnerability is the most critical threat and requires resilient boundary protection.

Researchers have introduced different approaches to make the detection model robust against data perturbation or variation in data distribution, including architectural variation [13], [14], metaheuristic hyperparameter tuning of DL [15], ensemble learning of varying structure [16], and multimodal fusion architecture [17]. Many scholars have also proposed robust ML classifiers, such as regularization for SVM, augmented kernel of SVM [18], and Kernel combination of CNN and SVM [19]. The approaches proposed in the literature are targeted at improving the mathematical models of the training algorithms, which enhances their robustness to a certain extent but still leaves them vulnerable to test data tampering attacks, dynamic attack models, and gray-box attack with the feeding of the networks at training time [20] for large attack surfaces, such as segmented networks (Fig. 1). To this extent, adversarial example generation and adversarial training are an important direction for resilient boundary protection. Evolutionary learning has been used to generate adversarial examples [21], which have suffered from computational efficiency. Recently, generative adversarial networks (GANs) have attracted much attention as one of the most promising approaches, capable of learning complex data distribution and at the same time generating realistic examples of data that are indistinguishable from natural data. GAN has been used in radar target recognition [22], license plate recognition systems [23], and image classification [24].

Standard GAN uses randomly sampled noise with input to generate adversarial samples. Therefore, the distribution of noisy inputs largely differs from the actual distribution of the data and shows less robustness against adversarial attacks. In contrast, in the proposed approach the generator is replaced by a downsampler that encodes the actual data in a form such that the performance of the discriminator is enhanced. Note that, instead of adversarial behavior of the generator in GAN, the proposed approach utilizes the downsampler to cooperate with the discriminator. In summary, in the proposed approach, a downsampler encoder is optimized to map the attack/nonattack training data (which might have embedded adversarial noise added to it) to a distribution such that the model's overall classification accuracy is enhanced. The training algorithm alternatively updates the weights of the encoder network and the decoder classifier and attempts to fit with the actual distribution. The novelty of the article is described as follows.

1) A robust decision boundary optimization algorithm (through altering the training for the downsampler and discriminator) is proposed to increase the robustness of the attack/nonattack detection engine for a nonnoisy as well as the adversarial environment.
2) A novel cooperative training algorithm is proposed to train the downsampler, which provides better distribution for noisy examples with actual distribution.
3) The proposed approach is verified using real IIoT attack data to support multilevel IIoT network protocols that can overcome the limitations of conventional IT security control techniques.

The remainder of this article is organized as follows. Section II presents a detailed literature review of recent approaches to combat cyberattacks, particularly adversarial attacks on IIoT networks. Section III presents the proposed approaches of this article. Section IV presents our experimental analysis, attack modeling, attack strategy framework, and results for the proposed model. Section V describes this study's conclusion.

II. Related Works

Various studies address adversarial attacks and defense against them in the image processing and computer vision domain [4]. However, limited attention has been paid to adversarial attacks in the industrial IoT environment. To tackle adversarial attacks, most researchers have focused on ML-based adversarial retraining, that is, training the ML or DL classifiers by mixing adversarial samples with the original ones [25]–[29]. For example, Wang [25] presented various adversarial attack methods against a DL based intrusion-detection system in an IoT network scenario. Khoda *et al.* [4] proposed various adversarial learning models in the malware domain in the IIoT environment based on the deep neural network. However, to generate adversarial samples, researchers are using a variety of techniques, such as the evolutionary learning-based approach [14] and the GAN model [30]. Because the first approach suffers from computational efficiency, the GAN model is one of the most popular approaches for generating adversarial examples, which it commonly does for training by using randomly sampled noise. However, the distribution of noisy inputs of GAN largely differs from the actual distribution of the data in IIoT networks owing to its large attack surface. Moreover, generating too many adversarial samples with GAN to cover the IIoT data surface can degrade the performance of a classifier as well as making it less robust against adversarial attacks. Therefore, to make trust-boundary protection more resilient to adversarial attacks in IIoT networks, instead of generating adversarial behavior from GAN, we utilize a downsampler encoder-based cooperative data generator to ensure a better capture of actual distribution of attack models for the large IIoT attack surface.

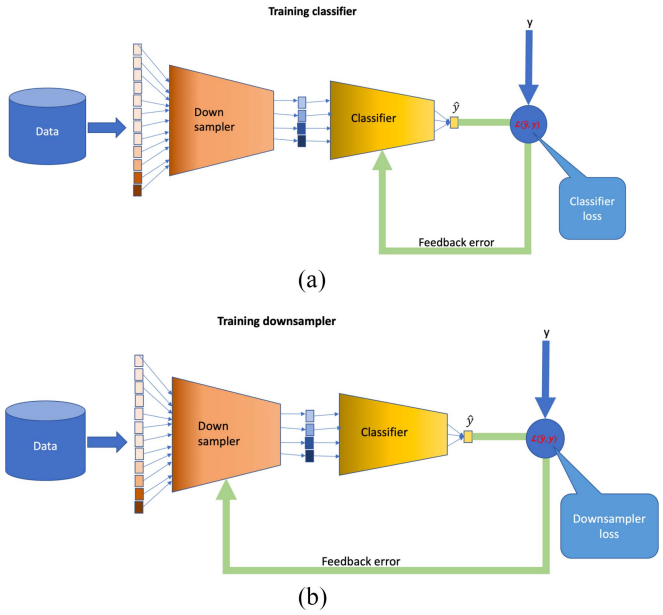

Fig. 3. Schematic of the proposed approach (training steps only). (a) Training of classifier while the weights of downsampler are kept unchanged. (b) Training of downsampler keeping the weights of the classifier unchanged.

The proposed downsampler-based data generator is alternatively updated and verified during training using a deepNN discriminator to ensure robustness.

III. Methodology

Our proposed approach uses two individual deepNN functions. The first function is used as a downsampler, whereby the high-dimensional input is downsampled to a lower number of features. Then, the second function is used as a classifier. Once high-dimensional input is downsampled to a lower dimension, the encoded/downsampled features are fed as input to the classifier to classify the class label. The challenge here is how to train the downsampler because no target value is available; thus, the traditional training algorithm is not applicable. In contrast, training and building the classifier is straightforward because the target values are available for each input sample. Fig. 3 shows the schematic of the proposed approach for both training and SCADA attack identification steps.

As shown in Fig. 3, our approach employs two deepNNs. Hence, the following three steps are adopted to build and train the overall model.

1) Build an initial downsampler g.
2) Build an initial classifier f.
3) Adapt weights/coefficients of downsampler and classifier.

Before describing the proposed model in detail, we represent Table I that lists the notations and corresponding meaning used in this article.

Definition: Given two functions f and g, the composite function $(f \circ g)(x) = f(g(x))$ represents our approach for identifying any attack attempt in SCADA. Here, f and g represent the embedded chained nonlinear functions of the respective DeepNNs. The input $\mathfrak{X}$ is an n-dimensional scalar vector, i.e., $\mathfrak{X} \in \Re^n$.

TABLE I
List of Notations and the Corresponding Meaning

Notation	Meaning
GAN	Generative Adversarial Network
$f(g(x))$	The proposed model
f	An ANN/deepNN used as a classifier
g	An ANN/deepNN used as a downsampler
$\mathfrak{X}$	n-dimensional input data vector
$\mathfrak{D}$	Dataset
x_j	jth element of input data
$\hat{y}_g^i$	Output produced by g
$\hat{y}_h^L$	Output computed for a neuron at $L$the hidden layer
$\hat{y}_i$	The output of the proposed model for ith input vector

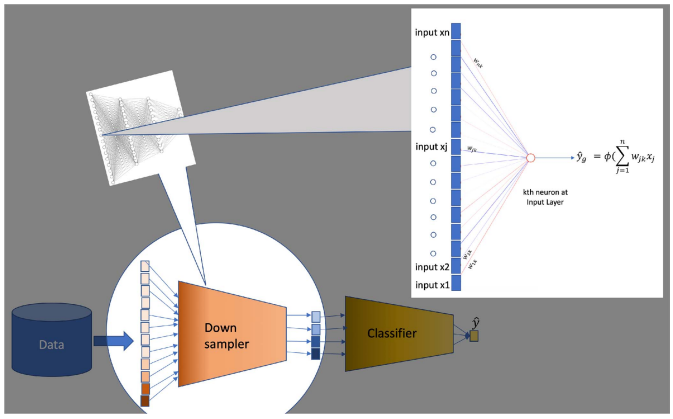

Fig. 4. kth neuron at input layer of downsampler g.

We describe these steps, which are involved in the developed approach, in detail in the sequel.

A. Downsampler

A DeepNN is used as a downsampler g where the input is $\mathfrak{X}$ and the output will be a comparatively low-dimensional data vector. In contrast to the GAN proposed by Goodfellow *et al.* [30], in our approach, the purpose of g is to downsample high-dimensional input vector. The output of g is Y_g^m, i.e., $g(x) = Y_g^m$. The dimensionality of the output vector of g is less than that of the input vector, i.e., $m < n$.

Let us consider a data set $\mathfrak{D}$ where, $\{\mathfrak{X}_1, \mathfrak{X}_2, \ldots, \mathfrak{X}_q\} \in \mathfrak{D}$ with corresponding given class labels $\{y_1, y_2, \ldots, y_q\} \in Y$, $Y \in \{0, 1\}$ ("0" represents normal and "1" represents attack in the SCADA). The output of a neuron (kth) at the input layer of g (alternatively, downsampler DeepNN) is computed as follows:

$$\hat{y}_g^i = g(\mathfrak{X}_i) = \phi\left(\sum_{j=1}^{n} w_{jk} \times x_j\right) \tag{1}$$

where k represents the kth neuron in the input layer, n is the dimensionality of the ith input vector $\mathfrak{X}_i$, x_j = jth element/attribute of input vector $\mathfrak{X}_i$ and w_{jk} is the weight connecting the jth input feature to the kth neuron in the input layer and $\phi(.)$ represents the activation function. Fig. 4 shows one such neuron.

There are n neurons in the input layer because the dimensionality of each input vector is n. There would be a number of hidden layers with the varying number of neurons as

predefined by the user. The input to a neuron (*h*th neuron) in the hidden layer is the output of the neurons from the previous layer.

The output ($\hat{y}_h^L$) of a neuron h at the hidden layer (Lth hidden layer) is computed as

$$\hat{y}_h^L = \phi\left(\sum_{j=1}^{r} w_{jh}^L \times \hat{y}_h^{L-1}\right) \tag{2}$$

where h represents the hth neuron; L represents the Lth hidden layer, r represents the total neurons at $L-1$ layer; $\hat{y}_h^{L-1}$ is the output of a neuron from $L-1$ layer, and w_{jh}^L is the weight/coefficient connecting jth neuron from layer $L-1$ to h neuron at layer L.

The number of neurons "m" at the output layer is a predefined value that is chosen based on the how many features would be generated from an n-dimensional input vector such that the number of features are downsampled from "n" to "m." Similar to the neurons at hidden layer, the output of a neuron ($\hat{y}_g^o$) at output layer is computed where the outputs $\hat{y}_h^{L-1}$ from the previous layer is fed as input to the neuron at output layer and then the same equation as (2) is used to compute the output.

To generate fewer features from high-dimensional data, the downsampler (deepNN) discussed in this section is used. Initially, the weights connecting neurons at different layers are generated randomly. Then, the weights are updated following the approach described in Section III-C5.

B. Classifier

A deepNN is generated and used to classify the data as either an attack or a nonattack, which can be represented as a function f. The output of f for a given input vector $\mathfrak{X}_i$ is computed as follows:

$$\hat{y}_i = f\left(\hat{Y}_g^i\right) \tag{3}$$

$$= f(g(\mathfrak{X}_i)) \tag{4}$$

that is, the input to the f is $\hat{Y}_g$, which is nothing but output produced by downsampler g for the ith input vector. In summary, the output of downsampler is fed as input to the classifier f. The input layer would consist of a total "m" neuron if the dimensionality of $g(\mathfrak{X}_i)$ is "m." The output layer contains only one neuron to produce an output as either "zero" (i.e., attack) or "one" (i.e., nonattack). There could be one or more hidden layers with varying number of neurons. The computations for the neurons at input, hidden and output layers are accomplished by following (1) and (2).

C. Train/Adapt Weights of Downsampler and Classifier

In our proposed approach there are two deepNNs: 1) downsampler g and 2) classifier f. However, training the two deepNNs simultaneously is not possible because the output of one deepNN is fed as input to the another output, and the target value of g is not known. Hence, the training is accomplished as follows.

1) Generate prediction using randomly generated weights.
2) Compute error/loss.
3) Given the loss, update weights of the classifier.
4) Compute error/loss.
5) Given the loss, update weights of the downsampler.

A detailed description of these steps is provided here.

1) Generate Initial Prediction: Using (4), the class labels $\hat{y}$ for each of the input data sample $\mathfrak{X}$ are predicted. Note that to compute the predictions $\hat{y}$, the values of $\hat{Y}_g$ should be available to f. These values of $\hat{Y}_g$ are computed using (1) as described in Section III-A. At this stage of training, the values of weights for both downsampler and classifier are randomly generated. Let us refer to the output predicted at this step as $\hat{y}_{t=1}$. The summary equation to compute this output can be written as

$$\hat{y}_{t=1} = (f \circ g)(x). \tag{5}$$

2) Compute Error: The classification error of the model is computed using the well-known binary cross-entropy function as shown in the following equation:

$$\mathcal{L}_{t=1} = \frac{1}{q}\sum_{i=1}^{q} \hat{y}_i \log P\left(\hat{y}_i\right) + \left(1-\hat{y}_i\right)\log P\left(1-\hat{y}_i\right) \tag{6}$$

where $P(\hat{y}_i)$ represents the probability of $\hat{y}_i$ being attack for all q samples, $P(1-\hat{y}_i)$ represents the probability of $\hat{y}_i$ being nonattack for all q samples, and "q" represents the total data samples in data set $\mathfrak{D}$.

3) Update Weights of the Classifier: Once the loss is computed using (6), this loss is fed back to the classifier to adapt the weights with an aim to reduce the loss and improve the classification performance of the classifier f following the backpropagation algorithm [31]. To adapt weights connecting the output layer to the hidden layer, the following equation is applied:

$$w_{oh}^{\text{new}} = w_{oh}^{\text{old}} - \eta \times \frac{v_t}{\sqrt{s_t+\epsilon}} \times \frac{\partial \mathcal{L}}{\partial w_{oh}^{\text{old}}} \tag{7}$$

where

$$v_t = \beta_1 \times v_{t-1} - (1-\beta_1) \times \frac{\partial \mathcal{L}}{\partial w_{oh}^{\text{old}}}$$

$$s_t = \beta_2 \times s_{t-1} - (1-\beta_2) \times \frac{\partial \mathcal{L}}{\partial w_{oh}^{\text{old}}}$$

η = learning rate; β_1 and β_2 are hyperparameters should be provided by the user.

Weights connecting the jth neuron from one hidden layer to the ith neuron of another hidden layer or input layer are adapted following these equations:

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} - \eta \times \frac{v_t}{\sqrt{s_t+\epsilon}} \times \delta_j \times \hat{y}_j \tag{8}$$

where

$$v_t = \beta_1 \times v_{t-1} - (1-\beta_1) \times \delta_j \times \hat{y}_j$$

$$s_t = \beta_2 \times s_{t-1} - (1-\beta_2) \times \delta_j \times \hat{y}_j$$

$$\delta_j = \phi'\left(\sum_{i=1}^{r} w_{ij}^L \times \hat{y}_j^{L-1}\right) \times \sum_{k=1}^{c} \delta_k w_{kj}$$

r = total neurons in the previous layer

c = total neurons in the next layer

β_1, β_2 = hyperparameters.

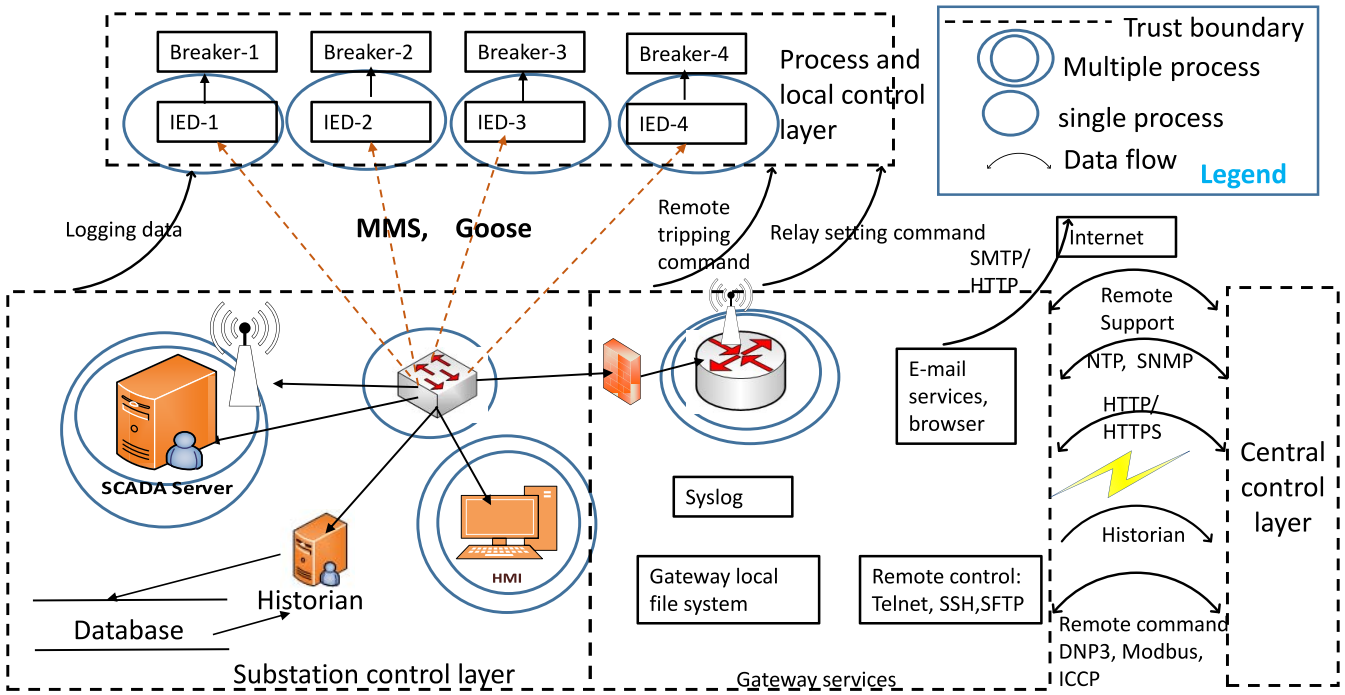

Fig. 5. Trust boundaries of a substation network and corresponding DFD.

Note, the weights of downsampler are not adapted at this step of training. Note that the weights of the downsampler are not adapted at this stage of training.

4) Compute Error/Loss: Once the classifier is trained, the error/loss of the model is computed following the steps described in Section III-C2, where the loss function is a binary cross-entropy.

5) Update Weights of the Downsampler: Section III-C4 features a description of computing error $\mathcal{L}$ of the model, where the classifier's weights have been adapted; however, the weights of the downsampler have not yet been adopted. At this step of the training process, (7) and (8) are adopted to update the weights connecting neurons at the output layer to the hidden layer and neurons at the hidden layer to another previous hidden layer or input layer, respectively. It is worth mentioning that, to update/adjust the weights connecting hidden-layer neurons to output-layer neurons, the error between predicted output and actual/desired output must be known. Because the target/desired output for the downsampler f is not known, the error is computed considering the output generated by the classifier g and the desired output from the training data. This error is thus propagated back to the connections between the output layer and hidden layer of downsampler f to adapt the weights of the respective connections. Equation (7) is used to adapt/update these weights. Once, the weights of the output layer to the hidden layer are adapted, (8) is used to adapt/update the weights connecting hidden layer to input layer. Note that the weights of classifier g are not adapted, whereas the weights of downsampler f are adapted.

Steps 1)–5) are repeated for several epochs until a predefined error is achieved or a maximum number of epochs has been executed.

IV. Experimental Setup and Results

A. Data Set

To validate our proposed approach, we considered a particular segment (domain) of an IIoT network, a substation of the smart grid network as shown in Fig. 5. We considered different communication protocols, such as MMS, GOOSE, DNP3, Modbus, and ICCp, for communication between the substation networks, transmission networks, and control center. Communication protocols are the main attack surface used in internal command flows and interdomain command flows. Attackers can use an attack framework as Fig. 6 shows. Generally, reconnaissance is used to obtain the networks' IP address range, DNS server, mail server, OS versions, and employee/resource-related information. This information is used in the next stage to find open ports and live services, and then the vulnerability catalogue can be used to find their vulnerabilities. Most of the protocols do not provide encryption and authentication, so scanning is not particularly difficult for attackers. In enumeration stages, authentication information is retrieved using tools, such as metasploit and brute-force attack to access close points and further penetrate the network. Then different types of actual exploitations are conducted using function code injections, man-in-the-middle attacks, and modification of the firmware or settings in the relays. DDoS attacks can be delivered to make the resources unavailable and cause further damage, such as creating a communication barrier between the SCADA master and slave units at the time of critical event responses.

The attack vectors of the attack models are derived by the attack framework shown in Fig. 6 and the data flow diagram (DFD) shown in Fig. 5. The attack surface is extremely large, and many of the subsequences of attack models are merged to a set of final attack vectors with sources/entry

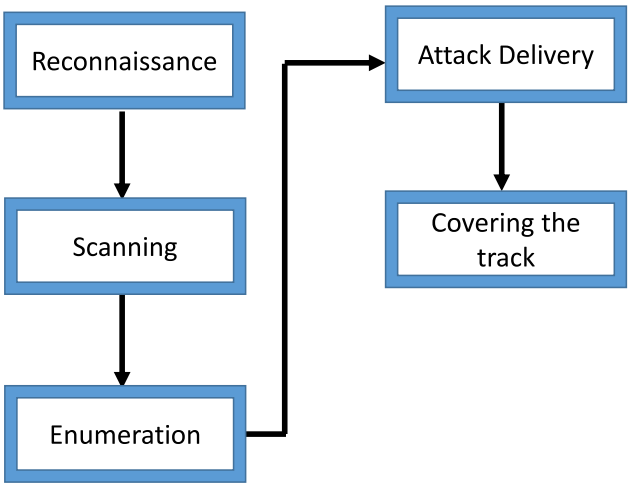

Fig. 6. Attack framework to exploit the vulnerabilities in IIoT networks and violate the security objectives.

points, such as compromised gateways and wireless switches, which end to the exploitation including malicious command injection, change relay settings, change parameter settings, and manipulation of automation logic. We mentioned the data flows related to those attack vectors in Fig. 5 and considered corresponding attack vectors. We have used 36 000 examples of which half are normal operational traffic and half are attack-type traffic.

B. Experimental Setup

To test the efficacy of the proposed model in identifying an SCADA attack, we classified the same data using the following techniques and compared the performances.

1) SVM.
2) Using a deepNN of the same structure that is described in Section III, but without using the same training approach. Instead, the whole deepNN was trained using a backpropagation (Adam) algorithm without differentiating the downsampling and classification.
3) Developing a GAN to generate adversarial attack data and attempting to classify these generated data (a brief description about this model is provided in Section IV-B1).
   a) Introducing 1% noise to the test data set and classifying the data. Noise is generated following equation:

$$\Xi \hookrightarrow \mathcal{N}\left(\mu, \sigma^2\right)$$
$$\text{ND} = \text{AD} + \text{AD} \times \Xi \times \alpha\% \tag{9}$$

   where AD = Actual Data, ND = Noisy Data, and α = percentage of noise which can be 1 to 100, $\mu = 0$, $\sigma = 1$, and $\mathcal{N}$ = Normal Distribution.
   b) Introducing 2% noise into the test data set and classifying the data. Equation (9) is used with $\alpha = 2$ to generate 2% noisy data.
   c) Introducing 5% noise into the test data set and classifying the data. Equation (9) is used with $\alpha = 5$ to generate 5% noisy data.
   d) Introducing 10% noise to the test data set and classifying the data. Equation (9) is used with $\alpha = 10$ to generate 10% noisy data.
   e) Introducing 20% noise to the test data set and classifying the data. Equation (9) is used with $\alpha = 20$ to generate 20% noisy data.

1) Generative Adversarial Network: The concept of GAN was introduced by Goodfellow *et al.* [30]. The aim of GAN is to generate data with characteristics similar to those of the input (actual) data using two different deepNNs, one used as a discriminator and the other used to generate data. A brief description of how GAN works is presented here.

GAN has two parts:

1) discriminator;
2) generator.

Initially, the generator is built by random weight generation, and its output is the same dimension as that of the actual data (i.e., if the actual data have *n*-dimensions, the output of the generator will be *n*-dimensions). Input to the generator is randomly generated values of user-defined dimensions (e.g., the dimension could be chosen as 10). All the data generated by the generator are considered as belonging to the class opposite that of the actual data.

Let us consider the class labels of actual data are "one." Hence, the class labels of the generated data will be "zero" (if the activation function of the output layer of the discriminator is chosen as a sigmoidal function). Once the data from both classes ("one" and "zero," where all the data from class "one" are considered as actual data and all the data from class "zero" are considered as generated data) are available, the discriminator classifies these data. After classification the error is computed following (6) and this loss/error is fed back to the discriminator to adapt its weight such that the overall classification loss is reduced. Once the discriminator can classify the data accurately in the next step, the generator tries to generate data such that it can deceive the discriminator (i.e., for the generated data the discriminator will classify them as "one" opposed to "zero"). The weight of the generator is updated following a backpropagation algorithm such that it can deceive the discriminator successfully. These processes continue until the generator can generate data such that the discriminator classifies them as "one". A detailed description of GAN can be found in [30].

2) Parameter Setup: Table II lists the parameters used to build the respective models in the experiments. We normalized the data in the range of [0, 1] such that the respective models could classify the data accurately.

C. Performance Metric

We used the cross-entropy loss function and accuracy metrices to evaluate the proposed model and the existing models. Cross-entropy function is defined in (6). The accuracy metric is defined as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{10}$$

where TP = True positive, TN = True negative, FP = False positive, and FN = False negative.

TABLE II
PARAMETERS USED TO BUILD AND TRAIN THE RESPECTIVE MODELS

Method	Parameters					
	Submodel	Input node	Total hidden layers	Nodes in each hidden layer	Activation function for each layer	Epochs
Proposed approach	Downsampler	114	5	100,70,40,15, 4	tanh, tanh, tanh, tanh, Relu	20
	Classifier	114	3	100, 50, 1	relu, relu, sigmoid	
deepNN	-	114	8	100,70,40,15, 4, 100, 50, 1	tanh, tanh, tanh, tanh, Relu, relu, relu, sigmoid	20
GAN	Generator	10	5	100,70,40, 15, 114	tanh, tanh, tanh, tanh, Relu	500
	Discriminator	114	3	100, 50, 1	relu, relu, sigmoid	

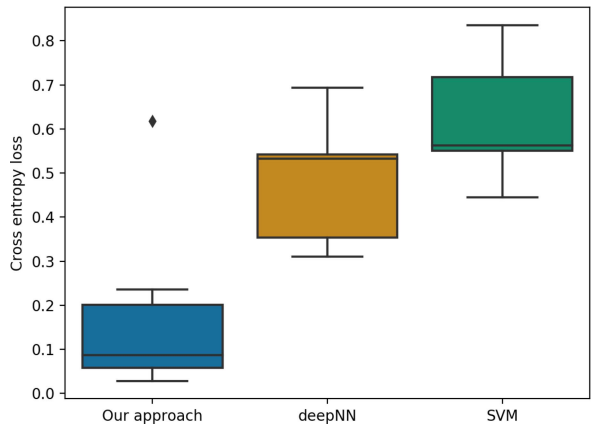

Fig. 7. Loss (binary cross-entropy) comparison among the three classifiers for many runs.

TABLE III
SCADA ATTACK/NONATTACK IDENTIFICATION PERFORMANCE IN TERMS OF BINARY CROSS-ENTROPY

Noise label	Our approach	deepNN	SVM (Sigmoid)
No noise	0.178533217	0.475562284	0.639425764
1% noise added	0.178628576	0.495559944	0.78765588
2% noise added	0.178749861	0.575559865	0.814599189
5% noise added	0.17934605	0.775574136	0.892406986
10% noise added	0.180868228	1.175614277	1.01780127
20% noise added	0.186306753	1.975718947	1.71675817
50% noise added	0.215987149	2.476928337	2.872630377

D. Results and Discussion

Table III shows the performance of the proposed approach as well as a few other techniques on classifying attack/nonattack incidents of SCADA. As shown in the table, the performance of the proposed approach is always better in terms of binary cross-entropy (it is also referred to as log-loss) compared to that of other classifiers, namely, deepNN and SVM. Cross-entropy scores close to zero reveal a better performance compared to a high score. Our approach achieved a cross-entropy score of 0.178533217, which is much lower than that of deepNN (for which this score is 0.47556228). Hence, our approach has $[(0.475562284 - 0.178533217) \div 0.178533217 \times 100]166.37\%$ higher performance than deepNN in classifying attack/nonattack incidents. Similarly, our approach performs 258.16% better than SVM in classifying attack/nonattack incidents when considering that the loss function is a binary cross-entropy. These performances are achieved when the data samples do not have any noise (i.e., actual data collected from the field). Fig. 7 summarizes the loss score variations for different runs (where the respective models are trained using randomly selected data from both attack and nonattack data sets). As shown in the figure, the proposed approach outperforms the other models in all the runs (executions). Furthermore, the proposed model is stable in that, with a varying training-test data set, the variation in performances is between 0.08 to 0.2 (average = 0.178533217), while for deepNN the performance varies between 0.35 to 0.53 (average = 0.475562284). This result evidently suggests the superiority of the proposed model over the other models reported in this study.

Table III also reports classification performances for data set with added noise at different levels ranging from 1% to 50% noise. As expected with added noise, the performance of each classifier degrades proportionally. However, the performance of our approach still remains the best among the other classifiers. One such performance proves the robustness of our approach empirically. We conjecture that the reason for such an encouraging performance is the downsampler, which acts as a filter to reduce the effect of noise by encoding high-dimensional inputs to low-dimensional data vectors. Hence an outstanding performance is achieved for any inputs with added noise or adversarial effect. Eventually, the architecture of the deepNN used is the same as that of our approach (if we consider the overall architecture of the downsampler and classifier); in deepNN training, weights are updated simultaneously from the input layer through the output layer, and thus the noise is propagated from the input to the output layer. Therefore, the performance of deepNN decays rapidly with added noise to the inputs. In summary, the traditional deepNN fails to classify attack/nonattack incidents if the adversarial effect is present at the input end. For the SVM, we identify that the performance of classifying actual data (without adding any noise to the input) is not particularly satisfactory. This can be attributed to the choice of optimal parameters. Because SVN performance is much worse than that of deepNN and our approach, we discuss our results for deepNN and our approach in the remainder of this section.

Fig. 8 summarizes the cross-entropy loss score variations with the variance of the noise level in the input space. As shown in the figure, the proposed method's performance varies with a standard deviation of 0.2 (approx.), while similar variation is seen for the deepNN. However, the average cross-entropy score of the proposed method is 0.18 (approx.) compared with that for deepNN at 0.45 (approx.). Furthermore, the figure shows that the performance score does not vary when noises are added to the input space for the proposed approach; however, the variation of performance score is random for the deepNN. Hence, from these experimental results, we see that the proposed method is stable and robust to the input space under the condition of the adversarial effect.

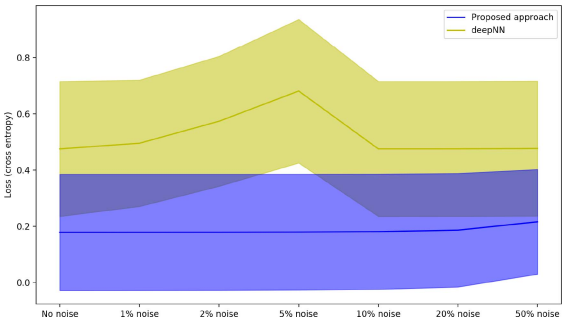

Fig. 8. Loss (binary cross-entropy) confidence comparison between our approach and deepNN for different level of noises.

TABLE IV
SCADA ATTACK IDENTIFICATION ACCURACY IN TERMS OF ACCURACY

Noise label	Our approach	deepNN	SVM (Sigmoid)
No noise	0.954219529	0.953536936	0.778820959
1% noise added	0.954025081	0.951540912	0.941955409
2% noise added	0.953858427	0.949556784	0.9434154699
5% noise added	0.953128287	0.942548857	0.931627547
10% noise added	0.951771174	0.934484569	0.92305323
20% noise added	0.948374459	0.914352816	0.90831820412
50% noise added	0.929184496	0.8593052887	0.8457829875

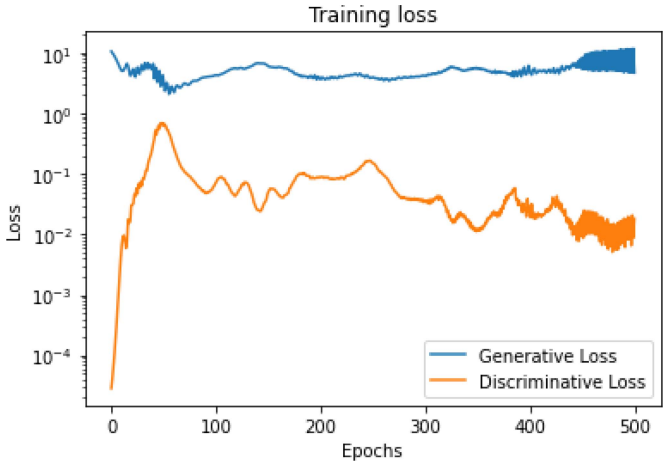

Fig. 9. Generator and discriminator loss curve during training GAN.

TABLE V
SCADA ATTACK IDENTIFICATION USING GAN GENERATED DATA

Method	Accuracy	Binary cross-entropy
Our approach	0.955536936	0.475562284
deepNN	0.9071428571	0.82458985

Table IV reports the performance of the classifiers in terms of accuracy. Our approach performs slightly better than that of deepNN (95.422% versus 95.35%) when no noise is added to the input.

It should be noted here that the performance difference between the proposed approach and deepNN in terms of cross-entropy is extremely high, while in terms of accuracy it is not that high. The reason is that when computing cross-entropy, the output of the classifier is considered as is, while in computing the accuracy, the output is first transformed following a threshold θ (in our experiment the value of θ was chosen 0.5) as shown in the following equation:

$$y = \begin{cases} 0, & \text{if } y \geq \theta \\ 1, & \text{otherwise.} \end{cases} \tag{11}$$

Because the actual output generated by the deepNN is further quantized using the above equation, the error terms for the overall accuracy in percentage are enhanced (e.g., if the actual output is 0.51, according to (11), the predicted class label is 1; note that, for the actual output the error is $1 - 0.51 = 0.49$ if the desired output is "1"; however, owing to the quantization the error is 0).

As such, the classification performances in terms of accuracy (in percentage) are extremely close for the proposed approach and the deepNN. However, with added noise at different levels, the performance of the proposed approach does not decay significantly (from 95.42% to 92.92% with a level of added noise from 1% to 50% respectively), whereas that of deepNN decays sharply from 9.5.35% to 85.93%. As mentioned earlier, the proposed approach sufficiently robust to deal with a data set with added adversarial effect (in terms of noises).

To test the efficacy of the proposed model to deal with a true adversarial data set, in the experiment an augmented data set is generated using a GAN. A data set of actual attack data with 300 samples is fed to the GAN to generate an augmented data set of 30 000 samples. The GAN is trained following the procedure described in Section IV-B1. Fig. 9 shows the convergence of generator and discriminator losses for the 500 epochs during training the GAN. As per the GAN characteristics, the class labels of the generated samples would be "attack" incidents. While attempting to classify the augmented data generated by GAN, the proposed approach can classify with 95.55% accuracy and a binary cross-entropy loss score of 0.475562284. In contrast, deepNN achieves a classification accuracy of 90.71% and binary cross-entropy loss score of 0.82458985. Evidently, the proposed approach can deal with adversarial perturbations in input space, and the classification performance is highly encouraging. Interestingly, deepNN failed to classify the GAN-generated samples with a low binary cross-entropy score.

To test how the proposed model performs with variance in the training data set, we varied the size of the training data from 600 to 1200, while the rest of the data were used as test data. It is worth mentioning that in this case, the total data samples considered are 72355, of which 52393 belong to the attack class and the rest of the data belong to the nonattack class. One such data distribution can be referred to as an imbalanced data, where the ratio of attack versus nonattack samples is 72.4% versus 27.6%. Fig. 10 compares the performances of our proposed approach with deepNN and SVM (Sigmoid) to classify attack/nonattack data. As shown in the figure, our approach performs with consistent accuracy compared with the other approaches. The performance of deepNN differs not only with the variation of training data size but also with the variation of training data samples. It is worth mentioning here that for each individual execution, the training

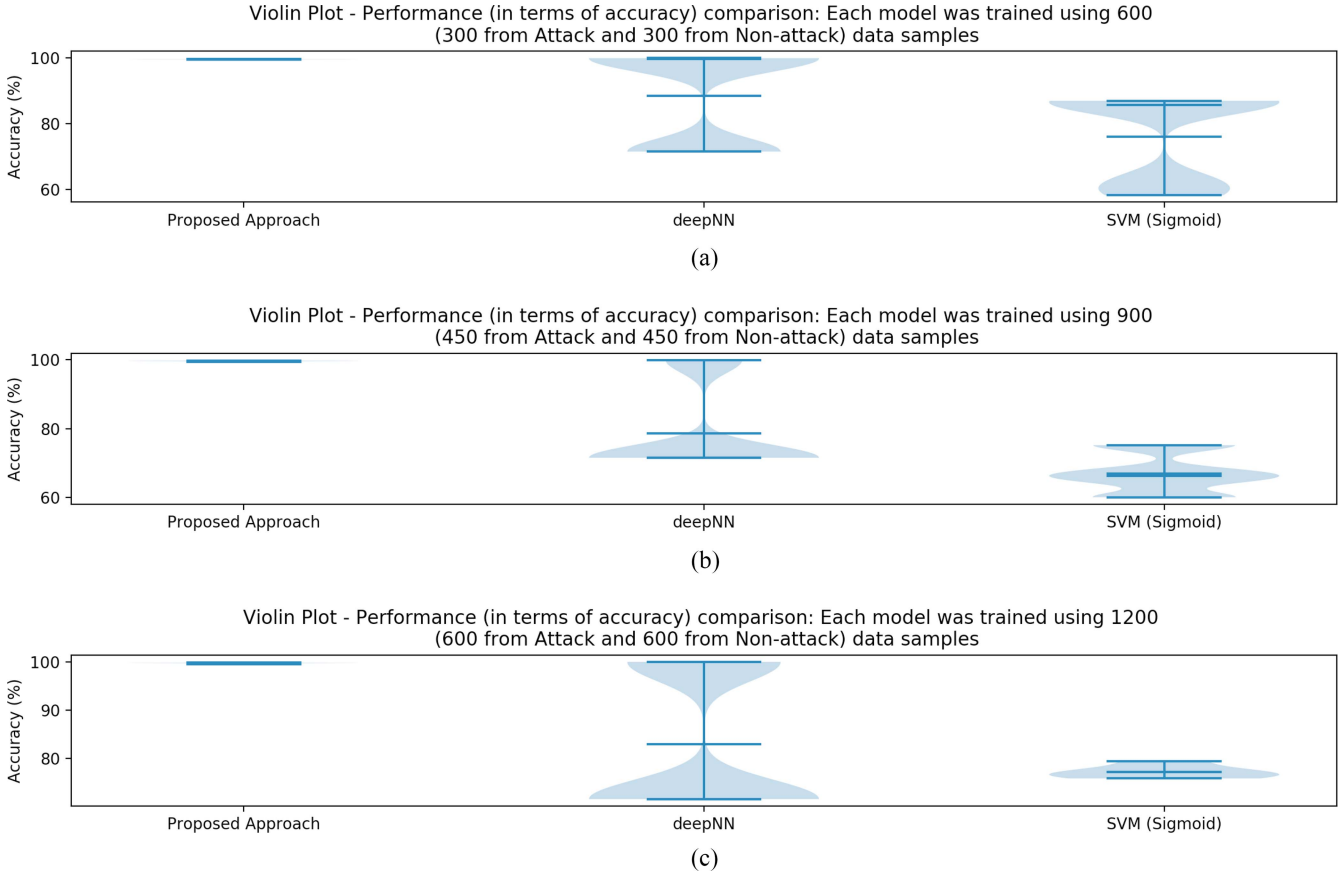

Fig. 10. (a) Classification performance of test data while each of the models was trained using 600 data samples only. (b) Classification performance of test data while each of the models was trained using 900 data samples only. (c) Classification performance of test data while each of the models was trained using 1200 data samples only.

data samples are chosen randomly. A similar variation in classification accuracy is seen for SVM (Sigmoid). Interestingly, our approach can classify the SCADA data with consistent accuracy. Furthermore, the training time using our approach required 2.58 s (for 600 training data samples), whereas for the same training data deepNN spent 5.93 s. We found a similar trend when increasing the training data size. SVM took very little time (i.e., 0.05 s) for the same training data; however, the performance of SVM is relatively poor compared with deepNN and our proposed approach. We executed all experiments using a Python program on a Macbook Pro PC with i7 microprocessor and 16 GB RAM. These experimental results evidently prove the superiority and robustness of our approach among the other approaches considered in this study.

V. Conclusion

In this article, we propose a downsampler encoder-based cooperative data generator, which is trained using an adaptive algorithm to ensure a better capture of actual distribution of attack models for the large IIoT attack surface. The proposed downsampler-based data generator is alternatively updated and verified during training using a deepNN discriminator. The discriminator guarantees the performance of the generator against input sets with a high noise level at the time of training and testing. Experimental evaluation on a real IIoT testbed demonstrates that the proposed approach is far more robust in an adversarial environment than the conventional deepNN technique. We also compared the proposed approach with an ML-based technique, such as SVM. Our proposed approach outperforms all of these existing approaches in terms of performance robustness against an adversarial/noisy environment. In the future, we plan to develop a multi-stage downsampler's discriminator approach for an adversarial environment.

References

[1] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial Internet of Things (IIoT): An analysis framework," *Comput. Ind.*, vol. 101, pp. 1–12, Oct. 2018.

[2] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.

[3] M. M. Hassan, A. Gumaei, S. Huda, and A. Almogren, "Increasing the trustworthiness in the industrial IoT networks through a reliable cyberattack detection model," *IEEE Trans. Ind. Informat.*, vol. 16, no. 9, pp. 6154–6162, Sep. 2020.

[4] M. E. Khoda, T. Imam, J. Kamruzzaman, I. Gondal, and A. Rahman, "Robust malware defense in industrial IoT applications using machine learning with selective adversarial samples," *IEEE Trans. Ind. Appl.*, vol. 56, no. 4, pp. 4415–4424, Jul./Aug. 2020.

[5] M. S. Mahmoud, M. M. Hamdan, and U. A. Baroudi, "Modeling and control of cyber-physical systems subject to cyber attacks: A survey of recent advances and challenges," *Neurocomputing*, vol. 338, pp. 101–115, Apr. 2019.

[6] S. Plaga, N. Wiedermann, S. D. Anton, S. Tatschner, H. Schotten, and T. Newe, "Securing future decentralised industrial IoT infrastructures: Challenges and free open source solutions," *Future Gener. Comput. Syst.*, vol. 93, pp. 596–608, Apr. 2019.
[7] S. Sharmeen, S. Huda, J. Abawajy, and M. M. Hassan, "An adaptive framework against android privilege escalation threats using deep learning and semi-supervised approaches," *Appl. Soft Comput.*, vol. 89, Apr. 2020, Art. no. 106089.
[8] S. Huda, J. Abawajy, B. Al-Rubaie, L. Pan, and M. M. Hassan, "Automatic extraction and integration of behavioural indicators of malware for protection of cyber–physical networks," *Future Gener. Comput. Syst.*, vol. 101, pp. 1247–1258, Dec. 2019.
[9] Y. Leng, X. Xu, and G. Qi, "Combining active learning and semi-supervised learning to construct SVM classifier," *Knowl. Based Syst.*, vol. 44, pp. 121–131, May 2013.
[10] S. Gupta and A. Dileep, "Relevance feedback based online learning model for resource bottleneck prediction in cloud servers," *Neurocomputing*, vol. 402, pp. 307–322, Aug. 2020.
[11] H. Guo, A. Zhang, and W. Wang, "An accelerator for online SVM based on the fixed-size KKT window," *Eng. Appl. Artif. Intell.*, vol. 92, Jun. 2020, Art. no. 103637.
[12] L. Zhu, K. Ikeda, S. Pang, T. Ban, and A. Sarrafzadeh, "Merging weighted SVMs for parallel incremental learning," *Neural Netw.*, vol. 100, pp. 25–38, Apr. 2018.
[13] S. Huda, S. Miah, J. Yearwood, S. Alyahya, H. Al-Dossari, and R. Doss, "A malicious threat detection model for cloud assisted Internet of Things (CoT) based industrial control system (ICS) networks using deep belief network," *J. Parallel Distrib. Comput.*, vol. 120, pp. 23–31, Oct. 2018.
[14] S. Sharmeen, Y. A. Ahmed, S. Huda, B. Ş. Koçer, and M. M. Hassan, "Avoiding future digital extortion through robust protection against ransomware threats using deep learning based adaptive approaches," *IEEE Access*, vol. 8, pp. 24522–24534, 2020.
[15] L. A. Passos and J. P. Papa, "A metaheuristic-driven approach to fine-tune deep Boltzmann machines," *Appl. Soft Comput.*, to be published.
[16] S. Huda, J. Yearwood, M. M. Hassan, and A. Almogren, "Securing the operations in SCADA-IoT platform based industrial control system using ensemble of deep belief networks," *Appl. Soft Comput.*, vol. 71, pp. 66–77, Oct. 2018.
[17] J.-H. Choi and J.-S. Lee, "EmbraceNet: A robust deep learning architecture for multimodal classification," *Inf. Fusion*, vol. 51, pp. 259–270, Nov. 2019.
[18] Y. Bai and M. Tang, "Robust visual tracking via augmented kernel SVM," *Image Vis. Comput.*, vol. 32, no. 8, pp. 465–475, 2014.
[19] Q.-Q. Tao, S. Zhan, X.-H. Li, and T. Kurihara, "Robust face detection using local CNN and SVM based on kernel combination," *Neurocomputing*, vol. 211, pp. 98–105, Oct. 2016.
[20] J. Li, Y. Liu, T. Chen, Z. Xiao, Z. Li, and J. Wang, "Adversarial attacks and defenses on cyber–physical systems: A survey," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5103–5115, Jun. 2020.
[21] P. Vidnerová and R. Neruda, "Vulnerability of classifiers to evolutionary generated adversarial examples," *Neural Netw.*, vol. 127, pp. 168–181, Jul. 2020.
[22] T. Huang, Y. Chen, B. Yao, B. Yang, X. Wang, and Y. Li, "Adversarial attacks on deep-learning-based radar range profile target recognition," *Inf. Sci.*, vol. 531, pp. 159–176, Aug. 2020.
[23] Q. Ya-guan *et al.*, "Spot evasion attacks: Adversarial examples for license plate recognition systems with convolutional neural networks," *Comput. Security*, vol. 95, Aug. 2020, Art. no. 101826.
[24] I. Oregi, J. Del Ser, A. Pérez, and J. A. Lozano, "Robust image classification against adversarial attacks using elastic similarity measures between edge count sequences," *Neural Netw.*, vol. 128, pp. 61–72, Aug. 2020.
[25] Z. Wang, "Deep learning-based intrusion detection with adversaries," *IEEE Access*, vol. 6, pp. 38367–38384, 2018.
[26] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," 2014. [Online]. Available: arXiv:1412.5068.
[27] T. Miyato, S.-I. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: A regularization method for supervised and semi-supervised learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 8, pp. 1979–1993, Aug. 2019.
[28] H. Jiang, H. Nai, Y. Jiang, W. Du, J. Yang, and L. Wu, "An adversarial examples identification method for time series in Internet of Things system," *IEEE Internet Things J.*, early access, Jun. 30, 2020, doi: 10.1109/JIOT.2020.3005688.
[29] H. Qiu, Q. Zheng, T. Zhang, M. Qiu, G. Memmi, and J. Lu, "Towards secure and efficient deep learning inference in dependable IoT systems," *IEEE Internet Things J.*, early access, Jun. 23, 2020, doi: 10.1109/JIOT.2020.3004498.
[30] I. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2014, pp. 2672–2680.
[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: arXiv:1412.6980.

Mohammad Mehedi Hassan (Senior Member, IEEE) received the Ph.D. degree in computer engineering from Kyung Hee University, Seoul, South Korea, in February 2011.

He is currently an Associate Professor with the Information Systems Department, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. He has authored and coauthored around over 180 publications, including refereed IEEE/ACM/Springer/Elsevier journals, conference papers, books, and book chapters. His research interests include edge/cloud computing, Internet of Things, cyber security, deep learning, artificial intelligence, body sensor network, 5G network, and social network.

Dr. Hassan is a recipient of a number of awards, including the Best Journal Paper Award from the IEEE SYSTEMS JOURNAL in 2018, the Best Paper Award from CloudComp in 2014 Conference, and the Excellence in Research Award from King Saud University (two times in a row) in 2015 and 2016.

Md. Rafiul Hassan (Member, IEEE) received the Ph.D. degree in computer science and software engineering from the University of Melbourne, Melbourne, VIC, Australia, in 2007.

He was a Research Fellow with the University of Melbourne from 2008 to 2010. He is currently an Associate Professor with the College of Arts and Sciences, University of Maine at Presque Isle, Presque Isle, ME, USA. He holds two U.S. patents and has conducted research in the areas of data science, machine learning, big data, and intrusion detection.

Shamsul Huda received the Ph.D. degree in computer science from the Center for Informatics and Applied Optimization, Federation University, Ballarat, VIC, Australia, in 2010.

He is currently a Lecturer with the School of Information Technology, Deakin University, Melbourne, VIC, Australia. He is a Certified Information System Security Professional by the International Information System Security Certification Consortium, $(ISC)^2$. He is also a Member of the Cyber Security Research and Innovation Centre, Deakin University. He has authored or coauthored more than 60 journal and conference papers in well-reputed journals including IEEE transactions. His main research interests include information security, cyber–physical systems, computational intelligence, and machine learning.

Victor Hugo C. de Albuquerque (Senior Member, IEEE) received the bachelor's degree in mechatronics technology from the Federal Center of Technological Education of Ceará, Fortaleza, Brazil, in 2006, the M.Sc. degree in teleinformatics engineering from the Federal University of Ceará, Fortaleza, in 2007, and the Ph.D. degree in mechanical engineering with emphasis on materials from the Federal University of Paraíba, João Pessoa, Brazil, in 2010.

He is a Full Professor and a Senior Researcher with the University of Fortaleza, Fortaleza, where he is the Data Science Director with the Superintendency for Research and Public Safety Strategy of Ceará State. He leads the Graduate Program with Applied Informatics and Electronics and Health Research Group. He mainly researches IoT, machine/deep learning, pattern recognition, and robotics.