Deep-Learning-Based Network Intrusion Detection for SCADA Systems

Huan Yang*, Liang Cheng†, and Mooi Choo Chuah‡
Department of Computer Science and Engineering, Lehigh University, Bethlehem, Pennsylvania 18015
E-mail: *huy213@lehigh.edu, †cheng@cse.lehigh.edu, ‡chuah@cse.lehigh.edu

***Abstract*—Supervisory Control and Data Acquisition (SCADA) networks are widely deployed in modern industrial control systems (ICSs) such as energy-delivery systems. As an increasing number of field devices and computing nodes get interconnected, network-based cyber attacks have become major cyber threats to ICS network infrastructure. Field devices and computing nodes in ICSs are subjected to both conventional network attacks and specialized attacks purposely crafted for SCADA network protocols. In this paper, we propose a deep-learning-based network intrusion detection system for SCADA networks to protect ICSs from both conventional and SCADA specific network-based attacks. Instead of relying on hand-crafted features for individual network packets or flows, our proposed approach employs a convolutional neural network (CNN) to characterize salient temporal patterns of SCADA traffic and identify time windows where network attacks are present. In addition, we design a re-training scheme to handle previously unseen network attack instances, enabling SCADA system operators to extend our neural network models with site-specific network attack traces. Our results using realistic SCADA traffic data sets show that the proposed deep-learning-based approach is well-suited for network intrusion detection in SCADA systems, achieving high detection accuracy and providing the capability to handle newly emerged threats.**

***Index Terms*—Network intrusion detection system, deep learning, convolutional neural network, SCADA system security, cyber attack detection, attacks on DNP3 protocol.**

I. Introduction

Monitoring and controlling critical infrastructure of industrial control system (ICS), Supervisory Control and Data Acquisition (SCADA) network interconnects field devices, engineering workstations, human-machine interfaces (HMIs), and various servers to ensure reliable system control and operation. As interconnectivity among SCADA hosts continues to increase, *network-based cyber attacks* (i.e., cyber attacks that exploit SCADA network vulnerabilities to realize their malignant purposes) have become one of the primary cyber threats to SCADA systems. As SCADA networks utilize both widely-used network protocols from the IT domain and specialized SCADA network protocols, SCADA hosts can become the target of both *conventional network-based attacks* (e.g., a malware targeting Windows-based engineering workstations) and *specialized attacks on SCADA protocols* (e.g., [1], [2]). In recently reported cyber-security incidents (e.g., [3], [4]), it is observed that network-based cyber attacks are a major component of sophisticated, multiple-stage attacks on critical ICS infrastructure. However, intrusion detection systems (IDSs) for IT networks cannot be directly ported into SCADA networks because traffic characteristics of SCADA hosts are shown to be distinctly different from hosts in corporate/campus networks [5]. To protect modern ICSs, it is therefore vital to design SCADA network intrusion detection solution that takes into account both operation needs and distinct traffic characteristics of SCADA systems.

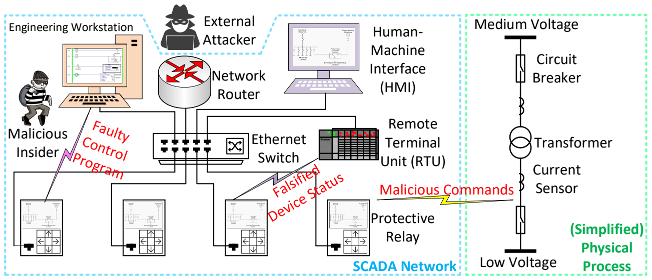

Fig. 1. Network-based attacks on SCADA system.

Real-world attacks on SCADA systems tend to follow a step-by-step method to compromise critical system devices [3], [6], [7], and network-based attacks may be launched at different steps to achieve malicious goals. Take the SCADA system in Fig. 1 as an example. Suppose that the ultimate goal of attackers is to induce severe damages to the physical process monitored and controlled by the SCADA system. If the attackers launch the Stuxnet attack [3], the Windows-based engineering workstation will first be compromised. This step can be accomplished by a malicious insider with a flash drive containing the Stuxnet malware. Then, the attackers can download malicious or faulty control programs to field devices such as protective relays. The compromised field devices may issue malicious network commands (e.g., tripping circuit breakers when no faults occur) or generate falsified device status reports to mask the execution of malicious commands. In the CrashOverride attack [4], [8], knowledge on SCADA communication protocols is exploited, and external attackers can even directly control system equipment without relying on the software within the engineering workstation. As shown in [4], [8]–[10], emerging cyber attacks on SCADA systems tend to be composed of multiple "primitive" attacks, many of which are network-based attacks. For instance, traces of both conventional attacks such as passive reconnaissance and active service/host discovery, and specialized attacks on SCADA protocols (e.g., falsified control commands), are found in the CrashOverride attack. Hence, network intrusion detection solution for SCADA systems must be able to combat both conventional as well as specialized network attacks.

In this paper, we propose a deep-learning-based network intrusion detection system (IDS) for SCADA networks, detecting network-based cyber attack primitives. The contributions of our work are as follows:

- *Deep-learning-based IDS for SCADA networks.* Instead of relying on hand-crafted features of individual packets, our proposed approach employs a convolutional neural network (CNN) to characterize salient temporal patterns of network behaviors of SCADA hosts. Leveraging traffic monitoring capabilities of existing SCADA networking devices, our approach inspects network traffic without interrupting SCADA system operation, which significantly lowers the barrier to deploying our solution.
- *Detection of SCADA network attack primitives.* Detecting both conventional network attacks and those purposely crafted for SCADA network protocols, our proposed approach will facilitate the detection of sophisticated, real-world attacks that launch different network attack primitives at different stages.
- *Capability to adapt to different SCADA environments.* Our solution provides a re-training scheme, enabling the operators to refine our neural network models with site-specific network attack traces and facilitating the rapid adaptation of these models to various SCADA environments that are subjected to different cyber attacks.
- *Evaluation on realistic data sets*. We design and evaluate our proposed network IDS using traffic data collected from different energy-delivery system test beds with real field devices running DNP3 protocol. Our results show that our approach achieves high detection accuracy with few false positives, making it well-suited for network intrusion detection in SCADA systems.

II. Related Work

A. Network-Based Cyber Attacks on SCADA Systems

In recently reported cyber-security incidents [3], [4], [8], [11], it is found that many sophisticated cyber attacks exploit certain network attack primitives to penetrate from corporate networks into SCADA networks, determine roles of different SCADA hosts, and cause damages to physical systems. *Primitive network-based cyber attacks* can be classified into two categories, i.e., conventional attacks exploiting vulnerabilities of IT network protocols [9] and specialized attacks on SCADA network protocols. Taxonomies and analyses of attacks on SCADA network protocols can be found in surveys on ICS security, such as [10], [12]. To detect real-world attacks, a SCADA network intrusion detection system must take into account both conventional and specialized attacks. In this work, we focus on SCADA systems running DNP3 protocol [13], which is widely adopted in SCADA systems across the world.

B. Intrusion Detection for SCADA Systems

SCADA hosts exhibit traffic characteristics that are significantly different from hosts in corporate/campus networks [5]. Therefore, intrusion detection systems for IT networks do not work well in SCADA network environments. Recently, a survey on intrusion detection solutions for industrial control systems is presented in [14]. It is found that many existing solutions utilize widely-used network security mechanisms, such as whitelisting, for SCADA network intrusion detection. However, such approaches are not effective for purposely crafted attacks on SCADA network protocols because legitimate SCADA network packets and attack packets can share the same application-layer header fields (e.g., [12]). To detect specialized attacks, application-layer information needs to be analyzed in detail to distinguish malicious packets from benign ones [15]. However, a wide variety of SCADA-specific attacks reviewed in [9], [10] cannot be detected by the approach proposed in [15] because of the lack of a mature way of modeling network behaviors of SCADA hosts. In [16], machine-learning-based algorithms are proposed to model network behaviors of SCADA hosts. However, the features selected in [16] cannot comprehensively model SCADA host behaviors (e.g., a robust model for temporal network behaviors is yet to be established) and thus many attacks reported in [10] are not considered. In this paper, we analyze network packets up to their application-layer headers and comprehensively model the network behaviors of SCADA hosts using a convolutional neural network (CNN).

III. System Design Overview

A. Threat Model

In our network IDS solution for SCADA systems, we assume that an attacker is capable of compromising certain SCADA hosts or slipping in a new device to launch network-based cyber attacks. For instance, in sophisticated cyber attacks launched by external attackers (e.g., [3], [4]), engineering workstations and field devices in SCADA systems are compromised. In addition to passive eavesdropping, some of the compromised hosts are exploited to issue malicious control commands. As another example, a malicious insider may either directly install malware on SCADA hosts [2], [17] or plant a new device [18] to launch attacks. Once a SCADA host is compromised (or a new device is planted), it may be exploited to send malicious attack packets with or without simultaneously generating normal/regular SCADA network traffic. However, this host will probably not use any network protocols that have not been previously observed on the SCADA network. Otherwise, it will easily be flagged or blocked by existing ICS intrusion detection tools [14].

We also assume that a malicious SCADA host must generate network packets at a certain attack stage. This is because our proposed detection algorithm identifies malicious network activities by analyzing packets generated by all the SCADA hosts. A malicious host can, for example, issue malicious control commands to cause system failures, mask impacts of malicious control actions by sending manipulated device status updates, and learn about the functions of other hosts by reading their detailed configurations [12]. However, if a cyber attack does not generate any network packet revealing its malignant purposes (e.g., [19]), operators must resort to other intrusion detection solutions (e.g., host-monitoring-based methods [20], [21]) to protect the SCADA system.

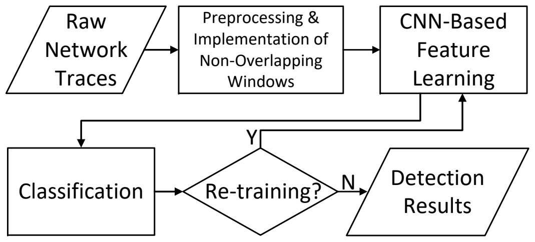
Raw Network Traces
Preprocessing & Implementation of Non-Overlapping Windows
CNN-Based Feature Learning
Classification
Re-training?
Y
N
Detection Results

Fig. 2. High-level architecture of the proposed network IDS solution.

Finally, we assume that network packets exchanged inside a SCADA site (e.g., a power substation) are not encrypted. This is the case because a SCADA site is typically secured physically and legacy field devices that do not support encryption/decryption are still widely in use.

B. System Design

The architecture of our proposed network IDS is depicted in Fig. 2. The input to our tool is raw SCADA network traces, i.e., network packets generated by all the SCADA hosts. In real-world SCADA systems, these packets can be collected by leveraging the traffic monitoring capabilities of network switches (e.g., the port mirroring feature [22] of Cisco switches). By properly configuring networking devices to duplicate all the network traffic to a monitoring port, our proposed approach will not interrupt normal system operation. The duplicated traffic is forwarded to a server, where our proposed detection algorithm is deployed.

Our algorithm first inspects each packet up to its application-layer header and extracts useful information for feature learning. Application-layer payload data are not analyzed by our algorithm, so an attack that manipulates process data will not be detected unless the corresponding application-layer header is also modified. The preprocessing stage extracts a 25-tuple from each packet and implements non-overlapping detection windows for the feature learning stage. Feature vectors of all packets contained in each detection window are passed to a convolutional neural network (CNN) for feature learning. At lower layers of the CNN stage, local characteristics of the tuple sequence are obtained. Then, at its higher layers, high-level salient patterns are extracted. The learned features are then fed to a classification stage to generate detection outputs. We attach a fully-connected softmax output layer to the feature-learning CNN for classification. In addition, outputs of the classification stage are also exploited as indicator for re-training. When a newly emerged attack is encountered, our algorithm will signal the SCADA system operator to inspect and label the corresponding time window. When the number of new attack instances is sufficiently large, a new class is added to the classification stage and the entire neural network is re-trained.

In essence, our detection method analyzes the temporal network behavior patterns of SCADA hosts and identifies time windows containing abnormal patterns. At the output end, our detection algorithm assigns a label to each detection window (e.g., “normal operation”). By inspecting detection windows containing anomalies, SCADA system operator can easily pinpoint the root cause and take proper measures to mitigate the attack impacts.

IV. Deep-Learning-Based IDS for SCADA Networks

A. Data Preprocessing

An operational SCADA network runs a set of network protocols for network connectivity and SCADA system operations. The data preprocessing stage extracts information from different types of network packets as follows:

- *Layer 2 packets.* For each Layer 2 packet, source and destination MAC addresses, as well as the EtherType value, are extracted. Size of the Layer-2 payload, excluding the 4-byte checksum, is also extracted.
- *Layer 3 packets.* Generally, IP and ICMP packets are present on SCADA networks. In addition to the 4 fields extracted from Layer 2, we extract source and destination IPs, protocol type, and the packet length field from IP packets. If the packet is an ICMP packet, the type and subtype fields are also obtained. If the packet is not an ICMP packet, both fields are set to 256.
- *TCP/UDP packets.* In addition to the 10 fields from Layers 2 and 3, we extract source and destination ports, TCP flags, sequence number, and acknowledgment number if a TCP packet is encountered. Size of TCP packet payload, excluding the TCP header, is also calculated and recorded. If the packet is a UDP packet, we extract source and destination port numbers. Additionally, UDP payload size excluding the UDP header is also computed. For a UDP packet, the TCP flags, sequence number, and acknowledgment number, are all set to 0.
- *DNP3 packets.* DNP3 packets can use either TCP or UDP. In addition to the 16 fields extracted from a DNP3 packet up to its TCP/UDP layer, we extract information from the DNP3 data-link, pseudo-transport, and application layers [13]. At the DNP3 data-link layer, we extract DNP3 source and destination addresses, the control byte value, as well as the length byte value. The one-byte pseudo-transport layer header is also extracted. At the DNP3 application layer, application control byte, function code byte, and internal indication field values are extracted.

Therefore, 24 fields are filled out for each packet at the preprocessing stage. In addition, *packet inter-arrival time*, i.e., the time interval between the arrival of the current packet and the arrival of its preceding packet is logged. For the first packet seen on the network, its inter-arrival time is set to 0.

For each packet, the output of the preprocessing stage is a 25-tuple. To capture temporal behavior patterns of SCADA hosts, we implement non-overlapping sliding windows on the sequence of SCADA packets collected from the monitoring port. The size of the detection window, in terms of the number of packets it contains, is denoted by r. These non-overlapping detection windows partition the multivariate sequence of packet feature vectors into small snippets. Each *input instance* to the feature learning and classification stages is a sequence of r 25-tuples.

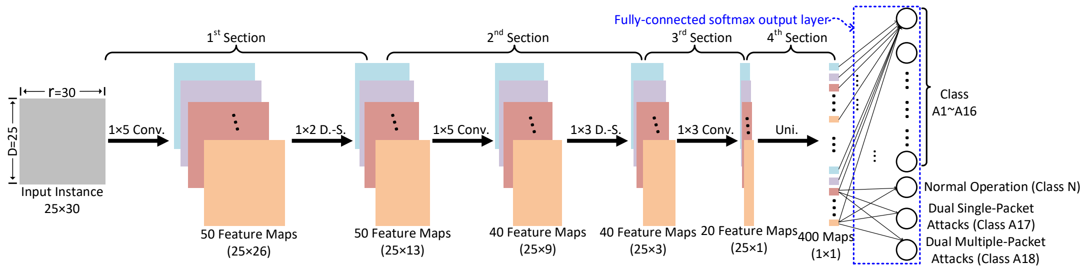

Fig. 3. High-level architecture of the proposed feature learning and classification stages. Note that only convolutional (Conv.) layers, down-sampling (D.-S.) layers (i.e., pooling layers), unification (Uni.) layer, and classification layer are explicitly shown.

B. *Convolutional Neural Network for Feature Learning*

Each input instance used by the CNN stage is a two-dimensional matrix. Each matrix contains r samples, and each sample has D features. We choose r to be the average number of packets observed within one second on a SCADA network over a long period of time (e.g., 24 hours). In our experiments (see Sec. V), we choose $r = 30$. For SCADA systems based on DNP3, we choose D to be 25 (see Sec. IV-A). In the training data set, the label of each instance (i.e., a $D\times r$ matrix) is determined by the type of network attack included in the corresponding detection window. Instances that do not include any attack packet share the same label, i.e., "normal operation". In the CNN, the ith layer's jth feature map is also a matrix. The element value at the rth column and the dth row is denoted by $v_{ij}^{d,r}, \forall d{=}1,\cdots,D$. In the CNN stage, multiple convolutional layers and pooling layers are employed.

1) Convolutional Layers: At each convolutional layer, the input feature maps are convolved with several convolutional kernels, which are learned during the training process. The output of the convolutional operators is added by a bias (also learned during training) and then passed to the activation function to generate the feature maps for the next layer. The value $v_{ij}^{d,r}$ is thus given by

$$v_{ij}^{d,r} = \tanh\left(b_{ij} + \sum_k \sum_{p=0}^{P_i-1} \omega_{ijk}^{p} v_{(i-1)k}^{d,r+p}\right), \tag{1}$$

where $\tanh(\cdot)$ is hyperbolic tangent function, b_{ij} is the bias for the feature map, k is the index for feature map from the i–1 layer, ω_{ijk}^{p} is the value at position p of the convolutional kernel, and P_i is the convolutional kernel length.

2) Pooling Layers: At each pooling layer, resolution of the input is reduced by pooling feature maps from the previous layer over their local temporal neighborhoods. A max pooling function is given by

$$v_{ij}^{d,r} = \max_{1\le q\le Q_i} \left(v_{(i-1)j}^{d,r+q}\right), \tag{2}$$

where Q_i is the length of the pooling region. Pooling is essentially a down-sampling operation that increases the invariance of features to distortions on the inputs (e.g., caused by different TCP/UDP/DNP3 payload sizes).

Using the convolution and pooling operations defined above, we construct a CNN shown in Fig. 3. We group layers of the CNN into four *sections*. The first and the second sections share similar high-level architecture. First, a convolutional layer convolves the input or feature maps from the previous layer's output using a set of kernels, which are learned during the training process. Then, a rectified linear unit (ReLU) layer maps the output of the previous layer with the function $relu(v) = \max(v, 0)$. Next, a max pooling layer is deployed to pick out the maximum feature map over its temporal neighborhood. Finally, a normalization layer is added to normalize the values in different feature maps from the previous layer with

$$v_{ij} = v_{(i-1)j}\left(\theta + \alpha \cdot \sum_{t\in M(j)} v_{(i-1)t}^2\right)^{-\beta}, \tag{3}$$

where α, β, θ are hyper-parameters, and $M(j)$ is the set of feature maps used in normalization.

The third section consists of a convolutional layer, an ReLU layer, and a normalization layer. Pooling is not needed as the temporal dimension of each feature map becomes 1.

At the fourth section, a fully-connected layer is developed to unify the feature maps from the previous layer. Mathematically, the unification operation is defined by

$$v_{ij} = \tanh\left(b_{ij} + \sum_k \sum_{d=1}^{D} \omega_{ijk}^{d} v_{(i-1)k}^{d}\right). \tag{4}$$

This unification layer is followed by an ReLU layer and a normalization layer.

C. *Design of Classification Stage and Re-training Scheme*

The classification stage leverages the features learned from the CNN stage to label input instances. In this stage, a fully-connected output layer is attached to the CNN stage to map learned features to output classes. Output of this stage is governed by the softmax function

$$v_{ij} = \frac{\exp(v_{(i-1)j})}{\sum_{j=1}^{C} \exp(v_{(i-1)j})}, \tag{5}$$

where C is the total number of output classes. An entropy cost function is constructed using the true labels of the training instances and softmax function's probability output.

When one or multiple previously unseen attack instances are present in a detection window, output neurons in the classification stage will have output values that are low and possibly close to each other. This facilitates the design of a re-training scheme that allows SCADA system operator to refine our neural network models with site-specific network attack traces. During the detection (i.e., testing) process, if the classification stage outputs for all the existing classes fall below a threshold R_{th} ($0{\le}R_{th}{\le}1$), we log the features extracted by the CNN stage for the corresponding detection window. Such a window will temporarily be label as "unknown", indicating that it includes previously unseen network behavior patterns. When a sufficiently large number of "unknown" instances are collected, we perform k-means clustering based on Euclidean distance metric to find the number of distinct clusters and request the SCADA system operator to inspect and properly label these clusters. Labeled instances from these clusters are then pumped into the existing training set, and the entire neural network is re-trained with previously unseen attack classes added to the classification stage. Note that the value of R_{th} needs to be empirically determined.

V. Evaluation

We evaluate our intrusion detection solution using SCADA traffic data collected from two SCADA cyber-security test beds, both emulating realistic operational SCADA networks for energy-delivery systems. One test bed is constructed by the National Center for Reliable Electric Power Transmission (NCREPT) at University of Arkansas. This test bed consists of a remote terminal unit (RTU), which controls multiple protective relays via serial connections. The RTU is connected to Ethernet, coordinating an HMI's access to the protective relays. From the HMI, an operator can periodically poll process data from the protective relays and/or issue control commands using DNP3. The other test bed is built by the cyber-security lab of the Electric Power Research Institute (EPRI). This test bed organizes a few dozens of protective relays from different vendors, RTUs, engineering workstations, and computing servers into a SCADA network. In addition to polling and sending commands, an engineering workstation can also update the configurations of protective relays using vendor-specific programming tools. SCADA hosts in both test beds have similar average DNP3 packet rates (i.e., about 120 packets per minute), and we monitor network traffic on both test beds for more than 8 hours.

A. Data Set Overview

After collecting realistic SCADA network traffic from different test beds, we create our data set as follows. First, packets generated by each SCADA host are extracted. Then, we edit the destination and source IP addresses of the packets so that SCADA hosts from different test bed can reside on the same local area network. Next, a set of network-based attacks on SCADA systems are injected into the data set. We create the attack packet sequences according to [9], [10], [12]. Specifically, network packets associated with the following attacks are created:

- *ARP spoofing attacks (A1) [9].* Such attacks can be launched by a device planted by a malicious insider or a SCADA host compromised by an external attacker. In SCADA environments, ARP spoofing attack can be launched at the beginning of a man-in-the-middle attack to hijack communication sessions between normal, legitimate SCADA hosts. It can also be launched to link multiple IPs to a single SCADA host, overloading the target host or causing its failure.
- *SYN flooding attacks (A2) [9].* For legacy SCADA field devices with limited computation capacity, such attacks can devour their system resources and block other SCADA hosts from establishing legitimate connections.
- *TCP RST attacks on telnet (A3).* In the collected SCADA traffic traces, we find that protective relays from a popular vendor can be reconfigured by the engineering workstation via Telnet. A malicious SCADA host can falsify a TCP packet from the engineering workstation with the RST flag set, effectively shutting down the reconfiguration session between engineering workstation and protective relay.
- *UDP flood attacks (A4).* UDP flood attacks can also be launched to overwhelm legacy SCADA field devices, which have limited computation resources.
- *DNP3-specific attacks (A5∼A16) [10], [12].* DNP3 attacks reported in the literature are also created and injected into the data set. These attacks can be categorized into two types. The first type is single-packet attacks. Each type of these attacks can be implemented by modifying a single packet in a normal DNP3 communication session. Single-packet attacks include application termination attacks (A5), outstation DFC flag attacks (A6), function reset attacks (A7), unavailable function attacks (A8), address alteration attacks (A9), outstation write without reading (A10), clear object attacks (A11), outstation data reset attacks (A12), and configuration capture attacks (A13). The second type is multiple-packet attacks, which can be realized by modifying multiple packets in a normal DNP3 session. Multiple-packet attacks include data-link layer length overflow attacks (A14), fragmented message interruption attacks (A15), and pseudo-transport layer sequence modification attacks (A16).

In addition to the "normal operation" class (i.e., class N) and attack classes A1∼A16, we also include two special classes (see Fig. 3), i.e., "dual single-packet attacks" (i.e., class A17) and "dual multiple-packet attacks" (i.e., class A18). A detection window with the label "A17" includes two single-packet attack instances, which can be any combination of instances from classes A5∼A13. A window with the label "A18" includes two multiple-packet attack instances, which can be any combination of instances from classes A14∼A16.

B. Experiment Settings and Evaluation Results

Throughout our experiments, we choose $\alpha = 3.5 \times 10^{-4}, \beta = 0.7, \theta = 1, R_{th} = 0.125$, and the size of $M(j)$ is 5. To choose the parameters in the CNN, we follow the guidelines introduced in [23] during the training process.

1) Experiment I: We first train our proposed algorithm using a training data set containing attack instances from classes A1∼A9 and A14 (see Sec. V-A). Table I summarizes the detection performance results when a testing data set with attack instances from the same classes as used in the training process is presented to our algorithm. We can observe that our algorithm is able to accurately identify instances of all the attack classes seen in the training process. The overall detection accuracy is 99.38%. In fact, only one

TABLE I
DETECTION PERFORMANCE OF EXPERIMENT I.

Class	N	A1	A2	A3	A4	A5
Precision	99.8%	100%	100%	99%	100%	100%
Recall	99.9%	96%	97%	100%	100%	100%
Class	A6	A7	A8	A9	A14	
Precision	94%	94.2%	96.8%	100%	94.9%	
Recall	94.9%	97%	93.9%	94%	100%	

detection window with no attack instance (out of 2200 total "normal operation" windows) is incorrectly labeled as an attack of type A3. We also observe that 7 instances (out of a total of 200 testing instances) from classes A1 and A2 are misclassified as "normal operation". Very few instances of classes A6∼A9 and A14 are incorrectly labeled as attacks from other classes, but they are never classified as "normal operation". As observed in real-world attacks (e.g., [8], [9]), primitive attacks of types A1 and/or A2 usually need to be launched multiple times, so our detection algorithm can still help SCADA system operator detect such attacks. These results show that the number of false positive instances generated by our tool is extremely low and only a few attack instances from classes A1 and A2 are misclassified as "normal operation". Therefore, our tool is well-suited for network intrusion detection in SCADA systems because most detection windows requiring the attention of the operator indeed contain primitive attacks.

2) Experiment II: Next, we add previously unseen attack instances into the testing set of Experiment I and re-execute the detection (i.e., testing) process. When the output of the classification stage and the threshold value R_{th} indicate that a detection window contains previously unseen network behavior patterns, we temporarily label it as an "unknown" class instance and record the features learned by the CNN stage. Once a sufficiently large number of "unknown" class instances are collected, we perform k-means clustering to find distinct clusters contained in the "unknown" class. Our tool then requests the SCADA system operator to inspect instances from different clusters and provide them with proper labels. Once the number of instances included in a cluster is sufficiently large, we pump these instances into the existing training set, add a new attack class in the classification stage, and re-train the entire neural network.

TABLE II
DETECTION RESULTS FOR TESTING PHASE IN EXPERIMENT II.

Actual Class	N		A1		A2		A3	A4	A5	A6	
Total Instance	3550		100		100		100	100	100	100	
Detected	3549		96		97		100	100	100	94	
Misclassified (Class | Count)	A3	1	N	4	N	3				A14	5
										A15	1

Actual Class	A7		A8		A9		A10		A11		A12	
Total Instance	100		100		100		100		100		100	
Detected	97		92		94		97		96		98	
Misclassified (Class | Count)	A8	3	A7	6	A6	6	A11	3	A10	1	A11	2
			A13	2					A12	3		

Actual Class	A13		A14		A15		A16		A17		A18	
Total Instance	100		100		100		100		100		100	
Detected	97		93		97		94		97		98	
Misclassified (Class | Count)	A8	3	A12	3	A12	1	A12	1	A6	2	A15	1
			A15	4	A14	2	A14	2	A8	1	A16	1
							A15	3				

Table II summarizes the detection results obtained after our algorithm is re-trained to include classes A10∼A13 and A15∼A18 in the classification stage. The overall detection accuracy is 99.84%, which is sufficiently high for intrusion detection in real-world SCADA systems. Note that a sufficiently large number of instances of a newly emerged attack class must first be collected to create a representative training set. In our experiment, we require that 100 instances for each new attack type must be labeled before being added into the existing training set for re-training because our experience suggests that adding fewer instances into the training set will result in significant deterioration of detection accuracy (e.g., previously unseen attacks may be incorrectly labeled as instances of other classes). Misclassifying previously unseen attacks may hinder SCADA system operator from becoming aware of newly emerged threats quickly and mislead him/her into taking ineffective countermeasures.

As shown in Table II, our algorithm still achieves high accuracy with very few false positives once sufficient previously unseen attack instances are pumped into the training set. Although a few instances from classes A6∼A18 are misclassified as other attack classes, they are never classified as "normal operation". These results suggest that our re-training scheme facilitates rapid adaptation of our neural network models to SCADA environments that are subjected to previously unseen primitive attacks. These properties, coupled with the fact that our approach identifies primitive attacks without interrupting SCADA system operation, make our approach suitable for network intrusion detection in SCADA systems.

VI. CONCLUSION AND FUTURE WORK

In this paper, we design and implement a network intrusion detection solution for SCADA systems based on deep neural networks. CNN-based feature learning is particularly useful in developing IDS solution for SCADA systems when temporal network behavior patterns of SCADA hosts need to be reliably modeled. Achieving high detection accuracy and permitting adaptations with network traces of previously unseen attacks, our proposed approach is well-suited for network intrusion detection in SCADA networks where both conventional and specialized attacks may be present.

For our future work, we will further evaluate our approach against mixed attack types and also enhance it to address attacks on other SCADA protocols [10]. We also expect to further refine our solution for ICS application domains other than energy-delivery systems.

ACKNOWLEDGMENT

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000779. We would also like to thank NCREPT at University of Arkansas and the EPRI cyber-security lab at Knoxville, TN for offering access to their SCADA system cyber security test beds and helping us create the data sets for our experiments.

References

[1] A. Ghaleb, S. Zhioua, and A. Almulhem, "On PLC Network Security," *International Journal of Critical Infrastructure Protection*, vol. 22, pp. 62–69, September 2018.

[2] L. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. A. Mohammed, and S. A. Zonouz, "Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit," in *Network and Distributed System Security Symposium (NDSS)*, 2017.

[3] R. Langner, "Stuxnet: Dissecting a Cyberwarfare Weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, May 2011.

[4] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the Cyber Attack on the Ukrainian Power Grid," *Electricity Information Sharing and Analysis Center (E-ISAC)*, March 2016.

[5] R. R. R. Barbosa, R. Sadre, and A. Pras, "Difficulties in Modeling SCADA Traffic: A Comparative Analysis," in *Passive and Active Measurement (PAM)*, March 2012, pp. 126–135.

[6] S. Mathew, S. Upadhyaya, M. Sudit, and A. Stotz, "Situation Awareness of Multistage Cyber Attacks by Semantic Event Fusion," in *2010 Military Communications Conference (MILCOM)*, October 2010, pp. 1286–1291.

[7] J. Navarro, A. Deruyver, and P. Parrend, "A Systematic Survey on Multi-Step Attack Detection," *Computers & Security*, vol. 76, pp. 214–249, July 2018.

[8] "US-CERT Alert TA17-163A: CrashOverride Malware," https://www.us-cert.gov/ncas/alerts/TA17-163A, accessed: March 2019.

[9] B. Zhu, S. Sastry, and A. Joseph, "A Taxonomy of Cyber Attacks on SCADA Systems," in *2011 IEEE International Conference on Internet of Things and 4th IEEE International Conference on Cyber, Physical and Social Computing (iThings/CPSCom)*, October 2011, pp. 380–388.

[10] A. Volkova, M. Niedermeier, R. Basmadjian, and H. de Meer, "Security Challenges in Control Network Protocols: A Survey," *IEEE Communications Surveys & Tutorials*, pp. 1–1, September 2018.

[11] B. Miller and D. Rowe, "A Survey SCADA and Critical Infrastructure Incidents," in *Proceedings of the 1st Annual Conference on Research in Information Technology (RIIT)*, October 2012, pp. 51–56.

[12] S. East, J. Butts, M. Papa, and S. Shenoi, "A Taxonomy of Attacks on the DNP3 Protocol," in *International Conference on Critical Infrastructure Protection (ICCIP)*, March 2009, pp. 67–81.

[13] "IEEE Standard for Electric Power Systems Communications–Distributed Network Protocol (DNP3)," IEEE Standard, October 2012.

[14] H. Hindy, D. Brosset, E. Bayne, A. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens, "A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets," *arXiv preprint arXiv:1806.03517*, 2018.

[15] P. Maynard, K. McLaughlin, and S. Sezer, "Using Application Layer Metrics to Detect Advanced SCADA Attacks," in *International Conference on Information Systems Security and Privacy (ICISSP)*, January 2018, pp. 418–425.

[16] R. L. Perez, F. Adamsky, R. Soua, and T. Engel, "Machine Learning for Reliable Network Attack Detection in SCADA Systems," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, August 2018, pp. 633–638.

[17] H. Yang, L. Cheng, and M. C. Chuah, "Detecting Payload Attacks on Programmable Logic Controllers (PLCs)," in *2018 IEEE Conference on Communications and Network Security (CNS)*, May 2018, pp. 1–9.

[18] P. Maynard, K. McLaughlin, and B. Haberler, "Towards Understanding Man-in-the-Middle Attacks on IEC 60870-5-104 SCADA Networks," in *International Symposium for ICS & SCADA Cyber Security Research (ICS-CSR)*, September 2014.

[19] A. Abbasi and M. Hashemi, "Ghost in the PLC Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack," *Black Hat Europe*, pp. 1–35, November 2016.

[20] N. N. Tran, R. Sarker, and J. Hu, "An Approach for Host-Based Intrusion Detection System Design Using Convolutional Neural Network," in *International Conference on Mobile Networks and Management (MONAMI)*, December 2017, pp. 116–126.

[21] Z. Tian, Y. Cui, L. An, S. Su, X. Yin, L. Yin, and X. Cui, "A Real-Time Correlation of Host-Level Events in Cyber Range Service for Smart Campus," *IEEE Access*, vol. 6, pp. 35 355–35 364, June 2018.

[22] "Catalyst Switched Port Analyzer (SPAN) Configuration Example," https://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html, accessed: October 2018.

[23] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient Backprop," in *Neural Networks: Tricks of the Trade, Second Edition*. Springer Berlin Heidelberg, 2012, pp. 9–48.