I. Siniosoglou et al., "NeuralPot: An Industrial Honeypot Implementation Based On Deep Neural Networks," presented at the 2020 IEEE Symposium on Computers and Communications (ISCC), Jul. 2020, doi: 10.1109/iscc50000.2020.9219712

NeuralPot: An Industrial Honeypot Implementation Based On Deep Neural Networks

Ilias Siniosoglou*, Georgios Efstathopoulos†, Dimitrios Pliatsios*, Ioannis D. Moscholios‖
Antonios Sarigiannidis¶, Georgia Sakellari‡, Georgios Loukas‡, Panagiotis Sarigiannidis*§
*Department of Electrical and Computer Engineering
University of Western Macedonia, Kozani, Greece
{isiniosoglou, dpliatsios, psarigiannidis}@uowm.gr
†0 INFINITY Limited
Imperial Offices, London, United Kingdom
george@0inf.com
‖Department of Informatics and Telecommunications
University of Peloponnese Tripoli, Greece
idm@uop.gr
¶Sidroco Holdings Ltd.
Limassol, Cyprus
asarigia@sidroco.com
‡Computing and Information Systems
University of Greenwich, London, United Kingdom
{g.sakellari, g.loukas}@greenwich.ac.uk

***Abstract*—Honeypots are powerful security tools, developed to shield commercial and industrial networks from malicious activity. Honeypots act as passive and interactive decoys in a network attracting malicious activity and securing the rest of the network entities. Since an increase in intrusions has been observed lately, more advanced security systems are necessary. In this paper a new method of adapting a honeypot system in a modern industrial network, employing the Modbus protocol, is introduced. In the presented NeuralPot honeypot, two distinct deep neural network implementations are utilized to adapt to network Modbus entities and clone them, actively confusing the intruders. The proposed deep neural networks and their generated data are then compared.**

***Index Terms*—Industrial Control System, SCADA, Honeypots, GAN Network, Autoencoder Network, Data Generation**

I. Introduction

Industrial Control Systems (ICS) are the fundamental control elements, both hardware and software, used to organize and oversee industrial network processes such as water and gas pipeline distribution, heavy manufacturing, generation and distribution of energy. A typical ICS system is composed of a central controller, and a number of distributed field devices, such as sensors and actuators. Custom communication protocols are used to enable the data exchange between the controller and the field devices. As the legacy ICS operated on isolated networks, using custom communication protocols, they were resistant to cyber attacks [1]. Driven by the need for high scalability, computational-intensive processes, and remote monitoring and control, as well the rapid evolution of Information and Communication Technologies (ICT), modern ICS are connected to the Internet. In addition, in order to provide seamless integration among various components, as well as different vendors, well-known communication protocols are utilized. As a result, modern ICS are exposed to numerous security threats.

A cyber attack against an ICS can have devastating consequences on public health and safety. For example, an attacker can compromise an ICS and shut down electricity, gas, and water services, or destroy critical military infrastructure. Reports in [2] and [3] show an increasing number of security incidents and cyber attacks against critical ICS infrastructure. Consequently, security considerations for ICS are gaining higher priority and consideration than those for traditional ICT systems due to the potential impact on the physical safety of employees, customers, or communities. The Repository of Industrial Security Incidents (RISI) [4] contains 242 reported incidents dating from 1982 to 2014. Each record contains the year, title, industry type, country and information about the incident and its impact.

In order to address these threats, ICS have adopted several security mechanisms and tools form the conventional computer networks. The Intrusion Detection Systems (IDS) are effective countermeasures against security threats. Depending on their operation, IDS can be categorized into signature-based and

§ The corresponding author is Panagiotis Sarigiannidis (psarigiannidis@uowm.gr)

978-1-7281-8086-1/20/$31.00

behavior-based detection systems [5]. Signature-based IDS, which are simpler to deploy and operate, utilize a database of previously known attack signatures and system vulnerabilities. However, they are inefficient against new and unknown attacks. Behavior-based IDS continuously monitors the network traffic and compare it to a reference traffic pattern. In case of any deviation from the reference traffic, the IDS classifies the traffic as a cyber attack.

A. Motivation and Contribution

The concept of honeypots has emerged as an effective method to generate the signature database, as well as to discover novel attack methods and tools [6]. Honeypots mimic the operation of applications, services, and devices in order to attract potential attackers to attack them instead of attacking the real ones [7].

Honeypots are extensively used in the protection of conventional computer networks. Nevertheless, the use of honeypots in industrial environments is limited, due to several challenges [8]. In this paper, a novel method of generating pseudo-traffic used for masking and adapting an industrial honeypot into a network is described.

This implementation aims to actively mislead attackers and redirect their interest away from the real network devices. To accomplice this, a Deep Neural Network (DNN) scheme is introduced. DNNs are used in a variety of technological and scientific fields due to their rapid evolution and implementation as well as their reliability and scalability. This work leverages DNNs as a dynamic method of generating Modbus traffic data. The generated data are not statically defined, but they are adapted to those of a real device.

Specifically, two different categories of Neural Networks are employed, namely the Generative Adversarial Network (GAN) [9] and the Auto-Encoder Network [10] in order to learn the device behavior and generate similar traffic. Finally, the DNN implementations are compared to evaluate their performance.

This work also introduces a new way of preprocessing and transforming Modbus response data from a network entity to Modbus memory data and consequently presenting them to the aforementioned deep neural networks. From this method all the required information is extracted to customize a honeypot to the specific network entity.

By adapting these techniques into modern honeypots and placing multiple of those honeypots into a network, due to their low hardware needs, will engulf any possible intruder with a plethora of digital interactive mines and will make the important components of the network almost indistinguishable to the attackers means.

Therefore, the contribution of this work is summarized as follows:

- Design an autonomous system that dynamically analyzes network traffic from Remote Terminal Units (RTUs) and Programmable Logic Controllers (PLCs).
- Design a Deep Neural Network (DNN) that generates network traffic that is adapted to the real network traffic.
- Implement a novel honeypot that utilizes DNNs to generate traffic in order to attract potential attackers and mislead them into attacking the honeypot instead of the real RTUs and PLCs.

The rest of the paper is organized as follows: Section II presents the related work, while section III provides the fundamental background. Section IV presents the design and the proof of concept implementation. In section V the evaluation results are presented and discussed. Finally, section VI concludes the paper.

II. Related Work

The notion of honeypots is quite popular in the literature. The authors in [11] reviewed and discussed the recent advances as well as the future trends in honeypot research. The survey suggests that honeypot research is on the rise due to the increasing number of connected devices. Moreover, research honeypots generate valuable data that are used to improve and develop new honeypots. Finally, the legal and ethical concerns of honeypot usage is an important research area.

Simoes et al. [12] investigated the utilization of honeypots in ICS environments, along with implementation and deployment strategies. In addition, the authors impended and compared two ICS honeypot systems, one hosted on a physical device, while the other is hosted on a virtual machine. The results indicate that low-cost machines can provide enough computational resources, and in cases where the location of the honeypot is irrelevant, the virtual honeypots are more flexible and cost-effective.

The authors in [13] presented the architecture of an ICS novel honeypot, and deployed a modular and scalable honegynet architecture on the Amazon EC2 cloud platform. In addition, they conducted a series of realistic experiments in order to validate feasibility of the proposed approach, as well as to highlight the impact of proper security mechanisms in ICS environments.

In [14], the authors presented the design of a high-interaction ICS honeypot that aims to address the main challenges related to ICS requirements. In addition, the authors utilized the MiniCPS framework in order to implement the proposed honeypot. In order to evaluate it, they organized a Capture The Flag (CTF) competition, hosted by Singapore University of Technology, where they deployed a water treatment testbed.

The authors in [15] presented a method for a dynamic honeypot configuration, deployment, and maintenance strategy based on machine learning techniques. The method utilizes an identification mechanism in order to cluster the devices in a network. Based on the clusters, a number of honeypots is smartly deployed in the network. The main benefit of the proposed approach is that no configuration and maintenance are required after the deployment.

Cao et al. [16] proposed DiPot which is a distributed industrial honeypot system, that provides deep data analytics and advanced visualization techniques. DiPot is a modular

honeypot that consists of three nodes, namely honeypot, processing, and management nodes. The honeypot node emulates an ICS device, while the data processing node periodically analyses raw log files. The management node facilitates user interaction and provides data visualization functionalities. In order to evaluate Dipot, large amounts of both legitimate and malicious network traffic were captured and analyzed.

The authors in [17] developed an intelligent honeypot that uses reinforcement learning to proactively engage with and learn from attacker interactions. Therefore, it adapts its behavior for automated malware to optimize the volume of data collected. To achieve its aim, the honeypot leverages machine learning techniques to retrospectively model botnet interactions.

Pauna et al. [18] proposed an SSH-based interactive honeypot using Reinforcement Learning. The honeypot aims to learn the attacker's behavior and generate a series of actions to maximize the defender's long-term reward. In order to train the honeypot, a deep neural network is utilized using the Deep Q-Learning method.

The authors in [19], designed an ICS honeypot that collects and feeds intelligence to real-world ICS cyber security monitoring services. The ICS system module emulates the HMI and the PLC devices, the simulation system that evaluates the process status variables in real time, and the cybersecurity monitoring infrastructure that collects and generates information about the cyber attackers. The honeypot continuously provides security intelligence and insights such as, correlation rules, IDS signatures, and general awareness of the cyber threat landscape.

The authors in [20] designed and implemented an interactive ICS honeypot, that emulates a physical ICS device by replicating realistic traffic from a real device. The implemented ICS honeypot is based on Conpot, while the Modbus ICS communication protocol is used for the communication between the ICS devices. The honeypot runs inside a virtual machine, in order to facilitate the emulation of the entire organization's ICS infrastructure.

III. Background

A. Conpot Honeypot

The proposed approach is based on the Conpot honeypot, which is an industrial honeypot that utilizes well-known industrial communication protocols [21]. These include the IEC 60870-104, Backnet, EtherNet/IP, Guardian AST, Kamstrup, Modbus, S7Comm communication protocols. For this work, the Modbus communication protocol was selected since it is widely used in industrial applications.

B. Modbus Communication Protocol

Modbus is an open and royalty-free communication protocol, that is widely used in industrial applications [22]. It is a simple and easy to deploy protocol, developed to facilitate the communication among PLCs and RTUs. Modbus supports both serial and Transmission Control Protocol (TCP) communication schemes.

TABLE I: Notations & Symbols

Term	Description
x_i	Feature i of input vector x
x'	Flattened data vector
G	Generator
D	Discriminator
z	Random noise
$p(\cdot)$	Probability function
y_i	Label of sample i
$\sigma(x)$	Normalized sigmoid function
n	Number of predictions
M	Number of features
μ_r	Real data
μ_p	Predicted data
Σ_r	Covariance matrix of real data
Σ_p	Covariance matrix of predicted data

The basic Modbus entities in a network are the Modbus clients, masters, and slaves. A client is a remote query terminal, such as a Human-Machine Interface (HMI) that requests information from the Modbus master and sends control information to them. The servers are usually PLCs or RTUs throughout the network, that manage the slaves (e.g., acquisition blocks), that oversee the field devices. Each server can have multiple slaves with unique slave IDs associated with them.

In the Modbus protocol, the data are stored in four tables, with each table corresponding to the discrete (called coils) and numerical (called registers) inputs and outputs, respectively. The master utilizes several Function Codes in order to communicate with the PLCs and RTUs. The most common function codes include the Read Coil Status (FC01), Read Input Status (FC02), Read Holding Registers (FC03), Read Input Registers (FC04), Force Single Coil (FC05), Preset Single Register (FC06), Force Multiple Coils (FC15), Preset Multiple Registers (FC16).

C. Network Traffic Dataset

The datasets for the training and testing of the DNNs are extracted from the real network traffic. The network traffic is collected and stored in a pcap file. The collected traffic corresponds to the communication of an HMI with a PLC and a RTU in the network. Specifically, the HMI sends requests to the PLC and RTU for an update on a value, that is stored in the device memory (i.e., Read Holding Registers (FC03)). Upon the reception of the request, the PLC or the RTU responds with a packet that contains the requested values.

IV. Design and Implementation

This section provides a detailed description of the design and implementation of the DNN that generates the Modbus network traffic. Table I lists the notations and symbols that are used in this work.

A. Problem Statement

Most of the works that were reviewed in section II, implement a honeypot using preconfigured traffic, in order to act as a real device and attract potential attackers. In this work, we adopt a novel approach in the implementation of a honeypot.

To achieve this, we utilize a DNN that generate network traffic, which is roughly identical to the real traffic. Consequently, the generated traffic is dynamic and has a higher probability to attract attackers.

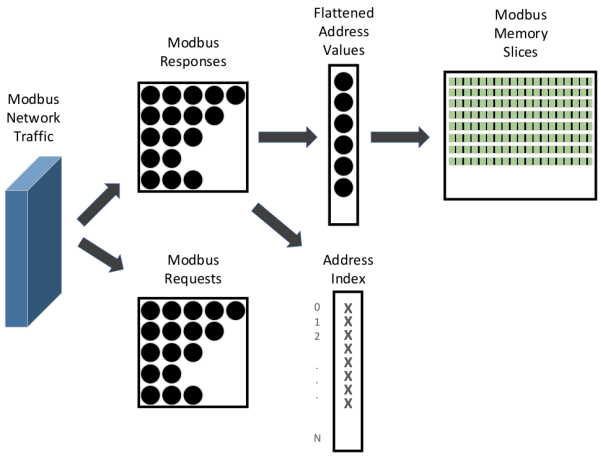

Fig. 1: Pcap file - Modbus traffic

B. Data Preprocessing

Dataset Generation: Fig. 1 depicts a high-level view of the dataset generation process. Two pipelines have been developed to extract and transform the data into a suitable structure that will be used in the training process. The first pipeline parses the raw traffic from a pcap file and extracts the selected features into two separate categories, one for the Modbus request and one for the Modbus responses, respectively. For the requests the selected features are: i) Relative-Time, ii) Type, iii) Transaction-ID, iv) Protocol-ID, v) Length, vi) Unit-ID, vii) Function Code, viii) Start Address, ix) Quantity (of Addresses). Regarding the responses, the Quantity feature is replaced with the Byte Count feature, while an additional feature, namely x) Address, is selected.

The second pipeline takes over the procedure of transforming the selected features in the appropriate form to be inputted in the training process and to create metadata, which will later be used to correlate the produced values with the modbus standard structure. Consequently, since the flattened data are not sorted, a sorting function is leveraged in order to include the different values of the addresses to the memory instance without omitting values. In order to transform the flattened data into an appropriate form, the process creates a tuple of all of the given values in an instance, which is considered as the tuple of values of addresses between two recurring addresses. Afterward, the generated tuples are exported to a csv file, that is used as input to the neural network. In order to improve the training and testing effectiveness, the datasets are scaled using a MinMax Scaler based on the following formula:

$$x' = \frac{x_i - min(x)}{max(x) - min(x)} \tag{1}$$

where x' is the scaled vector of data, x is the inputted vector of data and x_i is the different features in the data vector.

C. GAN Architecture

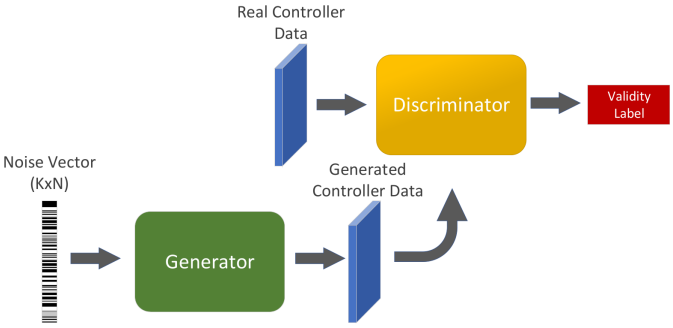

Fig. 2: GAN Architecture

The GAN architecture [9] [23], as shown in Fig. 2, is based on a pair of neural sub-networks, namely the Generator that generates the mimic data using noise as input, and the Discriminator that classify the generated data into fake and real. The GAN aims to generate data that the discriminator will classify as real. Equation (2) below shows the relationship between the Generator and the Discriminator (denoted as G and D, respectively) as a value function.

$$\begin{aligned} \min_G \max_D V(G,D) = \min_G \max_D \mathbb{E}_{x\sim p_{data}}[log(D(x))] + \\ \mathbb{E}_{z\sim p_z}[log(1-D(G(z)))] \end{aligned} \tag{2}$$

in which the G accumulates noise z from space Z and outputs x, which is forwarded to the D. The terms $p_{data}(x)$ and $p_z(z)$ denote the probabilistic distribution of spaces X and Z respectively. In the proposed implementation, the GAN consists of three different components. The first component is the Input module, the second is the Generator module and third the Discriminator module.

Input Module: The Input module of this GAN is a simple layer with an input size of 100, that describes the randomly generated input noise given to the Generator to produce the simulating data. The random noise is created using the normal distribution with mean $\mu = 0$ and a standard deviation of $\sigma = 1$.

Generator Module: The Generator module is one of the two neural sub-networks in the GAN architecture. It aims to produce an output that is almost identical to the real data. In this GAN, the Generator is composed of seven layers and it is compiled with the Binary Crossentropy loss function (3) and the Adam Optimizer [24].

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1-y_i)\cdot log(1-p(y_i)) \tag{3}$$

where N is the number of samples given, y is the data label, and $p(y_i)$ is the probability of the sample being a match to the label.

The architecture of the Generator module is shown in Fig. 3. The first layer is the Generator's input dense layer that has a size of 100 tuples. Among the remaining layers, three are

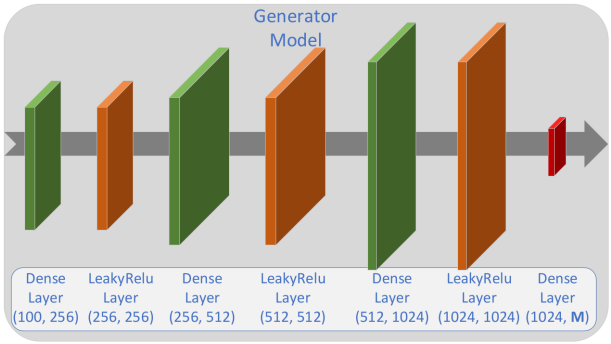

Fig. 3: Generator Module Architecture

dense layers, where the number of neurons is increasing from 256 to 1024. The output layer contains M number of neurons, where M is the number of selected features. The rest of the layers are Leaky ReLU layers that follow the first, second and third dense layers.

Discriminator Module: The second neural sub-network in the proposed GAN architecture is the Discriminator, which is responsible for the classification of the real data, originating from the input dataset, and the generated data, originating from the Generator module. The Discriminator is trained on both real and generated data.

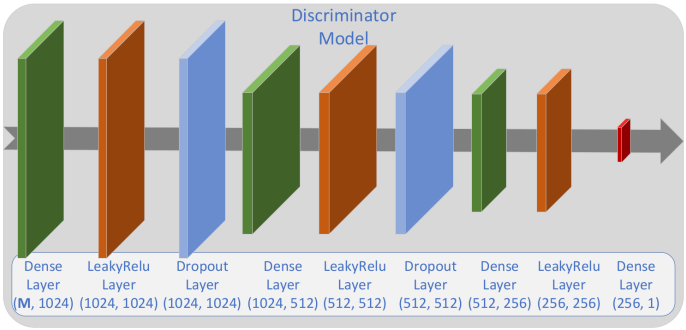

Fig. 4: Discriminator Module Architecture

The architecture of the Discriminator module is shown in Fig. 4. The module includes nine layers, consisting of Dense, LeakyReLU and Dropout layers. The first layer is the Discriminator's input layer, with an input dimension of M. Each one of the first three Dense layers is followed by a LeakyReLU layer. In order to prevent overfitting, each of the first two combinations of Dense and LeakyReLU is followed by a Dropout layer [25]. Finally, the last layer produces the output using a sigmoid activation function:

$$s(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

where x is the input data vector, and the output of the function is 0 or 1. The result is used as a label, indicating whether the input data was real or generated.

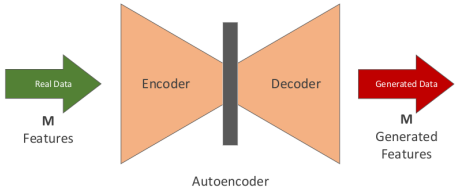

Fig. 5: Autoencoder Architecture

D. Auto-Encoder Architecture

The basic concept of the Auto-Encoder is the assimilation of given data of space X into a compressed manifold F of those data using the encoder module and consequently the scaling of that manifold F to the predicted value P of those given data by the decoder, where $P \sim X$. Fig. 5 depicts the architecture of the Autoencoder.

Encoder Module: The role of the Encoder module is to compress the input data to a pre-defined output size and forward the output to the Decoder for scaling. The architecture

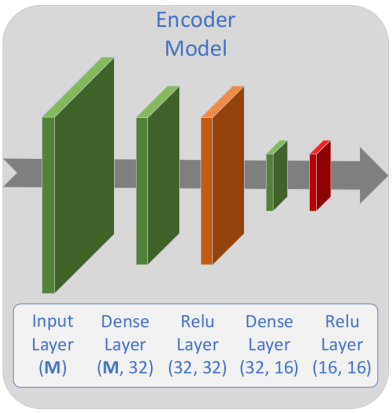

Fig. 6: Encoder Module Architecture

of the Encoder module is shown in Fig. 6. The Encoder module is comprised of an input layer followed by two Dense layers. The input layer has an input dimension of M and no activation function. The following two layers consist of 32 and 16 neurons, respectively. Both of them utilize the ReLU activation function, which replaces all negative values with zeros. The Encoder integrates the Mean Square Error loss function:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \tag{5}$$

where n represents the number of predictions, while Y and $\tilde{Y}$ are the samples and predicted values vector, respectively.

Decoder Module: The aim of the Decoder module is to scale the data generated by the Encoder in order to make them similar to the real data. Fig. 7 depicts the architecture of the Decoder module. The Decoder module consists of three Dense layers. The first two layers contain 16 and 32 neurons, respectively, and utilize the ReLU activation function. The last

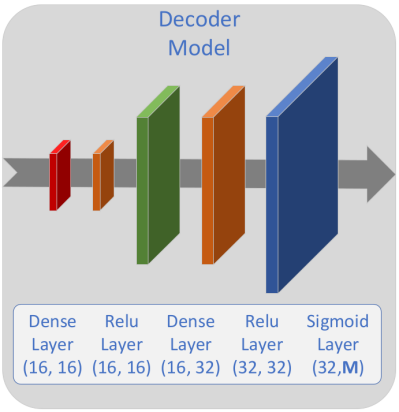

Fig. 7: Decoder Model

layer contains a variable number of neurons, depending on the number of features M of the input data. In addition, the last layer uses the sigmoid activation function (equation 4) to output the scaled data.

E. Conpot Integration

The proposed DNNs were integrated into the Conpot honeypot, by incorporating the trained model to Conpot's databus system, which performs the data acquisition and delivery within the honeypot. Two different indexes were used to cross-reference the generated values and update the Conpot's Modbus memory blocks. One of the manifests keeps the actual Modbus address index in reference to the network produced index. The manifest is a file that contains metadata from the Preprocessing. The manifest contains the essential information required, in order to cross-reference the information that the neural networks generated. This manifest is used with the Modbus memory block index to assign the correct values to their corresponding slave memory block. The index is produced from the profile that Conpot is simulating. Using this configuration, Conpot updates its memory block every time a query is received, successfully emulating a Modbus device.

V. Evaluation

The evaluation is branched into three parts. Firstly, the proposed DNNs are compared using quantitative metrics to evaluate the accuracy of the results. Secondly, the generated data of the two DNNs are statistically compared using similarity metrics and visualization. Finally, the duration of traffic generation is measured.

A. Quantitative Comparison Metrics

The performance of the DNNs is evaluated in terms of similarity with the real data, while the training dataset has a size of 1.0 gigabyte. The performance metrics are the arithmetic mean, the standard deviation, and the the Frechet Inception Distance (FID) [26], [27] score. The FID score is calculated as:

$$FID = ||\mu_r - \mu_p||^2 + tr(\Sigma_r + \Sigma_p - 2\sqrt{(\Sigma_r \cdot \Sigma_p)}) \quad (6)$$

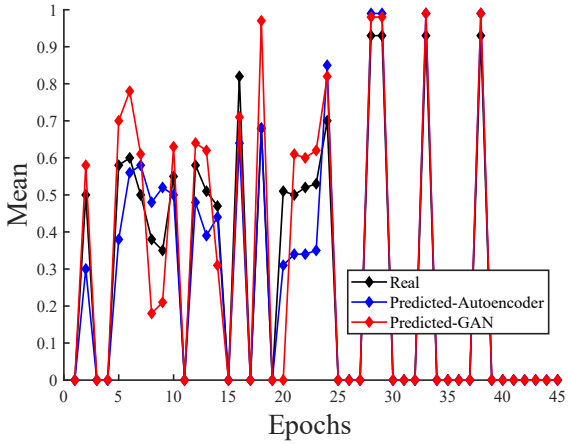

Fig. 8: Arithmetic mean of the data against the number of epochs

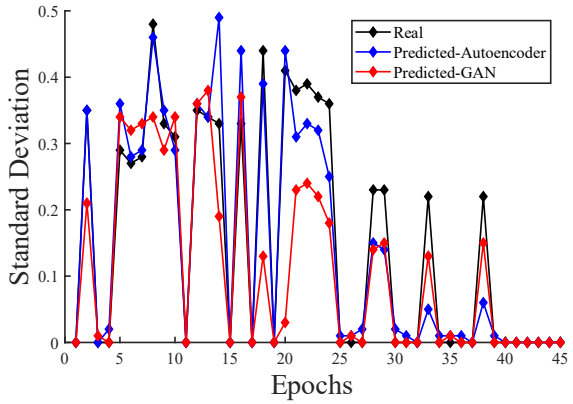

Fig. 9: Standard deviation of the data against the number of epochs

where μ_r and μ_p are the vectors of the real and predicted data respectively, while Σ_r and Σ_p are the covariance matrices of the aforementioned vectors. Finally, the term tr denotes the trace of the matrix.

Figures 8 and 9 depict the similarity between the generated and real data values, in terms of arithmetic mean and standard deviation. Particularly, Fig. 8 shows the arithmetic mean of the data against the number of epochs, ranging from 1 to 45. An epoch indicates the number of times the algorithm analyzes the entire dataset. Therefore, each time the algorithm analyzes all the samples in the dataset, an epoch is completed. Both of the approaches achieve a high overall similarity to the real values, with the GAN achieving a slightly better similarity.

Similarly, Fig. 9 shows the standard deviation of the values against the number of epochs, ranging from 1 to 45. Both approaches achieve a high overall similarity. n this case, GAN also achieves a slightly better similarity than the Autoencoder architecture.

As mentioned the Frechet Inception Distance is employed to measure the difference between the generated and the real samples. In the testing phase, GAN achieves an FID score of

31.29, while the Autoencoder's score is 29.94. Smaller FID score indicates a higher similarity, therefore the Autoencoder generates data with higher similarity compared to the GAN.

B. Time and Complexity

The elapsed time of data generation has a critical impact, as the honeypot has to generate the requested data in a very short time to effectively emulate a real network device. In order to measure the execution time of the proposed DNNs, a testbed has been deployed, where the DNNs run in a virtualized environment. An Intel Core i7-6700HQ has been utilized for the computation, with a 16GB of RAM to its disposal.

The GAN, having a more complex architecture, generates 128 values in 0.6969 ms. On the other hand, the Autoencoder achieved a time of 0.4116 ms. Both times are within the accepted limit (as defined in [28]), therefore both approaches can be effectively used for network traffic generation in real-time.

VI. Conclusion

In this work, we presented the design and implementation of a novel method that adapts honeypot technologies to the requirements of an industrial network. NeuralPot is a highly interactive adaptation of the Conpot honeypot, that generates network traffic based on an existing network entity. The two distinct DNN implementations are compared against each other, as well as against the actual Modbus network traffic. Even though the output-wise results of both DNNs are close, based on the quantitative metrics comparison, the GAN architecture is recommended due to its higher similarity with the real data.

In the future, we aim to deploy the implemented honeypot in a real ICS network containing a large number of ICS devices, in order to evaluate its efficiency in attracting attackers and record their behavior. Furthermore, we aim to incorporate additional well known ICS communication protocols, such as the IEC 60870-104, Backnet, EtherNet/IP, Guardian AST, Kamstrup, and S7Comm.

Acknowledgment

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No. 787011 (SPEAR).

References

[1] D. Pliatsios, P. Sarigiannidis, T. Lagkas, and A. G. Sarigiannidis, "A survey on scada systems: Secure protocols, incidents, threats and tactics," *IEEE Communications Surveys & Tutorials*, 2020.

[2] S. A. Baker, S. Waterman, and G. Ivanov, *In the crossfire: Critical infrastructure in the age of cyber war*. McAfee, Incorporated, 2009.

[3] B. Miller and D. C. Rowe, "A survey scada of and critical infrastructure incidents." *RIIT*, vol. 12, pp. 51–56, 2012.

[4] "RISI - The Repository of Industrial Security Incidents." [Online]. Available: http://www.risidata.com/

[5] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.

[6] C. Dalamagkas, P. Sarigiannidis, D. Ioannidis, E. Iturbe, O. Nikolis, F. Ramos, E. Rios, A. Sarigiannidis, and D. Tzovaras, "A survey on honeypots, honeynets and their applications on smart grid," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 93–100.

[7] A. Mairh, D. Barik, K. Verma, and D. Jena, "Honeypot in network security: a survey," in *Proceedings of the 2011 international conference on communication, computing & security*. ACM, 2011, pp. 600–605.

[8] B. Gupta and A. Gupta, "Assessment of honeypots: Issues, challenges and future directions," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 8, no. 1, pp. 21–54, 2018.

[9] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, 10 2017.

[10] P. Baldi, G. Guyon, V. Dror, G. Lemaire, D. Taylor, and D. Silver, "Autoencoders, unsupervised learning, and deep architectures editor: I," 09 2019.

[11] R. M. Campbell, K. Padayachee, and T. Masombuka, "A survey of honeypot research: Trends and opportunities," in *2015 10th international conference for internet technology and secured transactions (ICITST)*. IEEE, 2015, pp. 208–212.

[12] P. Simões, T. Cruz, J. Proença, and E. Monteiro, "Specialized honeypots for scada systems," in *Cyber Security: Analytics, Technology and Automation*. Springer, 2015, pp. 251–269.

[13] A. V. Serbanescu, S. Obermeier, and D.-Y. Yu, "A flexible architecture for industrial control system honeypots," in *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, vol. 4. IEEE, 2015, pp. 16–26.

[14] D. Antonioli, A. Agrawal, and N. O. Tippenhauer, "Towards high-interaction virtual ics honeypots-in-a-box," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*. ACM, 2016, pp. 13–22.

[15] D. Fraunholz, M. Zimmermann, and H. D. Schotten, "An adaptive honeypot configuration, deployment and maintenance strategy," in *2017 19th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2017, pp. 53–57.

[16] J. Cao, W. Li, J. Li, and B. Li, "Dipot: A distributed industrial honeypot system," in *International Conference on Smart Computing and Communication*. Springer, 2017, pp. 300–309.

[17] S. Dowling, M. Schukat, and E. Barrett, "Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware," *Journal of Cyber Security Technology*, vol. 2, no. 2, pp. 75–91, 2018.

[18] A. Pauna, A.-C. Iacob, and I. Bica, "Qrassh - a self-adaptive ssh honeypot driven by q-learning," 06 2018, pp. 441–446.

[19] Ó. Navarro, S. A. J. Balbastre, and S. Beyer, "Gathering intelligence through realistic industrial control system honeypots," in *International Conference on Critical Information Infrastructures Security*. Springer, 2018, pp. 143–153.

[20] D. Pliatsios, P. Sarigiannidis, T. Liatifis, K. Rompolos, and I. Siniosoglou, "A novel and interactive industrial control system honeypot for critical smart grid infrastructure," in *2019 IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks*. IEEE, 2019, p. to appear.

[21] A. Jicha, M. Patton, and H. Chen, "Scada honeypots: An in-depth analysis of conpot," in *2016 IEEE conference on intelligence and security informatics (ISI)*. IEEE, 2016, pp. 196–198.

[22] P. Huitsing, R. Chandia, M. Papa, and S. Shenoi, "Attack taxonomies for the modbus protocols," *International Journal of Critical Infrastructure Protection*, vol. 1, pp. 37–44, 12 2008.

[23] Y. Hong, U. Hwang, J. Yoo, and S. Yoon, "How generative adversarial nets and its variants work: An overview of gan," *ACM Computing Surveys*, vol. 52, 11 2017.

[24] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.

[26] S. Barratt and R. Sharma, "A note on the inception score," 01 2018.

[27] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," 05 2018.

[28] C. C. S. LLC, "Modbus Message Timing message description," 2018. [Online]. Available: https://ctlsys.com/support/modbus_message_timing/