NeuralPot: An Industrial Honeypot Implementation Based On Convolutional 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, which are developed to shield commercial and industrial networks from malicious activity. Honeypots act as passive and interactive decoys in a network by attracting malicious activity away from critical network devices. Given that the security incidents against industrial and critical infrastructure are getting sophisticated and persistent, advanced security systems are needed. In this paper, a novel industrial honeypot implementation is presented, which is based on the Modbus protocol, entitled NeuralPot. The presented NeuralPot honeypot is able to emulate industrial Modbus entities in order to actively confuse the intruders. It achieves this by introducing two distinct deep neural networks, a Generative Adversarial Network and an Autoencoder Network, which learn Modbus device behavior and generate realistic-looking traffic behavior. Based on the evaluation results, the proposed industrial honeypot performs well in terms of accuracy, similarity, and elapsed time of data generation.**

***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, which are used to organise and oversee industrial network processes such as water and gas pipeline distribution, heavy manufacturing, and energy generation and distribution. 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. 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 cyberattack against an ICS could 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 even destroy critical military infrastructure. Reports in [1] and [2] 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) [3] 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.

The ICS are vulnerable to many threats, such as Denial of Service (DoS) attacks, eavesdropping, Man-In-The-Middle (MITM) attacks, and virus and worm infections. DoS attacks aim to disrupt the operation of the system, by aggressively using all of the available resources of the system, so it cannot respond to the legitimate users. By eavesdropping, the attacker intercepts the communications, thus violating the confiden-

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

tiality of the communication. Mostly wireless communication systems are affected by this attack, as the radio signals spread in a large area, where anyone can intercept the signal and decode the message. In a MITM attack, the attacker acts as a legitimate user between the endpoints of the communication, therefore violating the confidentiality of the communication. Moreover, the attacker can also tamper with the exchanged messages. Finally, virus and worm infections aim to execute malicious code in a compromised system. In addition, worms can launch more cyberattacks from the compromised system.

Most proposed defences focus on secure authentication and intrusion detection. However, attacks cannot always be prevented or detected, especially if they are zero-day attacks. Here, honeypots have an advantage, by luring the otherwise successful attacker to the wrong resources.

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 [4]. Honeypots mimic the operation of applications, services, and devices in order to attract potential hackers to attack them instead of attacking the real ones [5].

Honeypots are extensively used in the protection of computer networks. Nevertheless, the use of honeypots in industrial environments is less common, since older ICS utilize analog communication schemes. In addition, many of the ICS communication protocols are proprietary, therefore they cannot be easily integrated into a honeypot system. In this paper, we present the design and implementation of a novel method that generates realistic traffic. Moreover, the proposed method is integrated into the Conpot honeypot [6], in order to be deployed in an industrial environment.

Our proposed NeuralPot 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 ([7], [8]) 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, since the Modbus protocol is widely used in ICS.

The generated data are not statically defined, but they are dynamically changing based on the data generated by a real device. Consequently, the network traffic is constantly changing, achieving a better emulation of a real device. Furthermore, since the generated data are adapted to the real data, the probability of successfully deceiving an adversary into attacking the device is increased.

Here, two different categories of DNNs 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. These DNN implementations are compared to evaluate their performance.

By adapting these techniques into modern honeypots and placing multiple of those honeypots into a network, it would be more easy to attract attackers, while important traffic traces will be captured. In addition, critical log files, captured by the honeypots, could be used for forensic operations.

In the light of the aforementioned remarks, the contribution of this work is summarised as follows:

- Design a dynamic network analyser for Modbus traffic coming from Remote Terminal Units (RTUs) and Programmable Logic Controllers (PLCs).
- Design a new DNN that generates network traffic adapted to the real network traffic.
- Implement a novel honeypot that utilizes DNNs to generate traffic that attracts 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 the topic of honeypots. 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 utilisation of honeypots in ICS environments, along with implementation and deployment strategies. In addition, the authors introduced two ICS honeypot systems, where the one is 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.

In [13], the authors presented a high-interaction ICS honeypot that aims to address the main challenges related to ICS requirements. In addition, the authors utilised 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.

Cao et al. [14] proposed DiPot, which is a distributed industrial honeypot system that provides deep data analytics and advanced visualisation 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 analysed.

The authors in [15], designed an ICS honeypot that collects and feeds intelligence to real-world ICS cybersecurity monitoring services. The ICS system module emulates the HMI and 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 cyberthreat landscape.

The authors in [16] 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. Most of the works found in the literature 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 utilise a DNN that generates network traffic, where the produced traffic is very similar to the real traffic. This allows adapting to changing conditions, such as new devices being introduced or existing ones being dropped. Consequently, the generated traffic is dynamic and the probability of attracting attackers is getting higher.

III. Background

This section provides a description of the main components, such as the Conpot honeypot, the Modbus communication protocol, and the utilized traffic dataset.

A. Conpot Honeypot

The proposed approach is based on the Conpot honeypot, which is an industrial honeypot that utilises well-known industrial communication protocols [6]. These include the IEC 60870-104, the Backnet, the EtherNet/IP, the Guardian AST, the Kamstrup, the Modbus, and the S7Comm communication protocol. In 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 [17]. 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.

The basic Modbus entities in a network are the Modbus masters, and slaves. A master is usually a remote query terminal, such as a Human-Machine Interface (HMI), that sends control and request information to the Modbus slaves. The slaves are usually PLCs or RTUs, deployed throughout the network. Each server can have multiple slaves with unique slave IDs associated with them.

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

C. Network Traffic Dataset

The datasets for the training and testing processes 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.

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

A. Data Preprocessing

Dataset Generation: Fig. 1 depicts a high-level view of the dataset generation process. Two approaches have been developed to extract and transform the data into a suitable structure, which will be used in the training process. The first approach 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. Regarding 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,

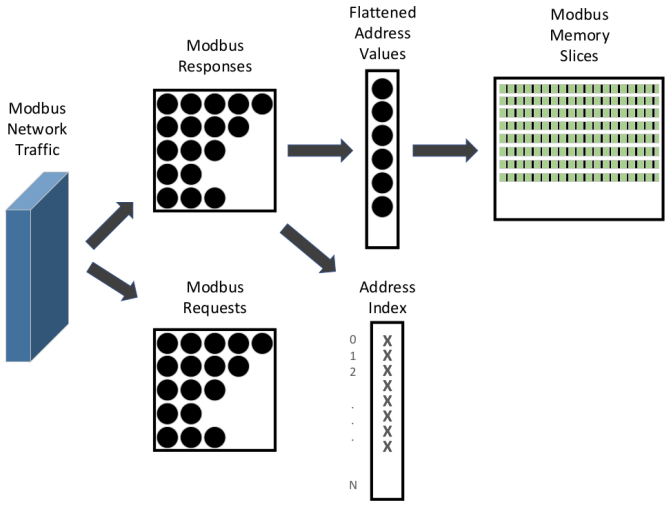


Fig. 1: Pcap file - Modbus traffic

the Quantity feature is replaced with the Byte Count feature, while an additional feature, namely Address, is selected.

Since the flattened data are not sorted, a sorting function is used 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, which 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 input vector of data and x_i is the different features in the data vector.

B. GAN Architecture

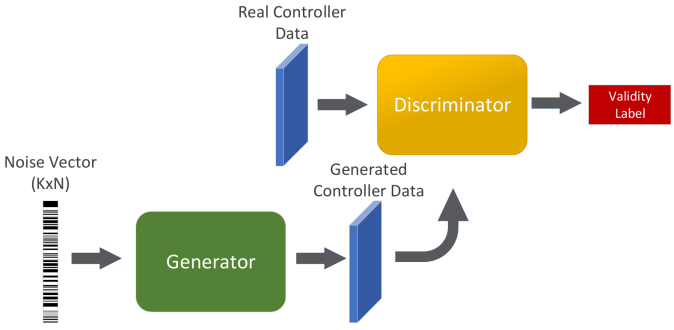


Fig. 2: GAN Architecture

The GAN architecture [9] [18], 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) shows the relationship between the Generator and the Discriminator (denoted as G and D, respectively) as a value function.

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

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, GAN consists of three different components. The first component is the Input module, the second is the Generator module and the third is the Discriminator module.

Input Module: The Input module of 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 GAN, the Generator is composed of seven layers and it is compiled with the Binary Crossentropy loss function (3) and the Adam Optimizer [19].

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

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.

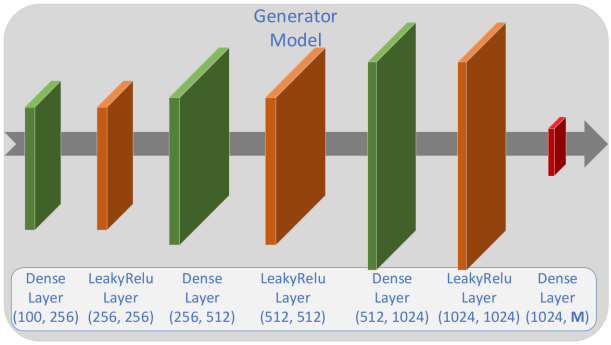


Fig. 3: Generator Module Architecture

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 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 Rectified Linear Units (ReLU) layers that follow the first, second and third dense layers.

Discriminator Module: The second neural sub-network, namely the Discriminator, 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.

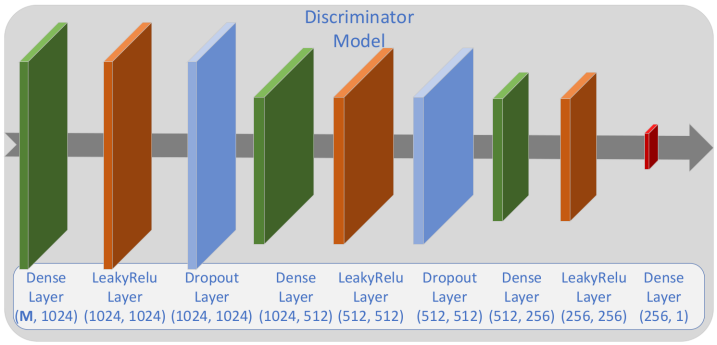


Fig. 4: Discriminator Module Architecture

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 having 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 [20]. 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.

C. Auto-Encoder Architecture

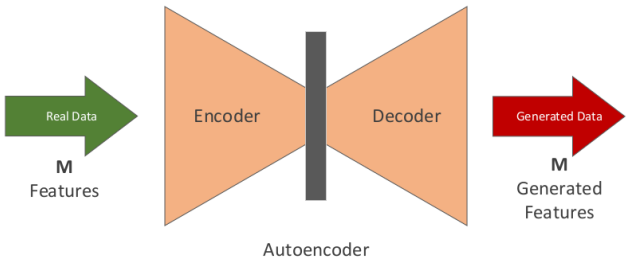


Fig. 5: Autoencoder Architecture

The basic concept of the Auto-Encoder is the assimilation of the 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 predefined output size and forward the output to the Decoder for scaling. The architecture of the

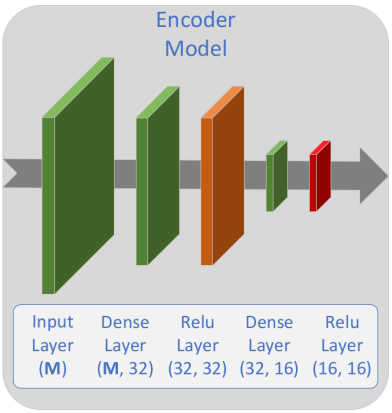


Fig. 6: Encoder Module Architecture

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 utilise 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 almost identical to the real data. Fig. 7 depicts the architecture

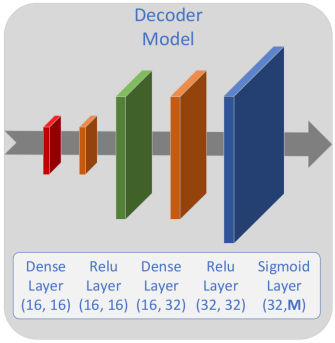


Fig. 7: Decoder Model

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 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.

D. 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 consists of two parts. Firstly, the proposed DNNs are compared in order to evaluate the accuracy of the results. Secondly, the required time for traffic generation is measured.

A. Accuracy

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 Frechet Inception Distance (FID) [21], [22] score. The FID score is calculated as:

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

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.

Fig. 8 and Fig. 9 depict the similarity between the generated and real data values in terms of arithmetic mean and standard deviation. In particular, Fig. 8 shows the arithmetic mean of the features, ranging from 1 to 45. Both of the approaches achieve a high overall similarity to the real values, while the GAN achieves a slightly better similarity.

Similarly, Fig. 9 shows the standard deviation of the feature values, ranging from 1 to 45. Both approaches achieve a high overall similarity. However, in this case, the Autoencoder achieves a slightly better similarity.

Fig. 10 presents a similarity comparison between the data generated by the Autoencoder and the GAN, as well as the real dataset. The graphs were rendered by plotting the numerical values of 43 features. According to the graphs, the Autoencoder has higher similarity to the real dataset, compared to the GAN. In addition, using equation (6), Autoencoder achieves a FID score of 29.94, while GAN achieves a FID score of 31.29. As a smaller FID score indicates higher similarity, Autoencoder performs better compared to GAN.

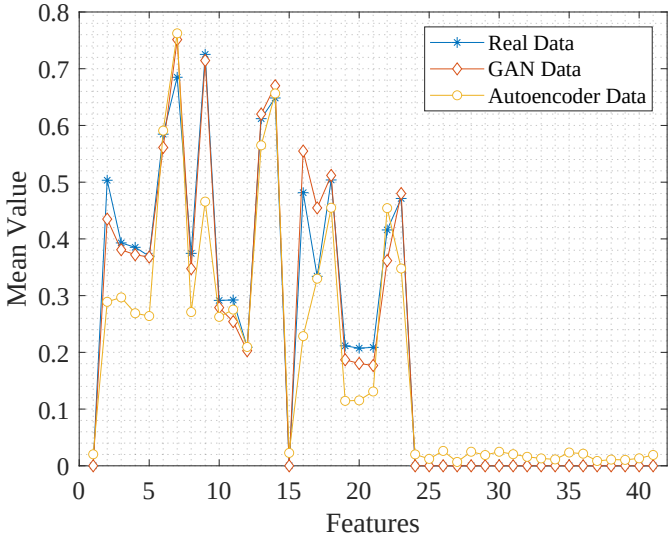


Fig. 8: Arithmetic mean of the data

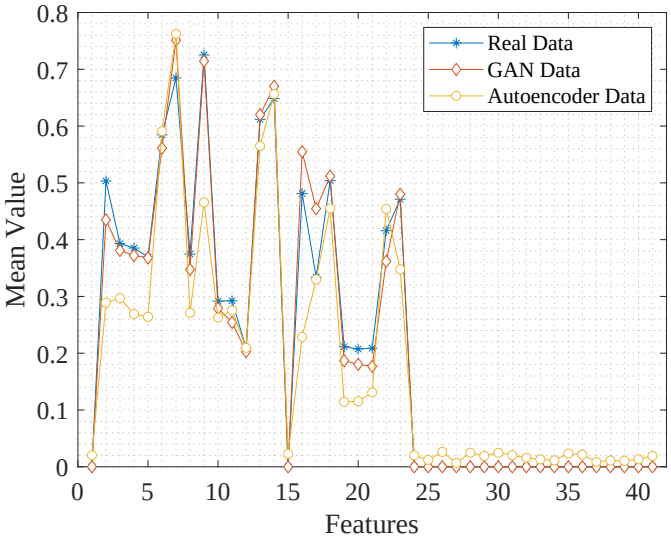
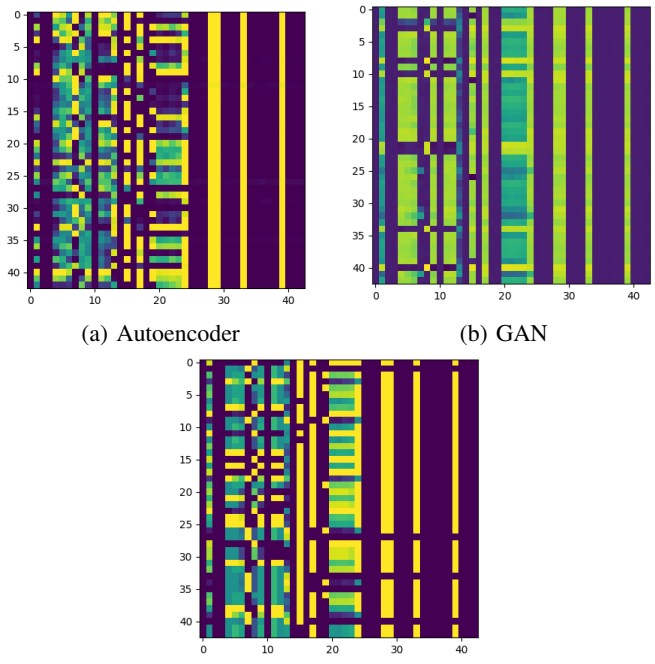


Fig. 9: Standard deviation of the data

B. Time

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 having a 16GB of RAM to its disposal.

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 (i.e., 500 ms, as defined in [23]), therefore both approaches can be effectively used for network traffic generation in real-time.



(a) Autoencoder (b) GAN

(c) Real

Fig. 10: Visualized Data using the FID function

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, namely Autoencoder and GAN, are developed and 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 in order to evaluate its efficiency in attracting attackers and record their behavior. Furthermore, we aim to incorporate additional well known ICS communication protocols.

Acknowledgment

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

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

[7] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8599–8603.

[8] Y. Yu, H. Lin, Q. Yu, J. Meng, Z. Zhao, Y. Li, and L. Zuo, "Modality classification for medical images using multiple deep convolutional neural networks," *J. Comput. Inf. Syst*, vol. 11, no. 15, pp. 5403–5413, 2015.

[9] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[10] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49.

[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] 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.

[14] 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.

[15] 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.

[16] 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 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2019, pp. 1–6.

[17] 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, 2008.

[18] Y. Hong, U. Hwang, J. Yoo, and S. Yoon, "How generative adversarial networks and their variants work: An overview," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, p. 10, 2019.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] S. Barratt and R. Sharma, "A note on the inception score," *arXiv preprint arXiv:1801.01973*, 2018.

[22] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *arXiv preprint arXiv:1805.08318*, 2018.

[23] "Modbus Message Timing - Continental Control Systems, LLC." [Online]. Available: https://ctlsys.com/support/modbus$_message_timing/$