



# **NAVAL POSTGRADUATE SCHOOL**

**MONTEREY, CALIFORNIA**

## **THESIS**

**USING GENERATIVE ADVERSARIAL NETWORKS FOR  
INTRUSION DETECTION IN CYBER-PHYSICAL SYSTEMS**

by

Jessica L. Purser

September 2020

Thesis Advisor:  
Co-Advisor:

Thuy D. Nguyen  
Neil C. Rowe

**Approved for public release. Distribution is unlimited.**

THIS PAGE INTENTIONALLY LEFT BLANK

<b>REPORT DOCUMENTATION PAGE</b>			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC, 20503.				
<b>1. AGENCY USE ONLY</b> (Leave blank)	<b>2. REPORT DATE</b> September 2020	<b>3. REPORT TYPE AND DATES COVERED</b> Master's thesis		
<b>4. TITLE AND SUBTITLE</b> USING GENERATIVE ADVERSARIAL NETWORKS FOR INTRUSION DETECTION IN CYBER-PHYSICAL SYSTEMS			<b>5. FUNDING NUMBERS</b>	
<b>6. AUTHOR(S)</b> Jessica L. Purser				
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Naval Postgraduate School Monterey, CA 93943-5000			<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>	
<b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> N/A			<b>10. SPONSORING / MONITORING AGENCY REPORT NUMBER</b>	
<b>11. SUPPLEMENTARY NOTES</b> The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
<b>12a. DISTRIBUTION / AVAILABILITY STATEMENT</b> Approved for public release. Distribution is unlimited.			<b>12b. DISTRIBUTION CODE</b> A	
<b>13. ABSTRACT (maximum 200 words)</b>  Cyber-physical systems (CPS) are widely used in mission-critical systems in the Department of Defense and the U.S. Navy. They also form the backbone of national critical infrastructure. However, CPS technologies often sacrifice security in exchange for increased availability and efficiency, thus becoming prominent targets in cyber-warfare. This thesis explored machine learning to develop training examples for intrusion-detection systems on cyber-physical systems. We developed two generative adversarial network (GAN) models and assessed their ability to generate and detect anomalous traffic at the packet level. We tested two CPS datasets that included attacks that exploit commonly known vulnerabilities in Internet-of-Things networks and industrial control systems. The results confirmed that a GAN could improve the performance of intrusion-detection systems for detecting anomalous CPS traffic.				
<b>14. SUBJECT TERMS</b> intrusion detection, cyber-physical systems, generative adversarial networks, network security, machine learning, anomaly detection			<b>15. NUMBER OF PAGES</b> 97	
			<b>16. PRICE CODE</b>	
<b>17. SECURITY CLASSIFICATION OF REPORT</b> Unclassified	<b>18. SECURITY CLASSIFICATION OF THIS PAGE</b> Unclassified	<b>19. SECURITY CLASSIFICATION OF ABSTRACT</b> Unclassified	<b>20. LIMITATION OF ABSTRACT</b>  UU	

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release. Distribution is unlimited.**

**USING GENERATIVE ADVERSARIAL NETWORKS  
FOR INTRUSION DETECTION IN CYBER-PHYSICAL SYSTEMS**

Jessica L. Purser  
Civilian, CyberCorps – Scholarship For Service  
BA, University of California, Santa Cruz, 2017

Submitted in partial fulfillment of the  
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL  
September 2020**

Approved by: Thuy D. Nguyen  
Advisor

Neil C. Rowe  
Co-Advisor

Gurminder Singh  
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

## **ABSTRACT**

Cyber-physical systems (CPS) are widely used in mission-critical systems in the Department of Defense and the U.S. Navy. They also form the backbone of national critical infrastructure. However, CPS technologies often sacrifice security in exchange for increased availability and efficiency, thus becoming prominent targets in cyber-warfare. This thesis explored machine learning to develop training examples for intrusion-detection systems on cyber-physical systems. We developed two generative adversarial network (GAN) models and assessed their ability to generate and detect anomalous traffic at the packet level. We tested two CPS datasets that included attacks that exploit commonly known vulnerabilities in Internet-of-Things networks and industrial control systems. The results confirmed that a GAN could improve the performance of intrusion-detection systems for detecting anomalous CPS traffic.

THIS PAGE INTENTIONALLY LEFT BLANK



# TABLE OF CONTENTS

<b>I.</b>	<b>INTRODUCTION.....</b>	<b>1</b>
<b>A.</b>	<b>MOTIVATION .....</b>	<b>1</b>
<b>B.</b>	<b>RESEARCH PLAN .....</b>	<b>2</b>
<b>C.</b>	<b>THESIS ORGANIZATION.....</b>	<b>3</b>
<b>II.</b>	<b>BACKGROUND .....</b>	<b>5</b>
<b>A.</b>	<b>NEURAL NETWORKS .....</b>	<b>5</b>
1.	Neural Network Basics .....	5
2.	Types of Neural Networks.....	6
3.	Applying Neural Networks to Intrusion-Detection.....	6
4.	Generative Adversarial Networks .....	8
5.	Generative Adversarial Networks in Intrusion Detection .....	10
<b>B.</b>	<b>CYBER-PHYSICAL SYSTEMS.....</b>	<b>12</b>
1.	Internet of Things .....	12
2.	Industrial Control Systems .....	12
<b>C.</b>	<b>INTRUSION-DETECTION SYSTEMS.....</b>	<b>14</b>
<b>III.</b>	<b>PROBLEM DESCRIPTION AND APPROACH .....</b>	<b>17</b>
<b>A.</b>	<b>CYBER-PHYSICAL SYSTEM SECURITY .....</b>	<b>17</b>
1.	Security Issues .....	17
2.	Attack Methods .....	18
<b>B.</b>	<b>MODEL DEVELOPMENT .....</b>	<b>19</b>
1.	Long Short-Term Memory in Packet-Level Classification .....	20
2.	Generative Adversarial Networks as an Intrusion- Detection Method .....	20
3.	Methods.....	21
<b>C.</b>	<b>DATASETS .....</b>	<b>21</b>
1.	Industrial Control System Instructional Cybersecurity Lab Dataset.....	21
2.	Internet of Things Dataset.....	22
<b>IV.</b>	<b>EXPERIMENTS .....</b>	<b>25</b>
<b>A.</b>	<b>MODEL DESIGN .....</b>	<b>25</b>
<b>B.</b>	<b>DATA PREPARATION.....</b>	<b>26</b>
1.	Packet Structure.....	26
2.	Preprocessor .....	27
3.	Preprocessor's Workflow .....	28

4.	Selector .....	31
C.	CPSGAN .....	32
1.	Generator .....	32
2.	Discriminator .....	34
3.	Training .....	36
D.	EXPERIMENTS .....	40
1.	Analysis Methods .....	40
2.	Principal-Component Analysis and K-Means Clustering .....	41
3.	CPSGAN Evaluation .....	42
V.	RESULTS AND ANALYSIS .....	45
A.	PRINCIPAL-COMPONENT ANALYSIS AND K-MEANS CLUSTERING .....	45
B.	CPSGAN RESULTS .....	47
1.	Experiment 1: Evaluation of the Discriminators on Real Data .....	47
2.	Experiment 2: Evaluation of the Discriminators Across Datasets .....	50
3.	Experiment 3: Training CPSGAN and Analysis of Model Loss .....	52
4.	Experiment 4: Evaluation of GAN-Trained Discriminators on Real Data .....	56
5.	Experiment 5: Evaluation of Pretrained Standalone Discriminators on Generated Data .....	58
VI.	CONCLUSIONS AND FUTURE WORK .....	63
A.	SUMMARY OF FINDINGS .....	63
B.	FUTURE WORK .....	63
	APPENDIX A. CIP TRAFFIC EXAMPLES .....	65
	APPENDIX B. WIRESHARK FILTERS .....	67
	APPENDIX C. CPSGAN MODELS .....	69
	LIST OF REFERENCES .....	71
	INITIAL DISTRIBUTION LIST .....	79

## LIST OF FIGURES

Figure 1.	The Design of a Generative Adversarial Network. Adapted from: [27].	9
Figure 2.	The Design of a Bidirectional Generative Adversarial Network. Source: [27].	10
Figure 3.	Overview of the CPSGAN Training Program	25
Figure 4.	Packet Structure with Emphasis on the IP Header Fields. Adapted from: [62].	26
Figure 5.	Behavior of the Selector.	29
Figure 6.	Step 1 of the CPSGAN Training Procedure	37
Figure 7.	Step 2 of the CPSGAN Training Procedure	38
Figure 8.	Step 3 of the CPSGAN Training Procedure	39
Figure 9.	PCA and K-Means Clustering on the ICSICL Dataset (K=6)	46
Figure 10.	PCA and K-Means Clustering on the IoT Dataset (K=10)	46
Figure 11.	Experiments 1A and 1B: ROC Curves of Standalone Discriminators	49
Figure 12.	Experiments 1A and 1B: Precision-Recall Curves of the Standalone Discriminators	50
Figure 13.	Experiments 2A and 2B: Precision-Recall Curves of Standalone Discriminators on Combined Dataset	52
Figure 14.	Experiments 3A and 3B: LSTM Discriminator Losses on Real and Fake Data	54
Figure 15.	Experiments 3A and 3B: LSTM Discriminator and Generator Losses	54
Figure 16.	Experiments 3A and 3B: MLP Discriminator Losses on Real and Fake Data	55
Figure 17.	Experiments 3A and 3B: MLP Discriminator and Generator Losses	56
Figure 18.	Experiments 4A and 4B: Precision-Recall Curves for GAN-Trained Discriminators	58
Figure 19.	Experiments 5A and 5B: Precision-Recall of the Experiment 1 Discriminators Evaluated on Generated Data	61

Figure 20.	Example of Benign CIP Traffic .....	65
Figure 21.	Denial of Service Using the Register Session Command.....	66
Figure 22.	Application of a TShark Rule in a Command-line Interface .....	67

## LIST OF TABLES

Table 1.	ICSICL Dataset: Malicious Traffic Composition .....	22
Table 2.	IoT Dataset: Malicious Traffic Composition .....	23
Table 3.	Parsed Header Fields used in CPSGAN .....	27
Table 4.	Binning of IP Addresses .....	30
Table 5.	Binning of Port Ranges .....	30
Table 6.	Total Number of Features in the ICSICL and IoT Datasets .....	31
Table 7.	Long Short-Term Memory Generator Parameters .....	33
Table 8.	Multilayer Perceptron Generator Parameters .....	34
Table 9.	Long Short-Term Memory Discriminator Parameters .....	35
Table 10.	Multilayer Perceptron Discriminator Parameters .....	36
Table 11.	Quantifying the Discriminator's Predictions .....	40
Table 12.	Model Testing Plan .....	43
Table 13.	Experiments 1A and 1B: Frequency Statistics of Standalone Discriminators .....	48
Table 14.	Experiments 2A and 2B: Frequency Statistics of Standalone Discriminators on Combined Dataset .....	51
Table 15.	Experiments 4A and 4B: Frequency Statistics of GAN-Trained Discriminators .....	57
Table 16.	Experiments 4A and 4B: Performance Metrics of GAN-Trained Discriminators .....	57
Table 17.	Experiments 5A and 5B: Frequency Statistics of Standalone Discriminators Evaluated on Benign and Generated Data .....	59
Table 18.	Experiments 5A and 5B: Performance Metrics of the Standalone Discriminators Evaluated on Generated Data .....	60
Table 19.	Creating a Benign Filter from Multiple Attack Filters .....	68

THIS PAGE INTENTIONALLY LEFT BLANK

## LIST OF ACRONYMS AND ABBREVIATIONS

AP	average precision
APT	advanced persistent threat
ARP	Address Resolution Protocol
AUC	area under the curve
CIP	Common Industrial Protocol
CNN	convolutional neural network
CPS	cyber-physical systems
CPSGAN	cyber-physical systems generative adversarial network
CSV	comma-separated values
DNS	Domain Name Service
DoS	denial of service
ENIP	EtherNet/IP (Industrial Protocol)
FTP	File Transfer Protocol
GAN	generative adversarial network
HCRL	Hacking Countermeasure Research Lab
HMI	human-machine interface
HTTP	Hypertext Transfer Protocol
ICSICL	Industrial Control System Instructional Cybersecurity Lab
ICS	industrial control system
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
LSTM	long short-term memory
MITM	man in the middle
MLP	multilayer perceptron
NPS	Naval Postgraduate School
PCA	principal-component analysis
PLC	programmable logic controller
RFC	Request for Comments

ROC	receiver operating characteristic
SCADA	supervisory control and data acquisition
TCP	Transport Control Protocol
UDP	User Datagram Protocol
WGAN	Wasserstein generative adversarial network



## **ACKNOWLEDGMENTS**

The pandemic, riots, political turmoil, wildfires, and writing a machine-learning thesis has made 2020 quite an interesting year. I'm beyond ecstatic that my cohort and I have made it through.

I would like to sincerely thank both of my advisors, Professor Thuy Nguyen and Dr. Neil Rowe. Their advice extended far beyond the technical aspects of cyber-physical systems and machine learning. Their guidance was crucial to my growth as a scientist.

To Mom, my sister Theresa, and my partner Andrew: I love you. Thank you for putting your full faith in me and sending me surprise sushi.

I've made lifelong friends in my cohort, and I'm excited to see what's in store for us. I also want to thank Dr. Irvine, Cecelia Davis, and LCDR Eric Regnier for their priceless advice and kind words throughout this whole process.

This material is based upon activities supported by the National Science Foundation under Agreement No. 1565443. Any opinions, findings, and conclusions or recommendations expressed are those of the author and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

# **I. INTRODUCTION**

The security of cyber-physical systems (CPS) is a growing concern as these networked systems provide important functions and services to society. Two main subcategories of cyber-physical systems are industrial control systems (ICS) and the Internet of Things (IoT) [1]. Millions of citizens rely on industrial control systems to provide for needs such as water, food, and electricity. Consumerism is experiencing an explosion in IoT devices in the form of smart homes, virtual assistants like Alexa or Google Home, networked appliances, and other similar products. Because of the conveniences and needs they serve, cyber-physical systems and devices have become deeply integrated into daily life, industry, and critical functions of the government. For this same reason, cyber-physical systems have drawn attention as potential targets by both cybersecurity experts and malicious adversaries.

## **A. MOTIVATION**

Much critical infrastructure in the United States has migrated to cyberspace without sufficient security considerations, and continues to be managed with inadequate protection against cyber threats. Consumer products are no different, as many IoT devices are developed and sold to the public with default security settings or using protocols that are well-known to be vulnerable [2]. Furthermore, dangers of these systems are not limited to the leakage of information. Insecure infrastructure could harm the public if it were suspended, destroyed, or controlled by unauthorized actors. As an example, the Northeast Blackout of 2003 lasted two days, left 50 million people without power, caused at least 11 fatalities, and cost \$6 billion. This event was a result of human error and faulty equipment [3]. Considering this was the cost of mishandling alone, malicious actors could clearly cause more harm to critical infrastructure.

Attacks on critical infrastructure are not rare. In 2008, the Stuxnet worm successfully destroyed the centrifuges in an Iranian nuclear facility and gained international attention [4]. This event set a precedent in cyber-weaponization and created an awareness that much of this critical infrastructure is open to foreign targeting. In 2018, the FBI

announced its detection of Russian cyber actors attacking the U.S. electrical grid and aviation facilities [5]. Since 2016, the Mirai worm has targeted insecure IoT devices to launch large-scale, distributed denial-of-service attacks [6]. When it was first discovered, traffic records indicated that the Mirai botnet had enlisted hundreds of thousands of compromised IoT devices [7].

Due to our considerable dependence on critical infrastructure, disruption to its operation risks a high cost to the public. The security requirements of cyber-physical systems diverge significantly from the security requirements of traditional computing environments due to unique properties of their proprietary protocols and how humans interact with them. Time-critical safety measures are especially important for industrial control systems. Compared to the security of general-purpose computing and networking systems, the security of cyber-physical systems is lagging.

Machine learning has recently become popular due to its many methods and their versatility on a broad variety of data. Many intrusion-detection methods use machine learning to understand new trends in network traffic and recognize deviations from normal behavior [8]. The method explored in this thesis, called a generative adversarial network (GAN), can create useful instances of data for training and testing many applications, including intrusion-detection systems for computing environments [9].

## **B. RESEARCH PLAN**

This thesis explores a GAN framework for cyber-physical systems in which an attacker (the generator) tries to evade detection and generates malicious data that an intrusion-detection system (the discriminator) will consider as legitimate. The proposed framework, CPSGAN, uses machine learning to improve both the generator and discriminator, and simulates a minimax game between them [10]. Minimax games incentivize each player to minimize the reward of their opponent with incomplete information. Ideally, the adaptive learning of these two players develops a better discriminator for defending computer systems, which may reveal obscurities or vulnerabilities about the system in question.

The generator tries to emulate and reproduce behavior classified as non-malicious by the discriminator, which we assume uses technology unknown to the generator. In the proposed model, the discriminator provides feedback to the generator about the generator's success. Upon receiving this new information, the generator alters its methods. Over time, the generator learns what kind of traffic the discriminator will accept, which acts as an intrusion-detection system.

This research investigates different approaches to implement the CPSGAN framework. It used long short-term memory (LSTM) neural networks (a specialization of the traditional recurrent neural network) and multilayer perceptron (MLP) neural networks to prototype the two players in the GAN. Two data corpora tested the efficacy of the prototype implementations: an IoT dataset [11] and an industrial network traffic dataset from a research system [12].

### **C. THESIS ORGANIZATION**

The rest of this thesis is structured as follows. Chapter II surveys GAN technology, its use in intrusion-detection systems, and the application of similar neural networks in cyber-physical systems. Chapter III describes the threat model, the development of the CPSGAN model, and the data for training and testing the generator and discriminator. Chapter IV discusses the experimental process and Chapter V shows the results and performance analysis of the different CPSGAN prototypes. Finally, Chapter VI summarizes this research and recommends future research.

THIS PAGE INTENTIONALLY LEFT BLANK

## **II. BACKGROUND**

This chapter surveys machine-learning methods, generative adversarial networks (GANs), intrusion-detection systems, and cyber-physical system protocols. It also reviews previous work on using generative adversarial networks in computer networking.

### **A. NEURAL NETWORKS**

#### **1. Neural Network Basics**

Because they can overcome the disadvantages of existing intrusion-detection methods, researchers are exploring artificial neural-network models in cybersecurity applications to catch novel attacks [13], [14]. An artificial neural network is modeled on the neural connections in the brain and contains simulated neurons that form layers. They are also called multilayer perceptron networks [15]. Neural networks usually do supervised learning, meaning that they learn from training examples.

Each neuron has weights on its inputs, which are the most important parameters that are varied to allow neural networks to learn. To train a neural network, input is subjected to computations in successive layers of neurons until the last layer where a numeric prediction is made. We measure how close the prediction is to the correct value and use this as training feedback. This feedback travels back through the network and changes the neuron weights so that the network can be more accurate in the future. These processes are called forward and backward propagation respectively. In backward propagation, the feedback is called “error” and updating the weights uses a technique called “gradient descent.” In supervised learning, error is computed from labels, which are the correct values assigned to samples in a training dataset.

Before running a neural network, features in the dataset must be chosen that are relevant to classification. In intrusion-detection research, feature-selection effectiveness can be limited by the developers’ understanding of new attack vectors [14], [16]. This means feature selection can be costly and difficult to manage, requiring human expertise and detailed analysis.

## **2. Types of Neural Networks**

Many neural-network architectures have been tested in intrusion-detection research, including convolutional neural networks, long short-term memory networks, and autoencoders. A convolutional neural network is often used for computer vision and image processing. The early layers in these networks focus on small local parts of the input, and the later layers combine the local parts [17]. Convolutional neural networks could be helpful with network packet data because much analysis is needed on small parts of a packet before the whole packet can be understood. This architecture can process individual packets in series data or segments simultaneously.

Another consideration is a recurrent neural network, which can remember inputs it has previously processed and use them to improve future predictions. A long short-term memory (LSTM) network is a type of recurrent neural network used extensively in natural-language processing. Also called sequence models, LSTM networks were created to have more persistent memory. This characteristic helps to recognize network attacks that are timing-based, such as flooding attacks [6], [18], [19].

Unsupervised machine learning could be used to recognize novel attacks. One approach is a neural network called an autoencoder, which learns how to efficiently encode its input [20]. The first half of its layers comprise an encoder, and the rest comprise a decoder. When learning, the autoencoder adds noise to the input or transforms it. Then, the autoencoder reconstructs, or decodes, the encoded input [21]. If the result differs from the original input, the autoencoder's weights are changed. To be successful, the hidden layers must be adjusted so that the autoencoder can distinguish between the added noise and the more important aspects of the dataset. Autoencoders were originally designed for feature learning and dimensionality reduction. They have also shown promise as generative neural networks, which falls into the same class as GANs [22].

## **3. Applying Neural Networks to Intrusion-Detection**

Autoencoder neural networks have been applied to attacks on smart-grid technology in [16], where a stacked autoencoder network dynamically identified new supervisory control and data acquisition (SCADA) system exploits. Adaptability is key to



power-grid modernization, which involves restructuring antiquated infrastructure and reducing dependency on human operators and developers as much as possible. The stacked autoencoder consists of multiple autoencoders, which undergo an unsupervised pre-training phase and are fine-tuned with a multilayer perceptron classifier at the final layer. Once trained, the autoencoder network can be deployed online with real-time data for dynamic security monitoring. To assess performance, accuracy can be used, which is the proportion of correct predictions over total instances tested. This approach outperformed a standalone multilayer perceptron classifier and achieved over 98% accuracy in three out of four event types: normal operation with load variation, data injection, and remote tripping command injection. With the fourth attack type, relay setting change, the stacked autoencoder achieved 94.91% while the multilayer perceptron classifier scored around 96%.

Another project applied convolutional neural network (CNN) and long short-term memory (LSTM) architecture to data collected from a secure water treatment testbed [14]. Their models included several designs of one-dimensional CNNs, a LSTM network, and a combined model using both architectures. CNNs and LSTMs can learn important characteristics in time-series data [17], [23]. As industrial traffic behavior is more predictable than standard network traffic, one could argue that using CNNs and LSTMs may work well for this purpose. In this study, the convolutional neural network did best, detecting 32 of the 36 attacks examined. By some metrics, performance was better than that of a support-vector machine and a graphical model-based approach on the same data [14]. The authors showed that their design can be integrated into other ICS intrusion-detection systems while achieving faster and more accurate results than recurrent network implementations.

In a different study, researchers designed a LSTM network and a combined CNN-LSTM network for payload-based traffic classification [19]. Payload data was split into four-bit slices, organized into sequences, then input to the models. The authors argued that the combined CNN-LSTM model would do better than the LSTM-only model. Results showed that for all payload sizes, the LSTM model achieved higher accuracy than the

combined CNN-LSTM. Other studies have also used LSTM networks for network traffic data [6], [18], [24].

Another promising method involved a CNN to classify industrial traffic data [25]. The CNN's performance was compared to a support-vector ensemble method, a hidden Markov model, and decision-tree method. The authors compared the ability of the models to detect six types of ICS attacks, including denial of service, reconnaissance, and several subcategories of injection attacks. They achieved the highest recall for five of the seven classifications: normal traffic, complex malicious response injections, malicious state command injections, parameter command injections, and reconnaissance. The precision of the support vector machine, Hidden Markov Model, and decision tree classifiers were 94.5%, 93.4%, and 93.1% respectively, while the CNN achieved 99.3% precision.

#### **4. Generative Adversarial Networks**

The generative adversarial network was first introduced as a minimax game between two neural networks: a discriminator and a generator [10]. The discriminator is a binary classifier that guesses the probability that a sample is from a given classification. The generator takes noise, modifies it, and gives the result to the discriminator to classify. The purpose of the generator is to learn what the true dataset may look like, which it does by observing how the discriminator responds to its generated guesses. If the discriminator is not fooled, the generator is penalized by backpropagation. This reinforces the generator to make more specific changes to its noise inputs. If the generator is effective at fooling the discriminator, the discriminator is penalized in a similar manner.

The generator does not do very well in early stages of training, especially if the dataset contains complex samples, such as images. Since the generator has no knowledge of the dataset, it outputs random guesses. Initially, the discriminator can easily distinguish between the between the generator's weak guess and a real sample. As the generator's guesswork begins to improve and resemble real samples from the dataset, the discriminator may misclassify the generator's fakes. When this happens, the discriminator incurs higher losses, which force it to learn how to recognize samples from the generator, instead of the real dataset. The generator reacts to the discriminator's improved performance by creating

higher-quality samples, and so on. Ideally, training concludes when the discriminator is classifying with 50% accuracy and neither network can improve.

This adversarial process is called a minimax game, a model of strategic tension between two decision-makers in a competitive environment [26]. In a minimax game, players are risk-averse and make choices that minimize their maximum potential loss. The original GAN model involved two competing multilayer perceptron classifiers, but effective GAN designs can use any neural network model [22].

Figure 1 shows the basic structure of the GAN. The generator  $G$  takes a noise vector  $z$  as input to create a fake instance,  $G(z)$ . The fake instance  $G(z)$  and a real sample  $x$  are input to the discriminator  $D$ . The discriminator determines the probability that the instance is real, represented as  $P(y)$ .

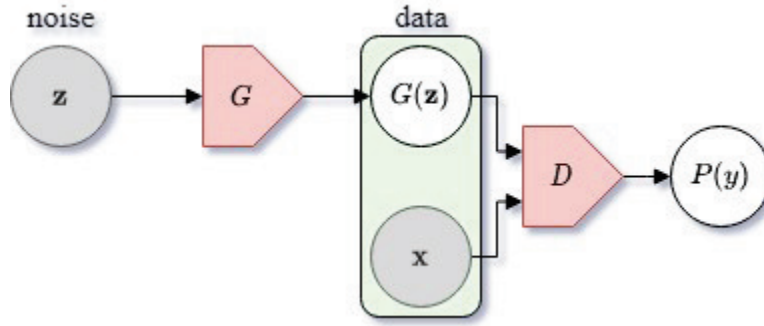


Figure 1. The Design of a Generative Adversarial Network.  
Adapted from: [27].

Bidirectional generative adversarial networks (BiGAN) include an encoder that feeds a sample and its encoding to the discriminator [27]. The encoder is a separate component that “mirrors” the generator’s structure, where its layers are in reverse order. The discriminator acts as an interface between the generator and encoder during training, as in Figure 2. In this case, the encoder  $E$  takes real sample  $x$  to create the encoding  $E(x)$ , which are both input to the discriminator as a real pairing  $(x, E(x))$ . Noise and the sample generated from the generator is input to the discriminator as a fake pairing  $(G(z), z)$ .

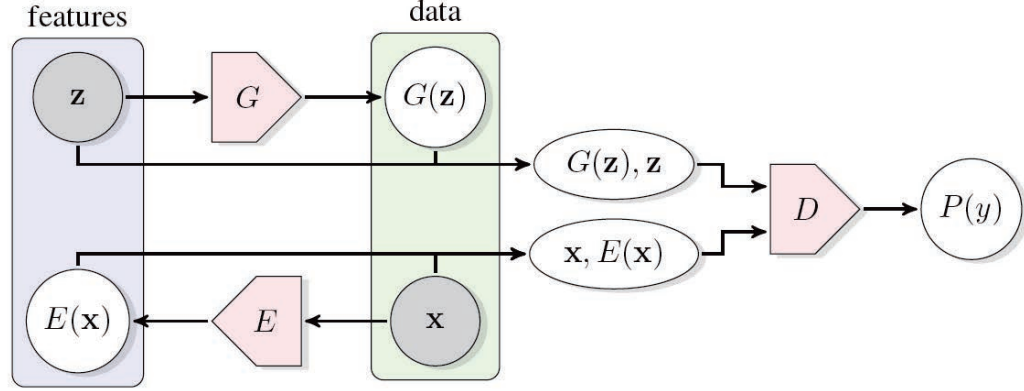


Figure 2. The Design of a Bidirectional Generative Adversarial Network.  
Source: [27].

The encoder provides the benefit of teaching the discriminator the probability distributions of both the real and generated samples. Without a direct connection between the generator and encoder, the two must learn the inverse of each other's distributions solely through responses from discriminator. The rigid constraints placed on the generator and encoder enable feature learning and emphasize anomalous patterns in the data.

## 5. Generative Adversarial Networks in Intrusion Detection

AnoGAN was the first generative adversarial network to be proposed for anomaly detection in image data [28], and was applied to GAN-based intrusion detection in [29]. AnoGAN was trained on benign data, then evaluated on benign and diseased medical image data. The generator received information about a real instance and, if this was anomalous, it would learn to produce a fake benign instance using a technique called feature matching. The fake was contrasted with the original input, which highlighted specific regions of disease in the original input. A major disadvantage is that AnoGAN required a time-consuming backpropagation process to define the mapping for anomaly scoring and reconstruction of the image [30].

The efficient GAN-Based Anomaly Detection (EGBAD) project used a BiGAN design to improve upon the performance of AnoGAN [29]. Both AnoGAN and EGBAD were evaluated on the KDD-99 network intrusion dataset [31]. The BiGAN encoder

eliminated much of the computation required for the reconstruction and scoring process that originally hindered AnoGAN’s training speed. This also increased binary classification performance. To evaluate the model, the authors used F1-scoring, which accounts for false positives and false negatives more than raw accuracy and provides a clearer estimation of performance. Whereas AnoGAN achieved an F1-score of 78.52%, EGBAD achieved 93.72% [28].

In IDSGAN, the generator created malicious instances of traffic and the discriminator received responses from a simulated intrusion-detection system as input. The model was tested on the NSL-KDD dataset [32]. It was implemented based on a generative adversarial network using the performance metric of Wasserstein loss [33], which produces an “authenticity” score instead of a probability that an instance belongs to the real dataset. In this case, authenticity measures how close to the true distribution the discriminator believes the sample to be. This gives the generator much more specific information about how to adjust its weights and has the advantage of creating a more stable model [34]. The intrusion-detection system was simulated with one of six machine-learning algorithms: a support-vector machine, Naive Bayes, linear programming, logistic regression, random forest, and K-nearest neighbors.

The generative adversarial network intrusion-detection system (GIDS) was implemented for raw controller area network (CAN) traffic data [13]. The CAN protocol is a bus protocol used in automotive vehicles and it resembles industrial control system protocols. The authors tested two discriminators. The first discriminator trained on real traffic data, both normal and malicious. The second discriminator trained on both real and fake data from the generator, which took a combination of noise and normal traffic data as input. To assess performance, the authors used accuracy, which is the proportion of correct predictions over total instances tested. Results showed that the first discriminator detected over 99% of the attack data it was tested on, while the second discriminator achieved over 98% accuracy for attacks that were previously unknown to the first discriminator. This suggests that combining the two discriminators in a defense-in-depth framework would bring accuracy closer to 100% in an operational setting.

## **B. CYBER-PHYSICAL SYSTEMS**

Cyber-physical systems (CPSs) integrate networked physical devices and are distinct from general-purpose computing resources. These systems provide many commodities and services with minimal human intervention [35]. CPSs are abundant in daily life and many common examples exist, such as traffic lights, home security systems, and hospital medical equipment. This area can be further categorized into two major domains: the Internet of Things (IoT) and the Industrial Internet. IoT is a broad term that covers billions of cyber-physical devices online today. The Industrial Internet, which uses industrial control systems, includes systems with automated control and production that support economic activity.

### **1. Internet of Things**

The definition of IoT covers an expansive and fast-growing system of networked and embedded technologies. In contrast with general-purpose computers, IoT devices handle a discrete set of functions and can be controlled through commands from people, sometimes remotely. Unless given explicit instructions to do otherwise, IoT devices operate autonomously and use networking protocols to report status updates to human operators. In many cases, these devices collaborate with each other to achieve some physical objective, as we see in industrial control processes [35].

Some IoT devices communicate using telemetry or proprietary protocols built on standard network protocols such as the Transport Control Protocol (TCP) or the User Datagram Protocol (UDP) [36].

### **2. Industrial Control Systems**

An industrial control system (ICS) consists of physical actuators that are controlled to achieve some industrial objective, such as manufacturing, energy production, or transportation. The term includes supervisory control and data acquisition systems (SCADA), distributed control systems, and configurations involving programmable logic controllers (PLC). One dataset in this thesis focuses on a SCADA testbed.

A SCADA system connects, monitors, and coordinates the control processes of geographically distributed systems, which themselves may be distributed control systems. SCADAs are like the brain of the ICS body, and the individual systems it commands constitute its limbs. PLCs collect data from sensors, control actuators, and can enable time-sensitive operations and communications across multiple devices. Their main purpose is to manage the components affecting a physical process. ICS sensory and device data is organized, translated, and communicated between components through a comprehensive suite of proprietary protocols. It is also stored for retroactive analysis. To provide control at a high level, a human-machine interface (HMI) system communicates with the PLC, receives status updates, sends commands to the PLC, or dynamically configures the PLC's control logic.

The most important design considerations of ICS are timing and control complexity, the organizational hierarchy, availability (or reliability), robustness of the system when operating in a degraded state or upon system failures, and the ability to detect and reduce unsafe conditions [36].

#### *a. Industrial Protocols*

One dataset in this thesis was from a testbed using two related protocols: the Ethernet Industrial Protocol (EtherNet/IP, also called ENIP) and the Common Industrial Protocol (CIP). EtherNet/IP is an application layer protocol that supports industrial-automation technologies with the Ethernet protocol (IEEE 802.3 using the TCP/IP suite) [37]. Its main functions are the organization of data and encapsulation over the lower network layers. It supports seamless end-to-end connectivity between components on a network [38]. EtherNet/IP belongs to a family of protocols based on the Common Industrial Protocol (CIP), which include ControlNet, DeviceNet, and CompoNet. Each supports a different type of network or set of operational needs, and all four are supported and managed by Open DeviceNet Vendors Association.

CIP uses an object-oriented design to organize industrial-application data such as device profiles or statuses. CIP uses several transport protocols, including ENIP. This

protocol follows a producer-consumer model, is media-independent, and was developed to provide flexible integration of different network architectures.

In CIP, device data is organized as an object, and each device is represented as a collection of objects belonging to a class. The two kinds of CIP objects are required and application [39]. Required objects include connection, TCP/IP, ENIP, and identity objects. Application objects relate to a type of device such as water-pressure, temperature, input, and output objects. Instances of an object inherit its attributes and properties. For example, the identity object of a device would have attributes for its serial number, vendor ID, manufacturer, and other identifying characteristics.

Each CIP object handles either common or object-specific services. Common services are predefined and can be generally requested from any object, while object-specific services are unique to an object type. These services are requested and carried over two types of messaging, explicit and implicit [38]. Explicit messages are sent over TCP by a server on a request from a client. Explicit messaging often pertains to requests for information about the object, instance, attribute of the device. Implicit messages are sent over UDP to transport short, predefined data. This information is sent at regular intervals to a server [39]. They can be seen as obligatory status updates containing real-time information, and are smaller than explicit messages to reduce network burden.

### **C. INTRUSION-DETECTION SYSTEMS**

An intrusion-detection system is a network-monitoring application for the automatic detection of malicious or unusual activity. Unlike other security applications, an intrusion-detection system does not quarantine files or otherwise heal a system, but only reports suspicious activity as it is detected.

The main kinds of intrusion-detection systems are misuse detection and anomaly detection [40]. In misuse detection, malicious behavior patterns are predefined and used as a basis for monitoring traffic. Misuse detection is often signature-based, where it checks a library of known malware signatures [41]. This type of system typically puts little processing burden on the host and achieves low false positive rates. While it is effective



for detecting known attacks, it cannot catch new exploits, which is an important ability for intrusion-detection systems to have.

Anomaly-detection systems track normal system behavior and flag behavior that diverges from this baseline. One subtype is behavior-based detection, which takes surrounding context from network traffic into consideration [40]. While anomaly detection is more suited for detecting new attack patterns, it has a high rate of false positives (behavior mistakenly labeled as malicious). Responding to intrusions can be complicated and expensive, so false positives are a more serious concern in cybersecurity than with other applications where machine learning is applied [41].

Machine learning can develop signatures for misuse-based intrusion-detection systems. The closed-loop control processes in cyber-physical environments show predictable patterns of behavior that are not as complex as other kinds of network data. This suggests that anomaly-based intrusion detection could work well for cyber-physical systems.

THIS PAGE INTENTIONALLY LEFT BLANK

### **III. PROBLEM DESCRIPTION AND APPROACH**

Cyber-physical systems (CPS) rely on prompt communications and mechanics to monitor or control physical processes. This creates security considerations that differ from most information technology (IT) systems [35]. With system availability being a high priority, efficiency is usually valued over security. CPSGAN could add security to cyber-physical systems without sacrificing availability.

#### **A. CYBER-PHYSICAL SYSTEM SECURITY**

##### **1. Security Issues**

CPS security is affected by legacy infrastructure and timing-control requirements [42]. Prompt and accurate messages ensure that CPS operations run smoothly, so industry developers emphasize the availability and integrity of data. Developers cannot easily apply the same security measures used in IT systems to critical physical infrastructure, as ensuring confidentiality in a CPS may hurt efficiency [43]. Efficient communications are desirable for most information technology, but they are essential in a CPS.

CPSs can range from a single device to many distributed components [35]. Some industrial systems can be 30 years or older and run continuously without downtime, though they are expanded with new equipment throughout their life cycles. To integrate new components with existing systems, many proprietary protocols provide backwards-compatibility and flexibility with legacy equipment. Many CPS protocols run over standard network protocols, like TCP/IP, and permit text communication of system data. Text communication in which data is sent unencrypted make CPS devices particularly vulnerable. Challenges can occur when manufacturers discontinue support for legacy software or devices. Sometimes, insecure legacy code is replaced by newer code but remains on the system and leaves it vulnerable [44].

Good latency and reliability over the network is needed for the production goals of the CPS and to maintain safe conditions. Delayed or lost network data can hurt the availability of the system, falsely represent the system's current state to operators, and put

the system into a dangerous state. Corruption of the integrity of messages can expose the system to these same risks. Text communications enable the attacker to study the CPS network's communication structure, then spoof other devices or inject malicious commands into network traffic.

## **2. Attack Methods**

Cyber-physical systems are vulnerable to a broad range of attack vectors [4], [36], [45]. The two datasets used in this thesis include reconnaissance activities, man-in-the-middle attacks, denial of service, distributed attacks, data exfiltration, and physical manipulation of controls. The last means that the attacker has physical access to the system and this activity is captured in device-status messages sent over the network. All other attacks mentioned may be done remotely.

Reconnaissance gathers information about a target's defenses, which attackers use to find an undefended entry point using techniques like scanning [46]. Scans involve sending packets to provoke a response from a target device [36], causing the device to confirm its presence or leak host information through the way it responds. Attackers can scan targets with host, port, and operating-system discovery methods using vulnerability detection tools like Nmap [47]. Host discovery determines the active IP addresses on a network, port discovery reveals which ports on a target device are running vulnerable services, and operating-system discovery infers the target's operating system through a process called "fingerprinting." If intrusion-detection methods flag reconnaissance before intrusions occur, it gives incident responders more time to react.

Man-in-the-middle attacks occur when an attacker can either listen passively or change the conversations between two communicating endpoints without detection [4]. This is easier when application data is sent unencrypted.

Denial-of-service attacks target the availability of systems and use large amounts of traffic to overwhelm a system or device. Denial-of-service attacks are also called "flooding," and have many variants that exploit different aspects of standard protocols. A well-known example called SYN-flooding issues a stream of incomplete connection

requests; eventually these half-open connections exceed the server’s memory and cause it to crash.

Distributed denial-of-service attacks exploit multiple machines to generate more traffic than a single machine can produce [2]. Some are executed by a botnet, a network of compromised machines. The Mirai worm, which primarily targets IoT devices, uses a botnet [7].

Data exfiltration is the unauthorized copying or transferring of protected information from a targeted system. Data exfiltration threatens the confidentiality of data and occurs with poor data flow controls or insider threats [36]. This is possible through some standard protocols with insecure settings, such as using passive mode with the File Transfer Protocol (FTP). A high-profile example of exfiltration was executed by the Advanced Persistent Threat OilRig (APT 34). OilRig actors wrote malware that used the Domain Name System (DNS) and the Hypertext Transfer Protocol (HTTP) services to covertly move data off of compromised machines [48].

## **B. MODEL DEVELOPMENT**

We studied two architectures for the CPSGAN model. The first design was based on long short-term memory (LSTM) architecture, which is a recurrent neural network [49]. We based our second design on a multilayer perceptron network.

The goal of the CPSGAN generator was to generate packet data that appears benign to an intrusion-detection system, which is modeled by the discriminator. The training process could create a more resilient discriminator for use in intrusion-detection systems. LSTM-based generative adversarial networks (GANs) that process network traffic for cyber defense are rare. The ability of LSTM-GANs to detect or produce new anomalous patterns have been evaluated on flow-based data [50] and cyber-physical device measurements [51], [52]. We used the results of previous work and packet-level LSTM classifiers [6], [18], [19] to guide the development of the LSTM-based CPSGAN.

## **1. Long Short-Term Memory in Packet-Level Classification**

Recurrent neural networks can process variable-length inputs and remember information from earlier parts of an input sequence to process its latter parts. LSTM networks consist of cells, which are gated recurrent units [23], [53]. The gates and point-wise operations in LSTM cells add or delete information from the LSTM’s cell state. These mechanisms are how a LSTM selectively remembers past information [54].

A LSTM could help intrusion detection because it could remember context in a per-packet detection method. Several packet-level classifiers used LSTMs in their designs. One study split each packet’s header into bytes, encoded the values, then fed the encoded byte array into their LSTM classifier [6]. Their input had 54 features, based on the 14-byte Ethernet header, the 20-byte IP header, and the 20-byte TCP header. Samples with missing headers were padded with 0s. A similar LSTM approach split packet payloads into 4-bit chunks [19].

## **2. Generative Adversarial Networks as an Intrusion-Detection Method**

The multilayer perceptron GAN is based on the network architecture used in PacketCGAN [55]. PacketCGAN could generate several types of packets and claimed to enhance encrypted-traffic datasets. The authors extracted packets from packet capture (PCAP) files and converted them into arrays of integers, which were then normalized between 0 and 1 before input to the discriminator. These are called “packet byte vectors” in PacketCGAN.

In PacketCGAN, the generator had five layers with a total of 2,476 neurons: a 100-neuron input layer that takes noise, followed by three hidden layers with 128, 256, and 512 neurons; and a 1480-neuron output layer. “Leaky rectified linear unit” (LeakyReLU) activation was applied to the hidden and output layers. The discriminator had five layers with a total of 2,377 neurons: a 1480-neuron input layer (matching the size of the generator’s output layer), followed by three hidden layers with 512, 256, and 128 neurons; and a one-neuron output layer for classification. Both networks were trained using the Adam optimization algorithm, cross-entropy loss, and a minibatch size of 64. Each network

trained for 200,000 rounds, using a standard GAN-training procedure described in Chapter IV, Section C under Training.

### **3. Methods**

We used the unsupervised learning technique of principal-component analysis (PCA) in a preliminary exploration of our datasets. PCA reduces the dimensionality of data and makes trends clearer, which helped us better see attacks that were separable from normal traffic. Other programs that supported training, such as the data preprocessor, were developed alongside the CPSGAN model. The supporting modules were designed to be flexible and suit the needs of the GANs in a “plug and play” manner, such as adapting the networks to take data of different shapes or transformations.

## **C. DATASETS**

The CPSGAN model was trained and tested against two cyber-physical system datasets. The Industrial Control System Instructional Cybersecurity Lab (ICSICL) dataset contains traffic data from an NPS industrial control system testbed of the same name [12]. The testbed traffic includes communications over the ENIP and CIP protocols described in Chapter II. The Internet of Things (IoT) dataset was captured from a network with several devices [11]. Both datasets are a collection of PCAP-format serialized data files containing packets captured from network traffic.

### **1. Industrial Control System Instructional Cybersecurity Lab Dataset**

The testbed environment for the ICSICL dataset used an Allen Bradley ControlLogix 1756-L71 chassis containing a 1756-L71 ControlLogix Logix5571 PLC [57], two ControlLogix Ethernet communication modules (EN2T, EWEB) [58], a Prosoft generic serial communications module [59], and several analog and digital input and output modules. The development software for these components was Rockwell Automation Studio5000 [60], which provided interfaces for control-logic development and monitoring devices. The PCAP files were collected on a separate computer connected to the ICSICL network using the tool Wireshark [60], which captures packets traveling through the network.

The ICSICL dataset contains 23 PCAP files that cover five types of traffic: normal operations, reconnaissance by HTTP Web activities, unauthorized manipulation of physical controls, data exfiltration using FTP file transfer in passive mode, and denial of service intended to silently shut down the PLC. The denial-of-service attacks are further split into two types that exploited the Register Session and Send Request/Reply Data commands. The Register Session denial-of-service capture was truncated to the first 20,000 packets in our experiments since many more of these packets occurred than the other attack types. Details of the attacks observed are listed in Table 1. In total, 80,876 packets were in the ICSICL dataset, of which 35,348 were considered malicious. Each packet in the dataset was preprocessed, labeled, and then fed into the discriminator during training. Appendix A shows examples of benign and malicious traffic in the ICSICL dataset.

Table 1. ICSICL Dataset: Malicious Traffic Composition

Category	Subcategory	Total Packets	Attack Packets
Normal	Normal	41,266	0
Reconnaissance	HTTP	16,061	14,787
Data Exfiltration	FTP Passive Mode	1,177	55
Physical Controls	Physical Controls	848	765
Denial of Service (DoS)	Register Session	20,000	19,281
	Send Request/Reply Data	1,524	460

## 2. Internet of Things Dataset

The IoT network intrusion dataset was captured by researchers from the Korea University Hacking and Countermeasure Research Lab (HCRL) to support intrusion and anomaly detection research. It is available through the IEEE DataPort platform online [11]. Two IoT devices, a Bluetooth speaker (SKT NUGU) and a Wi-Fi Camera (EZVIZ), simulated the victim or malicious actor in various runs. Some attacks interacted with other devices connected to the same network, such as laptops and smartphones.

The dataset includes 42 PCAP files of benign traffic, scanning, man-in-the-middle attacks, denial-of-service attacks, and distributed attacks caused by the Mirai worm. The attack categories of the IoT dataset are in Table 2, which gives the number of captured



packets and the number of malicious packets for each type of attack. The dataset contains 2,985,994 packets of which 1,232,845 packets are considered malicious. Each packet in the dataset is preprocessed, labeled, and fed into the discriminator during training.

Table 2. IoT Dataset: Malicious Traffic Composition

Category	Subcategory	Total Packets	Attack Packets
Normal	Normal	137,396	0
Man-in-the-Middle (MITM)	ARP Spoofing	194,184	101,885
Denial of Service (DoS)	SYN Flooding	141,709	64,646
Scanning	Host Discovery	310,480	3,127
	Port Scanning		20,939
	Operating-system/Version Detection		1,817
Mirai Botnet	UDP Flooding	1,187,114	949,284
	ACK Flooding	313,462	75,632
	HTTP Flooding	248,294	10,464
	Host Discovery	453,355	3,127
	Telnet Brute-force		1,924

THIS PAGE INTENTIONALLY LEFT BLANK

## IV. EXPERIMENTS

### A. MODEL DESIGN

CPSGAN consists of two neural networks: a generative model (generator) and a binary classifier (discriminator). The training of CPSGAN requires two additional modules, a preprocessor and a selector, which do packet-data preprocessing and training-data organization. CPSGAN and its supporting modules are in Figure 3. The preprocessor (shown in light blue) parsed packets, extracted features, made transformations to the data, and generated class labels for the data. The selector (shown in teal) formatted, shaped, and managed sequential inputs to both models. We discuss the data-preparation modules in the following sections. All modules shown were written in Python. The generator and discriminator were written using the Keras deep learning library [61].

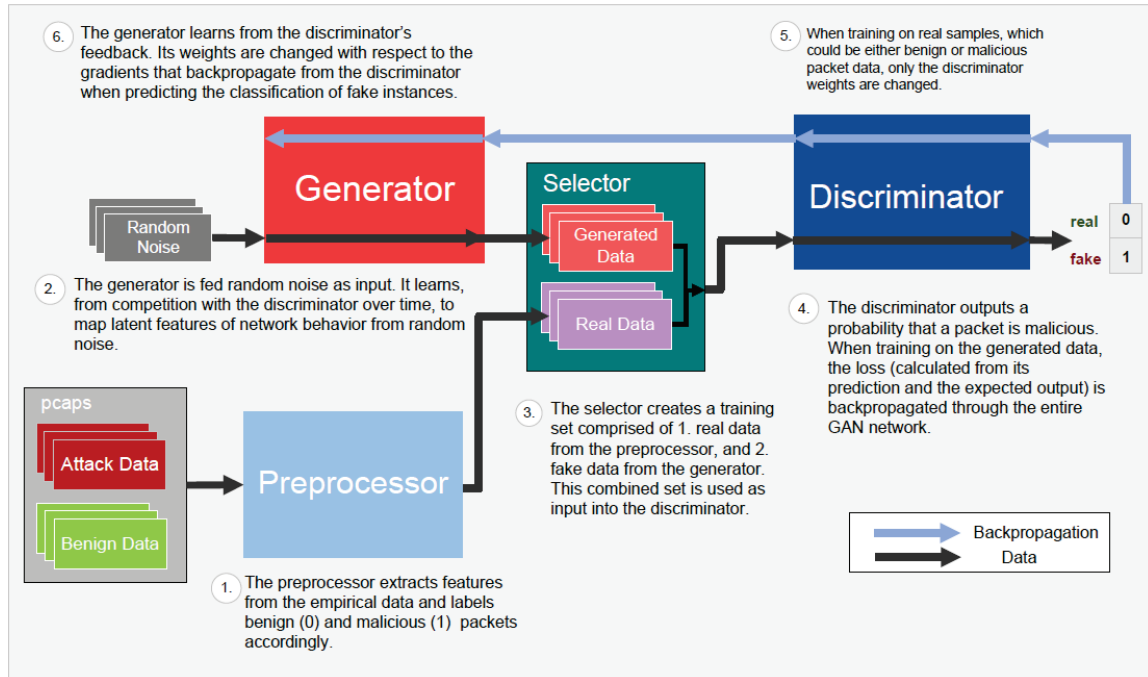


Figure 3. Overview of the CPSGAN Training Program

## B. DATA PREPARATION

### 1. Packet Structure

Our approach focused on TCP/IP packet headers, so data preparation involved converting raw packet headers to a format that could be input to the discriminator. Packets are organized bundles of data that coordinate network communications and interactions over the Internet. We can tell which protocols a packet uses by inspecting its headers, which contain metadata that help route a packet to its correct destination. Specific metadata, such as the protocols used or IP addresses, are in the fields of headers. An example is given in Figure 4.

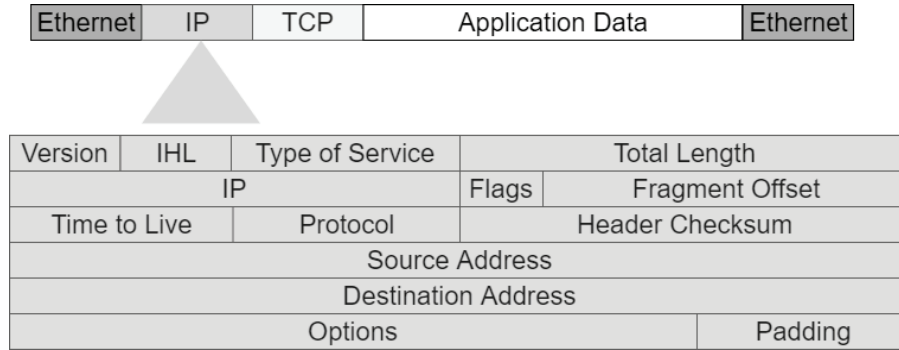


Figure 4. Packet Structure with Emphasis on the IP Header Fields.  
Adapted from: [62].

Standards help devices quickly parse and interpret the information in header fields. The syntax and semantics of each header field are defined in the Request for Comments 791 (RFC), which is maintained by the Internet Engineering Task Force [62]. Table 3 shows the packet header fields that we used for data preprocessing. The operating system adds a timestamp for each packet, which is stored with the individual packet in a PCAP file. A PCAP file is a serialized file format that contains a sequence of packets from captured traffic. Packet captures are taken using a tool like Wireshark [60] that collects network traffic as it is received by a network interface card.

Table 3. Parsed Header Fields used in CPSGAN

Packet Data	Length (in bytes)
Packet arrival time	N/A
Packet length	2
IP-layer protocol	1
IP address source	4
IP address destination	4
Port source	2
Port destination	2
TCP Flags	1
CIP ID Status*	2
ENIP Command*	2
ENIP Length*	2
ENIP Session*	4
ENIP Status*	4
ENIP Context*	8
ENIP Options*	4

\*Fields parsed in the ICSICL dataset only.

## 2. Preprocessor

The preprocessor converted raw PCAP contents into data that can be handled by the discriminator. A packet capture of an attack may contain a mixture of normal and attack traffic. Therefore, we must identify and label each packet as benign or malicious during preprocessing. The following tools were used for analyzing and preprocessing our test datasets:

1. TShark [60]: A Wireshark command-line tool that captures and analyzes network traffic. For each PCAP file in the dataset, TShark was called from the preprocessing program using our Wireshark filter rules written for the specific dataset.
2. Capinfos [60]: A Wireshark command-line tool that returns general information about the PCAP file, such as the number of packets it contains and its start and end times.

3. Numpy [63]: A Python package for array and matrix manipulation.
4. Pandas [64]: A Python package for array and matrix manipulation, which can also read and save CSV data.
5. Dpkt [65]: A Python package for reading PCAP files, parsing user-specified header fields in packets, and storing the parsed data in a Pandas DataFrame data structure.

We used Wireshark filters to extract only packets that met known characteristics of an attack, like the attacker’s IP address or the protocol that was exploited. For the IoT dataset, we used the filter rules provided by researchers who collected the PCAP files; we wrote our own rules for the ICISCL dataset. Further explanation of the filter rules is in Appendix B. Given a directory of PCAP files, the preprocessor applied Wireshark filter rules to each PCAP file to separate packets of interest and label them appropriately. Then, the preprocessor parsed each packet and extracted the header fields listed in Table 3.

### **3. Preprocessor’s Workflow**

In step 1 of Figure 5, the preprocessor called TShark with these rules to separate benign and attack packets of a PCAP file into benign traffic and attack traffic. In step 2, using the Dpkt Python module, the preprocessor parsed and extracted headers from each packet of the two kinds of traffic. The parsed headers of each packet formed a record in the preprocessed dataset. If any header fields were missing, the preprocessor padded the record with 0’s to make its output consistent in length. The preprocessor then labeled each packet in each array with a 0 for benign and 1 for attack and added this as a column to the array. After the labels were added, the preprocessor reordered records by timestamp and applied any necessary transformations to the data (described below). Reordering enabled testing against realistic sequences of traffic, and reordering is needed to compute packet interarrival times.

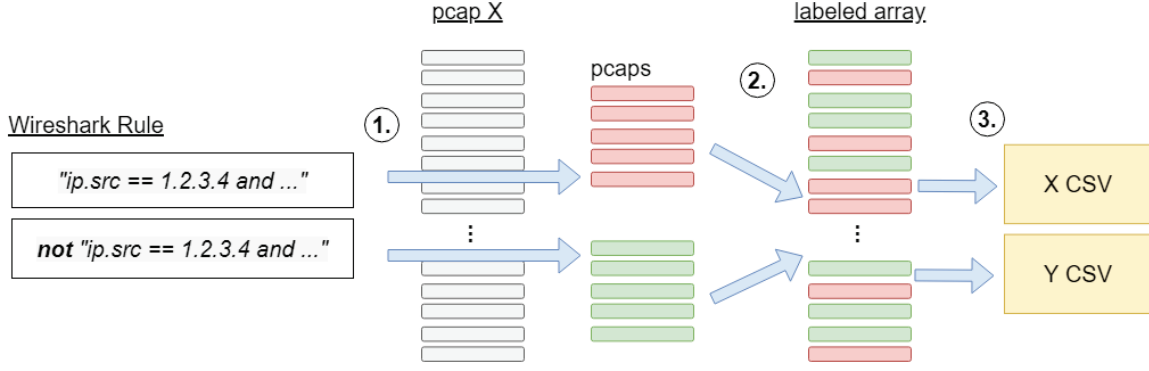


Figure 5. Behavior of the Selector

#### a. Data Transformation

The header-field values can be large and of variable length, which is undesirable for input to a neural network. For example, timestamps are recorded as UNIX epoch times and represent seconds since January 1st, 1970 [66]. We took the difference between consecutive packet timestamps and divided them by the maximum difference between timestamps to get a number between 0 and 1 [67].

We split packet data along a byte boundary, using a similar method in [6]. Therefore, input to our long short-term memory model consists of features in the range of 0 to 255. The packet interarrival times, calculated in the previous step, are scaled to a value in this range. These integers are “one-hot” encoded or embedded before they are input to the LSTM discriminator. One-hot encodings represent nonnumeric features as bit vectors with a single one bit. An embedding is an encoding that represents non-numeric features with a dense vector of values [18], [68]. We used one-hot encodings for the LSTM-CPSGAN model and embeddings from the Keras library for the standalone LSTM-discriminators [50]. This is because trainable embeddings cannot be placed directly between the generator and discriminator.

For the multilayer perceptron model, we used networking definitions to bin nonnumeric categorical data and create bit-mapped data. Table 4 describes the binning of the Classful IP addressing scheme [62] and Table 5 shows the binning of TCP/UDP ports [69], which we used for categorizing the IP address and TCP/UDP header field

values. TCP flags, which are expressed as one byte in the packet data, were separated into bit representations to be used as Boolean features. We handled the TCP flags SYN, RST, PSH, ACK, URG, ECE, CWR, and NS.

Table 4. Binning of IP Addresses

Class	IP Address Range
A	1.0.0.1 to 126.255.255.254
B	128.1.0.1 to 191.255.255.254
C	192.0.1.1 to 223.255.254.254
D	224.0.0.0 to 239.255.255.255
E	240.0.0.0 to 254.255.255.254

Table 5. Binning of Port Ranges

Type	Port Number Range
Well-known	0 to 1023
Registered	1024 to 49151
Private	49152 to 65535

Preprocessing extracted 50 features from the ICSICL dataset and 27 features from the IoT dataset, in Table 6. We limited features to the header fields in Table 6 because they are key for identifying the type of attack and the packets of the attack.

Because the ICSICL dataset is unbalanced due to the large volume of denial-of-service packets, we duplicated preprocessed benign packets to create a fuller training set for experimentation. This was straightforward for the ICSICL dataset as industrial-operations network traffic is very repetitive, meaning many packets contain the same values for these features.



Table 6. Total Number of Features in the ICSICL and IoT Datasets

ICSICL Features		IoT Features	
Packet arrival time	1	Packet arrival time	1
Packet length	2	Packet length	1
IP-layer protocol	1	IP-layer protocol	1
IP address source	4	IP addresses	5
IP address destination	4	IP addresses	5
Port source	2	Port source	3
Port destination	2	Port destination	3
TCP Flags	8	TCP Flags	8
CIP ID Status	2	<b>Total</b>	<b>27</b>
ENIP Command	2		
ENIP Length	2		
ENIP Session	4		
ENIP Status	4		
ENIP Context	8		
ENIP Options	4		
<b>Total</b>	<b>50</b>		

#### 4. Selector

The selector handled training and testing for CPSGAN by preparing sequences of preprocessed traffic data for the discriminator. When training CPSGAN, the selector used a batch size of 50 packets. This means for each iteration of training, the selector pulled 50 packets from the real dataset and 50 fake packets from the generator, then supplied them to the discriminator in the order they were captured. Therefore, in each iteration of training CPSGAN, the discriminator tried to predict the classifications of 100 packets.

The selector gave 80% of the real traffic to the discriminator for training and saved the other 20% for post-training testing. We wanted both training and testing sets to cover as many attack patterns in the dataset as possible. We also did not want to randomize individual packets, as this would fully randomize the capture order and remove important context for understanding certain attacks. Instead, the selector divided the preprocessed dataset into short subsequences, allocated 80% of the subsequences to a training set and 20% to a testing set, then selected and returned a random subsequence when requested.

To expose the models to equal amounts of traffic, each model trained on a predetermined number of subsequences from the 80% training set of each dataset. If the selector exhausted the training indices before training completed, the selector shuffled the subsequences and iterated through the training data in a new order.

### C. CPSGAN

Although the discriminator in our GAN used supervised learning with labels on malicious and benign data, the generator used unsupervised learning. The generator never saw the real training dataset and must learn it through incomplete information. Its only feedback are the losses backpropagated from the discriminator.

Each model required a loss function and an optimizer. An optimizer controls the method and degree to which weights are changed using the loss function's calculations. The four networks described in this section (i.e., both types of generators and discriminators) were trained using binary cross-entropy loss and the Adam optimizer [70] with a learning rate of 0.0002.

The number of features in the dataset determines the dimensions of each model, represented as  $N$  in this section. The layers in our LSTM model must be defined using  $N$  as the number of time-steps, where  $N$  defines the length of an input sequence and depends on the number of features in the dataset. For recurrent neural networks, the value of  $N$  controls how many parts of the input sequence to run through the LSTM.

#### 1. Generator

The generator in a GAN learns a joint probability distribution  $p(x, y) = p(x|y)p(y)$ . But further, its goal is to generate nontrivial outputs that the discriminator believes to be benign although they are actually malicious. The hope is that the generator could discover a new attack vector; but more importantly, we wanted to strengthen the discriminator against unknown types of traffic.

In both the LSTM and MLP generators, the output layer depended on the number of features in the dataset and used the same value of  $N$  as the discriminator's input layer.

This allowed the generator to output fake packets with the same dimensions as the real packets, which are provided by the selector.

***a. Long Short-Term Memory***

The generator in the long short-term memory (LSTM) design is based on a specialized recurrent neural network and is different from the selector. This architecture is suited for sequence-based data and pattern-recognition tasks [71], which motivated its choice in CPSGAN’s implementation. Both the generator and the discriminator in the LSTM-CPSGAN contain bidirectional LSTM layers.

A single LSTM is a three-layer neural network. A bidirectional LSTM has two LSTM networks that run in parallel. While the generator seems to have only one hidden layer in Table 7, each LSTM “layer” is truly a group of perceptron layers on the backend and is configurable through the Keras application programming interface.

Table 7. Long Short-Term Memory Generator Parameters

<u>Layer</u>	<u>Type</u>	<u>Neurons</u>	<u>Activation</u>
Input	Input Layer	$N$	None
Hidden 1	Bidirectional LSTM	256	Sigmoid, Tanh
Output	Dense	256	SoftMax [70]

The value of  $N$  specifies the time-steps parameter in the Bidirectional LSTM layer, which is not shown for this layer in Table 7 because it does not define the number of neurons. Setting the time-steps parameter to  $N$  ensures that the generator returns a sequence of equal to the length of a packet. The value of 256 in Table 7 corresponds to a one-hot encoding’s maximum-length representation for a byte. When given a noise input, the generator outputs  $N$  one-hot encodings, which represent a packet byte sequence with the same dimensions as the input layer of the LSTM discriminator.

***b. Multilayer Perceptron***

The perceptron generator, which is not the same as the selector, produced an encoded packet instance for input to the perceptron discriminator. The output layer

contained  $N$  neurons and output a packet instance, where  $N$  is the number of features in the dataset. Table 8 summarizes the MLP generator model.

Table 8. Multilayer Perceptron Generator Parameters

<u>Layer</u>	<u>Type</u>	<u>Neurons</u>	<u>Activation</u>
Input	Input Layer	50	None
Hidden 1	Dense	100	ReLU
Hidden 2	Dense	200	ReLU
Hidden 3	Dense	400	ReLU
Hidden 4	Dense	200	ReLU
Hidden 5	Dense	100	ReLU
Hidden 6	Dense	50	ReLU
Output	Dense	$N$	Tanh

## 2. Discriminator

The discriminator is an intrusion-detection mechanism that calculates the conditional probability that a packet sample is malicious. One simple approach is that if the conditional probability is below 0.5, the sample is classified as real and benign. This is the threshold that we used for interpreting predictions. Large probabilities mean that traffic was either malicious or produced by the generator, while 0 means that traffic was real and benign.

### a. Long Short-Term Memory

The LSTM discriminator is a classifier with an input layer and three hidden layers: a bidirectional LSTM, a standard LSTM, and a fully-connected layer with two neurons. The fifth (output) layer is a single predictive node that outputs the probability that an input packet is anomalous. We used the value of  $N$  to specify the number of time steps for the LSTM layers. This tells the network to process inputs up to the length of a packet. This value is not included in Table 9 since  $N$  does not describe the number of neurons in the layer.

Table 9. Long Short-Term Memory Discriminator Parameters

<u>Layer</u>	<u>Type</u>	<u>Neurons</u>	<u>Activation</u>
Input	Input Layer	256	None
Hidden 1	Bidirectional LSTM	256	Sigmoid, Tanh
Hidden 2	LSTM	80	Sigmoid, Tanh
Hidden 3	Dense	2	Tanh
Output	Dense	1	Sigmoid

The value of 256 in Table 9 is the length of a byte’s one-hot encoding. The LSTM discriminator took one-hot encoded packet samples as input. This is passed to a bidirectional LSTM layer, which contains a forward LSTM and a backward LSTM [72]. The input sequence to the bidirectional LSTM is copied, and the sequential order of this copy is reversed. The original sequence is passed through the forward LSTM and the reversed sequence is passed through the backward layer. The two outputs from the LSTMs are concatenated and passed to the next LSTM, which is unidirectional.

***b. Multilayer Perceptron***

The multilayer perceptron (MLP) discriminator had seven hidden layers. The number of neurons in the input layer and the sixth hidden layer are determined by the number of features in the dataset,  $N$ . The first six hidden layers used a rectified linear unit (ReLU) activation [70] and the seventh used a hyperbolic tangent activation (Tanh). A rectifier sums the inputs of the neuron and applies linear activation to it if the sum passes a given threshold (usually zero, as in this case). The output layer output a predictive value between 0 and 1. Table 10 shows the dimensions of the perceptron discriminator.

Table 10. Multilayer Perceptron Discriminator Parameters

<u>Layer</u>	<u>Type</u>	<u>Neurons</u>	<u>Activation</u>
Input	Input Layer	$N$	None
Hidden 1	Dense	100	ReLU
Hidden 2	Dense	200	ReLU
Hidden 3	Dense	400	ReLU
Hidden 4	Dense	200	ReLU
Hidden 5	Dense	100	ReLU
Hidden 6	Dense	$N$	ReLU
Hidden 7	Dense	2	Tanh
Output	Dense	1	Sigmoid

### 3. Training

In the training algorithm, the generator and discriminator took turns learning (adjusting their weights) [10]. For CPSGAN, the plan was that each iteration of training involved these steps:

1. The generator created packets using noise inputs.
2. The discriminator was trained on inputs chosen by the selector from either the preprocessed dataset or the output of the generator, created in step 1.
3. The generator's output layer was connected to the discriminator's input layer to form a single network, the GAN. Noise was input to the generator's input layer and forward-propagated through the GAN. The generator was trained by backpropagation from the discriminator within the GAN.

In step 1, neither network was trained. The generator only produced some data meant to train the discriminator, which did not change the weights of either the discriminator or generator (Figure 6). The purpose of this step is to provide fake inputs for the selector's pipeline in step 2.

In the first few iterations of training, the generator's output may appear to be random guessing (noise). If training was successful, then the generator's output may take a more specific form in later iterations of training.

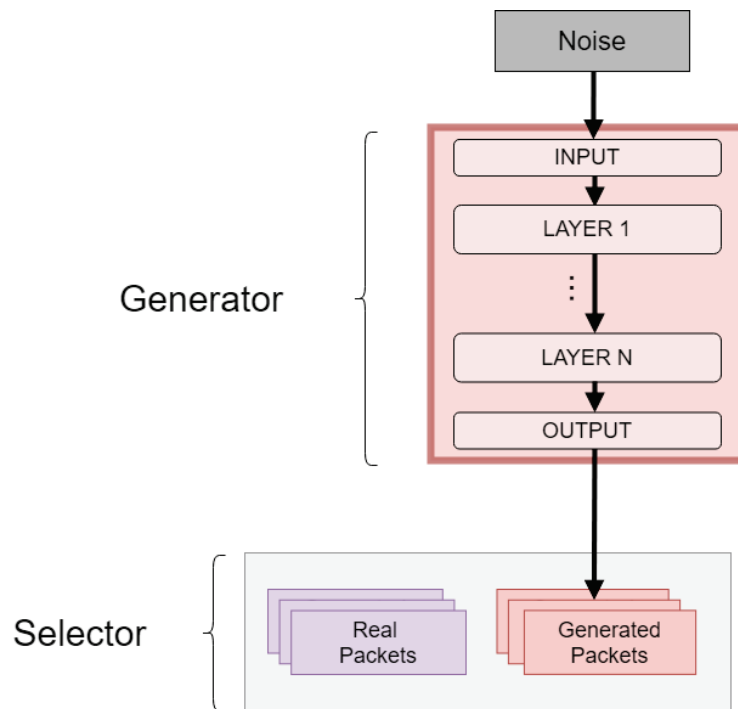


Figure 6. Step 1 of the CPSGAN Training Procedure

Step 2 trained the discriminator (Figure 7). The selector provided both real-labeled packets from the preprocessed dataset (2a in Figure 7) and crafted packets from the generator (2b in Figure 7) to the discriminator. Real packets may have either benign or malicious labels, which were determined by the preprocessor. For this step only, all generated packets were given malicious labels.

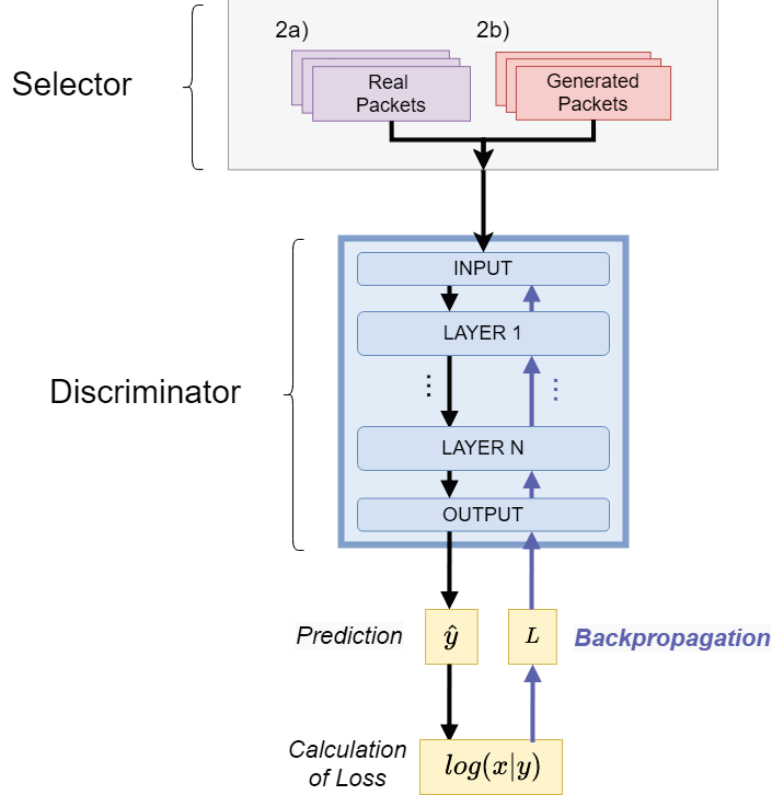


Figure 7. Step 2 of the CPSGAN Training Procedure

In step 3, the generator and discriminator were connected to form a GAN (Figure 8). The input to the generator of the GAN was noise. It flowed through the generator's layers and computed an instance with the packet parameters described in section 4.B. This generated instance was input to the discriminator. The discriminator judged to what degree the packet data was generated data or real attack data, which were both labeled as "malicious." We computed the discriminator's error as one minus the output if the data was malicious, and the output alone if the data was real and benign. The resulting error was backpropagated through the discriminator first, then through the generator. Thus, the generator received indirect feedback from the discriminator.



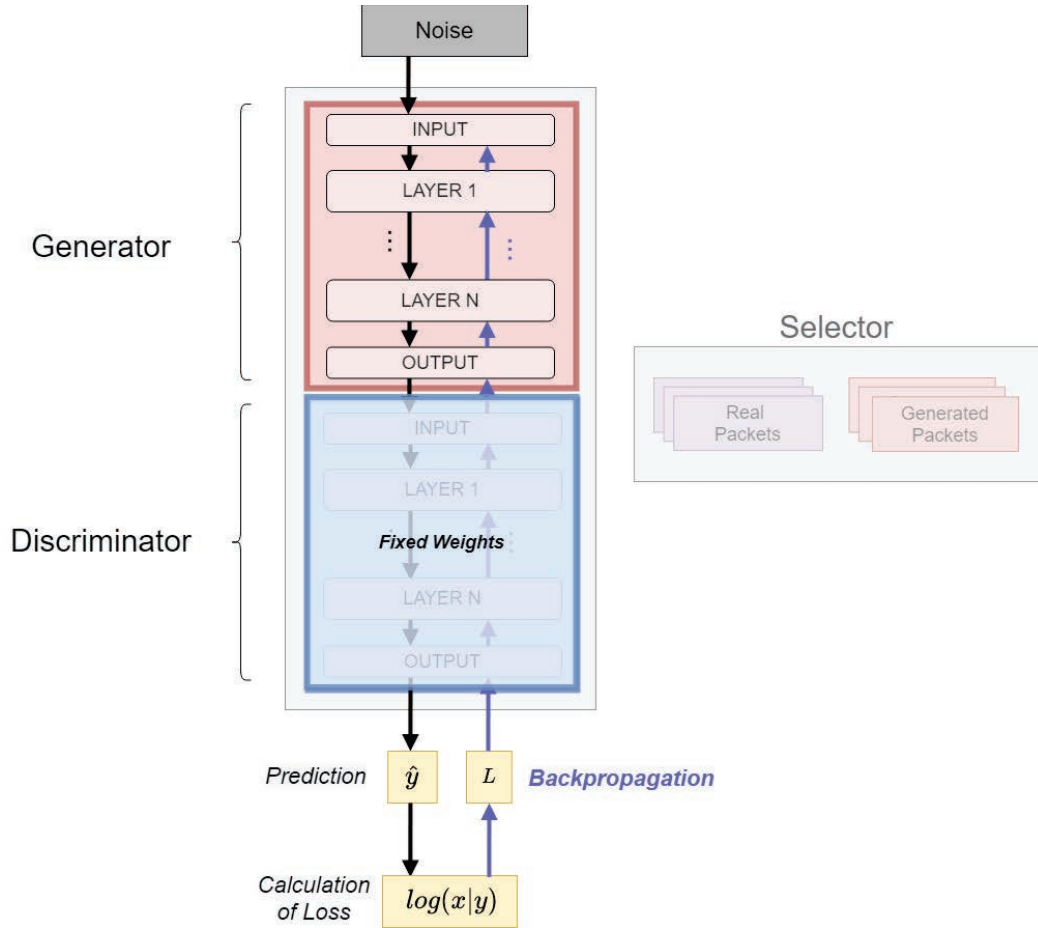


Figure 8. Step 3 of the CPSGAN Training Procedure

We deactivated weight updates to the discriminator in step 3 so that only the generator was updated. Backpropagation calculations still occurred throughout the discriminator and affected how the generator's weights changed, but the discriminator retained its original weights after this step. If the discriminator's judgements of real and fake were shifting while the generator was trying to learn, we would have been asking the generator to predict a moving target. Our GAN used binary cross-entropy as its loss function for both discriminator and generator training. Binary cross-entropy measures the distance between a model's prediction and classification probabilities. This is done by taking the logarithm of the error and multiplying it against the ground-truth probability for each of the classes (benign and malicious). Loss is the negated sum of these products. Both

networks want to minimize their respective losses, but the discriminator’s success is the generator’s failure and vice versa. This situation describes a minimax game.

During CPSGAN training, the multilayer perceptron model iterated through 10,000 rounds of the training procedure. The long short-term model iterates through 5,000 rounds of the training procedure.

## D. EXPERIMENTS

### 1. Analysis Methods

For our experiments, we measured accuracy and F1 scores to evaluate our model’s performance. They are derived from the number of true positives, false positives, true negatives, and false negatives that occurred during testing [18]. True positives ( $TP$ ) are the total number of correct predictions of malicious packets the model has made. False negatives ( $FN$ ) are the number of malicious packets the discriminator did not detect and false positives ( $FP$ ) are the number of times the model incorrectly flagged a benign packet as malicious. Table 11 summarizes this.

Table 11. Quantifying the Discriminator’s Predictions

<i>Truth</i>	<i>Predicted</i>	<i>Outcome</i>	<i>Behavior of Discriminator</i>
1	1	True Positive	An anomalous packet was caught
0	0	True Negative	A benign packet was correctly classified
1	0	False Negative	An anomalous packet was missed
0	1	False Positive	A benign packet was flagged as anomalous

Accuracy describes the proportion of correct predictions over total instances observed during testing:

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

F1-scores generally provide a better assessment of data with uneven distribution of classes than just accuracy alone. To calculate an F1-score, we first compute two relevance metrics, precision and recall. With respect to a classification, precision measures the

proportion of instances that were correctly detected out of all predictions. Recall measures the proportion of instances the model correctly detected out of all instances from a given classification. An F1-score is defined as:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

We also graphed results with receiver operator characteristic (ROC) curves and precision-recall curves. Precision-recall curves compare these two statistics across different thresholds to visualize performance on imbalanced datasets [73]. ROC curves compare the recall (equivalently, the true-positive rate) and the false-positive rate by measuring the rate at which the model misclassifies benign samples. The AUROC, or area under the ROC curve, measured the performance of our models. The false-positive rate is defined as:

$$FPR = \frac{FP}{FP + TN}$$

## 2. Principal-Component Analysis and K-Means Clustering

Our experimental results were also analyzed with principal-component analysis and K-Means clustering. Principal-component analysis is an unsupervised method for finding useful linear combinations of numeric features in high-dimensional data [21]. These linear combinations are called “principal components” and they compress information about the data’s covariance into a smaller dimensional space without losing significant amounts of information [67]. We used principal-component analysis to reduce the dimensionality of the ICSICL and IoT datasets, then applied K-Means clustering to their resulting principal components.

Before principal-component analysis, we scaled the data using z-score normalization [67], which subtracts the mean and divides by the standard deviation to give all variables a zero mean and standard deviation of 1 to make them easier to compare. We plotted the two components with the largest explained variance to see whether individual attacks were linearly separable from normal network behavior.

K-Means assigns data points to the cluster with the nearest cluster centroid, then recalculates the centroids iteratively. We clustered the dataset transformed by principal-component analysis with a goal number of clusters equal to the number of traffic types (several attack types plus benign traffic). The ICSICL dataset contained six traffic types and the IoT dataset contained ten traffic types. We used the performance metrics in Section IV.D.1 to assess how well the data was partitioned.

### **3. CPSGAN Evaluation**

CPSGAN was evaluated based on its discriminator’s performance as a classifier and the generator’s ability to emulate network traffic patterns. The results of K-Means classification were compared to the results from our experiments with the long short-term memory and multilayer perceptron GAN models. We used different combinations of network traffic from the datasets to train and evaluate these models. The test plan is described in Table 12, in which there are ten experiments. The experiments labeled with “A” were trained on the ICSICL dataset and the experiments labeled with “B” were trained on the IoT dataset. All ten experiments in Table 12 were applied to both the long short-term memory and multilayer perceptron GANs. Therefore, there were 20 tests in total.

Table 12. Model Testing Plan

<i>Exp.</i>	<i>Model</i>	<i>Trained On</i>	<i>Evaluated On</i>	<i>Exp.</i>	<i>Expected Result</i>
1A	Standalone Discriminator	ICSICL	ICSICL	Evaluate on benign and malicious packets	Good performance
1B	Standalone Discriminator	IoT	IoT	Evaluate on benign and malicious packets	Good performance
2A	Standalone Discriminator	ICSICL	ICSICL + IoT	Cross-validate with real IoT packets, labeled as anomalous	Similar performance to Experiment 1
2B	Standalone Discriminator	IoT	ICSICL + IoT	Cross-validate with real ICSICL packets, labeled as anomalous	Similar performance to Experiment 1
3A	GAN Generator	ICSICL	ICSICL	Train and sample generator output	Either convergence or mode collapse
3B	GAN Generator	IoT	IoT	Train and sample generator output	Either convergence or mode collapse
4A	GAN Discriminator	ICSICL	ICSICL	Evaluate on benign and malicious packets	Strong performance or 50% accuracy
4B	GAN Discriminator	IoT	IoT	Evaluate on benign and malicious packets	Strong performance or 50% accuracy
5A	Standalone Discriminator	ICSICL	ICSICL + Generated ICSICL Packets	Evaluate on benign and generated ICSICL packets	Weaker performance than Experiment 1
5B	Standalone Discriminator	IoT	IoT + Generated IoT Packets	Evaluate on benign and generated IoT packets	Weaker performance than Experiment 1

We trained and evaluated the discriminators on each dataset in Experiment 1, then tested them with cross-dataset packets in Experiment 2. In Experiment 3, we trained the generator and discriminator of each GAN and saved their weights. In Experiment 4, we evaluated the GAN-trained discriminator (from Experiment 3) on the 20% test set of real data. In Experiment 5, we used the discriminator weights from Experiment 1 (which were evaluated on real benign and malicious packets) and the Experiment 3 generator weights. We generated a number of packets equal to the number of benign packets in the 20% test set for the discriminators' test sets, and assessed whether they detected the generated instances.

The training set size is 80% of the preprocessed packets for a given dataset. For each round of training, the selector provided data in 50-packet sequences, where 50 is our batch size. Therefore, the total observed instances for a given training session is the number of packets that were sampled from each dataset to train a model, given by *training rounds* \* 50. Roughly speaking, the Experiment A series examined the ICSICL training data 10 times. The Experiment B series examined the same amount of traffic from the considerably larger IoT dataset.

## V. RESULTS AND ANALYSIS

### A. PRINCIPAL-COMPONENT ANALYSIS AND K-MEANS CLUSTERING

The first experiment applied a two-part approach to analyze the ICSICL and IoT datasets. This approach was first used to explore the data, but the results also provide some justification for choosing a more sophisticated method like CPSGAN. We used principal-component analysis (PCA) to reduce the datasets to two principal components, then clustered the transformed data using the K-Means algorithm.

The K in K-Means is the number of target clusters. We chose a number of clusters equal to the number of traffic types in the tested dataset (ICSICL or IoT), which were identified by the collectors of the data. We assessed the quality of clusters by using the metrics described in Section IV.D.1. True positives for PCA followed by K-Means were the attack packets that were in their correct cluster. False positives were benign packets found in an attack cluster and false negatives were attack packets found in the benign cluster. Identification performance varied widely for the attack types, with some having high recall and some having high precision, but none were good on both metrics.

For ICSICL, there were six clusters total covering the types of packets in the dataset: benign, FTP exfiltration, HTTP probe, physical manipulation of controls, and two denial-of-service exploits using the Register Session and Send Request/Reply Data commands. The highest recall recorded was 0.58 for the physical manipulation of controls, meaning that the K-Means algorithm correctly grouped this proportion of packets into the same attack cluster. The highest precision, 0.20, was recorded for the benign cluster, meaning this proportion of packets in benign cluster were truly benign packets.

In total for ICSICL, K-Means clustered 10,726 true positives, 24,574 false negatives, and 20,006 false positives. Generalizing the attack clusters as “anomalous” gives us a precision of 0.35 and a recall of 0.30 for this dataset. The K-Means algorithm grouped many malicious ICSICL packets into the benign cluster (false negatives). The true traffic categories after PCA and the K-Means clusters are in Figure 9.

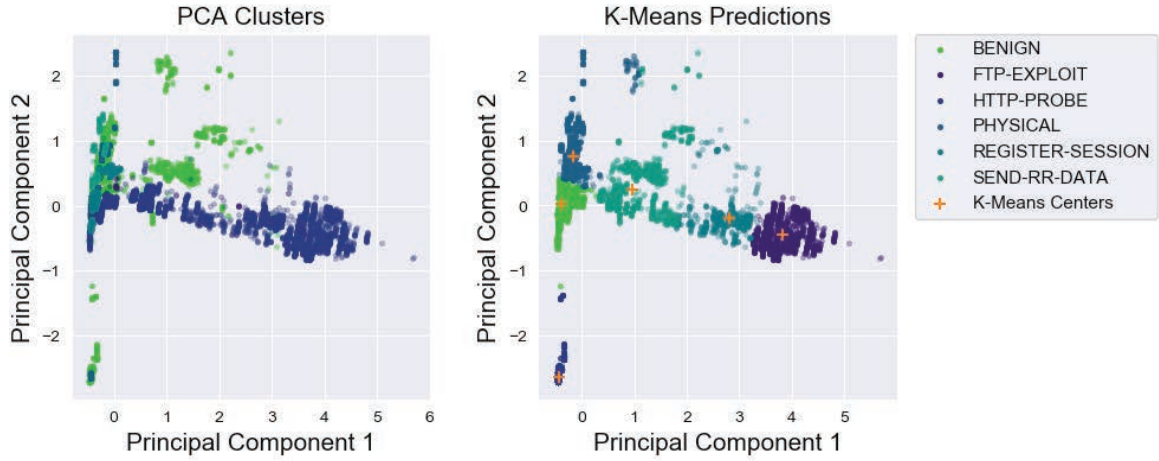


Figure 9. PCA and K-Means Clustering on the ICSICL Dataset (K=6)

For the IoT dataset, there were ten clusters total: benign, ACK-flooding, ARP-spoofing, host discovery, HTTP-flooding, OS-detection, port-scanning, SYN-flooding, telnet brute-force, and UDP-flooding (Figure 10). Overall, the precision and recall for the benign cluster were 0.90 and 0.23 respectively. When treating the attack clusters as a single classification, the K-Means algorithm achieved a precision of 0.39 and a recall of 0.95. While the attack clusters captured much of the malicious traffic, they also contained a good amount of benign traffic. In total for the IoT dataset, the K-Means algorithm recorded 674,492 true positives, 1,072,997 false negatives, and 37,531 false positives.

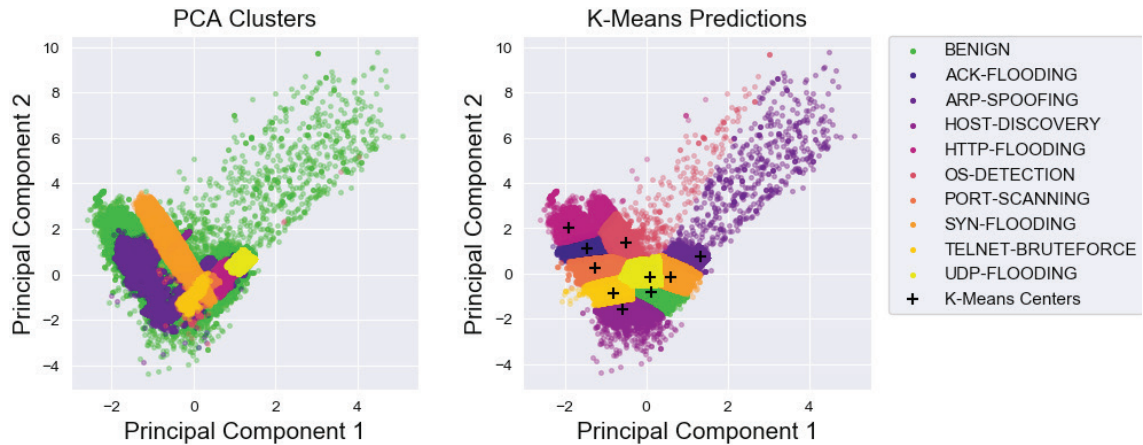


Figure 10. PCA and K-Means Clustering on the IoT Dataset (K=10)



These results may be due to the IoT dataset being large and its benign traffic being diverse. The IoT dataset was taken on a network with several user devices and covers a broader range of protocols, so it is more heterogeneous than the ICSICL dataset. This affected the readjustment of the cluster centers and increased the probability of overlap with attack data.

## **B. CPSGAN RESULTS**

Testing CPSGAN focused on the discriminator’s performance with combinations of real benign, real malicious, and generated instances from the two datasets. Results are organized based on the architecture being tested (long short-term memory or multilayer perceptron) and the dataset used to train the model (ICSICL or IoT). Therefore, we tested four combinations of architecture and dataset: LSTM and ICSICL; LSTM and IoT; MLP and ICSICL; and MLP and IoT.

The model test plan was described in Table 12 in Chapter IV. Unless stated otherwise, the training and testing sets are based on the 80/20 split of the dataset, determined at the start of Experiment 1. When interpreting the discriminators’ predictions, we used a threshold of 0.5 to define a “positive.” Predictions over this threshold were considered positives (malicious or generated). For the five experiments, we used the Scikit-Learn package [68] for the receiver-operating characteristic (ROC) and precision-recall curve calculations and plotting.

### **1. Experiment 1: Evaluation of the Discriminators on Real Data**

The first experiment individually evaluated the long short-term memory (LSTM) discriminator and the multilayer perceptron (MLP) discriminator in a supervised manner. Each discriminator trained on 500,000 packets from 80% of the dataset, then was evaluated on the remaining 20% after training concluded. The testing proportion of each dataset contained 12,350 packets for ICSICL or 422,300 packets for IoT. These parameters were chosen to expose the discriminators to an equal amount of traffic in Experiment 1, which set a basis for the rest of the experiments.

Table 13 shows the performance of each discriminator on classifying packets. True positives are malicious packets that the discriminator correctly classified as malicious; false positives are benign packets that the discriminator classified as malicious; and false negatives are malicious packets that the discriminator classified as benign.

Table 13. Experiments 1A and 1B: Frequency Statistics of Standalone Discriminators

<i>Exp.</i>	<i>Standalone Discriminator</i>	<i>Train Set (80%)</i>	<i>Test Set (20%)</i>	<i>True Pos.</i>	<i>False Pos.</i>	<i>False Neg.</i>	<i>Tested Instances</i>
1A	LSTM	ICSICL	ICSICL	7785	52	5	12350
1A	MLP	ICSICL	ICSICL	7185	66	605	12350
1B	LSTM	IoT	IoT	138256	8418	4626	422300
1B	MLP	IoT	IoT	119735	6355	23147	422300

Figure 11 shows the receiver-operating characteristic (ROC) curves of each architecture-dataset combination and the area under the curve, labeled “AUC.” The shaded regions of the plot are to visualize the AUC score of the model on a given dataset. The AUC score shown is a representation of how close the discriminator’s predicted probabilities are to the ground truth. The larger this area is, the better the AUC indicates good performance.



Figure 11. Experiments 1A and 1B: ROC Curves of Standalone Discriminators

Figure 12 shows the precision-recall curves of each discriminator on the ICSICL and IoT datasets. Figure 12 also includes the average precision (labeled “AP”) between the discriminator’s predictions and the ground truth. Precision-recall curves describe the tradeoff between precision and recall at different thresholds. Here, they show that the MLP model struggled most on the IoT dataset in Experiment 1B, compared to the other three tests in Experiment 1. The LSTM model in Experiment 1A achieved the best results of the four tests. Overall, the discriminators achieve comparable results in the first experiment.

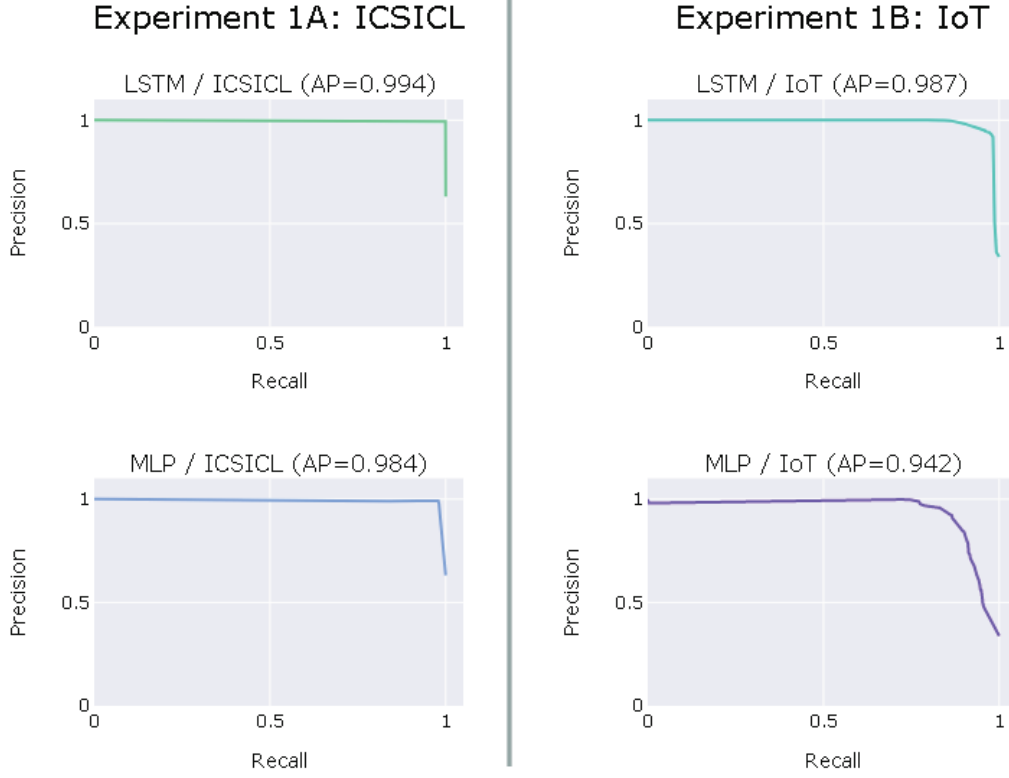


Figure 12. Experiments 1A and 1B: Precision-Recall Curves of the Standalone Discriminators

## 2. Experiment 2: Evaluation of the Discriminators Across Datasets

In Experiments 2A and 2B, we assessed whether the pretrained discriminators could detect traffic belonging to a different network. Networks have unique traffic signatures that would cause packet features to differ between datasets. The test sets of Experiments 2A and 2B selected benign packets from the 20% of the training dataset and selected an equal number of malicious packets from the other dataset, which contained network data that the model has not seen. For instance, if the model were trained on ICSICL, it was tested on a combination of reserved ICSICL and IoT data.

Under the “Test Set” column in Table 14, two datasets are listed. To keep the number of benign and malicious packets balanced, we made the number of benign packets and the number of malicious packets equal in the test sets. The lack of true positives in Experiment 2B indicates that both the long short-term memory and multilayer perceptron models could not recognize any malicious packets. The models did better on the ICSICL

dataset than on the IoT dataset, which could be either because the ICSICL traffic is more repetitive than the IoT traffic or because the ICSICL dataset is significantly smaller.

Table 14. Experiments 2A and 2B: Frequency Statistics of Standalone Discriminators on Combined Dataset

<i>Exp.</i>	<i>Standalone Discriminator</i>	<i>Train Set (80%)</i>	<i>Test Set (Combined)</i>	<i>True Pos.</i>	<i>False Pos.</i>	<i>False Neg.</i>	<i>Tested Instances</i>
2A	LSTM	ICSICL	ICSICL + IoT	4051	52	499	9100
2A	MLP	ICSICL	ICSICL + IoT	2083	66	2467	9100
2B	LSTM	IoT	IoT + ICSICL	3	163	7747	15500
2B	MLP	IoT	IoT + ICSICL	38	148	7712	15500

The LSTM adapted better to new IoT instances when first trained on ICSICL packets in Experiment 2A, but its performance decreased in Experiment 2B when trained on the IoT dataset and given ICSICL packets (Table 14). Both models did worse on the IoT dataset, especially in Experiment 2B. The MLP in Experiment 2B had worse performance than random guessing. This problem could be due to training neural networks in the absence of good training examples [14]. The MLP model's tradeoff between recall for precision is nearly 2 to 1 (Figure 13).

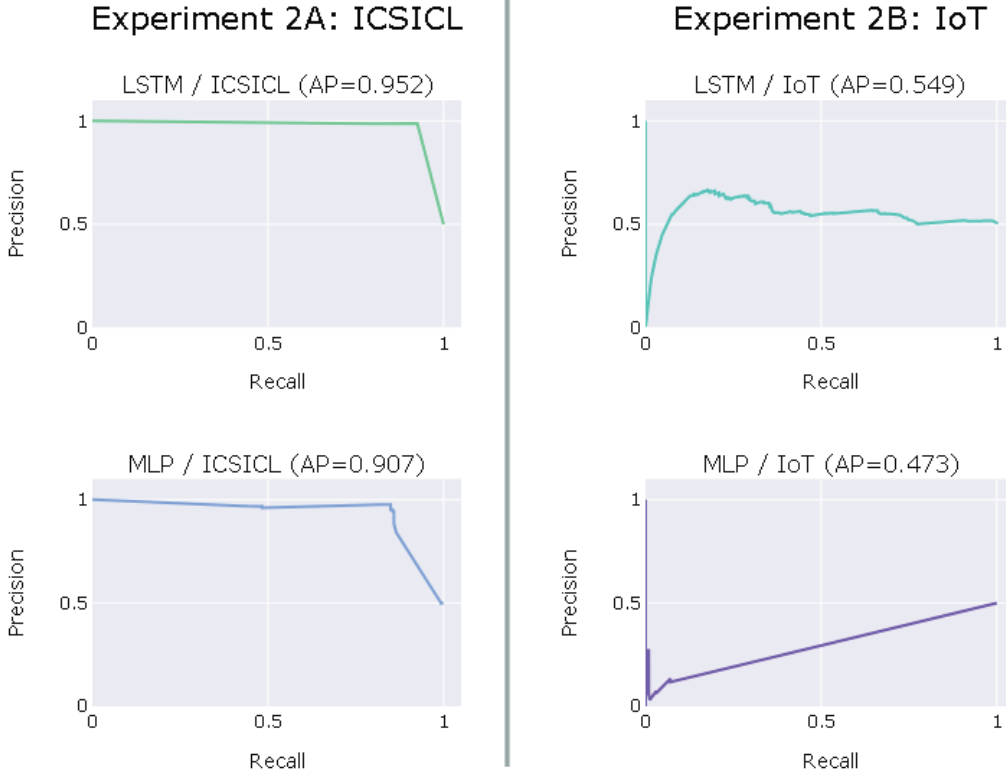


Figure 13. Experiments 2A and 2B: Precision-Recall Curves of Standalone Discriminators on Combined Dataset

### 3. Experiment 3: Training CPSGAN and Analysis of Model Loss

Experiments 3A and 3B studied CPSGAN’s training loss (error), which captures the stability and progression of learning. These experiments used the GAN-training algorithm described in Chapter IV, in which the discriminator and generator take turns training per iteration. Half of the discriminator’s training data were real instances from 80% of the preprocessed dataset with malicious and benign labels. The other half were output from the generator and labeled as malicious.

The MLP-GAN trained for 10,000 iterations, and the LSTM-GAN trained for 5,000 iterations, due to resource constraints. For each training iteration, the generator was trained on 50 noise vectors and the discriminator was trained on 100 packets (50 real, 50 generated). Therefore, the MLP generator trained on 500,000 noise vectors and the discriminator trained on 1,000,000 packets in total for a training session. The LSTM

generator and discriminator trained on 250,000 noise vectors and 500,000 packets respectively.

In Experiment 3, we tracked the discriminator's loss on real packets, its loss metric on fake packets, its average loss on real and fake packets, and the generator's loss. We compared the discriminator's losses on real and fake packets in Figures 14 and 16, where fake packets were made by the generator. Using these losses, we computed the discriminator's average loss and compared this to the generator's loss (Figures 15 and 17). Note that the discriminator's losses on fake packets reflect the degree to which the generator successfully tricked the discriminator during training. Note that the Keras application programming interface negates and returns loss values after training a network, so loss values are shown as positive in Experiment 3 results.

*a. Long Short-Term Memory CPSGAN Training*

The LSTM model had inconsistent behavior across experiments. Figure 14 shows the discriminator's averaged loss on a batch of real or fake packets per iteration of GAN training. Spikes in Figure 14 indicate when the discriminator (D) had trouble with real or generated instances. On the left (ICSICL), the discriminator was initially confused between real and fake packets, then showed decreasing loss on the generator's fake packets. This is strange behavior but could be due to dual-meaning labels (fake and malicious). The discriminator's losses in Experiment 3B are more consistent with our expected result, but no clear improvement occurs.

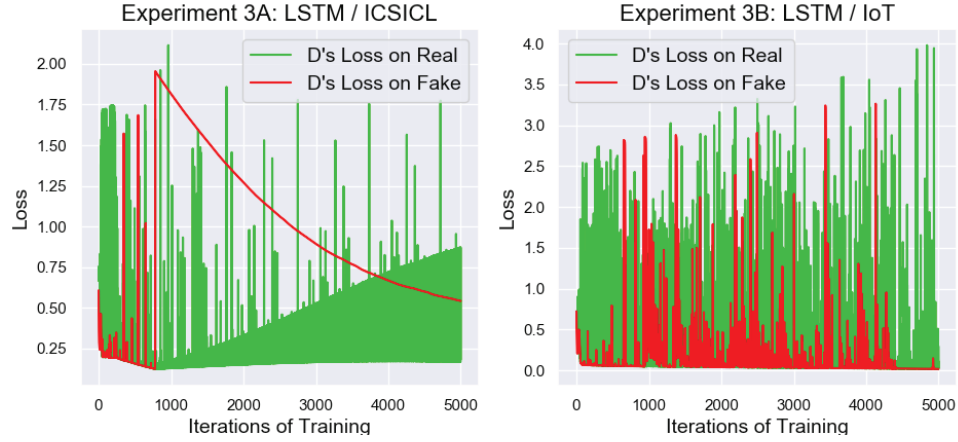


Figure 14. Experiments 3A and 3B: LSTM Discriminator Losses on Real and Fake Data

Figure 15 shows how the losses of the discriminator (D) and generator (G) relate during training. In Experiment 3A (left), a big change occurred around the 1,000<sup>th</sup> training iteration. After this, the generator's loss climbed steadily. The Experiment 3B network losses on the IoT dataset (right) show that the generator diverged from the discriminator. At the same time, the generator made some lucky guesses, indicated by the sudden decreases in its losses during training.

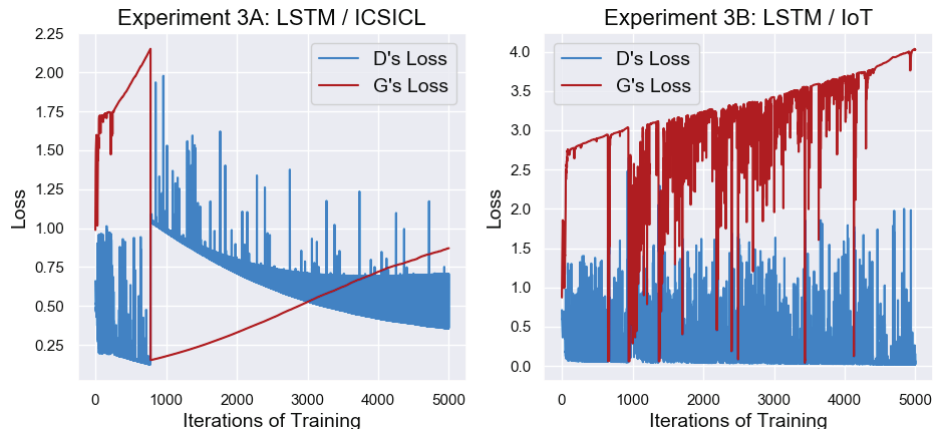


Figure 15. Experiments 3A and 3B: LSTM Discriminator and Generator Losses



***b. Multilayer Perceptron CPSGAN Training***

The MLP-GAN's training was more balanced than the LSTM-GAN. The loss graphs in Figure 16 indicate that the generator succeeded in confusing the discriminator at certain periods over time. The discriminator was relatively unfazed with fake ICSICL packets until the end of the training session (3A, late in training around the 10,000<sup>th</sup> iteration). The generator performed better when the discriminator was confused. The discriminator showed more consistent losses on fake IoT packets (right, Experiment 3B).

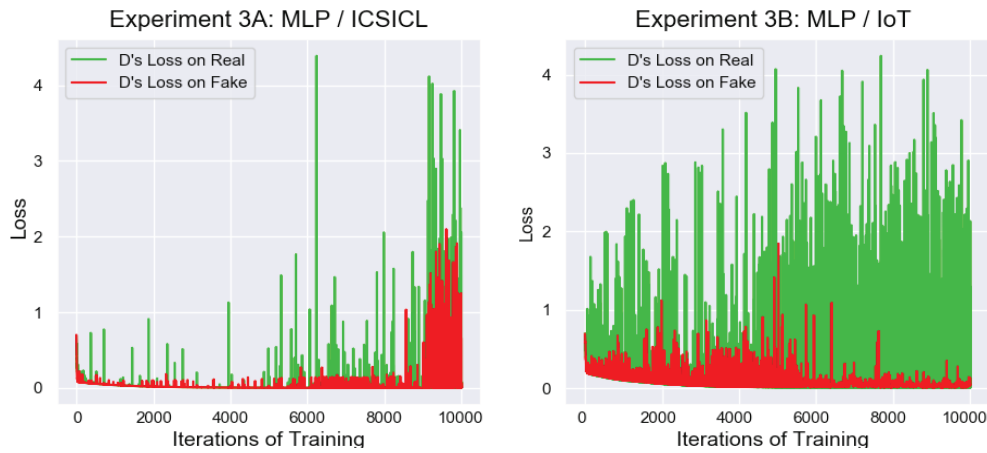


Figure 16. Experiments 3A and 3B: MLP Discriminator Losses on Real and Fake Data

For the MLP architecture in Figure 17, particularly in the right graph of Experiment 3B (IoT), the generator and discriminator losses showed early signs of convergence, though the generator's loss was increasing overall. The Experiment 3A loss graph in Figure 17 (left, ICSICL) show that the MLP-GAN struggled to converge on ICSICL data when compared to the Experiment 3B. The MLP generator in Experiment 3A recorded the highest loss of the Experiment 3 evaluations, exceeding a value of 7. This indicates that it had a hard time during training.

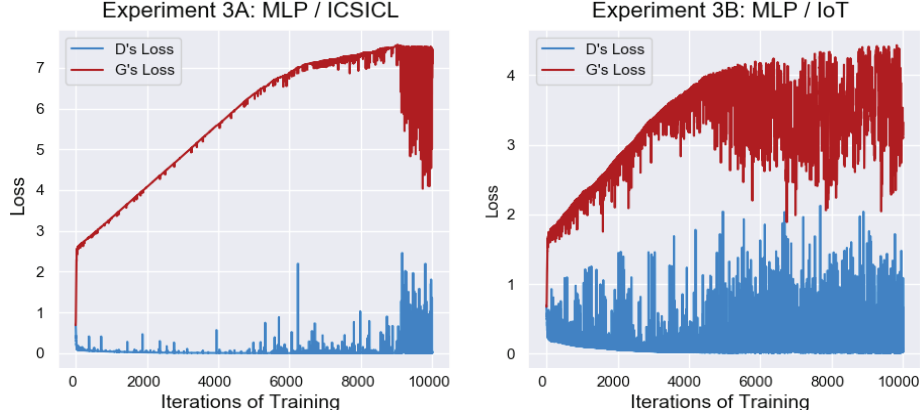


Figure 17. Experiments 3A and 3B: MLP Discriminator and Generator Losses

#### 4. Experiment 4: Evaluation of GAN-Trained Discriminators on Real Data

For Experiments 4A and 4B, we evaluated the Experiment 3 discriminators on 20% of the real dataset reserved for testing and this contained both benign and malicious instances. Table 15 shows the number of true positives, false positives, and false negatives that each discriminator made on the datasets. In Table 16, the LSTM discriminator of Experiment 4A predicted that all packets in the ICSICL test set were malicious, but the MLP model showed improved performance. Metrics that improved from Experiment 1 are in bold in Table 16.

Table 15. Experiments 4A and 4B: Frequency Statistics of GAN-Trained Discriminators

<i>Exp.</i>	<i>GAN Discriminator</i>	<i>Train Set (Combined)</i>	<i>Test Set (20%)</i>	<i>True Pos.</i>	<i>False Pos.</i>	<i>False Neg.</i>	<i>Tested Instances</i>
4A	LSTM	ICSICL (real + generated)	ICSICL (real)	7790	4560	0	12350
4A	MLP	ICSICL (real + generated)	ICSICL (real)	7642	109	148	12350
4B	LSTM	IoT (real + generated)	IoT (real)	134624	7666	8258	422300
4B	MLP	IoT (real + generated)	IoT (real)	119632	12341	23250	422300

Table 16. Experiments 4A and 4B: Performance Metrics of GAN-Trained Discriminators

<i>Exp.</i>	<i>GAN Discriminator</i>	<i>Train Set (Combined)</i>	<i>Test Set (20%)</i>	<i>Acc.</i>	<i>Rec.</i>	<i>Prec.</i>	<i>F1-Score</i>	<i>False Pos. Rate</i>
4A	LSTM	ICSICL (real + generated)	ICSICL (real)	0.631	1.000	0.631	0.774	1.000
4A	MLP	ICSICL (real + generated)	ICSICL (real)	<b>0.979</b>	<b>0.981</b>	<b>0.986</b>	<b>0.983</b>	0.024
4B	LSTM	IoT (real + generated)	IoT (real)	0.962	0.942	<b>0.946</b>	0.944	<b>0.027</b>
4B	MLP	IoT (real + generated)	IoT (real)	0.916	0.837	0.906	0.871	0.044

The precision-recall curves in Figure 18 show that the Experiment 4A discriminator's average precision (AP) was high, despite its high false-positive rate and

low F1-score. The tradeoff between precision and recall is steeper for the MLP model on the IoT dataset than the other models in Experiments 4A and 4B.

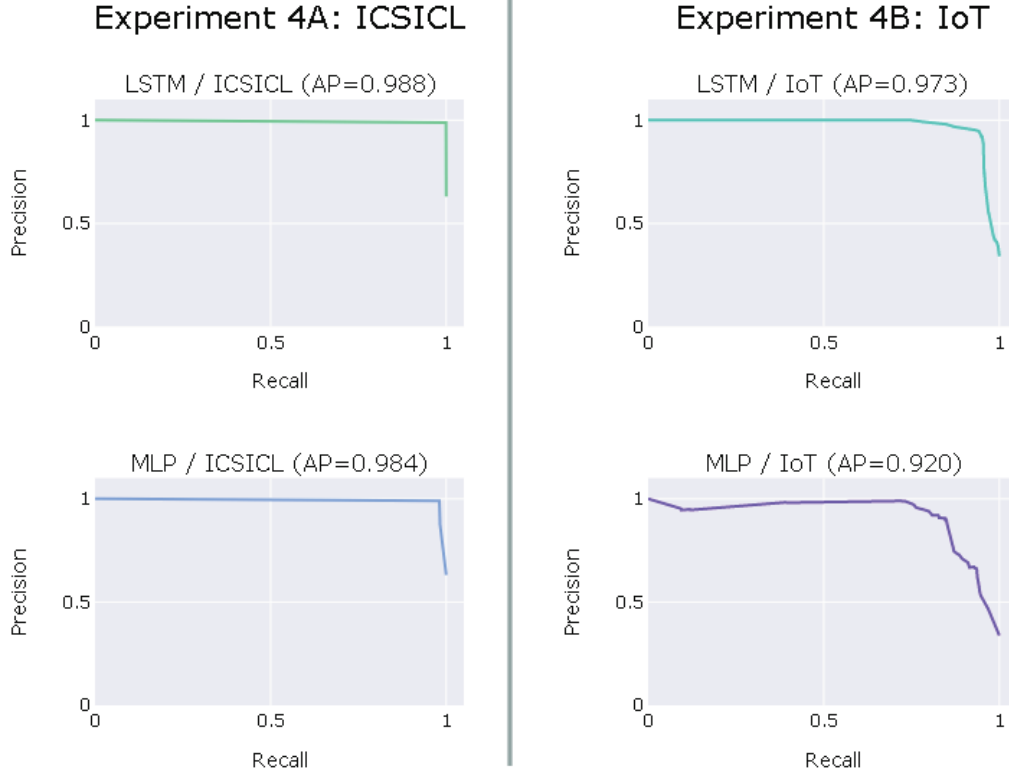


Figure 18. Experiments 4A and 4B: Precision-Recall Curves for GAN-Trained Discriminators

## 5. Experiment 5: Evaluation of Pretrained Standalone Discriminators on Generated Data

For this experiment, we reused the discriminator weights learned in Experiments 1A and 1B, where each discriminator was trained as a binary classifier on the benign and malicious packets of a dataset. In Experiments 5A and 5B, half of the testing data were benign packets pulled from the 20% of the dataset reserved for testing, and the other half were supplied by the trained generator from Experiments 3A and 3B. The generated packets were given malicious labels. This experiment tested how a trained neural network

classifier responded to packets from the generator, as it was trying to mimic anomalous packets from the same network.

In Experiment 5A of Table 17, the LSTM model detected 4545 of the 4550 generated ICSICL packets in the combined dataset, which is better than the MLP model. In Experiment 5B, the LSTM recorded many false negatives and was fooled by over 99% of the generated IoT instances in the test set. On the other hand, the MLP discriminator recorded much fewer false negatives, so it did significantly better in Experiment 5B than the LSTM discriminator.

Table 17. Experiments 5A and 5B: Frequency Statistics of Standalone Discriminators Evaluated on Benign and Generated Data

<i>Exp.</i>	<i>Standalone Discriminator</i>	<i>Train Set (80%)</i>	<i>Test Set (Combined)</i>	<i>True Pos.</i>	<i>False Pos.</i>	<i>False Neg.</i>	<i>Tested Instances</i>
5A	LSTM	ICSICL (real)	ICSICL (real + generated)	4545	52	5	9100
5A	MLP	ICSICL (real)	ICSICL (real + generated)	2818	66	1732	9100
5B	LSTM	IoT (real)	IoT (real + generated)	720	8418	278680	558800
5B	MLP	IoT (real)	IoT (real + generated)	266859	6355	12541	558800

The results in Table 18 suggest that each GAN architecture, MLP or LSTM, has advantages in certain kinds of datasets when it comes to packet generation. The LSTM-generated ICSICL instances had a negligible effect on the LSTM discriminator’s performance in Experiment 5A. Of the ICSICL packets produced by the MLP generator, about 38% fooled the MLP discriminator in Experiment 5A. As a result, the MLP discriminator lost almost 15% to accuracy and 33% to recall from Experiment 1A. Its decreased recall was due to a higher number of false negatives. Performance in Experiment 5B improved for the MLP discriminator, which could distinguish the fake packets easily. In contrast, the LSTM discriminator’s accuracy decreased from 96.9% (Experiment 1B) to less than 50% when generated packets were introduced. Its F1-score, recall, and precision from Experiment 1B were also significantly reduced.

Table 18. Experiments 5A and 5B: Performance Metrics of the Standalone Discriminators Evaluated on Generated Data

<i>Exp.</i>	<i>Standalone Discriminator</i>	<i>Train Set (80%)</i>	<i>Test Set (Combined)</i>	<i>Acc.</i>	<i>Rec.</i>	<i>Prec.</i>	<i>F1-Score</i>	<i>False Pos. Rate</i>
5A	LSTM	ICSICL (real)	ICSICL (real + generated)	0.994	0.999	0.989	0.994	0.011
5A	MLP	ICSICL (real)	ICSICL (real + generated)	0.802	0.619	0.977	0.758	0.015
5B	LSTM	IoT (real)	IoT (real + generated)	0.486	0.003	0.079	0.005	0.030
5B	MLP	IoT (real)	IoT (real + generated)	0.966	0.955	0.977	0.966	0.023

Figure 19 shows the precision-recall curves of each model on each dataset. The effect of the generated instances on the MLP discriminator in Experiment 5A is clearer when compared to the curves of the Experiment 5A LSTM and Experiment 5B MLP, which were the top two performers in this experiment.

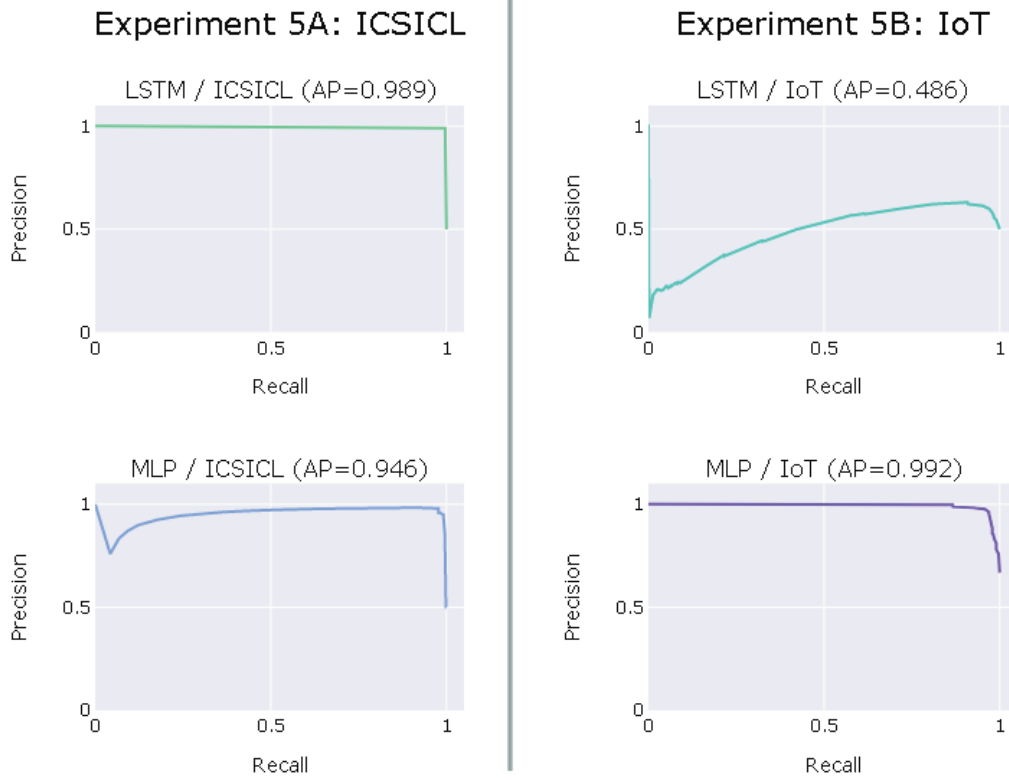


Figure 19. Experiments 5A and 5B: Precision-Recall of the Experiment 1 Discriminators Evaluated on Generated Data

THIS PAGE INTENTIONALLY LEFT BLANK



## **VI. CONCLUSIONS AND FUTURE WORK**

CPSGAN was motivated by the need for robust intrusion-detection mechanisms in cyber-physical systems. We used a supervised clustering approach and two semi-supervised GAN models: a long short-term memory (LSTM) network and multilayer perceptron (MLP). Each model was a binary classifier designed to detect anomalous traffic. We analyzed each model in their ability to detect attack packets in the ICSICL and IoT datasets, and the design appeared promising. It is possible for a GAN-based approach to improve classifier-based intrusion-detection methods.

### **A. SUMMARY OF FINDINGS**

Our results show that GANs could improve intrusion-detection systems for detecting anomalous CPS traffic. They are versatile in that any architecture or loss function can be used, but they come with different constraints and challenges [34].

We achieved significantly different results across experiments, possibly due to the effect each type of network traffic had on each model during training. Experiment 1 gave us a basis of performance for each model-dataset combination. Experiment 2 showed that it is harder to detect unknown packets either when the training data is more heterogeneous or has fewer features as in the IoT dataset. Experiments 3 and 4 indicated that it is possible for GAN-based training to improve classifier performance, as was the case with the MLP on ICSICL and LSTM on IoT tests. In Experiment 5, we concluded from the decreased performance of the standalone-trained discriminators (MLP on ICSICL and LSTM on IoT) that the generators from Experiment 3 could generate packets that fooled the two discriminators. In summary, two of our model-dataset combinations achieved the expected result.

### **B. FUTURE WORK**

Possible future work could use a convolutional neural network in the design of CPSGAN. This would permit traffic to be preprocessed as an image with early layers having local focus [8], [19], [25], [74]. Convolutional neural networks can be used for both

the discriminator and generator, or combined with long short-term memory architecture. Many examples of this architecture have been tried in intrusion-detection research [13], [14], [30].

Another possible type of GAN is the Wasserstein GAN (WGAN), which uses Wasserstein distance to train the GAN [33]. Wasserstein distance treats the discriminator as a “critic.” Instead of providing a probability of realness, the critic measures how real a sample seems. Wasserstein loss helps when we wish to stabilize training between the discriminator and generator. By providing a critique of realness, versus a classification, the gradients from Wasserstein loss tell the generator how far from the ground truth its generated output is. This way, the GAN is less likely to suffer from convergence failures during training, though this may cause slower training.

A LSTM-GAN network may benefit from a complex embedding structure and training algorithm to support backpropagation. It may be worth investigating and adapting models like the SeqGAN [75] and the MaskGAN [76]. A useful approach for input encoding is the Global Vectors for Word Representation (GloVe) training algorithm, which is an unsupervised method for numerically encoding words [68]. A similar approach has already been done specifically for packet data, called Packet2Vec [77].

## APPENDIX A. CIP TRAFFIC EXAMPLES

Wireshark provides a graphical user interface for network traffic analysis. Figure 20 shows an example in Wireshark of normal CIP communications between two devices in the ICSICL testbed. These packets show device status updates. Note that this traffic is very repetitive.

Source	Destination	Source Port	Dest Port	Protocol	Length	Info	Attribute
10.1.30.1	10.1.100.2	63483	44818	TCP	60	63483 → 44818 [ACK] Seq=327 Ack=4...	
10.1.30.1	10.1.100.2	63483	44818	CIP	380	Multiple Service Packet: Get Attr...	5,11,1,2,3,5,7,9,10,...
10.1.100.2	10.1.30.1	44818	63483	TCP	60	44818 → 63483 [ACK] Seq=431 Ack=6...	
10.1.100.2	10.1.30.1	44818	63483	CIP	484	Success: Multiple Service Packet:...	5,11,1,4,4,4,1,1,18,...
10.1.30.1	10.1.100.2	63483	44818	CIP	110	Class (0x73) - Get Attribute List	5
10.1.100.2	10.1.30.1	44818	63483	CIP	114	Success: Class (0x73) - Get Attri...	5
10.1.30.1	10.1.100.2	63483	44818	TCP	60	63483 → 44818 [ACK] Seq=709 Ack=9...	
10.1.30.1	10.1.100.2	63483	44818	CIP	380	Multiple Service Packet: Get Attr...	5,11,1,2,3,5,7,9,10,...
10.1.100.2	10.1.30.1	44818	63483	TCP	60	44818 → 63483 [ACK] Seq=921 Ack=1...	
10.1.100.2	10.1.30.1	44818	63483	CIP	484	Success: Multiple Service Packet:...	5,11,1,4,4,4,1,1,18,...
10.1.30.1	10.1.100.2	63483	44818	TCP	60	63483 → 44818 [ACK] Seq=1035 Ack=...	
10.1.30.1	10.1.100.2	63483	44818	CIP	380	Multiple Service Packet: Get Attr...	5,11,1,2,3,5,7,9,10,...
10.1.100.2	10.1.30.1	44818	63483	TCP	60	44818 → 63483 [ACK] Seq=1351 Ack=...	
10.1.100.2	10.1.30.1	44818	63483	CIP	484	Success: Multiple Service Packet:...	5,11,1,4,4,4,1,1,18,...
10.1.30.1	10.1.100.2	63483	44818	CIP	110	Class (0x73) - Get Attribute List	5
10.1.100.2	10.1.30.1	44818	63483	CIP	114	Success: Class (0x73) - Get Attri...	5
10.1.30.1	10.1.100.2	63483	44818	TCP	60	63483 → 44818 [ACK] Seq=1417 Ack=...	
10.1.30.1	10.1.100.2	63483	44818	CIP	380	Multiple Service Packet: Get Attr...	5,11,1,2,3,5,7,9,10,...
10.1.100.2	10.1.30.1	44818	63483	TCP	60	44818 → 63483 [ACK] Seq=1841 Ack=...	

Figure 20. Example of Benign CIP Traffic

Figure 21 shows traffic from a denial-of-service attack using the Register Session command, which caused a silent shutdown of the PLC. An attacker at IP address 10.1.40.1 continuously sent Register Session requests to the PLC at 100.1.100.2, and the PLC responded to each request by producing a unique session ID, called a Session Handle. The opened sessions were not terminated, which eventually exhausted the PLC's session ID pool, causing it to stop responding to new Register Session requests.

Source	Destination	Source Port	Dest Port	Protocol	Length	Info
10.1.40.1	10.1.100.2	44853	44818	TCP	74	44853 → 44818 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SAC
10.1.100.2	10.1.40.1	44818	44852	TCP	66	44818 → 44852 [ACK] Seq=29 Ack=30 Win=4096 Len=0 TSval=
10.1.100.2	10.1.40.1	44818	44853	TCP	74	44818 → 44853 [SYN, ACK] Seq=0 Ack=1 Win=4096 Len=0 MS
10.1.40.1	10.1.100.2	44853	44818	TCP	66	44853 → 44818 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=
10.1.40.1	10.1.100.2	44853	44818	ENIP	94	Register Session (Req), Session: 0x00000000
10.1.100.2	10.1.40.1	44818	44852	TCP	60	44818 → 44852 [RST, ACK] Seq=29 Ack=30 Win=4096 Len=0
10.1.100.2	10.1.40.1	44818	44853	ENIP	94	Register Session (Rsp), Session: 0x19029810
10.1.40.1	10.1.100.2	44853	44818	TCP	66	44853 → 44818 [ACK] Seq=29 Ack=29 Win=29312 Len=0 TSva
10.1.40.1	10.1.100.2	44853	44818	TCP	66	44853 → 44818 [FIN, ACK] Seq=29 Ack=29 Win=29312 Len=0
10.1.40.1	10.1.100.2	44854	44818	TCP	74	44854 → 44818 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SAC
10.1.100.2	10.1.40.1	44818	44853	TCP	66	44818 → 44853 [ACK] Seq=29 Ack=30 Win=4096 Len=0 TSval
10.1.100.2	10.1.40.1	44818	44854	TCP	74	44818 → 44854 [SYN, ACK] Seq=0 Ack=1 Win=4096 Len=0 MS
10.1.40.1	10.1.100.2	44854	44818	TCP	66	44854 → 44818 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=
10.1.40.1	10.1.100.2	44854	44818	ENIP	94	Register Session (Req), Session: 0x00000000
10.1.100.2	10.1.40.1	44818	44853	TCP	60	44818 → 44853 [RST, ACK] Seq=29 Ack=30 Win=4096 Len=0
10.1.100.2	10.1.40.1	44818	44854	ENIP	94	Register Session (Rsp), Session: 0x19029910

Figure 21. Denial of Service Using the Register Session Command

## APPENDIX B. WIRESHARK FILTERS

TShark is the command-line interface for the packet analysis and capture tool Wireshark. Filter rules are logical statements passed to the Wireshark application programming interface and are for filtering packets of a specific characteristic [78]. The following filter rules are examples of rules for identifying attack packets from a PCAP file:

Host discovery:

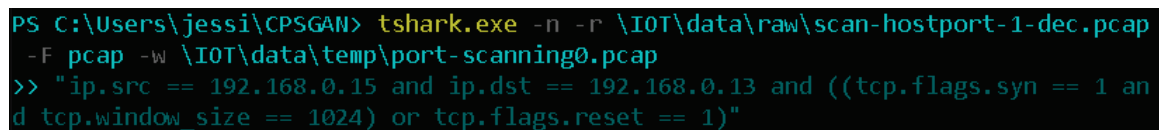
```
(eth.src == f0:18:98:5e:ff:9f
and arp
and eth.dst == ff:ff:ff:ff:ff:ff)
and frame.number < 13000
```

Port discovery:

```
ip.src == 192.168.0.15
and ip.dst == 192.168.0.13
and ( (tcp.flags.syn == 1
and tcp.window_size == 1024)
or tcp.flags.reset == 1)
```

The host discovery rule identifies ARP broadcast packets from the attacking device within the first 13,000 packets captured of that PCAP. The port discovery rule looks for packets with the TCP SYN flag set and specified window size or packets with TCP RST flag set, sent from 192.168.0.15 to 192.168.0.13.

When these filters are applied, Wireshark will iterate through a PCAP file and show only packets that match this condition. Using TShark and a write file command option, the preprocessor applies each filter to its associated PCAP file and stores the resulting packets into a new PCAP file. Figure 22 shows how a rule appears if run through a command-line interpreter (PowerShell in this case).



```
PS C:\Users\jessi\CPSGAN> tshark.exe -n -r \IoT\data\raw\scan-hostport-1-dec.pcap
-F pcap -w \IoT\data\temp\port-scanning0.pcap
>> "ip.src == 192.168.0.15 and ip.dst == 192.168.0.13 and ((tcp.flags.syn == 1 and
tcp.window_size == 1024) or tcp.flags.reset == 1)"
```

Figure 22. Application of a TShark Rule in a Command-line Interface

The preprocessor called TShark with the ‘-n’ and ‘-r’ options to disable name resolution and read from the PCAP file named scan-hostport-1-dec.pcap. The file we want to read, scan-hostport-1-dec.pcap, was defined after these options. The ‘-F pcap -w’ option tells TShark to write output to a PCAP file named port-scanning0.pcap, followed by the filter rule surrounded by quotes. The filter rule in Figure 22 was written by the IoT dataset’s creators.

For a PCAP file with only one attack, we isolated benign packets by encapsulating the attack rule in parentheses and appending “not” to the beginning of the rule. For a PCAP file with multiple attacks, we joined the multiple attack rules with an “or” operator and then apply the “not” and parenthesis to the combined attack rule. A simple example of this process is shown in Table 19.

Table 19. Creating a Benign Filter from Multiple Attack Filters

<i>Attack Rule 1</i>	<code>(ip.addr == 10.1.40.1)</code>
<i>Attack Rule 2</i>	<code>(frame.number &gt; 2000)</code>
<i>Benign Rule</i>	<code>not ((ip.addr == 10.1.40.1) or (frame.number &gt; 2000))</code>

## APPENDIX C. CPSGAN MODELS

```
# *****
#
# Long Short-Term Memory models
#
# *****

from tensorflow.keras import Input
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Bidirectional
from tensorflow.keras.layers import Embedding
from tensorflow.keras.models import Model

#   D I S C R I M I N A T O R
def build_discriminator(b, m):
    d_shape = (m, 256)
    d_batch = (b, m, 256)

    x = Input(batch_input_shape=d_batch)
    h = Bidirectional(LSTM(128, input_shape=d_shape, return_sequences=True))(x)
    h = LSTM(80, return_sequences=True)(h)
    h = Flatten()(h)
    h = Dense(2, activation='tanh')(h)
    out = Dense(1, activation='sigmoid')(h)
    model = Model(x, out, name='LSTM_Discriminator')
    return model, d_shape, d_batch

#   G E N E R A T O R
def build_generator(b, m):
    g_shape = (m, 256)
    g_batch = (b, *g_shape)

    z = Input(batch_input_shape=g_batch)
    h = Bidirectional(LSTM(128, input_shape=g_shape, return_sequences=True))(z)
    Gz = Dense(256, activation='relu')(h)
    model = Model(z, Gz, name='LSTM_Generator')
    return model, g_shape, g_batch
```

```

# *****
#
#   MultiLayer Perceptron Models
#
# *****

from tensorflow.keras import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dropout
from tensorflow.keras.models import Model

#   D I S C R I M I N A T O R
def build_discriminator(b, m):
    d_shape = (m,)
    d_batch = (b, m,)

    x = Input(batch_input_shape=d_batch)
    h = Dense(100, activation='relu')(x)
    h = Dense(200, activation='relu')(h)
    h = Dense(400, activation='relu')(h)
    h = Dense(200, activation='relu')(h)
    h = Dense(100, activation='relu')(h)
    h = Dense(m, activation='relu')(h)
    out = Dense(2, activation='tanh')(h)
    Dx = Dense(1, activation='sigmoid')(out)
    model = Model(x, Dx, name='MLP_Discriminator')
    return model, d_shape, d_batch

#   G E N E R A T O R
def build_generator(b, m):
    g_shape = (m,)
    g_batch = (b, *g_shape)

    z = Input(batch_input_shape=g_batch)
    h = Dense(100, activation='relu', input_dim=g_shape)(z)
    h = BatchNormalization(momentum=0.8)(h)
    h = Dense(200, activation='relu')(h)
    h = BatchNormalization(momentum=0.8)(h)
    h = Dense(400, activation='relu')(h)
    h = BatchNormalization(momentum=0.8)(h)
    h = Dense(200, activation='relu')(h)
    h = BatchNormalization(momentum=0.8)(h)
    h = Dense(100, activation='relu')(h)
    h = BatchNormalization(momentum=0.8)(h)
    h = Dense(50, activation='relu')(h)
    h = BatchNormalization(momentum=0.8)(h)
    Gz = Dense(m, activation='tanh')(h)
    model = Model(z, Gz, name='MLP_Generator')
    return model, g_shape, g_batch

```



## LIST OF REFERENCES

- [1] C. Greer, M. Burns, D. Wollman, and E. Griffor, “Cyber-physical systems and internet of things,” National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 1900–202, Mar. 2019. doi: 10.6028/NIST.SP.1900-202
- [2] T. Polk, M. Souppaya, B. Haag, and W. C. Barker, “Mitigating IoT-Based Distributed Denial of Service (DDoS),” U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, 2017.
- [3] J.R. Minkel, “The 2003 northeast blackout, five years later,” *Scientific American*, Aug. 13, 2008. [Online]. Available: <https://www.scientificamerican.com/article/2003-blackout-five-years-later/>
- [4] A. Humayed, J. Lin, F. Li, and B. Luo, “Cyber-physical systems security – A survey,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1802–1831, 2017.
- [5] J. A. Dlouhy and M. Riley, “Russian hackers attacking U.S. power grid and aviation, FBI warns,” *Bloomberg*, Mar. 15, 2018. [Online]. Available: <https://www.bloomberg.com/news/articles/2018-03-15/russian-hackers-attacking-u-s-power-grid-aviation-fbi-warns>
- [6] R. H. Hwang, M. C. Peng, V. L. Nguyen, and Y. L. Chang, “An LSTM-based deep learning approach for classifying malicious traffic at the packet level,” *Applied Sciences*, vol. 9, no. 16, pp. 3414, Aug. 2019. [Online]. doi: 10.3390/app9163414
- [7] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein *et al.*, “Understanding the Mirai botnet,” in *Proceedings of the 26<sup>th</sup> USENIX Security Symposium*, 2017. [Online]. Available: <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf>
- [8] Z. Lin, Y. Shi, and Z. Xue, “IDSGAN: Generative adversarial networks for attack generation against intrusion detection,” 2019. [Online]. Available: [arXiv:1809.02077](https://arxiv.org/abs/1809.02077)
- [9] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, “GANomaly: Semi-supervised anomaly detection via adversarial training,” in *Proceedings of the Asian Conference on Computer Vision*, pp. 622–637, 2018. [Online]. doi: 10.1007/978-3-030-20893-6\_39
- [10] I. J. Goodfellow *et al.*, “Generative adversarial networks,” in *Proceedings of the 24<sup>th</sup> International Conference on Neural Information Processing Systems*, vol. 2, pp. 2672–2680, Dec. 2014. [Online]. doi: 10.5555/2969033

- [11] H. Kang, D. H. Ahn, G. M. Lee, J. D. Yoo, K. H. Park, and H. K. Kim, Internet of Things Network Intrusion Dataset. 2019. [Online]. Available: <https://ieee-dataport.org/open-access/iot-network-intrusion-dataset>
- [12] T. D. Nguyen and C. E. Irvine, “Development of industrial network forensics lessons,” in *Cybersecurity Symposium*, Coeur d’Alene, Idaho, 2017. [Online]. Available: <http://hdl.handle.net/10945/56212>
- [13] E. Seo, H. M. Song, and H. K. Kim, “GIDS: GAN based intrusion detection system for in-vehicle network,” *2018 16th Annual Conference Privacy, Security, and Trust*, pp. 1–6, Aug. 2018. [Online]. doi: 10.1109/PST.2018.8514157
- [14] M. Kravchik and A. Shabtai, “Detecting cyberattacks in industrial control systems using convolutional neural networks,” in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and Privacy*, Association for Computing Machinery, New York, USA, 2018. [Online]. doi: 10.1145/3264888.3264896
- [15] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Burlington, MA, USA: Morgan Kaufmann, 2011.
- [16] D. Wilson, Y. Tang, J. Yan, and Z. Lu, “Deep learning-aided cyber-attack detection in power transmission systems,” in *2018 IEEE Power & Energy Society General Meeting*, 2018. [Online]. doi: 10.1109/PESGM.2018.8586334
- [17] Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time-series,” in *The Handbook of Brain Theory and Neural Networks*, M.A. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1998, pp. 255–258.
- [18] H. Gwon, C. Lee, R. Keum, and H. Choi, “Network intrusion detection based on LSTM and feature embedding,” 2019. [Online]. Available: [arXiv:1911.11552](https://arxiv.org/abs/1911.11552)
- [19] Lim, Kim, Kim, Hong, and Han, “Payload-based traffic classification using multi-layer LSTM in software defined networks,” *Applied Sciences*, vol. 9, no. 12, pp. 2550, Jun. 2019. [Online]. doi: 10.3390/app9122550
- [20] G. Liu, H. Bao, and B. Han, “A stacked autoencoder-based deep neural network for achieving gearbox fault diagnosis,” *Mathematical Problems in Engineering*, vol. 2018, pp. 1–10, Jul. 2018. [Online]. doi: 10.1155/2018/5105709
- [21] M. A. Kramer, “Nonlinear principal component analysis using autoassociative neural networks,” *AIChE Journal*, vol. 37, no. 2, pp. 11, 1991.
- [22] I. Goodfellow, “NIPS 2016 tutorial: Generative adversarial networks,” 2017. [Online]. Available: [arXiv:1701.00160](https://arxiv.org/abs/1701.00160)

- [23] S. Hochreiter and J. Schmidhuber, “Long short term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [24] M. Cheng, Q. Xu, J. L. V., W. Liu, Q. Li, and J. Wang, “MS-LSTM: A multi-scale LSTM model for BGP anomaly detection,” in *2016 IEEE 24th International Conference on Network Protocols*, Singapore, Nov. 2016, pp. 1–6, doi: 10.1109/ICNP.2016.7785326
- [25] Y. Lai, J. Zhang, and Z. Liu, “Industrial anomaly detection and attack classification method based on convolutional neural network,” *Security and Communications Networks*, vol. 2019, pp. 1–11, Sep. 2019. [Online]. doi: 10.1155/2019/8124254
- [26] N. Hammoud, “Game theory: minimax, maximin, and iterated removal.” Lecture at University of Oxford, UK. March 14, 2017. [Online]. Available: [http://people.maths.ox.ac.uk/griffit4/Math\\_Alive/3/game\\_theory3.pdf](http://people.maths.ox.ac.uk/griffit4/Math_Alive/3/game_theory3.pdf)
- [27] J. Donahue, P. Krähenbühl, and T. Darrell, “Adversarial feature learning,” in *Proceedings of the 5<sup>th</sup> International Conference on Learning Representations*, Toulon, FR, April 2017. [Online]. Available: arXiv:1605.09782
- [28] F. Di Mattia, P. Galeone, M. De Simoni, and E. Ghelfi, “A survey on GANs for anomaly detection,” 2019. [Online]. Available: arXiv:1906.11632
- [29] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, “Efficient GAN-based anomaly detection,” 2019. [Online]. Available: <https://openreview.net/forum?id=BkXADmJDM>
- [30] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery,” *Proceedings of the International Conference on Information Processing in Medical Imaging*, Boone, NC, June 2017. [Online]. Available: arXiv:1703.05921
- [31] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Ottawa, ON, Canada, Jul. 2009, pp. 1–6. [Online]. doi: 10.1109/CISDA.2009.5356528
- [32] University of New Brunswick, NSL-KDD Dataset. [Online]. Available: <https://www.unb.ca/cic/datasets/nsl.html>
- [33] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN,” *Proceedings of the 34<sup>th</sup> International Conference on Machine Learning*, vol. 70, pp. 214–223, 2017. [Online]. doi: 10.5555/3305381.3305404

- [34] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, “Are GANs created equal? A large-scale study,” *Advances in Neural Information Processing Systems*, pp. 700–709, 2018. [Online]. Available: arXiv:1711.10337
- [35] E. R. Griffor, C. Greer, D. A. Wollman, and M. J. Burns, “Framework for cyber-physical systems: Volume 1, overview,” National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 1500–201, Jun. 2017. [Online]. doi: 10.6028/NIST.SP.1500-201
- [36] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, “Guide to industrial control systems (ICS) security,” National Institute of Standards and Technology, NIST SP 800-82r2, Jun. 2015. [Online]. doi: 10.6028/NIST.SP.800-82r2
- [37] Open DeviceNet Vendor Association, Inc., *The CIP Networks Library Volume 2: EtherNet/IP Adaptation of CIP*, 1.23 ed., Open DeviceNet Vendor Association, Inc., Ann Arbor, MI, USA, 2017.
- [38] Open DeviceNet Vendor Association, Inc., *The CIP Networks Library Volume 1: Common Industrial Protocol*, 3.22 ed., Open DeviceNet Vendor Association, Inc., Ann Arbor, MI, USA, 2017.
- [39] N. H. Desso, “Designing a machinery control system (MCS) security testbed,” M.S. thesis, Dept. of Comp. Sci., NPS, Monterey, CA, USA, 2014. [Online]. Available: <http://hdl.handle.net/10945/43902>
- [40] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *2010 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, pp. 305–316, 2010. [Online]. doi: 10.1109/SP.2010.25
- [41] R. Mitchell and I.R. Chen, “A survey of intrusion detection techniques for cyber-physical systems,” *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–29, Mar. 2014. [Online]. doi: 10.1145/2542049
- [42] N. B. Carr, “Development of a tailored methodology and forensic toolkit for industrial control systems incident response,” M.S. thesis, Cyber Systems and Operations, NPS, Monterey, CA, USA, 2014. [Online]. Available: <http://hdl.handle.net/10945/42595>
- [43] E. R. Griffor, C. Greer, D. A. Wollman, and M. J. Burns, “Framework for cyber-physical systems: volume 2, working group reports,” National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 1500–202, Jun. 2017. [Online]. doi: 10.6028/NIST.SP.1500-202

- [44] F. Tacliad, T. D. Nguyen, and M. Gondree, “DoS Exploitation of Allen-Bradley’s legacy protocol through fuzz testing,” in *Proceedings of the 3rd Annual Industrial Control System Security Workshop*, San Juan PR USA, Dec. 2017, pp. 24–31. [Online]. doi: 10.1145/3174776.3174780
- [45] A. Giaretta, N. Dragoni, and F. Massacci, “IoT security configurability with security-by-contract,” *Sensors*, vol. 19, no. 19, Sep. 2019, doi: 10.3390/s19194121
- [46] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains,” *Leading Issues in Information Warfare & Security Research*, vol. 1 pp. 80, 2011.
- [47] D. Hyun, “Collecting cyberattack data for industrial control systems using honeypots,” M.S. thesis, Department of Computer Science, NPS, Monterey, CA, USA, 2018. [Online]. Available: <http://hdl.handle.net/10945/58316>
- [48] MITRE ATT&CK, “OilRig, IRN2, HELIX KITTEN, APT34, Group G0049,” July 4, 2020. [Online]. Available: <https://attack.mitre.org/groups/G0049/>
- [49] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, vol. 385. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.
- [50] M. Rigaki and S. Garcia, “Bringing a GAN to a knife-fight: Adapting malware communication to avoid detection,” in *2018 IEEE Security and Privacy Workshops*, May 2018, pp. 70–75. [Online]. doi: 10.1109/SPW.2018.00019
- [51] D. Li, D. Chen, L. Shi, B. Jin, J. Goh, and S.-K. Ng, “MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks,” 2019. [Online]. Available: [arXiv:1901.04997](https://arxiv.org/abs/1901.04997)
- [52] D. Li, D. Chen, J. Goh, and S. Ng, “Anomaly detection with generative adversarial networks for multivariate time series,” *Sensors*, vol. 20, no. 2, pp. 3336, 2019. [Online]. Available: [arXiv:1809.04758](https://arxiv.org/abs/1809.04758)
- [53] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A search space odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017. [Online]. doi: 10.1109/TNNLS.2016.2582924
- [54] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with LSTM,” *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000. [Online]. doi: 10.1162/089976600300015015

- [55] P. Wang, S. Li, F. Ye, Z. Wang, and M. Zhang, “PacketCGAN: Exploratory study of class imbalance for encrypted traffic classification using CGAN,” *IEEE International Conference on Communications*, Dublin, IR, 2020. [Online]. doi: 10.1109/ICC40277.2020.9148946
- [56] Allen Bradley, Inc., *ControlLogix System User Manual*, Allen Bradley, Milwaukee, MI, USA, 2017. [Online]. Available: [https://literature.rockwellautomation.com/idc/groups/literature/documents/um/1756-um001\\_-en-p.pdf](https://literature.rockwellautomation.com/idc/groups/literature/documents/um/1756-um001_-en-p.pdf)
- [57] Allen Bradley, Inc., *1756 ControlLogix Communication Modules Specifications*, Allen Bradley, Milwaukee, MI, USA, 2020. [Online]. Available: [https://literature.rockwellautomation.com/idc/groups/literature/documents/td/1756-td003\\_-en-e.pdf](https://literature.rockwellautomation.com/idc/groups/literature/documents/td/1756-td003_-en-e.pdf)
- [58] Prosoft Technology, Inc. *MVI56E-GSC/GSCXT CompactLogix or MicroLogix*, Prosoft Technology, Bakersfield, CA, USA, 2019. [Online]. Available: [https://www.prosoft-technology.com/prosoft/download/2900/22366/version/14/file/MVI56E\\_GSC\\_GSCXT\\_UM001.pdf](https://www.prosoft-technology.com/prosoft/download/2900/22366/version/14/file/MVI56E_GSC_GSCXT_UM001.pdf)
- [59] Allen Bradley, Inc., *Studio 5000 View Designer User Manual*, Allen Bradley, 2019. [Online]. Available: [https://literature.rockwellautomation.com/idc/groups/literature/documents/um/9324-um001\\_-en-d.pdf](https://literature.rockwellautomation.com/idc/groups/literature/documents/um/9324-um001_-en-d.pdf)
- [60] Gerald Combs and the Wireshark Team, USA. 2019. The Wireshark Network Analyzer, ver. 3.2.1. [Online]. Available: <https://www.wireshark.org/>
- [61] F. Chollet. Mountainview, CA, USA. 2020. Keras: The Python Deep Learning API, ver. 2.3.1. [Online]. Available: <https://keras.io>
- [62] J. Postel, “Internet protocol, DARPA internet program protocol specification,” Sep. 1981. [Online]. Available: <https://tools.ietf.org/html/rfc791>
- [63] Travis E. Oliphant, USA. 2020. NumPy, ver. 1.16.0. [Online]. Available: <https://numpy.org/>
- [64] Wes McKinney, USA. 2020. pandas – Python Data Analysis Library, ver. 0.25.3. [Online]. Available: <https://pandas.pydata.org/>
- [65] Dug Song, USA. 2020. dpkt Python Module, ver. 1.9.2. [Online]. Available: <https://dpkt.readthedocs.io/en/latest/>
- [66] Bell Labs, *Unix Programmer’s Manual*, Murray Hill, NJ, USA, 1971. [Online]. Available: <https://www.bell-labs.com/usr/dmr/www/pdfs/man22.pdf>

- [67] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [68] Jeffrey Pennington, Richard Socher, and Christopher D. Manning, Stanford, CA, USA. 2020. GloVe: Global Vectors for Word Representation. [Online]. Available: <https://nlp.stanford.edu/projects/glove/>
- [69] Internet Assigned Numbers Authority, “Service name and transport protocol port number registry.” [Online]. Available: <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>
- [70] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation functions: Comparison of trends in practice and research for deep learning,” 2018. [Online]. Available: arXiv:1811.03378
- [71] A. Sherstinsky, “Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network,” *Physica D: Nonlinear Phenomena*, Mar. 2020. [Online]. doi: 10.1016/j.physd.2019.132306
- [72] N. Tavakoli, “Modeling genome data using bidirectional LSTM,” in *2019 IEEE 43rd Annual Computation Software and Application Conference*, Milwaukee, WI, USA, Jul. 2019, pp. 183–188. [Online]. doi: 10.1109/COMPSAC.2019.10204
- [73] J. Davis and M. Goadrich, “The relationship between precision-recall and ROC curves,” in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, USA, 2006, pp. 233–240. [Online]. doi: 10.1145/1143844.1143874
- [74] L. Yu, W. Zhang, J. Wang, and Y. Yu, “SeqGAN: Sequence generative adversarial nets with policy gradient,” in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017. [Online]. doi: 10.5555/3298483.3298649
- [75] W. Fedus, I. Goodfellow, and A. M. Dai, “MaskGAN: Better text generation via filling in the \_\_\_\_\_,” *International Conference on Learning Representations*, 2018. [Online]. Available: <https://openreview.net/forum?id=ByOExmWAb>
- [76] E. L. Goodman, C. Zimmerman, and C. Hudson, “Packet2Vec: Utilizing Word2Vec for feature extraction in packet data,” Sandia National Laboratories, Livermore, CA, USA, No. SAND2019-0818C, 2020. [Online]. Available: <https://www.osti.gov/servlets/purl/1596405>
- [77] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, “Network traffic classifier with convolutional and recurrent neural networks for internet of things,” *IEEE Access*, vol. 5, pp. 18042–18050, 2017. [Online]. doi: 10.1109/ACCESS.2017.2747560

- [78] G. Combs. “Wireshark display filter reference.” The Wireshark Team, 2020.  
[Online]. Available: <https://www.wireshark.org/docs/dfref/>



## **INITIAL DISTRIBUTION LIST**

1. Defense Technical Information Center  
Ft. Belvoir, Virginia
2. Dudley Knox Library  
Naval Postgraduate School  
Monterey, California