Omni SCADA Intrusion Detection

by

Jun Gao
B. Eng, North China University of Technology, 2016

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Omni SCADA Intrusion Detection

by

Jun Gao
B. Eng, North China University of Technology, 2016

Supervisory Committee

Dr. Tao Lu, Supervisor
(Department of Electrical and computer engineering)

Dr. Issa Traore, Departmental Member
(Department of Electrical and computer engineering)

Supervisory Committee

Dr. Tao Lu, Supervisor
(Department of Electrical and computer engineering)

Dr. Issa Traore, Departmental Member
(Department of Electrical and computer engineering)

ABSTRACT

We investigate deep learning based omni intrusion detection system (IDS) for supervisory control and data acquisition (SCADA) networks that are capable of detecting both temporally uncorrelated and correlated attacks. Regarding the IDSs developed in this paper, a feedforward neural network (FNN) can detect temporally uncorrelated attacks at an F_1 of 99.967±0.005% but correlated attacks as low as 58±2%. In contrast, long-short term memory (LSTM) detects correlated attacks at 99.56±0.01% while uncorrelated attacks at 99.3±0.1%. Combining LSTM and FNN through an ensemble approach further improves the IDS performance with F_1 of 99.68±0.04% regardless the temporal correlations among the data packets.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My family and my friends for supporting me in the low moments. I would like to thank my parents for their endless support and my girlfriend Xiaoxuan Yu for her love.

Supervisor Dr. Tao Lu, for mentoring, support, encouragement, and patience. I benefited from his advice at many stages of my research. Furthermore, his positive outlook has a great inspiration to me.

Dr. Issa Traoré for his insightful courses that lead me to the security field and helpful comments.

Dr. Xuekui Zhang for acting as my supervisory committee member and insightful comments.

Fortinet Inc., for funding me with a Scholarship.

DEDICATION

To my family and friends.

Chapter 1

Introduction

1.1 Context

Supervisory control and data acquisition (SCADA) is a well established industrial control system (ICS) to automate/monitor processes and to gather data from remote or local equipment such as programmable logic controller (PLC), remote terminal units (RTU) and human-machine-interfaces (HMI), etc. SCADA became popular in the 60's for power plants, water treatment [1], and oil pipelines [3], etc. A typical SCADA diagram is shown in Fig. 1.1. The components connected by the blue line is the high level structure of a SCADA system. The "SCADA" at the top layer connects to the server and Modbus Gateway directly by using Eternet (TCP/IP) in order to monitor and control the whole system. The high level is used for internet connection and sends commands to the low level components connected by red lines. The low level components are in charge of industrial operation and controlling. In this Figure the low level components contains embedded controller, PLC and HMI. Assuming this SCADA system is controlling a water tank system, the water pump motor speed is controlled by a PID controller which loads on a embedded controller. The PLC is receiving the water level values from the sensors and transfer this value to SCADA through Modbus Gateway and the HMI. The HMI can translate the received bytes value to readable values then people can read and monitor the water tanks. The converter would transfer the commands from external user or internet to control the water tanks. This is a typical working process of modern SCADA system. However, the original SCADA did not have so many function. It only has a few temperature sensors, and final control elements. It was trivial to maintain

and modify the system because of two reasons: Firstly, the protocols used by the SCADA were mostly proprietary. Only a few experts who master those protocols can make changes of the SCADA system. The second reason is that the access of the SCADA is restricted. The technician should monitor the SCADA manually and locally. Physical access control is secure enough as no internet was connected to SCADA. With the evolution of industrial control system, common communication protocols for SCADA were developed and the structure of SCADA are becoming complicated and powerful. Distributed network protocol (DNP3), IEC 60870-5 and Modbus. Modbus is a serial communication protocols published 1979 for controlling PLCs. Modbus supports master-slave mode which is well suitable in SCADA system. The HMI plays a role modbus master which sends commands to the slaves that are normally PLCs or RTUs. The advantage of SCADA is that it's easy to deploy and to implement. Modbus has a few variants: Modbus/RTU, Modbus/ASCII and Modbus/TCP etc. Modbus/RTU is primary used in used on asynchronous serial data lines like RS-485. Modbus/ASCII can be seen on 7- or 8-bit asynchronous serial lines. Modbus/TCP is encapsulated on the Ethernet which enables internet connection. Modbus/RTU and Modbus/ASCII were implemented for a long time and Modbus/TCP is a relatively new variant especially for internet connectivity. The SCADA was secure because it wasn't connecting to the internet. However, since more and more SCADA systems are adopting its Modbus protocol over TCP and are accessible via the Internet, SCADA are exposed to all of the cyberattacks that threats TCP/IP protocols. In the mean while, security means were barely implemented. The gaps between the security requirement and reality leads to dangerous SCADA breaches. In 2010, Stuxnet [4] was spread over the world and damaged Iranian nuclear power plants. Since then, the need for industrial network security became urgent.

To safeguard SCADA networks, an intrusion detection system (IDS) needs to be implemented. IDS can be classified as two main categories: signature-based and heuristic (anomaly)-based. One of the most famous signature-based IDS is Snort. The signature-based IDS detects malicious activities by matching specific patterns. The patterns are also called signatures. Security researchers analyze the malware and threats to find the signatures. One primary way to analyze the threat is reverse engineering. Reverse engineering is a process by which a man-made object is deconstructed to reveal its designs, architecture, or to extract knowledge from the object. In the security fields, the object is normally malicious software. By diving into the source code of the malware the researcher learns how the malware injects, propa-

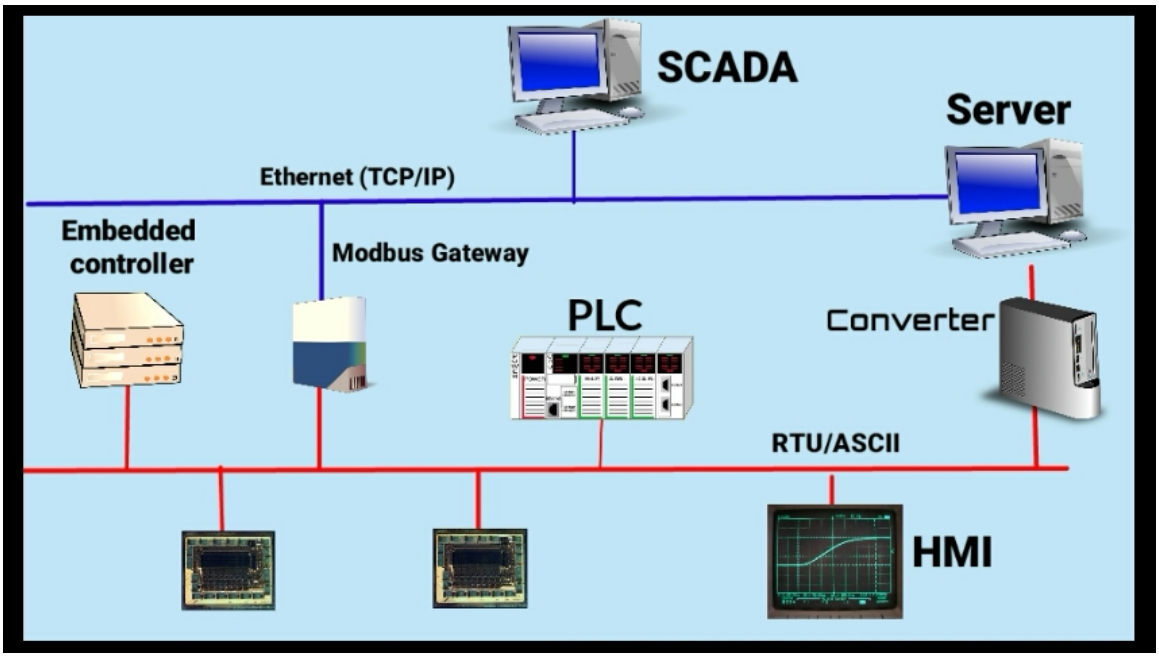


Figure 1.1: SCADA [2]

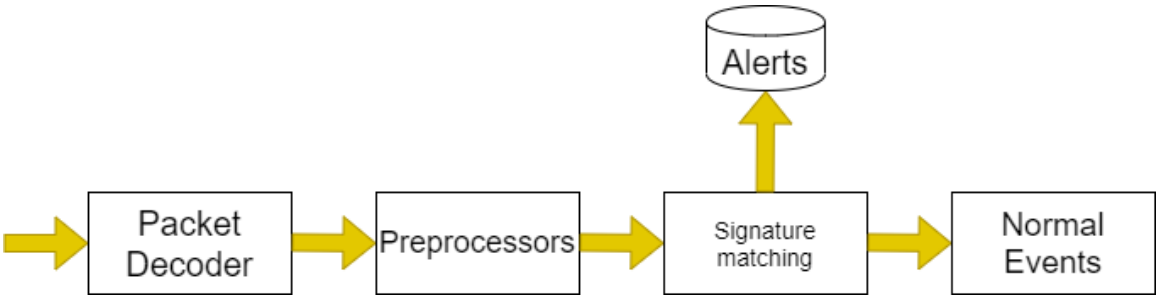


Figure 1.2: Signature-based IDS workflow

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"Win32.Downloader.Ursu - Downloading Malicious File"; flow:established,from_client;
dsize:<60; content:"GET"; http_method; content:"/download/updater.exe"; fast_pattern; http_uri; content:!"Accept|3a|"; http_header; content:!"Referer|3a|";
http_header; content:!"User-Agent|3a|"; http_header; content:!"Connection|3a|"; http_header; pcre:"/GET\x20\/download\/updater\.exe\x20HTTP\/1\.1
\x0d\x0aHost\x3a\x20/"; reference:nokia,MSIL; classtype:Downloader; sid:519112050; rev:1; )
```

Figure 1.3: Signature example

gates exploits the victim hosts. Another common method to find signature is more straightforward. The researchers run the malware in an isolated sandbox which is normally a virtual machine to generate malicious network traffics. By distinguishing the malicious patterns from the normal traffics, the researcher find the signatures of the malware. Both of these methods, however, requires the expert domain knowledge of security and reverse engineering. Another drawback is the signature-based IDS cannot detect unknown threats. The process of signature-based IDS is shown in Fig. 1.2. An example of Snort signature is shown in Fig. 1.3. In this example, the signature detects the malware by matching the HTTP headers of each single out-going HTTP traffic packet. The first thing that the signature matches is the packet size as "dsize". In this case the signature only looks at the traffic whose size is less than 60 bytes. This is an effective way to prevent false positive. Then the signature would look at the the HTTP method headers. A packets whose HTTP header is GET and URI is "download/update.exe" without HTTP referer would be labeled as malicious traffic. One drawback of this may lead to false positive as the URI is pretty common. Even if the signature restricted the size of the traffic, it is still possible being triggered by normal traffic. Generally, the malware would do post exploitation such as sending system information to command and control (CC) servers after landing the victim's computer. If the IDS is capable to detect correlation between landing and post exploitation instead of only looking at each single packet, the IDS may decrease the false positive rate. An anomaly-based IDS [20] overcomes these challenges by introducing machine learning to identify attack patterns from data. Machine learning

can be classified as supervised learning and unsupervised learning.

Supervised learning algorithm relies on the ground truth label of the data. The function of supervised learning algorithm is mapping from the data instances to the labels. In this process, the label would involve and guide the supervised learning algorithm to build the mapping. Supervised learning algorithm normally address two types of problem: Regression and classification. Regression problem is mapping a data instance to a target value. Classification is mapping the data instance to a target class.

Unsupervised learning does not require the involvement of labels. Instead, unsupervised learning is seeking the insight of data instances. So the problem for an unsupervised learning algorithm to solve is clustering. Clustering is a task that groups a set of data instances which are similar or in a same group. A typical example of unsupervised clustering algorithm is Kmeans clustering. Unsupervised learning is widely used in anomalous detection in cyber security area. The reason is that it's hard to obtain the ground truth of the network traffics. However, a disadvantage of unsupervised learning algorithm is the high false positive rate as the unsupervised learning does not have the knowledge of ground truth.

Recurrent neural network (RNN) is a class of artificial neural network which has the capacity of exploring correlations of between nodes of the neural network. Long-short term memory (LSTM) is one of the typical RNN. LSTM cells (neurons) are designed to transfer information from the previous cells to the upcoming ones. This advantage makes LSTM perfectly suitable for detecting correlated attacks. Based on the experiments, however, LSTM is not the most ideal model because of the limitation of detecting uncorrelated attacks. To address this problem, we propose an ensemble learning model in the thesis.

Thus, the main purposes of this thesis are:

1. Research deep learning models that can be used to for temporal dependence.

2. Implement IDS that is able to embed deep learning models for online anomalous detection.

3. Compare the performance of various machine learning models

1.2 Research Problem

Internet of Thing (IoT) is one of the most promising industry fields due to the convenience it brings to people. With the highly developed 5G, embedded systems and machine learning, IoT could be extremly powerful and helpful [10] as shown in Fig. 1.4. IoT can be used on autonomous driving [11], health care [44], wearables and industrial automation. However, the security of IoT was overlooked. Losses can be caused by the exploitation of IoT security vulnerabilities. As a subset of IoT system, SCADA is also facing the same problem and urgently to be solved. Specifically, the micro-controllers in SCADA system are in a low computation capacity which makes it vulnerable to DoS attacks. Unfortunately, traditional signature based IDS does not have an effective way to address this problem. The common method of detecting DoS attack in signature-based IDS is setting threshold of the suspicious packets. However, this method will lead to a high false positive rate. Therefore, a noval method with capacity to automatically and accurately identify DoS attacks is required.

In this thesis the attacks are classified into two categories: correlated and uncorrelated. Correlated attacks is launched by sending multiple legitimate traffic and uncorrelated is performed by single packet. DoS attack can be seen as a correlated attack. In order to detect the patterns of a correlated attack, the algorithm should have the capacity to find the correlated patterns. Most of machine learning algorithms such as logistic regression and neural network can only find the patterns of each individual packet instead of a sequence.

1.3 Contributions

In this thesis three types of machine learning algorithms: Feed-forward neural network, LSTM network and ensemble learning are proposed to address the detection of both uncorrelated and correlated attacks including DoS attack and MITM attack in SCADA system using Modbus/TCP protocol. Two types of implementation of LSTM: Many-to-Many (MTM) and Many-to-One (MTO) are proposed and compared. An ensemble learning model combines the strength of the FNN and LSTM is implemented and obtain a perfect result. In order to test the performance of the machine learning algorithms, three labeled data sets that contain various attacks are generated. These data sets are created from the SCADA testbed developed by L. Zhang at [30]. These data can be used future analysis of machine learning algorithms.

Figure 1.4: Internet of Things

1.4 Outline of the thesis

The outline of this report is organized as below:

1. Chapter 1 illustrates the introduction of the thesis.
2. Chapter 2 discusses the SCADA security risk and the anomaly-based IDS
3. Chapter 3 discusses the SCADA system and the data synthesis method
4. Chapter 4 illustrates the implementation of Anomaly-based Framework
5. Chapter 5 is the machine learning algorithms embedded into the IDS framework
6. Chapter 6 is the evaluation and results of the model
7. Chapter 7 is the conclusion and future work

Chapter 2

SCADA security and anomaly-based detection

2.1 Deep learning overview

Deep learning is a subset of machine learning in artificial intelligence (AI). The deep learning also has two types: supervised and unsupervised. The representative of unsupervised deep learning algorithm is deep belief network (DBN) [16]. The supervised deep learning algorithms include RNN, convolutional neural network (CNN). RNN and CNN are widely used in applications of computer vision [5], machine translation [6], etc. Deep learning started to be popular since ImageNet Visual Recognition challenge in 2012 where the AlexNet in [5] implemented by a deep CNN achieved the best performance. Deep learning models normally contains more hidden layers and complicated structures which enables deep learning model to see tiny elements of the object such as the smaller pixels of an image. This capacity brought by deep learning models incredibly increases the ability of pattern recognition. Fig. 2.1 is an application of object detection and segmentation. The author of this paper propose a Regionl-CNN (R-CNN) method to perform real-time object detection.

Except for the application on computer visions using CNN, deep learning also covers the field of natural langauge processing (NLP) using RNN. Unlike detecting the object on the image, NLP problems require that the model to understand the context of the sentences. In cyber security area, the context the relationships between the network traffics. The details of RNN and its application on security fields would be illustrated in Chapter 2.4

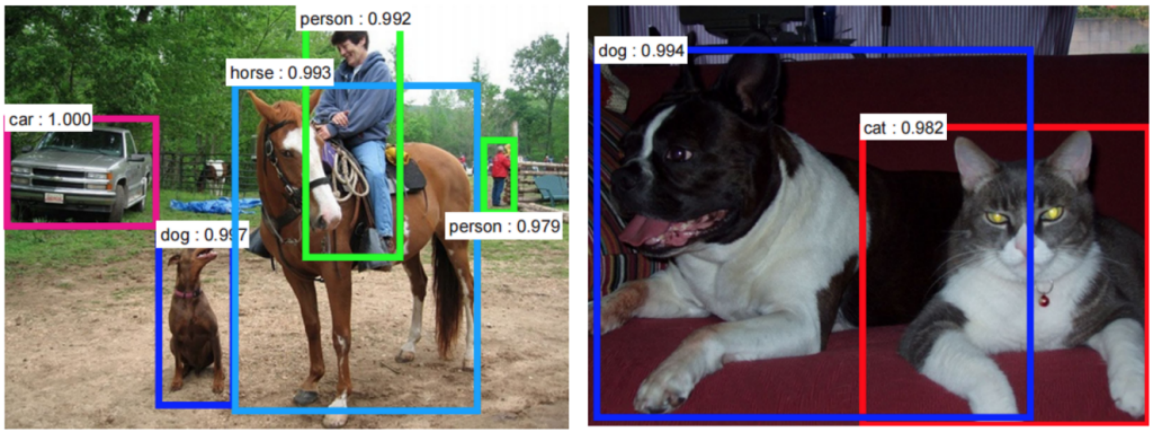


Figure 2.1: CNN in object detection [47]

2.2 Attack Classification

In order to test the performance of the machine learning models, attacks are required to generate malicious traffics. In this thesis we classify the attacks into two categories: uncorrelated and correlated.

2.2.1 Uncorrelated attack

Uncorrelated attacks can be reflected by single network packet. For example, a buffer overflow exploit sends a string whose size is much larger than normal to an application in order to inject malicious codes. The IDS can directly identify the packet containing exceptionally long input as malicious.

2.2.2 Correlated attack

On the contrary, the correlated attacks can only be recognized by looking at multiple packets. SYN flood attack is a typical example. SYN flood attacks exploit the vulnerability of TCP 3-way handshake shown in Fig. 2.2. Sender sends a SYN packet at the beginning of the TCP session. The receiver would response a SYN/ACK packet back to the sender in order to show that the receiver received the SYN and be able to talk. Once the sender receives the SYN/ACK from the sender, the sender would send a ACK and start to transfer information. This is TCP 3-way handshake process. If the sender only sends SYN but not response to the SYN/ACK sent by the receiver, the receiver would keep sending SYN/ACK till the receiver hears a ACK back or the timer is out. This would take too much resources to wait for the response. An attacker sending thousands of SYN without response would crash the receiver's server. Looking at each single packet, however, is not able to find this attack because the packet is either normal SYN or SYN/ACK packet. The SYN flood attack belongs to Denial of Service (DoS) attack. Another correlated attack is Man-in-the-middle (MITM) attack illustrated in Fig. 2.3. The hacker is the man in the middle of two communicating parties. The traffic sent by the senders would be delivered to the hacker first and the hacker would retransmit the traffic to the receivers in order to pretend that the two parties are communicating normally.

The traffic of MITM attack are shown in Fig. 2.4. The SCADA keeps exchanging information between PLC and Modbus Master. But when the system is under MITM attack the packets would retransmmit again and again.

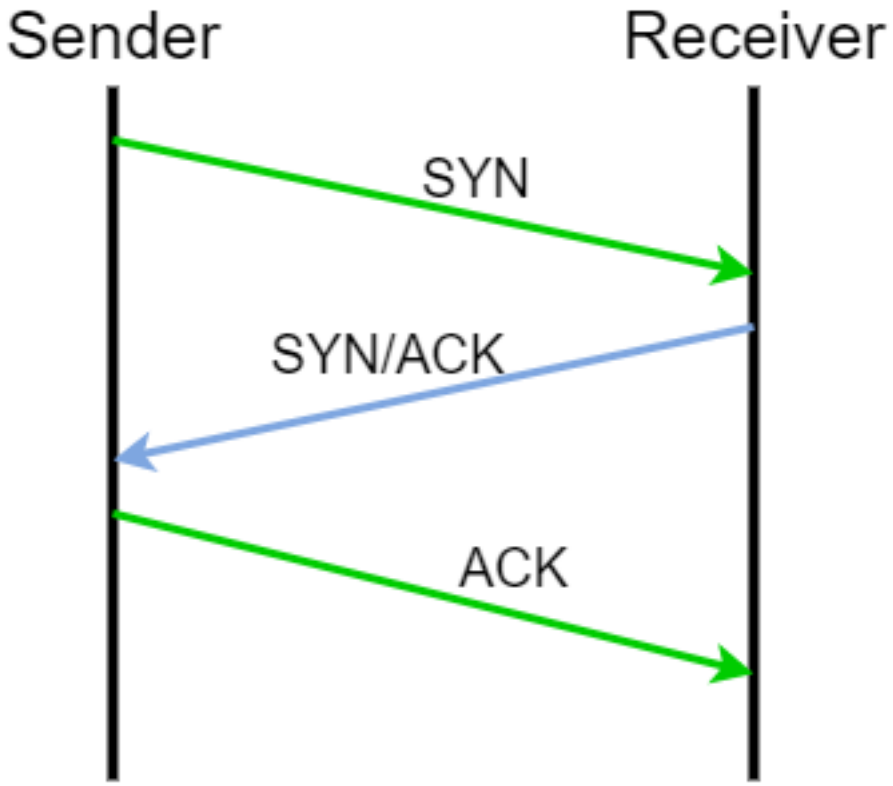


Figure 2.2: TCP 3-way handshake

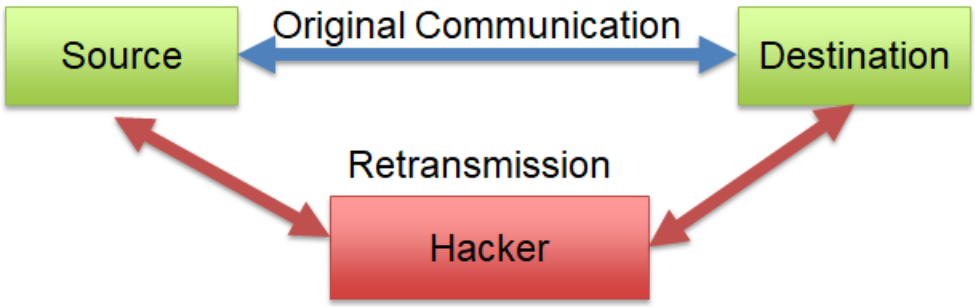
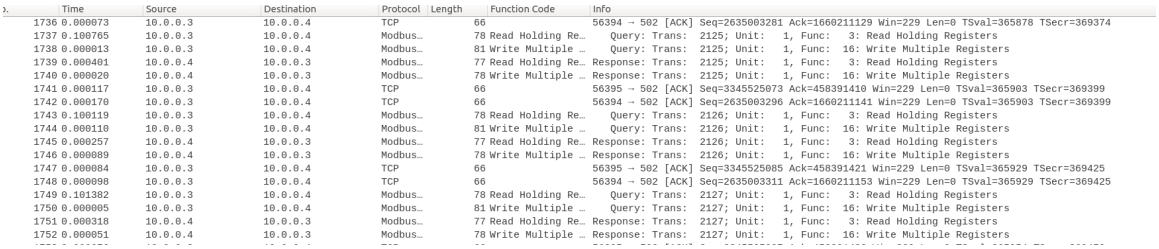
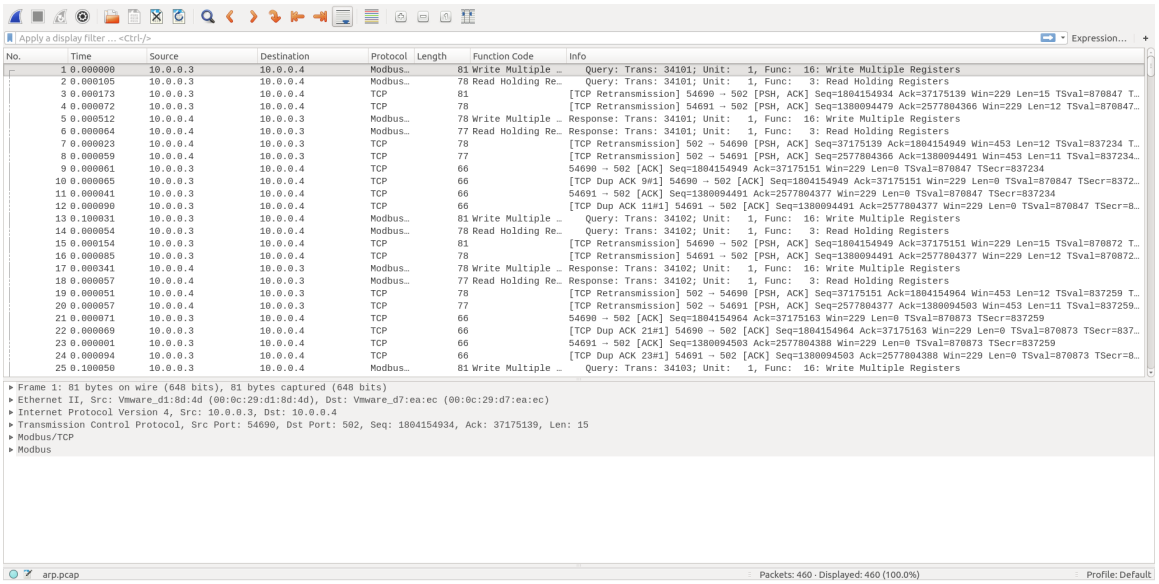


Figure 2.3: MITM attack



	Time	Source	Destination	Protocol	Length	Function Code	Info
1736	0.000073	10.0.0.3	10.0.0.4	TCP		66	56394 → 502 [ACK] Seq=2635003281 Ack=1660211129 Win=229 Len=0 TSval=365878 TSecr=369374
1737	0.100765	10.0.0.3	10.0.0.4	Modbus…		78 Read Holding Re…	Query: Trans: 2125; Unit: 1, Func: 3: Read Holding Registers
1738	0.000013	10.0.0.3	10.0.0.4	Modbus…		81 Write Multiple …	Query: Trans: 2125; Unit: 1, Func: 16: Write Multiple Registers
1739	0.000401	10.0.0.4	10.0.0.3	Modbus…		77 Read Holding Re…	Response: Trans: 2125; Unit: 1, Func: 3: Read Holding Registers
1740	0.000020	10.0.0.4	10.0.0.3	Modbus…		78 Write Multiple …	Response: Trans: 2125; Unit: 1, Func: 16: Write Multiple Registers
1741	0.000117	10.0.0.3	10.0.0.4	TCP		66	56395 → 502 [ACK] Seq=3345525073 Ack=458391410 Win=229 Len=0 TSval=365903 TSecr=369399
1742	0.000170	10.0.0.3	10.0.0.4	TCP		66	56394 → 502 [ACK] Seq=2635003296 Ack=1660211141 Win=229 Len=0 TSval=365903 TSecr=369399
1743	0.100119	10.0.0.3	10.0.0.4	Modbus…		78 Read Holding Re…	Query: Trans: 2126; Unit: 1, Func: 3: Read Holding Registers
1744	0.000110	10.0.0.3	10.0.0.4	Modbus…		81 Write Multiple …	Query: Trans: 2126; Unit: 1, Func: 16: Write Multiple Registers
1745	0.000257	10.0.0.4	10.0.0.3	Modbus…		77 Read Holding Re…	Response: Trans: 2126; Unit: 1, Func: 3: Read Holding Registers
1746	0.000089	10.0.0.4	10.0.0.3	Modbus…		78 Write Multiple …	Response: Trans: 2126; Unit: 1, Func: 16: Write Multiple Registers
1747	0.000084	10.0.0.3	10.0.0.4	TCP		66	56395 → 502 [ACK] Seq=3345525085 Ack=458391421 Win=229 Len=0 TSval=365929 TSecr=369425
1748	0.000098	10.0.0.3	10.0.0.4	TCP		66	56394 → 502 [ACK] Seq=2635003311 Ack=1660211153 Win=229 Len=0 TSval=365929 TSecr=369425
1749	0.101382	10.0.0.3	10.0.0.4	Modbus…		78 Read Holding Re…	Query: Trans: 2127; Unit: 1, Func: 3: Read Holding Registers
1750	0.000005	10.0.0.3	10.0.0.4	Modbus…		81 Write Multiple …	Query: Trans: 2127; Unit: 1, Func: 16: Write Multiple Registers
1751	0.000318	10.0.0.4	10.0.0.3	Modbus…		77 Read Holding Re…	Response: Trans: 2127; Unit: 1, Func: 3: Read Holding Registers
1752	0.000051	10.0.0.4	10.0.0.3	Modbus…		78 Write Multiple …	Response: Trans: 2127; Unit: 1, Func: 16: Write Multiple Registers

(a)



No.	Time	Source	Destination	Protocol	Length	Function Code	Info
1	0.000000	10.0.0.3	10.0.0.4	Modbus…		81 Write Multiple …	Query: Trans: 34101; Unit: 1, Func: 16: Write Multiple Registers
2	0.000105	10.0.0.3	10.0.0.4	Modbus…		78 Read Holding Re…	Query: Trans: 34101; Unit: 1, Func: 3: Read Holding Registers
3	0.000173	10.0.0.3	10.0.0.4	TCP		81	[TCP Retransmission] 54690 → 502 [PSH, ACK] Seq=1804154934 Ack=37175139 Win=229 Len=15 TSval=870847 T…
4	0.000072	10.0.0.3	10.0.0.4	TCP		78	[TCP Retransmission] 54691 → 502 [PSH, ACK] Seq=1380094479 Ack=2577804366 Win=229 Len=12 TSval=870847…
5	0.000512	10.0.0.4	10.0.0.3	Modbus…		78 Write Multiple …	Response: Trans: 34101; Unit: 1, Func: 16: Write Multiple Registers
6	0.000064	10.0.0.4	10.0.0.3	Modbus…		77 Read Holding Re…	Response: Trans: 34101; Unit: 1, Func: 3: Read Holding Registers
7	0.000023	10.0.0.4	10.0.0.3	TCP		78	[TCP Retransmission] 502 → 54690 [PSH, ACK] Seq=37175139 Ack=1804154949 Win=453 Len=12 TSval=837234 T…
8	0.000059	10.0.0.4	10.0.0.3	TCP		77	[TCP Retransmission] 502 → 54691 [PSH, ACK] Seq=2577804366 Ack=1380094491 Win=453 Len=11 TSval=837234…
9	0.000061	10.0.0.3	10.0.0.4	TCP		66	54690 → 502 [ACK] Seq=1804154949 Ack=37175151 Win=229 Len=0 TSval=870847 TSecr=837234
10	0.000065	10.0.0.3	10.0.0.4	TCP		66	[TCP Dup ACK 9#1] 54690 → 502 [ACK] Seq=1804154949 Ack=37175151 Win=229 Len=0 TSval=870847 TSecr=8372…
11	0.000041	10.0.0.3	10.0.0.4	TCP		66	54691 → 502 [ACK] Seq=1380094491 Ack=2577804377 Win=229 Len=0 TSval=870847 TSecr=837234
12	0.000090	10.0.0.3	10.0.0.4	TCP		66	[TCP Dup ACK 11#1] 54691 → 502 [ACK] Seq=1380094491 Ack=2577804377 Win=229 Len=0 TSval=870847 TSecr=8…
13	0.100031	10.0.0.3	10.0.0.4	Modbus…		81 Write Multiple …	Query: Trans: 34102; Unit: 1, Func: 16: Write Multiple Registers
14	0.000054	10.0.0.3	10.0.0.4	Modbus…		78 Read Holding Re…	Query: Trans: 34102; Unit: 1, Func: 3: Read Holding Registers
15	0.000154	10.0.0.3	10.0.0.4	TCP		81	[TCP Retransmission] 54690 → 502 [PSH, ACK] Seq=1804154949 Ack=37175151 Win=229 Len=15 TSval=870872 T…
16	0.000085	10.0.0.3	10.0.0.4	TCP		78	[TCP Retransmission] 54691 → 502 [PSH, ACK] Seq=1380094491 Ack=2577804377 Win=229 Len=12 TSval=870872…
17	0.000341	10.0.0.4	10.0.0.3	Modbus…		78 Write Multiple …	Response: Trans: 34102; Unit: 1, Func: 16: Write Multiple Registers
18	0.000057	10.0.0.4	10.0.0.3	Modbus…		77 Read Holding Re…	Response: Trans: 34102; Unit: 1, Func: 3: Read Holding Registers
19	0.000051	10.0.0.4	10.0.0.3	TCP		78	[TCP Retransmission] 502 → 54690 [PSH, ACK] Seq=37175151 Ack=1804154964 Win=453 Len=12 TSval=837259 T…
20	0.000057	10.0.0.4	10.0.0.3	TCP		77	[TCP Retransmission] 502 → 54691 [PSH, ACK] Seq=2577804377 Ack=1380094503 Win=453 Len=11 TSval=837259…
21	0.000071	10.0.0.3	10.0.0.4	TCP		66	54690 → 502 [ACK] Seq=1804154964 Ack=37175163 Win=229 Len=0 TSval=870873 TSecr=837259
22	0.000069	10.0.0.3	10.0.0.4	TCP		66	[TCP Dup ACK 21#1] 54690 → 502 [ACK] Seq=1804154964 Ack=37175163 Win=229 Len=0 TSval=870873 TSecr=837…
23	0.000001	10.0.0.3	10.0.0.4	TCP		66	54691 → 502 [ACK] Seq=1380094503 Ack=2577804388 Win=229 Len=0 TSval=870873 TSecr=837259
24	0.000094	10.0.0.3	10.0.0.4	TCP		66	[TCP Dup ACK 23#1] 54691 → 502 [ACK] Seq=1380094503 Ack=2577804388 Win=229 Len=0 TSval=870873 TSecr=8…
25	0.100050	10.0.0.3	10.0.0.4	Modbus…		81 Write Multiple …	Query: Trans: 34103; Unit: 1, Func: 16: Write Multiple Registers

(b)

Figure 2.4: (a) Normal Traffic (b) MITM traffic

2.3 SCADA exploits and Intrusion detection

Traditionally, signature-based IDS is the mainstream to detect SCADA attacks. It identifies specific patterns from traffic data to detect the malicious activities and can be implemented as policy rules in IDS software such as Snort [12, 13]. [17] investigates a set of attacks against Modbus and designs rules to detect attacks. [18] proposes a state-relation-based IDS (SRID) to increase the accuracy and decrease the false negative rate in denial-of-service (DoS) detection. However, these detection methods are sophisticate and only valid for specific scenarios. Overall, as discovered in the previous research, signature based IDS is only efficient at finding known attacks and its performance relies heavily on the experts' knowledge and experiences.

An anomaly-based IDS [20] overcomes these challenges by introducing machine learning to identify attack patterns from data. It is also widely used in other applica-

tions such as mobile data misuse detection [21], software [22] and wireless sensor security [23]. Several machine learning algorithms are proposed to develop anomaly-based IDS. Linda *et al.* [25] tailored a neural network model with error-back propagation and Levenberg-Marquardt learning rules in their IDS. Rrushi and Kang [26] combined logistic regression and maximum likelihood estimation to detect anomalies in process control networks. Poojitha *et. al.* [27] trained a feedforward neural network (FNN) to classify intrusions on the KDD99 dataset and the industrial control system dataset. Zhang *et. al.* [28] used support vector machine and artificial immune system to identify malicious network traffic in the smart grid. Maglaras and Jiang [29] developed a one-class support vector machine module to train network traces off-line and detect intrusions on-line. All these machine learning algorithms are excellent in observing the pattern of attacks from the in-packet features. None of them, however, takes into account of the temporal features between packets and thus will not perform well on attacks such as DoS which has strong temporal dependence.

DoS attacks are among the most popular attacks to slow down or even crush the SCADA networks. Most of the devices in SCADA operate in low power mode with limited capacity and are vulnerable to DoS [30]. Up to date, various DoS types, including spoofing [32], flooding and smurfing [33], etc., have been reported. Among all types of DoS, flooding DoS is widely-exploited where hackers send a massive number of packets to jam the target network. In [34], the author exploits TCP syn flooding attack against the vulnerability of TCP transmission using hping DoS attack tool. Flooding DoS, along with all other DoS, is difficult to detect because the in-packet features extracted from each data packet may not display any suspicious pattern [35].

Similar to DoS, man-in-the-middle (MITM) is another attack that is hard to detect from observing the in-packet features. It will be more efficient to detect them by observing the inter-packet patterns in time domain.

One attack script is implemented by Linux shell script to generate both attacks and normal traffic randomly. The imbalance of the attack and normal traffic are approximately 60%-40% in order to avoid bias on the offline training of the machine learning models. The attacks generated from the scripts consist of both correlated and uncorrelated attacks.

Anomaly-based IDS on DoS and MITM becomes popular along with the advances of machine learning. For example, in [36], an auto-associative kernel regression (AAKR) coupled with the statistical probability ratio test (SPRT) is implemented to

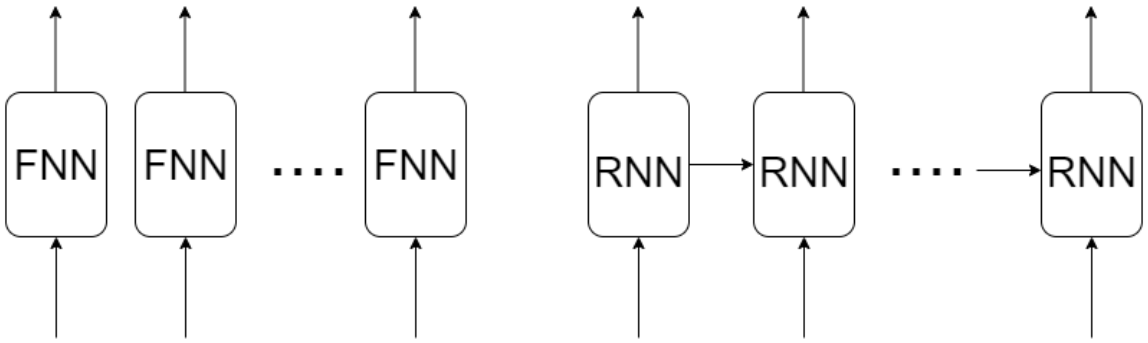


Figure 2.5: Difference between RNN and FNN

detect DoS. The result is not satisfactory because the regression model does not take the temporal signatures of DoS into consideration. In [37], FNN is used to classify abnormal packets in SCADA with 85% accuray for MITM-based random response injection and 90% accuracy for DoS-based random response injection attacks but 12% at Replay-based attacks. The author exploits various attacks including DoS and MITM in the testbed built in Modbus/RTU instead of Modbus/TCP. In [39], the authors propose one-class support vector machine (OCSVM) combined with k-means clustering method to detect the DoS. They set flags on every 10 packets to reflect the relationships of time series. But the handcrafted features can be easily by-passed by expert attackers. Besides supervised learning, unsupervised algorithms such as auto-encoder are also widely used for its comparative advantages on unknown attacks detection [38]. Here, one of such algorithms [14] is implemented for comparison.

2.4 Recurrent neural network

Recurrent neural network (RNN) is a class of deep neural network that connects nodes along a temporally correlated sequence. The difference between neural network and RNN is that RNN has the capacity to find the correlations in a sequence. As shown in Fig. 2.5. Each square represents a neuron of neural network. The left side are the neurons for FNN and the right side are the neurons for RNN. The input of the neurons are the data samples and the output of the neurons are the predicted classes. The FNN neurons are independent to others while the RNN neurons are correlated to others. In this figure this is a directional RNN. Also there is bi-directional RNN [8] whose neurons can deliver information to both previous and next neurons.

RNN is a widely-used sequence model. In [7] the author conbines RNN and

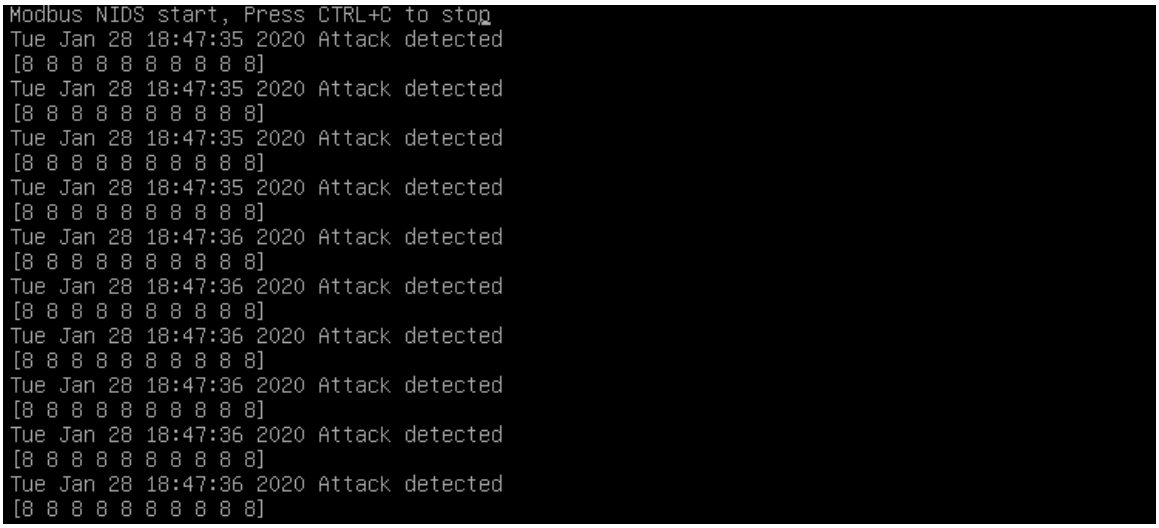


Figure 2.6: IDS Alert

multi-task learning to do text classification. RNN and CNN are combined in [9] to classify the handwrite dights in MNIST dataset. The typical RNN are LSTM [49] and GRU [50]. RNN is also widely adopted in security area. In [45, 46] the author use LSTM to detect distributed denial of service (DDoS) attacks. The reason of choosing LSTM is because DDoS attack is a correlated attack. In [43], the author uses bidirectional LSTM autoencoder to perform anomaly detection. However, RNN on SCADA especially on Modbus/TCP still need to be explored.

2.5 IDS

IDS is shown in Fig. 2.6. IDS generates alerts for MITM attack whose index is 8. The alert contains the timestamp of detection and the types of attacks for every 10 consecutive packets. Not only shown in the command interface, the IDS would also log all of these information to a csv file. The workflow of IDS is shown in Fig. 2.7. The PyShark plays a role of traffic decoder. The decoded packet would be processed in order to fit the requirement of the input of the machine learning models. The machine learning models are the detection engines which would automatically generate alerts for the anomalous behaviors. All of the packets would be eventually saved as log files. The source code of the IDS can be found at appendix.

To verify the performance of RNN on the anomalous detection, an IDS embedding the trained deep learning models is required. But no open source IDS combined with

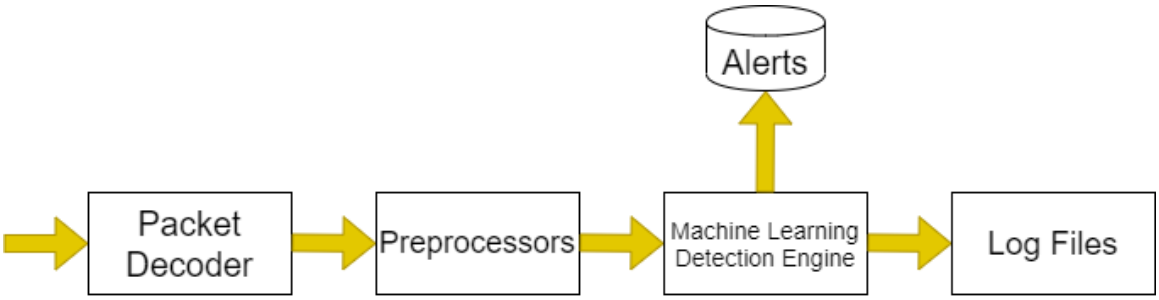


Figure 2.7: IDS work flow

deep learning models is available. Therefore, a prototype IDS is implemented. The IDS should have the following features:

1. An inline sensor that collects the network traffics.
2. A parser that extracts features from Modbus/TCP traffics.
3. A processor that is able to raise alerts when detecting anomalous behaviour.
4. Be able to embed deep learning models as engines for anomalous detection.
5. Log all of the information above to CSV files for future training and verification.
6. The IDS should be easy to deploy on the testbed. Also it is low cost when it's operating on a personal computer.
7. The IDS should be easy to update or replace machine learning models.

Chapter 3

SCADA Testbed and Data Synthesize

3.1 Testbed introduction

Our IDS is tested on a simulated SCADA testbed. A simulated network has the advantage of being easy to maintain, change and operate and is less costly than a real device network. A software testbed, which simulates a SCADA industry network and emulates the attacks was built by L. Zhang [30] on the basis work of T. Morris [52]. In the past, several preliminary researches on SCADA security had been conducted on this testbed [53, 54]. The attack target is a simple SCADA network, consisting of two tanks using Modbus over TCP. The liquid level of the tanks is controlled by pumps and measured by sensors via Modbus control information. The purpose of this network is to attract hackers and study possible defense methods. Such a system is called Honeypot, as it fools the attacker while exploiting his behaviour. This tank system is developed by the MBLogic HMIBuilder and HMIServer toolkit [55] and has been extended by L. Zhang in [30]. The HMI's purpose is to pull data from the sensor or send the desired pump speed to the motor periodically. The back end of the HMI is a PLC while the front end is a web browser.

As this system is simulated, we make use of four virtual machines as shown in Fig. 3.1. The SCADA system runs on a Modbus master and several slaves. On a virtual host called Nova the HMI is deployed, thus we refer to this host as Modbus master. In order to extend the network, some Modbus slaves such as PLCs are simulated by the HoneyD software [56]. This will provide a more realistic Honeypot. The role of a Modbus slave is to process commands from the master by pulling sensory data about the tank system from the PLCs and sending it back to the master.

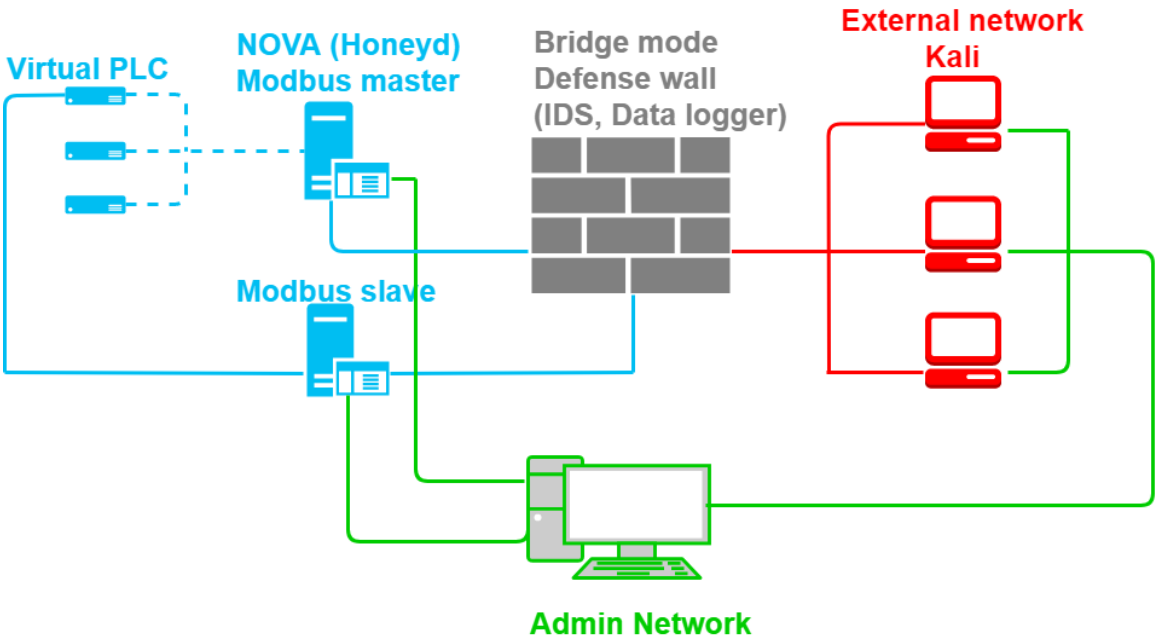


Figure 3.1: Testbed architecture [30]

The data needed to feed the neural network is generated by an attack machine using a virtual host named Kali. Kali is a Debian-derived Linux host used for penetration testing and features many attack and defense tools. Additional to the message exchange between the Modbus master (Nova) and its slaves we can launch normal traffic mixed with various attacks from Kali. A command line tool, Modpoll [57], is used to send Modbus instructions to the PLC which controls sensible tank system variables. An example Modpoll instruction which sends a pump speed of 5 to the system looks like this:

```
$ modpoll -0 -r 32210 10.0.0.5 5
```

The command addresses a simulated PLC with an IP address of 10.0.0.5 and a register address which contains either a threshold value (register 42212 - 42215), the current pump speed (32210) or the tank level (42210,42211), measured by the sensors. The threshold has 4 types: HH, H, LL, L. If the water level is higer than HH or lower than LL, the SCADA would generate an alert. If the water level is higher than H or lower than L, the SCADA would generate a warning. the Modpoll will send Modbus requests with function code 16 to attempt a write action to the specified registers. By modifying the pump speed the attackers can exceed the allowed tank level and create serious damage to a system. A script on Kali will randomly choose between

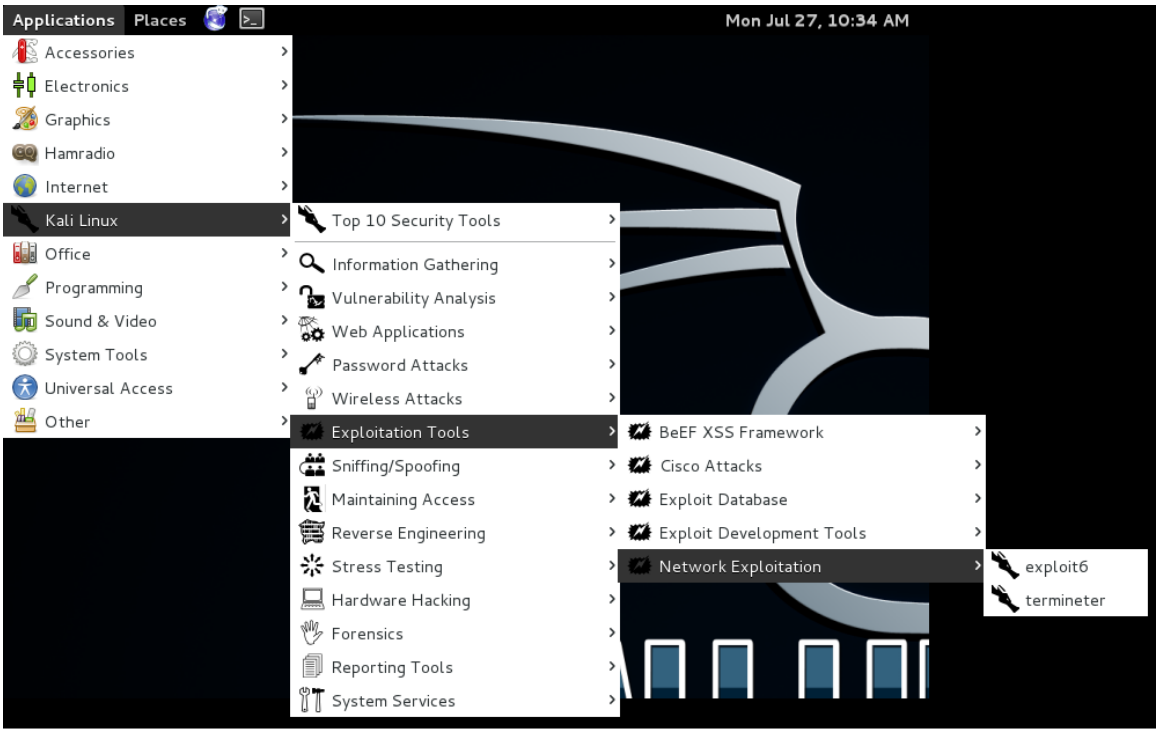


Figure 3.2: Kali

these normal or malicious Modbus instructions and will launch a Modpoll instruction with another randomly chosen parameter. This will ensure desired distribution of attack/non-attack data.

The traffic will be recorded by the fourth virtual machine referred to as "Defense Wall", which operates in the bridge mode and thus is invisible to the attacker. With PyShark we capture the traffic between Nova and Modbus slaves and between the attacker machine Kali 3.2 and the PLCs. During this process we can label each packet as malicious or normal.

3.2 Features extracted from the data packets

In our testbed, we use a self-developed IDS installed on "Defense Wall" to extract 19 features from each data packet captured. They are listed below:

1. Source IP address;
2. Destination IP address;
3. Source port number;

4. Destination port number;
5. TCP sequence number;
6. Transaction identifier set by the client to uniquely identify each request;
7. Function code identify the Modbus function used;
8. Reference number of the specified register;
9. Modbus register data;
10. Modbus exception code;
11. Time stamp;
12. Relative time;
13. Highest threshold;
14. Lowest threshold;
15. High threshold;
16. Low threshold;
17. Pump speed;
18. Tank 1 water level;
19. Tank 2 water level.

Here, the "Relative time" represents the time in seconds for packets relative to the first packet in the same TCP session. To reduce the periodicity of this feature, we reset it to zero when "Relative time" reaches 3,000 seconds.

3.3 Feature Normalization

Feature normalization is an essential part of feature engineering. The range of feature values varies wildly which increases the difficulty to optimize the losses. For example, two samples x_1 and x_2 in a same class are fit into a neural network. The range of value of x_1 is (0,1000) and the range of value of x_2 is (0,1). As both of these two samples

shares the same weights of neural network, the output classes of the neural network might be totally different. However, the ground truth is that the two sample are in the same class. Then the loss would increase because the classifier made wrong prediction. Therefore, the value of data samples are expected in a small range. The function of feature normalization is rescaling the value of features. Gaussian distribution is an expected value distribution which is suitable for neural networks. Two commonly method are used for feature normalization: min-max and feature scaling.

3.3.1 Min-max normalization

Min-max normalization uses the minimum and maximum value of a feature to rescale the value to range [0,1]:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{3.1}$$

Where x' is the re-scaled feature from x, $min(x)$ is the minimum value and $max(x)$ is the maximum value in the feature.

The advantage of min-max normalization is that it is easy to understand and easy to calculate. Also the value of each feature can be ranged to [0,1] which is convenient to neural networks. The drawback of the min-max normalization is it cannot handle outliers well. if the difference between minimum value and maximum value is too large, the scaled value distribution might be either left skewed or right skewed.

3.3.2 Feature scaling

To overcome the drawback of min-max normalization, In our IDS, we adopt feature scaling of each feature x in the dataset according to

$$x' = \frac{x - \bar{x}}{\sigma_x} \tag{3.2}$$

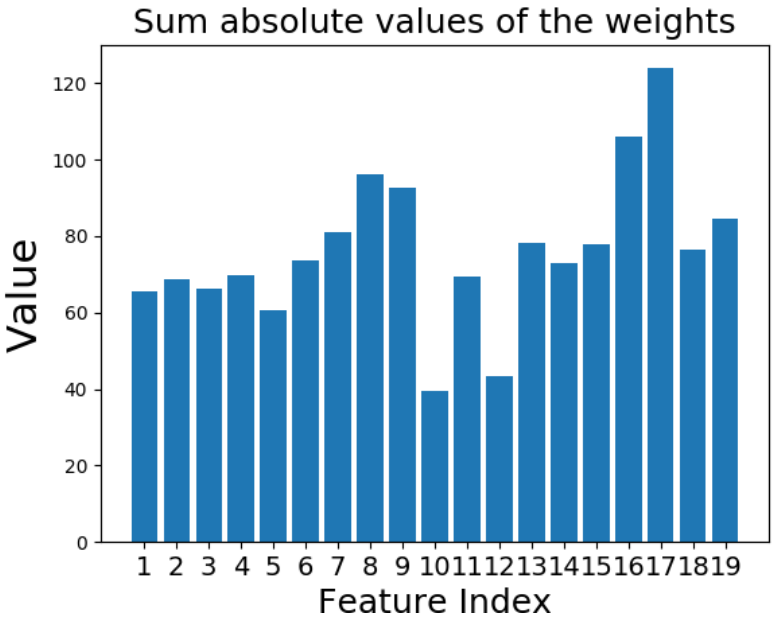
where $\bar{x}$ and σ_x are the mean and standard deviation of original feature x and x' is the re-scaled feature from x with zero mean and unity variance which is a normal distribution.

3.4 Feature Impact and analysis

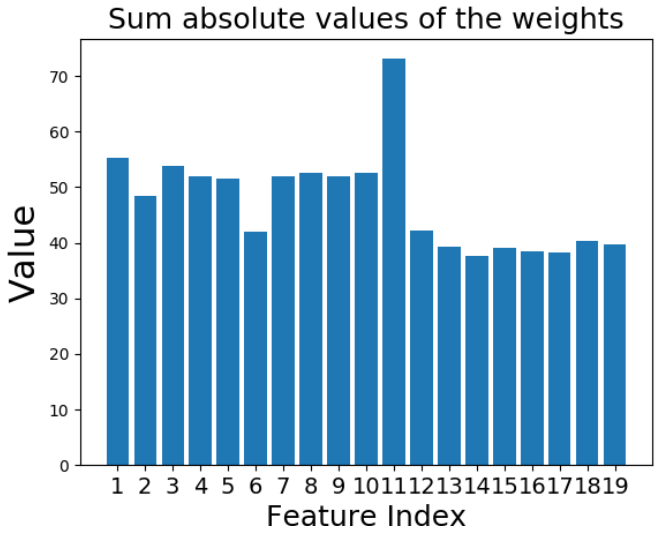
As an artificial dataset, the impact of the features should be illustrated concretely to avoid the existence of dominant features. The feature analysis, however, could be trivial and obfuscated since the structure of deep learning models are complicated. Normally, researchers could perform cross validation on the features. Concretely, one or more features are removed from the data during training and testing. If the performance of the machine learning model is not affected after removing the specific features, those features could be seen as unessential. On the contrary, if the performance of the model drops or increases sharply, the feature could be seen as important features.

Another method to evaluate the impact of the features is plotting the weights of input layer. The input layer connects to the data features directly. From the norm of the weights $|W|$ we have the summation of the weights. The greater the value of the weight is, the feature is more important.

Fig. 3.3 shows the difference of feature impact between correlated and uncorrelated attacks. In uncorrelated attacks, the feature "exception code (10)", "Relative time (12)", "sequence number (5)" related to the sequence information has a few contribution to the classification while the feature 8, 9, 16, 17 which related to the SCADA environment achieve high values. The situation is totally on the contrary of correlated attack where the "timestamp" (11) feature is the dominant feature. This is under expectation as the correlated attacks detection rely on the time information. Based on the above discussion, the weights of the feature can reflect the importance of the features. Fig. 3.4 is the weights of input layer of models trained by online testing dataset. Online testing dataset contains both correlated and uncorrelated attacks. From this figure we can see that the 12th feature: Relative time has the least value of the summation. The results in Table. 5.4 illustrates that the remove of feature "Relative Time" does not affect the performance of the models of online testing. Fig. 3.4b illustrates that the FNN weighted SCADA environment features (feature 13-19) instead of the time-related features (8-12). This can explain why FNN does not have the capacity of correlated attack detection. On the contrary, the difference between time related features and SCADA environment feature is not too much. This enables the ability of detecting correlations between packets.



(a)

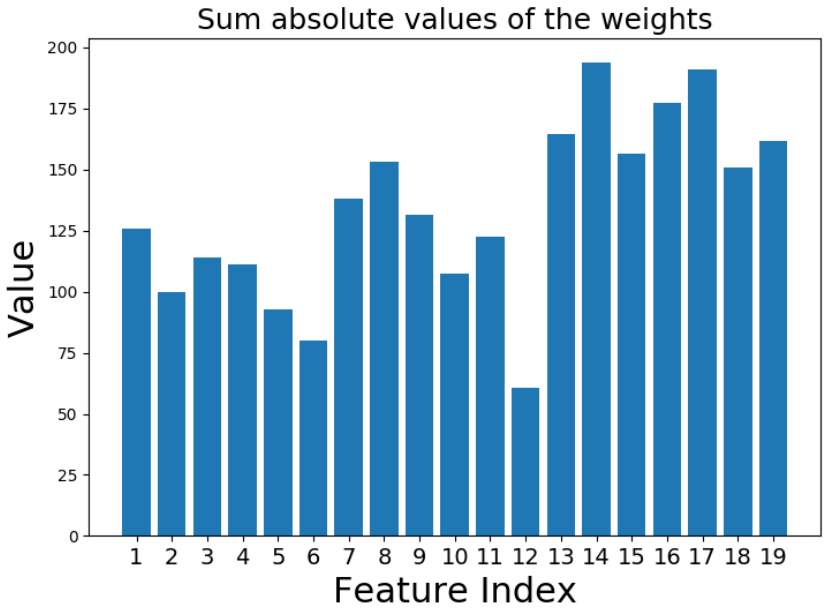


(b)

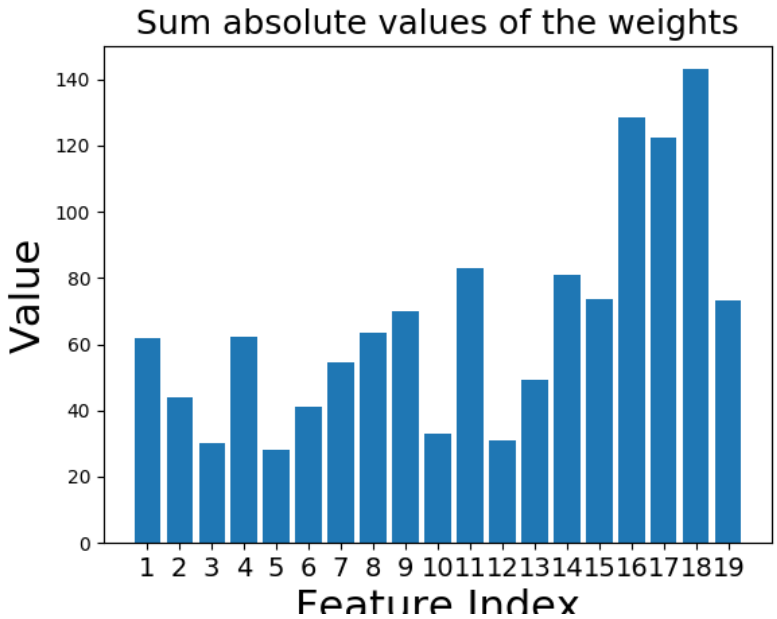
Figure 3.3: (a) Feature impact of LSTM on Dataset I (b) Feature impact of LSTM on Dataset II

3.5 Types of attacks in our datasets

As mentioned in the introduction, the attacks are classfied as two categories: correlated and uncorrelated. The correlated attacks include DoS and MITM attack. The



(a)



(b)

Figure 3.4: (a) Impact of features in LSTM in online testing (b) Impact of features in FNN in online testing

uncorrelated attacks are command injection attacks. The command injection attack is exploited by sending malicious Modbus/TCP command to the PLC. One example

Table 3.1: Macro-average comparison of feature

	Precision	Recall	F_1
All	99.54±0.03	99.01±0.07	99.27±0.05
No "relative time"	99.38±0.04	99.34±0.3	99.36±0.04

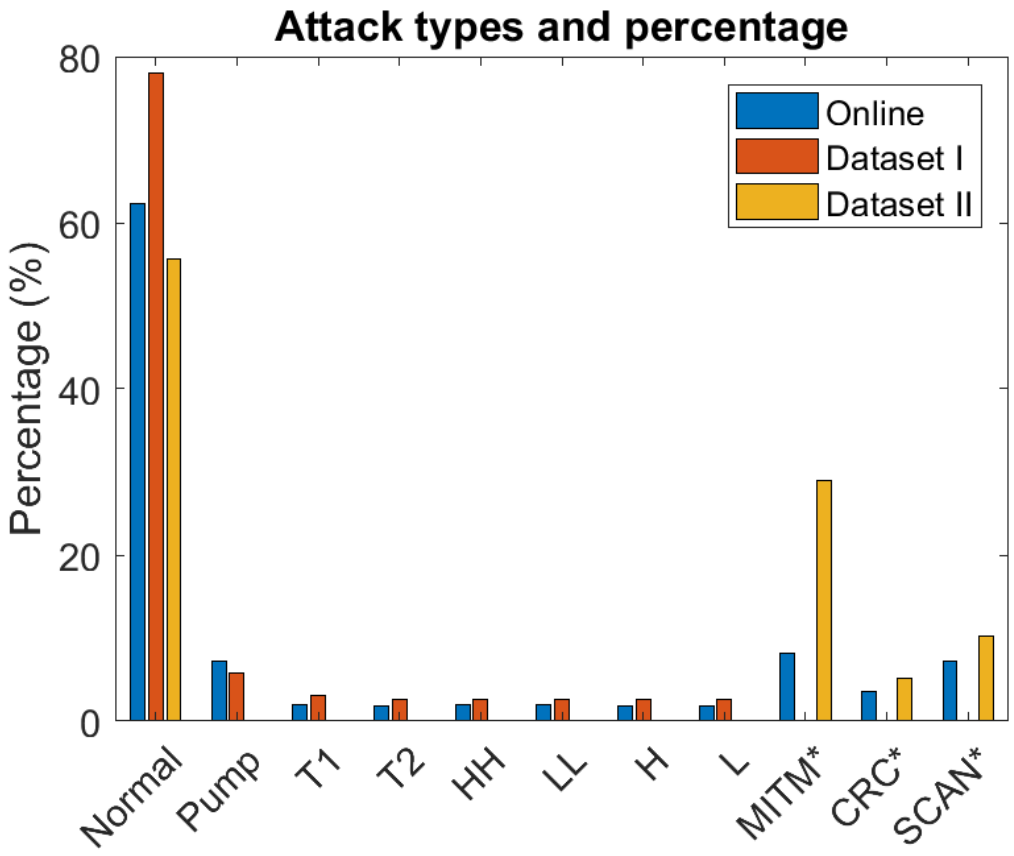


Figure 3.5: Data packet types distribution in Dataset I, II and online script. The ones with a superscript "*" are temporally correlated attacks.

Table 3.2: Percentage of normal, temporal uncorrelated and correlated attack traffics in Datasets and the online testing script.

	Dataset I	Dataset II	Online script
Total packets	298, 728	201, 311	1, 088, 448
Normal (%)	78	56	62
Uncorrelated (%)	22	0	19
Correlated (%)	0	44	19

is:

```
$ modpoll -0 -1 -r 42212 10.0.0.5 120
```

Attack Name	Description	Type
Pump	Inject invalid value to pump speed	Uncorrelated
T1	Inject invalid value to Tank 1 level	Uncorrelated
T2	Inject invalid value to Tank 2 level	Uncorrelated
HH	Inject invalid value to the highest threshold	Uncorrelated
LL	Inject invalid value to the lowest threshold	Uncorrelated
H	Inject invalid value to the high threshold	Uncorrelated
L	Inject invalid value to the low threshold	Uncorrelated
MITM	Man-in-the-middle attack	Correlated
CRC	DoS attack by sending multiple incorrect CRC packets	Correlated
SCAN	DoS attack by sending multiple scan requests	Correlated

Table 3.3: Attack types

The -r 42212 points to the register that storing the threshold value of tank level. This command change the threshold to 120 out of 100 which would lead the water tank overwhelmed without an alert. The details of the command injection attacks can be found at Table. 3.3 Using our scripts, we created two datasets. As illustrated in Fig. 3.5, in addition to "Normal" data packets, Dataset I contains attacks that are uncorrelated in time domain while Dataset II contains temporally dependent attacks. Here we have incorporated 10 attacks in our testbed. 7 of them are temporally uncorrelated while the remaining 3 are correlated. The temporally uncorrelated attacks include "Pump Speed" (Pump), "Tank 1 Level" (T1), "Tank 2 Level" (T2), "Threshold Highest" (HH), "Threshold Lowest" (LL), "Threshold High" (H) and "Threshold Low" (L).

Among all temporally correlated attacks, two types of flooding DoS attacks are included [51]. The first labelled as "Scan flooding" (SCAN) is to send massive scan command, resulting in increasing latency of communications between the HMI and the sensors in SCADA. The second type labelled as "Incorrect CRC" (CRC) is sending massive packets with incorrect cyclic redundancy check (CRC) to cause latency of master.

Another temporally correlated attack included in this testbed is MITM attack which is described in Sec. 2.2.2. It is an eavesdropping where the attacker monitors the communication traffics between two parties secretly. Here, the MITM attack is launched by Ettercap [58] using ARP spoofing [59]. The screenshot of Ettercap graphical interface is shown in Fig. 3.6. In the script, a command interface of Ettercap is used to generate ARPspoofing attack showns as Fig. 3.7One effective way to detect

ARP spoofing is identifying the Media Access Control (MAC) address in layer 2 of OSI model. However, most of Network IDSs (NIDS) do not support the protocols in layer 2 such as ARP and MAC protocols. Even Snort requires an ARP spoof preprocessor [60] to collect the MAC address information to detect ARP spoofing. Besides, the victim host of ARP spoofing attack would experience packets retransmissions. For SCADA networks, packet retrasmissions or delay may cause great damages. Therefore, the IDS should raise alert when it detects either MITM attack or packets retransmissions. To make the IDS robust in detecting both MITM and packets retransmissions we remove the MAC address feature which was used for labeling MITM attack from the datasets for training neural networks.

At the first stage, FNN and LSTM IDSs will be trained as binary classifiers that only predict attacks from normal traffic and tested on these datasets separately for performance comparisons. In on-line phases, these two IDSs along with our FNN-LSTM ensemble IDS will be trained as multi-class classifiers by the combined datasets to predict various types of attacks from normal traffics and implemented on the testbed. In addition, we also implement a script that can launch realtime attacks for online testing. The online script will randomly launch normal traffic, temporally uncorrelated and correlated attacks with ratios shown in the table to examine the omni-detection capability of different IDSs.

3.6 Label method

The label method of uncorrelated attack is straightforward. Theoretically, A signature-based IDS can perfectly capture all uncorrelated attacks. For example, IDS would raise an alert when it detects an operation trying the change the water level to 120. The correlated attacks, however, is not easy for signature-based IDS to capture. In order to have the ground truth of correlated attacks in the dataset, a tricky way is implemented by using flags. At the beginning and the end of an correlated attack, a packet with special reference number is sent to mark the position. The packets in between the marks with special IP address are labeled as correlated attacks. The Fig. 3.8a.

As shown, the reference number is 52210 not used by SCADA. After this packet, the packets with "Exception returned" which is the signal of CRC attack arrived. At the end of the attack, the packet with reference number 52211 shows up to represent that the DoS attack is end. By using this trick, the DoS attack packets are isolated

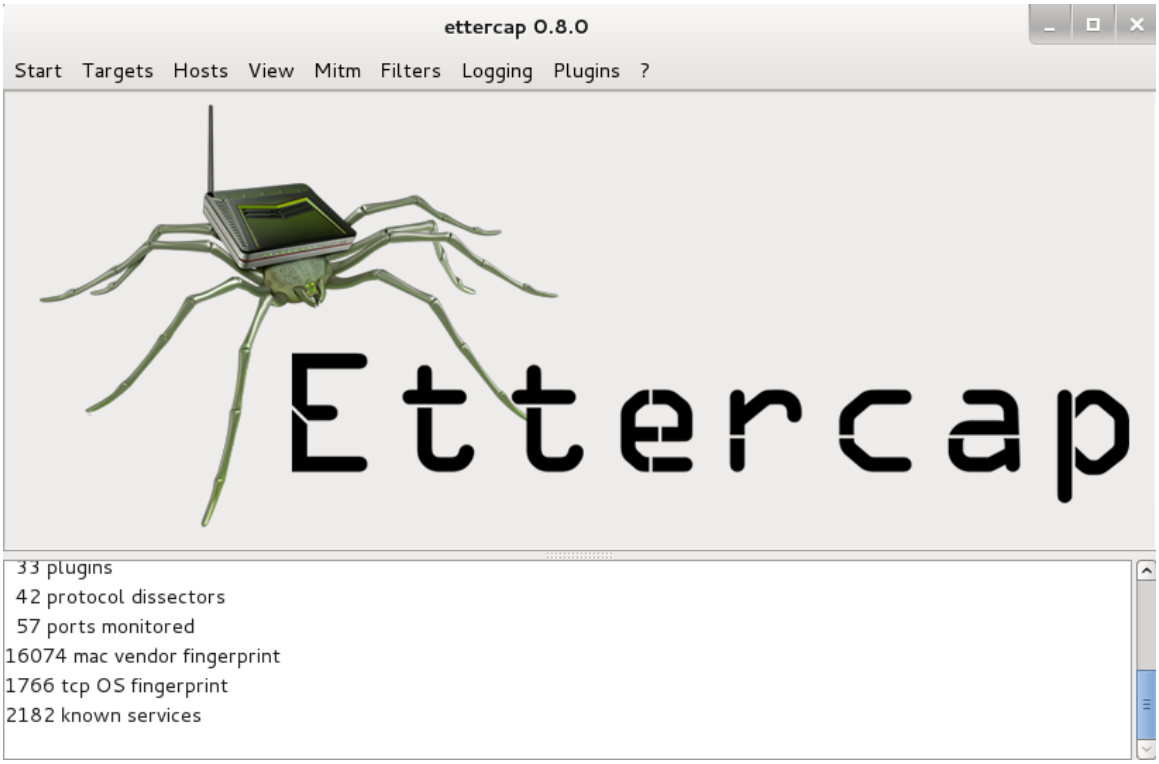


Figure 3.6: Ettercap

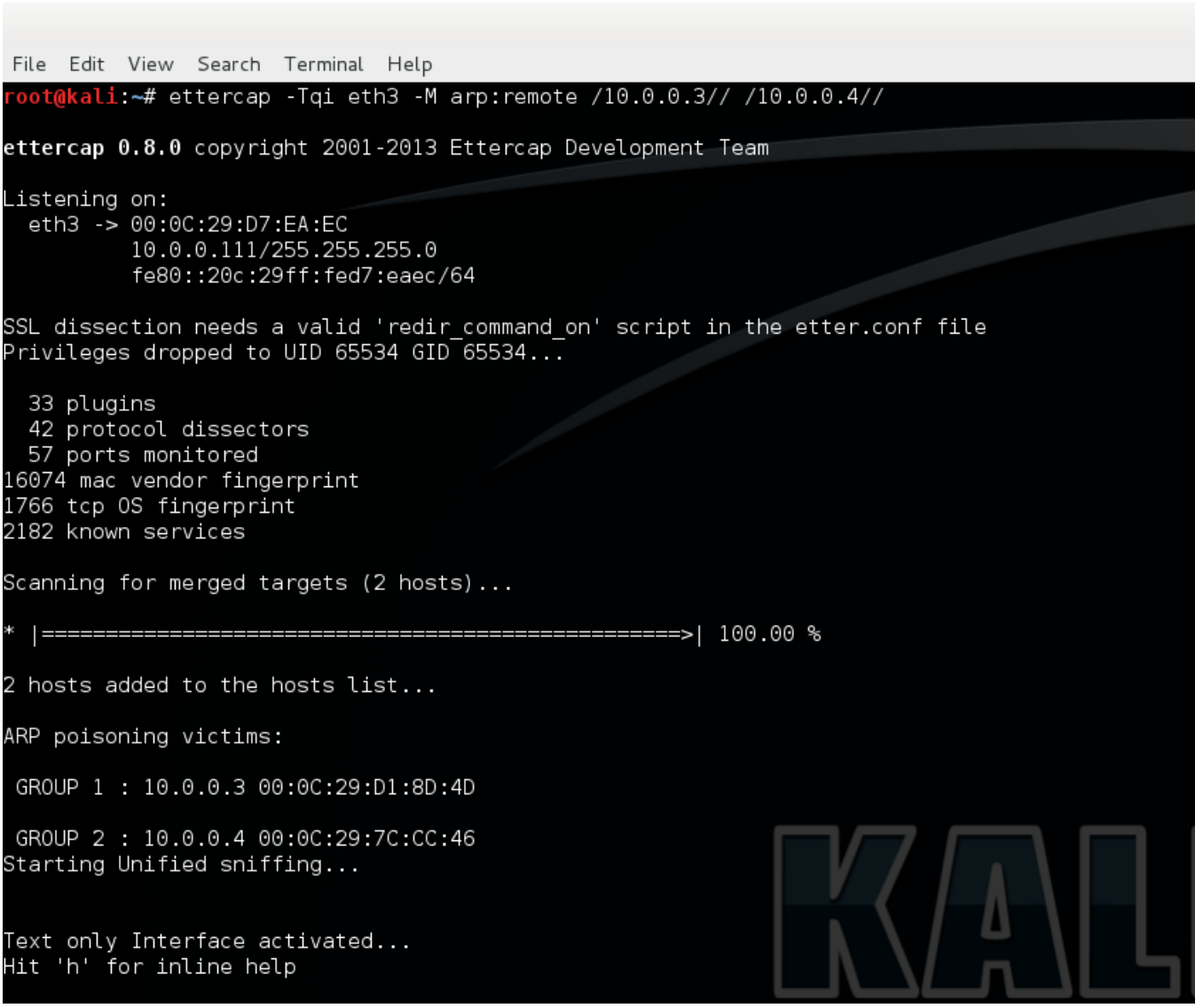
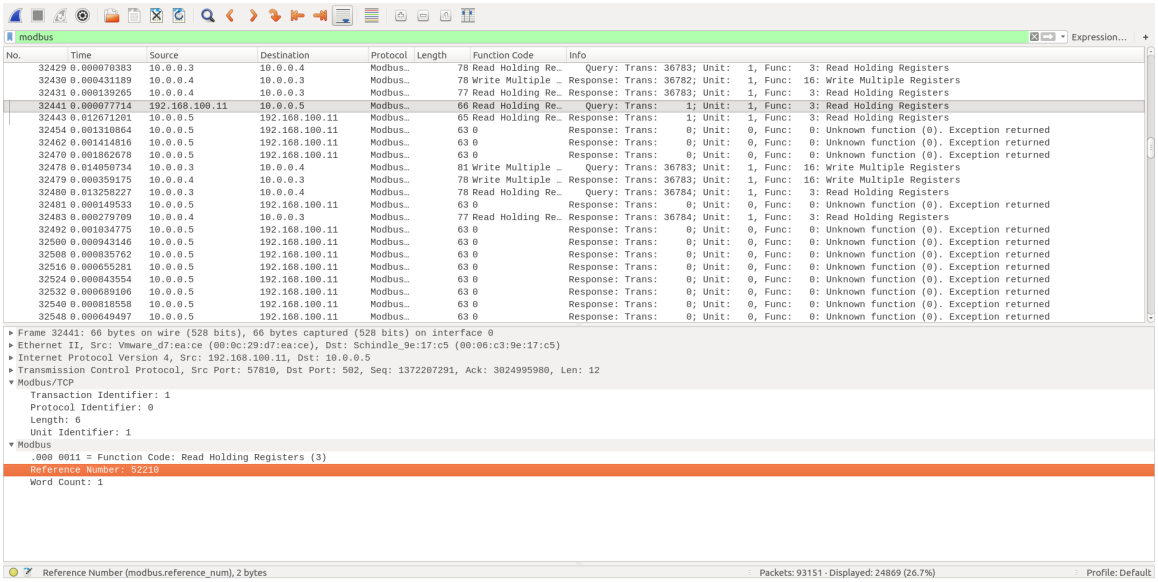
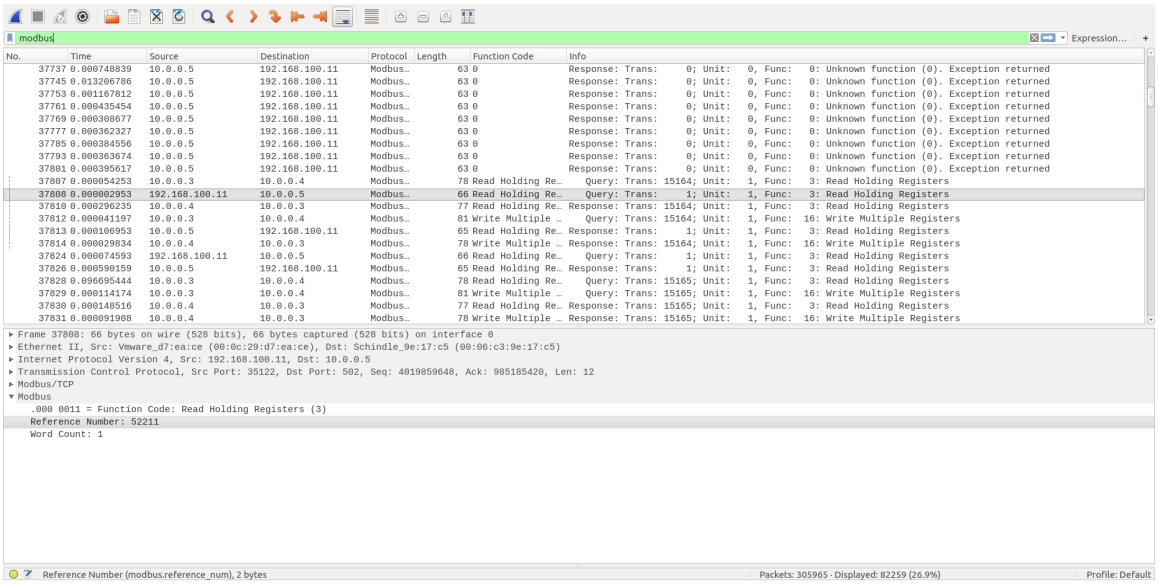


Figure 3.7: MITM Traffic



(a)



(b)

Figure 3.8: Flags of CRC Attack

in this range. Combining the special IP address which can be easily obtained, the DoS attack packets can be labeled precisely. After labeling the data, the flag packet would be removed from the dataset. Otherwise, the machine learning algorithms may track the flag packets to find the attacks instead of looking at the the patterns. In other word, the dataset does not contain any packet with the special packets.

Chapter 4

Feed-forward neural network

4.1 FNN neuron

Feed-forward neural network (FNN) is a classic artificial neural network model. The basic unit of FNN is the neurons. The function of neuron can be illustrated as Fig. 4.1. Given a data sample x_n with m features $\tilde{\mathbf{x}_n} = \{\tilde{\mathbf{x}_n^1}, \tilde{\mathbf{x}_n^2}, \tilde{\mathbf{x}_n^3}, ..., \tilde{\mathbf{x}_n^m}\}$, the neuron assigns each scalar element of the sample a weight $w_i, i \in (1, m)$. The sum of the weighted input is:

$$z = \sum_{i=1}^{m} w_i \tilde{\mathbf{x}}_n^i + b \tag{4.1}$$

But until this step the neuron can only do linear mapping. In reality, non-linear mapping is required. To fix this problem, the neural network introduces activation function.

4.2 Activation function

Activation function can perform non-linear transmission of the weighted sum input signals. By using the activation function the neural network can decide the activation of neurons. The commonly used activation function includes: Sigmoid, softmax, ReLu and tanh.

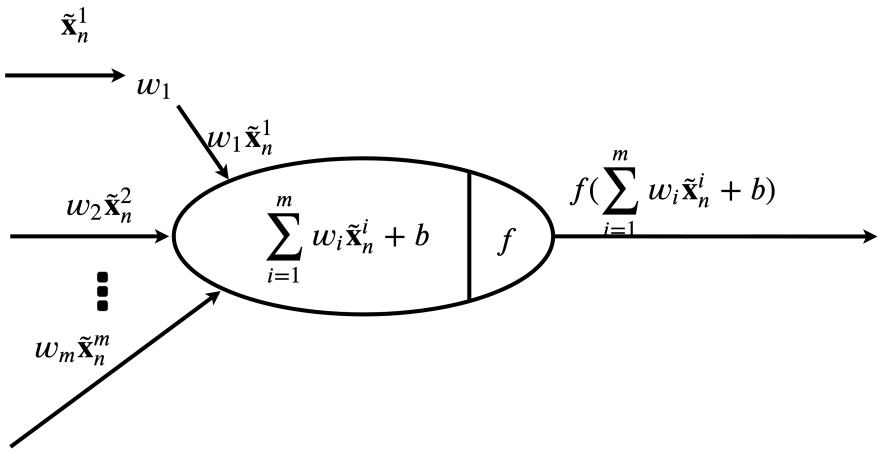


Figure 4.1: Details of each neuron in FNN

Sigmoid

The sigmoid function is widely used for binary classification. A sigmoid function is expressed as:

$$\sigma(z) = \frac{e^z}{1 + e^z} \tag{4.2}$$

where z is selected as the weighted sum of the input signal. The sigmoid function looks like a "S" curve. The range of output of sigmoid is in [0,1]. The greater the z is, the closer the $\sigma(z)$ is to one. On the contrary, if z is negative infinite, the $\sigma(z)$ is close to 0. Benefits to the output range of sigmoid function, the output can be seen as a probability. A threshold would be set (commonly 0.5) to decide whether to activate the neuron.

ReLu

The full name if ReLu is rectified linear unit. The expression of ReLu is:

$$f_h(x) = max\{0, x\} \tag{4.3}$$

ReLu function transfers negative value to zero and leave non-negative value as is. ReLu is computation efficiency and more close to linear function than sigmoid. Neural network training process may experience gradient vanishing especially in deep neural networks. The nonlinear activation function would make the deep neural network harder to perform back propagation. So a "linear" activation function as ReLu is commonly adopted by the hidden layers of deep learning structures.

Tanh

Hyperbolic tangent (Tanh) is also a commonly-used activation function. The expression of tanh is:

$$tanh(z) = \frac{sinh(z)}{cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{4.4}$$

The range of tanh(z) is (-1, 1) which makes it more suitable for some special activation such as gates activation of RNN neurons.

Softmax

Softmax function is used for multi-label classification problem. Given a vector $\mathbf{z}$ has K elements, softmax function calculates the possibilities of each element by:

$$\sigma(\mathbf{z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4.5}$$

Where i is the index of element of vector $\mathbf{z}$, σ is the softmax function. The benefit of softmax is that the sum of elements in $\sigma(\mathbf{z})$ is 1. Softmax is widely used in multi-label classification.

4.3 FNN structure

The basic structure of the FNN IDS is illustrated in Fig. 4.2. It consists an input layer, an output layer and one or more hidden layers in-between. Each layer has a number of neurons that use the neuron outputs from the previous layer as their input and produces output to the neurons in next layer. In our case, the final outputs are the predictions of attacks and normal events. Mathematically, the FNN is expressed as:

$$\begin{aligned}
\mathbf{z}^{<1>} &= \tilde{\mathbf{W}}^{<1>}\tilde{\mathbf{x}_n} + \mathbf{b}^{<1>}, \mathbf{h}_1 = \mathbf{f}_h(\mathbf{z}^{<1>}) \\
\mathbf{z}^{<2>} &= \tilde{\mathbf{W}}^{<2>}\mathbf{h}_1 + \mathbf{b}^{<2>}, \mathbf{h}_2 = \mathbf{f}_h(\mathbf{z}^{<2>}) \\
&... \\
\mathbf{z}^{<N+1>} &= \tilde{\mathbf{W}}^{<N+1>}\mathbf{h}_N + \mathbf{b}^{<N+1>}, \hat{\mathbf{y}} = \mathbf{z}^{<N+1>}
\end{aligned} \tag{4.6}$$

where N is the number of hidden layers, $\mathbf{x}_n$ is the feature vector of n-th single packet. $\mathbf{f}_h$ is the activation function.

The elements in the weighting matrices $\tilde{\mathbf{W}}^{<1>}, \tilde{\mathbf{W}}^{<2>}, ..., \tilde{\mathbf{W}}^{<N+1>}$, and bias vectors $\mathbf{b}^{<1>}, \mathbf{b}^{<2>}, ..., \mathbf{b}^{<N+1>}$ are the parameters to be trained. m is the number of features. $\{w_1, w_2, w_3, ..., w_m\}$ is the weight column vector for one FNN neuron in input layer. The output of the neural network is the output layer $\hat{\mathbf{y}}$ which is the predicted label. Our model structure is shown in Fig. 4.3. The FNN structure contains one input layer, one hidden layer with 100 neurons and one output layer. In the training process, $\hat{\mathbf{y}}$ would be compared with the ground truth label y in order to provide guidance for the next training steps. The measurement of the difference is called loss function.

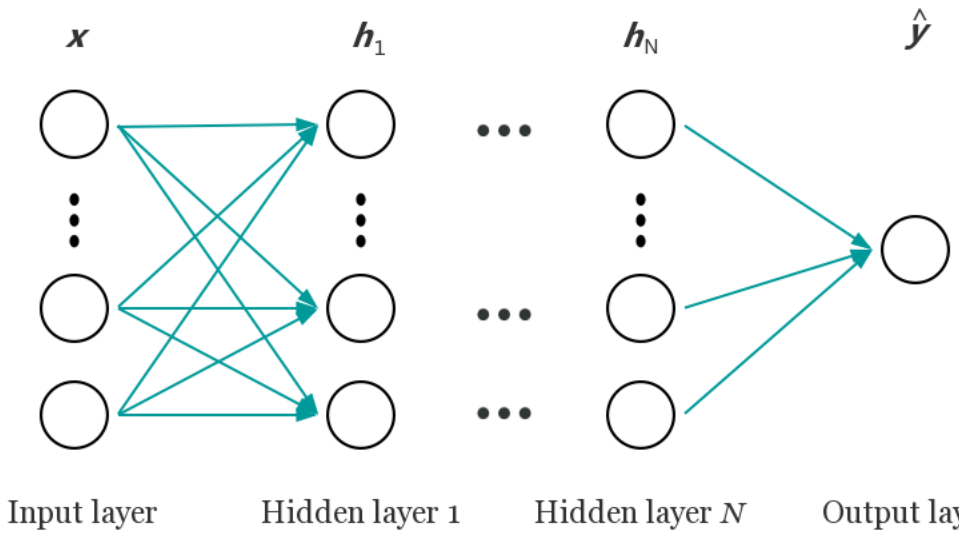


Figure 4.2: The schematics of the FNN IDS

4.4 Loss function

Loss function is an evaluation of difference between two label vectors. The commonly used loss function includes: mean square error (MSE), cross entropy, hinge loss etc. In this thesis MSE and cross entropy are used.

Mean Squared Error

Mean Squared Error (MSE) is L2 norm of the difference vector between the actual label and predicted label:

$$MSE = \frac{\sum_{i=1}^{n} \|y_i - \hat{y}_i\|_2}{n} \tag{4.7}$$

In auto encoder model, MSE is used to measure the performance of sample reconstruction.

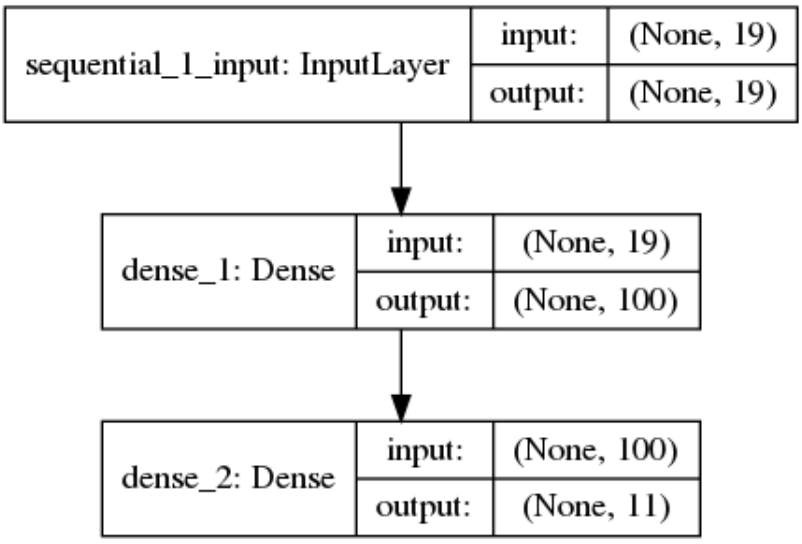


Figure 4.3: FNN archtecture of Keras Model

Cross entropy

Cross entropy derived in information theory to measure the difference between two probability distributions. In machine learning field, the cross entropy is used for classification problem. The expression of cross entropy is:

$$-y_i log(\hat{y_i}) - (1 - y_i) log(1 - \hat{y_i}) \tag{4.8}$$

Here we use softmax cross entropy as our loss function, which can be expressed as

$$f_L(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{p=1}^{C} \mathbf{y}_p \log(\mathbf{f}_s(\hat{\mathbf{y}_p})) \tag{4.9}$$

where $\hat{\mathbf{y}}$ is the predicted label and $\mathbf{y}$ the ground truth. C is the number of all possible classes, $\mathbf{y}_p$ and $\hat{\mathbf{y}_p}$ are the actual and predicted labels that belongs to class p, and f_s is the softmax function.

4.5 Optimizer

Once the difference between the predicted label and the actual label is obtained by loss function, the model should update the weights in order to reduce the loss. This is a typical optimization. The optimizer can be used to solve this problem.

Gradient Descent Optimizer

Gradient descent is a straightforward method to reduce the loss. In a training iteration t, the optimizer calculates the loss function $L(\theta_t)$ and gradient of the loss function $\nabla_\theta L(\theta_t)$ where θ represents the parameters (weights and bias) of the neural network. Then the optimizer updates the parameters by performing:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta_t) \tag{4.10}$$

Where α is called learning rate that controls the parameter changes of each iteration. If the learning rate is too large, the optimizer may skip the minimizer. If the learning rate is too small, the optimizer may stuck in a local minimizer.

Adam Optimizer

Adaptive Moment Estimation (Adam) is an advanced optimization method. Adam calculates the first and second moment as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta L_t(\theta) \tag{4.11}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)[\nabla_\theta L_t(\theta)]^2 \tag{4.12}$$

where hyperparameters β_1 and β_2 are close to 1. Since the two estimates m_t and v_t have biases towards zero, the following estimates $\hat{m}_t$ and $\hat{v}_t$ are used instead:

$$\hat{m}_t = \frac{m^t}{1 - \beta_1^t}\hat{v}_t = \frac{v^t}{1 - \beta_2^t} \tag{4.13}$$

Parameter θ_t is updated with the following rule:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{4.14}$$

Chapter 5

Comparison between Two LSTM netowrks

5.1 LSTM networks

The LSTM network is constructed from single LSTM cells. The structure of a single LSTM cell is shown in Fig. 5.1. The LSTM cell has three gates: input gate, forget gate and output gate. The LSTM gates allows LSTM to remember valuable information from the previous LSTM time steps and forget irrelavant information.

Forget gate

Forget gate is the first gate of a LSTM cell. Forget gate look at the input x_t and the hidden state from the last LSTM cell h_t and decide which part of information to discard. Also the forget gate merge the information from the last cell state C_{t-1}. The output of the forget gate can be seen as the weights of the last cell state. The sigmoid activation function limit the range of output of forget gate to be (0,1). If the output of the forget gate is 0, the corresponding values in C_{t-1} would be discarded. If the output of the forget gate is 1, the value would be passed to the next steps. The forget gate can be described as:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{5.1}$$

Where W_f, U_f and b_f is the weights of forget gate. σ is the sigmoid function.

Input gate

The input gate i_t decides which part of new information to store in the LSTM cells. The input gate consists of two parts. The first part is to select which part of information to update by using a sigmoid function:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{5.2}$$

Similar to forget gate, the W_i, U_i and b_i is the parameters of this part of input gate. The second part of input gate is a hyperbolic tangent function σ_g. This part decides that which part of information can be added to the cell state C_t:

$$\tilde{c}_t = \sigma_g(W_c x_t + U_c h_{t-1} + b_c) \tag{5.3}$$

Where W_c, U_c and b_c are the parameters and σ_g is the hyperbolic tangent function.

Cell state

The cell state C_t is the combination of forget gate f_t and input gate i_t and $\tilde{c}_t$. C_t represent the useful information of the input data and previous LSTM cells and the useless information would be dropped by forget gate:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \tag{5.4}$$

Output gate

Output gate is in charge of which part of information to output. The input of the output gate is x_t and h_{t-1}:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{5.5}$$

Still the parameters are W_o, U_o and b_o. The hidden state h_t is the output of the LSTM cells:

$$h_t = o_t \circ \sigma_g(c_t) \tag{5.6}$$

By using hyperbolic tangent function the range of $\sigma_g(c_t)$ is in (-1, 1). In this case the next hidden state can be both increasing and decreasing.

In this paper, we implement two types of LSTM networks: Many-To-Many (MTM)

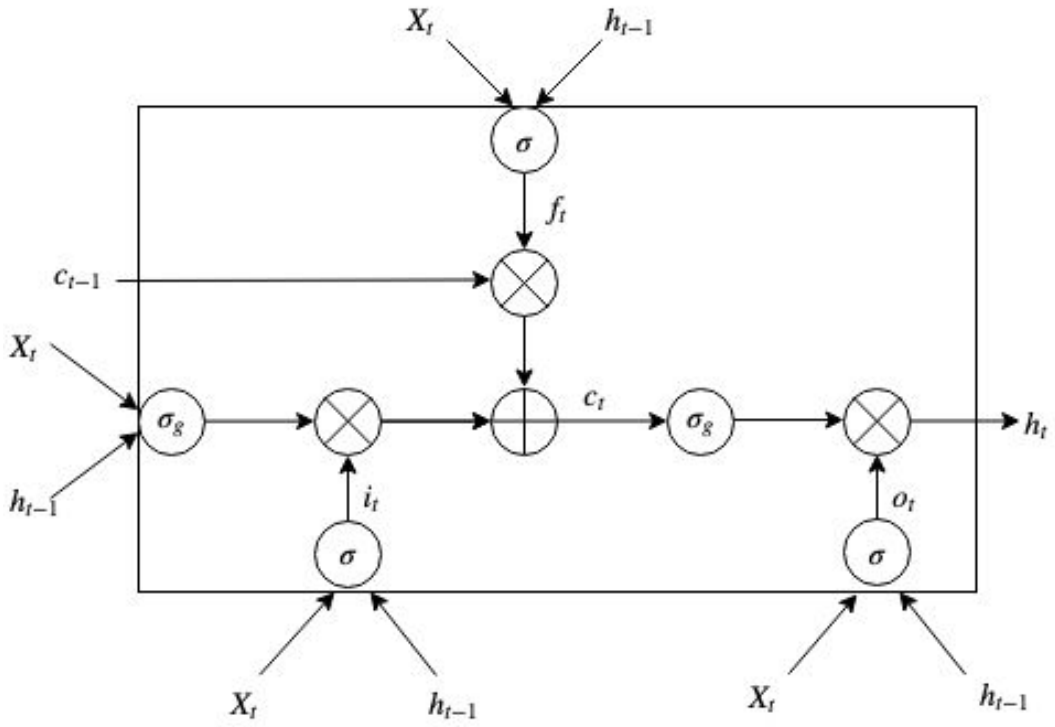


Figure 5.1: Single LSTM cell.

and Many-To-One (MTO). In MTM, a sequence of data packets is read into the IDS to predict if each data packet is a normal or attack traffic. In contrast, MTO takes in the sequence of data packets and make prediction on the last packet in the queue only.

5.2 MTM

Many-to-many LSTM takes a sequence of extracted packets and label the packets based on the information of the corresponding and previous ones. The structure of MTM is shown in Fig. 5.2. The loss function is cross entropy and the activation function is softmax. The first LSTM contains 64 units that equals the dimension of h_1. The second LSTM contains 32 units. Each sequence has $t = 10$ LSTM cells. x_i, where $i \in (1, t)$, is the i-th single input packet in a sequence. $\hat{y}_i$ is the label prediction for that packet. The optimizer we used is Adam [64] optimizer. MTM is a conventional structure of LSTM for multi-label classification but the disadvantage is that not all of the information in a sequence is available for each time step. The last

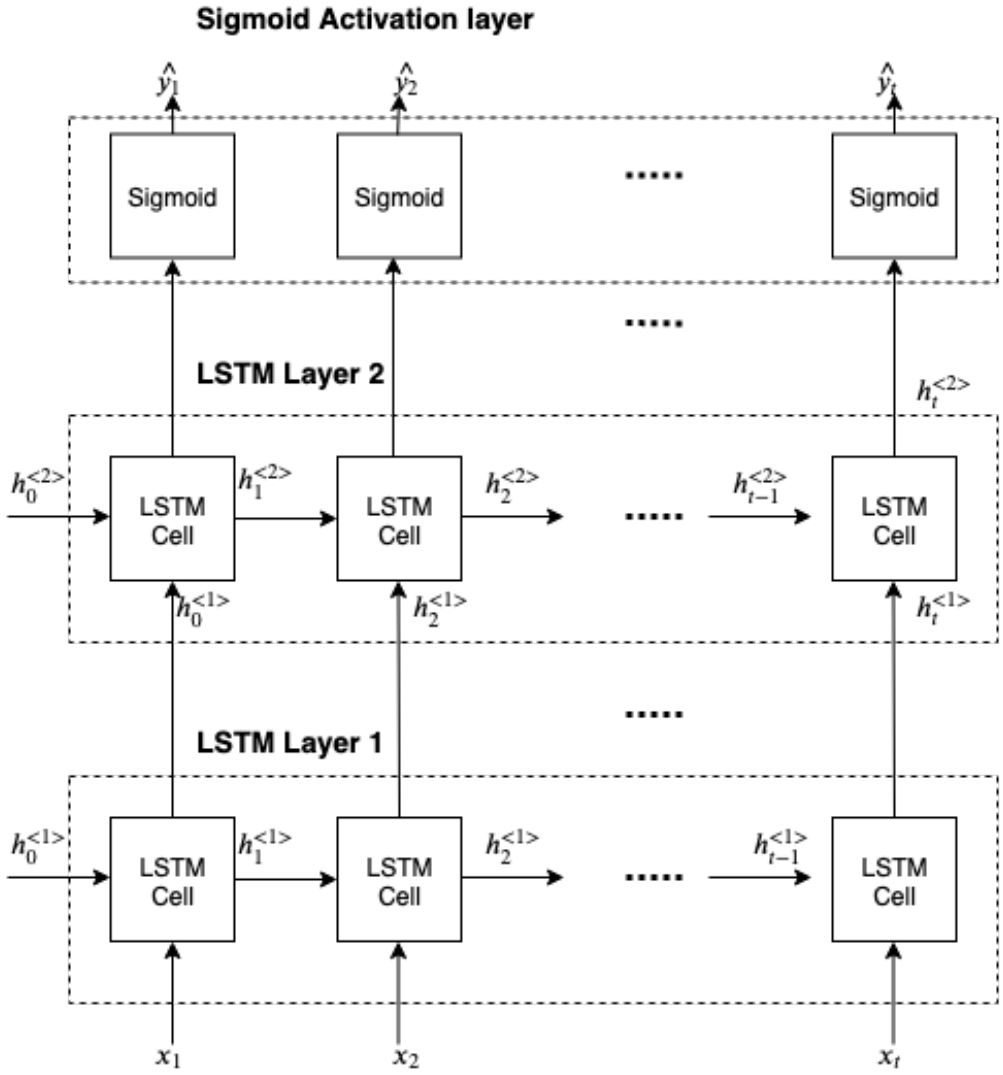


Figure 5.2: Many to many LSTM

few of the packets in each sequence would have more information than the beginning ones.

5.3 MTO

To address the drawbacks of MTM, we implement a MTO LSTM. The input of MTO is the same as MTM. The difference is that MTO only predicts the last data packet in the same sequence. The activation function is sigmoid. Other settings are the same as the MTM. The MTO structure is shown in Fig. 5.3.

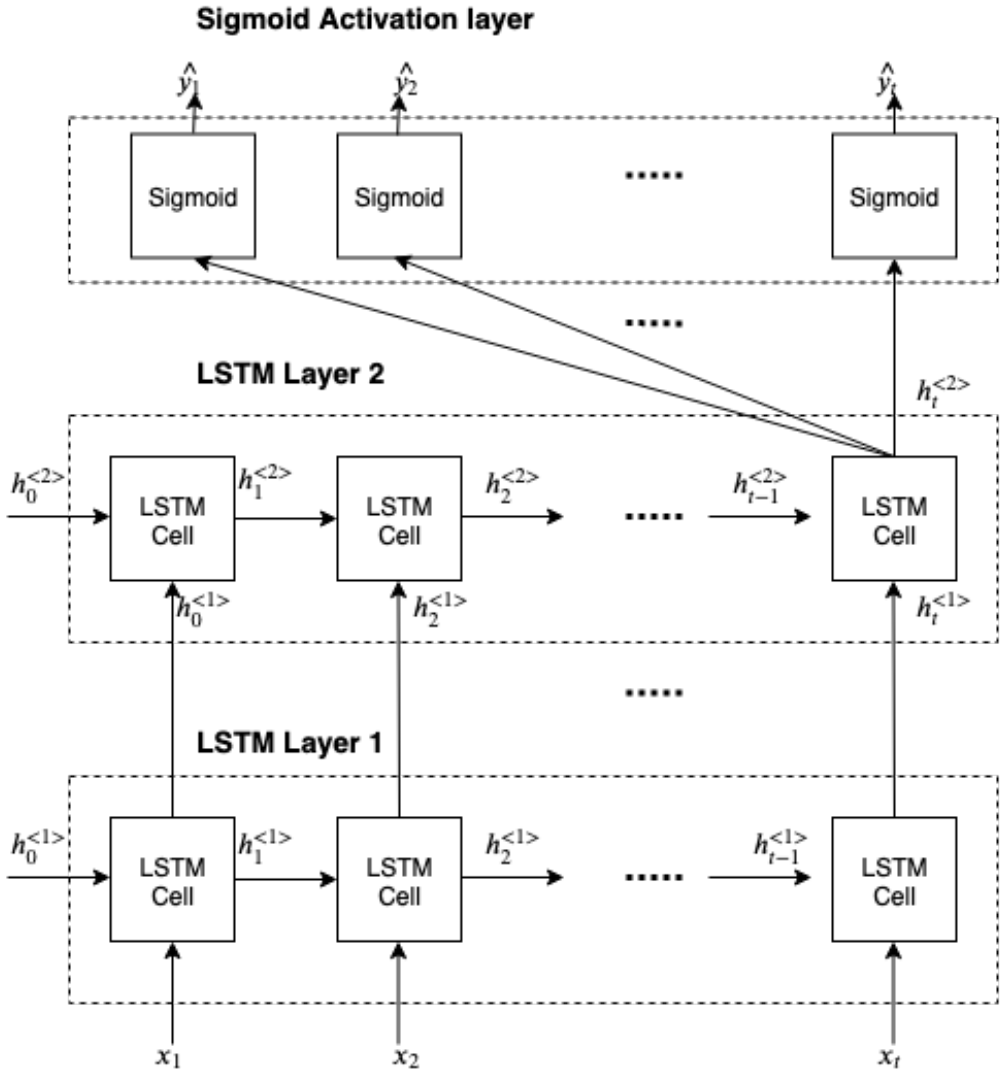


Figure 5.3: Many to one LSTM

5.4 Comparison

In experiments, Both of MTO and MTM models are trained by 80% of randomly chosen samples from the two datasets and the rest 20% sample are used for testing. The train/testing are performed 10 times to obtain mean and standard deviation of precision, recall and F_1. Standard deviation is evaluation of model stability.

We first compare the performance of both IDSs under temporal uncorrelated attacks. In doing so, both IDSs are trained and tested using Dataset I. Table 7.3 reveals that both of the two LSTM models achieve above 98.6% F_1 score for uncorrelated

Table 5.1: Comparison of the temporal uncorrelated attacks detection.

	Precision	Recall	F_1
MTO	98.9±0.2	98.3±0.1	98.6±0.1
MTM	99.5±0.2	99.0±0.3	99.3±0.2

Table 5.2: Comparison of temporal correlated attacks (%)

	Precision	Recall	F_1
MTO	99.1±0.1	98.9±0.1	99.0±0.1
MTM	93.2±0.2	92.0±0.1	92.4±0.1

Table 5.3: Macro-average comparison of online testing

	Precision	Recall	F_1
MTO	99.54±0.03	99.01±0.07	99.27±0.05
MTM	98.23±0.07	97.37±0.1	97.69±0.07

attacks and MTM performs slightly better than MTO.

We then compare their performance for correlated attacks by using Dataset II. Table 7.5 illustrates that the MTO has an outstanding performance with over 99% in precision, recall and F_1 for correlated attacks. In contrast, MTM only reaches 92% F_1 score.

Finally we train both models with combined Dataset I and II and implement them on the testbed for multi-class real time intrusion detection. As shown in Table 5.4, macro-averaged precision, recall and F_1 confirm that MTO outperfoms MTM in detecting all types of attacks despite that both experience degraded performance due to the inclusion of temporal uncorrelated attacks. Therefore, we adopt MTO as the LSTM IDS for the anomalous detection. Note that the results shown next are from MTO structure.

Our model structure is shown in Fig. 5.4. The input of the LSTM network is a 3 dimensional metrics. t is the timestep. $n_features$ is the number of features. Originally, the dataset is a $(n, n_features)$ metrics. By dividing the dataset into $\frac{n}{t}$ groups, we obtain a dataset with $(\frac{n}{t}, t, n_features)$ metrics as shown in Fig. **??**. This process is required by Keras. Each training time Keras would fit one chunk, (10, 19), in our cases to the LSTM network. In the prediction time, the LSTM would also take in one chunk and return the labels of the 10 sample of the chunk.

Two ways to organize the data into chunks: First is as shown above. The dataset is divided from n groups to $\frac{n}{5}$ groups. The other way is using a sliding window

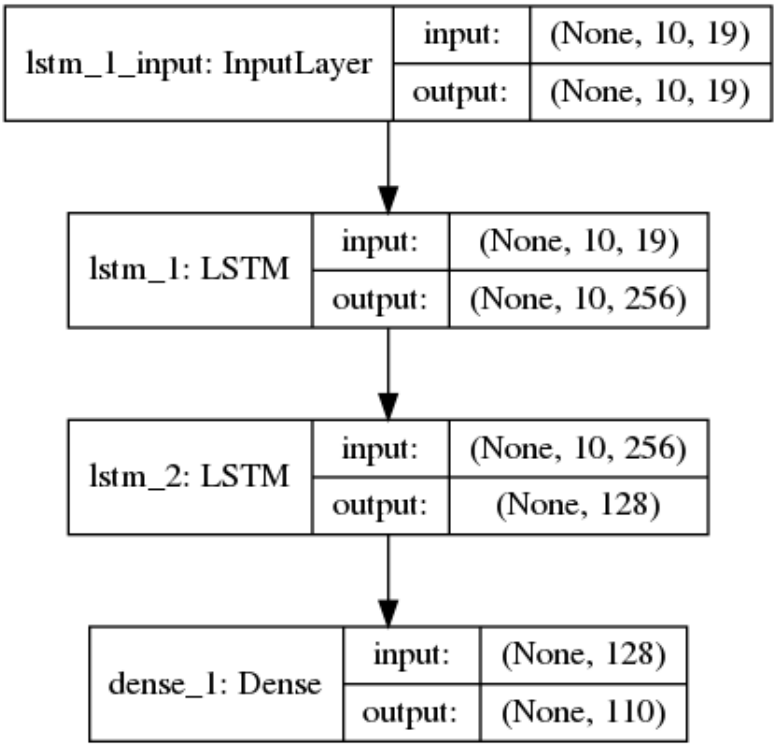


Figure 5.4: LSTM archtecture of Keras Model

Table 5.4: Macro-average comparison of online testing

	Precision	Recall	F_1
Chunk	99.54±0.03	99.01±0.07	99.27±0.05
Sliding Window	99.55±0.01	99.53±0.01	99.544±0.008

whose windows size is the time step. Each time the sliding window moves one step further. By using sliding window the dataset would still has n groups. Each group, however, contains t samples instead of 1. For example, given a data collection with n sample, $\{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, ..., \mathbf{x}^n\}$ and timestamp t, the output of the sliding window would be $\{\{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, ..., \mathbf{x}^t\}, \{\mathbf{x}^2, \mathbf{x}^3, \mathbf{x}^4, ..., \mathbf{x}^{t+1}\}, ..., \{\mathbf{x}^{n-t}, \mathbf{x}^{n-t+1}, \mathbf{x}^{n-t+2}, ..., \mathbf{x}^n\}\}$.

Only Dataset II is used for comparing the performance of two methods of organizing data. The result is illustrated in Table. **??**. As expected, the result of sliding window is identical to chunk. One advantage of chunk is the chunk saves spaces of RAM and time of training because the size is trunk is $\frac{1}{t}$ of sliding window. So the chunk method is selected for the LSTM training.

5.5 Public dataset

To validate the performance of MTO model, a testing is performed on a public SCADA dataset [67]. This dataset contains both correlated and uncorrelated attacks on SCADA system. The experiment is performing multi-label classification on the dataset. Since the imbalance attack/normal packet is not half-half, the evaluation metrics is using weighted precision, recall and F_1 score instead of macro. The results are shown in Table. 5.5. From the result we can see that the F_1 score of

Table 5.5: Weighted-average results of MTO on public dataset

	Precision	Recall	F_1
Results	93±2	94.7±0.08	93±1

MTO on public dataset is 93% which is 6% lower to 99%. 6% is an acceptable difference caused by the usage of different datasets. Therefore, the results from artificial datasets generated in this thesis is validated.

Chapter 6

Ensemble Learning

6.1 Ensemble Learning

Ensemble learning is a process that combines multiple machine learning models. Ensemble learning improves the performance of classification by aggregating several classifiers which were trained separately. One advantage of ensemble learning is explicitly reducing over-fitting.

The commonly used ensemble learning algorithms includes Bagging, boosting, random forest and stacking. In this thesis the random forest is adopted for the NDAE RF [14] reconstruction and stacking for the FNN-LSTM IDS implementation.

6.2 Stacking

Our FNN-LSTM ensemble IDS aims to combine the advantages of both FNN and LSTM while avoiding their weaknesses [65]. The schematics of this model is as shown in Fig. 7.2. In this model, the data packet features are fed into FNN and LSTM simultaneously to predict attacks as a multi-class classifier. The output labels of both are concatenated as the input of a multi-layer perceptron (MLP).

Our model structure is shown in Fig. 6.1. The output of both LSTM and FNN are $\in \mathcal{R}^{11}$ as there are totally 11 classes. By concatenating the output of FNN and LSTM, the input of ensemble are vectors with 22 features. The number of neurons in the hidden layer of the ensemble MLP is 100. The activation function is relu. The output layer would would output the probabilities of the 11 target classes. The optimizer is Adam optimizer. The loss function is cross entropy.

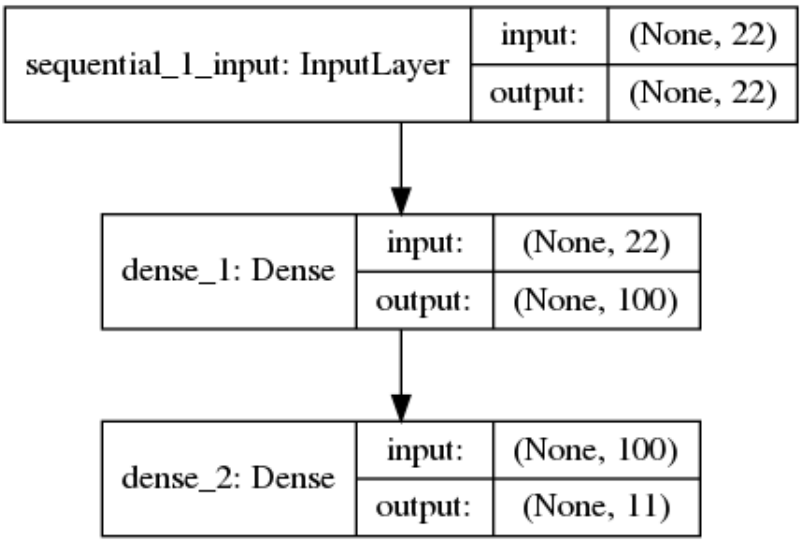


Figure 6.1: Ensemble archtecture of Keras Model

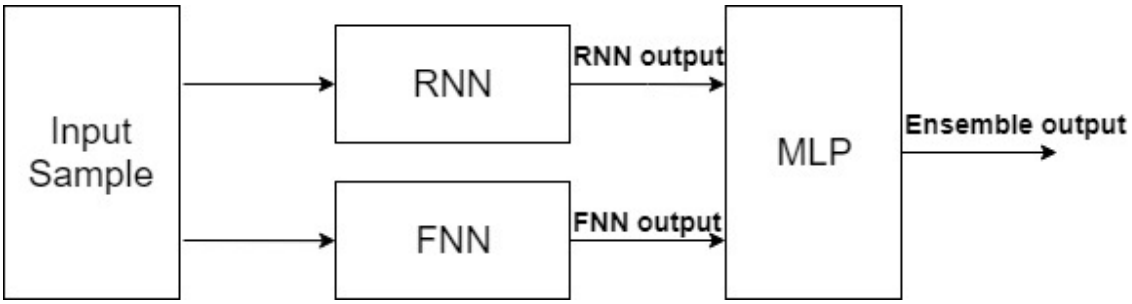


Figure 6.2: Ensemble Model.

6.3 Random forest

Random forest is a classic ensemble learning algorithm. Random forest is derived from the decision tree algorithm [66].

Decision Tree

A diagram of using decision tree to perform anomalous detection is shown in Fig. 6.3. The input of a decision tree is a data feature vector. The decision tree would split when a specific condition was met. In this diagram, the decision tree would first look at the destination IP address as the root. Then the tree split based on whether this IP address is an external or internal IP. The next split occurred based on whether the function is reading or writing register values. Eventually the decision tree made decision of whether this instance is abnormal at the leaves nodes. The drawback of decision tree is that it is very likely to be over fitting because the decision is made upon a few condition matching. An effective method to adddress this problem is random forest which aggregates a large number of decision trees.

Random Forest

In a classification problem, each trained decision tree can be seen as an "expert". But each of them may have different opinion of the classification problem. The random forest is the vote system that gathers the opinions of all the decision trees. Two benefits can be found by random forest. First is the precision of the decision. Another is preventing over-fitting. The diagram of random forest can be seen in Fig. 6.4. The data instance is fit as input to multiple decision trees. Each decision tree has four levels and outputs a class of the input instance. A majority voting system would gather the votes from the decision trees and return a final class.

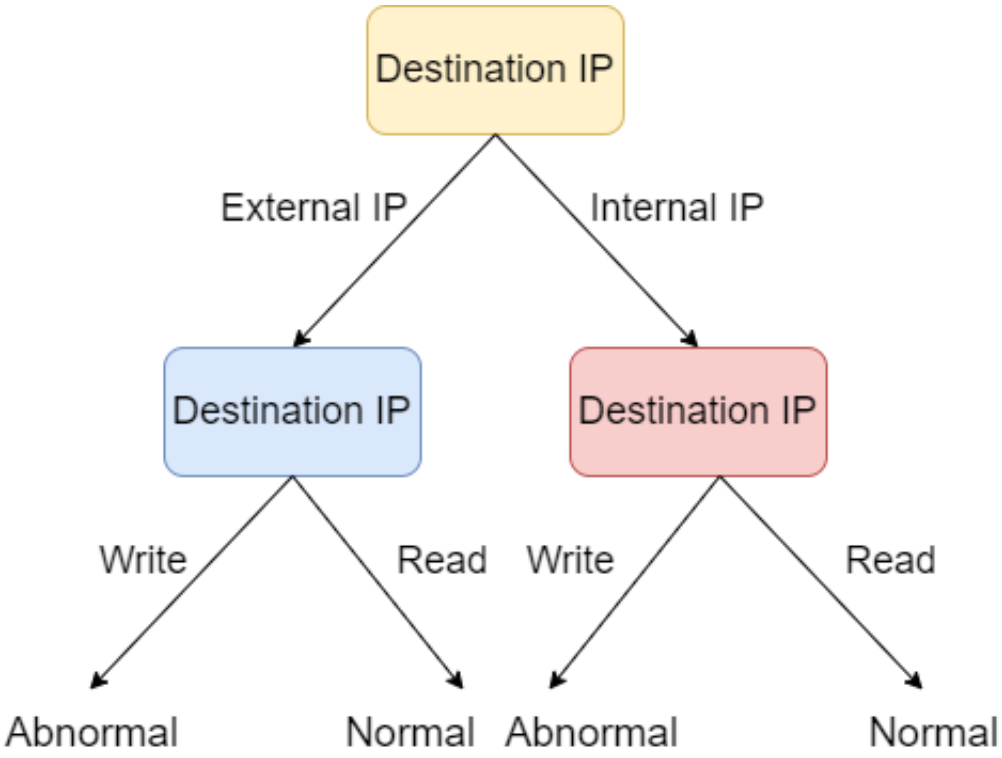


Figure 6.3: Decision Tree

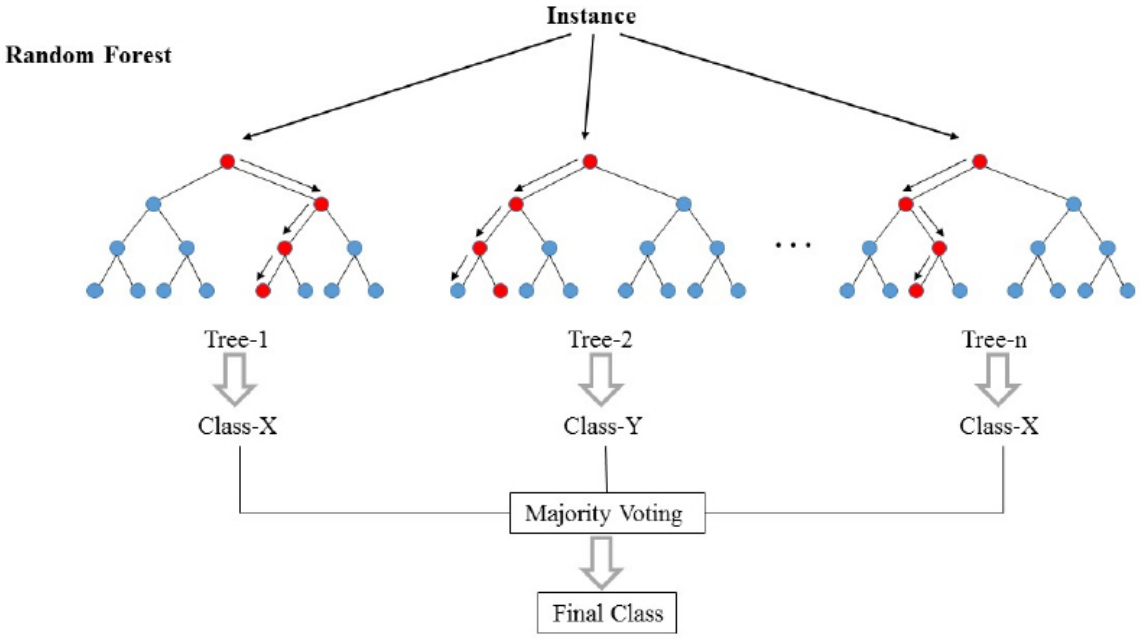


Figure 6.4: Random forest [15]

Unlike other machine learning algorithms such as logistic regression and neural networks, the hyper parameters of random forest doesn't have optimizers. Instead, the parameters of random forest includes:

1. Number of estimators: Number of decision tree
2. Max depth: The maximum depth of the tree
3. objective: Similar to loss function that measure the quality of split
4. Colsample_bynode: ratio of the features to be used for each node.
5. Colsample_bylevel: ratio of the features to be used for each level.
6. Colsample_bytree: ratio of the features to be used for each tree.
7. Learning rate: Same to FNN

The more decision tree involved the training, the more precise the classification result is. Also more decision trees can effectively reduce over fitting. The max depth is to control the quality of random forest. If the max depth is too small, the decision would be too shallow to make the correct prediction. If the max depth is too large, the system may not have enough resources to train the models. The objective would normally be cross entropy for classification problem. Colsample_bynode, Colsample_bylevel, Colsample_bytree parameters control the ratio of features to be used for training. This parameter should be fine tuned in order to prevent over fitting. Subsample is the sample used for training.

In this thesis, we borrow an idea call nonsymmetric deep autoencoder (NDAE) from [14] which combines the autoencoder algorithm and an supervised random forest anomalous detection. The structure can be seen as Fig. 6.5. The first part is the NDAE whose function is feature reduction. More details of autoencoder can be found in Appendix. A.1 The random forest part in our thesis is using XGBoost [19] by Tianqi Chen. The parameter is fine tuned using 10 times train/testing and grid search. The parameters is shown as Table. 6.1. The other parameters are the default in Xgboost.

From this diagram we can see that the NDAE is still working on each individual data samples instead of a sequence. It can be expected that the performance of NDAE might be better than FNN but not LSTM and the stacking model.

Colsample_bynode	0.9
max_depth	0.8
Number of estimators	100
Subsample	1.0

Table 6.1: Random forest parameters

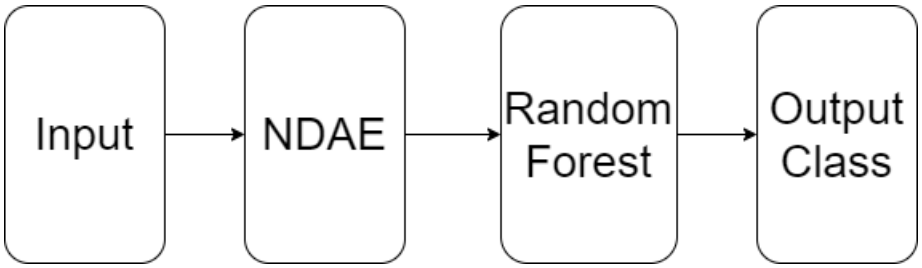


Figure 6.5: NDAE Random forest

Chapter 7

Results and Evaluation

7.1 Experiment and Result

To demonstrate their capability for detecting attacks with/without temporal correlation, we first implement FNN and LSTM IDSs to establish references for comparison. At this stage, the IDSs only conduct binary classification to predict if the data packet under investigation is normal (labeled as "0") or attack (labeled as "1"). Consequently, sigmoid function is selected as the activation function.

7.2 Evaluation

In this thesis we evaluate the performance of models by Precision, recall, and F1 score. These evaluation are derived from confusion matrix. A confusion matrix contains True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) shown as Table. 7.1. Confusion matrix illustrates details of the classification performance. However, it's not intuitive. Therefore, a single value being able to demonstrate the performance of evaluation is required. In this thesis, precision, recall and F_1 score are adopted to evaluate the performance as they are effective and easy to calculate.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Table 7.1: Confusion matrix example

Precision

Precision score is evaluating the ability of the classifier not labeling positive samples as negatives. The expression of the precision is:

$$Precision = \frac{TP}{TP + FP} \tag{7.1}$$

Recall

Recall score is measuring the ability of classifier labeling actual positive samples as positive. The expression of recall is:

$$Recall = \frac{TP}{TP + FN} \tag{7.2}$$

F_1

Both precision and recall may not be able to perfectly demonstrate the result. Normally, either one of the precision or recall is very high and the other one is very low. For example, if a classifier predict all of the samples to be positive, the recall score is 100% as $FN = 0$. However, the precision might be low because all of the actual negative sample are predicted as positive which leads to a high FP. Then the evaluation is still not clear as we don't know which one to trust. F_1 score is prompt to solve this problem by combining precision and recall score:

$$F1 = \frac{2 \times (Recall \times Precision)}{Recall + Precision} \tag{7.3}$$

F_1 score would be in between precision and recall. This balance the bias of precision and recall and becomes a reliable evaluation metrics.

Macro average

The discussion above is based on binary classification i.e. the result of a classifier is either positive (1) or negative (0). However, in multi-class classification, the defination of positive and negative need some modification: The positive represents the labels being labeled correctly and the negative means the labels being labeled wrongly. Nevertheless, we have precision, recall and F_1 score for each of the classes. Evaluating the results of each single class is trivial and not intuitive. In order to merge the results of each classes, averaging methods are used. The commonly used averaging methods

include micro, macro and weighted. In this thesis we only use macro average. Macro average calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account. In our cases, the dataset is in balance so that the macro average can be used to evaluate the performance.

7.3 Hyper parameters tuning

Both IDSs are trained using 70% of the randomly chosen samples from the two datasets and tested with the remaining 30% samples following the 10-fold training/testing procedure so that the average and standard deviation of figures of merits including precision, recall and F_1 can be used for evaluation.

To determine the number of hidden layers necessary for our FNN, we computed F_1 with 0, 1 and 2 hidden layers where the values of 99.22%, 99.96% and 99.97% are obtained respectively. As shown, employing 1 hidden layers in FNN will increase the F_1 by more than 0.74% while using 2 hidden layers the improvement is minimal. Therefore, we select 1 hidden layer in our FNN implementation.

In addition, to circumvent overfitting, we further adopted early stop procedure in FNN such that the optimization stops when the number of epochs whose relative differences of loss between consecutive ones are less than 10^{-6} reaches 35 [68]. Similarly, LSTM adopts early stop if either maximum epochs reach 3.

In implementation of LSTM, we connect 10 LSTM cells in input layer where the features from 10 consecutive data packets are entered into the cells to predict if the last packet is normal or an attack. We adopt mini-batch with a batch size of $1,000$ for training with a 10 times train/testing.

7.4 Time consumption, workload and scalability

Using an Nvidia GTX1070 GPU with 151 W maximum power consumption, 1920 CUDA cores operating at 1506 MHz base clock and 8 GB DDR5 memory, we tested the training time for all the models. During the traing, the GPU used 7.758 MB memory with power consumption of around 36 W and volatile GPU utilization of 23%. The results are displayed in Table 7.2. Among all models, the LSTM converges the fastest at 229 seconds while FNN and NDAE RF converge within 2,046 and 2,053 seconds respectively. The apparent short time consumption of 656 seconds for the ensemble is due to the fact that the FNN and LSTM in this model are pre-trained. In addition,

Table 7.2: Comparison of time consumption.

	Training Time (s)
FNN	2,046
LSTM	229
NDAE RF	2,053
Ensemble	656

Table 7.3: Comparison of the temporally uncorrelated attacks detection (%).

	Precision	**Recall**	**F_1**
FNN	99.996±0.006	99.84±0.05	99.92±0.03
LSTM	99.88±0.06	98.7±0.4	99.3±0.1
NDAE RF	99.2±0.1	97.61±0.09	98.44±0.08

our model is scalable on larger SCADA networks where more network elements are incorporated.

7.5 Detection of temporally uncorrelated attacks

We exploit the Dataset I described in Section 3.2 to compare the detection capability of FNN, LSTM for temporally uncorrelated attacks. To verify the models, learning curves are plotted in Fig. 7.1 where training and testing losses as a function of training samples are plotted. Here the average value and standard deviation after 10 fold training/testing are represented by circle markers and error bars respectively. As shown, with training samples exceeding 40,000, FNN training and testing losses (blue dashed lines) start to converge while LSTM (red solid lines) converges at sample size larger than 60,000. Overall, it confirms that the number of samples in Dataset I is sufficient for the training and testing of our IDS.

After the IDSs are trained, we use 30% of samples in Dataset I for 10 fold testing. Also shown in Table 7.3 and 7.4, on average, for FNN, only 0.6 of the 69,846 normal data packets are mislabelled as attacks while only 30.7 out of 19,771 actual attacks are mislabelled as normal traffic, yielding the precision, recall and F_1 to be 99.996±0.006%, 99.84±0.05%, and 99.92±0.03% in detecting known attacks while comparison of unknown attacks is out of the scope of this paper. In comparison, LSTM mislabelled 22.8 normal packets as attacks and 241.9 attacks as normal packets, resulting the figures of merits to be 99.88±0.06%, 98.7±0.4% and 99.3±0.1%. Throughout this article, we use the standard deviation of measurements to estimate

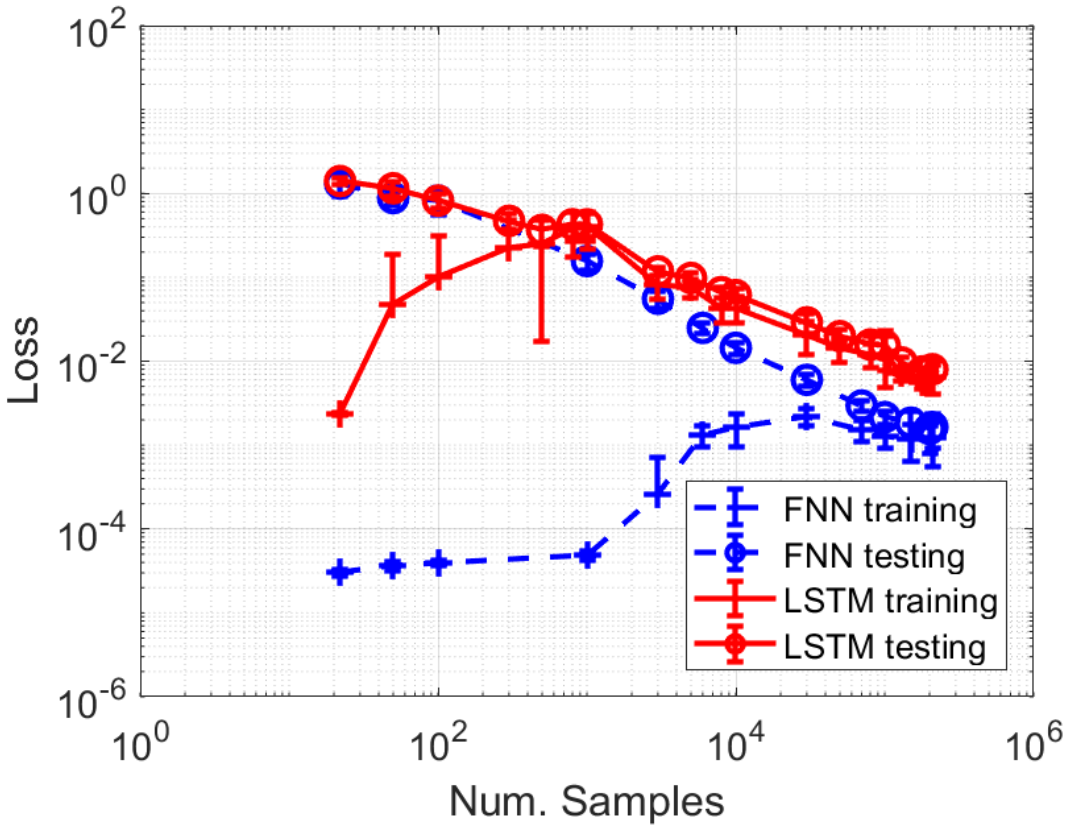


Figure 7.1: Learning Curves of FNN and LSTM using temporally-uncorrelated-attacks dataset (Dataset I).

Table 7.4: Confusion matrices of temporally uncorrelated attacks detection using Dataset I (averaged over 10 trials).

			Predicted	
			Normal	**Attacks**
Actual	**Normal**	**FNN**	69,845.4	0.6
		LSTM	69,902.2	22.8
		NDAE RF	69714.5	139.4
	Attacks	**FNN**	30.7	19,741.3
		LSTM	241.9	19,448.1
		NDAE RF	471	19293.1

Table 7.5: Comparison of temporally correlated attacks (%).

	Precision	**Recall**	**F_1**
FNN	73±2	49±4	58±2
LSTM	99.60±0.01	99.52±0.02	99.56±0.01
NDAE RF	76.0±0.2	59.9±0.3	67.0±0.2

Table 7.6: Confusion matrix of temporally correlated attacks.

			Predicted	
			Normal	**Attacks**
Actual	**Normal**	**FNN**	28,668.3	5,044.7
		LSTM	33,504.0	105.0
		NDAE RF	28,570.5	5,045.1
	Attacks	**FNN**	13,510.4	13,169.6
		LSTM	128.4	26,652.6
		NDAE RF	10,723.1	16,054.3

the precision. In addition, we also compared our results with NDAE RF using the same dataset and training/testing splitting ratio. As shown, NDAE RF achieves lower precision (99.2±0.1%), recall (97.61±0.09%) and F_1 (98.44±0.08%). The comparison demonstrates that FNN outperformed LSTM and NDAE RF in detecting temporally uncorrelated attacks where recognition of the in-packet feature patterns is critical.

7.6 Detection of temporally correlated attacks

In this subsection FNN and LSTM are re-trained and tested using Dataset II for comparison of their temporally correlated attacks detection comparison. Again the learning curves in Fig. 7.2 shows that both FNN (blue dashed lines) and LSTM (red

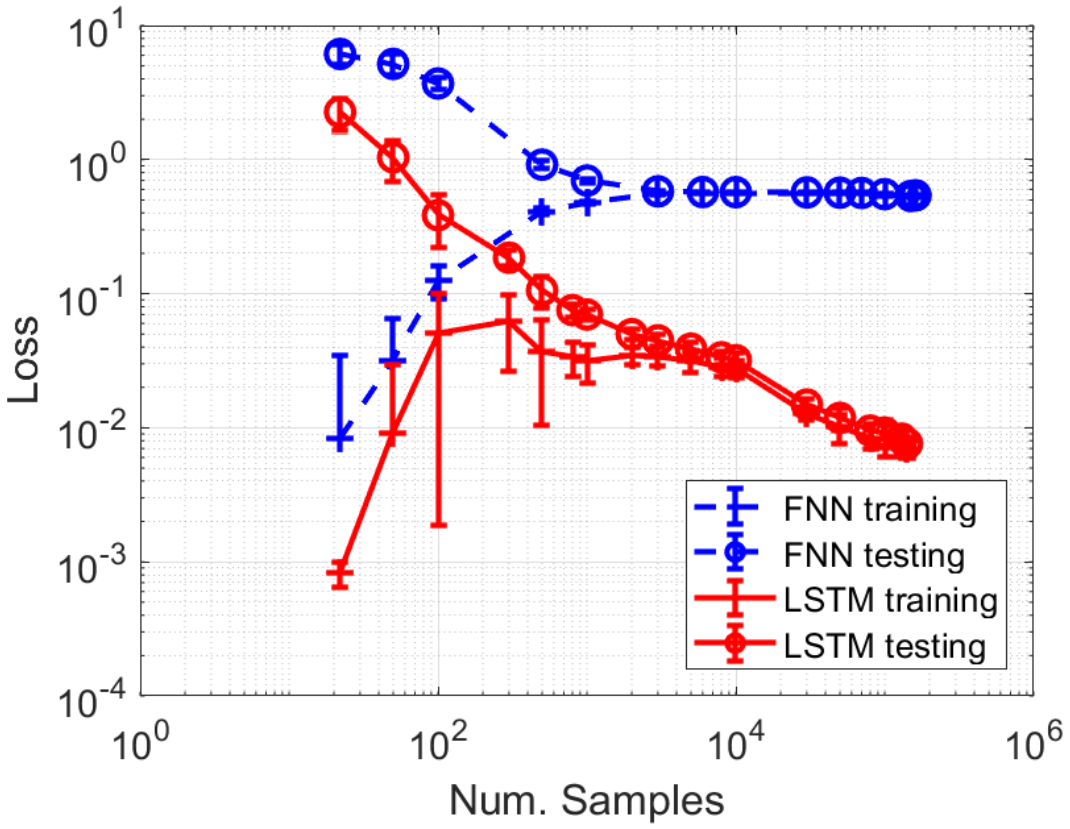


Figure 7.2: Learning Curves of FNN and LSTM using temporally-correlated-attacks dataset (Dataset II).

solid lines) converge at training samples exceeding 10,000 while LSTM clearly shows lower testing loss. This confirms the sufficiency of our dataset to generalize the IDS models.

The performance of each model is compared in Table 7.5 and 7.6. As shown, FNN is inefficient in detecting temporally correlated attacks with precision, recall and F_1 scores as low as 73±2%, 49±4% and 58±2 respectively. In particular, 5,044.7 out of 33,713 normal packets are mislabelled to attacks while 13,510.4 out of 26,680 actual attacks are mislabelled to normal traffic. It is evident that the poor performance of FNN is caused by its inability to identify inter-packet feature patterns. NDAE RF demonstrates a better performance than FNN in temporally correlated attacks but achieves relatively low precision (76.0±0.2%), recall (59.9±0.3%) and F_1 (67.0±0.2%). In contrast, LSTM displays an outstanding performance as the corresponding figures of merits reach 99.60±0.01%, 99.52±0.02% and 99.56±0.01% where only 105.0 normal

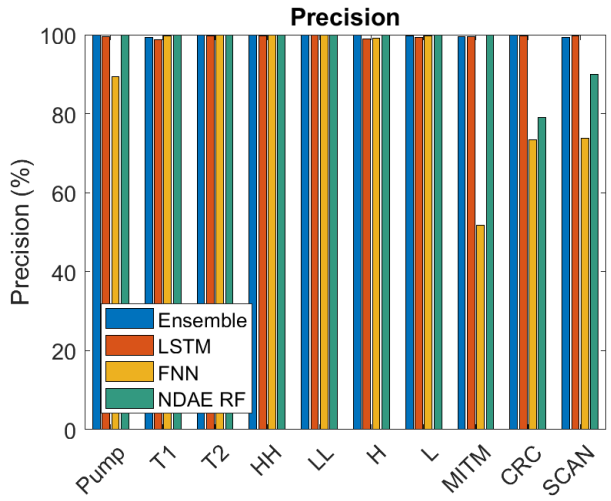
Table 7.7: Macro-average comparison of omni-attacks detection (%)

	Precision	**Recall**	**F_1**
FNN	88±1	89.2±0.8	87.4±0.6
LSTM	99.54±0.03	99.01±0.07	99.27±0.05
NDAE RF	96.9±0.5	97±1	96.8±0.9
Ensemble	99.76±0.05	99.57±0.03	99.68±0.04

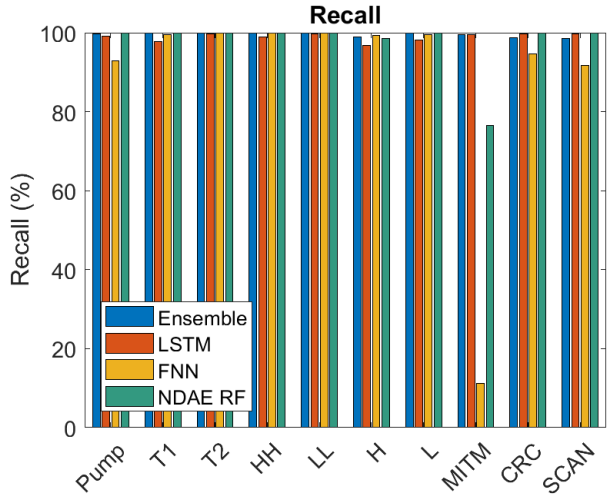
packets are mislabelled as attacks and 128.4 attacks packets are mislabelled as normal traffic. As expected, LSTM outperforms FNN and NDAE RF in detecting temporally correlated attacks due to its inherent nature to observe data pattern in time domain.

7.7 Omni attacks detection

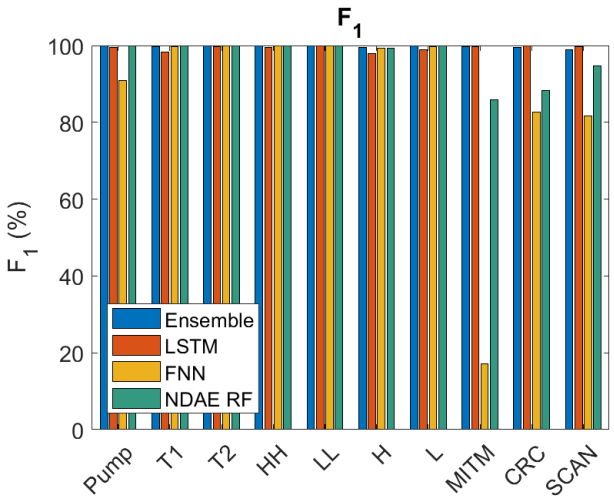
Recognizing the mutual strength and weakness of FNN and LSTM IDSs in detecting temporally correlated and uncorrelated attacks, we here combine the advantages of both for an omni attacks detector through ensemble approach. The structure of FNN-LSTM ensemble is described in Subsection 6. To implement, we first remodelled FNN and LSTM to multi-class classifiers so that different attacks can be distinguished. Dataset I and II are combined and used to train FNN and LSTM independently. The outputs of both are combined to form the input features of a multilayer perceptron for training. After training, FNN, LSTM and FNN-LSTM ensemble IDSs are integrated into our SCADA testbed to detect and classify attacks online. The online traffic is generated using the script that generates a pre-determined ratio of normal, temporally correlated and uncorrelated attacks as described in Fig 3.5. To estimate the figures of merits, we evenly divide the predicted labels to 10 portions and compute the average and standard deviation of macro-averaged precision, recall and F_1. As shown in Table 7.7, among all the four IDSs, the FNN achieves lowest performance with macro-averaged figures of merits of 88±1%, 89.2±0.8% and 87.4±0.6%. NDAE RF performs better with 96.9±0.5%, 97±1% and 96.8±0.9% while LSTM reaches even higher scores of 99.54±0.03%, 99.01±0.07% and 99.27±0.05%. In contrast, the FNN-LSTM ensemble IDS further outperforms all other IDSs with figures of merits to be 99.76±0.05, 99.57±0.03 and 99.68±0.04. Detailed analysis in Fig. 7.3 confirms that the under-performance of FNN (yellow bars) are due to the mislabels of temporally correlated attacks (MITM, CRC and SCAN) while the performance of LSTM (red bars) by temporally uncorrelated attacks ("Pump Speed (Pump)", "Tank



(a)



(b)



(c)

Figure 7.3: (a) Precision, (b) Recall and (c) F_1 of individual attacks in omni-attacks detection.

1 Level (T1)", and "Threshold High (H)", etc.). Overall, the FNN-LSTM ensemble demonstrates a consistent out-performance over them in all types of attacks.

Chapter 8

Future works and Conclusion

8.1 Conclusion

In this paper we implement and compared two variants of LSTM networks. An unsupervised learning classifier NDAE RF is reconstructed. Also this paper demonstrated that the FNN-LSTM ensemble IDS can detect all types of cyberattacks regardless of the their temporal relevance. In opposite, FNN only performance well in temporally uncorrelated attacks and LSTM is relatively weak in uncorrelated attacks. In future research we will further improve our model through field trials.

8.2 Future work

The future work could be:

1. Develop new threats of SCADA including buffer overflow, IoT malware, worms (stuxnet) etc.
2. New features in SCADA such as adding PID controllers
3. Research anomaly-based unsupervised learning model such as autoencoder following a K means clustering algorithm.
4. Feature engineering: adding more features from TCP, Modbus protocols and environment conditions
5. Implement generative neural networks to analyze the traffic such as GAN

6. Search for bi-directional RNN to overcome the drawback of MTM LSTM.
7. Deploy the SCADA to a hardware testbed in order to obtain a real test environment
8. Synthesis packet loss data.
9. Tune imbalances of SCADA dataset. Fix the imbalance of dataset I to be 50% attacks. Also generate security dataset with 5% attacks for testing.

Appendix A

Additional Information

A.1 Autoencoder

Autoencoder is an unsupervised learning algorithm that is used for anomaly detection. Autoencoder has the similar structure of FNN but the target is the input itself instead of a class. Fig. A.1 shows the structure of a basic autoencoder. This is a neural network with one hidden layer. The input of the autoencoder is the data sample $\mathbf{x}$. The output of the autoencoder is the reconstructed input sample . The autoencoder can be separated to two parts: encoder and decoder. The encoder part take in the input $\mathbf{x}$ and perform:

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{A.1}$$

Where σ is the activation function. $\mathbf{W}, \mathbf{b}$ is the parameters of the encoder.

The decoder take $\mathbf{h}$ as the input and perform:

$$\hat{\mathbf{x}} = \sigma(\hat{\mathbf{W}}\mathbf{h} + \hat{\mathbf{b}}) \tag{A.2}$$

Where $\hat{\mathbf{W}}, \hat{\mathbf{b}}$ is the parameters of decoder. The loss function is square error between $\mathbf{x}$ and :

$$\mathcal{L} = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 \tag{A.3}$$

The purpose of training the autoencoder is to minimise the loss function in order to reconstruct the $\mathbf{x}$.

Once the training phase finished, The hidden layer $\mathbf{h}$, also called coding layer, learns how to map the feature to a low dimension vector. Therefore, autoencoder are often used for feature reduction. The output of coding layer can be seen as the reduced

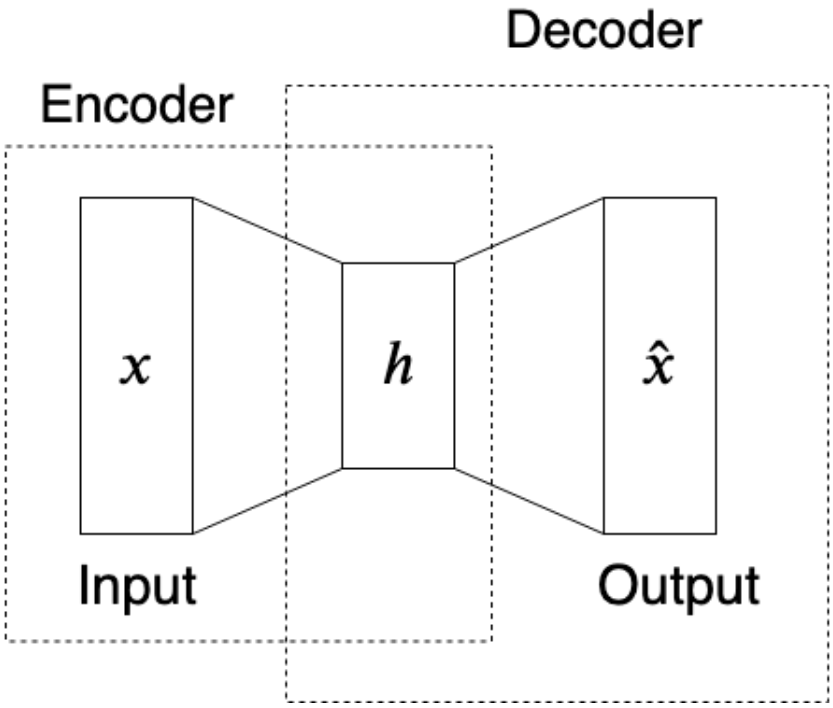


Figure A.1: Simple autoencoder

feature vector which reserves useful information and discards irrelevant information from the raw feature vector. Thus, the decoder would be discarded as we do not need the reconstructed input sample in this thesis. The output of coding layer would be sent to a supervised learning algorithm such as random forest.

Bibliography

[1] S. Adepu and A. Mathur, "An investigation into the response of a water treatment system to cyber attacks," in *17th IEEE International Symposium on High Assurance Systems Engineering (HASE 2016)*, pp. 141–148, 2016.

[2] SCADA diagram. https://www.electricalinput.com/2018/08/scada-supervisory-control-and-data.html

[3] H. Huang, W. Zhang, G. Qi, S. Ma, Y. Yang, F. Yan, and P. Chen, "Research on accident inversion and analysis method of the oil and gas pipeline SCADA system," in *2014 Sixth International Conference on Measuring Technology and Mechatronics Automation*, pp. 492–496, 2014.

[4] S. Karnouskos, "Stuxnet worm impact on industrial cyber-physical system security," in *IEEE 37th Annual Conference of the Industrial Electronics Society IECON 2011*, pp. 4490–4494, 2011.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

[6] Cho, Kyunghyun, van Merrienboer, Bart, Gülçehre, Çaglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger and Bengio, Yoshua. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.." Paper presented at the meeting of the EMNLP, 2014.

[7] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent neural network for text classification with multi-task learning. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16). AAAI Press, 2873–2879.

[8] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," in IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673-2681, Nov. 1997.

[9] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. 2015. Convolutional LSTM Network: a machine learning approach for precipitation nowcasting. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15). MIT Press, Cambridge, MA, USA, 802–810.

[10] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities," in IEEE Internet of Things Journal, vol. 1, no. 1, pp. 22-32, Feb. 2014.

[11] Anderson, James M., Nidhi Kalra, Karlyn D. Stanley, Paul Sorensen, Constantine Samaras, and Tobi A. Oluwatola, Autonomous Vehicle Technology: A Guide for Policymakers. Santa Monica, CA: RAND Corporation, 2016.

[12] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, ser. LISA '99. Berkeley, CA, USA: USENIX Association, pp. 229–238, 1999.

[13] M. A. Aydın, A. H. Zaim, and K. G. Ceylan, "A hybrid intrusion detection system design for computer network security," *Computers & Electrical Engineering*, vol. 35, no. 3, pp. 517 – 526, 2009.

[14] N. Shone, T. N. Ngoc, V. D. Phai and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41-50, 2018.

[15] Random forest. https://www.linkedin.com/pulse/random-forest-algorithm-interactive-discussion-niraj-kumar/

[16] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. Neural Comput. 18, 7 (July 2006), 1527–1554.

[17] W. Gao and T. Morris, "On cyber attacks and signature based intrusion detection for modbus based industrial control systems," *Journal of Digital Forensics, Security and Law*, Vol. 9, No. 1, Article. 3, 2014.

[18] Y. Wang, Z. Xu, J. Zhang, L. Xu, H. Wang, and G. Gu, "SRID: State relation based intrusion detection for false data injection attacks in SCADA," in *Computer Security - ESORICS 2014*, pp. 401–418, 2014.

[19] Chen, Tianqi, and Carlos Guestrin. "XGBoost." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16 (2016): n. pag. Crossref. Web.

[20] V. Jyothsna, V. V. Rama Prasad, and K. Munivara Prasad, "A review of anomaly based intrusion detection systems," *International Journal of Computer Applications*, vol. 28, pp. 26–35, 08 2011.

[21] D. Damopoulos, S. A. Menesidou, G. Kambourakis, M. Papadaki, N. Clarke, and S. Gritzalis, "Evaluation of anomaly-based IDS for mobile devices using machine learning classifiers," *Security and Communication Networks*, vol. 5, no. 1, pp. 3–14, 2012.

[22] G. Nascimento and M. Correia, "Anomaly-based intrusion detection in software as a service," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 19–24, June 2011.

[23] M. S. Islam and S. A. Rahman, "Anomaly intrusion detection system in wireless sensor networks: security threats and existing approaches," *International Journal of Advanced Science and Technology*, vol. 36, no. 1, pp. 1–8, 2011.

[24] The Modbus Organization, "Modbus application protocol specification" [Online]. Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

[25] O. Linda, T. Vollmer, and M. Manic, "Neural network based intrusion detection system for critical infrastructures," in *2009 International Joint Conference on Neural Networks*, pp. 1827–1834, 2009.

[26] J. Rrushi and K.-D. Kang, "Detecting anomalies in process control networks," *Critical Infrastructure Protection III*, pp. 151–165, 2009.

[27] G. Poojitha, K. N. Kumar, and P. J. Reddy, "Intrusion detection using artificial neural network," in *Computing Communication and Networking Technologies (ICCCNT), 2010 International Conference on Computing, Communication and Networking Technologies.* pp. 1–7, 2010.

[28] Y. Zhang, L. Wang, W. Sun, R. C. Green II, and M. Alam, "Distributed intrusion detection system in a multi-layer network architecture of smart grids," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 796–808, 2011.

[29] L. A. Maglaras and J. Jiang, "Intrusion detection in scada systems using machine learning techniques," in *2014 Science and Information Conference*. London, pp. 626–631, 2014.

[30] L. Zhang, "An implementation of scada network security testbed," *Master's thesis, University of Victoria, Victoria, BC*, 2015.

[31] S. Patel, "IEC-61850 Protocol Analysis and Online Intrusion Detection System for SCADA Networks using Machine Learning"Master's thesis, University of Victoria, Victoria, BC, 2017.

[32] L. T. Heberlein and M. Bishop, "Attack class: address spoofing," in *Proceedings of the 19th National Information Systems Security Conference*, 1997.

[33] S. Kumar, "Smurf-based distributed denial of service (ddos) attack amplification in internet," in *Second International Conference on Internet Monitoring and Protection (ICIMP 2007)*, San Jose, CA, pp. 25–25, 2007.

[34] B. Chen, N. Pattanaik, A. Goulart, K. Butler-Purry, and D. Kundur, "Implementing attacks for modbus/tcp protocol in a real-time cyber physical system test bed," *Proceedings - CQR 2015: 2015 IEEE International Workshop Technical Committee on Communications Quality and Reliability*, pp. 1–6, 2015.

[35] N. Sayegh, A. Chehab, I. H. Elhajj, and A. Kayssi, "Internal security attacks on scada systems," in *2013 Third International Conference on Communications and Information Technology (ICCIT)*, pp. 22–27, 2013.

[36] D. Yang, A. Usynin, and J. Hines, "Anomaly-based intrusion detection for SCADA systems," in *Proceedings of the 5. International Topical Meeting on Nuclear Plant Instrumentation Controls, and Human Machine Interface Technology*, vol. 43, no. 47, pp. 797–803, 2006.

[37] W. Gao, T. Morris, B. Reaves, and D. Richey, "On scada control system command and response injection and intrusion detection," *2010 eCrime Researchers Summit*, Dallas, TX, pp. 1-9, 2010.

[38] Mirsky. Yisroel, Tomer. Doitshman, Yuval. Elovici and Asaf. Shabtai. "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection." in *ArXiv abs/1802.09089 (2018): n. pag.*

[39] L. A. Maglaras, J. Jiang, and T. Cruz, "Integrated ocsvm mechanism for intrusion detection in SCADA systems," *Electronics Letters*, vol. 50, no. 25, pp. 1935–1936, 2014.

[40] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," *arXiv e-prints*, p. arXiv:1303.5778, Mar. 2013.

[41] D. Eck and J. Schmidhuber, "A first look at music composition using lstm recurrent neural networks," Technical Report. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 2002.

[42] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *arXiv e-prints*, p. arXiv:1409.3215, Sep. 2014.

[43] Chawla, A., Lee, B., Jacob, P., Fallon, S. (2019). Bidirectional LSTM Autoencoder for Sequence Based Anomaly Detection in Cyber Security.

[44] A. M. Salem, K. Revett and E. A. El-Dahshan, "Machine learning in electrocardiogram diagnosis," 2009 International Multiconference on Computer Science and Information Technology, Mragowo, 2009, pp. 429-433.

[45] C. D. McDermott, F. Majdani, and A. V. Petrovski, "Botnet detection in the internet of things using deep learning approaches," in *2018 International Joint Conference on Neural Networks (IJCNN)*, Rio de Janeiro, pp. 1–8, 2018.

[46] Alguliyev, R. M., Aliguliyev, R. M., Abdullayeva, F. J. (2019). The Improved LSTM and CNN Models for DDoS Attacks Prediction in Social Media. International Journal of Cyber Warfare and Terrorism (IJCWT), 9(1), 1-18.

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Trans. Pattern Anal. Mach. Intell. 39, 6 (June 2017), 1137–1149.

[48] C. Feng, T. Li, and D. Chana, "Multi-level anomaly detection in industrial control systems via package signatures and lstm networks," in *2017 47th Annual IEEE/I-*

FIP International Conference on Dependable Systems and Networks (DSN), Denver, CO, pp. 261–272, 2017.

[49] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[50] Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. In NIPS 2014 Workshop on Deep Learning, December 2014

[51] T. Morris and W. Gao, "Industrial control system traffic data sets for intrusion detection research," in *Critical Infrastructure Protection VIII*, J. Butts and S. Shenoi, Eds. ICCIP 2014. IFIP Advances in Information and Communication Technology, vol 441. pp 65–78,2014.

[52] T. Morris, A. Srivastava, B. Reaves, W. Gao, K. Pavurapu, and R. Reddi, "A control system testbed to validate critical infrastructure protection concepts," *International Journal of Critical Infrastructure Protection*, vol. 4, no. 2, pp. 88 – 103, 2011.

[53] H. Wang, T. Lu, X. Dong, P. Li, and M. Xie, "Hierarchical online intrusion detection for scada networks," *arXiv*, p. 1611.09418, 2016.

[54] S. Patel, "IEC-61850 Protocol Analysis and Online Intrusion Detection System for SCADA Networks using Machine Learning," Master's thesis, University of Victoria, Victoria, BC, 2017.

[55] MBLogic. Mblogic homepage. [Online]. Available: http://mblogic.sourceforge.net/index.html

[56] N. Provos. Honeyd. [Online]. Available: http://www.honeyd.org/

[57] ProconX Pty Ltd. Modpoll modbus master simulator. [Online]. Available: http://www.modbusdriver.com/modpoll.html

[58] Ettercap, a comprehensive suite for man in the middle attacks. [Online]. Available: http://openmaniak.com/ettercap.php

[59] Z. Trabelsi and W. El-Hajj, "Arp spoofing: A comparative study for education purposes," in *2009 Information Security Curriculum Development Conference*, ser. InfoSecCD '09. New York, NY, USA: ACM, pp. 60–66, 2009.

[60] Snort ARP spoof preprocessor. [Online]. Available: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node17.html

[61] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[62] M. Abadi, et. al. "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

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

[64] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[65] T. G. Dietterich, "Ensemble methods in machine learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, ser. MCS '00. London, UK, UK: Springer-Verlag, pp. 1–15, 2000.

[66] J. R. Quinlan. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (March 1986), 81–106.

[67] Morris, T., Gao, W., "Industrial Control System Network Traffic Data sets to Facilitate Intrusion Detection System Research," in Critical Infrastructure Protection VIII, Sujeet Shenoi and Johnathan Butts, Eds. ISBN: 978-3-662-45354-4. Due November 14, 2014.

[68] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the Trade*. London, UK, UK: Springer-Verlag, pp. 55–69, 2012.