T.C.

ALTINBAS UNIVERSITY

(ELECTRICAL AND COMPUTER ENGINEERING)

CYBER ATTACK DETECTION IN REMOTE TERMINAL UNITS OF SCADA SYSTEM

Ali Hasan DAKHEEL

Master Thesis

Supervisor: Prof. Dr. Osman N. UCAN

Istanbul, (2019)

CYBER ATTACK DETECTION IN REMOTE TERMINAL UNITS OF SCADA SYSTEM

by

Ali Hasan DAKHEEL

Electrical and Computer Engineering

Submitted to the Graduate School of Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

ALTINBAS UNIVERSITY

2019

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Osman Nuri UÇAN
Supervisor

Examining Committee Members (first name belongs to the chairperson of the jury and the second name belongs to supervisor)

Prof. Dr. Osman Nuri UÇAN	School of Engineering and Natrual Sciences, Altinbas University	_____
Assoc. Prof. Dr. Oğuz BAYAT	School of Engineering and Natrual Sciences, Altinbas University	_____
Asst. Prof. Dr. Adil Deniz DURU	Physical Education and Sport, Marmara University	_____

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Çağatay AYDIN
Head of Department

Approval Date of Graduate School of Science and Engineering: ____/____/____

Assoc. Prof. Dr. Oğuz BAYAT
Director

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Ali Hasan Dakheel

ACKNOWLEDGEMENTS

Above all, great thanks should go to the Almighty Allah for his support, guidance and help

I would like to express my deep gratitude, appreciation and thanks to my supervisor Prof.Dr.Osman Nuri Ucan , for his help, support and for providing me with a lot of up-to-date references .In fact, without his guidance and suggestions, this work would have suffered a lot.

My thanks should go to the head of Electrical and computer Engineering department for his support and help. I am also indebted to the teaching staff members in the department of Electrical and computer Engineering/ Altinbas University. I owe them much more than I can express in words.

I would like to express my , respect and gratitude to my parents , for their advice , fatherly and motherly care and encouragement. It is also through them that my dreams have come true. My love and gratitude are also due to my dear fiancée , brothers and sisters whose devoted encouragement , advice and care made this work possible.

ABSTRACT

CYBER ATTACK DETECTION IN REMOTE TERMINAL UNITS OF SCADA SYSTEM

DAKHEEL, Ali Hasan
M.S, Electrical and Computer Engineering, Altınbaş University,
Supervisor: Prof. Dr. Osman Nuri Uçan
Date: March / 2019
Pages: 101

Cyber security threats are becoming more and more prominent and the connection links at the Internet Service Provider (ISP) are getting faster. These cyber intrusion detection networks are not able to cope with the high increase of network security threats, as the majority of them are designed to detect specific intrusions that are previously known, or trained with intrusion-free traffic. Also, most of them rely on payload inspection, which can create a bottleneck in real-time detection. Furthermore, now that frequently every communication is done through encrypted messages, it is almost impossible to interpret the observed payload. In order to counter these limitations, we propose a Cyber Attack Detection that detects malicious hosts by using flows of RTU in SCADA systems and machine learning, that detects a specific observation in a given standard time period. This work will therefore present a proposal for a machine learning-based Cyber Attack Detection Systems (CADS) capable of detecting malicious hosts without any a priori knowledge. The results from this proposed system were validated with a realistic text files with malicious data provided by the network operators online. Using SCADA systems, unauthorized access to network and switches could be more tightly controlled while keeping a human in the loop; that is, human supervision and interaction were, and still are, part of SCADA systems. However, technological advances and the maturation of SCADA systems has pushed more of the supervisory function onto the computer systems that make up modern SCADA systems. In the early development of SCADA systems attention was

given to physical security, but virtually no attention was given to electronic or cyber security. The systems were obscure and the skills and technology needed to interact with the systems and update network security were simply not readily available; security of this type is often referred to as security through obscurity. This pattern has continued and today, most dedicated SCADA applications have not included built-in security. Unfortunately, open protocols, advanced telecommunication networks, cheap computer electronics, and unlimited access to even the most obscure information through the World Wide Web have made SCADA's security through obscurity up to date.

Keywords: Cyber Attack Detection Systems, Network Flows, Machine Learning, Intrusion based systems, Remote Terminal Units.

TABLE OF CONTENTS

LIST OF TABLES

Pages

LIST OF FIGURES

Pages

LIST OF ABBREVIATIONS

BSS	:	Between Sum of Squares
CAD	:	Cyber Attack Detection
DNS	:	Domain Name System
DoS	:	Denial of Service
DDoS	:	Distributed Denial of Service
DPI	:	Deep Packet Inspection
ED	:	Euclidean Distance
HDFS	:	Hadoop Distributed File System
HMI	:	Human Machine Interface
HTTP	:	Hyper Text Transfer Protocol
ICMP	:	Internet Control Message Protocol
IDS	:	Intrusion Detection Systems
IEEE	:	Institute of Electrical and Electronics Engineers
IP	:	Internet Protocol
IRC	:	Internet Relay Chat
ISP	:	Internet Service Provider
K-NN	:	k-Nearest Neighbors
LR	:	Local to Remote
LL	:	Local to Local
MPLS	:	Multi-Protocol Label Switching
MTU	:	Master Terminal Unit
NIDS	:	Network Intrusion Detection System
PCA	:	Principal Component Analysis
PoVE	:	Percentage of Variance Explained
RBF	:	Radial Basis Function
RTU	:	Remote Terminal Units
SCTP	:	Stream Control Transmission Protocol
SCADA	:	Supervisory control and data acquisition
SMTP	:	Simple Mail Transfer Protocol
SSH	:	Secure Shell

SVM	:	Support Vector Machines
TCP	:	Transfer Control Protocol
TSS	:	Total Sum of Squares
UDP	:	User Datagram Protocol

1. INTRODUCTION

This section will start by providing some background on how traditional Cyber Attack detection systems (CADS) operate and which drawbacks they present, followed by a proposal to counter such issues. Before moving onto the next chapter, there will also be a roadmap to the remainder of the document.

1.1 PROBLEM STATEMENT

Observing the present conditions of Cyber Attacks. We aim to extinguish the fire before it spreads. Detecting Cyber Attack contents accurately would fulfill our main objective. We will use various machine learning approaches to see which performs better for detecting Cyber Attack contents.

Preserving the security of a network is becoming increasingly important nowadays. The number of security threats is growing day by day, and the networks' security systems must be able to keep up with this. Moreover, Internet Service Providers are increasing the capacity of their backbone links, now operating in the order of 1 to 10 Gbps. Traditional Cyber Attack Detection Systems (CADS) usually operate by using Deep Packet Inspection (DPI) methods. This would be feasible for link connections that were rather slow, but now it is very difficult to analyze the payload of every packet passing through the routers and be able to process them in real-time, without creating a bottleneck. Also, nowadays most of the traffic payload is encrypted, which makes this kind of detection even harder.

There are basically two main approaches when it comes to classifying a CAD systems: 1) Signature-based detection; 2) Behavior-based detection. In the former, the system looks for a certain sequence of bits in the observed traffic in order to match it with a known intrusion.

1.2 MOTIVATION

Cyber Attack causes severe depression and sometimes even leads the victim to disrupt activities. It increases crime and violence and deteriorates the victim system wholly. But it's too difficult to track Cyber Attack in the vast online platforms manually and in remote terminal units of SCADA systems. So, we are profoundly motivated to detect these vicious

crime with an automatic system that enables to monitor the online platforms on a larger scale. The outcome will eventually assist the organizations in the maintenance of cyber-security on large multi-national platform and mostly on industrial level to reduce the risk of failure in all electric management system, water management system and gas-pipeline system using remote terminal units in SCADA systems which will detect any Cyber Attack in future.

1.3 SCOPE

Detecting Cyber Attack is always challenging. Several problems has to be faced with the dataset, algorithm, building e perfect model, accuracy level of result etc. And gas pipeline industry security concerns and terms, policies are volatile. Our purpose is to surpass those challenges with a proficient and effective system with the following features:

- High Detection Rate: Detection rate must be very much high otherwise it will be of no use. We are going to build SVM model with the capability of detecting Cyber Attack with maximum accuracy level.
- Reducing unusual Act: As per research, many case studies have been done on the impact of Cyber Attack. One of the severe impacts was the rate of teenagers system hacked because of being Cyber Attacked. If system can highly be skilled that people won't attack one at least in fear of being detected, this rate can be reduced. We want to build this kind of strong system.
- False Identification of Cyber Attack: This means if a word is a bully. But still a system detecting it as a non-attack one. This happens when the system trains more non-attack word than attack. This kind of false positive detection can lessen the possibility of detecting real Cyber Attack. So, we will try our best to look into this issue and prevent the amount of false positive detection rate.
- Reduced Performance and computation time: The system has to deal with a vast dataset. So, saving real processing time is a big deal and important. We want to build a model with optimal performance and processing time.
- Getting Direct Resultant classes: Some model gives the probability as result. But in gas pipeline and related industry, it is very much significant to get the direct result of Cyber Attack in gas pipeline to get the desired properties plus attributes of the gas pipeline

industry. Probability is sort of ambiguous. So, we are working with SVM which gives direct resultant classes.

1.4 OBJECTIVES

The main goal of this dissertation is to present a solution that improves these limitations. Such solution features a combination of the two major machine learning techniques:

- Unsupervised Learning
- Supervised learning

The difference between these two is that the former does not need ground truth, i.e. it does not require any labeling of the data: the machine is able to provide results based on the raw data alone; the latter, on the other hand, does require labeling of the data, as the algorithm will take as input a set of training data that will teach it to make the right labeling to the remainder of the data it will receive from there on.

With this dissertation, we aim to present a flow-based Cyber Attack Detection System CADS that is able to overcome the drawbacks mentioned in Section. This means creating a hybrid CADS operating on a flow-level - rather than packet-level, and that is able to detect intrusions without a priori knowledge. This is done by combining the two techniques mentioned above. By achieving this, we have a system that provides increased autonomy and is, in a way, self-learning.

This approach will allow the detection of the source of the attacks, rather than to identify which kind of attack it is, i.e. it detects the devices (be them desktops, laptops, mobile phones or compromised servers) that are behaving maliciously. Also, this system focuses on detecting volumetric attacks, i.e. it allows to unveil relevant patterns in large scale and intensity network attacks, therefore not allowing to discover attacks such as Buffer Overflows, SQL Injections, Phishing, Cross Site Scripting, and so on.

To achieve so, the first step consists in preprocessing the incoming flows, by mining a set of features from the raw dataset, therefore eliminating features that are not needed, such as its payload. Upon this filtering, the flows will be aggregated in order to give the program a wider and more granular view of what is happening, as it will be explained further ahead. Having the flows preprocessed and aggregated, the two machine learning techniques that were previously mentioned will come into play. For the unsupervised learning, a clustering

algorithm will be used, which will group together flows that share similar patterns. Having the clusters been generated, there will be a need for an expert intervention, in order to label the clusters. This is necessary for the next step, which applies supervised learning, in which Support Vector Machines (SVM) will be used. The SVM will then receive the labeled flows, and will be able to indicate whether the input flows are perceived by the system as being malicious, or benign behavior, therefore highlighting the intrusions present in system.

1.5 THESIS ORGINAZATION

This thesis describes the research and work developed and it is organized as follows:

Chapter 1: Aims to explore relevant studies on this field.

Chapter 2: Provides the backbone structure, requirements and the solution's architecture.

Chapter 3: Describes the implementation of the solution as well as the technologies involved in its development.

Chapter 4: Describes the evaluation tests performed and the corresponding results.

Chapter 5: Describes the developed work, and result analysis with result evaluation.

Chapter 6: Summarizes the developed work, as well as conducts the comparison of this work with all work done previously.

Chapter 7: Conclude the thesis as well as future work prediction in this domain.

2. BACKGROUND

This section provides an overview of same major contributions in this area. Section 2.2 provide an insight on some of the existing tools to perform flow analysis. Section 2.3 and 2.4 proceeds to point out some network monitoring applications that were built in a flow-based fashion. Section gives a full view of some of the most addressed network intrusions and respective works that show how to detect them using a flow-level analysis rather than payload inspection. At last, Section 2.5 and 2.6 gives a brief explanation of what RTU in SCADA systems and section 2.9 describes a machine learning is and how it can be used to achieve our goal.

2.1 NETWORK FLOWS AND BASIC FLOW TOOLS

As stated previously, network flows allow a different approach in analyzing, monitoring and securing a network. While deep packet inspection allows for signature-based approaches, making it easier to detect some kinds of attacks, it is not scalable for high speed networks, e.g. Gbit/s [3] - the packets' payloads cannot be analyzed in real time, for such speeds. Also, nowadays most packets exchanged have their payload encrypted, making it even more difficult to inspect, even if it was possible to process that many packets in real time.

In Cisco developed the first network protocol to handle network flows: NetFlow [4]. This consists in a built-in software in their routers, and is used to collect and export flow records. As the years went by, new versions arrived and were more and more complete. The most recent version - NetFlow v - includes integration with protocols such as Multi-Protocol Label Switching (MPLS) that were not supported in the previous versions.

Before we proceed to detail this protocol, first we must explain what a flow is. A flow is defined as being a unidirectional sequence of packets, passing through an observation point that satisfies a set of common features in a given period of time [7]; in this case, an observation point would be a NetFlow-enabled router. These features must be defined a priori, in order for the device to perform the matching of features (an example of such features will be presented further ahead).

Being this technology built-in in network devices, it allows to select, from all the traffic passing through that device, what we want really to analyze. For example, by deploying this in a border router, all of the traffic going in and out of that network will be filtered by NetFlow.

Upon the reception of an IP packet, the network device looks at that packet's fields in order to find any matching feature with those previously defined. In case the packet's features do match, then an entry is created in a data structure called flow cache, for that flow. Note that a flow may correspond to several packets, and many different flows can be collected. This process is depicted in Figure 2.1.

As this cache cannot be kept indefinitely, certain policies were defined for its management. According to those policies, when one of them is satisfied, the entries belonging to that cache will be exported to another device. So, a packet will be exported when: (i) the end of a flow is detected; (ii) a flow is inactive (i.e. when, for a given timeout, there are no longer packets belonging to a flow); (iii) belonging to a long-lasting flow, the timeout is reached; (iv) the device is in need of resources, e.g. internal memory.

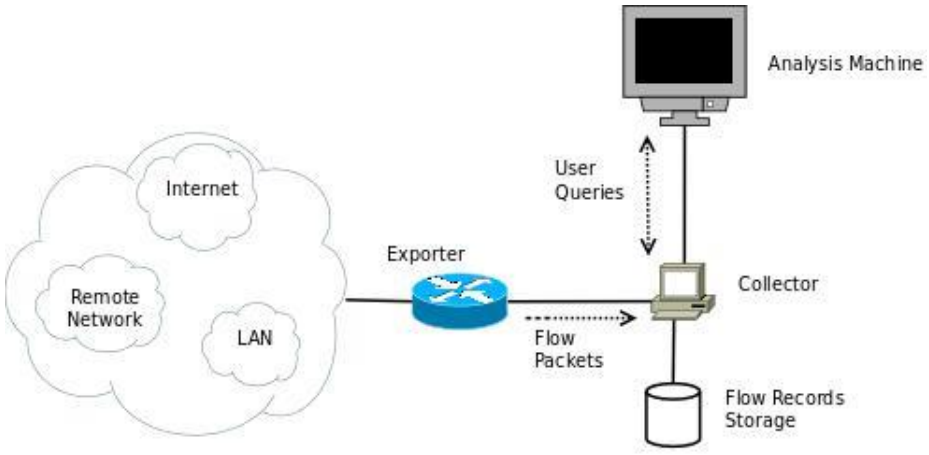

Figure 2.1: A typical NetFlow architecture

The device to which the flow records are exported is referred to as the Collector, and is physically located in another device. In order to achieve maximum efficiency latency-wise, these records are encapsulated in a UDP datagram. However, NetFlow also provides the possibility of exporting the data through SCTP (Stream Control Transport Protocol), if we want to operate in a congestion-aware environment.

However, this technology initially did not have in consideration any security issue - no confidentiality, integrity or authentication is guaranteed. This was designed as such due to efficiency and scalability issues - the deployment of such technology in large networks would not be able to provide real time measures.

2.2 SCADA SYSTEMS

Supervisory Control and Data Acquisition (SCADA) came into existence in the mid 1960's coinciding with the development of the minicomputer. SCADA provides a means for remotely monitoring and controlling many kinds of industrial systems by providing users of the system with the ability to remotely control one or more specific devices and to monitor the performance of those devices from a central and physically remote location. The IEEE std C37.1-1994 [7] defines SCADA to be:

"A system operating with coded signals over communication channels so as to provide control of RTU equipment. The supervisory system may be combined with a data acquisition system by adding the use of coded signals over communication channels to acquire information about the status of the RTU equipment for display or for recording functions." [7].

An excellent example of such a SCADA system is the distribution system used by electric utilities, which is one of the oldest and most familiar SCADA systems. In electricity distribution SCADA is used to collect information from remote parts of a power distribution grid; for example the volts, amps or phase angle of a particular line in a substation, and provide it to a central control installation. In addition, SCADA allows an operator at the centralized control station to trip breakers at remote substations in response to conditions reported by the SCADA system. Other well-known industries that use SCADA are the gas and oil utilities and nuclear power production.

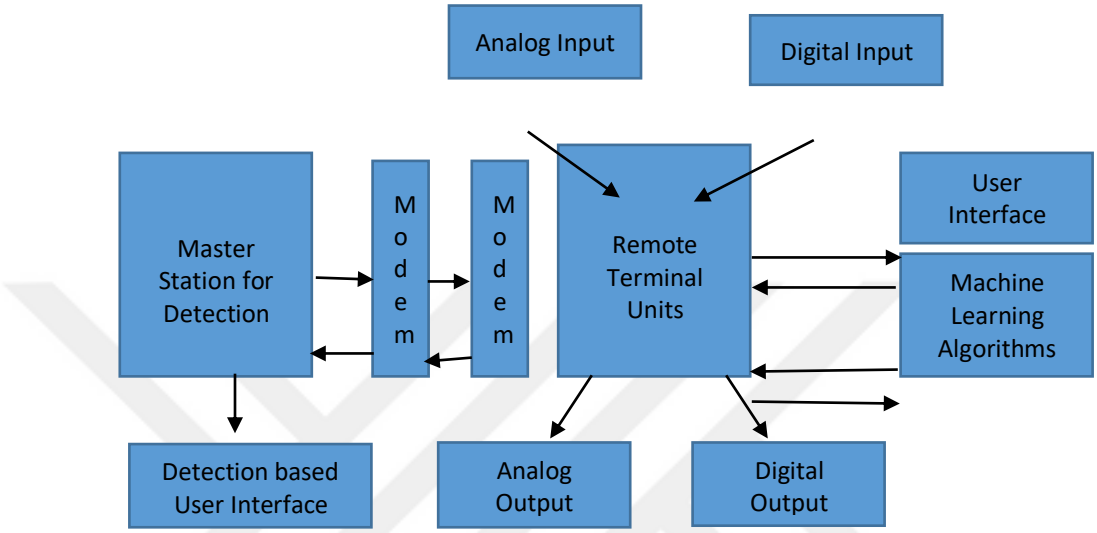

Figure 2.2: Typical Supervisory System using Machine Learning.

2.2.1 SCADA Components

There are four main components that make up a SCADA systems: the supervisory system or master terminal unit (MTU), remote terminal units (RTU), a communications network, and field instruments or devices [8-10]. The exact nature of the different components depends greatly on the specific SCADA system and its topology. A typical supervisory system is shown in figure 2.2 and each subsystem is explained in detail in the following paragraphs. A small SCADA system might consist of only one MTU and one RTU, and is referred to as single-master, single-remote [11]. A more common configuration is the single-master, multiple-remote system with a single MTU connected to many RTUs. In large SCADA systems it possible to have multiple MTUs and hundreds of RTUs [4]. 8 Master station (MTU) The master station or master terminal unit (MTU) has traditionally been located in a control room where human operators interact with the system through a user interface (UI). The MTU is responsible for polling remote devices for data, processing the data, providing various representations of the data (including alarms) and sending operator initiated control signals back to the field devices. In some situations the UI is carried out by a separate system called a HMI (human machine interface) system. The HMI

system provides an interface between an operator and the MTU, freeing up the MTU from providing a UI. In this case the MTU continues to carryout polling and control activities, but the high level representation is left to the HMI machine.

2.2.2 SCADA Architectures

As computer and network technology have evolved and matured, so have SCADA systems. The evolution of SCADA systems is generally broken down into three separate successive generations [14-15] monolithic, distributed, and networked. The changing architecture of SCADA systems has been a contributing factor to the cyber security issues faced by modem SCADA systems.

- First Generation.
- Second Generation
- Third Generation.

2.2.3 SCADA protocols

At the heart of SCADA networks are SCADA protocols. These provide the template for communication between SCADA components, typically between the MTU and the RTU. Early SCADA systems, the first and second generation SCADA architectures discussed previously, used proprietary protocols, but in more recent years there has been a move to open standards in SCADA protocols. RTUs are connected to MTUs by a variety of different communication channels and both the cost and availability of the communication channels has affected protocol design [17]. The limited bandwidth of early communication channels resulted m a very compact message format, supporting only the most basic information needed to achieve RTU to MTU communication. Figure 2.3 shows the structure of the basic SCADA message format. The four bit RTU address allows multiple RTUs to share a single communication channel, rather than requiring a separate communication channel for each 15 RTU. The eight bit function code specifies what operation is to be performed by the RTU. The bits following the function code are an addressing scheme that indicates the set point, control point, or data on which the operation is to be carried out. This address has no special meaning to the RTU, and it is up to the

MTU and SCADA software to correctly associate an RTU address with the real world value it represents.

Figure 2.3: Typical SCADA protocol message format, adapted from [17].

2.3 SCADA CYBER SECURITY THREATS AND VULNERABILITIES

The primary cyber based threat to SCADA systems is that an unauthorized person or agent will access the SCADA system and interfere with its operation. "Entry into the substation [RTU] via different lines or other industrial based networks for the manipulation or disturbance of electronic devices. These devices include digital relays, fault recorders, equipment diagnostic packages, automation equipment, computers, PIC, and communication interfaces." [20]. 16 Typical attack scenarios like those described in [21-22] center around an attacker making changes to control settings, physical device parameters, or sending control commands directly to field devices. These attacks would result in a malfunctioning of the SCADA system which might cause a disruption in service, or possibly environmental damage or loss of human life. These threats might be carried out by a number of potential threat agents, including hostile nation states, industrial spies, disgruntled employees, and malicious hackers.

2.4 DIFFERENCE BETWEEN SCADA AND TRADITIONAL IT

Though SCADA systems are increasingly adopting technologies from traditional IT environments, SCADA and traditional IT systems are very different in several ways. One of the most important differences is how security is prioritized. In traditional IT systems, security engineers usually consider confidentiality the most important followed by integrity and then availability. However, as discussed by Miller in [30], for control systems

availability is most important, followed by integrity and then confidentiality. For example, when you switch on a light, it needs to come on; when you pick up the phone there should be a dial tone. This is availability; it is what we expect from the systems and services that make use of SCADA. Moreover, down time for the services that SCADA systems operate can run into the millions of dollars per hour [30], making availability of paramount importance.

SCADA systems and other control system also tend to have very different performance needs from traditional IT systems. Delaying the delivery of information even for a relatively brief moment is not acceptable in SCADA systems, though they often do not require a high degree of throughput. However, IT systems typically do require a high throughput but are much more tolerant of delays or jitter. In addition, many SCADA systems may have much greater resource constraints than would be found in traditional IT systems. This lack of computing resources along with performance constraints can make it difficult or impossible to apply standard security technologies.

2.5 SCADA RESEARCH CHALLENGES AND CURRENT RESEARCH

Over the past several years' industry groups and academics have begun to work towards addressing the SCADA security issue. This can be seen in the increasing number of publications related to SCADA security identifies three research challenges in the field of SCADA security. Solutions to these challenges must take into consideration the unique demands of SCADA systems.

One of the primary security tools is encryption, and there are several articles which present SCADA security solutions that deal with encryption. Leading the way in SCADA and encryption, particularly for legacy systems is the AGA 12 working group established in 2001 by the American Gas Association (AGA). The working group was to recommend solutions to that would help protect gas utility SCADA equipment from Cyber Attack. The group determined that unprotected serial based communication channels posed the greatest threat. In response to this threat AGA has developed a serial SCADA protection protocol (SSPP) which is implemented by a separate device called SCADA Cryptographic Modules (SCM); these are installed on either end of a communication channel [5-13]. Figure shows

the proposed architecture of using two SCMs to provide encrypted communications between an MTU (SCADA Host) and an RTU.

2.6 RTU CONSTRAINTS

The previous sections have described and identified additional criteria on which access control decision should be based, and defined three types of constraints to add to the RTU access control model that reflect these additional access control factors. The final RTU access control model can be defined. It incorporates the constraints, and unites together the different access control factors. The factors identified in section 2.5 can be incorporated into the RTU access control model using constraints. Recall from chapter two that there are several different possible types of constraints that the model might choose to express. Separation of duty constraints have been among the most popular, but chapter two identified seven different types or categories of constraints [44]. The RTU access control model can support access control decisions based on the factors identified in section 2.5 by using exogenous constraints. Exogenous constraints are constraints whose attributes are not a part of the core RBAC model, and were defined in chapter two. The constraints will affect the relations in the model, specifically the UA and PA relations. Since these attributes will change during the runtime execution of the RTU, they are dynamic constraints. We incorporate into the model three types of constraints, role activation constraints, permission activation constraints, and point type constraints.

Instead, it assumed that the exporters (and also collectors) are deployed within a private, restricted and controlled network, rather than a public network, in which anyone could sniff these records, or even forge them.

Apart from NetFlow in RTU, many other vendors have their own implementation for flow collection and exporting. Examples of such implementations are NetFlow-lite, sFlow, NetStream, etc.

Due to the heterogeneous nature of these technologies from each of the vendors existing in this market, the Internet Engineering Task Force (IETF) joined forces to create a standard in flow collection and exportation, thus allowing for the clients to easily deploy their flow-based applications. This protocol was given the name RTU (Remote Terminal Units) [21].As previously stated, packets which share common properties are grouped in flows,

and in the IPFIX terminology they are referred to as flow keys and these can be, for example, a tuple such as:

(IP_source, IP_destination, port_source, port_destination, typeOfService)

In NetFlow, the flow exportation was done by encapsulating the records into UDP or SCTP datagrams. RTU provides, in addition to these protocols, the option of exporting flow records through TCP. Also, it fills the security gap in Cisco's implementation: RTU provides both confidentiality, integrity, and authentication. These three properties are guaranteed by using secure channels - either through TLS (Transport Layer Security), if the records are exported through TCP, or through DTLS (Datagram Transport Layer Security) if the records are to be exported through UDP or SCTP datagrams. Furthermore, X. Certificates are also used, in order to reinforce the latter property.

Then, they used a simple single queuing model in order to study to effects on the flow monitoring application. This queue operates in a FIFO fashion, and each job represents a flow packet, which, in its turn, only RTU carries the number of normal and attack flow records. They point out to the fact that if such a system is not sufficiently well dimensioned, such an attack could easily overload it: if the service rate μ is much smaller than the arrival rate λ, the systems is easily flooded. Furthermore, they showed that the effective rate of processed flows is a function of the rate arrival rate of attack records, and not only the normal ones. This means that during an attack, the normal flow traffic is also affected, which means that, for our case, the development of a CADS, it might be interesting to also analyze the "normal" traffic in order to detect suspicious activity.

2.7 CYBER ATTACK DETECTION BASED ON NETWORK FLOWS

Nowadays, there are a vast variety of network attacks. From simple port or network scans to complex botnet infrastructures, there are numerous types of different attacks, both in type, in scale or severity of impact. However, as the variety of the attacks is indeed enormous, we cannot focus on the detection of all of them. Moreover, a flow-based intrusion detection approach is not able to detect all kinds of attacks, as it relies on the inspection of header information. Logic attacks such as Buffer Overflows or SQL injections can only be detected by inspecting the payload of the packets in the network, which represents a major limitation if one is to use an exclusively flow-based CADS.

With the help of some of the technologies and tools referred in Section 2.6, we present some approaches on the detection of these attacks, namely Port Scans, Denial of Service, Worms and Botnets.

2.7.1 Flow-based Network Monitoring

Amongst many other applications, such as the monitoring of applications, hosts, security, account and billing, the analysis of network flows has been widely deployed for network monitoring. In this subsection we refer some of the existing work done in this area [18], presented a flow-based network monitoring system called FACT - Flow-based Approach for Connectivity Tracking. Their goal was to deliver a monitoring system focused on remote hosts and networks, and to check if they are reachable from inside their (network operators) network or a costumer's network, and to trigger an alarm in case there are any kinds of connectivity problems. Such problems could be for example an unusually high number of outgoing connections from the inside of the network to remote host (which could indicate an ongoing port scan).

2.7.2 Port Scanning

A port scan is defined as the act of consistently probing a target host (which may be either a single machine or an entire network - network scan), by sending a large amount of generally small packets.

Although not being considered an attack, it is generally the first step of almost every network attack (such is the case in DoS/DDoS, worms and botnets), thus being its detection a crucial step for a NIDS. As the attacker sends a great number of packets, even though these may be very small, it will produce many flows, therefore making it possible to rely on flow-based approach to detect it, as it can be very easily addressed [16]. Divides this attack in three distinct categories:

- Horizontal scan, in which a single host scans multiple ports in a single machine.
- Vertical scan, in which a single host scans one single port in multiple machines.
- Block scan, a combination of the two scans above - a single host scanning multiple ports in multiple machines.

Whatever category the attack falls in, this may create an anomaly in the normal network traffic pattern, and many kinds of different flows can be observed.

Most of these attacks are investigated by observing a flow characteristic that registers the most significant difference when compared to normal traffic: the unusually high number of incoming and outgoing connections in a host. This is due to the fact that the attacker, or attackers, probe many different ports and machines, therefore generating an anomalous amount of new flows.

A type of attack that falls into this category is the brute-force SSH network attack [15]. This is a particularly interesting attack in this field of study. The attack consists in three phases (that will be explained better further ahead): it begins by consistently scanning a certain number of victims, until a running port is found; then, it attempts to login in those victims through a brute-force dictionary attack; once it gains access, the attacker can do whatever he wants with the victim, and also to others that might belong to the same network. Therefore, SSH attacks can be potential harmful not only to the host individually, but also to the network it is connected.

However, the detection of these attacks, as they rely mainly on scanning, can be address by performing an analysis of the network traffic at a flow-level. [19]

Their solution was based on the observation made by [14-16]. They observed that the behavior of the attacks over time, in terms of flows, follows a pattern of evolution, and it can be identified in three distinct phases, as described below:

- Scanning phase, in which the attacker performs a port scan for a certain IP address block.
- Brute-force phase, in which the attackers tries to login to a certain number of hosts, by means of a dictionary attack - various combinations of usernames and passwords
- Die-off phase, in which after successfully login into the victim host, the traffic volume is drastically reduced, leaving only residual traffic

By tracking these three phases in a flow pattern, this attack can easily be identified. Moreover [22], also proposed a solution to detect this intrusion by, once again, observing flow-based traffic patterns. This approach will be explained in more detail in the next section.

2.7.3 Denial of Service

A Denial of Service (DoS), is the attempt of an attacker to make a certain server, firewall, or even a whole network unable to reply, by flooding it with several requests and thus draining all of its resources. In a DoS attack there is generally one single attacker. This has some limitation to the attacker, such as the fact that it may not scale when targeting a big number of victims, or even the fact if one single attacker is producing all the traffic, it is easier to detect, due to the large anomalous volume of traffic. Another kind of DoS attacks is the Distributed Denial of Service, in which now there as various sources that are producing this large volume of traffic, thus making it more difficult to detect, as the traffic is now more evenly distributed and hidden (this was already discussed in Section 2.6).
According to Peng et al. [23] and Kim et al. [24], there can be various types of DoS/DDoS attacks:

- SYN flood - this attack aims to explore the vulnerability in the three-way handshake, the connection establishment procedure in the TCP protocol. As depicted in Figure 2.1, to establish a TCP session, the one initiating the connection (the client) will send a packet with an activate SYN flag, to which the entity on the other end (the server) replies with a SYN/ACK flagged packet
- ICMP flood - this attack is labeled as a bandwidth attack, as its goal is to congest the communication channels in a network. When a broadcast message in sent, there will be as many packets being sent in the network as the number of connected hosts; by sending multiple broadcast messages, the bandwidth can easily be depleted
- Smurf - this attack consists in sending an ICMP Echo Request packet whose destination source is the broadcast address of a network. Also, the source address is external to the compromised network and is propagated with the use of intermediates (who can also be victims in this attack)
- Fraggle - this attack is performed like the Smurf attacks, except now the UDP protocol is used, instead of ICMP
- Ping of Death - this attack consists in sending multiple (e.g. thousands per second) spoofed pings with an abnormally large size

- Land - this attack is performed by sending packets whose .This way, the victim continuously reply to themselves, resulting in a Local Area Network Denial (hence the name of the attack)
- Ping Pong - this attack consists in sending packets whose both destination and source ports are reflecting ports, which are used for services such as echo, in UDP

Apart from these, nowadays there are more variations of the Distributed Denial of Service, namely the DDoS HTTP Flood and the DDoS DNS Amplification attacks. The former, such as the SYN Flood attack, makes use of a vulnerability in the protocol used. Located in the network's Layer (the Application Layer), the attacker send an HTTP request to the server.

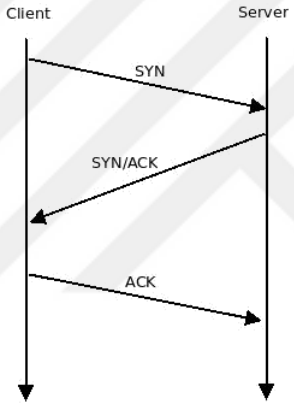

Figure 2.4: TCP Three-way handshake

It may be a POST or GET message, aiming to make the server reply with messages having the maximum resources possible, in order to drain its resources, e.g. by sending requests of downloads of very large files. If this attack is well coordinated, by sending these multiple request from multiple different users, the server will exhaust its resources, not being able to respond to legitimate requests. The latter makes explore a vulnerability in the Domain Name System (DNS) protocol behavior. The attacker perpetrates his attacks by sending a spoofed Source IP address - being this address the IP address of the victim - to various DNS servers, and the query contains an Internet domain that has many different DNS records for the same name. This way, the DNS servers will all respond these multiple records to the victim, therefore flooding it with a large volume of traffic. However, for the best of our efforts, no literature was found where these attacks were detected using flow-based approaches.

An interesting work presented a system for detecting abnormal traffic in a network, using a flow-based approach, extracting the traditional -tuple from the flow extractor [25]. The authors define the term abnormal as traffic that induces a malicious purpose, as is the case of the traffic generated by DoS/DDoS, worms or even scanning. Moreover, this work is particularly interesting as they focus their effort on detecting DoS/DDoS attacks, as we will be describing below.

Whenever one of these two modules detects an attack, an alarm is triggered, indicating the network manager that an attack might be occurring, very much like a signature-based NIDS would operate. The first module is in charge of inspecting the fields of the gathered flow headers. This allows the detection of some flooding attacks that possess specific values, e.g. having Broadcast as the IP destination address. Table presents an extensive list of such values that will be validated and the respective attack that they might point out to. When validating the protocol fields, the detection is pretty straightforward, needing only to inspect the corresponding.

Table 2.1: Summary of fields in flow headers that can trigger attacks [22].

Protocol	**Fields**	**Attack**
ICMP	Echo Request + Destination IP = Broadcast	Smurf
	Large Flow size or Packet Count	Ping of Death
	Large Packet Count and Flow Size	ICMP Flooding
TCP	Source IP address = Destination IP address Or Source Port = Destination Port	Land
	Large Packet Count and Flow Size	TCP Flooding
UDP	Destination/Source Port = Reflecting Port	Ping-Pong
	Destination IP address = Broadcast And Destination Port = Reflecting Port	Fraggle
	Large Packet Count and Flow Size	UDP Flooding

Fields and check is they match or not, leading to a possible trigger. However, when referring to the packet count and flow size, the term large is rather ambiguous, as they depend on the volume of traffic at the time, and the kind of traffic that is being produced. And so, to address this, this value is obtained by calculating a percentile threshold of the observed values.

During a scan, an attacker makes a great amount of connection attempts, therefore generating many flows in which the packet count is small (approximately bytes). If this scanning falls in the category of port scanning, the attacker will sweep a great amount of ports in the target host, therefore generating traffic in which the destination IP address is constant. However, if it is a network scanning, the attacker will make many connections with many different destination IP addresses, searching for a service availability in one of them.

The need for a second detection module derives from the fact that some attacks (and consequently, their patterns) cannot be detected only by inspecting the flow headers fields and analyzing flows individually. In order to detect these, they need the traffic information capable of identifying patterns in it, and this can be achieved by aggregating related flows. From these aggregations, it is now possible to detect both flooding and scanning attacks (not only port scans, but also network scans), by checking the information sent and received from a host).

Another approach for detecting DoS attacks was done using D sketches. Not only this detects such attacks (their main focus relies on TCP SYN flooding detection), but also can cope with the detection of the various types of port scans [16-18]. A sketch is hash table used to quickly store information, mainly counting the occurrences of a given event. When this concept first came up, these hash tables had only one dimension, but the work presented by [22-24] introduces the use of a two-dimensional table. With this data structure it is possible to Characterized the observed traffic, in order to draw some conclusions, which in this case is the detection of port scans and DoS attacks.

As it was just mentioned, this system is able to detect both port scans (which is also extensible to detect most of large-scale worm propagations) and TCP SYN flooding attacks, using sketches as a base for statistical intrusion detection. To achieve so each of the network routers (may be either edge or backbone routers) are configured to record network traffic into sketches. From the multiple created sketches (one for each router), they

summarize all of them into one aggregated sketch, therefore allowing to distinguish from many different attacks observed. Then, they apply time series algorithms on these in order to obtain a forecast sketch, which will be used for the change detection. By subtracting this forecast sketch to the current one (i.e. in the present time), it is obtained a forecast error, which, if sufficiently large, indicates that there is an anomaly. By reversing the sketches, they obtain the key characteristics of the recorded flows, allowing to mitigate the attacks. In case of the values recorded trespass a certain defined threshold, the system triggers an alarm, indicating an attack.

2.7.4 Worms

A worm consists on a harmful software that, unlike the well-known case of a virus, has the capability to autonomously explore software vulnerabilities, thus making it capable of replicating itself throughout a network.

This specific attack is usually divided in two distinct phases: (i) a scanning phase, in which the worms probes several machines in order to find a vulnerability and then proceed to spread the infection; (ii) the transfer phase, in which the worm proceeds to send the harmful code and infect the victim. As the first phase is a well know case (as described above, in the port scanning phase), the discovery of the scan can be crucial in identifying the attacker. Due to the fact that most of the network traffic relies on secure connections, and therefore the payload of traveling packets in most of the times is encrypted (e.g. TCP traffic), a major emphasis is given to the detection of the first phase, as is would be very difficult to detect malicious content of packets in a flow-based analysis.

In that research, only four protocols were considered:

- HTTP, identified by observing a connection in which one of the peers uses port (either destination or source); this represents the majority of the traffic observed
- SMTP, identified by observing connections where one of the peers uses port ; this is the second most active protocol producing traffic in the network
- FTP, which can be identified by observing the usage of port by one of the entities in a connection

- Oracle, identified when one of the peers is using port. As this protocol requires a login and password to authenticate an user, it is expected that users connected to a smaller number of servers

In order to construct the graph, flow records were extracted from NetFlow-enabled routers, in a large intercontinental network. These records were collected throughout a period of days. As stated by the authors, many worm detection systems on the detection of an unusual high number of frequent connections between peers, and track them by inspecting connection evidences such as half-open TCP connections. So, to avoid this issue, the attackers can use hit lists, which consists in a list of previously identified servers. Using this, they need not to contact random servers across the network and can focus on these known ones, making it harder for the systems to detect this scanning phase.

The detection model consists in the hypothesis that an attacker that contacts different servers through a hit list will affect the graph in two ways. The system will raise an alarm whenever one of two conditions are verified:

- The total size of the graph is bigger than its mean value in the observation period π along with its standard deviation over time.
- The largest component size is bigger than its mean value in the observation period π along with its standard deviation over time

2.7.5 Botnets

A botnet is a network of infected hosts (referred to as bots) that is controlled by an entity that is referred to as master or bot-master. The master controls its bots through a Command and Control C&C - infrastructure, and uses them as a third party to launch attacks and intrusions in the network, such as Spam, Phishing, DDoS and Identity Theft. This C&C infrastructure could either be an IRC (Internet Relay Chat) channel, or an HTTP server. However, by identifying it, most of the problem is solved [19].

Botnets are considered to the date as one of the biggest security threats, and the most difficult to detect, as the bot-master is not easy to identify, and requires a long period of observation.

When a computer is infected (note that this study refers only to Windows operating systems), the botnet begins by generating a .txt or .data file in the system directory. Also it

injects its code into other processes in order to hide its presence, and disables any anti-virus software and task manager, if needed, and may open one or more ports for further communication, establishing connections with bot-master or peers in order to launch its attacks. This run-time behavior is transformed into a behavior vector, consisting of-features.

All together are intrinsic to a botnet behavior. As the network analyzer is able to gather network-level information, the in-host analyzer need to focus on aspects cannot be observed externally, and also features that can, and so the network stack features. The suspicion-level generator uses a Support Vector Machine, a supervised machine learning algorithm to quantify the suspicion level. The training data fed to the system consists in both benign and malicious behavior profiles. Based on this training process, a hyper-plane is created, which will correspond the classification. When a new behavior vector arrives, the Support Vector Machine will compute its distance from the hyper-plane, and decide whether to classify it as a benign or a malicious behavior.

After all this data is extracted both from the network analyzer and the in-host analyzer, the correlation engine performs a last analysis to check if a botnet is or not present. When a group of hosts are clustered, the respective host analyzers are requested the suspicion-level together with some network statistic. There are two possible outcomes from this step:

- The network statistics sent from the in-host analyzer may differ from those observed by the network analyzer, implying that the host if sending falsified data, therefore triggering immediately an alarm.
- If the results are consistent from both analysis, then the suspicion-level and clustering quality must be analyzed by the co-relator. This is done by a function with two parameters. The first is the suspicion-level, which itself is a quantitative measure. The second is the distance between the hosts presented in the same cluster, which is computed by a simple Euclidean Distance.

With these two modules, the authors managed to track, from real-world data, different kinks of botnets, achieving low false-positive and false-negative rates. However, it is made clear that flow-based CADS alone are not able to detect Botnets, as it is still necessary to gather information that is not provided by network flows.

2.8 SECURITY HARDENING RTU'S

SCADA systems in general. The traditional IT security approach of soft on the outside (peripheral systems) and hard in the middle (servers) is not appropriate for SCADA systems. In SCADA installations the peripheral devices, such as RTU, must be security hardened as well. The focus of this dissertation is to identify and develop hardening techniques for RTUs and to develop a security hardened RTU. As discussed in the previous chapter, in the past, these devices faced primarily physical threats, but today they are increasingly network enabled and network accessible. Security hardening these devices is a major challenge facing the development of secure SCADA systems. Two security hardening approaches are explored in this dissertation, an RTU role based access control model and a reduced kernel OS. Previous work on both role based access control constraints and minimal kernels for operating systems was presented in chapter two. This chapter introduces the architecture for a security hardened RTU.

2.9 CYBER ATTACK DETECTION BASED ON MACHINE LEARNING

An increasing trend in CADS is the use of machine learning techniques [25-27].
This knowledge may be refined and improved at each iteration, by learning from previous experiences and observations. Such method has been used in an enormous number of different applications, in many different fields of science, such as natural language processing, speech recognition, bioinformatics, spam detection, network intrusion, among many others.
Machine learning algorithms can be divided into two major fields:

- Supervised learning
- Unsupervised learning

The first one relies on a labeled training dataset. The data consists in an extensive list of input data that aims to train the system, making correspondences between keywords and their meaning or interpretation that is expected to the system. After this training phase, the system is ready to classify data based on the training set it was trained to. Examples of this method are the algorithms k-nearest neighbor and Support Vector Machine. While Supervised methods relies initially on the introduction of training data, Unsupervised methods only receives as input a feature vector without any kind of labeling, and is used to

means such as discovering similar groups within a data set. Clustering is an example of this kind of learning.

In the field of Network Cyber Attack, machine learning has been able to classify network traffic and identify both anomalous patterns and potentially harmful machines. When it comes to embed this technology in an Intrusion Detection Systems, generally this is the strategy adopted:

- Anomaly-based detection (or behavior-based), in which normal traffic patterns are differentiated from anomalous ones. It focuses its attention on finding patterns that would not be expected from the machine's behavior. Unlike misuse-based CADS, these patterns are unknown to the system.

These two make up the classical approaches, being that machine learning is applied to the anomaly-based approach, rather than the misuse-based one. However, there are some variants to these, as we will now be discussing. Eskin et al. were one of the first to address unsupervised learning in intrusion detection systems [23]. They present a new technique, which they entitle Unsupervised Anomaly Detection. This technique allows to train the system with a dataset of completely unlabeled data, providing the chance of detecting unknown attacks to the network, which would not be possible when training the system with labeled data - in this case, the system is only able to recognize those labeled intrusions; and also, the manual classification of data can be very hard and tiresome. All summed up, the authors considered a total of features. Also, the dataset was filtered, so that there would only exist a percentage of to 87 % of attacks vs. to 77% normal traffic instances [27]. This is done because of the need of the system to learn to distinguish intrusion instances from the normal ones, and the original dataset was composed mainly of intrusions.

Their solution was based on two assumptions:

- The number of normal instances greatly outnumbers from the number of intrusions.
- The intrusions themselves are qualitatively different from the normal instances.

These two assumptions are extremely important, as these will be the foundation for the intrusion detection approach.

The system clusters the collected data through an algorithm that computes a distance-based metric. However, because of the different distributions that each feature vector may have, these have to be normalized in order to apply the same metric to all of the vectors. After

computing the clustering algorithm, these new clusters can now be classified as being normal traffic instances or an intrusion [44]. The first assumption implies that small clusters correspond to the intrusion instances, as opposed to the bigger clusters that represent normal instances; the second assumption implies that normal and intrusion instances will not be under the same clusters because of their qualitative differences.

In order to measure the performance of the system, they used the following metrics:

- Detection rate, which represents that ratio of intrusions detected by the system, by the intrusions present in the dataset
- False positives, which is the ratio of the total number of intrusions that were incorrectly detected by the system, by the total number of normal instances (as defined by the author [41])

In this solution, there is an inevitable trade off between these two indicators, as one scales with the other. However, they managed to obtain a fairly reasonable ratio of detection rate and false positives.

This thesis presents mainly two advantages to the traditional Cyber Attack detection systems. The first, is that it does not require any kind of manual classification, and the second is that the system is capable of detect intrusion that were previously unknown.

Similar to this work, and more recently, a system that goes by the name of UNIDS which is able to detect unknown attacks without requiring any labeling, signatures or training. In order to understand the results obtained, they always rely on the assumption, just like the previously stated work, that the vast majority of the observed traffic is considered normal, rather than anomalous.

Each of the aggregated flows is described by $x_i \in R^m$, a set of m traffic features, such as the number of sources or destination ports, and a component of $X \in R^{m \times n}$, the complete matrix of all the features, being n the number of aggregated flows. The previously mentioned clustering algorithm will be performed the feature space X. First, an algorithm called Sub-Space Clustering will be applied to create N data partitions X_i out of X, by selecting k features from the complete set of m features. From these partitions, another clustering algorithm called Density-Based Clustering will be applied to each one of them, generating partitions P_i. Apart from the partitions, there will also be generated the previously mentioned outliers, which will later on be ranked in order to find classify the anomalies. To this, they use the concept of Evidence Accumulation, which uses the clustering results of

the multiple partitions, and produces similarity measures that are able to better reflect their groupings.

Based on the fact that machine learning methods achieve better results when classifying inliers than when classifying outliers, the training set consisted only in malicious data, rather than benign data. This way, the inliers will be the malicious data, for which the training algorithms will achieve better results. The first step in their system was, upon the receiving of a flow stream, the processing of data.

In this step, they filter the flows from the dataset in order to achieve a feasible value that would not overload the learning process. Then, they needed to tune the OC-SVM. To do so, they needed to achieve the best trade off between the parameters ν and γ which will determine the fraction of outliers in the set and the width of the RBF (Radial Basis Kernel) - that was added to the algorithm, respectively. Also, in this step, the most valuable features will be selected. This optimization was done first in a coarse grained fashion, to determine the best feature subset and the best trade off between the two parameters; and then, in a finer grained fashion, that would further explore the relation between the two parameters to achieve better results.

With this setup, they managed to achieve 87% false alarm rate, and a miss rate of only 6%, approximately [39]. This shows that this approach might be interesting to consider to our solution, as the results obtained were extremely good.

3. METHODOLOGY

This section provides an overview of implementation and preview of methodology in this area. Section 3.2 provide an insight on some of the tools and approaches. Section 3.3 and 3.4 proceeds to point out some network monitoring applications in this thesis.

3.1 CYBER ATTACK DETECTION APPROACH

In this section we describe a flow-based Cyber Attack Detection Systems CADS (capable of detecting unknown network attacking hosts. Moreover, this chapter provides a description of the new ideas introduced and to be further implemented.

. The inability to react to an unknown pattern, given a real-world dataset, i.e. containing both clean and malicious traffic

. The slow processing and analysis of the traffic payload, as well as the inability to interpret its content

The first drawback may be countered by using an unsupervised machine learning algorithm, and the second is tackled by performing the analysis at a flow-level.

3.2 APPROACH OVERVIEW

Figure 3.1, provides an overview of the structure of the system, as it will now be explained, following the workflow.

This filtering will consist of removing unnecessary features from the flows, e.g. its payload content and date (as the flows are organized in such way that each flow is stored in a file whose name has information regarding the latter). Therefore, apart from the traditional -tuples, three more features will be extracted (more details in Section 3.2), as depicted by step in the figure.

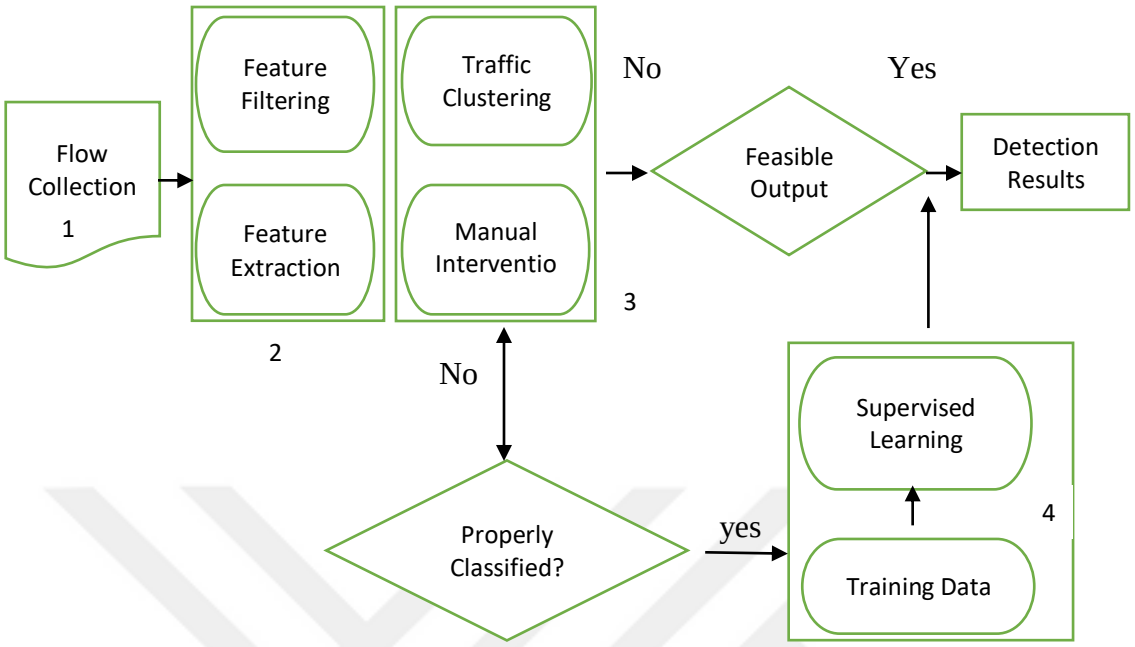

Figure 3.1: An overview of the Cyber Attack detection approach using supervised classification.

The extracted tuples will be fed to a detecting algorithm (step), which separates the whole traffic into groups of hosts that share a common pattern. We will take into account the assumptions that the majority of the observed traffic is benign rather than malicious, as well as that malicious traffic is qualitatively different from the regular, normal traffic, when analyzing the data. Upon the clustering of the data, a manual intervention takes place. This manual intervention is performed on the outlier clusters produced by the algorithm, in order to better perceive and analyze the characteristics of this traffic, ultimately leading to the production of a labeled dataset that will serve as training for the next step. Once the flows are properly classified, they are then passed on to the next step.

Step corresponds to a supervised learning module. In this module, the system runs a supervised learning algorithm that is trained with the labeled data produced by the former step, and will proceed to classify the traffic that was perceived as being outlier by the clustering algorithm. Upon this classification, the system should be able to correctly identify the observed malicious traffic, thus allowing the detection of the malicious hosts. This process is to be performed on a daily basis. If the analysis period was smaller, some attacks would not be possible to detect, as some of them last for long periods of time; if it was longer than a day, the obtained values would become noisy, as the flows are

aggregated, some IP addresses may be reused from one day to another, therefore achieving very high feature values, misleading us to think that it is indeed an attack.

The first step - filtering and feature extraction - will be crucial for the performance of the whole system, as it will be the input for the rest of it. If these features are carefully chosen, the system may be able to produce good results, performing an accurate clustering of data; if not, the clustering result might be completely inconclusive, thereby impairing the remaining modules.

Furthermore, the decision of using aggregated flows rather than simply analyzing individual flows, was due to the observation made in the intrusion detection community [14,21-23] that aggregated flows provide a more precise analysis, e.g. in the case of detecting DDoS. If some attacks were to be analyzed using individual flows, it would be much harder to detect them. The choice of using the source and destination IP addresses as aggregation keys was inspired by [34] that used them to distinguish groups of -to-N and N-to- anomalies.

The unsupervised clustering algorithm will be applied on these aggregated features, which will proceed to form various large groups of hosts, and some outliers. According to the previously mentioned assumption, these outliers may represent an attack, although this may not always be the case. Such outliers could also represent, for instance, some applications that are less frequently used, or even a machine whose characteristics are not very common, therefore producing flow features that are different from regular traffic that is found in bigger clusters. So, it is of utmost importance to analyze them, in order to differentiate the actual attacks from these benign outlier traffic patterns [28]. In a first run of the system, the supervised learning module has not yet any knowledge at all, and so there is a need for a manual intervention that will classify and label these outliers. This manual classification will serve as input to the supervised learning algorithm that, over time, will come more and more capable of classifying on its own. In the following runs, there may not be a need for manual intervention if the classification produced by this learning algorithm is feasible (which is to be validated with ground truth, as will be discussed in the next section); else, an expert must classify this traffic manually and feed the algorithm once again. The supervised learning algorithm that we will be using is the SVM.

3.3 SECURITY HARDENED RTU FOR DETECTION

Chapter two presented the security threats and vulnerabilities that currently face SCADA systems in general. The traditional IT security approach of soft on the outside (peripheral systems) and hard in the middle (servers) is not appropriate for SCADA systems. In SCADA installations the peripheral devices, such as RTU, must be security hardened as well. The focus of this dissertation is to identify and develop hardening techniques for RTUs and to develop a security hardened RTU. As discussed in the previous chapter, in the past, these devices faced primarily physical threats, but today they are increasingly network enabled and network accessible. Security hardening these devices is a major challenge facing the development of secure SCADA systems. Two security hardening approaches are explored in this dissertation, an RTU role based access control model and a reduced kernel OS. Previous work on both role based access control constraints and minimal kernels for operating systems was presented in chapter two. This chapter introduces the architecture for a security hardened RTU. The RTU role based access control model is presented in detail in chapter four and the minimal kernels for RTU.

Before considering specific RTU threats it is important to define, from a security perspective, the security perimeter of an RTU. This approach parallels the definition of a physical security perimeter that is a standard approach in securing physical places. SCADA systems are large distributed systems, and in developing a layered approach to security for them it is important to identify security boundaries for different components.

For field devices, the security boundary, or electronic perimeter, is defined to be the point at which the device makes contact with the SCADA network. For example, if the RTU connects to the SCADA network using Ethernet, then the electronic perimeter is the Ethernet controller card. It is important to establish such a perimeter; if the perimeter were too encompassing, the RTU's security perimeter would include components over which it has no control.

3.3.1 RTU Security Vulnerabilities

Vulnerabilities in RTUs can occur at many different layers. The highest and most abstract layer is the protocol layer. Protocols are abstract descriptions, and must be implemented, typically in software. Below the protocol layer is the software application layer which will

implement interfaces to the SCADA network, particularly SCADA protocols, but this also applies to other protocols that might be used now or in the future.

It is rare today that software applications are written to run at the hardware level. Instead software usually makes use of libraries, and other applications to achieve its goals. This creates yet another layer of shared software libraries and binaries, which is often referred to as middleware.

As mentioned in chapter two, initial SCADA protocols did not include security features, which resulted in vulnerabilities to message modification, spoofing, and sniffing attacks. The protocol vulnerabilities are really outside the RTU security perimeter, at least in their specification, but it is important to mention them, and keep them in mind in considering lower layers. One reason for keeping these in mind is the principle of easiest penetration [13]. If an RTU supports an insecure SCADA protocol that can easily be attacked and used to control or damage the connected physical processes, it will be impossible for a lower level prevention mechanism to protect the RTU since it has no way of differentiating between authentic and un-authentic SCADA communications.

Two excellent solutions to address the shortcomings in SCADA protocols have been presented in the literature review in chapter two [29-31]. Both solution, response approach adequately address the vulnerabilities in SCADA protocols.

3.4 FEATURE EXTRACTION

In order to obtain an overall improvement on the system's performance, we use parallel processing of the data, thus reducing the computational processing time.

To achieve so, the system implements a Map Reduce algorithm. Map Reduce, which was first introduced by Google [35], is a Big Data programming paradigm whose goal is to provide maximum scalability, in order to process massive amounts of data. It achieves so by dividing the task in two parts: the Mapper and the Reducer phases. The Mapper phase takes as input a set of data blocks and divides it into <key,value> pairs, as specified by the programmer; the Reducer phase receives as input the output generated by the former phase, and combines key pairs with the same keys, performing some operation to their respective values. Figure 3.2 depicts the basic workflow of the algorithm.

As it was previously mentioned, the flows are aggregated, which means that all the flows that have the same IP address (Destination and Source addresses for destination and source

aggregations, respectively) will be merged into one. This is when the Map Reduce algorithm will be applied. The mapper nodes will, for every entry in the data, extract the Source and

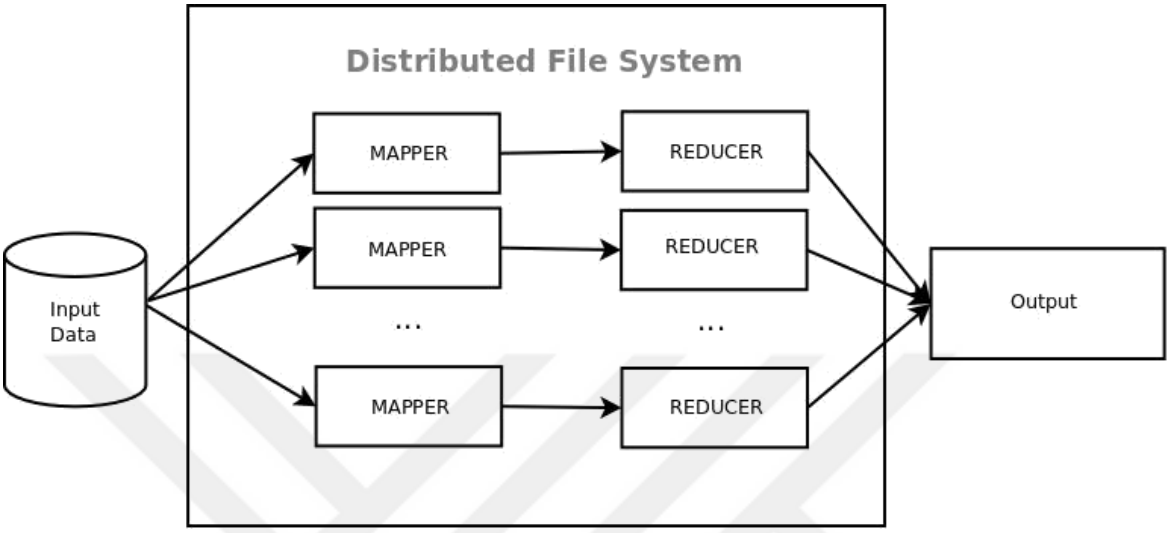

Figure 3.2: Map Reduce Algorithm Overview

Destination IP addresses, Source and Destination ports, number of sent packets, which protocol was used, TCP flag (if any), number of exchanged bytes and its duration. Each of these values will be sent individually to the reducer (i.e., for every entry, records will be sent), being the key a string in the following form: Source/Destination, feature, IP-Address, and the value will be the correspondent value of the feature in question. For example, in order to aggregate the number of bytes of a certain Source IP destination, the Mapper would produce a tuple, where would be one of the key IP addresses, and the number of bytes sent in that flow. The reducer nodes, on the other hand, will receive these records, and sum the value of all the records that have the same key.

The Map Reduce algorithm calculates the values (these sums) of a set of features, and produces some more feature that derive from these. These results are presented in Table 5.1. The Aggregation Key feature is obtained simply by creating a new tuple for each different IP address, both for source and destination. When the reducer receives tuples with an Aggregation Key that already has been seen before, it will generate the results described in Table 5.1.

3.4.1 Data Normalization

This step is again crucial. As we will be working solely with numerical data, we need to keep every value in one common scale. Moreover, there are some features that are not expressed in a numerical manner, such as the IP addresses of the IP's associated country. In these cases, these features are mapped to numerical values, which can be reversed to text.

$$y_i = \frac{y_i - \min(x)}{\max(x) - \min(x)}, y_i \in [0,1] \quad (3,1)$$

There is also a summation for all the times that a SYN flag is sent, and in the end of the processing, this value is divided by the total number of packets for that key, resulting in the feature SYN-Rate, and the same applies to ICMP-Rate, where the total number of times the Internet Control Message Protocol (ICMP) protocol is used is divided by the total number of packets. From the last features, the first were generated based on online databases, whose goal is to identify known threats, while the last one was fetched from a Python IP tracker module.

3.5 DETECTION THROUGH CLUSTERING

As stated in the previous section, machine learning algorithms can be divided roughly in two categories – supervised and unsupervised learning.

The idea behind clustering is to group different instances of a data into k distinct groups, i.e. clusters, according to their characteristics. For instance, applying a clustering algorithm to a dataset of network traffic would generate k clusters, where one would be representative of regular DNS traffic, another one would be simple SMTP traffic, and so on. This is done by feeding a set of vectors to the algorithm, which will then proceed to obtain groups of elements for that set of vector. The previously mentioned data normalization was performed specifically for this step. Depending on the algorithm used, the value of k may or may not be chosen automatically. For example, the DBSCAN algorithm [36] does not need a predefined value for the number of clusters; on the contrary, K-Means requires it. Two different clustering algorithms have been used throughout this work:

- K-Means
- Mini Batch K-Means

3.5.1 K-Means Clustering

This algorithm starts by receiving a set of vectors $X = (x_1, ..., x_n)$, $x_n \in R^d$, where d is the number of features. Based on the k predefined number of clusters, K-Means initializes a set of k random centroids (cluster centers) centroids, $C = (c_1, ..., c_k)$, $c_k \in R^d$. The first step is, for each point x_i in the dataset, it will compute its nearest centroid, using an Euclidean Distance (ED), which is computed using $d(\mathbf{a}, \mathbf{b}) = P^d_{i=1}(a_i - b_i)2$. This step is followed by another iteration, this time for each and every centroid. Each centroid will be updated so that its coordinates are the result of the mean value of all points belonging to that cluster. The algorithm will come to an end when the variations in the new centroid position are no longer significant, i.e. when $c_i > \xi$. Such behavior is described in algorithmically in Algorithm.

Although K-Means was proven to be an efficient algorithm, and widely implemented [36], it may not always be the best fit for every dataset. Its complexity grows with the number of clusters k, the number of dimensions d and the number of instances of the dataset, n. As stated by [33], briefly, its complexity grows as this:

- The time needed to assign the first round of points to its centroids is $O(n * d * k)$
- Computing time of each centroid is $O(n * d)$
- And, at last, the time spent in calculating the error function is $O(n * d)$ as well

So we see here that the computation time grows almost as we increase the dataset and its dimension.

3.5.2 Mini Batch K-Means

In order to overcome the computational cost of the K-Means algorithm when applied to large datasets, a new algorithm was proposed by Sculley et al. Mini Batch K-Mean [34].

While K-Means is applied to all of the instances of the dataset at once, Mini Batch K-Means takes randomly chosen smaller batches, and performs its computation from these smaller batches. For each of these batches, a cluster is assigned to each instance. Instead of using the mean of the values for each instance of the cluster, it uses a gradient descent update, which also speeds up the computational time. In addition to what the previous algorithm received as input, Mini Batch K-Means also requires the maximum number of iterations and the size of the batches.

Although this technique can be much faster than the standard K-Means algorithm, it produces slightly different results. For this reason, both techniques were tested and compared, in order to analyze which one was the best fit for our purpose.

3.6 SUPERVISED CLASSIFICATION

The malicious flows, i.e. the flows that correspond to intrusions, will be placed in the clusters with smaller size. Once the clusters are generated, each one of them will be manually inspected, in order to identify if they are malicious. In order to obtain a coarse grained overview of each cluster's content, each feature of each cluster will be described by its mean value and standard deviation. This way, it is possible to have an idea of each cluster's behavior. Of course, this is only a preliminary method to indicate each cluster's feature distribution. Then, we focus on the clusters with smaller size and higher feature values, and proceed to label one of the flows that they contain. Each of these labeled flows is then fed to the SVM, will learn from these examples.

Supervised learning problems can be divided into two different categories. 1) Regression, and, 2) Classification. The former produces outputs for R, while the latter produces outputs for N. Since our goal is to detect malicious hosts, we want our outputs to be discrete values, identifying if one is or is not malicious, we fall therefore under the second category. Having the data separated into different groups according to their respective behavior and being properly labeled, from the last module, we have now gathered the necessary conditions to apply such a technique. For the purpose of this work, we chose to apply a SVM. This decision was due to the fact that this algorithm is proved to behave well when it comes to intrusion detection [36-39].

3.6.1 Support Vector Machine

The goal of the SVM algorithm is to find an optimal hyper-plane for patterns that are linearly separable. A hyper-plane can be defined as Euclidean space that has one less dimension than the space provided for the problem, e.g. for a -dimensional space, the hyper-plane would simply be a line. The optimal hyperspace is that which maximizes the margins between itself and the vectors provided, i.e. the classes that will be used for classification. The SVM receives as input $n \times f$ training vectors $\mathbf{X} = (x_1, x_2, ..., x_f)$, where f is the number of features in the dataset, and a $n \times 1$ vector, which will be the

corresponding class for each of the input **X** vectors. It will produce as output a set of weights w, whose linear combination will predict the value of y, for each **X** provided from now on.

For the purpose of this work, the SVM will have two class, namely and – a binary output. For each input flow, it will label it as if the flow is benign, or as if the flow is malicious. The reason why there is only one class is that the goal of this work is not to classify, but to identify attacks, and therefore there is no need to differentiate them. Our aim is to identify malicious hosts, regardless of the type of attack they conduct, and therefore the SVM output will identify each flow as being malicious or benign.

3.6.2 Random Forest

Features are obtained simply by summing all the features values in each flow, that is, for each flow the random forest algorithm will produce a sum of all the different IP addresses contacted, all the different Source Ports and Destination Ports, number of packets and number of bytes, respectively using the random forest. There are also features that will count the occurrences of contacting ports, or, and , i.e. The RF algorithm receives as input $n \times f + 1$ training vectors $\mathbf{X} = (x_1, x_2, ..., x_{f+1})$, where f is the number of features in the dataset, and a $n \times 2$ vector, which will be the corresponding class for each of the input **X** vectors.

By dividing the total number of packets by the total duration of the Aggregation Key, we obtain the Pkt-Rate. As we mentioned earlier, one of the features that composes a flow is what TCP flag is being sent, if any using random forest algorithm.

There is also a summation for all the times that a SYN flag is sent to random forest distinctive algorithm, and in the end of the processing, this value is divided by the total number of packets for that key, resulting in the feature SYN-Rate, and the same applies to ICMP-Rate, where the total number of times the Internet Control Message Protocol (ICMP) protocol is used is divided by the total number of packets. From the last features, the first were generated based on online databases, whose goal is to identify known threats, while the last one was fetched from a Python IP tracker module. The Precision and accuracy is also measured and attained using random forest approach.

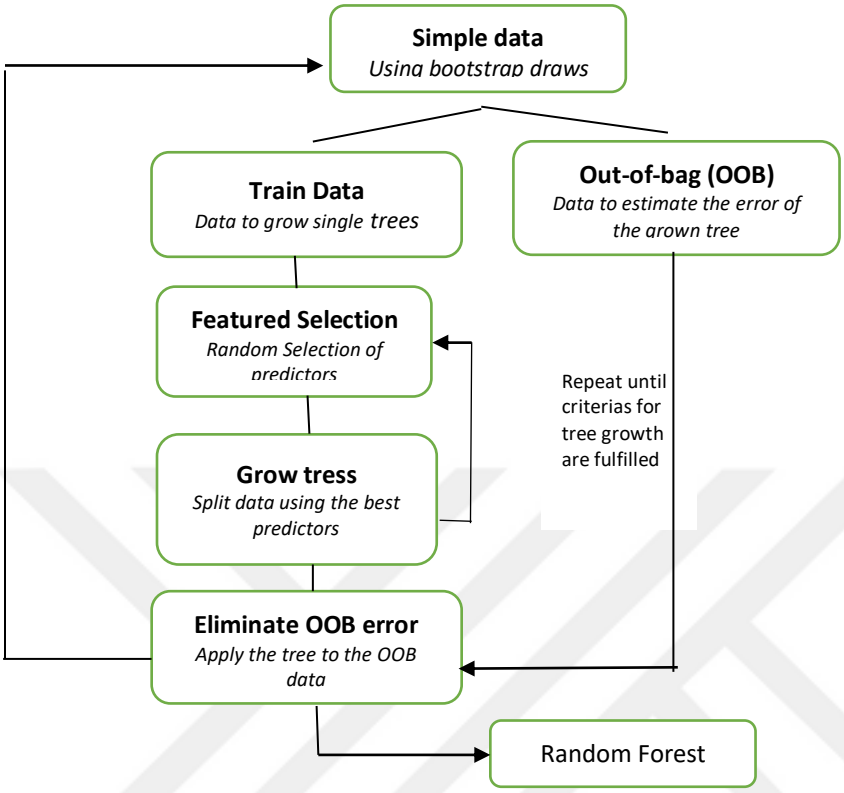

Figure 3.3: Random Forest Algorithm used for classification of Cyber Attack.

In the feature selection phase the algorithm will proceed to receive as input the outputs produced by the train data and process them in order to produce the features that are described in figure 3.3. Once these features are computed, the final output of the program will be written in two files:

One containing the data per set, including the file where each of the flows is located; another where the file is excluded and the IP addresses and the Location Codes are converted to a numerical format, in order to be able to perform some mathematical operations to the data, which will be necessary for remainder of the program.

4. SOLUTION

This section will describe the implementation of the CADS. All of the following modules were developed in Python, as it has a wide open source community, with a number of machine learning libraries available, already optimized for the purpose of this work.

As the supervised learning and the classification modules are the sections that require the more interaction with the user, a terminal interface was developed in order to facilitate such tasks. The interface for the unsupervised learning module features a menu that allows the user to: 1. Obtain a brief description of the data (Text or Data file); 2. Plot graphics to aid in the choice of the number of groups; 3. Generate and visualize the group; 4. Analyze each of the generated group; 5. Interactively visualize the generated group; 6. Access a data to validate the attacks, if possible; 7. Plot two features, one against the other; 8. Alternate between data files; 9. Change machine learning algorithm. As for the supervised learning module, the interface features a menu that allows to: 10. Predict values for a given data 11. Train the system. With these interfaces, it was much simpler to analyze and interpret the results obtained in this section thus allowing to properly classify the observed traffic, as it allows for an easier interaction between the user and the data. The remainder of the system does not feature a terminal interface, but it does have scripts that automate the processes, as it did not require so much interaction as the former modules, it can be made automatically.

4.1 MAP REDUCE

As it operates in a distributed fashion, it can take advantage of a group of computers, using its computational power to achieve a higher speed during the data processing. To support the parallel data processing, Hadoop implements Google's Map Reduce algorithm [44]. This model operates on a virtual environment called Hadoop Distributed File System (HDFS), which has both Mapper and Reducer nodes. This model can be divided in two main steps: Mapping and Reducing. First and foremost, the input data is split in equal sized blocks (MB by default), and each of these blocks is fed to a Mapper node. Each of these nodes divides the received chunks into <key,value> pairs, that will then be sent across the network to the Reducer nodes. Before proceeding to execute the algorithm, a filtering to the dataset is performed, maintaining only the following features:

- Source Internet Protocol (IP) address
- Destination IP address
- Source Port
- Destination Port
- Number of packets sent
- Protocol used
- Which TCP flag was sent, if any
- Number of bytes sent
- Duration of the flow

Having the data filtered, the system proceeds to execute the Map Reduce algorithm. This consists in two distinct parts, as the name suggests: the Mapping and the Reducing phases. The goal of the former is to aggregate values by a specific key; the goal of the latter is to perform some sort of operation on the aggregated keys generated by the Mapping phase. In our case, the aggregation keys will be both Source IP and Destination IP. However, for each of these keys, a number of operations on each feature must be performed. Therefore, the Mapper consists in two functions:

- Source Features
- Destination Features

Both will produce <key,value> pairs for each one of the features mentioned above. Apart from these, the mapper will also produce some more features:

- A counter for the number of times the port was used for SVM
- A counter for the number of times the port was used for Random Forest
- A counter for the number of times the port was used k-NN

These ports relate to the protocols HTTP, IRC, SMTP and SSH, respectively. The goal here is to count the occurrences of each one, whilst preserving the aggregation key. For the Source IP as aggregation key, the mapper will produce the following outputs:

<"S,bytes,sip,file",#Bytes>

<"S,dstIP,sip,file",DstIP>

<"S,dstPort,sip,file",DstPort>

<"S,duration,sip,file",Duration>

<"S,packets,sip,file",#Packets>

<"S,protocol,sip,file",Protocol+Flag>

<"S,srcPort,sip,file",SrcPort>

<"S,HTTPPort,sip,file",Yes/No>

<"S,IRCPort,sip,file",Yes/no>

<"S,SSHPort,sip,file",Yes/No>

The outputs produced are analogous the the destination function, also present in the Mapper phase, simply replacing the 'S' key by 'D', and 'sip' by 'dip'.

In the Reducer phase the algorithm will proceed to receive as input the outputs produced by the Mapper and process them in order to produce the features. Once these features are computed, the final output of the program will be written in two files:

One containing the data per set, including the file where each of the flows is located; another where the file is excluded and the IP addresses and the Location Codes are converted to a numerical format, in order to be able to perform some mathematical operations to the data, which will be necessary for remainder of the program.

4.2 MODEL 1

Having the data filtered, the system proceeds to execute the machine learning algorithms. This consists in two distinct parts, as the name suggests: the machine phase and the learning phases. The goal of the former is to aggregate values by a specific key; the goal of the latter is to perform some sort of operation on the aggregated keys generated by the Machine phase. In our case, the aggregation keys will be both Source IP and Destination IP. However, for each of these keys, a number of operations on each feature must be performed. The libraries are import and data is loaded based on different algorithms of machine learning.

Figure 4.1: Model 1 depicts the whole distinctive algorithms used for detecting and analyzing the type of attack.

4.3 MODEL 2

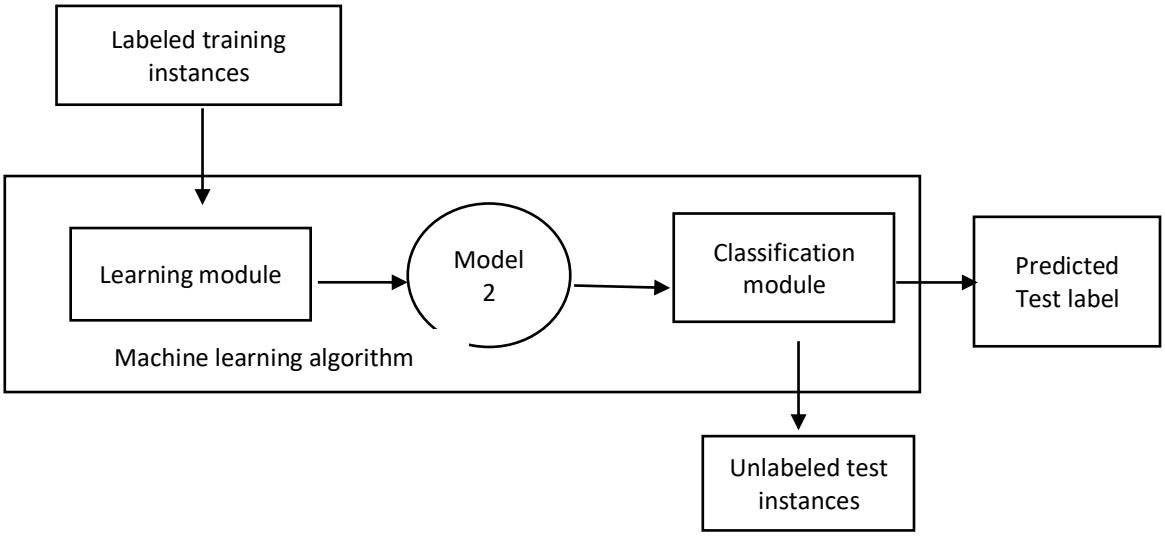

Figure 4.2: Training model for the labeled and unlabeled test for predicting the unusual Cyber Attack or intrusion from external network.

4.4 UNSUPERVISED LEARNING

For the third step of Figure 4.3, the user must run the unsupervised learning module. Once the flows are aggregated, the unsupervised learning module may take place, in order to cluster the flows.

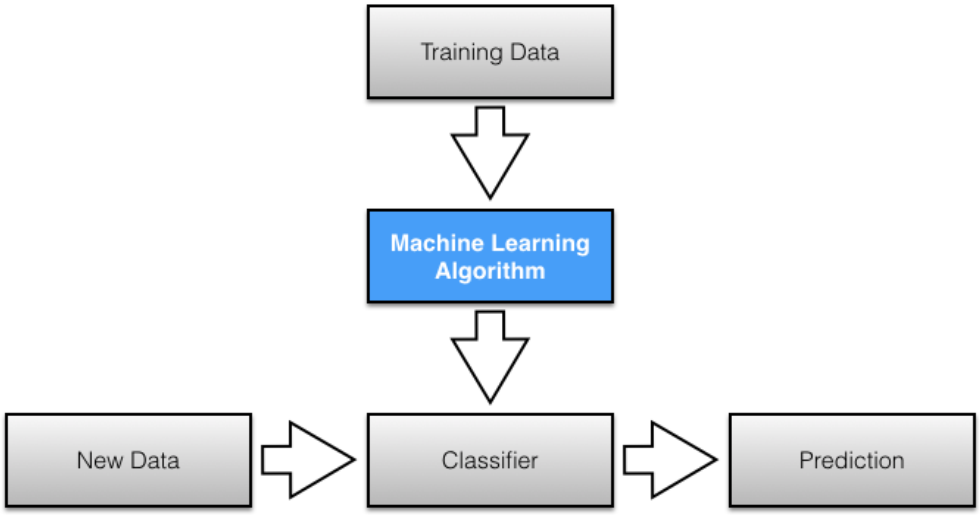

Figure 4.3: Terminal user interface for the unsupervised learning module

This module presents a interface to the user, which consists in a group of task implemented in the CADS (see Figure 4.2):

- Obtain a statistical description of the data.
- Obtain a representation of the optimal number of the clusters for the data presented.
- Generate the clusters and visualize them (either in R^2 or R^n).
- Obtain a statistical description of each cluster.
- Launch an interactive representation of the clusters, to aid visualization.
- Validate attacks in the data, from data.
- Generate a graphic which maps just features
- Alternate between data (source or destination)
- Alternate between grouping algorithms (SVM, Random Forest and k-NN)

The first option provides the user with a brief description of the data being analyzed, featuring the number of entries of the dataset, and, for each feature, it also presents its minimum and maximum value, and also the mean value and standard deviation. This way, we get a simple overview of the data, statistically. The fourth option presents, for each

cluster and features, its mean value and standard deviation. As for the fifth option, window is opened in the browser, that allows to choose which cluster to analyze, and provides an interactive visualization of the dataset it contains, as seen in Figure 5.1.

4.5 ENCRYPTION METHOD IN CADS

As it was discussed in Section 5.3, both SVM and Random forest algorithms require a number k groups to be specified. Upon the receiving of the data, we do not know exactly into how many groups we want to divide our data. However, there are some techniques that give us a hint of what the value of k should be. Such is the case of the Encryption Method (see Figure 5.4). These clustering algorithms converge when the variation of the distance between the data points and the clusters centers start converging to section 5.2. With this in mind, the Encryption Method starts by computing the error function that is used as a stopping criterion in the algorithm, known as:

Within Sum of Squares (WSS), which is mathematically defined as such:

$$WSS = P \sum_{i=0}^{K} P_{x \in i}\ dist(x, ci)^2 \tag{4.1}$$

This equation will produce values for k in a specified range, which will be provided by the user. By plotting this values against its respective k value, we obtain a graphic that will be decreasing with the increase of k. The optimal value of the WSS would be same, but this value is only obtained when the number of k clusters is equal to the number of entries in the dataset, which would mean that each data point would be in its own clusters, and this process would provide us no information at all.

Instead, the Encryption Method tells us that the best value of k is when the slope of the WSS has a sudden break. Apart from the WSS, the Encryption Method also plots a metric called Percentage of Variance Explained (PoVE) for each value of k. This metrics reflects the ratio of the Between Sum of Squares (BSS) and Total Sum of Squares (TSS), which will indicate an optimal k when it suffers an abrupt change and encrypting the files for example text file and data files.

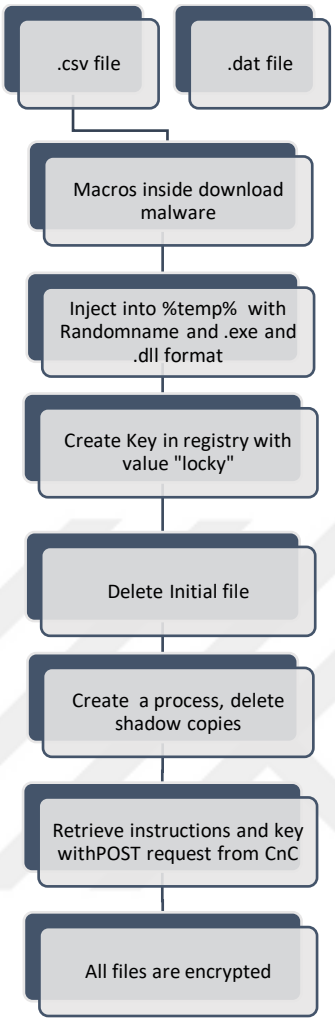

Figure 4.4: Encryption model based on Lock value in text and data files for detecting the Cyber Attack or intrusion in the Network.

Although this method provides us with some more insight than what we previously had, this is a heuristic method, which may be statistically correct, but may not fit our dataset with maximum precision. With this in mind, this tool is used only to give a hint of where a correct k value might be. After plotting the graphic, we still have to try several runs with values around the suggested k. Accord to the Encryption Method, the optimal k would be – in Figure 4.4 we see that the biggest slope is found where the x-axis is equal to –, which is not correct. Ultimately, the finding the optimal number of clusters cannot be an automated method, as human intervention is always necessary, due to the fact that the number of groups will always be dependent on our type of data, and the purpose of clustering. However, we do know that the value lies nearby , as shown be the Encryption Method, and after several runs with different values for k, we found that k = 10 is a number of clusters

that successfully and coherently divides the different datasets of various sizes. Also, if we analyze Figure 5.4, we see that around the value of the values start to stabilize, and its variation is close for encrypting the text and data files.

4.6 GENERATING THE CLUSTERS

This task corresponds to the third option prompted in the interface – Generate the clusters and visualize them. When the data is described in more than three dimensions, it becomes a difficult task to represent it visually. For this reason, a -Dimensional representation of the dataset is also provided. Instead of dropping several features until only two are left, the dataset is rearranged such that there are now two features that describe the original ones. This process is called dimensionality reduction, and may be achieved through Principal Component Analysis [42]. PCA is a Factual technique that is used to identify hidden patterns in the data, by identify a number of uncorrelated features smaller than the original dataset. These smaller number of features are called the Principal Components. This method is able to reduce the number of features in a dataset by computing its eigenvectors and respective eigenvalues from a correlation matrix. This allows to retrieve the information that best describes the correlation between features, therefore synthesize the information into a smaller number of features.

As for plotting the dataset in its original dimension, a technique called Parallel Coordinates was used [43]. With this technique, the x-axis is divided into f sections, being f the number of features, while the y-axis is maintained as if in a normal projection. This technique provides a visual aid for better understanding of our data, as the data can now be fully plotted in its entirety, therefore allow to easily identify which features have significantly higher values than others. Such representation is depicted in Figure 4.4.

Upon the creation of the clusters, a file for each one of them will be generated, in order to further analyze them in depth. All of the data that is generated in this module is stored in a dataset, in order to maintain a persistent storage of the data. With the dataset, it is possible to trace-back all the information regarding the flows of gas and detecting the Cyber Attack. This is necessary, given that the data handled by the algorithm is strictly numerical, and the raw data contains textual information as well, namely the file where the flow is contained, the aggregation and also the location of the flow (i.e. its Respective Gas Pipeline Flow).

4.7 SUPERVISED LEARNING

For the fourth step of the figure, the user must run the supervised learning module, which is described below.

In order to generate the hyper-plane that divides the data into different classes (in our case, the classes are either), the SVM algorithm uses kernel functions, which allow to divide it in linearly or non-linearly separable classes. A kernel function is a similarity function, i.e. a function that describes how similar two inputs are between each other. These kernel functions can be either Linear or Random Forest (RF). While the former is simply a linear combination of weights and is used for linearly separable datasets and requires no additional inputs, the latter will decided which is the optimal hyper-plane using either a Polynomial or Gaussian approximation. Also, this latter function requires an additional parameter γ, which will determine the width of the function. The system was tested using both Linear and RF kernel functions, and when using a linear function the machine was able to produce more accurate results.

The results of the SVM are stored persistently, in a file. This way there is no need to train the system every-time we want to use the SVM. The system starts by verifying if this file exists, and, if not, it trains itself with the data from the first day of the analysis period – if the system is to be evaluated in a period of one week, it the SVM is trained with the first day of the week. Upon this training it is possible (see Figure 4.2)

In the first case, the system will present the user the flows (if any) that were labeled as malicious, for that day. These flows are identified, once again, by their unique ID, thus making it possible to trace-back it back to its IP address stored in the database mentioned in Section 4.6.

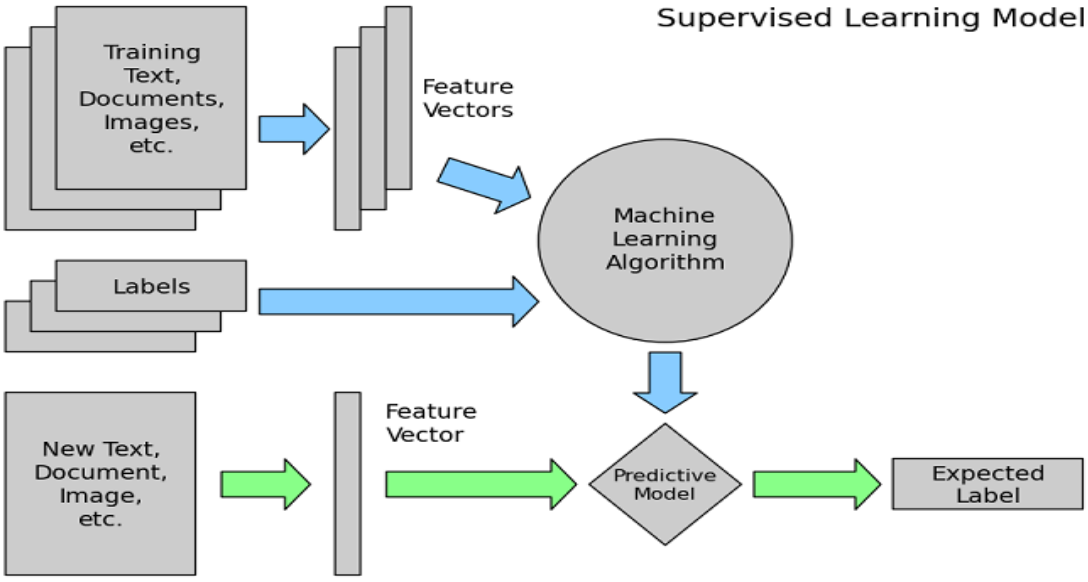

Figure 4.5: The optimality of machine learing in RTU of SCADA system to help detect the Cyber Attack in the Industrial control system, but this value is only obtained when the number of k clusters is equal to the number of entries in the dataset, which would mean that each data point would be in its own clusters based on dataset.

For the second case, more knowledge will be added to the SVM. This new data that will train the system must first be merged with the data that was already present, and then the system is trained with this new set of data. With this feature, it is possible to refine the results generated, as it allows to give new insight to the SVM [58]. This means that, as we add more data to the system, it will have a better understanding of it, and will allow for it to detect malicious intent more accurately according to our analysis. For example, if in a first analysis some sort of malicious activity was not detected, and the system as trained with this data, it may not detect intrusions that follow that mislabeled pattern [62-65]; but if later on we perceive that activity as malicious, it is possible to train the system once again with this new information. This process therefore allows to refine the SVM accuracy throughout the analysis period.

5. RESULTS

This section will display the results of the CADS. All of the following modules were developed in Python, as it has a wide open source community, with a number of machine learning libraries available, already optimized for the purpose of this work based on implementation parts.

5.1 CLASSIFICATION

Now the classification part comes. We used 'scikit-learn' to import classification algorithms. We classified our dataset from "Tommy Morris Dataset for Industrial Control System (ICS)" of Gas Pipeline System with many classifiers like SVM, Random Forest and k-NN. Some of the classifiers were quite efficient but some other were not up-to the mark. The Recall and Accuracy are represented in the result analysis part below.

5.2 RESULT ANALYSIS

K-NN: We examined K-NN classifier with different values of K. But 2-nearest neighbors earned the best result among them. We also set the weight of the algorithm according to the distance between two points. The leaf size was restricted to 30. Setting all these parameters, we got an accuracy of 87.37%.

RANDOM FOREST: This classifier worked with less efficiency for our corpus. We set the criterion of the algorithm to 'entropy' and got the result of 85.50% accuracy.

SVM: SVM is one of the best fitted classifier in our work. There are several kernels used in SVM but we implemented 'k-NN' and 'RF'. Linear SVM gained an accuracy of 86.04% setting the gamma value of the algorithm to 10, SVM with RF kernel gave us the best result of 85.50% accuracy.

5.3 EVALUATION

A file for each one of them will be generated, in order to further analyze them in depth. All of the data that is generated in this module is stored in a Gas Pipeline dataset, in order to maintain a persistent storage of the data. With the database, it is possible to trace-back all the information regarding the flows. This is necessary, given that the data handled by the

algorithm is strictly numerical, and the raw data contains textual information as well, namely the file where the flow is contained, the aggregation and also the location of the flow.

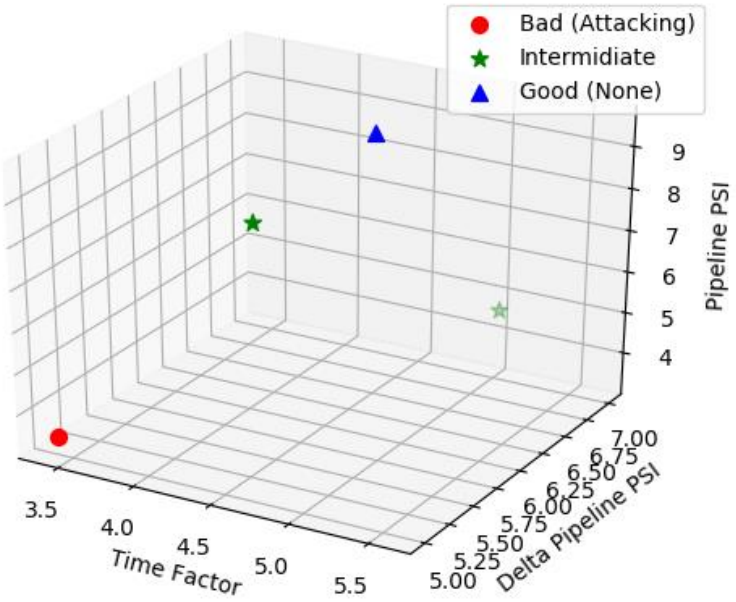

Figure 5.1: The Pipeline PSI clusters is in y-axis and the Delta Pipeline represented on x-axis with time factor cluster on z-axis of detection graph. The Dataset is dividing into four clusters for detection of attack.

This latter function requires an additional parameter of Data-length and Function Code of data in which amount of variables were provided in dataset, which will determine the width of the function. The system was tested using both Linear and RF kernel functions, and when using a linear function the machine was able to produce more accurate results.

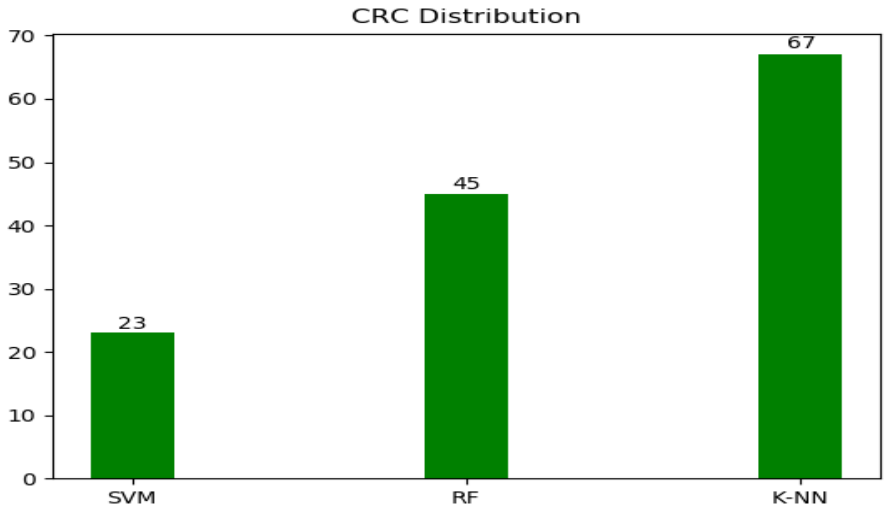

Figure 5.2: The CRC of dataset is represented through the clusters of distribution and for all the algorithms of supervised learning SVM, k-NN and Random Forest.

A kernel function is a similarity function, i.e. a function that describes how similar two inputs are between each other.

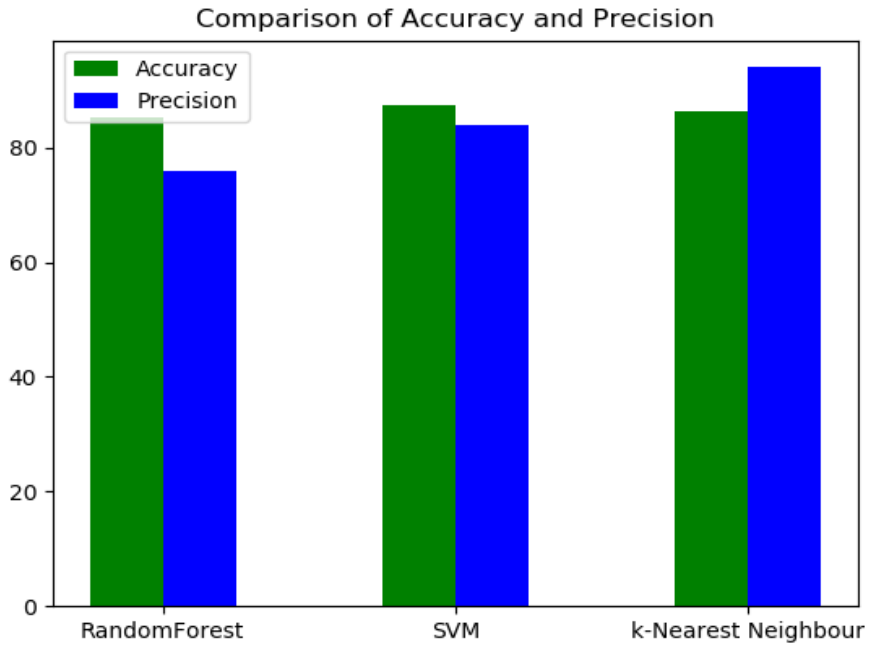

Figure 5.3: The accuracy and precision of all three supervised learning algorithm for detecting the Cyber Attack in remote terminal units of SCADA system.

Table 5.1: The precision and accuracy of all three algorithms extracted from the dataset of gas pipeline in remote terminal units of SCADA system.

SVM		k-NN		RF	
Detection Accuracy	**Detection Precision**	**Detection Accuracy**	**Detection Precision**	**Detection Accuracy**	**Detection Precision**
86.04%	94.04%	87.37%	83.90%	85.50%	75.89%

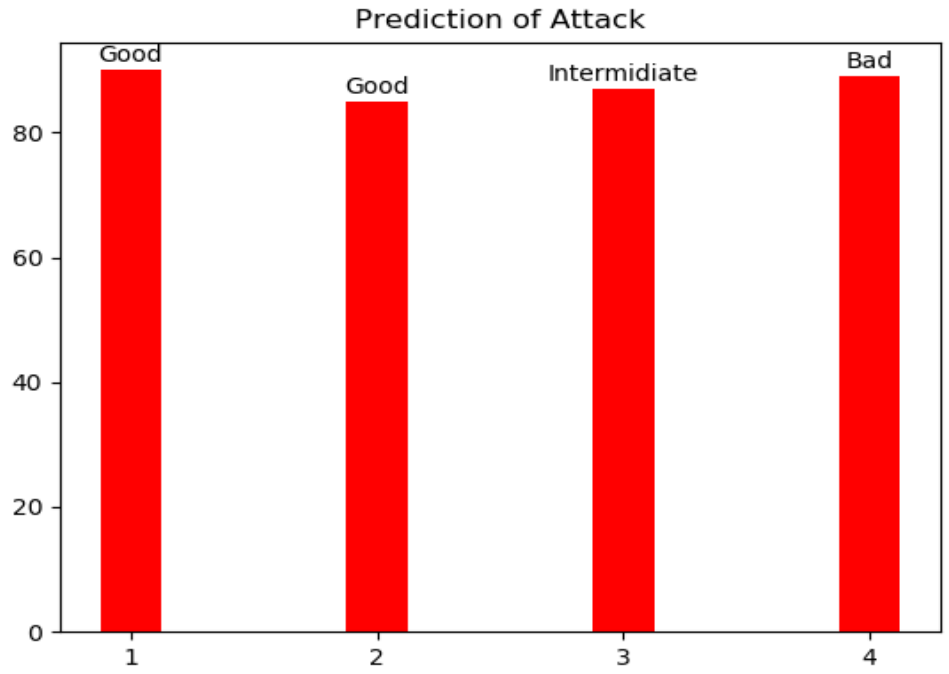

Figure 5.4: The prediction of attack carried out on the autonomous system of gas pipeline depends on the ration of pipeline PSI both in delta and non-delta pipe line PSI and exhibiting the attack prediction that whether there is an attack or not based on the clusters of dataset.

Factual technique that is used to identify hidden patterns in the dataset of gas pipeline system, by identify a number of uncorrelated features smaller than the original dataset. These smaller number of features are called the Principal Components. This method is able to reduce the number of features in a dataset by computing its eigenvectors and respective eigenvalues from a correlation matrix to detect the Cyber Attack carried out in the services of supervised system. This allows to retrieve the information that best describes the correlation between features, therefore synthesize the information into a smaller number of features.

6. DISCUSSION

This section will discuss the results of the CADS with other CADS already in existence. All of the following modules were developed in Python, as it has a wide open source community, with a number of machine learning packages available, already optimized for the purpose of this work based on implementation parts.

The attacks that were perpetrated in a big scale were correctly detected and distinguished from the regular traffic, as the system detected both DoS and a portion of DDoS attacks. Place was a Distributed Denial of Service (DDoS) IRC Botnet attack, which is a botnet that takes control of its bots through IRC C&C servers, and uses them as a third party to induce DDoS attacks. On this day, the system was only able to correctly identify one malicious flow (only when manually analyzing the cluster, this SVM was not able to detect it), as the rest of them were not discovered [51-54]. The remainder of the malicious flows were able to mask themselves among clusters that did not have high feature values, therefore making them indistinguishable, especially having this clusters a very high number of entries. Moreover, although this attack was an IRC Botnet, the values for the number of IRC communications remained undetectable, once again proving that the majority of the attacks was able to mask themselves. At last, Thursday, just as Saturday, had a Brute-Force SSH attack in the remote terminal units of SCADA systems [55-60], which as correctly identified. However, in this day, the system led was to think that there were also two other clusters that also corresponded to an attack, and in fact they were not [64].

The tables shown in this evaluation were all for the source aggregation shown, as it was found that the flow direction of all the attacks was either Local to Remote (L R) or Local to Local (L L). This way, all the attacks are identifiable by tracking the Source IP addresses, and not the destination IP addresses, since we aim to find the malicious hosts; by analyzing the destination aggregation key, in this case, we would find the victims of the attacks, rather than the attackers, which is not the goal of this work. A general view of the system's detection results can be seen in Table 6.1.

The reason that some attack were not identified, is that this system's focus is on large volumetric attacks, i.e. attacks that occur in large volumes, that exhaust the bandwidth of a network, and with feature values that tend to inflate [59]. In the case of the DDoS attack,

the system was only able to identify the one host that had the largest attack volume, as the rest of the host were producing a silent attack, that the system was not able to detect.

6.1 DATA ANALYSIS

For a set of reasons, we were not able to analyze real data corresponding to several days. The real data was obtained in the scenario of Figure 5.1 and 5.2 and corresponds to part of data collected for detecting the Cyber Attack. We collected around GB of data, which corresponded to around million flows in the Gas Pipeline System of Industries.

Unlike the ICS dataset, this real data is not labelled, i.e., there is no ground truth on existing attacks. Therefore, we have analyzed both source and destination keys. Moreover, the number of groups generated for each of the aggregation keys was instead of the ICS dataset. The reason for this was that the volume of the data was much than immensely bigger than the previously analyzed data, and so the number of clusters could have not been the same [62]. After running the system with several different values, we found that groups was a good value for the number of groups.

For both source and destination aggregation keys, the grouping was performed with the SVM and Random Forest algorithms. We can see that although the SVM algorithm has a bigger computational complexity and slower performance, it provides more detail on the data. It allows to unveil patterns that were hidden when using Random Forest, and so we will focus our analysis on the clusters generated by the SVM algorithm.

When looking at the destination aggregation key clustering content (Table 6.1), we see that groups 8,13,21,27 and are those whose features are the most alarming. Therefore, this may also correspond to infected hosts that are being used as a third party for attacks, but contacting its bot-master through a C&C server other than an IRC, or they could be victims of an attacker who is using spoofed IP addresses to use them as a third party.

Table 6.1: Data analysis results using summarizes the information regarding the intrusions detected throughout this analysis for the feature numbers.

Groups #	Aggregation Key	Highlighted Features	Type of Attack
8	Source	2,4,6,9,13,17	Spam / DoS
9	Destination	2,4 ,5	DoS

13	Source	9	Brute-Force SSH
14	Destination	2,5,11	Network Scan
21	Source	8,11	Botnet Communication
23	Destination	2,4,6,13	Web Application Probing
27	Source	2,3,5,7,12,14	DDoS IRC Botnet
29	Destination	2,4,5,9,14	DDoS Botnet

6.2 RELATED WORK

Paper [24] being the first to detect Cyber Attack in text as per their literature review. They have approached through text mining and lexicon based technique. At first they have gone through some case studies. Then they surveyed on this issue like how much aware people are about Cyber Attack and so on. In their survey they found that girls (45%) are having more experience on Cyber Attack than of boys (28%). For applying methodologies, they used data set of tweet conversation in text or data files. At first they develop a corpus of tweet conversations. In text mining approach, firstly they applied preprocessing techniques. Then they used it to train and test classifier that can detect Cyber Attack. In lexicon based approach, firstly they developed an English lexicon with attacking words. Then the corpus of tweet conversation got checked with the lexicon. If they found number of attacking words beyond some threshold, it could label as Cyber Attack. In the paper, basically they have discussed the challenge of Cyber Attack in modern world after surveying and studied several cases and showed up two methods to detect Cyber Attack. But they didn't implement the methods yet. So, how the results will be with these methods are ambiguous. Besides, the corpus of lexicon on English language still doesn't exist. It must be developed first. The methods also can detect words as Cyber Attack which was not meant to say as Cyber Attack.

Research paper [23] thinks of an exploration on arrangement of Cyber related remarks and assaults on gas pipeline industry. They drew nearer with the system, Support Vector Machine which is made with R dialect. As information, they have picked remarks from records of malaysian famous people in the wake of studying that individuals get assaulted in gas pipeline industry increasingly (42%) than different gas pipeline industry. All the

more decisively they utilized records of two clients who are Indonesian. 1053 remarks and assaults were taken as preparing records and 34 as test reports. For actualizing the strategy, right off the bat, they made a report term network with R so as to shape SVM demonstrate. When the development of SVM finished, they utilized it to foresee remark and assault whether it is Cyber Attack or not. They indicated a remark by inscriptions - 1 on the off chance that it was Cyber Attack and 1 in the event that it was not Cyber Attack. At long last they got exactness level of 79.412% as result utilizing SVM. They didn't utilize any part of SVM and in paper [31] SVM with poly bit demonstrates a superior precision then this current papers' outcome (99.41%).SO, utilizing portion may give better exactness.

Paper [27] worked on application of machine learning to detect and recognize the potential attack in the terminal units of autonomous chemical units which are intruding and attacking. Actually, it looked into the machine learning algorithms and compared which one is better. Their experiment includes algorithm Multinomial Naïve Bayes (MNB), Random Forest (RF) and Support Vector Machine (SVM). The kernels of SVM used are linear, Radial Bias Function (RBF), Polynomial and Sigmoid and achieved 78.55% of total accuracy as in comparison with our model and work.

7. CONCLUSION

Furthermore, the solution was designed in a way that it allowed for the detection of large volumetric attacks, attacks that would produce very high feature values, so that the hosts producing such traffic, were easily distinguished from the remainder of the traffic, and producing patterns that allowed us to detect such an intrusion in the RTU of SCADA systems and its protocols. However, as shown, this approach allows us only to detect a small portion of the attacks that may be going through a network, as many of them are done almost silently, making this approach infeasible. Still, the attacks that were perpetrated in a big scale were correctly detected and distinguished from the regular traffic, as the system detected both DoS and a portion of DDoS attacks, Brute-Force SSH attacks, and was also able to detect part of an intrusion from the inside of a network. When analyzing the data, although we do not have ground truth to perform a validation of the system's performance, the CADS was able to unveil some interesting pattern using the remote terminal units RTU in SCADA systems usually using the machine learning algorithms (SVM, K-NN and RF) for detecting the Cyber Attack and precision of detection. Even with the great amount of data, it was able to isolate a small number of flows that presented alarming patterns, ultimately led to identifying them as being malicious. In this data, the system was able to locate a machine producing major amounts of traffic, leading us to believe that it was either a major spammer or perpetrating a DoS attack, small DoS attacks, a few network scans, and identify the perpetrators of these attacks, thus accomplishing our goal of detecting malicious hosts. Still, it does not allow to identify every network intrusion event, as some of them are performed with low intensity, thus being able to evade the system.

For our thesis work, to detect Cyber Attack in the gas pipeline industry specially using the balanced approaches of supervised machine learning algorithms, we chose a quite balanced data that had reasonable amount of data; cleaned and preprocessed it. We used model to fit and transform our data into machine recognizable form to detect and recognize the effect of appearing attack in future. Extracted the best features by using suitable methods. Split the data and applied various supervised machine learning classifying algorithms such as Support Vector Machine, Logistic, K-Nearest Neighbor and Random Forest to detect performance. We tweaked and experimented with our data and model to achieve best accuracy possible. Eventually, we got the best result using Support Vector Machine using

kernel for detecting the Cyber Attacks in the giant gas pipeline industry based on introductory technology of SCADA system.

7.1 FUTURE WORK

Now, we are only detecting whether it's Cyber Attack or not in remote terminal units of SCADA system using machine learning algorithms for detecting the Cyber Attack. Our accuracy is quite satisfactory but we will try more data manipulation, preprocessing and also hybrid classifying classifiers to increase the accuracy if possible. We might even try using different approach for detecting Cyber Attack if needed.

REFERENCES

[1] L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar, "Adversarial Machine Learning," in ACM workshop on Security and artificial intelligence, 2011.

[2] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler. SPINS: Security protocols for sensor networks. Wireless Networks 8:521–534, 2002.

[3] A. Perrig, R. Canetti, D. Song, and J. Tygar. Efficient and secure source authentication for multicast. In: Network and Distributed System Security Symposium, NDSS'01, 2001.

[4] A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In: IEEE Symposium on Security and Privacy, 2000.

[5] Washington Post. Dissertation could be security threat. http://www.washingtonpost.com/ac2/wp-dyn/ A23689-2003Jul7.

[6] A. K. Wright, J. A. Kinast, and J. McCarty. Low-Latency Cryptographic Protection for SCADA Communications. In: Proc. 2nd Int. Conf. on Applied Cryptography and Network Security, ACNS 2004, pages 263-277, LNCS 3809, Springer 2004.

[7] F. Pierazzi, G. Apruzzese, M. Colajanni, A. Guido, and M. Marchetti, "Scalable architecture for online prioritization of cyber threats," in International Conference on Cyber Conflict (CyCon), 2017.

[8] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection," in IEEE International Conference on Platform Technology and Service (PlatCon), 2016.

[9] P. Torres, C. Catania, S. Garcia, and C. G. Garino, "An analysis of Recurrent Neural Networks for Botnet detection behavior," in IEEE Biennial Congress of Argentina (ARGENCON), 2016.

[10] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013.

[11] G. D. Hill and X. J. Bellekens, "Deep Learning Based Cryptographic Primitive Classification," arXiv preprint, 2017.

[12] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015.

[13] M. Z. Alom, V. Bontupalli, and T. M. Taha, "Intrusion detection using deep belief networks," in IEEE National Aerospace and Electronics Conference (NAECON), 2015.

[14] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), 2016.

[15] Y. Li, R. Ma, and R. Jiao, "A hybrid malicious code detection method based on deep learning," International Journal of Security and Its Applications, 2015.

[16] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "DL4MD: A Deep Learning Framework for Intelligent Malware Detection," in International Conference on Data Mining (DMIN), 2016.

[17] G. Tzortzis and A. Likas, "Deep belief networks for spam filtering," in IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2007.

[18] G. Mi, Y. Gao, and Y. Tan, "Apply stacked auto-encoder to spam detection," in International Conference in Swarm Intelligence, 2015.

[19] M. Stevanovic and J. M. Pedersen, "An efficient flow-based botnet detection using supervised machine learning," in IEEE International Conference on Computing, Networking and Communications (ICNC), 2014.

[20] S. Ranjan, Machine learning based botnet detection using real-time extracted traffic features, Google Patents, 2014.

[21] B. Rahbarinia, R. Perdisci, A. Lanzi, and K. Li, "Peerrush: mining for unwanted p2p traffic," in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2013.

[22] A. Feizollah and e. al, "A study of machine learning classifiers for anomaly-based mobile botnet detection," in Malaysian Journal of Computer Science, 2013.

[23] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throwaway traffic to bots: detecting the rise of DGA-based malware," in USENIX Security Symposium, 2012.

[24] T. Chakraborty, F. Pierazzi, and V. Subrahmanian, “Ec2: Ensemble clustering and classification for predicting android malware families,” IEEE Transactions on Dependable and Secure Computing, 2017.

[25] C. Annachhatre, T. H. Austin, and M. Stamp, “Hidden Markov models for malware classification,” Journal of Computer Virology and Hacking Techniques, 2015.

[26] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, “On the feasibility of online malware detection with performance counters,” in ACM SIGARCH Computer Architecture News, 2013.

[27] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, “A comparison of machine learning techniques for phishing detection,” in ACM Proceedings of the Anti-Phishing Working Groups, 2007.

[28] Marihart, D.J., Communications technology guidelines for EMS/SCADA systems, Power Delivery, IEEE Transactions on, Volume: 16, Issue: 2, April 2001 Pages:181 - 188

[29] Qiu, B., Gooi, H.B., Web-based SCADA display systems (WSDS) for access via Internet Power Systems, IEEE Transactions on , Volume: 15 , Issue: 2 , May 2000 Pages:681 - 686

[30] Bruce, A.G., Reliability analysis of electric utility SCADA systems, Power Systems, IEEE Transactions on, Volume: 13, Issue: 3, Aug. 1998 Pages: 844 – 849

[31] Ghoshal, K., Distribution automation: SCADA integration is key, Computer Applications in Power, IEEE, Volume: 10 , Issue: 1 , Jan. 1997 Pages:31 - 35

[32] Marcuse, J., Menz, B., Payne, J.R., Servers in SCADA applications, Industry Applications, IEEE Transactions on , Volume: 33 , Issue: 5 , Sept.-Oct. 1997 Pages:1295 - 1299

[33] Luque, J., Gomez, I., The role of medium access control protocols in SCADA systems, Power Delivery, IEEE Transactions on , Volume: 11 , Issue: 3 , July 1996 Pages:1195 - 1200

[34] Luque, J., Gomez, I., Escudero, J.I., Determining the channel capacity in SCADA systems using polling protocols, Power Systems, IEEE Transactions on , Volume: 11, Issue: 2 , May 1996 Pages:917 - 922

[35] Sciacca, S.C., Block, W.R., Advanced SCADA concepts, Computer Applications in Power, IEEE , Volume: 8 , Issue: 1 , Jan. 1995 Pages:23 - 28

[36] Flowers, T., Houle, B., Refzer, J., Ramanathan, R., Routing SCADA data through an enterprise WAN, Computer Applications in Power, IEEE, Volume: 8 , Issue: 3 , July 62 1995 Pages:40 – 44

[37] Bernard, J.-P., Durocher, D., An expert system for fault diagnosis integrated in existing SCADA system, Power Systems, IEEE Transactions on , Volume: 9 , Issue: 1 , Feb. 1994 Pages:548 – 554

[38] G. Xiang, J. Hong, C. P. Rose and, L. Cranor, "Cantina+: A feature-rich machine learning framework for detecting phishing web sites," ACM Transactions on Information and System Security (TISSEC), 2011.

[39] G. Apruzzese, M. Marchetti, M. Colajanni, G. Gambigliani Zoccoli, and A. Guido, "Identifying malicious hosts involved in periodic communications," in IEEE International Symposium on Network Computing and Applications (NCA), 2017.

[40] F. S. Tsai, "Network intrusion detection using association rules," International Journal of Recent Trends in Engineering, 2009.

[41] Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory, ACM Trans. Inf. Syst. Secur. 3 (2000), no. 4, 262–294.

[42] Dhruba Kumar Bhattacharyya and Jugal Kumar Kalita, Network anomaly detection: A machine learning perspective, CRC Press, 2013.

[43] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and J¨org Sander, Lof: identifying density-based local outliers, ACM Sigmod Record, vol. 29, ACM, 2000, pp. 93–104.

[44] KDD Cup 1999 Data, Kdd cup 1999 data, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, Web. Nov 2014.

[45] IBM. IBM Security Services 2014 Cyber Security Intelligence Index, Ibm statistics on data breach epidemic, http://www-935.ibm.com/services/us/en/it-services/security-services/data-breach/, April 2014, Web. 16, Nov 2014.

[46] CBSNews. CBS Interactive, Fbi director james comey on threat of isis, cybercrime, http://www.cbsnews.com/news/fbi-director-james-comey-on-threat-of-isis-cybercrime/, October 2014, Web. 16, Nov 2014.

[47] Aleksandar Lazarevic, Levent Ert¨oz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava, A comparative study of anomaly detection schemes in network intrusion detection, SDM, SIAM, 2003, pp. 25–36.

[48] Shawn Ostermann, Tcptrace-official homepage, http://www.tcptrace.org/, Web. Nov 2014.

[49] Robin Sommer and Vern Paxson, Outside the closed world: On using machine learning for network intrusion detection, Security and Privacy (SP), 2010 IEEE Symposium on, IEEE, 2010, pp. 305–316.

[50] Vladimir Svetnik, Andy Liaw, Christopher Tong, J Christopher Culberson, Robert P Sheridan, and Bradley P Feuston, Random forest: a classification and regression tool for compound classification and qsar modeling, Journal of chemical information and computer sciences 43 (2003), no. 6, 1947–1958.

[51] MIT Lincoln Laboratory: Communication Systems, Cyber Security: Cyber Systems, and Technology, Darpa intrusion detection evaluation, http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/1998data.html, Web. Nov 2014.

[52] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali-A Ghorbani, A detailed analysis of the kdd cup 99 data set, Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009, 2009.

[53] S. Cheung. An efficient message authentication scheme for link state routing. In: 13th Annual Computer Security Applications Conference, 1997.

[54] DOE. 21 steps to to improve cyber security of SCADA networks. http://www.ea.doe.gov/pdfs/21stepsbooklet.pdf

[55] GAO-04-628T. Critical infrastructure protection: challenges and efforts to secure control systems. Testimony Before the Subcommittee on Technology Information Policy, Intergovernmental Relations and the Census, House Committee on Government Reform. March 30, 2004. http://www.gao.gov/new. items/d04628t.pdf

[56] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. J. of the ACM 33(4):792–807, 1987.

[57] Bruce, A., Lee, R., A framework for the specification of SCADA data links, Power Systems, IEEE Transactions on , Volume: 9 , Issue: 1 , Feb. 1994 Pages:560 - 564

[58] Ghoshal, K., Douglas, L.D., GUI display guidelines drive winning SCADA projects, Computer Applications in Power, IEEE, Volume: 7, Issue: 2, April 1994 Pages:39–42

[59] Gaushell, D.J., Block, W.R., SCADA communication techniques and standards, Computer Applications in Power, IEEE, Volume: 6, Issue: 3 , July 1993 Pages:45 - 50

[60] Chan, E.-K., Ebenhoh, H., The implementation and evolution of a SCADA system for a large distribution network, Power Systems, IEEE Transactions on , Volume: 7 , Issue: 1, Feb. 1992 Pages:320 - 326

[61] Pollet, J., Developing a solid SCADA security strategy, Sensors for Industry Conference, 2002. 2nd ISA/IEEE, 19-21 Nov. 2002 Pages: 148 - 156

[62] Duo Li, Serizawa, Y., Mai Kiuchi, Concept design for a Web-based supervisory control and data-acquisition (SCADA) system, Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. IEEE/PES , Volume: 1 , 6-10 Oct. 2002 Pages:32 - 36 vol.1

[63] Wu Sitao, Qian Qingquan, Using device driver software in SCADA systems, Power Engineering Society Winter Meeting, 2000. IEEE, Volume: 3, 23-27 Jan. 2000 Pages: 2046 - 2049 vol.3

[64] Chen Qizhi, Qian Qinquan, The research of UNIX platform for SCADA, Power Engineering Society Winter Meeting, 2000. IEEE, Volume: 3, 23-27 Jan. 2000 Pages: 2041 - 2045 vol.3

[65] Ebata, Y., Hayashi, H., Hasegawa, Y., Komatsu, S., Suzuki, K., Development of the Intranet-based SCADA (supervisory control and data acquisition system) for power system, Power Engineering Society Winter Meeting, 2000. IEEE, Volume: 3, 23-27 Jan. 2000 Pages: 1656 - 1661 vol.3

[66] Chen Qizhi, Optimization of a SCADA system based on client/server mode, Power System Technology, 1998. Proceedings. POWERCON '98. 1998 International Conference on, Volume: 2, 18-21 Aug. 1998 Pages: 1237 - 1240 vol.2

[67] Medida, S., Sreekumar, N., Prasad, K.V., SCADA-EMS on the Internet, Energy Management and Power Delivery, 1998. Proceedings of EMPD '98. 1998 International Conference on, Volume: 2, 3-5 March 1998 Pages: 656 - 660 vol.2

[68] Bruce, A.G., Lee, R., A framework for the specification of SCADA data links, Power Industry Using Machine Leaning Computer Application Conference, 1993. Conference Proceedings, 4-7 May 1993 Pages: 117 - 121

APPENDIX A

PREPROCESSING

```
# @Author:  Ali Hasan Dakheel
# @Project: Cyber Attack Detection in remote terminal units of
SCADA using ML.

import nltk
from nltk.stem import WordNetLemmatizer
import re
import itertools
import html.parser
import sys

lemmatizer = WordNetLemmatizer()

re_http = r'https?.*?( |$)'
re_at = r'@.*?( |$)'
re_number = r'[0-9]+[.]?'
re_space = r'\s+'
re_remove_punctuation = r'[^a-zA-Z ]'
re_hash= r'#.*?( |$)'

apostrophe_lookup = {"'s": ' is', "'re": " are", "'d" :" would",
"'ll":" will", "'ad": " had", "'t":" it", "'m": " am", "'ve":"
have", "won't":"will not" ,"shan't":"shall not", "n't":" not"}

def lemmatizing(sent):
    l = []
    for i in sent.split():
        l.append(lemmatizer.lemmatize(i))
    return l

file = open('Dataset/Gas_Pipeline_Dataset.csv', 'r')

## training Y transfer list ##
## according to "training_attack_types.txt" ##
classlist=['guess_passwd.', 'nmap.', 'loadmodule.', 'rootkit.',
'warezclient.', 'smurf.', 'portsweep.', 'neptune.', 'normal.',
'spy.', 'ftp_write.', 'phf.', 'pod.', 'teardrop.',
'buffer_overflow.', 'land.', 'imap.', 'warezmaster.', 'perl.',
'multihop.', 'back.', 'ipsweep.', 'satan.']

'''
ynew=['r2l', 'probe', 'u2r', 'u2r', 'r2l', 'dos', 'probe', 'dos',
'normal', 'r2l', 'r2l', 'r2l', 'dos', 'dos', 'u2r', 'dos', 'r2l',
'r2l', 'u2r', 'r2l', 'dos', 'probe', 'probe']
output=['dos','u2r','r2l','probe','normal']
'''
```

```
## First, train normal and abnormal ##
ynew=['abnormal', 'abnormal', 'abnormal', 'abnormal', 'abnormal',
'abnormal', 'abnormal', 'abnormal', 'normal', 'abnormal',
'abnormal', 'abnormal', 'abnormal', 'abnormal', 'abnormal',
'abnormal', 'abnormal', 'abnormal', 'abnormal', 'abnormal',
'abnormal', 'abnormal', 'abnormal']
output=['abnormal','normal']

def basic_cleaning(string):
    sent = html.parser.unescape(string)
    sent = re.sub( re_http, "", sent )
    sent = re.sub( re_at, "", sent )
    sent = re.sub( re_number, "", sent )
    sent = re.sub( re_space, " ", sent )
    sent = re.sub(re_remove_punctuation, " ", sent)
    sent = ''.join( ''.join( s )[:2] for _, s in
itertools.groupby(sent))
    for a in apostrophe_lookup.keys():
        if a in sent:
            sent= sent.replace(a, apostrophe_lookup[a])

    words = sent.split()
    new = []
    for x in words:
        if x.startswith('#'):
            x  = x.replace("#","")
            x_ = " ".join( re.findall( '[A-Z][^A-Z]*', x ) )
            if x_:
                x = x_
            x = x.split( " " )
            for ww in x:
                new.append(ww)
        else:
            new.append(x)
    sent = " ".join(new)
    return sent.lower().strip()

def preprocessing_stage1(string):
    sent=basic_cleaning(string)
    sent = lemmatizing(sent)
    return sent

def preprocessing_for_pos_tags(features):
    tweet_tags = []
    for t in features:
        tags_words = nltk.pos_tag( nltk.word_tokenize( t ) )
        tags = [x[1] for x in tags_words]
        tag_str = " ".join( tags )
        tweet_tags.append( tag_str )

    return tweet_tags

def preprocessing_stage2(string):
```

```
    sentence = re.sub( re_http, "URLHERE", string )
    sentence = re.sub( re_at, "MENTIONHERE", sentence )
    sentence = re.sub( re_hash, "HASHTAGHERE", sentence )
    sentence = re.sub( re_number, "", sentence )
    sentence = re.sub( re_space, " ", sentence )
    sentence = re.sub( re_remove_punctuation, " ", sentence )
    return sentence.lower()
file.close()
```

APPENDIX B

RANDOM FOREST ALGORITHM

```
# @Author:  Ali Hasan Dakheel
# @Project: Cyber Attack Detection in remote terminal units of
SCADA using ML.
from __future__ import absolute_import
from __future__ import unicode_literals
from __future__ import division

import numpy as np
import scipy.sparse as sp
from sklearn.ensemble.forest import BaseForest
from sklearn.tree import DecisionTreeClassifier
from sklearn.utils import check_random_state, check_array
from sklearn.preprocessing import OneHotEncoder


def bootstrap_sample_column(X, n_samples=None, random_state=1234):

    random_state = check_random_state(random_state)
    if n_samples is None:
        n_samples = X.shape[0]

    return random_state.choice(X, size=n_samples, replace=True)


def uniform_sample_column(X, n_samples=None, random_state=1234):
    """uniform_sample_column
    Sample a column uniformly between its minimum and maximum
value.
    Parameters
    ----------
    X : np.ndarray (n_samples,)
        Column to sample.
    n_samples : int
        Number of samples to generate. If `None` then generate
        a bootstrap of size of `X`.
    random_state : int
        Seed to the random number generator.
    Returns
    -------
    np.ndarray (n_samples,):
        Uniformly sampled column.
    """
    random_state = check_random_state(random_state)
    if n_samples is None:
        n_samples = X.shape[0]

    min_X, max_X = np.min(X), np.max(X)
```

```
    return random_state.uniform(min_X, max_X, size=n_samples)


def generate_synthetic_features(X, method='bootstrap',
random_state=1234):
    """generate_synthetic_features
    Generate a synthetic dataset based on the empirical
distribution
    of `X`.
    Parameters
    ----------
    X : np.ndarray (n_samples, n_features)
        Dataset whose empirical distribution is used to generate
the
        synthetic dataset.
    method : str {'bootstrap', 'uniform'}
        Method to use to generate the synthetic dataset.
`bootstrap`
        samples each column with replacement. `uniform` generates
        a new column uniformly sampled between the minimum and
        maximum value of each column.
    random_state : int
        Seed to the random number generator.
    Returns
    -------
    synth_X : np.ndarray (n_samples, n_features)
        The synthetic dataset.
    """
    random_state = check_random_state(random_state)
    n_features = int(X.shape[1])
    synth_X = np.empty_like(X)
    for column in xrange(n_features):
        if method == 'bootstrap':
            synth_X[:, column] = bootstrap_sample_column(
                X[:, column], random_state=random_state)
        elif method == 'uniform':
            synth_X[:, column] = uniform_sample_column(
                X[:, column], random_state=random_state)
        else:
            raise ValueError('method must be either `bootstrap` or
`uniform`.')

    return synth_X


def generate_discriminative_dataset(X, method='bootstrap',
random_state=1234):
    """generate_discriminative_dataset.
    Generate a synthetic dataset based on the empirical
distribution
    of `X`. A target column will be returned that is 0 if the row
is
    from the real distribution, and 1 if the row is synthetic. The
```

```
    number of synthetic rows generated is equal to the number of
rows
    in the original dataset.
    Parameters
    ----------
    X : np.ndarray (n_samples, n_features)
        Dataset whose empirical distribution is used to generate
the
        synthetic dataset.
    method : str {'bootstrap', 'uniform'}
        Method to use to generate the synthetic dataset.
`bootstrap`
        samples each column with replacement. `uniform` generates
        a new column uniformly sampled between the minimum and
        maximum value of each column.
    random_state : int
        Seed to the random number generator.
    Returns
    -------
    X_ : np.ndarray (2 * n_samples, n_features)
        Feature array for the synthetic dataset. The rows
        are randomly shuffled, so synthetic and actual samples
should
        be intermixed.
    y_ : np.ndarray (2 * n_samples)
        Target column indicating whether the row is from the
actual
        dataset (0) or synthetic (1).
    """
    random_state = check_random_state(random_state)
    n_samples = int(X.shape[0])

    synth_X = generate_synthetic_features(
        X, method=method, random_state=random_state)
    X_ = np.vstack((X, synth_X))
    y_ = np.concatenate((np.ones(n_samples), np.zeros(n_samples)))

    permutation_indices =
random_state.permutation(np.arange(X_.shape[0]))
    X_ = X_[permutation_indices, :]
    y_ = y_[permutation_indices]

    return X_, y_


class RandomForestEmbedding(BaseForest):
    """Very similar to sklearn's RandomTreesEmbedding;
    however, the forest is trained as a discriminator.
    """
    def __init__(self,
                 n_estimators=10,
                 criterion='gini',
                 max_depth=5,
```

```
                 min_samples_split=2,
                 min_samples_leaf=1,
                 min_weight_fraction_leaf=0.,
                 max_features='auto',
                 max_leaf_nodes=None,
                 bootstrap=True,
                 sparse_output=True,
                 n_jobs=1,
                 random_state=None,
                 verbose=0,
                 warm_start=False):
        super(RandomForestEmbedding, self).__init__(
                base_estimator=DecisionTreeClassifier(),
                n_estimators=n_estimators,
                estimator_params=("criterion", "max_depth",
"min_samples_split",
                                  "min_samples_leaf",
"min_weight_fraction_leaf",
                                  "max_features",
"max_leaf_nodes",
                                  "random_state"),
                bootstrap=bootstrap,
                oob_score=False,
                n_jobs=n_jobs,
                random_state=random_state,
                verbose=verbose,
                warm_start=warm_start)

        self.criterion = criterion
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf
        self.min_weight_fraction_leaf = min_weight_fraction_leaf
        self.max_features = max_features
        self.max_leaf_nodes = max_leaf_nodes
        self.sparse_output = sparse_output

    def _set_oob_score(self, X, y):
        raise NotImplementedError("OOB score not supported in tree
embedding")

    def fit(self, X, y=None, sample_weight=None):
        self.fit_transform(X, y, sample_weight=sample_weight)
        return self

    def fit_transform(self, X, y=None, sample_weight=None):
        X = check_array(X, accept_sparse=['csc'], ensure_2d=False)

        if sp.issparse(X):
            # Pre-sort indices to avoid that each individual tree
of the
            # ensemble sorts the indices.
            X.sort_indices()
```

```
        X_, y_ = generate_discriminative_dataset(X)

        super(RandomForestEmbedding, self).fit(X_, y_,

sample_weight=sample_weight)

        self.one_hot_encoder_ = OneHotEncoder(sparse=True)
        if self.sparse_output:
            return
self.one_hot_encoder_.fit_transform(self.apply(X))
        return self.apply(X)

    def transform(self, X):
        if self.sparse_output:
            return
self.one_hot_encoder_.fit_transform(self.apply(X))
        return self.apply(X)
```

APPENDIX C

(SVM) SUPPORT VECTOR MACHINE ALGORITHM

```
# @Author:  Ali Hasan Dakheel
# @Project: Cyber Attack Detection in remote terminal units of
SCADA using ML.
import sys
import numpy as np
import argparse
from collections import Counter
from scipy.sparse import csr_matrix
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

def tokenize(sentence, grams):
    words = sentence.split()
    tokens = []
    for gram in grams:
        for i in range(len(words) - gram + 1):
            tokens += ["_*_".join(words[i:i + gram])]
    return tokens

def build_counters(filepath, grams, text_row, class_row):
    """Reads text from a TSV file column creating an ngram count
    Args:
        filepath, the tsv filepath
        grams, the n grams to use
        text_row, row in the tsv file where the text is stored
        class_row, row in the tsv file where the class is stored
    """
    counters = {}

    with open(filepath) as tsvfile:
        n = 0
        for line in tsvfile:
            row = line.split('\t')
            try:
                c = int(row[class_row])
            except:
                print(n)
                print (class_row)
                print (row[class_row])
                print (filepath)
                sys.exit(0)
            n = n + 1
            # Select class counter
            if c not in counters:
                # we don't have a counter for this class
                counters[c] = Counter()
            counter = counters[c]
```

```
            # update counter
            counter.update(tokenize(row[text_row], grams))

    return counters

def compute_ratios(counters, alpha=1.0):
    """Computes the log-likelihood ratios for each class
    """
    ratios = dict()

    # create a vocabulary - a list of all ngrams
    all_ngrams = set()
    for counter in counters.values():
        all_ngrams.update(counter.keys())
    all_ngrams = list(all_ngrams)
    v = len(all_ngrams)  # the ngram vocabulary size

    # a standard NLP dictionay (ngram -> index map) use to update
the
    # one-hot vector p
    dic = dict((t, i) for i, t in enumerate(all_ngrams))

    """
    # for each class we calculate a ratio (r_c)
    for c in counters.keys():
        p_c = np.full(v, alpha)
        counter = counters[c]
        for t in all_ngrams:
            p_c[dic[t]] += counter[t]
        # normalize (l1 norm)
        p_c /= np.linalg.norm(p_c, ord=1)  # = p_c / sum(p_c)
        ratios[c] = np.log(p_c / (1 - p_c))
    """

    # sum ngram counts for all classes with alpha smoothing
    # 2* because one gets subtracted when q_c is calculate by
subtracting p_c
    sum_counts = np.full(v, 2*alpha)
    for c in counters:
        counter = counters[c]
        for t in all_ngrams:
            sum_counts[dic[t]] += counter[t]

    # calculate r_c for each class
    for c in counters:
        counter = counters[c]
        p_c = np.full(v, alpha)     # initialize p_c with alpha
(smoothing)

        # add the ngram counts
        for t in all_ngrams:
            p_c[dic[t]] += counter[t]
```

```
        # initialize q_c
        q_c = sum_counts - p_c

        # normalize (l1 norm)
        p_c /= np.linalg.norm(p_c, ord=1)  # = p_c / sum(p_c)
        q_c /= np.linalg.norm(q_c, ord=1)

        # p_c = log(p/|p|)
        p_c = np.log(p_c)
        # q_c = log(not_p/|not_p|)
        q_c = np.log(q_c)

        # Subtract log(not_p/|not_p|
        ratios[c] = p_c - q_c

    return dic, ratios, v


def count_lines(data_file):
    """Counts the number of lines in a file
    """
    lines = 0
    with open(data_file) as f:
        for line in f:
            lines += 1
    return lines


def load_data(data_path, text_row, class_row, dic, v, ratios,
grams):
    """Create Train or Test matrix and Ground Truth Array
    """
    n_samples = count_lines(data_path)
    # n_r = len(ratios)
    classes = ratios.keys()
    Y_real = np.zeros(n_samples, dtype=np.int64)

    # One X (sample) matrix and binary Y (truth) per class
    X = dict()
    Y = dict()
    data = dict()
    indptr = [0]
    indices = []
    for c in classes:
        Y[c] = np.zeros(n_samples, dtype=np.int64)
        data[c] = []

    with open(data_path) as tsvfile:
        n = 0
        for line in tsvfile:
            row = line.split('\t')
            try:
```

```
                t = int(row[class_row])
            except:
                print (n)
                print (class_row)
                print (data_path)
                sys.exit(0)

            for c in classes:
                Y[c][n] = int(c == t)
            Y_real[n] = t

            ngrams = tokenize(row[text_row], grams)
            for g in ngrams:
                if g in dic:
                    index = dic[g]
                    indices.append(index)
                    for c in classes:
                        # X[c][n][idx] = ratios[c][idx]
                        data[c].append(ratios[c][index])
            indptr.append(len(indices))

            n += 1

    for c in classes:
        X[c] = csr_matrix((data[c], indices, indptr),
shape=(n_samples, v),
                          dtype=np.float32)

    return X, Y, Y_real


def save_counters(counters, filepath):
    """Writes counters to files: ngram \t count \n
    """
    for cl in counters:
        cpath = filepath + '.' + str(cl)
        counter = counters[cl]

        with open(cpath, 'w') as fout:
            for ngram, count in counter.most_common():
                count = str(count)
                fout.write(ngram + '\t' + count + '\n')
    sys.exit(0)


def main(train, test, text_row, class_row, ngram,
debug_counters=None):
    print('Building computing ratios')
    ngram = [int(i) for i in ngram]
    counters = build_counters(train, ngram, text_row, class_row)

    if(debug_counters):
        save_counters(counters, debug_counters)
```

```
    dic, ratios, v = compute_ratios(counters)
    classes = ratios.keys()
    print (v)

    print('Loading Data')
    Xs_train, Ys_train, y_train = load_data(train, text_row,
class_row,
                                            dic, v, ratios, ngram)
    Xs_test, Ys_test, y_true = load_data(test, text_row,
class_row,
                                         dic, v, ratios, ngram)

    print('Training Classifiers')
    print('classes in train: %d' % len(set(y_train)))
    print('classes in test: %d' % len(set(y_true)))

    svms = dict()
    for c in classes:
        svms[c] = LinearSVC()
        svms[c].fit(Xs_train[c], Ys_train[c])

    print('Testing')
    preds = dict()
    for c in classes:
        preds[c] = svms[c].decision_function(Xs_test[c])

    # not calculate the argmax
    pred = np.zeros(len(y_true))
    for idx in range(0, len(y_true)):
        max_score = float('-inf')
        for c in classes:
            if preds[c][idx] > max_score:
                max_score = preds[c][idx]
                pred[idx] = c

    # finally the scores
    acc_svm = accuracy_score(y_true, pred)
    print('SVM: %f' % (acc_svm,))
```

APPENDIX D

(KNN) K-NEAREST NEIGHBORS ALGORITHM

```
# @Author:  Ali Hasan Dakheel
# @Project: Cyber Attack Detection in remote terminal units of
SCADA using ML.
from collections import defaultdict
import sys
from numpy import *


class kNN(object):

    def __init__(self, dataset, k, feature):
        """Create a new nearest neighbor classifier.
        dataset - a list of data points. Each data point is an (x,
y) pair,
                  where x is the input and y is the label.
        k - the number of neighbors to search for."""
        # Note how we don't have to do any initialization!
        # Once we have a dataset, we can immediately get
predictions on new values.
        self.dataset = dataset
        self.k = k
        self.feature = feature

    def distance(self, p1, p2):
        # print shape(p1), shape(p2)
        # print type(p1), type(p2)
        if self.feature == 'input_space':
            return self.euclidean_distance(p1, p2)
            # return self.euclidean_distance(p1, p2)
        elif self.feature == 'fds':
            return self.fds_distance(p1, p2)

    def fds_distance(self, p1, p2):
        ''' Fourier descriptors Euklidean distance '''
        p1 = p1 / p1[1]
        p2 = p2 / p2[1]
        p1_abs = absolute(p1)
        p2_abs = absolute(p2)
        # print shape(p1_abs)
        # sys.stdin.read(1)
        dist = linalg.norm(p1_abs - p2_abs)  # Euklidean distance
        return dist

    def euclidean_distance(self, img1, img2):
        # Since we're using NumPy arrays, all our operations are
automatically vectorized.
        distance = sum((img1[:][:] - img2[:][:]) ** 2)
```

```
        #        print distance
        #        print shape(distance)
        return distance

    def L3(self, img1, img2):
        # Since we're using NumPy arrays, all our operations are
automatically vectorized.
        distance = (sum((img1[:][:] - img2[:][:]) ** 3) ** (1 /
3))
        #        print distance
        #        print shape(distance)
        return distance

    def get_majority(self, votes):
        '''For convenience, we're going to use a defaultdict.
          This is just a dictionary where values are initialized
to zero
          if they don't exist. '''
        counter = defaultdict(int)
        for vote in votes:
            # If this weren't a defaultdict, this would error on
new vote values.
            counter[vote] += 1

        # Find out who was the majority.
        majority_count = max(counter.values())
        for key, value in counter.items():
            if value == majority_count:
                return key

    def predict(neighbours, k):
        top_k = [Counter(x[:k]) for x in neighbours]
        predicted_labels = [x.most_common(1)[0][0] for x in top_k]

        return predicted_labels

    # ------------------------------------------------------------

    """
    Finds the optimal value of k using cross validation.
    The value of k with minimum error is the optimal one
    """

    def find_k(neighbours, real_validation_labels,
similarity_measure):
        k_values = []
        error_values = []

        real_validation_labels = list(real_validation_labels)

        """
        Its a convention to start from k = 1 to k = sqrt(N) where
N is the size of training data
```

```
        """
        for k in range(math.ceil(math.sqrt(training_size))):
            k += 1

            predicted_labels = predict(neighbours, k)

            # check accuracy
            acc = accuracy_score(real_validation_labels,
predicted_labels)

            k_values.append(k)
            error_values.append(1 - acc)

        if similarity_measure == 1:
            s = "Cosine Similarity"
        else:
            s = "Euclidean Distance"

        k = k_values[np.argmin(error_values)]

        """ Plotting the Validation Error Curve """

        plt.ylabel('Validation Error', fontsize=14)
        plt.xlabel('K', fontsize=14)
        plt.title("Validation Error Curve using %s" % s,
fontsize=16, color='green')
        plt.plot(k_values, error_values, 'bo--')
        figure = plt.gcf()  # get current figure
        figure.set_size_inches(13, 7)

        plt.savefig("Validation Error Curve using %s.png" % s,
dpi=300)
        plt.clf()

        """
        The value of K which gave minimum validation error is the
optimal value of k
        """
        return k_values[np.argmin(error_values)]
    def classify(self, point):
        # We have to copy the data set list, because once we've
located the best
        # candidate from it, we don't want to see that candidate
again, so we'll delete it.
        candidates = self.dataset[:]

        # Loop until we've gotten all the neighbors we want.
        neighbors = []
        while len(neighbors) < self.k:
            # Compute distances to every candidate.
            distances = [self.distance(x[0], point) for x in
candidates]  # list of arrays INSTEAD of list of floats
            # print distances, type(distances)
```

```
            # sys.stdin.read(1)
            # Find the minimum distance neighbor.
            best_distance = min(distances)
            index = distances.index(best_distance)
            neighbors.append(candidates[index])

            # Remove the neighbor from the candidates list.
            del candidates[index]

        # Predict by averaging the closets k elements.
        prediction = self.get_majority([value[1] for value in
neighbors])
        return prediction
```

APPENDIX E

DATASET TRAINER

```
# @Author:  Ali Hasan Dakheel
# @Project: Cyber Attack Detection in remote terminal units of
SCADA using ML.
import argparse
import cv2
import numpy as np
import os

from os.path import isfile, join
from scipy.io import loadmat, savemat
from math import floor
from datetime import datetime
from random import shuffle, choice

LOWEST_ALLOWED_CHAR = 33
HIGHEST_ALLOWED_CHAR = 126
MAX_ROTATION = 5
STEP = 1
TARGET_IMAGES = 1000
ADDITIONAL = [40, 41, 42, 43, 45, 58, 61]
class Dataset:
    def __init__(self, batch_size=32):
        self._train_labels = list()
        self._test_labels = list()

        self.batch_size = batch_size
        self._load_dataset()

    def _load_dataset(self):
        self.data = loadmat('Dataset/Gas_Pipeline_Dataset.csv')

    def _append_to_dataset(self, test_data=False):
        if test_data:
            test_data = self.data['dataset'][0][0][1][0][0]
            self.data['dataset'][0][0][1][0][0][0] =
np.append(test_data[0], self._test, axis=0)
            self.data['dataset'][0][0][1][0][0][1] =
np.append(test_data[1], self._test_labels, axis=0)

            self._test_labels = list()

        else:
            train_data = self.data['dataset'][0][0][0][0][0]
            self.data['dataset'][0][0][0][0][0][0] =
np.append(train_data[0], self._train, axis=0)
            self.data['dataset'][0][0][0][0][0][1] =
np.append(train_data[1], self._train_labels, axis=0)
```

```
            self._train_labels = list()
            self._train = list()

    def add_image(self, image, label, test_data=False):
        if len(image) !=
len(self.data['dataset'][0][0][0][0][0][0][0]):
            raise Exception("Data should be an array of length
784")

        reverse_mapping = {kv[1:][0]:kv[0] for kv in
self.data['dataset'][0][0][2]}
        m_label = reverse_mapping.get(ord(label))

        if m_label is None:
            raise Exception("The dataset doesn't have a mapping
for {}".format(label))

        if test_data:
            self._test_images.append(image)
            self._test_labels.append([m_label])
        else:
            self._train_images.append(image)
            self._train_labels.append([m_label])

        if len(self._test_images) >= self.batch_size or
len(self._train_images) >= self.batch_size:
            self._append_to_dataset(test_data)

    def save(self, do_compression=True):
        if len(self._test_images) > 0:
            self._append_to_dataset(test_data=True)

        if len(self._train_images) > 0:
            self._append_to_dataset()

        file_name =
'dataset/Gas_Pipeline_Dataset.csv'.format(str(datetime.now()).repl
ace(' ', '-').replace(':', '-'))
        savemat(file_name=file_name, mdict=self.data,
do_compression=do_compression)

    def add_images_from_files(self, images, label, test_data):
        for img in images:
            self.add_image(img, label, test_data)

def gray_scale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img

def normalize(img):
    img = np.reshape(img, 28 * 28)
    img = img.astype('float32')
```

```
    return img

def rotate_image(img, angle):
    # Calculate center, the pivot point of rotation
    (height, width) = img.shape[:2]
    center = (width // 2, height // 2)

    # Rotate
    rot_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
    img = cv2.warpAffine(img, rot_matrix, (width, height),
flags=cv2.INTER_CUBIC, borderMode=cv2.BORDER_REPLICATE)
    return img

def can_shift(img, i, j):
    shift = True

    if i == -1:
        shift = not np.any(img[0, :])
    elif i == 1:
        shift = not np.any(img[27, :])

    if j == -1 and shift:
        return not np.any(img[:, [0]])
    elif j == 1 and shift:
        return not np.any(img[:, [27]])
    return shift

def shift(img, i, j):
    top, bottom, left, right = 0, 0, 0, 0

    if i == -1:
        img = img[1:, :]
        bottom = 1
    elif i == 1:
        img = img[:27, :]
        top = 1

    if j == -1:
        img = img[:, 1:]
        right = 1
    elif j == 1 and shift:
        img = img[:, :27]
        left = 1

    return cv2.copyMakeBorder(img, top, bottom, left, right,
cv2.BORDER_CONSTANT, value=[0, 0, 0])

def shift_image(img):
    images = list()
    for i in range(-1, 2):
        for j in range(-1, 2):
            if can_shift(img, i, j):
                shifted = shift(img, i, j)
```

```
                images.append(normalize(shifted))
    return images

def extend_image_set(images, count):
    extra = list()
    while len(images) + len(extra) < count:
        extra.append(choice(images))
    images.extend(extra)
    return images

def arguments():
    parser = argparse.ArgumentParser()

    args, unknown = parser.parse_known_args()

    # for i in range(LOWEST_ALLOWED_CHAR, HIGHEST_ALLOWED_CHAR +
1):
    for i in ADDITIONAL:
        directory =
'Dataset/Gas_Pipeline_Dataset.csv'.format(images_path, i)
        if os.path.exists(directory):
            files = [f for f in os.listdir(directory) if
isfile(join(directory, f)) and f != ".DS_Store"]
            images = list()

            for file in files:
                file_path = '{}/{}'.format(directory, file)
                img = cv2.imread(file_path)
                img = gray_scale(img)

                for angle in range(-MAX_ROTATION, MAX_ROTATION +
STEP, STEP):
                    rotated = rotate_image(img, angle)
                    images.extend(shift_image(rotated))

            shuffle(images)
            training_count = floor(len(images) * 0.8)

            print('Dataset for Gas-Pipeline has been trained: {},
Set Length: {}'.format(chr(i), len(images)))
            training_set =
extend_image_set(images[:training_count], round(min_images * 0.8))
            testing_set =
extend_image_set(images[training_count:],  round(min_images *
0.2))

            dataset.add_images_from_files(training_set, chr(i),
False)
            dataset.add_images_from_files(testing_set, chr(i),
True)

    dataset.save()
```

APPENDIX F

CYBER DETECTION SYSTEM

```
# @Author:  Ali Hasan Dakheel
# @Project: Cyber Attack Detection in remote terminal units of
SCADA using ML.
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn
import preprocessing
import RF_Algo
import SVM_Algo
import KNN_Algo
from sklearn.metrics import confusion_matrix
from mpl_toolkits.mplot3d import axes3d

def autolabel(rects,ax):
    """
    Attach a text label above each bar displaying its height
    """
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2.,
1.005*height,'%d' % int(height),ha='center', va='bottom')

def autolabel2(rects,ax,category):
    """
    Attach a text label above each bar displaying its height
    """
    i=0
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2.,
1.005*height,'%s' % category[i],ha='center', va='bottom')
        i+=1

def compare_three_classifiers(accuracies,precisions):

    n= 3;
    ind= np.arange(n)
    width= 0.25

    fig, ax= plt.subplots()
    rects1= ax.bar(ind, accuracies,width,color='g')
    rects2= ax.bar(ind+width, precisions,width, color='b')

    ax.set_title('Comparison of Accuracy and Precision')
    ax.set_xticks(ind + width / 2)
```

```
    ax.set_xticklabels(('RandomForest','SVM','k-Nearest
Neighbour'))
    ax.legend((rects1[0], rects2[0]), ('Accuracy', 'Precision'))
    plt.show()


def
confusion_matrix_voted_classifier(y_test,y_pred,list_of_labels):
    confusion_matrix = confusion_matrix(y_test,y_pred)
    matrix_proportions = np.zeros((3,3))
    for i in range(0,3):
        matrix_proportions[i,:] =
confusion_matrix[i,:]/float(confusion_matrix[i,:].sum())

    names=list_of_labels
    confusion_df = pd.DataFrame(matrix_proportions,
index=names,columns=names)
    plt.figure(figsize=(5,5))
    seaborn.heatmap(confusion_df,annot=True,annot_kws={"size":
12},cmap='gist_gray_r',cbar=False, square=True,fmt='.2f')
    plt.ylabel(r'True categories',fontsize=14)
    plt.xlabel(r'Predicted categories',fontsize=14)
    plt.tick_params(labelsize=12)

    plt.show()

def sample_ploting(sum_tfidf,num_words,FKRA,target,list_labels):
    x1=[]
    y1=[]
    z1=[]
    x2=[]
    y2=[]
    z2=[]
    x3=[]
    y3=[]
    z3=[]

    for i in range(len(target)):
        if target[i]== 0:
            x1.append(sum_tfidf[i])
            y1.append(num_words[i])
            z1.append(FKRA[i])

        elif target[i]== 1:
            x2.append(sum_tfidf[i])
            y2.append(num_words[i])
            z2.append(FKRA[i])

        else:
            x3.append(sum_tfidf[i])
            y3.append(num_words[i])
            z3.append(FKRA[i])
```

```
    fig= plt.figure()
    ax1=fig.add_subplot(111,projection='3d')

    ax1.scatter(x1,y1,z1,c='r',s=50,label=list_labels[0])
    ax1.scatter(x2,y2,z2,c='g',s=60,
marker="*",label=list_labels[1])
    ax1.scatter(x3,y3,z3,c='b',s=50,
marker="^",label=list_labels[2])

    def preprocessing_stage1(string):
        sent = basic_cleaning(string)
        sent = lemmatizing(sent)
        return sent

    def preprocessing_for_pos_tags(features):
        tweet_tags = []
        for t in features:
            tags_words = nltk.pos_tag(nltk.word_tokenize(t))
            tags = [x[1] for x in tags_words]
            tag_str = " ".join(tags)
            tweet_tags.append(tag_str)

        return tweet_tags
    ax1.set_xlabel('Time Factor')
    ax1.set_ylabel('Delta Pipeline PSI')
    ax1.set_zlabel('Pipeline PSI')
    ax1.legend()
    plt.show()
def corpus_distribution(num_of_each_labels,tuple_of_label_name):

    n= 3;
    ind= np.arange(n)
    width= 0.25
    fig, ax= plt.subplots()
    rects1= ax.bar(ind,num_of_each_labels ,width,color='g')

    ax.set_title('CRC Distribution')
    ax.set_xticks(ind)
    ax.set_xticklabels(tuple_of_label_name)
    autolabel(rects1,ax)
    plt.show()
    def classify(self, point):
        # We have to copy the data set list, because once we've
located the best
        # candidate from it, we don't want to see that candidate
again, so we'll delete it.
        candidates = self.dataset[:]

        # Loop until we've gotten all the neighbors we want.
        neighbors = []
        while len(neighbors) < self.k:
            # Compute distances to every candidate.
```

```
            distances = [self.distance(x[0], point) for x in
candidates]  # list of arrays INSTEAD of list of floats
            # print distances, type(distances)
            # sys.stdin.read(1)
            # Find the minimum distance neighbor.
            best_distance = min(distances)
            index = distances.index(best_distance)
            neighbors.append(candidates[index])

            # Remove the neighbor from the candidates list.
            del candidates[index]

        # Predict by averaging the closets k elements.
        prediction = self.get_majority([value[1] for value in
neighbors])
        return prediction

def show_predictions(confidence,category):
    n= len(confidence);
    ind= np.arange(n)
    width= 0.25
    l=[]
    for i in range(len(confidence)):
        l.append(i+1)

    fig, ax= plt.subplots()
    rects1= ax.bar(ind,confidence,width,color='r')
    ax.set_title('Prediction of Attack')
    ax.set_xticks(ind)
    ax.set_xticklabels(tuple(l))
    autolabel2(rects1,ax,category)
    plt.show()

target=[0,1,2,1]
sum_tfidf=[3.4,5.6,4.2,4.7]
num_words=[5,6,7,5]
FKRA=[3.4,6.7,8.4,9.5]
list_labels=['Bad (Attacking)','Intermidiate','Good (None)']

sample_ploting(sum_tfidf,num_words,FKRA,target,list_labels)

num_of_each_labels=[23,45,67]
tuple_of_label_name=('SVM','RF','K-NN')

corpus_distribution(num_of_each_labels,tuple_of_label_name)

accuracies=[85.3,87.4,86.3]
precisions=[76,84,94]

compare_three_classifiers(accuracies,precisions)

confidence=[90,85,87,89]
category=["Good","Good","Intermidiate","Bad"]
```

```
show_predictions(confidence,category)
```

CURRICULUM VITAE

Ali Hasan Dakheel was born in Babil, Iraq on November 25, 1991. He earned a B.Tech Computer Science from the University of Technology Baghdad. He was accepted in the graduate program in 2017-2018 in Electrical and Computer Engineering Altinbas University, Istanbul Campus and will graduate in 2018-2019.