Summer 2015

Intrusion Detection System of industrial control networks using network telemetry

Stanislav Ponomarev
Louisiana Tech University

INTRUSION DETECTION SYSTEM OF INDUSTRIAL CONTROL NETWORKS USING NETWORK TELEMETRY.

by

Stanislav Ponomarev, B.S., M.S.

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

August 2015

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

May 21, 2015
Date

We hereby recommend that the dissertation prepared under our supervision by Stanislav Olegovich Ponomarev

entitled Intrusion Detection System of Industrial Control Networks Using Network Telemetry.

be accepted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Engineering, with concentration in Cyberspace

[illegible]
Supervisor of Dissertation Research

[illegible]
Head of Department

Engineering
Department

Recommendation concurred in:

[illegible]

[illegible]

[illegible]

[illegible]

Advisory Committee

Approved:
[illegible]
Director of Graduate Studies

[illegible]
Dean of the College

Approved:
[illegible]
Dean of the Graduate School

GS Form 13a
(6/07)

ABSTRACT

Industrial Control Systems (ICSs) are designed, implemented, and deployed in most major spheres of production, business, and entertainment. ICSs are commonly split into two subsystems - Programmable Logic Controllers (PLCs) and Supervisory Control And Data Acquisition (SCADA) systems - to achieve high safety, allow engineers to observe states of an ICS, and perform various configuration updates. Before wide adoption of the Internet, ICSs used "air-gap" security measures, where the ICS network was isolated from other networks, including the Internet, by a physical disconnect [1]. This level of security allowed ICS protocol designers to concentrate on the availability and safety of operation of physical systems while decreasing the need for many cyber security implementations. As the price of networking devices fell, and the Internet received global adoption, many businesses became interested in the benefits of attaching ICSs to wide and global area networks. However, since ICS network protocols were originally designed for an air-gapped environment, it did not include any of the security measures needed for a proper operation of a critical protocol that exposes its packets to the Internet.

This dissertation designs, implements, and evaluates a telemetry based Intrusion Detection System (IDS). The designed IDS utilizes aggregation and analysis of the traffic telemetry features to classify the incoming packets as malicious or benign. An IDS that uses network telemetry was created, and it achieved a high classification accuracy, protecting nodes from malicious traffic. Such an IDS is not vulnerable to address or encryption spoofings, as it does not utilize the content of the packets to differentiate between malicious and benign traffic. The IDS uses features of timing and network sessions to determine whether the machine that sent a particular packet and its software is, in fact, a combination that is benign, as well as whether or not it resides on a network that is benign. The results of the experiments conducted for this dissertation establish that such system is possible to create and use in an environment of ICS networks. Several features are recognized and selected as means for fingerprinting the hardware and software characteristics of the SCADA system that can be used in pair with machine learning algorithms to allow for a high accuracy detection of intrusions into the ICS network. The results showed a classification accuracy of at least 95% is possible, and as the differences between machines increase, the accuracy increases too.

TABLE OF CONTENTS

ABSTRACT iii

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER 1 INTRODUCTION 1

1.1 ICS Vulnerabilities 2

1.2 ICS Protocol Security 5

1.3 Intrusion Detection Systems 6

CHAPTER 2 LITERATURE REVIEW 8

2.1 ICS Protocols 8

2.1.1 Modbus 9

2.1.2 DNP3 11

2.2 ICS Attacks 13

2.2.1 SCADA-side Attacks 14

2.2.2 PLC-side Attacks 15

2.3 Intrusion Detection Systems for ICS 16

2.3.1 State Based IDS 16

2.3.2 Rule Based IDS 18

2.3.3 Physical Properties Based IDS 19

2.3.4 Statistical and Machine Learning IDS 20

2.3.5 Network Telemetry and Metadata Based IDS 22

CHAPTER 3 NETWORK TELEMETRY BASED INTRUSION DETECTION SYSTEM 27

3.1 Telemetry Data 29

3.2 Classification Algorithms 33

3.3 Data Acquisition 35

CHAPTER 4 INTRUSION DETECTION OF MALICIOUS SSH CALLS 39

4.1 Experiment 41

4.1.1 Experimental Setup 41

4.2 Results 42

CHAPTER 5 SESSION INSTANTIATION 46

5.1 TCP Session Flows 46

5.2 Session Duration Based Instantiation 48

5.3 Session Length Based Instantiation 52

5.4 Conclusion 55

CHAPTER 6 OPTIMAL CAPTURE INTERVAL DERIVATION 56

6.1 Minimizing the Training Window 56

6.2 Accuracy Validation Against a Third Machine 60

6.3 Conclusion 67

CHAPTER 7 SOFTWARE VARIANCES AS FEATURE CONTRIBUTORS... 68

7.1 Hop Relationship 72

7.2 Results 73

7.2.1 Insider Classification 73

7.2.2 Outsider Classification 75

7.2.3 Decreasing Capture Intervals 76

7.2.4 Conclusion 78

CHAPTER 8 IDS VERIFICATION OVER MULTIPLE PLATFORM CONDITIONS 79

8.1 IDS Accuracy for Identical Clients 81

8.2 IDS Accuracy for Different Programming Languages 85

8.3 IDS Accuracy for Different Operating Systems 90

8.3.1 Gentoo and Windows Classification 91

8.3.2 Debian And Windows Classification 95

8.3.3 Debian And Gentoo Classification 99

CHAPTER 9 CONCLUSIONS 104

APPENDIX A CODE LISTINGS 107

BIBLIOGRAPHY 110

LIST OF TABLES

Table 2.1: TCP Modbus packet [2] 11

Table 2.2: DNP3 Frame [3] 12

Table 5.1: Generated session counts for varying silence intervals 49

Table 5.2: Classification results of varying silence intervals 50

Table 5.3: Classification results of varying conversation lengths 52

Table 5.4: Classification results of varying conversation lengths 54

Table 6.1: Maximum accuracies and corresponding Capture Times of classifiers for Classification of two clients 59

Table 6.2: Maximum accuracies and corresponding Capture Times of classifiers for Classification of three clients 63

Table 7.1: Accuracies of separating outsider and insider traffic by varying n 71

Table 8.1: 10-Fold validation of selected classifiers on the dataset of two identical machines. 82

Table 8.2: 10-Fold validation of selected classifiers on the dataset of C and Python based clients. 87

Table 8.3: 10-Fold validation for Gentoo and Windows Classification 91

Table 8.4: 10-Fold Validation for Debian and Windows Classification 95

Table 8.5: 10-Fold Validation for Debian and Gentoo Classification 99

LIST OF FIGURES

Figure 3.1: Client-Server Session Graph 28

Figure 3.2: Time delays introduced by spoofing 33

Figure 3.3: Average Response time in a session over 1 day 35

Figure 4.1: Experimental Setup 39

Figure 4.2: Server's normal operation 40

Figure 4.3: SSH RSA handshake 42

Figure 4.4: Packet arrival time differences 43

Figure 4.5: Standard Deviation of packet arrival differences 44

Figure 4.6: Standard Deviation of packet arrival differences during an attack 45

Figure 5.1: Client-Server Session Graph 47

Figure 5.2: Silence Interval Detection session counts 49

Figure 5.3: Silence Interval Detection Accuracy 50

Figure 5.4: Silence Interval Session Duration and class Distributions for 0.3s Silence Interval 52

Figure 5.5: Session Length based extraction counts 53

Figure 5.6: Session Length Extraction Accuracy 53

Figure 5.7: Session Length of 3 Session Duration and class Distributions 55

Figure 6.1: Capture Time Maximization Graph for Classification of two clients.. 57

Figure 6.2: Maximum Accuracy within Tested Window 59

Figure 6.3: Capture Time Maximization Graph for Classification of three clients 61

Figure 6.4: Maximum Accuracy within Tested Window for Classification of three clients 64

Figure 6.5: Distribution of Features for the testing dataset. Blue is malicious, Red is benign. 65

Figure 6.6: Distribution of Features for the training dataset. Blue is malicious, Red is benign. 66

Figure 7.1: Average Response time in a session over 1 day 71

Figure 7.2: Classifier Accuracy for Insiders 74

Figure 7.3: Classifier Accuracy vs Model Build Time 74

Figure 7.4: Classifier Accuracy for Outsiders 75

Figure 7.5: Classifier Accuracy vs Time to Build Model for Outsiders 76

Figure 7.6: Classifier Accuracy for Insiders for traffic captured over 5-minute window 77

Figure 7.7: Classifier Accuracy vs Time-to-Build Model for Insiders for traffic captured over 5-minute window 77

Figure 8.1: Feature Distribution of the data set for two identical machines. Blue is malicious and red is benign 83

Figure 8.2: Accuracies of Different Capture Intervals 84

Figure 8.3: Model Build Times for Given Capture Intervals 84

Figure 8.4: Feature Distribution of the data set for C and Python based clients. Blue is malicious and red is benign 88

Figure 8.5: Accuracies of Different Capture Intervals 89

Figure 8.6: Model Build Times for Given Capture Intervals 89

Figure 8.7: Accuracy of Gentoo vs Windows classification 92

Figure 8.8: Model Build Time of Gentoo vs Windows classification 93

Figure 8.9: Accuracy of Classifying Debian With a Gentoo vs Windows Training Set 94

Figure 8.10: Accuracy of Debian vs Windows Classification 96

Figure 8.11: Model Build Time of Debian vs Windows classification 97

Figure 8.12: Accuracy of Classifying Gentoo With a Debian vs Windows Training Set 98

Figure 8.13: Accuracy of Debian vs Gentoo Classification 100

Figure 8.14: Model Build Time of Debian vs Gentoo Classification 101

Figure 8.15: Accuracy of Classifying Windows With a Debian vs Gentoo Training Set 102

CHAPTER 1

INTRODUCTION

Industrial Control Systems (ICSs) are designed, implemented, and deployed in most major spheres of production, business, and entertainment. From orchestrating advanced maneuvers of the International Space Station to controlling the speed of a roller coaster, ICSs are able to process complex data and safely perform designed tasks. Safety is a number one priority when dealing with physical devices [1].

ICSs are commonly split into two subsystems - Programmable Logic Controllers (PLCs) and Supervisory Control And Data Acquisition (SCADA) systems - to achieve high safety, allow engineers to observe states of an ICS, and perform various configuration updates. PLCs are small processing systems which are able to modify the behavior of the controlled devices and receive input from the system's sensors. The SCADA system allows engineers to monitor the ICS state and modify its parameters as needed.

The communication between PLCs and SCADA system can be transmitted over a wide variety of protocols and mediums. Originally, ICSs used wired communication methods that utilized serial protocols that were based on RS-232 specification. The medium was a dedicated transmission channel that could be used only by ICS.

1.1 ICS Vulnerabilities

Before wide adoption of the Internet, ICSs used "air-gap" security measures, where the ICS network was isolated from other networks, including the Internet, by a physical disconnect [1]. To perform an attack on an isolated ICS, an attacker would have to gain physical access to the ICS' network, and either penetrate and use the hardware of the ICS to transmit malicious commands, or attach a new node to the ICS network. This level of security allowed ICS protocol designers to concentrate on the availability and safety of operation of physical systems while decreasing the need for many cyber security implementations.

As the price of networking devices fell, and the Internet received global adoption, many businesses became interested in the benefits of attaching ICSs to wide and global area networks. The benefits included engineers gaining an ability to monitor and fix critical problems remotely. Off site engineers were now able to remotely reconfigure ICSs, which gave them more time to work on problem solving when the system malfunctioned. The price of implementing geographically dispersed ICSs, where PLCs and SCADA systems may be miles away from each other, decreased with the spread of the Internet. The critical infrastructure Smart Power Grid design implements one such ICS where each house has its own controller - a smart meter - that transmits usage information to the power company and may turn off the house's power for maintenance or lack of payment [1].

ICSs started to use some of the aspects of business' networks for their purpose. Protocols were created to allow system control of ICS machines. One of the most common protocols used for transmitting ICS data is Modbus. Originally this protocol was created for use over a serial RS-232 communication. To allow for a seamless integration and merger of ICS and business networks, the Modbus protocol was wrapped into a Transmission Control Protocol commonly used in computer networking and became known as Modbus/TCP. Modbus packets could now traverse the Internet and connect ICS nodes that were miles apart without the need to lay expensive copper wiring between those nodes.

Since Modbus was originally designed for an air-gapped environment, it did not include any of the security measures needed for a proper operation of a critical protocol that exposes its packets to the Internet. The attack vectors for attacking PLCs and SCADA systems alike include man-in-the-middle, DoS, spoofing, packet injection, and reconnaissance attacks. However, depending on the direction of packet flow, these attacks may affect PLCs, SCADA, or both. In addition, there exists many different ICS protocols. While some standard ICS protocols were created, different manufacturers choose to use different protocols, which make it difficult to assess possible vulnerabilities for all ICSs [4].

Possible vulnerabilities can result in an attacker performing many tasks from planting a virus, to getting systems' sensitive data or joining a machine to the botnet. Christodorescu says that it is a "game between malicious code writers and researchers working on malicious code detection" [5] when talking about the use of malware and the creation of anti-virus methods, but the same principle applies to many aspects of security, including network security. Networks transmit any data that the nodes provide. The data transmitted can be any sequence of commands that a host machine understands, or it can be a machine code used to avert the underlying machine's logic.

Any Turing-complete machine can run malicious code that is designed to "harm or subvert a system's intended functionality." Applications utilizing such code are known as "malware" [6, 7]. Turing-complete machines include a vast set of devices - from personal computers and cell phones, to machinery automation and utility distribution controllers that ICS represents. All of these devices can communicate over a network, which means they can be infected by malicious code without any user interaction. Non-networked devices can also be compromised by such code through user interaction - connecting it to a computer, inserting a flash drive that contains infected files, or transferring data in any other means.

Stuxnet was one of the largest and most complicated attacks deployed that targeted an industrial control system [8]. Stuxnet was a worm, a malicious application that is capable of replicating and spreading itself over a network, discovered in June 2010. However, unlike most worms, Stuxnet was developed to target ICS networks. To hide the worm from the cyber-security community, the attackers prevented it from inflicting any damage in networks that did not contain specific properties of an ICS network. Once inside a vulnerable computer with no ICS software, the Stuxnet worm could propagate using USB drives or network connectivity. Stuxnet was allowed to duplicate itself only 3 times when propagating over a USB drive, but there was no limit on network propagation [8, 9, 10]. However, when ICS properties were discovered, Stuxnet would present itself to PLCs as a SCADA system and perform man-in-the-middle attacks, reprogramming PLCs to perform different actions, thereby bringing the system to a critical state. Stuxnet penetrated the SCADA system of Iranian nuclear facilities and caused centrifuge motors to change their spinning frequency, which destroyed centrifuges [8].

ICSs have been a target of many cyber-attacks. Unlike attacks of information technology, attacks on ICSs may not only destroy a company's rating and cause financial problems, but may also damage equipment and the environment, as well as present a threat to human lives. When engineers design the logic of ICS controllers, their primary goal is to maintain system stability and safety. However, as ICS controllers get attached to wide and global area networks, they become vulnerable to a multitude of attacks.

1.2 ICS Protocol Security

Network models that are currently implemented allow for easy access to various networked systems through a multitude of mediums, devices, and implementations. Cell phones can communicate with high-end servers or even supercomputers over WiFi, desktops can use cell phone's cellular networks, smart watches can communicate with the same servers by using Bluetooth, and cell phone's connection. Current approaches have allowed widespread adoption of computer networks. Network models create abstraction layers and keep some information such as properties of the sender's connection, sender's architecture and configuration, as well as sender's location, away from applications. Lack of such information reduces the ability to establish the authenticity of the host transmitting the data [11]. Additionally, an attacker can perform data manipulations on the network to achieve penetration of target machines and spoof the authenticity of the transmitted packets [12].

Many protocols have been created since the introduction of Modbus/TCP. Some of them were adopted from the serial communication over copper wires based on RS-232 protocol and just like Modbus, were wrapped in a TCP protocol. Others were specifically designed for use in a new interconnected configuration of ICS networks. However, there are currently no set standard protocol for use in an ICS environment. Many manufactures create their own proprietary protocols that are incompatible with other manufacturers. The ICS market is very segmented, which makes it hard to secure all of the ICSs due to millions of different vulnerabilities being spread across of millions of different protocols [13].

1.3 Intrusion Detection Systems

One of the approaches to securing the communication of network attached ICSs is an Intrusion Detection System (IDS). Such a system operates by monitoring data that is transmitted through the network. At its core, IDSs must be able to alert system engineers when intrusions are detected, however, some IDSs are also able to block the malicious traffic, resulting in an intrusion prevention. The state of the art research of IDSs is covered in Chapter 2. Many current research efforts are attempting to apply the Information Technology (IT) perspective to the problem of securing ICSs. However, ICSs have unique features that prevent proper operation of high-confidence Intrusion Detection Systems. System engineers may sometimes need to execute the same sequences of commands that an attacker may want to execute in order to deal damage to the system. For example, an engineer may want to turn a high-speed fan off for repairs, or an attack may want to turn it off to overheat another part of the control system. Additionally, stability and availability of the ICS is crucial to its operation. Unlike a business network, an unstable ICS may lead to damaged equipment or even loss of life.

This dissertation develops and evaluates a machine learning based IDS that utilizes a unique set of features which allows for a transparent intrusion detection without affecting the stability of underlying ICS. The author hypothesises that such system is possible to create and use in an environment of ICS networks. The features used for traffic classification are obtained by analysis of TCP session flows that will be covered in detail in Chapter 3. By utilizing TCP session flow it becomes possible to differentiate between the machines that created TCP packet even if two packets that originate from different machines contain identical signatures. This means that many techniques that help the attackers to mask themselves from many IDSs will not work. In addition, the developed IDS will not need any exception handling for critical situations such as a component shutdown by an engineer. The IDS identifies packets' source which is a combination software and hardware as malicious or benign. Benign packets are allowed to traverse the network while malicious packets are logged and dropped from the network. The developed IDS is evaluated in Chapters 4-8.

CHAPTER 2

LITERATURE REVIEW

Stuxnet was a wakeup call to many ICS engineers. Securing industrial control systems from cyber attacks became a top priority [1]. Unlike business networks, an attack on ICS can result not only in information leak, but a physical damage. As a result, security of cyber-physical systems is now a world-wide concern, and various intrusion detection, attack mitigation, and attack obfuscation techniques are being researched.

Various security implementations for ICSs have been suggested. Sridhar *et al.* [14] covers many problems specific to these systems while targeting the security of electric power grid. Security, forensics, accountability, as well as resilience, and dependency of the information communication channels are important in the realm of ICS security. In order to better understand the problem of securing industrial control systems, the state-of-the-art research, as well as current vulnerabilities and attack vectors of the Industrial Control Systems will be covered in the following sections.

2.1 ICS Protocols

While much work has been put into the research of Intrusion Detection systems, research efforts also target the creation of new communication protocols that allow for safe communication between ICS nodes, avoiding many different attack vectors.

Mo *et al.* [15] suggests the use of physical watermarking for ICS protocols. Physical watermarking works similar to cryptography, however, it is based on the fact that many components of an ICS work in tandem, and injecting a specific signal at one point of a control system will propagate changes throughout all of the sensors used by the ICS. Mo was able to achieve sensor verification by injecting a crafted noisy signal and observing the sensor changes reported. Sensors are deemed malicious if the reported change did not match the change that has to be created by the input signal.

Creating new, secure protocols allows mitigation of many cyber attacks. However, the implementation of these protocols often requires replacement of the controllers currently used as well as inability to use controllers that rely on different protocols for their security. There are many manufacturers that create incompatible industrial system controllers. The creation of new control protocols results in even greater segmentation of the controller market. To prevent segmentation, controller manufacturers instead use the protocols that are compatible with more controllers. Many cyber-physical system engineers also prefer well-evaluated protocols due to their reliability [16]. Reliability is the second highest priority while designing most ICSs. Modbus and DNP3 are a few of the well established, reliable protocols on the market.

2.1.1 Modbus

Modbus is a control system serial data transmission protocol which originated in 1979. The first Modbus implementations used bit oriented frame formats that were transmitted over RS-232 serial communication interface. Modern variants of Modbus have different implementations that allow it to be transmitted over both wired and wireless communications including ISM, WiFi, SMS, GPRS, and TCP transmissions [2]. This research will concentrate on Modbus/TCP implementation.

Table 2.1 shows a structure of a Modbus packet sent over TCP network protocol. Transaction ID is 2 bytes long, and is similar to TCP's sequence number. Transaction ID is used to maintain a proper data synchronization between different nodes of the network. Protocol ID is used to identify the variation of Modbus protocol. TCP implementation sets this value to zero. The length field specifies the length of the unit address, function code, and data sections. Unit address represents the address of the responding PLC or the PLC being queried in situations where information is routed using the PLCs. Otherwise, this value is set to 255 and is not used. The function code is used for main device operation and can include functions to read and write data to and from the PLC's inputs and output, to upload new logic, or to get current logic in binary representation, and to perform various diagnostic procedures such as getting event logs, device ID numbers, and hardware exception status. Depending on the function, data field may be filled by the communicating SCADA system, or the PLC during its response. [17]

All Modbus implementations are big-endian, however, different implementations utilize different data types as value representations. Some of the more common data types are floating points, 32-bit integers, or bit-fields. TCP implementation uses 8-bit characters. [18]

Modbus protocol does not implement any security features required for authentication, integrity checks, or anonymity. Once inside the control system network, an attacker may inject any packet that follows similar protocol fields into the network, which will then be accepted by PLCs or the SCADA system as legitimate packets.

Table 2.1: TCP Modbus packet [2]

Name	**Length (bytes)**	**Function**
Transaction ID	2	Synchrinization data
Protocol ID	2	Type of carrier used
Length	2	Bytes in this frame
Unit address	1	For configurations where PCSs route packets
Function code	1	ID of a function to be executed
Data	variable	Result of the function executed

2.1.2 DNP3

Distributed Network Protocol (DNP) is a set of control system protocols, commonly used in utilities' control sector. DNP3 is the latest revision of the DNP protocol [19]. Unlike Modbus, DNP3 is more robust, supports protocol encapsulation, some integrity checking, and routing support. As most utilities can be represented as geographically dispersed control systems, DNP3 was designed to support communications over a range of various mediums and devices.

Unlike Modbus, DNP3 is separated into the Application Layer, Pseudo Transport Layer, and Link Layer. The information transmitted over DNP3 protocol is split into frames by the Application layer. The DNP3 frame is represented in Table 2.2. Each frame begins with two bytes of a sync pattern, similar to an Ethernet frame. The length field specifies the length of the remainder of the frame in bytes, ignoring the CRC length. Link control field is used to coordinate link usage over multiple devices. DNP3 protocol was designed to operate as a peer-to-peer model, and carries 2 bytes of source and destination addresses, allowing different nodes to route the information appropriately. For every 16 bytes of data, a separate CRC value is calculated and recorded in the CRC field. Data payload can be a maximum of 250 bytes, which requires 16 different CRC values.

Table 2.2: DNP3 Frame [3]

Name	Length (bytes)	Function
Sync pattern	2	Frame start pattern
Frame length	1	Length in bytes of the rest of the frame
Link Control	1	Link layer control data
Dst. Address	2	Destination device ID
Src. Address	2	Source device ID
CRC	variable	Cyclic redundancy check value for every 16 bytes of data
Data	variable	Data transmitted by the application layer

The data section of the DNP3 protocol is written by the Application layer, and user-side software, which will vary with different implementations. The data section will ultimately include the function code and the data passed to or returned from a proper function. Some implementations support basic encryption, and authentication fields. Any DNP3 field can be spoofed by software, preventing the server from being able to determine whether the packets are arriving from an authorized location [20, 21].

That is one of the reasons DNP3 allows for encryption based algorithms, to prevent illegal access and enforce data integrity [22]. However, no matter how secure a cryptographic function is, it can be attacked with at least a brute force attack [23]. Encryption, while effective in deterring most attackers, can be broken in many different ways. Hashes can be reversed, users can be tricked into giving their passwords to phishing sites, and weak passwords can be guessed [24].

While supporting some security measures, DNP3 shows itself to be more robust than the Modbus, however, all of the security measures must be implemented by the application used, which offloads security research on the developers of a specific application. While encryption may deter low level attackers, the ICSs that use the DNP3 protocol are still vulnerable to more complex attacks, which is unacceptable due to the high risk of ICS malfunction.

2.2 ICS Attacks

Sayegh *et al.* [25] documented a wide range of attacks on ICS that include Denial of Service (DoS), replay, cryptographic, and fragmentation attacks. One type of DoS attack is a CPU shutdown attack, where an attacker transmits a command to shut down the CPU of a PLC, requiring an engineer to perform a hard-reset of the device. While CPU is shut down, the PLC can not communicate or process any information, which results in a denial of service.

Zhu *et al.* created a taxonomy of attacks on ICS in [12]. While most of the attacks require access to the ICS LAN, malicious applications such as backdoors and trojans can grant that access to an attacker since most of the SCADA is run on PC architecture machines which commonly run Windows operating system. To better understand the threat model of the attacks on ICS, threats can be separated into two categories - SCADA and PLC side attacks.

2.2.1 SCADA-side Attacks

SCADA attacks target the packets transmitted to the SCADA system. SCADA systems are commonly run on PCs [26], which are vulnerable to any attacks on their operating system. An attacker may acquire protocol-specific SCADA identifiers to mask themselves as an engineer to inject PLC values into the network that do not reflect the current network state, and to stall the SCADA machine to prevent the SCADA engineer from knowing the state of the ICS [12].

Response injection attacks require the knowledge of the underlying control process, but can drastically change the representation of the control system at the engineer's monitoring system [27]. A maliciously crafted response packet may result in a display of a stable system while the actual physical process is in a critical state.

Since SCADA systems are commonly run on PCs, a vast range of attacks on the PC of the SCADA system can result in a successful penetration. The Symantec threat report [28] states that more than 250 million different PC malware applications were detected in 2013. Even if the malware does not target ICS as its main goal, a malware present on a SCADA system can result in system malfunction.

If an attacker uses malware to penetrate the PC a SCADA system runs on, response injection attacks can also be created without the need of transmitting a malicious packet over the network, since the packet can be sent over a loop-back interface. Other attacks include memory space attacks on the allocated memory of the SCADA software which can result in full control of the displayed ICS state by the attacker.

2.2.2 PLC-side Attacks

PLC attacks target the packets being sent to the controller. Once communication with the controller is established without a proper IDS, an attacker has an ability to shut down the CPU, disable memory protection and perform code injection. This allows an attacker to modify the code that the PLC is running, get the state of a system that PLC is monitoring, find packet structure that the PLC sends to SCADA, overwrite the values that PLC is reporting to SCADA, and overwrite some or all logic that is used to control the ICS. While some of these attacks may result in information leakage, others can damage the physical system being controlled or misrepresent the system state to the monitoring engineer [29].

Wei *et al.* [30] developed False Data Injection attacks that target PLCs and do not require attackers to know the whole underlying topology of the ICS. Such attacks can result in a predictable malicious outcome, and damage the underlying physical system such as the substation.

Denial of Service (DoS) attacks covered in [27] pose a danger to both the control process and the information flow to the engineers. DoS attacks prevent PLCs from responding to queries and often need a physical (power cycle) reset in order to mitigate the attack. DoS attacks are present in many forms. Some utilize low throughput of the devices and overwhelm them with the amount of transmitted packets. Others use ill timed or maliciously crafted packet injections that exploit bugs in the controller's code and become unresponsive.

Command injection attacks covered in [27] allow attackers to mask themselves as engineers and send control requests to the PLCs. Similar to the response injection attacks covered in Section 2.2.1, command injections utilize protocol vulnerabilities to allow the attackers to send maliciously formed packets to the PLCs to shift the state of the underlying control process.

2.3 Intrusion Detection Systems for ICS

2.3.1 State Based IDS

Carcano *et al.* [31] proposes a state-based intrusion detection system, which listens for the ICS network traffic, maintains an ICS image based on that traffic, and monitors for a set of state anomalies. In [32], a model based IDS is developed, where the communication model using Modbus protocol is analyzed and an alarm is raised when packets with certain fields set are transmitted.

Long et al. proposed methods of mitigating denial of service attacks that originate from either local or wider area networks by modeling stochastic process of packet delay jitter and loss [33]. Mitigating attacks on industrial control systems can be much harder due to the physical nature of damage occurring during an attack. However, some research into PLC shadowing and data duplication has been done [26]. Attack mitigation techniques include the use of firewalls, recommended by the National Institute of Standards and Technology [13], and some protocol modifications to prevent man-in-the-middle attacks [34].

In [19], a state based IDS was developed for Modbus and DNP3 protocols that could identify multi-packet attacks. While each individual packet of such an attack can be considered benign, a unique pattern of received packets may send the systems in a critical state. Critical state is defined as a state of an ICS that will result in damage.

The latest research by Fovino *et al* [35] developed a critical state based intrusion detection system that is able to prevent damage to physical plants. For an IDS to follow the state changes of an ICS, protocols used to transmit the data must be parsed, and state changes recorded. Fovino's latest IDS can parse Modbus and DNP3 protocols. Several filtering and monitoring techniques were developed that can describe unwanted states of the ICS. However, there is still no single solution that can guarantee the security of an ICS. While a state-based IDS will monitor the state of the system being secured, some measures to prevent the attackers from modifying the code run on PLCs have to be implemented and some of the control packets have to be blocked. To achieve this, an IDS provides a packet language that can describe signatures of unwanted Modbus and DNP3 packets

The state-based intrusion detection system was evaluated in terms of accuracy and performance. Since the predicate condition will always be evaluated, the accuracy of IDS depends on the system capturing every packet and being able to maintain a synchronous state representation of the underlying ICS. To test the efficiency of the IDS, SCADA-to-PLC communications were simulated by transmitting 40 read requests, 50 write requests, and 10 special requests. The IDS was preconfigured with a set of 2000 rules. The system was able to handle up to 1.215 Mb/s of traffic without losing any alerts. When traffic was higher than 1.215 Mb/s, both the packets and alerts loss increased linearly. [31]

Goldenberg *et al.* [36] developed an IDS that was able to achieve high accuracy of detecting response and command injection attacks into the Modbus/TCP protocol by developing an algorithm to construct an ICS model based on Deterministic Finite Automaton (DFA). DFA can be constructed due to the periodic and deterministic nature of an ICS network and Modbus protocol.

2.3.2 Rule Based IDS

Morris *et al.* [27] wrote a plugin to allow Snort to monitor Modbus over TCP traffic and apply intrusion detection rules to it. In their research Snort acted as a rule based firewall that could filter Modbus packets based on a set of rules. Snort is a multi-use tool for network analysis, intrusion detection, and penetration testing [37]. Snort can utilize various algorithms for intrusion detection such as signature based intrusion detection - where some features within the packets' data match signatures of packets transmitted by an intruder, statistical anomaly based intrusion detection - detection using stochastic behavior of a previously captured traffic, as well as stateful protocol analysis - a technique similar to state-based ICS IDS created by [31] and described above.

In [38], an additional set of rules specific to Modbus protocol is generated. Rules are applied to the network packet flow using a Snort plugin that allows researchers to parse Modbus/TCP traffic. Morris *et al.* developed a set of 50 rules that can detect malicious activity. Each rule is designed to prevent a specific attack, however no evaluation of the rules is reported.

Parvania *et al.* [39] created an IDS that utilizes the "hybrid control" rule set to detect intrusion in Modbus or DNP3 protocols. The researchers created a set of special rules that can detect whether the system is operating in a maintenance state or state of normal operation, and switch the intrusion detection rule set accordingly. This approach resolves the problem of activating IDS while performing maintenance on an ICS network. In their research, a significant timing delay was noticed while monitoring packet arrival during normal operation of the ICS and an incoming attack.

2.3.3 Physical Properties Based IDS

Wallace *et al.* [40] suggests designing an IDS custom-tailored to a specific ICS, where the underlying physical properties of the objects being controlled manifest themselves as features of the ICS that can be used to distinguish between the states of ICS normal operation, and attacks on a given ICS.

By utilizing principal component analysis in [41], high accuracy state classification of power grid was achieved. The reduced feature set was compared to new power grid states using Hotelling's T^2 value. If the value was too high, then the new feature set could not exist and was determined to be malicious.

Recent research efforts [42] discovered that method execution timing of the PLC code can be used to determine the authority of the running code. This is a unique property of cyber-physical systems that allowed Zimmer *et al.* to achieve tracking of the execution process. The execution timings are compared to preset bounds, resulting in a code execution time signatures that can be validated.

Similar to Wallace *et al.*, research effort led by Valenzuela *et al.* [43] used principal component analysis of the power flow history generated by a Monte Carlo simulation of the power system. The researchers tested their IDS on an IEEE 24-bus and 118-bus reliability test systems. For the majority of the data injection attacks the IDS was able to achieve an accuracy of more than 90%, while in some instances it reached an accuracy of 99.8%.

2.3.4 Statistical and Machine Learning IDS

Mantere et al. [44] states that machine learning IDS can be very useful in a deterministic network such as an ICS network. Unlike typical business networks, an ICS network has a specific periodic packet flow that contains little to no noise during its normal operation. Mantere et al. proposes the use of throughput, IP address, average packet size, timing, flow direction, as well as payload data as features used for machine learning.

In their further work in [45] analyzed many features listed in their original research, including network timing features - data that is critical to the research of this dissertation. Mantere et al. was not able to detect useful behavior in timing features to detect anomalies. However, the presented research did not target any attacks on the network. The research concluded that ICS networks overall have many anomalies present in the traffic due to misconfiguration of the hardware.

Most recent research by Mantere *et al.* [46] focuses on creating a complementary network security monitoring using Self-Organizing Maps as a means of machine learning. Their approach targets restricted IP networks and does not use any network timing features, but uses packet data instead. The research concludes that deterministic properties of ICS networks make machine learning a viable tool for network anomaly detection.

Gao et al. [47] used a Neural Network to classify the ICS network traffic as normal and abnormal based on the operation of MSU SCADA Security Laboratory waster tank control system. The experiment resulted in a 100% accuracy classification of negative false data injection, 95% for positive false data injection, and 84.9% for a random data response injection. Unfortunately, the Neural Network developed achieved only 12.1% accuracy for a replay attack.

Yoon *et al.* [48] developed a framework that operates on multi-core systems and allows real time intrusion detection. The developed framework uses one of the cores of a multi-core system to operate while protecting the processes running on the other cores of such system. The underlying IDS uses timing execution data and trace trees to create a statistical profile of the other working cores. By utilizing core operation data, researchers were able to achieve high accuracy of code intrusion detection.

Visumathi *et al.* [49] used a new Fuzzy C-Means clustering and Genetic Algorithm to create classification based IDS. By using KDD cup'99 data set, researchers were able to successfully train the classifiers to be able to detect intrusions. The research concluded that the use of new machine learning algorithms improves intrusion detection accuracy over previously used methods.

2.3.5 Network Telemetry and Metadata Based IDS

The Internet Protocol (IP) commonly utilizes Ethernet frames to forward packets between multiple nodes of the network until they reach their final destination. To achieve this, Ethernet frame headers contain the source and destination media access control (MAC) addresses. While IP addresses are being assigned to different machine interfaces by network administrators or self-configuration protocols, MAC addresses are uniquely assigned to each network interface during the manufacturing process of the interface's controller circuit [50].

Both IP and MAC addresses can be spoofed by software, preventing the server from ascertaining whether the packets are arriving from an authorized location [20, 21]. Spoofing is one of the reasons security-critical algorithms implement encryption to prevent illegal access and enforce data integrity [22]. Encryption, while effective in deterring most attackers, can be broken, and no matter how secure a cryptographic function can be, it can be attacked with at least a brute force attack [23]. Hashes can be reversed like in Stuxnet [10], users can be tricked into giving their passwords to a phishing site, and weak passwords can be guessed [24]. Network telemetry data can be used to detect a network intrusion when an attacker is using a different machine or even a different software of the authenticated machine.

There have been various studies in anomaly detection which use data mining and machine learning facilities to detect anomalies [51]. NetMine is a data mining application that specializes in understanding traffic data correlations and interactions [52]. While implementing methodologies similar to the ones used in this research effort, these studies focused on feature generation to improve traffic quality and network stability rather than network security.

In their works, Erman *et al.* [53] were able to successfully cluster similar packet types by analyzing transport layer statistics. By using K-Means and DBSCAN algorithms they were able to successfully identify protocols being used without extracting the data from packets [53]. Sheng *et al.* showed that it is possible to detect host spoofing by analyzing statistical fluctuations in received signal strength of the packets transmitted over wireless networks [21].

Alexander *et al.* [54] proposed to secure critical infrastructures of industrial control systems by merging the Interface for Metadata Access Points (IF-MAP) protocol with specification-based intrusion detection protocols. IF-MAP protocol is an XML based protocol that can be used to transmit events about possible network intrusions to either humans or machines responsible for ICS security. The Metadata Access Point aggregates the network metadata transmitted by MAP clients and can decide whether to raise alarm, store, delete, or ignore given metadata messages. The ICS nodes then become the carriers of the MAP client code, which allows them to analyze the incoming traffic and submit event messages to the access point. To gather the required metadata, each node encompasses a MAP client, a specification-based IDS, and an anti-tampering mechanism. Specification-based IDS utilize a compliance metric that measures a deviation of the current state of a secured device from a specified compliance state. The proposed anti-tampering protocol is meant to run seamlessly over the established communication protocol stack, and is not meant to break any communication for any security events, but simply to raise a flag if a tampered communication was detected.

In [55] a data traffic prediction model was built based on autoregressive moving average of data's time series. The intrusions were observed over a 2.4GHz wireless link. Researchers were able to achieve a detection ratio of above 90% by utilizing timing signals used for radio frequency communication. Their IDS was able to detect a majority of the attacks and secure a wireless sensor network.

Later, researchers proposed an IDS similar to this research in that it uses temporal packet data to identify traffic anomalies[56]. Different packet signatures were generated for a given protocol, and probability functions were used to identify if a given packet was expected to arrive to the SCADA system. The paper targets BACnet protocol, but mentions that new protocols can be supported without changing the core functions of the IDS. This approach, however, can produce many false positives when the anomaly happens in the physical domain of the ICS (damaged plant, broken wire, low pressure, etc.), and the engineers try to reprogram PLCs to mitigate the problem.

The developed IDS presents means of detecting malicious traffic over both wired and wireless networks. The IDS detects address spoofings that originate from insider and outsider traffic, and it does not generate false positives during critical states of the supervised ICS. Though the detection approach presented here may not withstand the dynamics of a typical enterprise LAN, the approach will be beneficial in the detection of spoofed hosts in control system LANs. Control system LANs are unique in that hosts generally communicate in set intervals set by the polling protocol utilized.[36] The detection scheme developed for this research effort is able to determine when communication is initiated and maintained outside of these intervals and, upon detection, will alert on a possible intrusion. Furthermore, with the use of open source tools that automate the attack process against control systems [57], this work proves to be fruitful as it can distinguish between malicious and benign packets.

The detection approach utilizes Wireshark, Tcpdump, and Weka applications for data aggregation and classification. Wireshark is a software network analyzer which captures network traffic and displays it in real time. This analyzer utilizes the libpcap library to capture network packets. It also allows system administrators to save all of the received packets for future analysis and extract useful information about these packets. Wireshark was used in this research effort to extract packet arrival times into a comma separated values format used to generate graphs and interpret the data [58].

Waikato Environment for Knowledge Analysis (WEKA) is an application developed by the Waikato university of New Zealand that is used to help researchers utilize machine learning algorithms [59]. WEKA can be used to analyze arbitrary data sets using a variety of machine learning algorithms. More details about used algorithms can be found in Section 3.2. WEKA commonly uses ARFF formatted files as means to store all of the features related to the classification of the data.

Tcpdump is a command-line tool developed as a front-end to the libpcap library [60]. Tcpdump allows researchers to capture network traffic in a file that can be processed by libpcap or Wireshark. The captured file format is known as the pcap file, and contains entries of metadata for every packet received during the capture process as well as all of the data contained in captured packets. The following chapter will utilize these tools to develop and test a Telemetry based Intrusion Detection System.

CHAPTER 3

NETWORK TELEMETRY BASED INTRUSION DETECTION SYSTEM

The designed IDS utilizes aggregation and analysis of the traffic telemetry features to classify the incoming packets as malicious or benign. Data used for design and evaluation of the IDS in the following chapters was generated by Conpot - a contol system honeypot project aimed to simulate an industrial control system network to attract possible intruders [61]. The SCADA part of the ICS is implemented in Python and C using the modbus protocol stack - pymodbus and libmodbus. Conpot was set to simulate a control system network that has two Siemens SIMATIC S7-200 PLCs. The simulation environment was chosen due to its ability to simulate the CPU of a PLC. Even though the timing of an actual PLC will differ from a simulated one, the classification depends on relative timing differences between server and clients, therefore classification results will not be skewed by simulation of PLCs.

A telemetry based IDS uses the understanding of session flow in a networked server-client model (Figure 3.1). Experimental scenarios include variations of a benign SCADA system and a malicious attacker's machine using different hardware and software combinations. These systems were also tested while having different network distances between each other. The origin of control system traffic can change the amount of hops between the nodes as determined by the network's routing algorithms. Increased hop length is more likely to introduce packet delays and dropped packets. The delays and packet counts are used as features for the machine learning environment. Possible attacks, as well as benign traffic, can originate from within the control system Local Area Network (LAN), within the corporate LAN, or within the Internet. Control System LAN traffic can be the result of a normal operation, a benign modification by the system engineer, or a malicious packet injection by an attacker. ICS traffic originating from the corporate LAN can be the result of a malicious attack from an insider, or any benign control and maintenance operation. However, the Department of Homeland Security recommends not to use corporate LANs as a means of entering the ICS network. The two should be separated by at least a firewall [1].

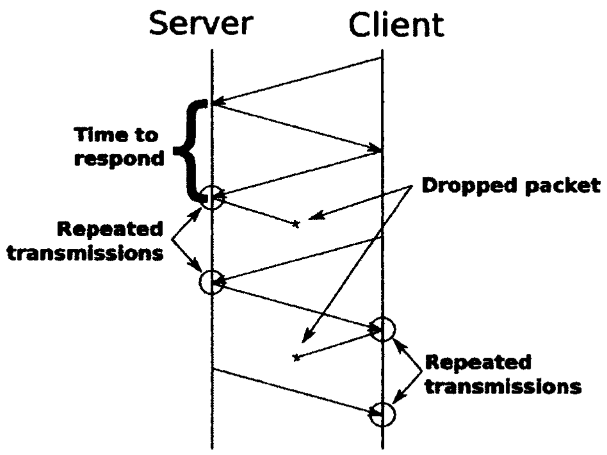

Figure 3.1: Client-Server Session Graph

The goal of Network Telemetry based IDS is to protect PLCs and the underlying physical plant from malicious activity - unauthorized access and control of PLC hardware. Therefore, this IDS is implemented as a standalone device that monitors traffic between the PLCs and the rest of the network. This IDS can be launched on the same machine as the SCADA System to prevent response injection attacks, or on a dedicated machine that can shield PLCs from command injection, false data injection, or some denial of service attacks. The IDS must operate at every point of entry for a potential malicious traffic, therefore, if the secured ICS is localized - only one machine running this IDS is needed, however, multiple machines are required to secure distributed systems. Once the intrusion is detected, malicious packets are removed from the network and saved for future analysis by the system engineers who are informed about the detected anomaly.

An IDS that uses network telemetry can be created and it can achieve a high classification accuracy, protecting nodes from malicious traffic. Such an IDS will not be vulnerable to address or encryption spoofings, as it does not utilize the content of the packets to differentiate between malicious and benign traffic; rather, it uses features of timing and network sessions to determine whether the machine that sent a particular packet and its software is, in fact, a combination that is benign, as well as whether or not it resides on a network that is benign.

3.1 Telemetry Data

A list of features was created by analyzing a total of 838,818 packets generated by the SCADA communication with the honeypot over a period of forty-eight hours. While the selected period may not be enough to capture all of the network variations for a business network due to weekly changed in the business' network users, the period of two days provides plenty of information about the ICS network as the system is deterministic and usually operates on a period of information flow that is in the order of seconds [4]. The gathered data is the result of capturing the traffic from designated benign and malicious machines to the PLCs emulated by Conpot using libpcap - a library created for capturing live networking data. Both malicious and benign machines ran the same code to achieve similar patterns of packet transmissions, which represent situations when both an attacker and an engineer want to execute the same code but for a different purpose - for example, an engineer wants to power a saw after the maintenance, and an attacker wanting to power a saw during the maintenance, while workers are in close contact with the saw. The transmission of similar functions is done to test the accuracy of the IDS for similar packet patterns. Different functions can have unique transmission patterns which are much easier to detect. The packet class was identified by packets' source IP address. This information was only used for classifier evaluation and IP addresses were not selected as features during the classification process. The following features were selected and their appropriate labels are used within this document:

- time it takes the client to respond to server's message: avgTimeToRespond.
- amount of client-side dropped packets: totalClientRetransmissions.
- amount of server-side dropped packets: totalServerRetransmissions.
- time between the repeated packet transmissions when packet drops happen: avgClientRetransmissionTime, and avgServerRetransmissionTime.
- Session duration and the amount of packets present in the session: conversationLength, and conversationDuration.

Figure 3.1 shows the features listed above in a communication session graph. After the data was captured, the features described above were extracted from the recorded pcap file, and converted to an ARFF format that can be used with a machine learning environment. Refer to the end of Chapter 2 for more information about these files.

A TCP latency model described in [62] was used to determine a set of telemetry features for the IDS. While Cardwell *et al.* published their model 15 years ago, it is still appropriate for modelling short-lived TCP connections such as connection used by Modbus/TCP protocol [63]. The features were selected based on their variability from different machines, as well as the level of difficulty for the attackers to modify those features. The level of difficulty was obtained by judging the amount of time and money the attacker would have to spend to control these features.

The time to respond feature includes round trip time for the communication path as well as processing delays introduced by the client's machine. While an attacker can introduce delays to his code, speeding up packet delivery times requires relocation and/or an upgrade of the attacking machine. Moreover, the attacker would have to know the exact patterns of the captured features the IDS is looking for, which can only be achieved by attacking the machine that runs the IDS or obtaining the original training set. Additionally, an attacker would have to know that the IDS utilizes those features for protection. The IDS works transparently, and aside from dropping malicious packets, does not modify ICS network. The features of the amount of dropped packets on either side of communication relates to network congestion and the features of delays between those repetitions provide information about delays introduced by network schedulers. The combination of these two feature sets allow the classifiers to extract information about the network used for communication. The attacker can change these values if they have control of all of the nodes between the malicious and benign machines. Their other option would be to relocate the malicious machine to a network path that matches the benign path in its congestion. Figure 3.1 shows the relationship of these features.

The selected features are protocol independent and are based on the data that an attacker has little to no control over. For example, an attacker may be able to introduce delays into the packet flow of their machine, but the amount of dropped packets is largely dependent on the state of the network between client and server. Introduced delays would also have to be adjusted to match the current value of the IDS system, which would require attacking the IDS system first and acquiring the data. In addition, if the attacker's system is not fast enough to match the delays of the benign software, the only way to speed it up would be to upgrade their system.

Due to the nature of the telemetry data if hardware, software, and network combination can be matched by an attacker to those used by a benign machine, little to no anomalies will be detected. However, such matching requires vast knowledge of the hardware, software, and network performances of the benign machine. In addition, different attack software used to spoof MAC and IP addresses can introduce different telemetry signatures. Figure 3.2 shows an average delay between packet arrival on two similar machines; one uses benign SCADA software to query the PLCs, another uses a metasploit plugin to spoof mac and IP address and poll the PLCs afterwards. If the attacker uses the SCADA machines for an attack, Network Telemetry based IDS is one of a few IDS that can detect an intrusion, as long as some packet transmitting software was changed, for example, an added backdoor.

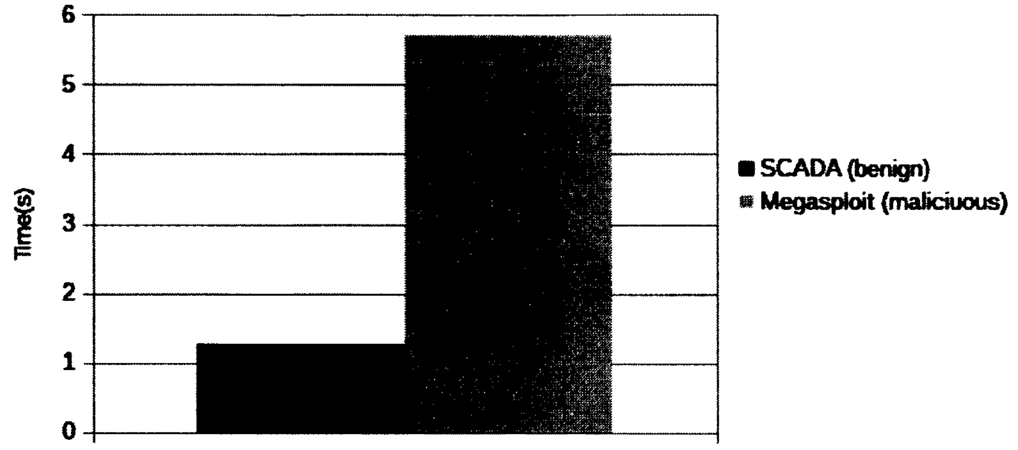

Figure 3.2: Time delays introduced by spoofing

3.2 Classification Algorithms

In order to achieve the highest accuracy, many algorithms were tested. Some qualities of the data set such as noise and size were taken into account when choosing classifiers. Aside from individual classifiers, boosting was used as a supervisory algorithm to reduce the bias of other classifiers. Boosting is an ensemble learning algorithm that aims to create a strong classifier out of many weak ones. Bayesian classifiers' performance is linearly scaled with the data set and can be used in a time critical code. Bagging is also known as bootstrap aggregating, and is an ensemble machine learning algorithm that can improve accuracy of its base algorithm. For this research, REPTree algorithm was used as a base machine learning algorithm. REPTree uses information variance to build a decision tree, and then prunes it using reduced-error pruning. A set of bagging-aided classifiers was chosen because they perform well with training sets containing large noise [64]. Several decision tree based classifiers were also used due to a flow-like dependence of the outcome of the classification on the features. The following paragraphs describe the algorithms used in this research with details on their characteristics and implementation.

Naïve Bayes classifier assumes that all the features are independent of each other and follows a Bayesian probabilistic model:

$$p(C|F_0, F_1, ..., F_n) = \frac{p(C) \cdot p(F_0, F_1, ..., F_n|C)}{p(F_0, F_1, ..., F_n)}$$

where C is the class of dataset, $F_0, ..., F_n$ is a set of features, and $p()$ is a probability function.[65]

Multinomial model of the Bayesian classification, however, is dependent on the frequency of reappearing features in the data-set. This allows for a higher classification accuracy of data with some repeating patterns. [66]

Simple Logistic classifier utilizes a binary logistic regression to describe an outcome in only two possible classes. It takes a set of features and applies regression analysis to create classification parameters. [67]

Ripple-Down Rule learner generates a default rule and creates an exception list using weighted error rates. Then exceptions are rated and filtered to remove conditions that fit multiple exceptions. [68]

A *Decision Stump* generates a single-feature condition that results in a binary classification. This results in a high speed classification, but lacks accuracy in a high-dimensional orthogonal data.[69]

J48 is an implementation of *C4.5* machine learning algorithm. This algorithm generates a decision tree based on the information gain ratio if the classification was split on a given tree node. [70]

3.3 Data Acquisition

Classification accuracy of Network Telemetry based IDS highly depends on processing times introduced by computer hardware, software, and the network communication paths between the benign machines, malicious machines, and the server. Telemetry based measurements are highly dependent on the amount of hops between the client and server. As nodes are separated by a larger amount of hops, added network systems introduce different delays into packet propagation. The IDS carries two classification profiles - for insider traffic and for outsider traffic. Figure 3.3 shows how the delay increases with the amount of hops.

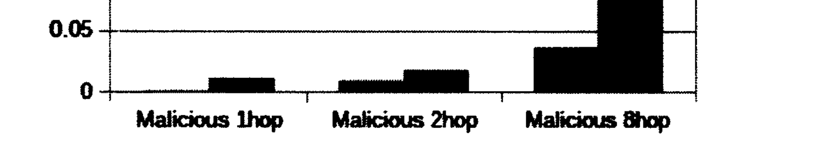

Figure 3.3: Average Response time in a session over 1 day

Differentiating between insider and outsider traffic is trivial as an outsider's delays are an order of magnitude larger than the insider's (Figure 3.3). If an outside attack is further from the PLCs than the system engineer's outside point of entry, differentiating between an attack and normal control packets becomes trivial as well. However, when benign and malicious traffic originates from the same amount of hops to the PLCs, the system's hardware is the one that introduces the most important delays - hardware and software delays become signatures that can be used for traffic classification. Most of the tested classifiers performed better when analyzing outsider traffic than the same classifiers analyzing the insider data due to different network delays present in different network paths used for communication. The results show that the network path adds a significant amount of data to the used feature set to differentiate between two machines with a high accuracy.

The IDS is designed to notify system engineers of malicious traffic. For this research malicious traffic is defined as any traffic directed to the PLCs that are not originated from the designated SCADA system. To achieve classification, the IDS must receive all of the network packets sent to the PLCs. The IDS utilizes libpcap [71] to capture all the traffic transmitted to the PCLs over the modbus protocol. For this research Conpot was emulating Modbus/TCP protocol on port 502. The system maintains a Floating Delay Separation Boundary (FDSB) which separates traffic fingerprinting methods to outsider and insider classification engines. This is done to improve the accuracy of classifying traffic originating from local to the ICS machines, versus the traffic originating from machines on the Internet. For example, after analyzing Netgear WNDR4500, D-Link DI-604, and Linskys WRT300N routers the delay feature range is under 5 milliseconds while the communication utilizes local area network, but it can be several orders of magnitude larger if the communication path connects distant nodes on the Internet. However, not all of the traffic originating from a distant node can be malicious (as in the example of an engineer using remote access to troubleshoot an ICS at the remote site). Separation must be established in order to compare local paths with local paths and distant paths with distant paths. The floating delay separation boundary is defined as

$$FDSB = \frac{\sum_{i=0}^{n} \frac{max_i(I)}{n} + \sum_{i=0}^{n} \frac{min_i(O)}{n}}{2}$$

where I is the set of delay values from the packets originating from the inside of the ICS, O is the set of delay values from the packets originating from the outside of the ICS, max_i and min_i are the ith maximum and minimum values in the given set, and n is the amount of samples to take for the delay value, which is the same for both sums. Having a value of n too high will use older network states which may not take place any more. However, having a value of n too low will result in a lower accuracy of the threshold calculation. Values $n = 5$ were determined experimentally to achieve high accuracy of feature separation while sets of O and I have 50 items. These values were determined by iteratively running the classification over 5-minute intervals for all the captured dataset and determining the combination that results in the best classification. The initial $FDSB$ value was set to 0.025. These values were determined by analyzing 838,818 packets acquired over two days' operation of an ICS. Figure 3.3 shows minimum and maximum response times between two different machines on the same LAN (malicious and benign 1 hop), and the delays introduced by moving the same machine further along the communication path (malicious 1 hop, 2 hops, and 8 hops).

Once the packet is identified as outsider/insider and sent to a proper classifier, its arrival time is recorded and compared against the previous packet that was sent to the same classifier. Inside-originating packets are only compared with telemetry for other inside-originating packets, and outside-originating packets are compared with other outside-originating packets which allows IDS to mark some external traffic as benign as in the case of a system engineer working from home. Training traffic to the PLCs is tagged as inside/outside traffic based on the packet's IP address. Code running on the PLCs is assumed benign and IP address spoofing from the PLCs is not expected.

Several machine learning classifiers have been tested for this IDS as discussed earlier: Naïve Bayes, Multinomial Naïve Bayes, Logistics, REPTree, Bagged REPTree, DecisionStump, Degged DecisionStump, Ridor, and C4.5. Naïve Bayes classifiers were chosen to determine probability models for telemetry data. Bagging modelling of REPTrees was chosen because they perform well with training sets containing large amounts of noise [64].

CHAPTER 4

INTRUSION DETECTION OF MALICIOUS SSH CALLS

The work described in this chapter was published in [72]. To test the proof of concept for this research, an attack model was developed that contained benign and malicious clients trying to communicate with a secure server (Figure 4.1) similar to methodologies of Sheng at al. [21]. However, instead of using received signal strength of the wireless transceiver, this research effort is aiming to detect malicious hosts, not only on a wireless network, but in wired networks and networks of various infrastructures. To achieve this, inter-packet delays were used.

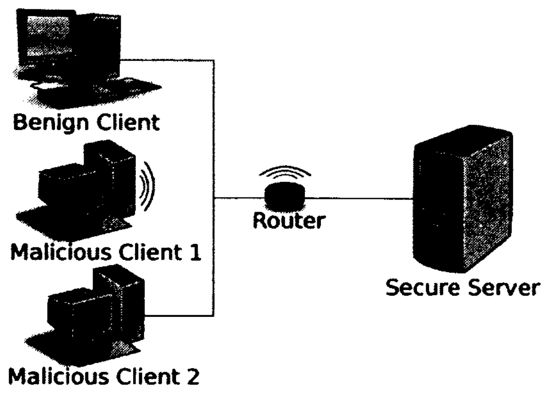

Figure 4.1: Experimental Setup

Unlike the data in an Ethernet packet, the attacker has less control of the timings of the packets being transmitted. All of the server processed packets were captured using wireshark. Those packets were then graphed by time of arrival. Figure 4.2 shows the generated graph. All of the protocols that wireshark was able to identify are separated into individual categories. After observing the SSH handshake pattern, it became evident that a classifier can be made to differentiate between hosts.

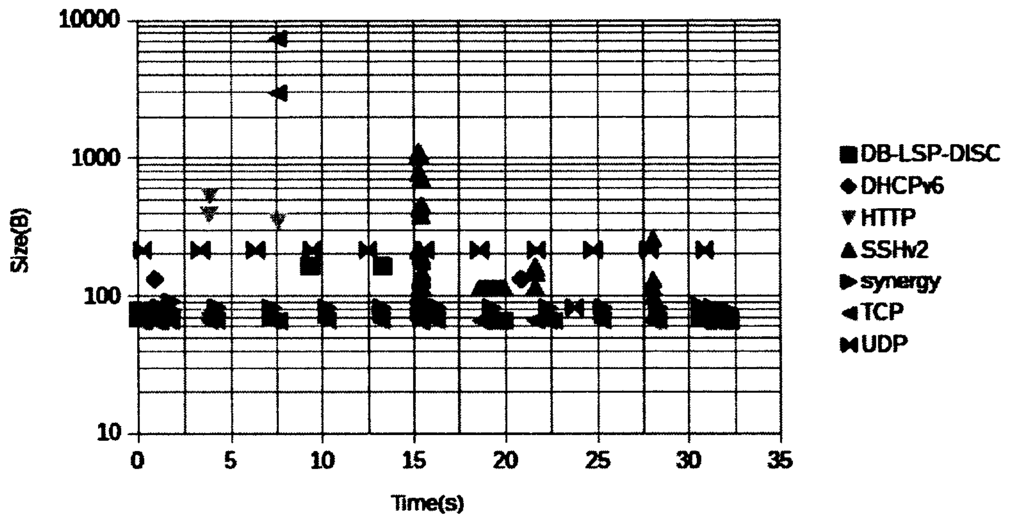

Figure 4.2: Server's normal operation

Looking at the relationships between packets, Erman *et al.* had to use various features to be able to cluster similar packets [53]. However, a node fingerprint can be built simply by observing differences in sequential packet arrival times. There are many different parameters that can affect packet arrival times. They range from the CPU load of the node to the load on the network between nodes. Because of these parameter variations, methodologies presented by this research are better suited to secure connections over networks of constant or near constant load, such as control system LANs.

To account for variance in the network and machine CPU load, standard deviation was calculated over the received samples of the benign client (Equation 4.1). The authenticated window was built by $p \in [\bar{x} - S_N, \bar{x} + S_N]$ where p is the inter-packet arrival time, S_N is standard deviation, and $\bar{x}$ is the average inter-packet arrival time of the benign client.

$$S_N = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{4.1}$$

4.1 Experiment

In the experimental attack scenario, an attacker was able to gain access to a control system's LAN. A portable computer was attached to the system's LAN and used to execute the attacks. The attacker was able to gain access to the encryption keys an authorized user had, as well as spoof the workstation's MAC and IP addresses, to prevent the secure server from recognizing or logging any potentially malicious activity. Once the spoofing detection algorithm was established, a second trial from a different machine is executed to test the intrusion detection algorithm.

4.1.1 Experimental Setup

The experimental scenario is comprised of several machines. A secure server was running Gentoo Linux environment on Intel Core 2 Duo E7500 CPU, 2GB RAM, and Intel's 82567L-M-3 ethernet controller. A benign client was running on a separate machine of the same configuration. The first malicious client was running Debian Linux, on a BCM2835 SoC controller connected to the LAN using RTL8188CUS

802.11n WLAN Adapter. The second malicious client was running on a Windows 7 AMD Phenom II X2 555 CPU machine, 8GB of RAM, and a Realtek PCIe GBE family network controller was used. All of the machines were connected with each other though a Netgear WNDR3500 wireless router (Figure 4.1).

4.2 Results

Tcpdump utility was used on the secure server to log all of the packets captured by the server's network card during the experiments. The first experiment determined a pattern of the proper SSH RSA handshake. The timings of SSH RSA handshake for the benign machine can be found in Figure 4.3.

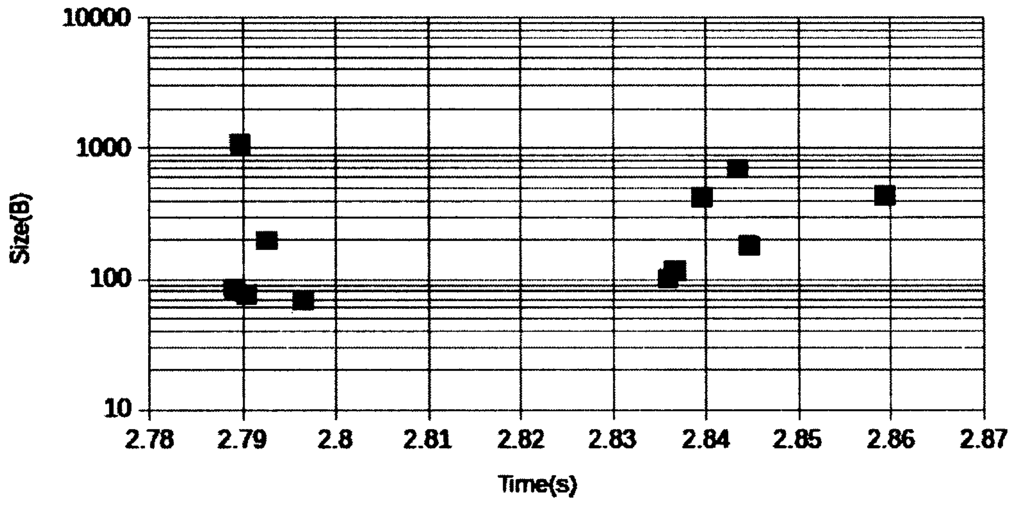

Figure 4.3: SSH RSA handshake

To remove the human input timings, an *ssh-agent* program was used to cache the RSA decryption key. This allows the RSA public key to be encrypted and does not require a user input to decrypt it while calling the *ssh* command.

After the handshake pattern was established, both benign and malicious clients executed an SSH connection command to the secure server, while capturing all the packets with the tcpdump utility. SSH software was instructed to establish a connection using a public/private key pair, execute a *uname -a* command which returns a string of text and exits.

```
# ssh secure_server uname -a
```

All of the experimental data was recorded in a comma separated values (.csv) file. The deltas between packet arrival times were calculated using the first SSH handshake packet as a time zero packet. Time deltas were graphed against the SSH packet number. (Figure 4.4). The first three benign trials are SSH sessions from the benign machine to the secure server. RSA Handshake Pattern highlights the packets that are used in authentication. Malicious trials 1 through 3 are from the portable computer, while Malicious trial 4 is run on the verification machine. Packets of the benign connection had very small deviation, which improved detection accuracy.

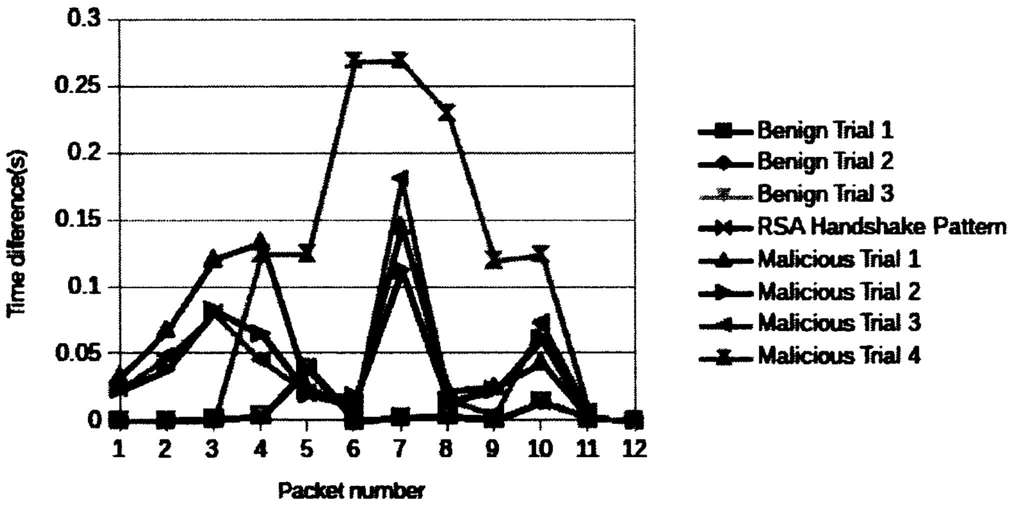

Figure 4.4: Packet arrival time differences

A classifier was then designed to measure these packet arrival differences, and decide whether a connected host was authorized or not. The variance in packet arrival times from an authorized host was measured, and standard deviation was used to determine whether the connection is authorized or not. Figure 4.5 shows an average arrival time, as well as the standard deviation window. Each column represents a packet in a sequence of SSH handshake authentication. Column height shows the average time between each packet being processed. Window on top of each column represents the window of benign authentication derived by $p \in [\bar{x} - S_N, \bar{x} + S_N]$.

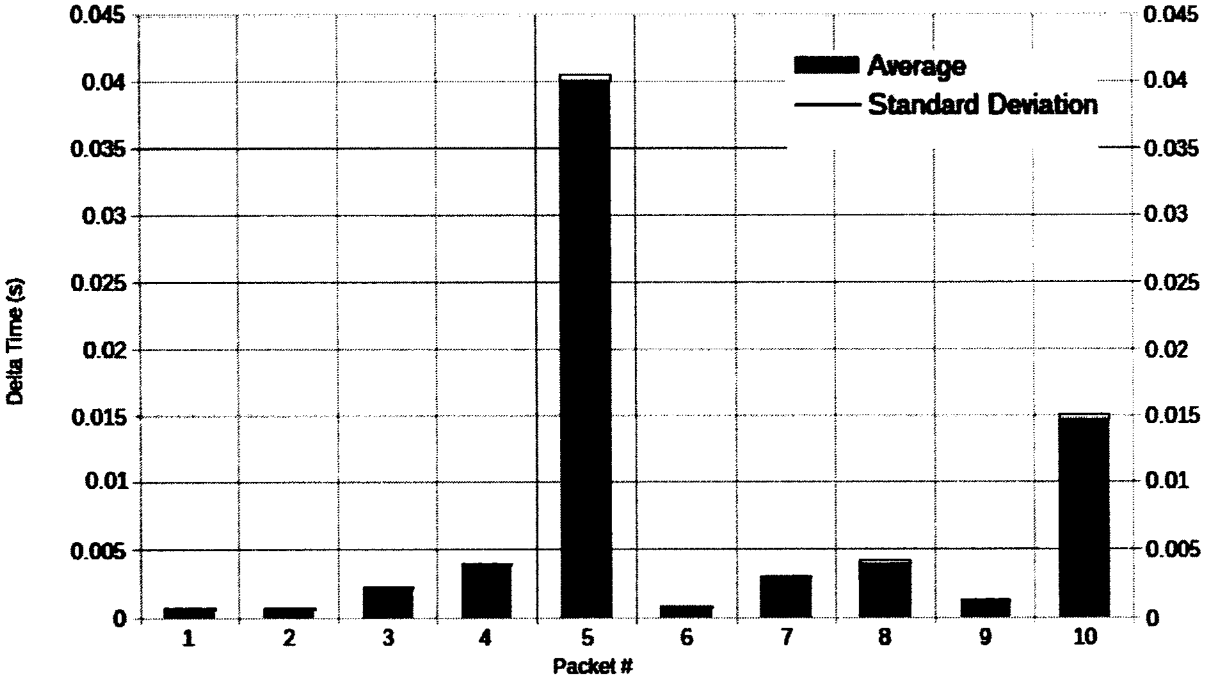

Figure 4.5: Standard Deviation of packet arrival differences

The same process was applied to the data extracted after a malicious host attempted to log into the secure server. Figure 4.6 shows packet arrival differences for one of the malicious login attempts. All of the times between processed packets were different from the times in the benign trial, allowing to classify this connection as malicious. These results show that a high accuracy classification can be done using simple classifiers when the network differences are high.

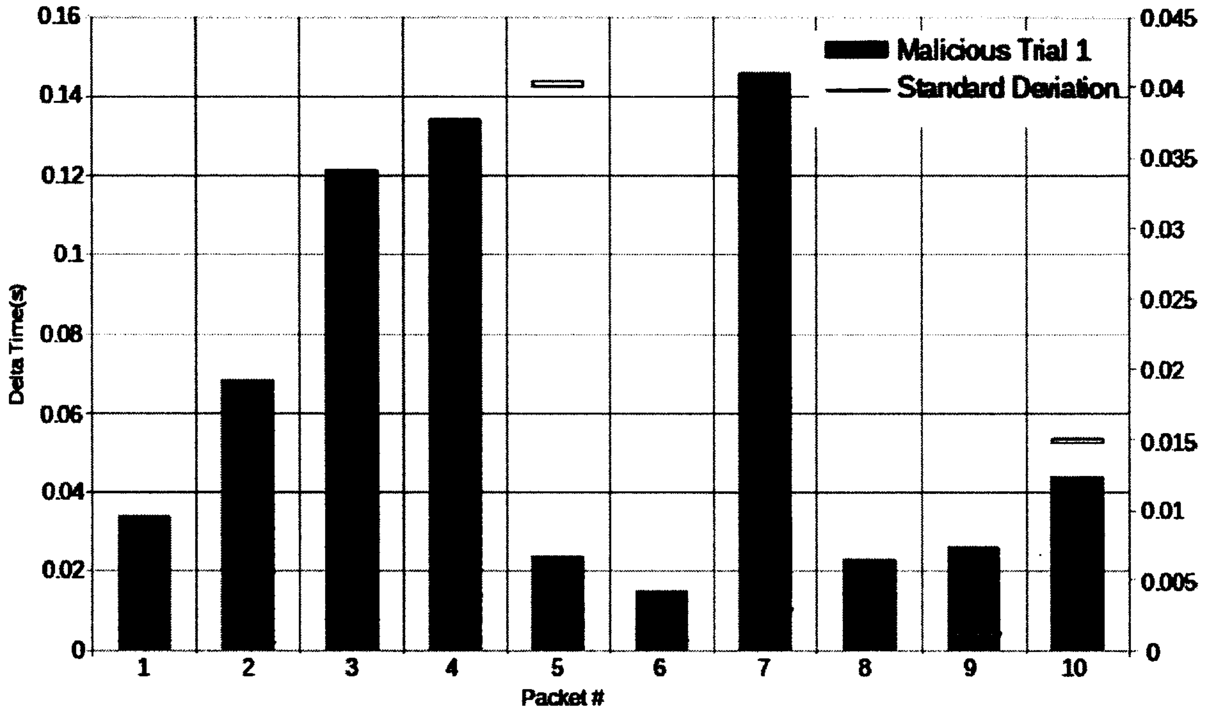

Figure 4.6: Standard Deviation of packet arrival differences during an attack

CHAPTER 5

SESSION INSTANTIATION

One of the critical components of the developed Intrusion Detection System algorithm is the session capture method. When an attacker is spoofing MAC and IP addresses, there are two methods for them to inject data to the control system network - either by using TCP session spoofing, or instantiating new TCP connection. This section covers the basics of TCP sessions, and discusses experiments performed to evaluate different session instantiation techniques.

The possibility of spoofed addressing as well as TCP session injections makes it impossible to differentiate between communication sessions as defined by the TCP flow model. Therefore, to logically group incoming traffic into sessions a new session definition must be created. Several session aggregation techniques were selected and tested for this research.

5.1 TCP Session Flows

An example TCP session flow is shown in Figure 5.1. TCP is a connection oriented protocol that utilizes internal state variables in order to maintain the connection. TCP session begins with a TCP handshake. The TCP handshake includes a 3-way packet exchange that is commonly explained as SYN, SYN/ACK, ACK. SYN is a synchronization packet sent from the client to the server. This packet starts a TCP session and asks the server to establish a connection. The server then responds with a SYN/ACK packet, acknowledging the connection request. Afterwards, the client acknowledges the server's acknowledgment with an ACK packet. This completes the TCP handshake and the data exchange can be started. The TCP session is closed with a FIN, ACK, FIN, ACK sequence. When the client decides to terminate the connection, it sends a FIN packet to the server. Server then sends an ACK packet, acknowledging the request to terminate the connection. After sending the ACK packet, server has time to release all of the resources used by the connection, and send a FIN packet of its own to notify the client that all of the resources have been cleared. The client then sends its final packet - ACK - to acknowledge the connection termination. TCP sessions are sets of all of the packets between two nodes that start with the TCP handshake and end with the FIN, ACK, FIN, ACK packet sequence.

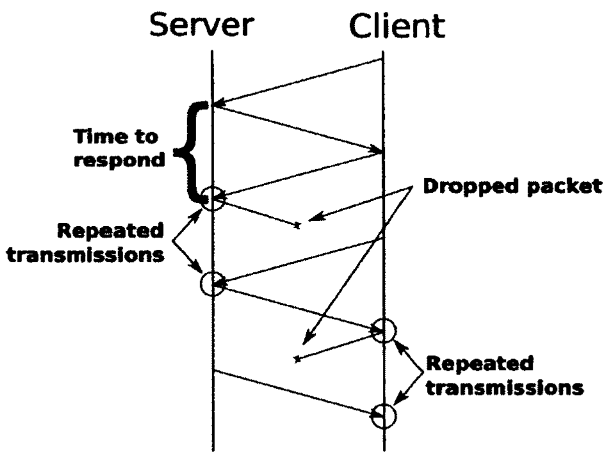

Figure 5.1: Client-Server Session Graph

Whenever an attacker wants to spoof and inject information into the TCP session, he has to inject packets in the middle of an already established TCP connection.

But proper network telemetry can only be obtained if the data is extracted from a session flow model, rather than a by-packet basis because there is no performance information can be present in analysis of a single packet. A single packet will only have the time stamp of its arrival to the server. In order to capture relative performance information of the TCP communication, two methods have been created and tested for this research.

5.2 Session Duration Based Instantiation

Silence time based session detection identifies periods of communication silence when silence is larger than a time threshold. New sessions are defined as a set of packets between the detected silence intervals when the next packet is sent towards the server (PLCs). This method was selected to match the periodical silence of the polling nature of Modbus/TCP SCADA architecture. Silence time threshold was determined experimentally by using 0.1 second interval increments to maximize C4.5 based classifier accuracy. Table 5.1 and Figure 5.2 show session counts for a given time interval, while Table 5.2 and Figure 5.3 show accuracies achieved by the C4.5 algorithm as well as Model Build Times (MBT) in seconds.

Malicious session defines a session that has at least one malicious packet. As the length of the inter-session silence increased, the count of benign sessions decreased, while the count of malicious sessions increased. This trade off happened due to the definition of the malicious session. A benign session could not have any malicious packets at all, while a malicious session could have any number of benign packets, as long as it had as least one malicious packet.

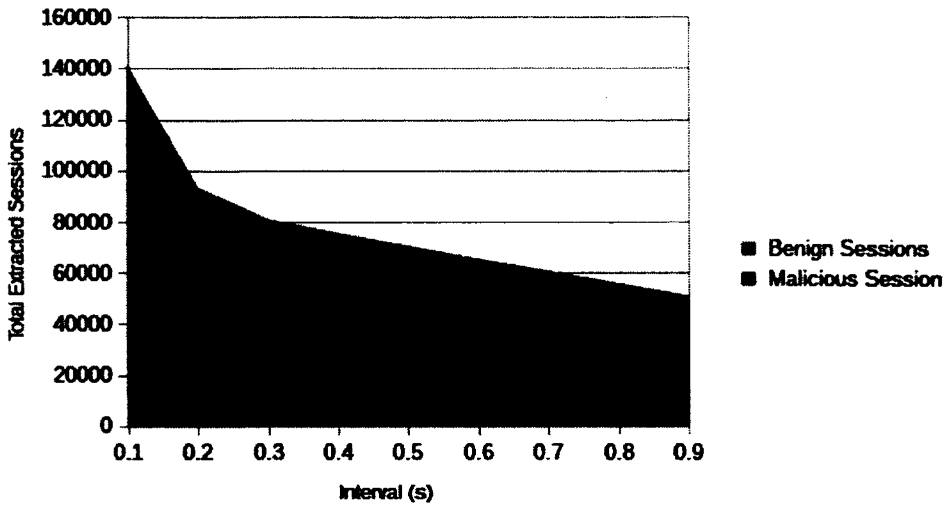

Figure 5.2: Silence Interval Detection session counts

Table 5.1: Generated session counts for varying silence intervals

Interval (s)	Malicious Session	Benign Sessions	Total Sessions
0.1	51156	89943	141099
0.2	51021	42125	93146
0.3	50893	29972	80865
0.4	50893	24458	75351
0.5	50891	19430	70321
0.6	50891	14454	65345
0.7	50888	9648	60536
0.8	50886	4869	55755
0.9	50508	403	50911

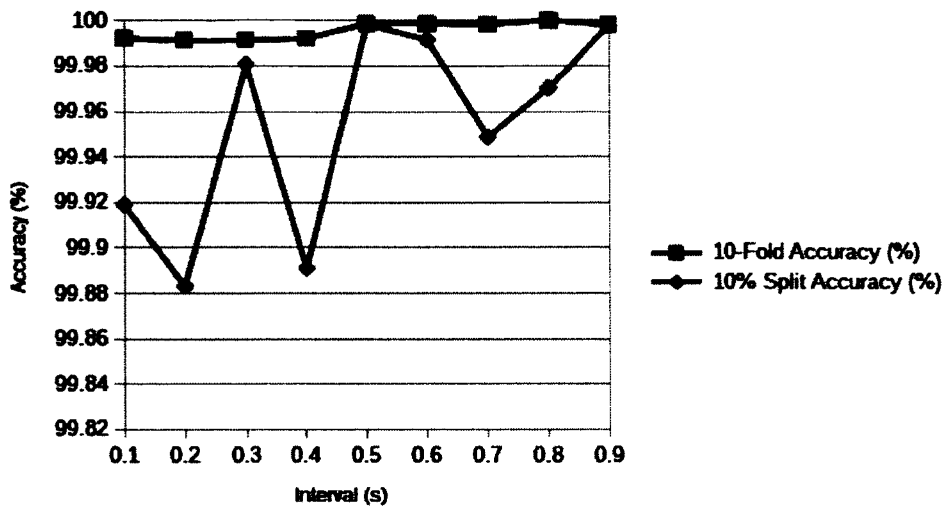

Figure 5.3: Silence Interval Detection Accuracy

Table 5.2: Classification results of varying silence intervals

Interval (s)	10-Fold Accuracy (%)	10% Split Accuracy (%)	MBT (s)
0.1	99.9922	99.9189	3.33
0.2	99.9914	99.8831	0.87
0.3	99.9913	99.9808	0.42
0.4	99.9920	99.8909	0.33
0.5	99.9986	99.9984	0.29
0.6	99.9985	99.9915	0.25
0.7	99.9983	99.9486	0.17
0.8	100.000	99.9701	0.49
0.9	99.9980	99.9978	0.11

Table 5.2 provides accuracies for two different classification experiments: 10-Fold validation, and 10% data split, as well as the time the C4.5 classifier took to build its model. The first experiment takes the dataset and splits it into 10 chunks. The classifier then uses the first 9 chunks for training, and the last chunk - for accuracy verification. The chunks are then rotated and the experiment is repeated ten times. The accuracy value in the table is the average accuracy of classification for all ten experiments. While this accuracy is not a good estimate of the overall classifier

accuracy, it can be used to verify the information distribution of the dataset. If some parts of the dataset contained lower information about the subject, the 10-Fold accuracy would be a lower number. Unlike 10-Fold validation, the 10% Split experiment uses first 10% of the dataset for training, and the other 90% for validation.

The accuracy given in the 10% Split column refers to the classifier accuracy when classifying 90% of the dataset. 10% of the dataset translates to approximately 2.5 hours of captured traffic. The interval of 0.9 was the last usable interval. Intervals above 0.9 are not listed because only malicious sessions were created while using such a large interval. While Classification accuracy seems to increase as the interval increases (Figure 5.3), The amount of malicious sessions in the data set start to outweigh the amount of benign sessions drastically. For example, if we take the amounts of sessions for 0.9 second interval, just choosing a malicious class over all of the data will result in 99.208% accuracy, as there are 50508 malicious features, yet only 403 benign features. However, selecting a malicious class for 0.3 second interval would result in an accuracy of 63.936%, while the C4.5 classifier was able to achieve 99.9808% accuracy for a 10% Split experiment.

For the use of silence interval as a session separation method, best value is determined to be 0.3 seconds, as it provides high classification accuracy as well as approximately equal amounts of malicious and benign features. Figure 5.4 shows the distribution of features based on session duration and features' class for the value of 0.3 second interval. Figure 5.4 shows a duration based distributions of sessions based on the silence interval of 0.3 seconds.

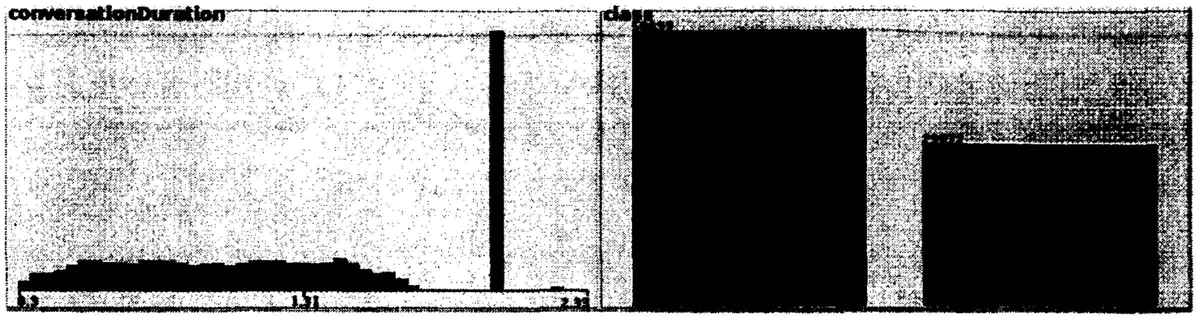

Figure 5.4: Silence Interval Session Duration and class Distributions for 0.3s Silence Interval

5.3 Session Length Based Instantiation

Session length can be used as an alternative method of extracting sessions from a non-interruptible TCP traffic flow. In these experiments, sessions were separated by counting the amount of packets already in the session. If that number is larger than some length and the next packet was sent to the server, a new session was instantiated. Tables 5.3 and 5.4 as well as Figures 5.5 and 5.6 present the data from varying session length separation value.

Table 5.3: Classification results of varying conversation lengths

Length (packets)	Malicious Session	Benign Sessions	Total Sessions
1	343202	255209	598411
2	283280	157384	440664
3	183664	99144	282808
4	167121	53786	220907
5	132017	49889	181906
6	133396	21106	154502
7	120452	16223	136675
8	112428	3897	116325

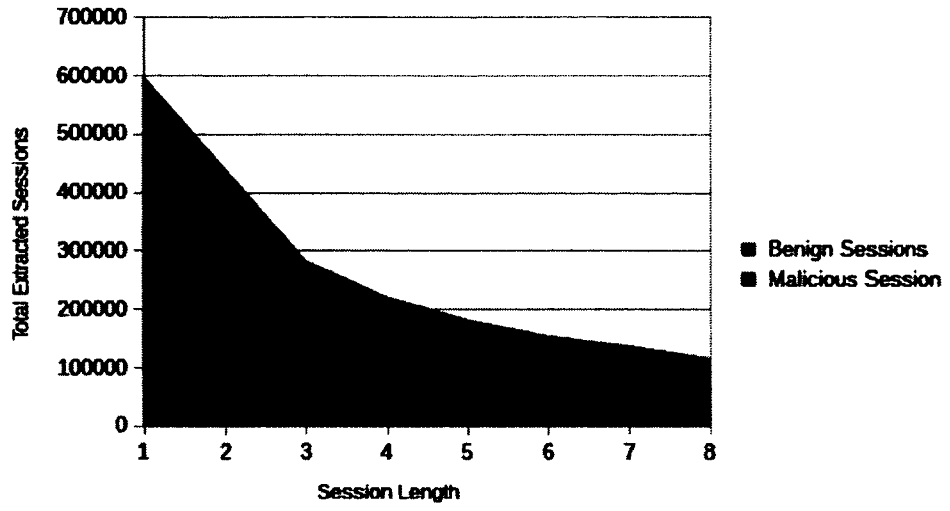

Figure 5.5: Session Length based extraction counts

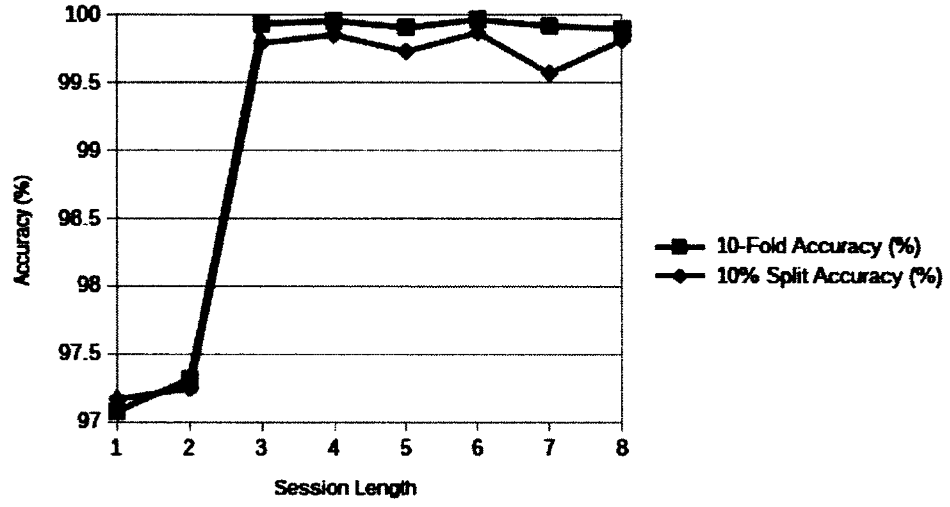

Figure 5.6: Session Length Extraction Accuracy

Table 5.4: Classification results of varying conversation lengths

Length (packets)	10-Fold Accuracy (%)	10% Split Accuracy (%)	MBT (s)
1	97.0796	97.1679	41.6
2	97.3172	97.2468	20.9
3	99.9346	99.7929	13.5
4	99.9570	99.8521	8.31
5	99.9082	99.7306	6.47
6	99.9663	99.8698	4.28
7	99.9188	99.5699	4.34
8	99.9003	99.8147	1.62

The same trend can be noticed in Table 5.3 as for the silence based extraction - the larger extracted session are, the less benign features are extracted due to the benign session being defined as having no malicious packets. However, the amount of malicious sessions extracted is not as stationary as with the silence based extraction. As the session length increases, the amount of extracted malicious sessions decreases, as seen in Figure 5.5.

At no point of using session length as a session detection parameter was there equal amount of malicious and benign sessions extracted. Moreover, when the amount of extracted sessions was comparable to each other, the classification accuracy was significantly lower than that of a silence based extraction. Comparable accuracy is achieved when the session has at least 3 packets. The 10% Split Accuracy of 99.7929 is achieved however, it takes 13.5 seconds to build the model. When the distribution of extracted sessions is graphed (Figure 5.7), the majority of the sessions span very little time in contrast to silence based sessions extraction.

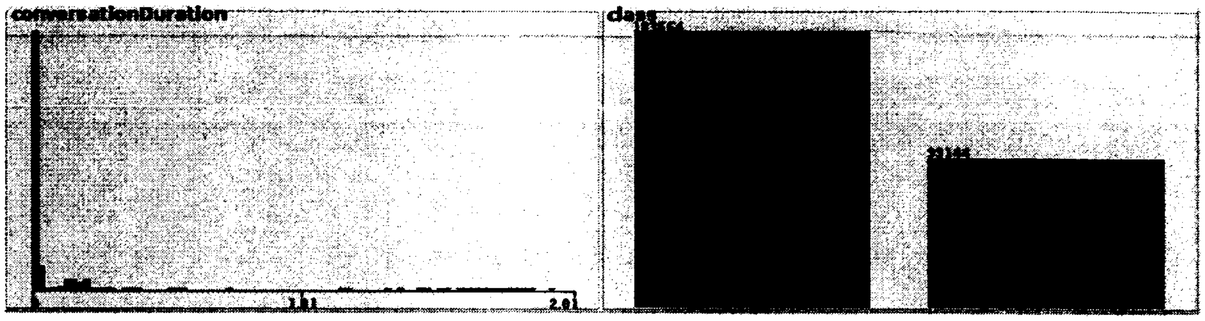

Figure 5.7: Session Length of 3 Session Duration and class Distributions

5.4 Conclusion

When an attacker is spoofing their machine's addresses and injects traffic in the middle of an already-established TCP session, it becomes impossible to isolate the attacker's TCP sessions and TCP sessions from a benign machine. In order to collect the information that can be used for an intrusion detection, two session aggregation methods were created and tested.

Overall, silence based feature extraction presents better accuracy and efficiency of detecting network intrusions. Both classification accuracy and model build time achieved were better for this method in comparison to session length based extractions. The proportion of benign and malicious features used for classifier training should be of concern. During the deployment of the developed IDS, the ratio of the benign and malicious features to the total feature count should be approximate to 0.5.

CHAPTER 6

OPTIMAL CAPTURE INTERVAL DERIVATION

Unlike business networks, ICS networks are much more predictable and experience little to no changes in their traffic flows [45]. The most changes are experienced when an exceptional situation requires engineers to modify the ICS by interrupting a normal ICS flow [44]. Network changes are particularly important when the engineer communicates with an ICS over a long distance because longer distances are more likely to include segments of business-like networks where network performances are susceptible to high magnitude changes.

Optimal capture interval must therefore be large enough to not only include information about the hardware performance at capture time, but all of the network changes needed to accurately monitor traffic. To understand the capture time requirements for a training dataset, machine learning algorithms described in section 3.2 were used on datasets of varying lengths to verify the performance of the classification and identify the minimal capture times needed for a high accuracy classification.

6.1 Minimizing the Training Window

Figure 6.1 shows classification accuracies for multiple classifiers as the capture time increases. The whole dataset was captured in 101,890 seconds or approximately 28 hours. The machines used as SCADA clients were identical in hardware, but

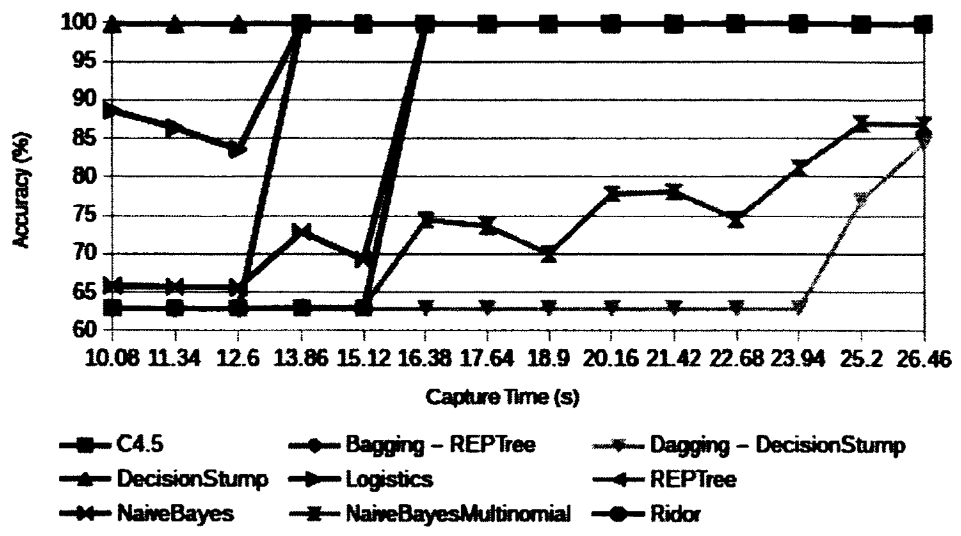

Figure 6.1: Capture Time Maximization Graph for Classification of two clients

used two different operating systems - Gentoo and Windows. Silence based session extraction was used as covered in Chapter 5. Total of 80,865 instances were captures, out of which 50,893 instances were malicious and 29,972 instances were benign. 10.08 seconds capture interval corresponds to approximately 8 captured instances and 26.46 seconds capture interval corresponds to 21 captured instances. The increment of 1.26 seconds was found to be an average conversation duration for the whole 28-hour capture segment when using a 0.3 second silence interval detection.

It is interesting to note that Bagging and Ridor classifiers' accuracies experienced a very sharp accuracy increase with an addition of just one more instance to the training dataset. Since bagging is an ensemble modeling technique that splits the dataset further into chunks, and the addition of one more feature allowed its chunks to be split such that the information about the network was present in all of the chunks. Ripple Down Rule learner (Ridor), however, is not an ensemble modeling based classifier. Ridor creates a default rule and then iterates over the dataset to create multiple exceptions. The best exceptions are then chosen based on the amount of data points that the exception encompasses. The additional data point, therefore, allows new exceptions to encompass more data points than the original default rule. Decision Stump classifier was able to maintain high accuracy for all the sampling times. Naïve Bayes Multinomial and Decision Stump based Dagging did not exhibit high accuracy jumps, though Dagging started having accuracy increases on larger data sets in comparison to the other tested classifiers. This happened due to Dagging's way of further splitting the training data for ensemble modeling, but splitting an already small training data set resulted in loss of critical information.

Table 6.1 and Figure 6.2 show maximum accuracies of classification achieved within the selected capture time range. When the capture window reached 21.42 seconds, all but two classifiers had an accuracy above 99.88%. Since such a small window resulted in a very high classification accuracy for the rest of the 28-hour long capture, a dynamic classification may provide a viable insight as a future work. Past PLC communication of 20 seconds can be captured in real time and used for training classifiers to update network changes if such a need arises. But the captured data shows that ICS network changes so little over time that a 20 second capture acquires enough information about the environment to accurately classify the rest of 28 hours of operation.

Table 6.1: Maximum accuracies and corresponding Capture Times of classifiers for Classification of two clients

Classifier	Capture Time (s)	Maximum Accuracy
C4.5	13.86	99.8837
Bagging - REPTree	13.86	99.8837
Dagging - DecisionStump	26.46	84.3538
DecisionStump	10.08	99.8837
Logistics	20.16	99.8924
REPTree	16.38	99.8837
NaiveBayes	21.42	99.9481
NaiveBayesMultinomial	25.2	86.9219
Ridor	16.38	99.8837

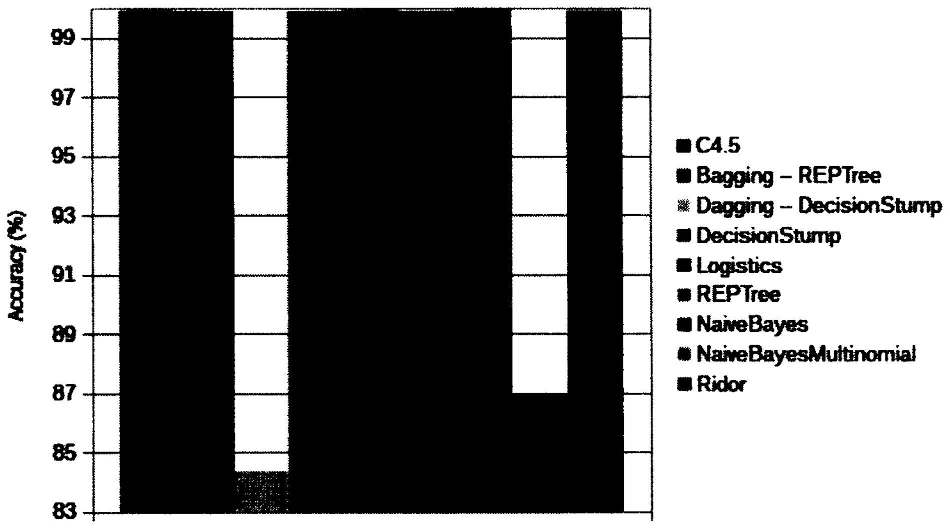

Figure 6.2: Maximum Accuracy within Tested Window

Machine learning algorithms used were able to achieve very high classification accuracy. More over such a high accuracy was achieved when only 25 instances were used for training. The resulted classification models were able to accurately classify between 80,750 to 80,800 other instances depending on the selected classifier. Overall, Naïve Bayes was able to achieve the highest accuracy of 99.9481% with the training time of just 21.42 seconds.

Dagging ensemble modeling based on the DecisionStump classifier performed the worst - at 84.3538% accuracy. Moreover, dagging was not able to classify instances at all when the capture time was under 12.6 seconds. Dagging utilized breaking the data in several folds to create an ensemble of classifiers, and at capture time under 12.6 seconds there were too few training instances to be broken apart for ensemble modeling. Unlike dagging, bagging samples its instances with replacement, which allows repetition of the instances, so bagging was able to perform better.

6.2 Accuracy Validation Against a Third Machine

Classification of a different, testing dataset was used to classify the robustness of the utilized feature extraction method and the ability to differentiate between multiple attacking targets. This different dataset contains no instances captured from the malicious machine used for training the classifiers, but it includes as many instances of a malicious machine that ran the same software while using a different operating system. Figure 6.3 shows the performances of selected classifiers when the training model was built based on the same instances used in the previous section of this chapter. The third machine was the same in hardware configuration as the two machines used in the previous section, but its operating system was Windows.

Figure 6.3 shows similar or better performances for most classifiers. C4.5 classification algorithm was able to achieve 100% accuracy. After using less than 25 instances to build its training model, C4.5 algorithm was able to accurately classify all 80,890 instances that were not present in the training dataset nor belong to the malicious machine used for training. Other classifiers achieved very high accuracies as well. All of the classifiers had accuracies above 94% overall. Such high accuracies suggest that it is possible to accurately detect a set of machines that is larger than the set of machines used for training. Such a high accuracy of classifying an unknown machine also leads to possibilities of utilizing the developed IDS to detect intrusions from unknown machines.

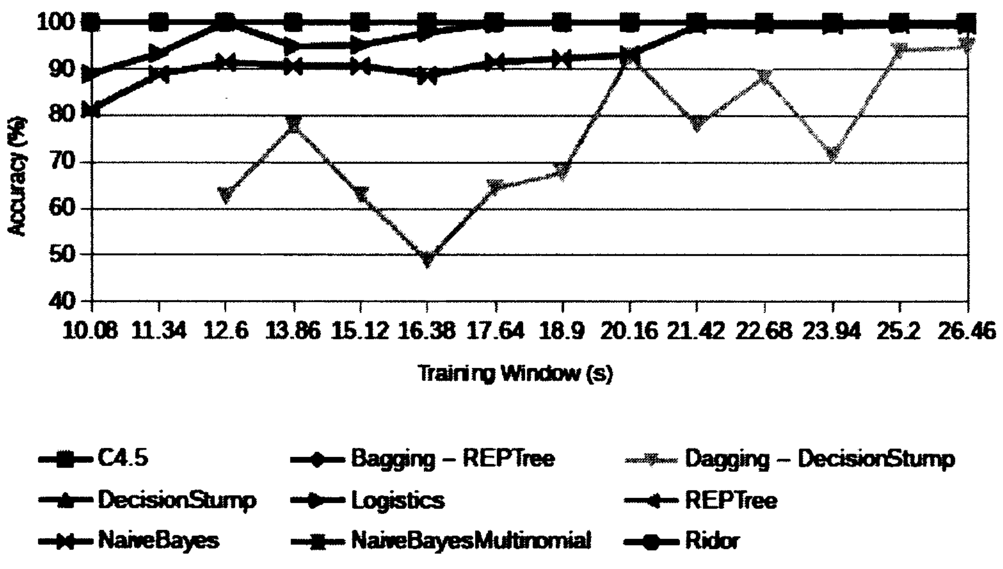

Figure 6.3: Capture Time Maximization Graph for Classification of three clients

Dagging classifier's performance fell down to under 50% for classifying 16.38 second capture time. But the accuracy increased above 94% when the classification range was extended. No steep accuracy increases are also detected. All of the classifiers are approaching 100% accuracy. Naïve Bayes and Logistics classifiers had accuracies under 99% while the capture window was under 21.42 seconds. All but Dagging classifier were able to achieve accuracies above 99%. Lower performance is the result of Dagging's method of splitting the training data into folds and the training dataset containing few instances. The accuracies will increase as the training set size increases, but the point of this experiment was to identify the smallest training dataset size possible for high accuracy classification.

Though Dagging classifier's maximum accuracy increased by the use of a new dataset, its worst accuracy also decreased. Minimal accuracy decreases past 50% are indicative of a use of a testing dataset that differs from the training dataset. For example, if the amount of instances in one class is bigger than the amount of instances for another class, but a feature based pattern can not be determined, the classifier can increase its accuracy by always selecting a class that has more instances in it - that way the accuracy will equal to the percentage of count difference between instances of classes. In a newly supplied dataset though, the classifier can not know the class' instance count difference during the training stage, therefore accuracies under 50% are possible.

Table 6.2 shows maximum accuracies for selected classifiers and corresponding window time. C4.5 was able to achieve a 100% accuracy. That is, C4.5 was able to classify 81,100 instances correctly after using 12 instances created from a different machine for training. Though this should not be an expected result for all of the experiments as C4.5 was achieving 99.8837% accuracy when classifying the same machines it was trained on.

Table 6.2: Maximum accuracies and corresponding Capture Times of classifiers for Classification of three clients

Classifier	Capture Time (s)	Maximum Accuracy
C4.5	15.12	100
Bagging - REPTree	10.08	99.9975
Dagging - DecisionStump	26.46	94.8335
DecisionStump	10.08	99.9975
Logistics	26.46	99.9815
REPTree	10.08	99.9975
NaiveBayes	25.2	99.6005
NaiveBayesMultinomial	10.08	99.9975
Ridor	10.08	99.9975

Figure 6.4 shows the maximum accuracies of the classifiers in a bar graph. All of the classifiers' performance increased in comparison to the dataset classification in the previous section. NaïveBayesMultinomial performance has shown the highest increase from 86.9219% to 99.9975%. Dagging based classification also increased from the maximum of 84.3538% for the same machine classification to 94.8335% for the classification of the new machine. This suggests that some features extracted from The Windows machine are further away from the multidimensional set of features between the Debian and Gentoo machines. Such difference will be further evaluated in Chapter 8.

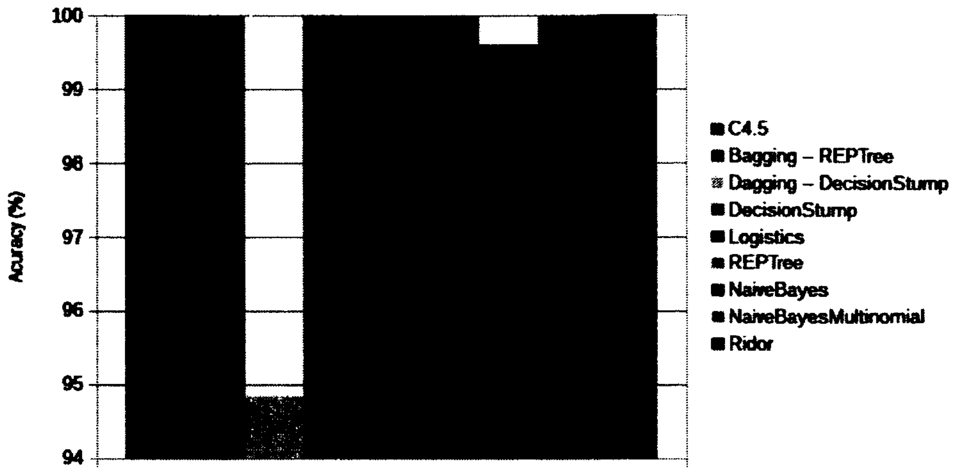

Figure 6.4: Maximum Accuracy within Tested Window for Classification of three clients

Figures 6.5 and 6.6 show feature distributions between the Windows based attacker's machine and the PLCs and the Debian based attacker's machine and PLCs respectively. These graphs list all 8 of the features that describe a conversation. Figure 6.6 shows all of the instances captured from the Debian based machine within the 28-hour period, though, only a maximum of 26.46 second capture was used for training the classifiers. While some features have no visible patterns, such as conversation duration or average time to respond, others, such as total client and total server retransmissions can be seen to vary greatly. These variations are what contributes to the high accuracy classification of the data.

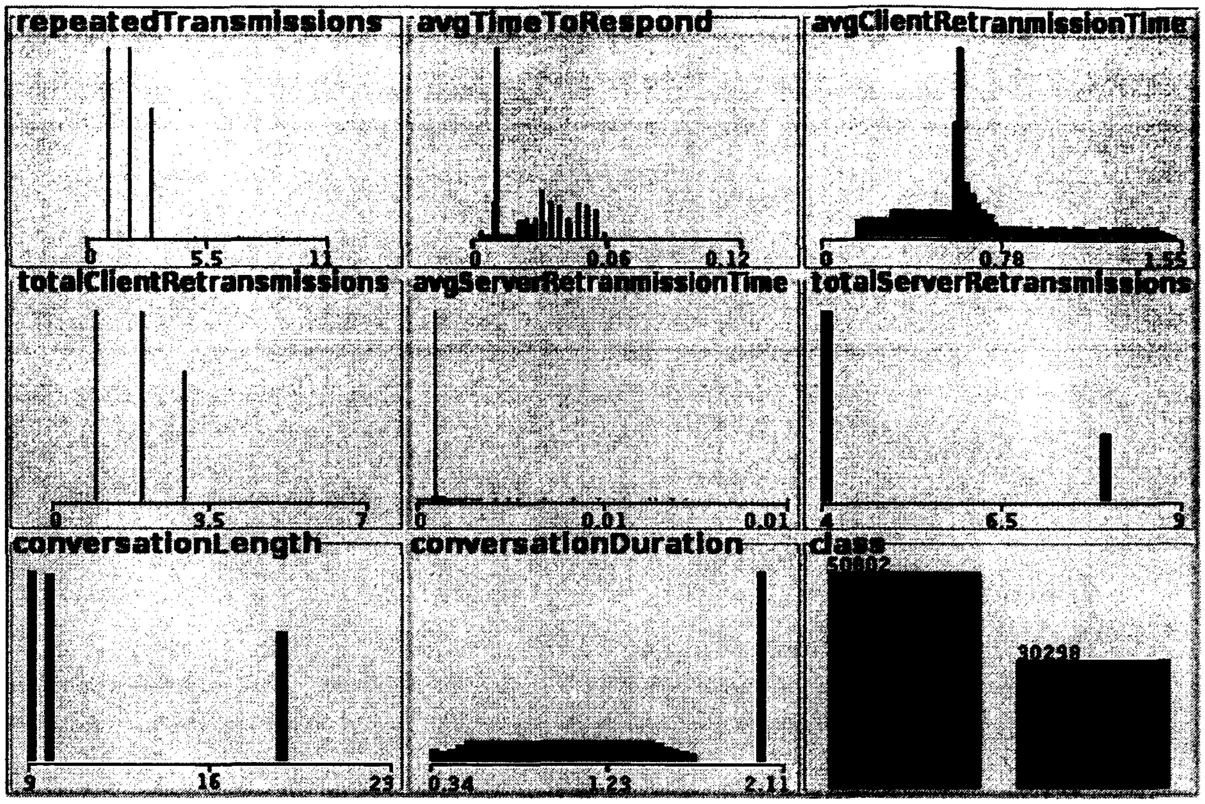

Figure 6.5: Distribution of Features for the testing dataset. Blue is malicious, Red is benign.

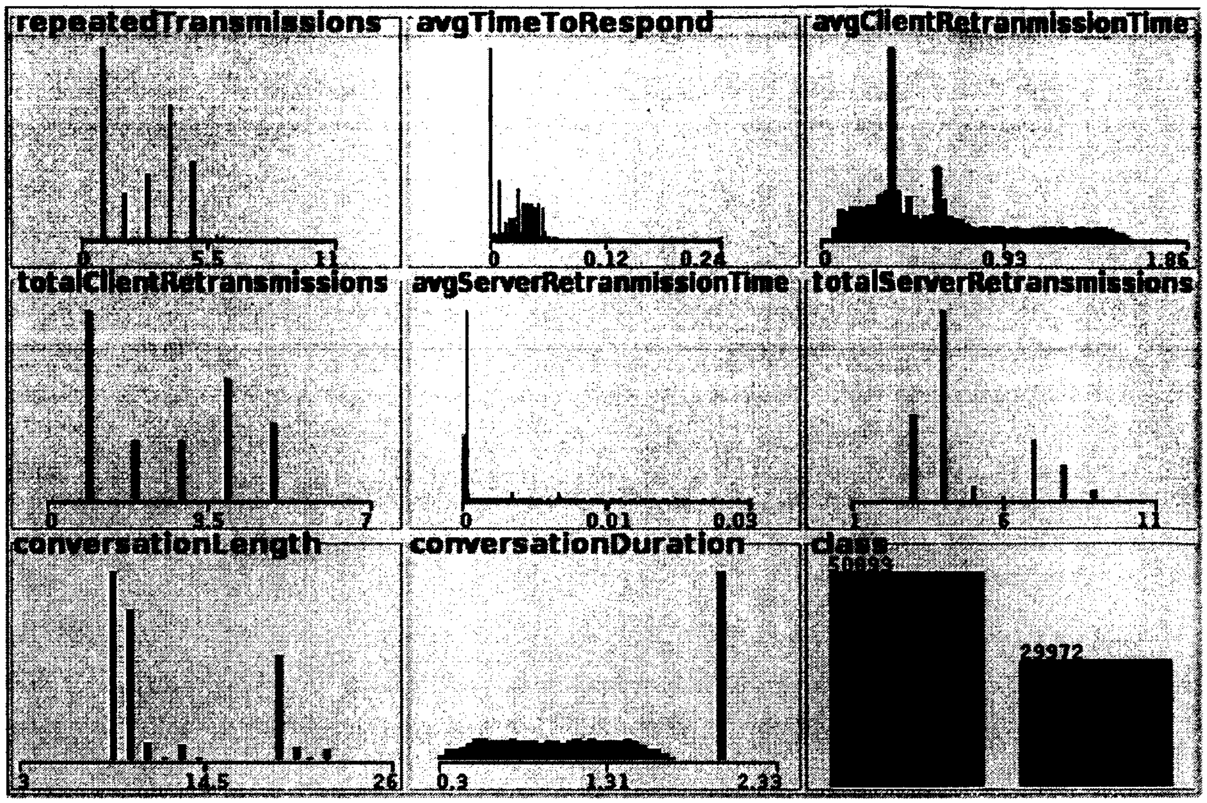

Figure 6.6: Distribution of Features for the training dataset. Blue is malicious, Red is benign.

6.3 Conclusion

The smallest time of packet capture needed to provide enough information using a 0.3 session instantiation method for a high accuracy classification is under 25 seconds for the majority of tested classifiers. All of the datasets acquired for this research contain more than 24 hours worth of packet captures, which is 345,600% longer than the minimum time required for high accuracy classification.

In addition, when testing capture time variations in training dataset against a dataset that contained features from different malicious machines, the accuracy of all of the classifiers increased or stayed the same. Such a high accuracy of classifying an unknown machine leads to possibilities of utilizing the developed IDS to detect intrusions from unknown machines.

CHAPTER 7

SOFTWARE VARIANCES AS FEATURE CONTRIBUTORS

Classification accuracy of a Network Telemetry based IDS highly depends on processing times introduced by computer hardware as well as the network communication paths between the benign or malicious machines and PLCs. For these experiments, both benign and malicious machines were identical at a given hop-distance, and the following machines were used:

1-hop Classification

- CPU: Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz
- RAM: 2GB
- NIC: Intel Corporation 82567LM-3 Gigabit Network Connection
- OS: Gentoo Linux, kernel-3.12.13-gentoo
- Python version: 2.7.1

8-hop Classification

- CPU: Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz
- RAM: 16GB
- NIC: Broadcom Corporation NetLink BCM57781 Gigabit Ethernet PCIe
- OS: Gentoo Linux, kernel-3.12.13-gentoo
- Python version: 2.7.1

For all of the experiments, the PLCs were emulated using conpot on a machine with the following configuration:

- CPU: Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz
- RAM: 2GB
- NIC: Intel Corporation 82567LM-3 Gigabit Network Connection
- OS: Gentoo Linux, kernel-3.12.13-gentoo
- Python version: 2.7.1

The IDS is designed to notify system engineers of malicious traffic. For this research, malicious traffic is defined as any traffic directed to the PLCs that are not originated from the designated SCADA system. To achieve classification, the IDS must receive all of the network packets sent to the PLCs. The IDS utilizes libpcap [71] to capture all the traffic transmitted to the PCLs over the Modbus protocol. For this research, Conpot emulated Modbus/TCP protocol on port 502. The system maintains a Floating Delay Separation Boundary (FDSB) which separates traffic fingerprinting methods to outsider and insider classification engines. This is done to improve accuracy of classifying traffic originating from local to the ICS machines, versus the traffic originating from machines on the internet. For example, after analyzing Netgear WNDR4500, D-Link DI-604, and Linskys WRT300N routers, the delay feature range is under 5 milliseconds while the communication utilizes local area network, but it can be several orders of magnitude larger if the communication path connects distant nodes on the internet. However, not all of the traffic originating from a distant node can be malicious (as in the example of an engineer using remote access to troubleshoot an ICS at the remote site). Separation must be established in order to compare local paths with local paths and distant paths with distant paths. The floating delay separation boundary is defined as

$$FDSB = \frac{\sum_{i=0}^{n} \frac{max_i(I)}{n} + \sum_{i=0}^{n} \frac{min_i(O)}{n}}{2}$$

where I is the set of delay values from the packets originating from the inside of the ICS, O is the set of delay values from the packets originating from the outside of the ICS, max_i and min_i are the ith maximum and minimum values in the given set, and n is the amount of samples to take for the delay value which is the same for both sums. Having a value of n too high will use older network states which may not take place any more. However, having a value of n too low will result in a lower accuracy of the threshold calculation. For this experiment $n = 5$ is used, while sets of O and I have 50 items. These values were determined by iteratively running the classification over 5-minute intervals for all the captured dataset and determining the combination that results in the best classification (Table 7.1). The initial $FDSB$ value was set to 0.025. These values were determined by analyzing 838,818 packets acquired over two days of operation of an ICS. Figure 7.1 shows minimum and maximum response times between two different machines on the same LAN (malicious and benign 1 hop), and the delays introduced by moving the same machine further along the communication path (malicious 1 hop, 2 hops, and 8 hops).

Table 7.1: Accuracies of separating outsider and insider traffic by varying n

Accuracy (%)	n
91.2%	2
97.3%	3
100%	4
100%	5
100%	6

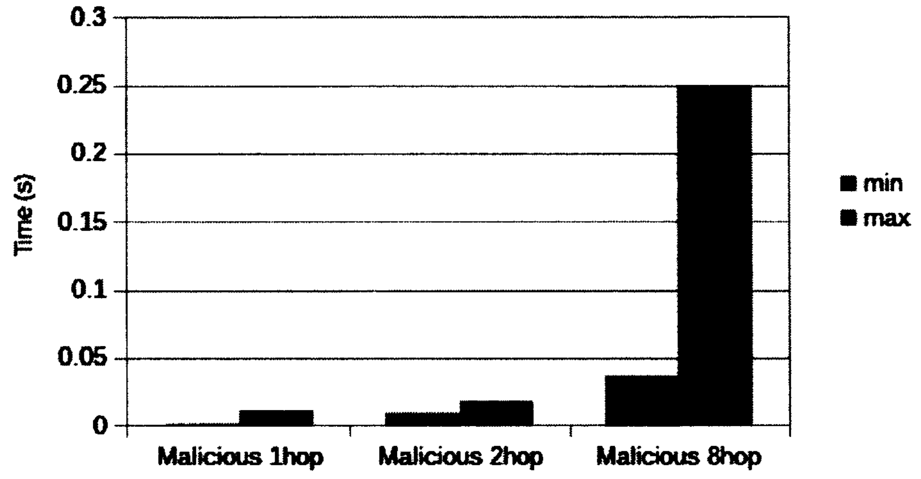

Figure 7.1: Average Response time in a session over 1 day

Once the packet is identified as outsider/insider and sent to a proper classifier, its arrival time is recorded and compared against the previous packet that was sent to the same classifier. Inside-originating packets are only compared with the telemetry of other inside-originating packets, and outside-originating packets are compared with the telemetry of other outside-originating packets which allows the IDS to mark some external traffic as benign as in the case of a system engineer working from home. Traffic to the PLCs is tagged as inside/outside traffic based on the subnet of the packet's IP address. Code running on the PLCs is assumed benign and IP address spoofing from the PLCs is not expected. However, since the developed IDS uses traffic anomalies, it may be possible to detect anomalies on either side of the IDS networks. Multidirectional attack detection is part of the future work of this research. For the sets of experiments in this chapter, one side had to be assumed benign to maintain control of the changing variables in the experiments.

On every new packet arrival, telemetry data discussed in Section 3.1 is extracted and sent to the classifier. A new session is instantiated on every new packet. However, the calculations of features is done for all the packets received in the 2-second interval - the same interval that an experimental SCADA system uses for polling PLCs. For example, if there were 9 packets received by the classifier in the past 2-seconds, the first packet of session $n + 8$ is the last packet of session n.

Several machine learning classifiers have been tested for this IDS as discussed earlier: Naïve Bayes, Multinomial Naïve Bayes, Logistics, REPTree, Bagged REPTree, DecisionStump, Degged DecisionStump, Ridor, and C4.5. Naïve Bayes classifiers were chosen to determine probability models for telemetry data because the features used for classification can be largely independent from each other. Bagging modeling of REPTrees was chosen because they perform well with training sets containing large amounts of noise [64].

7.1 Hop Relationship

Telemetry based measurements are highly dependent on the amount of hops between the client and server. As nodes are separated by a larger amount of hops, added network systems introduce different delays into packet propagation; this introduces noise. The IDS carries two classification profiles - for insider traffic and for outsider traffic. Figure 7.1 shows that the delay increases with the amount of hops.

Differentiating between insider and outsider traffic is trivial as an outsider's delays are an order of magnitude larger than the insider's (Figure 7.1). If an outside attack is further from the PLCs than the system engineer's outside point of entry, differentiating between an attack and normal control packets becomes trivial as well. However, when benign and malicious traffic originates from the same amount of hops away from the PLCs, the system's hardware is the one that introduces the most important delays. Hardware and software delays become signatures that can be used for traffic classification. Most of the tested classifiers performed better when analyzing outsider traffic than insider traffic due to additional network delays present in varying network paths. The results show that the network path adds a significant amount of data to the used feature set to differentiate between two machines with a high accuracy.

7.2 Results

7.2.1 Insider Classification

The bagging technique of REPTree classifier was able to reach the highest accuracy of 92.2% when classifying traffic between two computers of different hardware configuration. These computers were separated by 1 hop. Figure 7.2 shows the classification accuracy of all used classifiers.

The Bagging method for the REPTree classifier achieves the maximum accuracy. However, considering the time required to create classification models, REPTree classifier achieves the best accuracy while maintaining lower processing requirements. Figure 7.3 shows the relationship of classifiers' accuracy versus the time it took the classifier to build a classification model. The time axis is logarithmic to better show model development duration distribution. This telemetry based IDS includes network load information in its feature set. Therefore classification is a time critical process that requires fast training of the classifiers and accurate results.

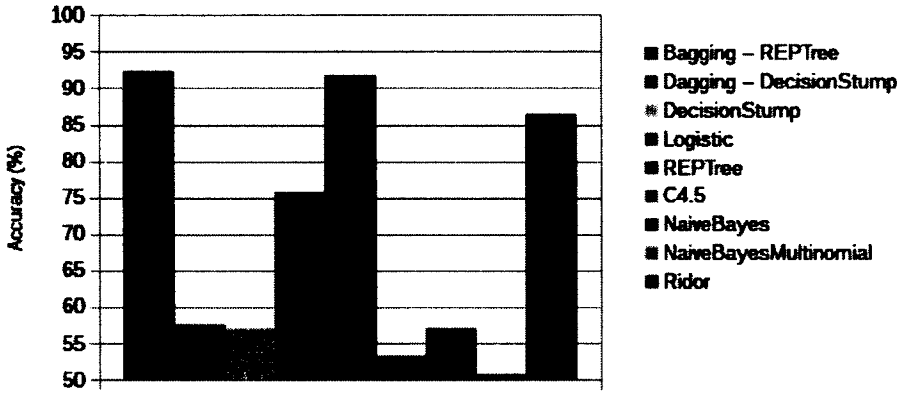

Figure 7.2: Classifier Accuracy for Insiders

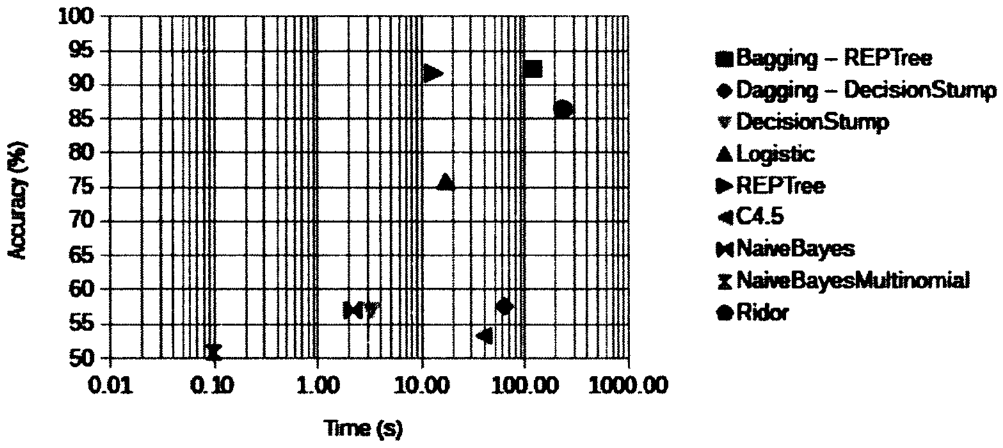

Figure 7.3: Classifier Accuracy vs Model Build Time

Despite very fast processing speeds of modern computers, differences between computer hardware manifests itself in network telemetry in such a way that machine learning classifiers can still detect those differences for a successful classification.

7.2.2 Outsider Classification

Unlike the Insider classification results, most classifiers were able to achieve very high accuracy in classifying packets from different machines. The bagging method of REPTree classifier was able to achieve an accuracy of 99.6%. The C4.5 classifier fell shortly behind having an accuracy of 99.5% (Figure 7.4).

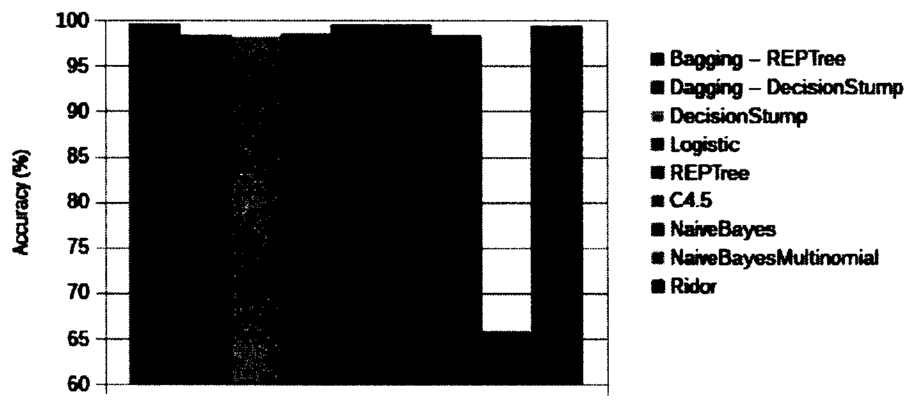

Figure 7.4: Classifier Accuracy for Outsiders

The Decision Stump and Naïve Bayes classifiers achieved a faster model build time - 1.71s and 1.41s respectively - while maintaining a high classification accuracy of 98.3% (Figure 7.5). Having traffic separated into insider and outsider groups also allows for a mix and match of different classifiers for different tasks.

Features extracted from the communication of machines separated by large network paths contain enough information to be accurately classified by machine learning classifiers. It may be further possible to generate signatures based on these features that would allow not only to detect an intrusion on an industrial control system's network, but to be able to fingerprint an attacker and determine whether the attacker has attempted intrusions before.

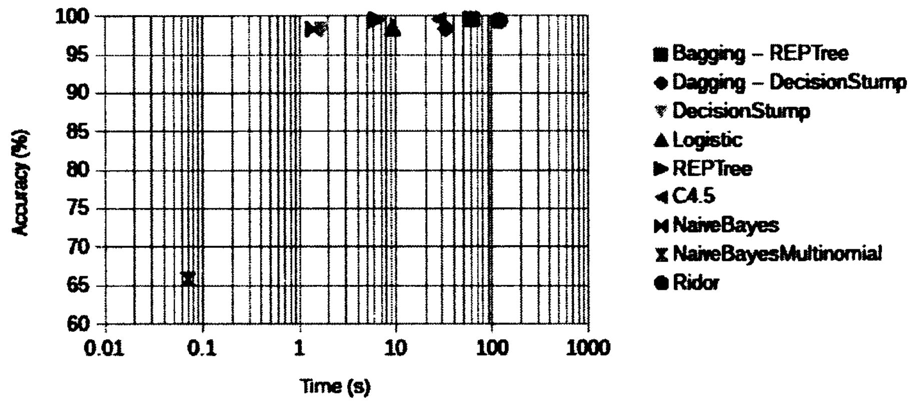

Figure 7.5: Classifier Accuracy vs Time to Build Model for Outsiders

7.2.3 Decreasing Capture Intervals

The results above show classification accuracy for all of the Modbus/TCP traffic captured by the IDS over a day. However, fluctuations of the network's congestion and throughput may reduce the classifier's accuracy. To account for this, another traffic capture of the communication separated by 1 hop was performed for an interval of 5 minutes. Figure 7.6 shows the classifier accuracy for this interval. The accuracy of Bagging for REPTree, REPTree, C4.5, and Ridor increased in comparison to the 24-hour capture interval (Figure 7.2). Ridor was able to achieve the highest accuracy of 94.3%.

The times required to build classification models have all been reduced significantly (Figure 7.7) in comparison to a 24-hour capture (Figure 7.3). This is an important result as ICS IDS is processing time critical data. REPTree was the fastest classifier with a model build time of 6.00 milliseconds, and an accuracy of 92.7%. The best performing classifier, C4.5 had a model build time of 110 milliseconds which may still be used for a dynamic classification of incoming packets.

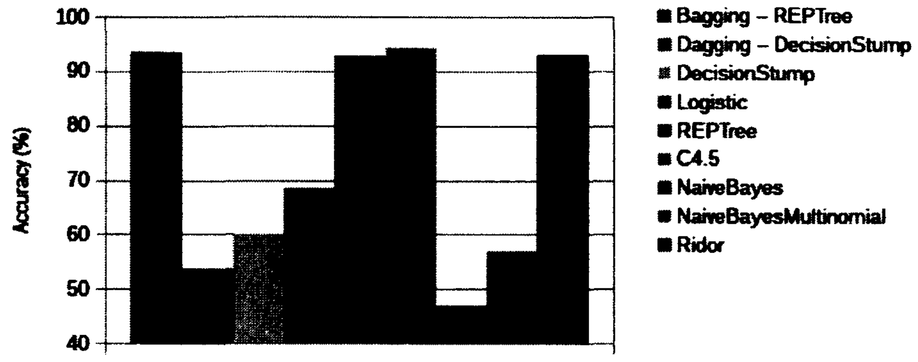

Figure 7.6: Classifier Accuracy for Insiders for traffic captured over 5-minute window

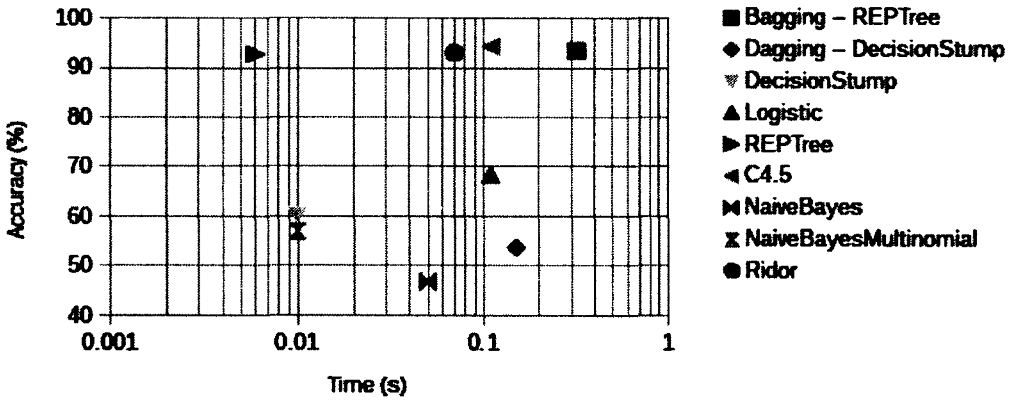

Figure 7.7: Classifier Accuracy vs Time-to-Build Model for Insiders for traffic captured over 5-minute window

By decreasing the capture window time to 5 minutes, accuracy of many classifiers was increased while the time to build classification models was reduced by several orders of magnitude. This goes along with the fact that network delays fluctuate with time due to different usage factors.

7.2.4 Conclusion

The developed IDS reaches high accuracy classification even when tasked to identify malicious behavior between two identical machines. Several classifiers were able to achieve an accuracy of above 90% of packets, while maintaining low time demand to build classification models. These results show that the developed IDS is able to perform well without the need for high performance computations. Average model build times were approximately 10 seconds. However, this step needs to be performed only once, during the set up of the IDS. Once the models are built, classification times of new information is in the order of microseconds, and creates an insignificant burden on the system.

The results show an important relationship between the inside and outside classified traffic. Even though the outside machines were identical, the networks between those machines and the secured PLCs were not, which introduced more information into the dataset and allowed the classifiers to further increase their performance.

CHAPTER 8

IDS VERIFICATION OVER MULTIPLE PLATFORM CONDITIONS

In order to obtain the IDS performances that can be seen in the real world, the IDS was tested over various intrusion conditions. Since more ICSs are joining typical business networks, possible insider threats can originate from within the business LAN. Most businesses buy workstation machines in bulk to lower the price of purchase [73]. This means that majority of the workstation machines a business owns are the same hardware and software configuration. Since the developed IDS works under the assumption that the provided features capture a difference between machines and softwares, it is safe to assume that when the attack originates from a machine that possesses the same hardware as well as software fingerprints, the IDS will not perform as well as the results seen prior. However, this scenario is almost impossible since an attacker must inject some code into a machine in order to gain control of it. The only scenario when such a condition can occur, is if the attacker has physical access to the benign machine, as is able to execute its attack without introducing new code. This scenario can be found in Section 8.1 of this chapter.

It is common for attackers to use software other than the one used by the SCADA system to launch their attack. If an attacker infiltrates one of the workstations that has access to an ICS, but the attacker wants the attack to stay hidden, some malicious code has to be used to execute the attack, resulting in modification of the original SCADA software. Additionally, there exists a multitude of different programming languages and their architecture. Both benign SCADA software and malicious attack software can be written in a variety of compiled, interpreted, or virtualized languages. Section 8.2 covers the IDS performance for an attack launched by a code created through different programming languages.

Another possibility is for an attack to be launched from a workstation that utilized a different operating system than the benign SCADA machine. In this example, the process scheduler, network driver, and network scheduler can be written by different developers and therefore result in different timing patterns of the output. Variation of these features will result in increased accuracy of classification for this IDS. An example of use of such system may be an insider threat where the insider utilized a "live" image of an operating system. A live image means that the operating system does not need to be installed on the hard drive of a machine and can be used as a stealthy source of an attack. In addition, operating systems commonly used for penetration testing can be launched live, and they contain many scripts necessary to perform an attack. If an attacker is not knowledgeable enough to create their own attack software, they can use of such an operating system. The results of the IDS testing over such a scenario can be found in Section 8.3.

8.1 IDS Accuracy for Identical Clients

If the used SCADA software contains vulnerabilities that allow the attacker to transmit malicious packets from the same system, or an insider manages to obtain and configure the same SCADA software on a machine that reflects SCADA hardware, packet signatures extracted by the developed IDS may not contain enough information to differentiate against such an attack. In this experiment, two machines of identical configuration were used. The hard drive of one machine was copied to the hard drive of the other machine. The only thing that was changed was the host name of the second machine to prevent DNS collisions on the testing network.

For this experiment, training time of 26.46 (the highest accuracy time for all of the classifiers tested in Chapter 6) seconds as well as training time of 1 minute, 5 minutes, 30 minutes, and 3 hours was used to determine the accuracy of selected classifiers. This scale was chosen to understand information gains in this experiment given widespread of capture time. Higher values of the training set capture are not used due to those times being impractical for an actual IDS deployment.

Table 8.1 shows the 10-Fold validation accuracy of all of the selected classifiers. For this experiment, the dataset was broken into 10 segments - folds, with each fold lasting approximately 2.24 hours. Nine folds were used for training and one for testing. Therefore, this is an equivalent of using 20.1-hour capture for training and 2.24 hours for testing. While this method does not provide any significant data about the data set that can be used to improve the IDS, 10-Fold validation provides an approximate value of how much information the data set contains about the problem.

Table 8.1: 10-Fold validation of selected classifiers on the dataset of two identical machines.

Classifier	10-Fold Accuracy (%)
Bagging	80.0208
Dagging	74.1563
DecisionStump	64.073
Logistic	74.8511
REPTree	79.1979
C4.5	75.69
NaiveBayes	64.292
NaiveBayesMultinomial	64.2699
Ridor	76.8364

Bagging based ensemble modeling of REPTrees was able to achieve the highest 10-Fold accuracy of 80.0208% for the data set, though, the data set is biased. Figure 8.1 shows the distribution of the data set values. There are almost twice as many malicious sessions as there are benign, therefore, if a classifier was to always choose a malicious class, it would have an accuracy of 64.2699%.

Figures 8.2 and 8.3 show accuracies and model build time for all of the selected classifiers while training over a certain period of time. Most classifiers had an accuracy around 75% while Naïve Bayes and Naïve Bayes Multinomial were unable to provide an accuracy higher than blindly choosing a malicious class.

Though there was a distinct drop in classification accuracy for all of the classifiers, it is a little surprising that some classifiers were able to achieve accuracy as high as 79%. While the malicious and benign machines were identical in hardware and software, they were plugged into different ports of the same switch. The switch also had several other machines plugged in it.

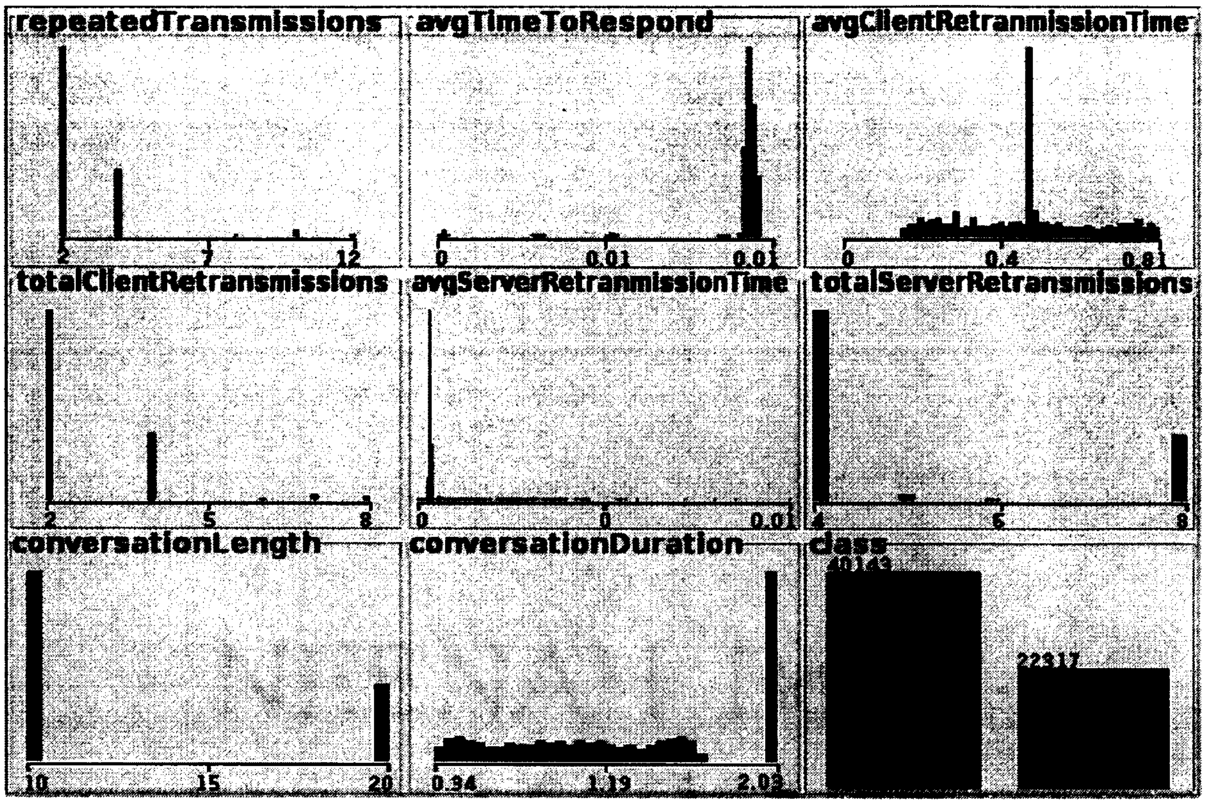

Figure 8.1: Feature Distribution of the data set for two identical machines. Blue is malicious and red is benign

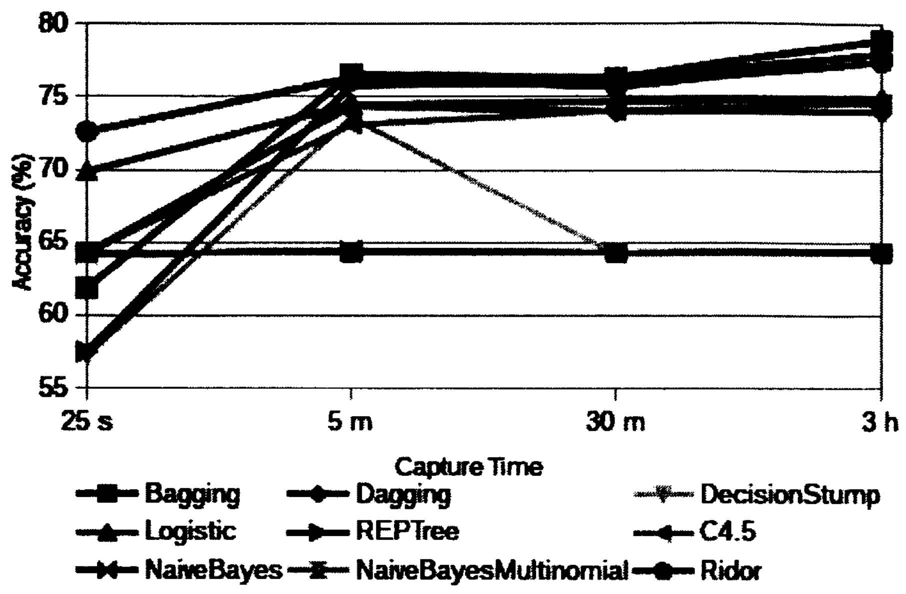

Figure 8.2: Accuracies of Different Capture Intervals

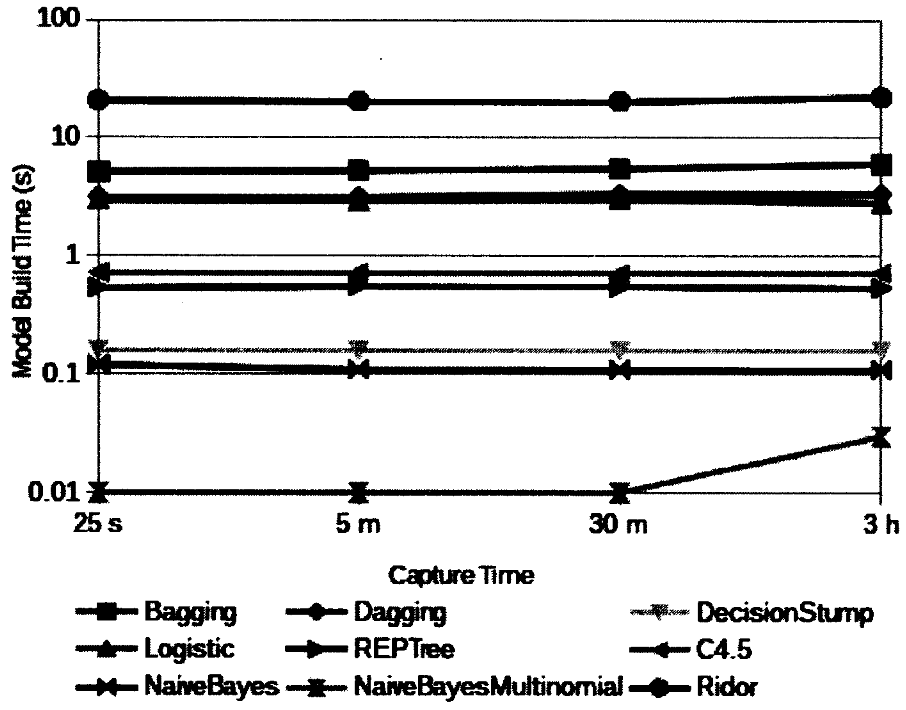

Figure 8.3: Model Build Times for Given Capture Intervals

The switch's packet scheduler can introduce some patterns that enabled classifiers to differentiate between two machines with some however small accuracy. Classifier accuracy for most of the selected classifiers increased dramatically when the training time was changed from 25 seconds to 5 minutes, which suggests that increasing the training window will have accuracy improvements for conditions that contain little feature difference between the malicious and benign datasets.

Figure 8.3 shows the time required to build the model. The y-axis of the graph - Model Build Time uses a logarithmic scale to better show the differences between classifiers. Most of the classifiers had very similar build times despite orders of magnitude variance of the size of the training data set. Overall, Bagging based ensemble modeling using REPTree classification had the best accuracy, but one of the highest model build times. However, a single REPTree based classifier was able to maintain high accuracy, while the time it took to build its model is under a second.

8.2 IDS Accuracy for Different Programming Languages

Often times the attacker may reverse engineer the protocol used by the SCADA system in order to present its own set of commands to seem like they came from a fully functional system [74]. Such attacks may often by-pass packet signature based IDSs as packet signatures become identical to the benign SCADA system. While the internal content of the packets becomes the same as the benign system, the attackers are more likely to use a programming language that is different from the one used by the developers of the SCADA system.

This experimental scenario tests the operation of the developed IDS under the conditions of a malicious SCADA software that was written in a different programming language being used for the attack. Here, the benign SCADA software was written in Python and its source code can be found in listing 1 in the appendix. Modbus protocol was implemented by Python's PyModbus library. The malicious SCADA software was written in C and its source code can be found in listing 2 in the appendix. Both of the codes result in the same logic being transmitted over the network. The same capture times were used as in the previous section in order to provide a good comparison of the information gain that results in varying the programming language used as a part of the experiment.

Table 8.2 shows the 10-Fold accuracies of all but one of the selected classifiers. C4.5 algorithm was excluded from this experiment. Most of the classifiers were able to build its models after 15 seconds of operation, however, after 10 minutes of operation C4.5 classifier was closed by the operating system as it used all of the 8GB of RAM. Even if C4.5 classifier was able to achieve a high accuracy of classification, the cost of time and memory needed for proper operation outweigh its benefit. Another outlier for this experiment was the Logistic classfier. It was able to achieve high accuracy of 99.1770% for its 10-Fold Validation experiment. Here, the 10-Fold validation experiment is the same as in the previous section in its times used to train and test the model.

Table 8.2: 10-Fold validation of selected classifiers on the dataset of C and Python based clients.

Classifier	10-Fold Accuracy (%)
Bagging	77.5547
Dagging	61.3923
DecisionStump	60.4182
Logistic	99.177
REPTree	76.8417
C4.5	N/A
NaiveBayes	60.409
NaiveBayesMultinomial	60.5759
Ridor	75.217

Figure 8.4 shows the feature distribution of the data set for this experiment. While this distribution looks similar to the one with two identical systems, some of the differences were enough to be picked up by the Logistic classifier to achieve a very high accuracy classification. However, there are still more malicious instances than benign, so the data set is biased. If a classifier was to blindly choose a malicious class, it would have an accuracy of 60.5759%.

Figures 8.5 and 8.6 show the classifier accuracies and model build times respectively. Logistic classifier is a clear outlier that was able to achieve a 99.0562% accuracy for 25 second capture time. Its accuracy increased to 99.4323% for the capture window of three hours. Other classifiers were able to create its models to be approximately 75% accurate, while Dagging and Naïve Bayes based classification was as accurate as the bias of the data - at approximately 60.5%.

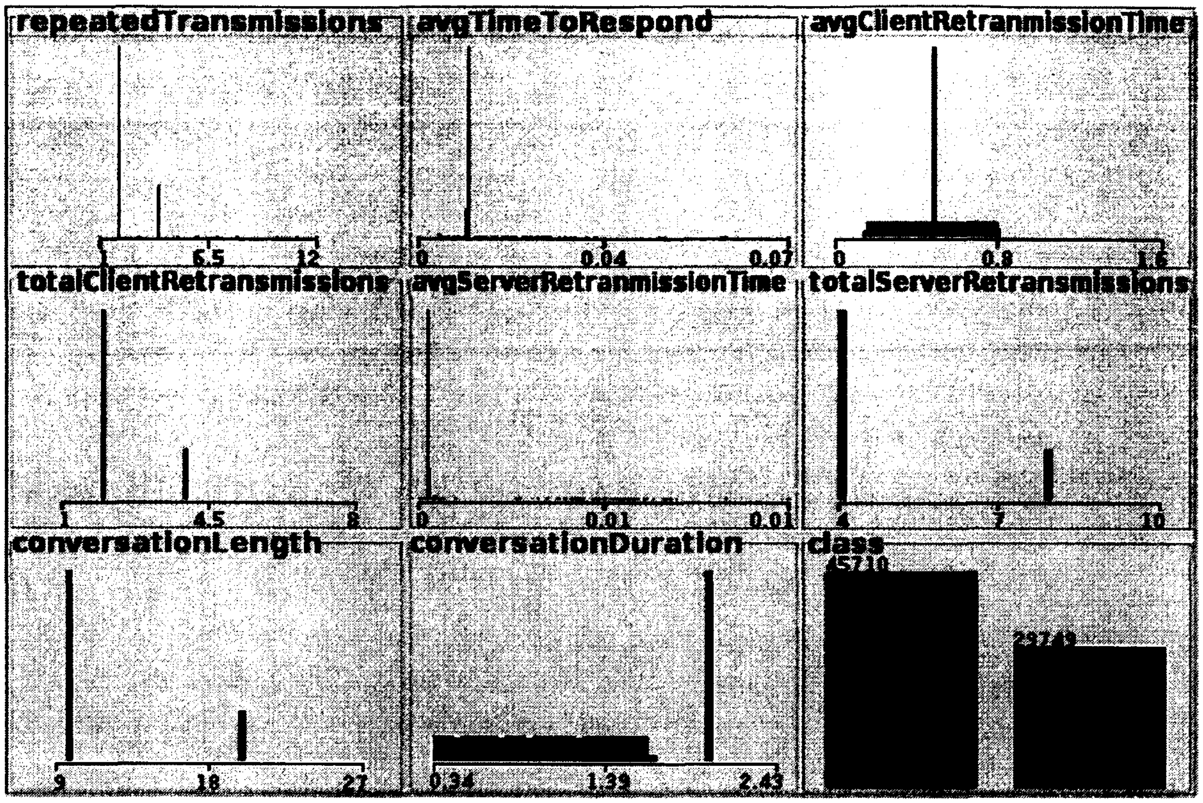

Figure 8.4: Feature Distribution of the data set for C and Python based clients. Blue is malicious and red is benign

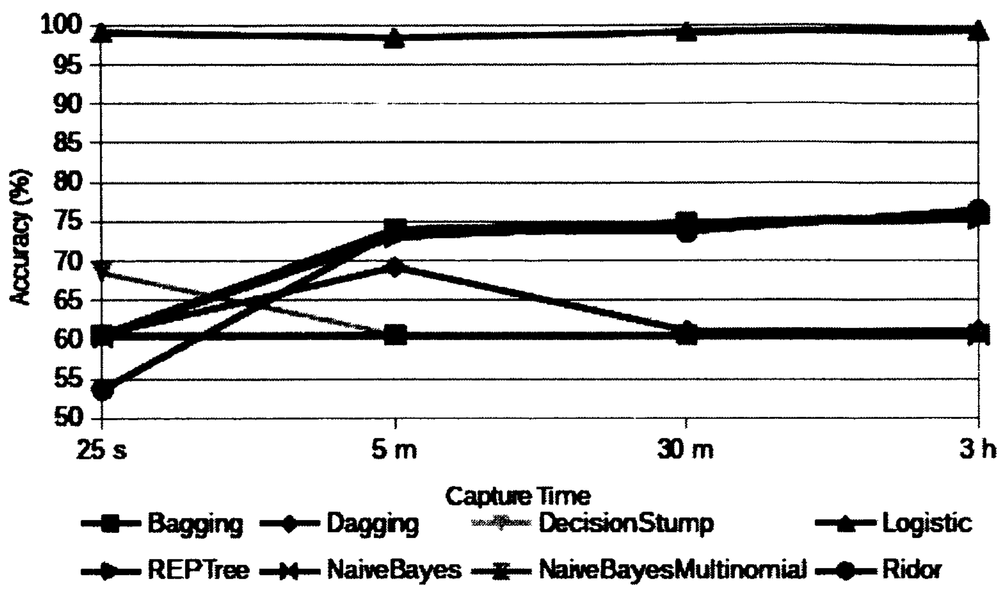

Figure 8.5: Accuracies of Different Capture Intervals

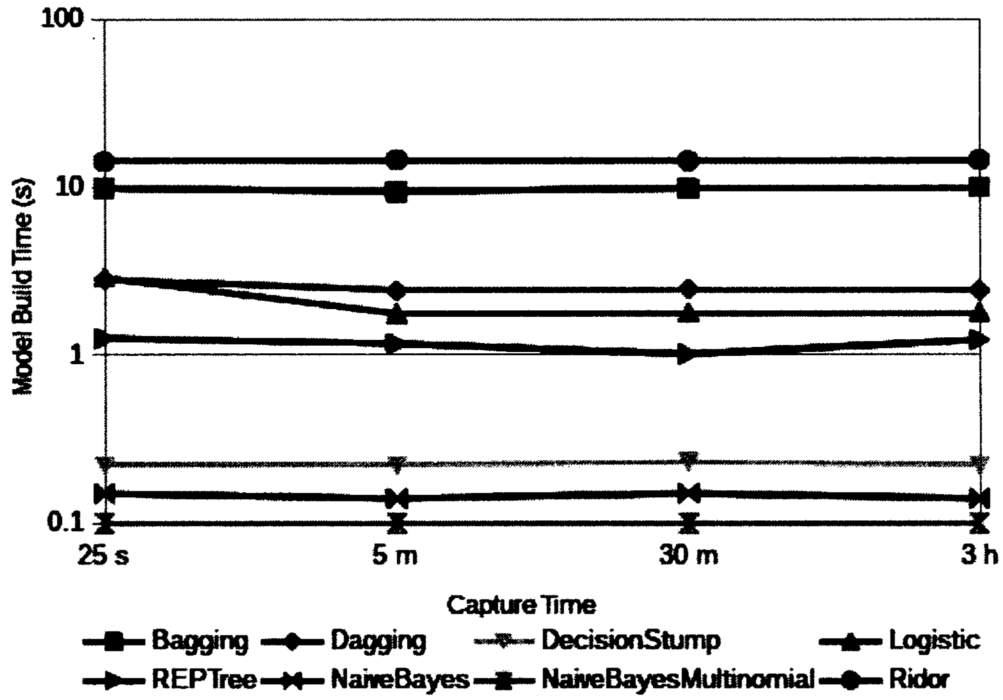

Figure 8.6: Model Build Times for Given Capture Intervals

Based on the feature distributions in Figure 8.4, it is hard to say why the logistic classifier was able to perform so well. However, as the logistic function is known to perform the best with a binary class system, it may be possible that the data provided in this dataset had a high multidimensional logistical probability of belonging to either benign or malicious class.

Figure 8.6 shows the model build times needed to create the models for a given capture time period. The y-axis of the graph - Model Build Time uses a logarithmic scale to better show the differences between classifiers. Most of the classifiers had very similar build times despite orders of magnitude variance of the size of the training data set. However, the pattern among all of the classifiers changed in comparison to the previous section. This suggests that the model time is more dependent on the information and entropy present in the data set rather than data set size. The Logistic classifier was able to achieve both high accuracy classification and an acceptable model build time - 2.89 seconds for 99.0562% accuracy.

8.3 IDS Accuracy for Different Operating Systems

The next step for determining the information present in the chosen features for the developed IDS is identifying the differences in the operating systems of the benign and malicious machines. For this experiment, 3 machines with identical hardware were configured with a version of Windows, Debian, and Gentoo operating systems. For each of those operating systems, a Python interpreter, PyModbus library, and a test SCADA software written in python was uploaded and used for communication with the PLCs protected by the developed IDS. The next three subsections look at accuracy of classifying the difference between two machines, as well as using the developed model to test the accuracy when the third machine attacks.

8.3.1 Gentoo and Windows Classification

Table 8.3 Shows the 10-Fold validation of the Gentoo and Windows dataset. All of the classifiers resulted in a high accuracy classification with the worst accuracy of 99.4843% for a Naïve Bayes Multinomial classifier.

Table 8.3: 10-Fold validation for Gentoo and Windows Classification

Classifier	10-Fold Accuracy (%)
Bagging	99.9938
Dagging	99.927
DecisionStump	99.8838
Logistic	99.9777
REPTree	99.9963
C4.5	99.9913
NaiveBayes	99.9518
NaiveBayesMultinomial	99.4843
Ridor	99.9975

Figure 8.7 Shows classification accuracies for given capture windows. Most classifiers performed well, having accuracies above 99%, however, Dagging and Naïve Bayes Multinomial classifiers improved their accuracy when the capture windows further grew from the maximum 25 seconds used by the experiments in chapter 6. This suggests that the total information available in 25 second capture is not enough for these classifiers to achieve their highest possible accuracy.

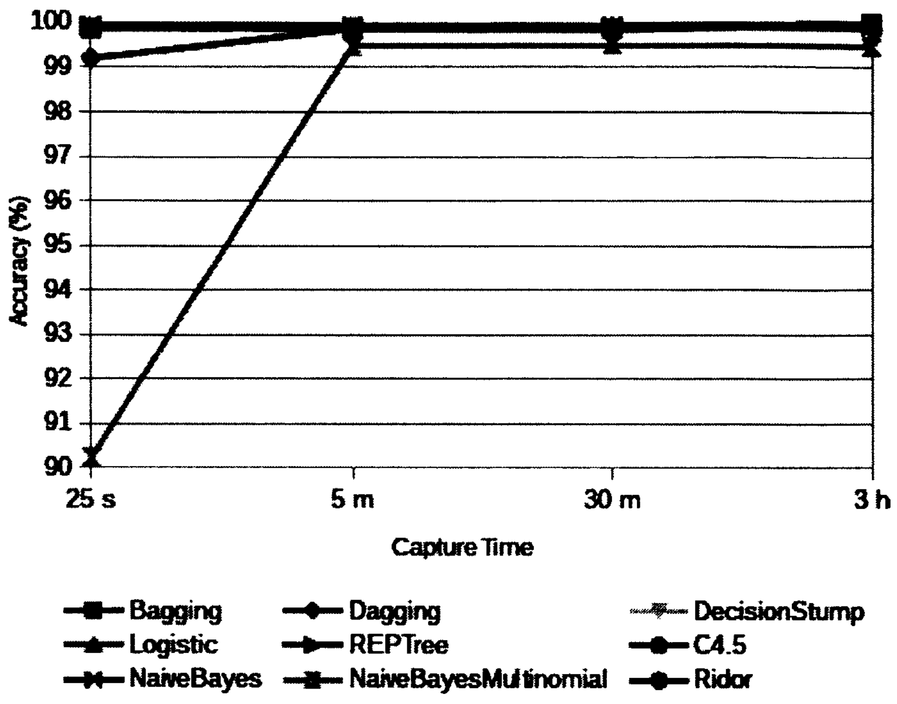

Figure 8.7: Accuracy of Gentoo vs Windows classification

Figure 8.8 shows the model build times for all of the selected classifiers for a given capture time. Once again, the timing changes with the size of the training dataset are orders of magnitude lower than the actual times, except for Naïve Bayes Multinomial and C4.5 classifiers when changing from a 30-minute to a 3-hour capture interval. Both of these classifiers have to iterate over the dataset several times in order to build their models. Both of them were also written in Java for these experiments. With an increased dataset size it is likely that Java's garbage collection was activated in the middle of model building, which resulted in an increased time requirement.

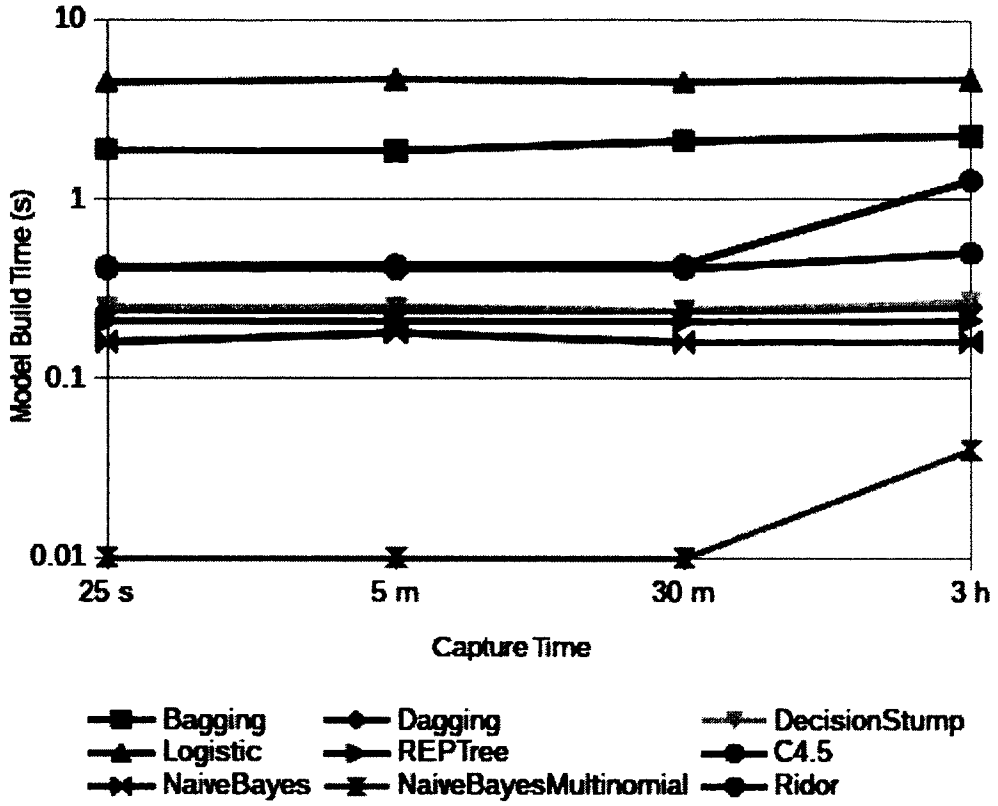

Figure 8.8: Model Build Time of Gentoo vs Windows classification

Figure 8.9 shows the accuracy of classifying a Debian machine when the Gentoo and Windows dataset was used for training. Gentoo was assigned a benign class, Windows was assigned a malicious class. Then the amount of features corresponding to the appropriate capture time on the figure was extracted from the data set and used to train the classifiers. Then, a Debian and Gentoo dataset was used; since Gentoo was the same OS as in the training data set, Gentoo was once again asigned a benign class while Debian was assigned a malicious class. The resulting graph shows the accuracies of classifying the Gentoo and Debian dataset with the Gentoo and Windows training set.

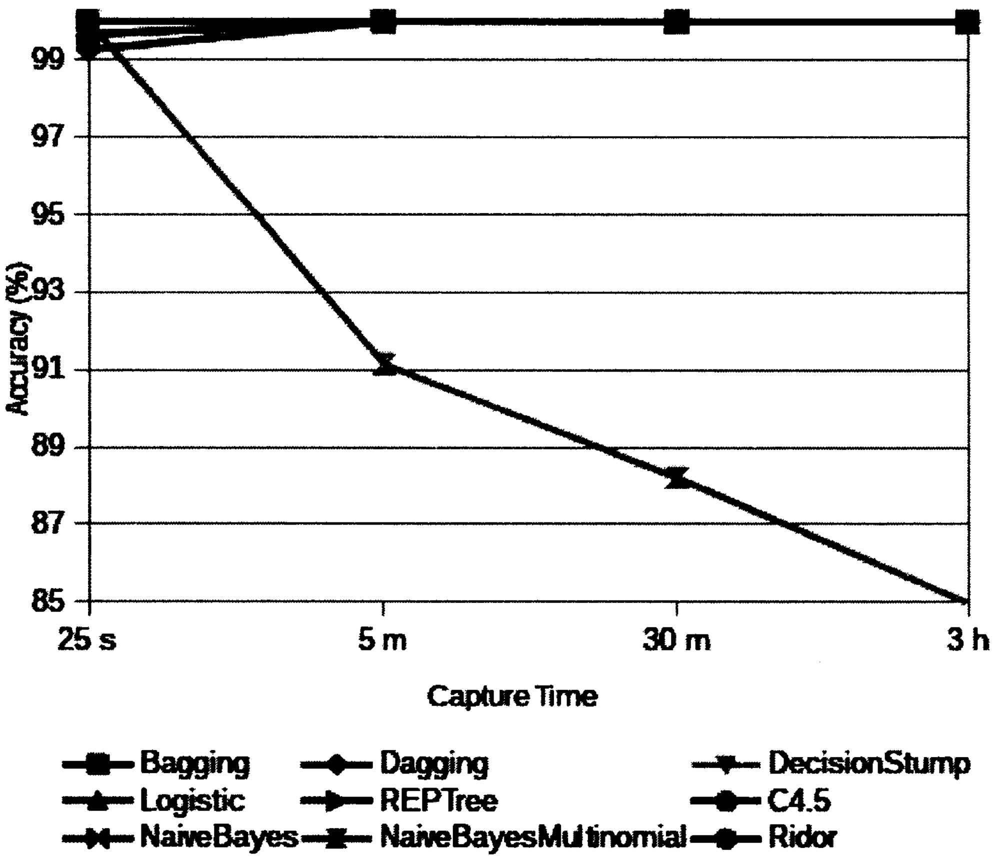

Figure 8.9: Accuracy of Classifying Debian With a Gentoo vs Windows Training Set

8.3.2 Debian And Windows Classification

Table 8.4 Shows the 10-Fold validation of the Debian and Windows dataset. All but one of the classifiers resulted in a high accuracy classification. Naïve Bayes multinomial performed the worst, having an accuracy of 65.1597%, however the second worst accuracy is Dagging based ensemble modeling with 90.7043% accuracy.

Table 8.4: 10-Fold Validation for Debian and Windows Classification

Classifier	**10-Fold Accuracy (%)**
Bagging	97.755
Dagging	90.7043
DecisionStump	95.5253
Logistic	90.9226
REPTree	97.7360
C4.5	97.6565
NaiveBayes	91.1266
NaiveBayesMultinomial	65.1597
Ridor	97.5332

Figure 8.10 shows the accuracies of classifiers when classifying the Debian and Windows dataset. Unlike the Gentoo and Windows results, accuracies given in this experiment are lower. This suggests that the timing patterns of the Debian operating system are closer to the Windows operating system than Gentoo. However, the differences are still prevalent in order for the most classifiers to achieve an accuracy above 90%.

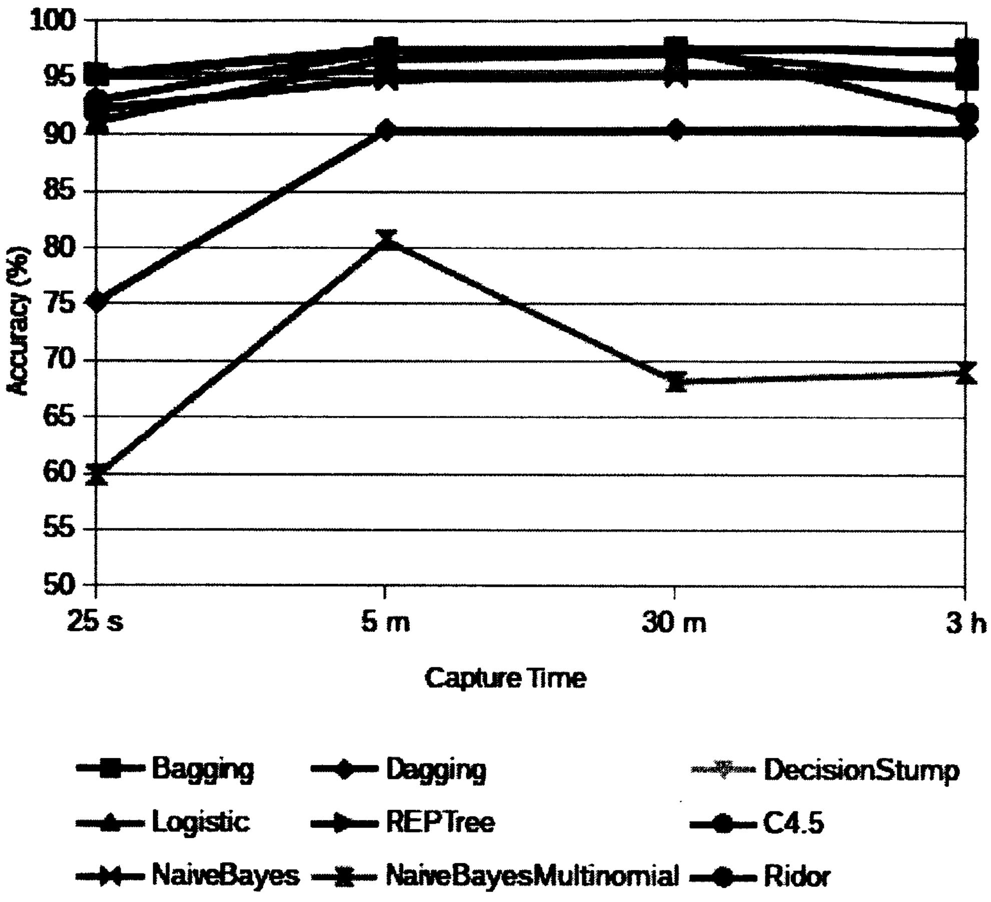

Figure 8.10: Accuracy of Debian vs Windows Classification

Figure 8.11 shows the model build times for the Debian and Windows dataset. As expected, the times differ with a change in the classifier and not with a change in the training dataset size. However, model build time of Dagging based ensemble modeling increased by an order of magnitude in comparison to the Gentoo and Windows dataset.

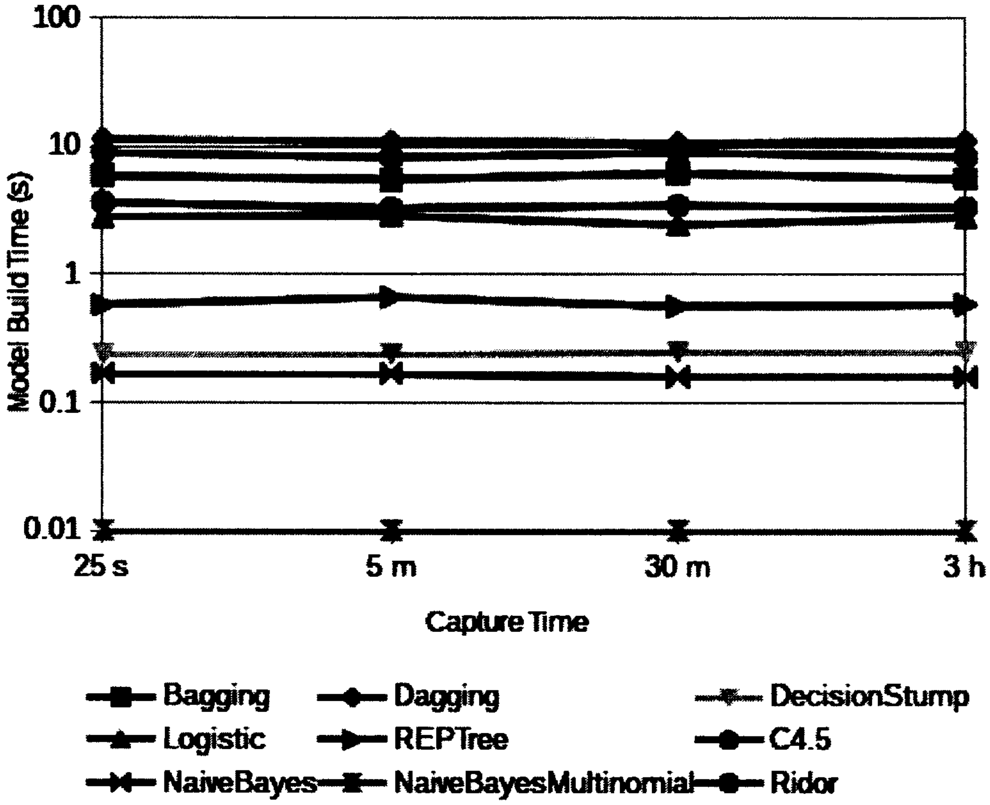

Figure 8.11: Model Build Time of Debian vs Windows classification

Figure 8.12 shows the accuracy of classifying Gentoo machine when the Debian and Windows dataset was used for training. Though most of the classifiers did not perform as well, some of them were able to learn enough information using a 3-hour dataset to achieve an accuracy above 90%. Naïve Bayes was also able to achieve an accuracy of 97.7238% when using 25s of the training data set.

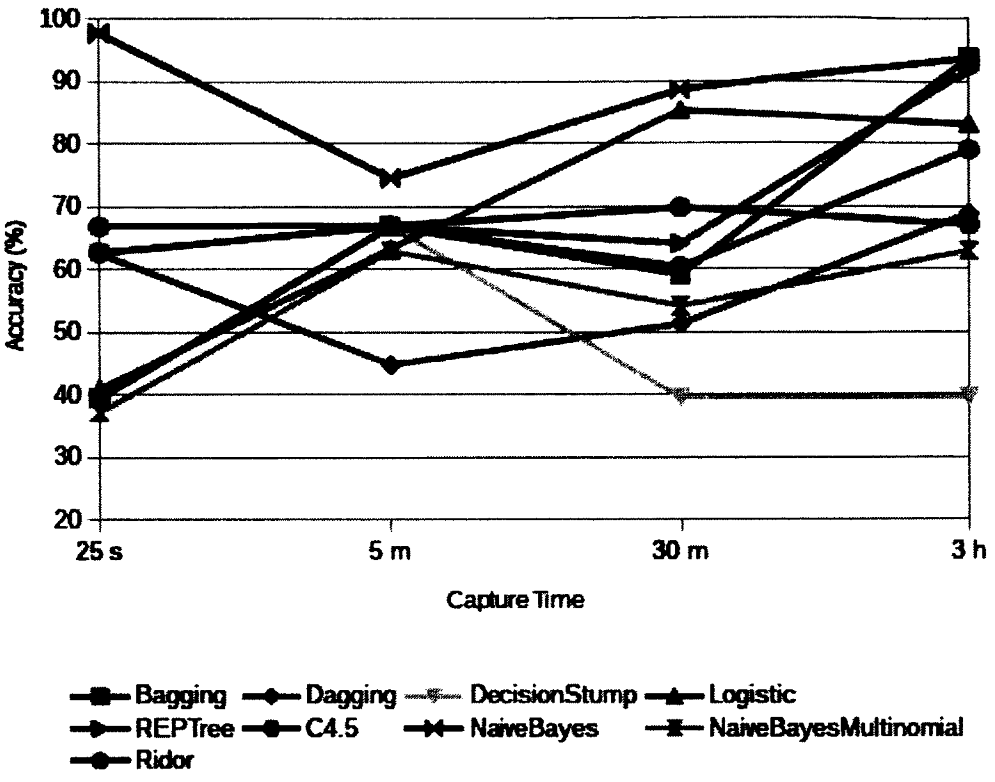

Figure 8.12: Accuracy of Classifying Gentoo With a Debian vs Windows Training Set

8.3.3 Debian And Gentoo Classification

Table 8.5 Shows the 10-Fold validation of the Debian and Gentoo dataset. Once again, all but the Naïve Bayes Multinomial classification was very high, with the second worst being Naïve Bayes, at 98.9167%.

Table 8.5: 10-Fold Validation for Debian and Gentoo Classification

Classifier	10-Fold Accuracy (%)
Bagging	99.9839
Dagging	99.8875
DecisionStump	96.7971
Logistic	99.9518
REPTree	99.9852
C4.5	99.9815
NaiveBayes	98.9167
NaiveBayesMultinomial	72.3898
Ridor	99.9889

Figures 8.13 and 8.14 show the accuracy variation and model build time for all of the selected classifiers when varying the capture time. Once again, Dagging and Naïve Bayes Multinomial were the major outliers while the rest of the classifiers performed well. However, Dagging accuracy increased as the training set capture time increased to 5-minute interval.

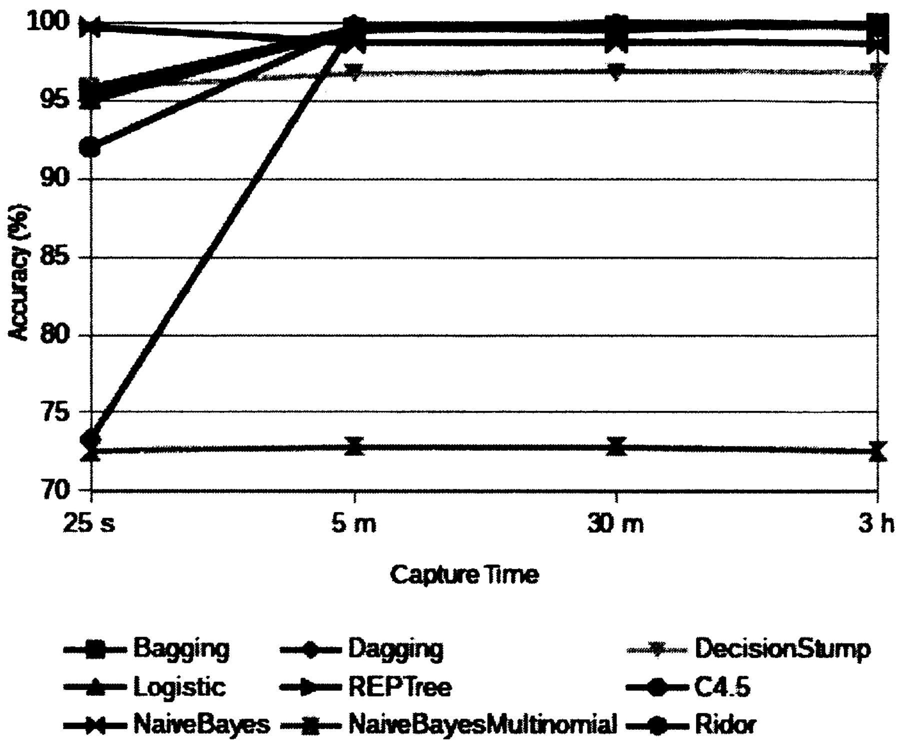

Figure 8.13: Accuracy of Debian vs Gentoo Classification

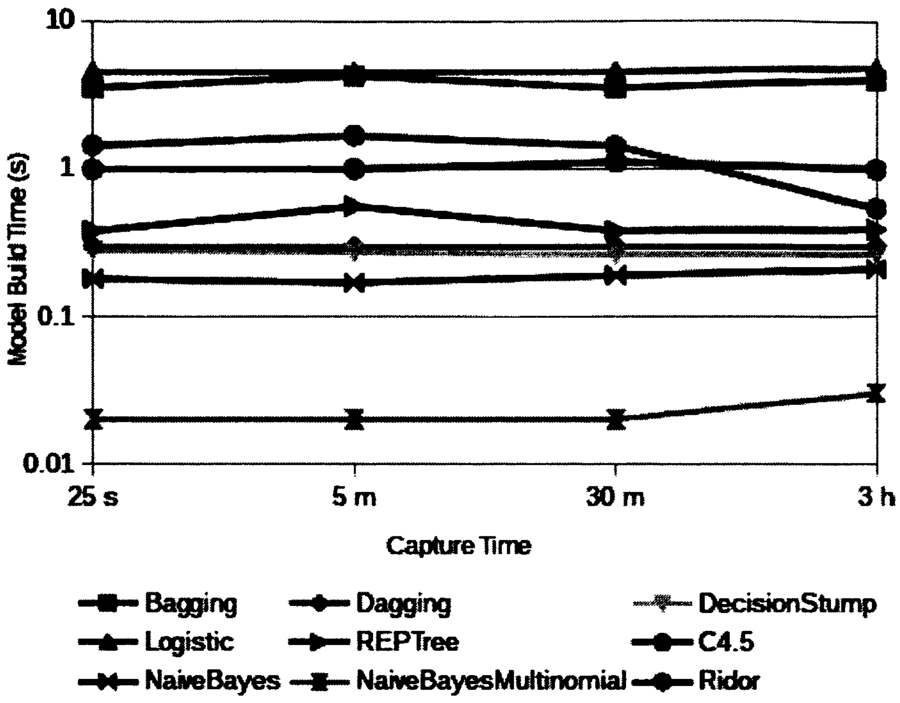

Figure 8.14: Model Build Time of Debian vs Gentoo Classification

Finally, Figure 8.15 shows the accuracies of classifying Windows machine with the training dataset of Debian and Gentoo. The results are much worse for most of the classifiers. The author speculates that the Windows machine's features are too different than the Debian and Gentoo machines, resulting in the confusion of the classifiers. Dagging was able to achieve the best accuracy of 74.6119% with a training dataset of 25-second capture time.

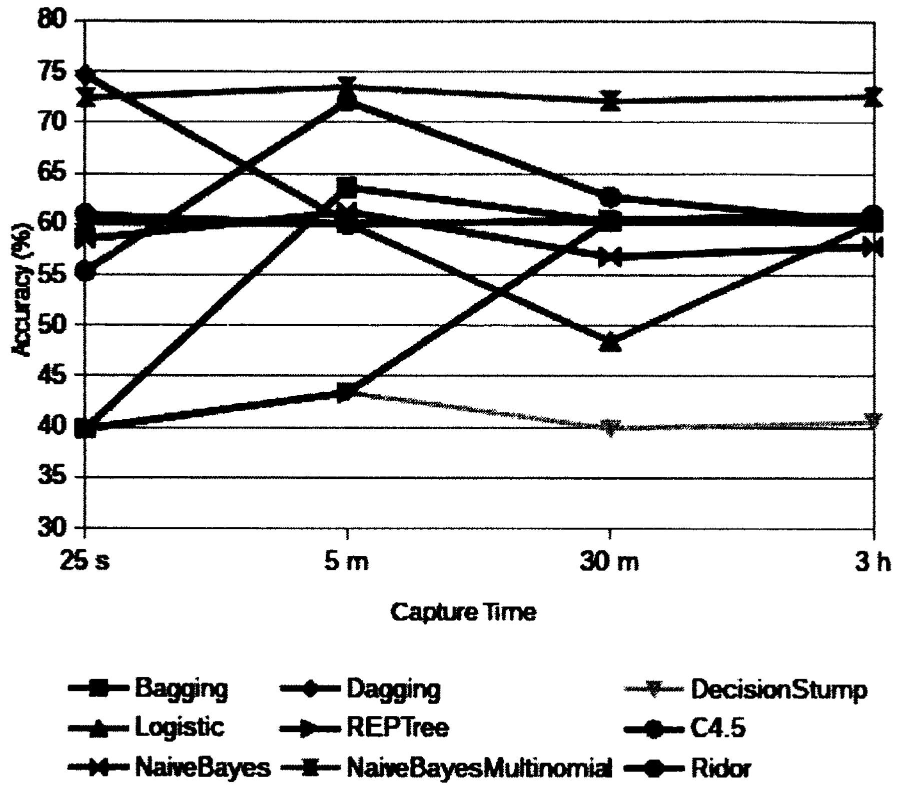

Figure 8.15: Accuracy of Classifying Windows With a Debian vs Gentoo Training Set

While the decreased differences of classification of the Windows machine while using Debian and Gentoo datasets shows potential pitfalls of the IDS, such a condition occurred only when there were no hardware differences between the machines. Different hardware should introduce more information to the system that would allow for a more accurate classification. In addition, the classifiers were only given a single machine as an example of malicious communication. Increasing the machine count given to the training stage of classifiers should improve classification accuracy.

The results also show that a high accuracy classification of a machine not present in the training dataset is possible. The IDS can, therefore be trained against a few malicious machines to be able to differentiate among many machines during the normal operation. This method makes such an IDS a viable candidate for an ICS deployment.

CHAPTER 9

CONCLUSIONS

This dissertation covers the development and testing of a Telemetry Based Intrusion Detection System for Industrial Control Systems. An IDS that uses network telemetry can be created and it can achieve a high classification accuracy, protecting nodes from malicious traffic. Such an IDS will not be vulnerable to address or encryption spoofings, as it does not utilize the content of the packets to differentiate between malicious and benign traffic; rather, it uses features of timing and network sessions to determine whether the machine that sent a particular packet is, in fact, a machine that is benign, as well as whether or not it resides on a network that is benign.

The results of the experiments conducted for this dissertation establish that such a system is possible to create and use in an environment of ICS networks. Several features are recognized and selected as means for fingerprinting the hardware and software characteristics of the SCADA system that can be used in pair with machine learning algorithms to allow for a high accuracy detection of intrusions into the ICS network. The features are extracted from the TCP flow model and include:

- Time it takes the client to respond to server's message.
- Amount of client-side dropped packets.

- Amount of server-side dropped packets.
- Time between the repeated packet transmissions when packet drops happen.

Several feature extraction parameters are tested and optimized for maximum performance. To determine session boundaries for feature extraction, silence based interval of 0.3 seconds was found to result in the best accuracy while maintaining a high amount of malicious and benign sessions. Each session is considered malicious if it includes as least one malicious packet.

Then, the IDS was tested against identical machines, machines that were identical in hardware but used different programming languages for SCADA communication software, machines with different operating systems, machines with different hardware specifications that were located on the same network, and machines with different hardware specifications that were located on different networks. Majority of results showed a classification accuracy of at least 95% was possible, and as the differences between machines increased, the accuracy increased too.

However, during one of the experiments with the operating systems, it was determined that the Windows machine was too different for the classifiers to perform accurately when the training dataset contained only linux based machines. The decreased differences of classification of the Windows machine while using Debian and Gentoo datasets shows potential pitfalls of the IDS, such a condition occurred only when there were no hardware differences between the machines. Different hardware should introduce more information to the system that would allow for a more accurate classification. In addition, the classifiers were only give a single machine as an example of malicious communication. Increasing the machine count given to the training stage of classifiers should improve classification accuracy.

The primary goal of further evaluation of the presented research is to test the developed IDS on more machines at once. Given several malicious machines during the training stage, the IDS should perform well on a larger amount of unknown machines. Due to the high accuracy classification of only two machines, however, the author believes it may be possible to use machine learning to create attack features of any given machine, allowing the IDS not only to detect an incoming attack, but to verify if the machine has attempted an attack before.

APPENDIX A

CODE LISTINGS

Source Code 1 Python version of the polling SCADA System

```
import sys, time
from pymodbus.client.sync import ModbusTcpClient

if len(sys.argv) < 2:
  sys.exit("Usage: %s IP" % sys.argv[0])


client = ModbusTcpClient(sys.argv[1], 5502)

while True:
  try:
    result = client.read_coils(1,20, unit=0x01)
    tmpStr = "20 coils: "
    for x in range(1,20):
      tmpStr += str(result.bits[x-1])+" "
    print tmpStr
    #time.sleep(1)

    result = client.read_discrete_inputs(10001,8, unit=0x01)
    tmpStr = "8 descrete inputs: "
    for x in range(1, 8):
      tmpStr += str(result.bits[x-1])+" "
    print tmpStr
    #time.sleep(1)

    result = client.read_input_registers(30001, 8, unit=0x02)
    tmpStr = "8 analog values: "
    for x in range(1, 8):
      tmpStr += str(result.registers[x-1])+" "
    print tmpStr
    #time.sleep(1)

    result = client.read_holding_registers(40001, 8, unit=0x02)
    tmpStr = "8 holding registers: "
    for x in range(1, 8):
      tmpStr += str(result.registers[x-1])+" "
    print tmpStr
    time.sleep(2)
  except KeyboardInterrupt:
    print 'KeyboardInterrupt caught'
    break


#result = client.read_coils(1,1, unit=0x01)
#print result.bits[0]

client.close()
```

Source Code 2 C version of the polling SCADA System

```
#include <stdio.h>
#include <unistd.h>
#include <modbus/modbus.h>

int main(int argc, char *args[]) {
  modbus_t *mb;
  uint16_t tab_reg[32];
  uint8_t coils[32];
  int i;

  mb = modbus_new_tcp("192.168.1.27", 502);
  modbus_connect(mb);

  while (1) {
  modbus_set_slave(mb, 1);
  modbus_read_bits(mb, 1, 20, coils);
  printf("20 coils: ");
  for (i=0; i<20; ++i) {
    printf("%d ", coils[i]);
  }
  printf("\n");

  modbus_read_input_bits(mb, 10001, 8, coils);
  printf("8 descrete inputs: ");
  for (i=0; i<8; ++i) {
    printf("%d ", coils[i]);
  }
  printf("\n");

  modbus_set_slave(mb, 2);
  modbus_read_input_registers(mb, 30001, 8, tab_reg);
  printf("8 analog values: ");
        for (i=0; i<8; ++i) {
                printf("%d ", tab_reg[i]);
        }
        printf("\n");
  modbus_read_registers(mb, 40001, 8, tab_reg);
  printf("8 holding registers: ");
        for (i=0; i<8; ++i) {
                printf("%d ", tab_reg[i]);
        }
        printf("\n");

  Sleep(2000);
  }
  modbus_close(mb);
  modbus_free(mb);
}
```

BIBLIOGRAPHY

[1] Recommended practice: Improving industrial control systems cybersecurity with defense-in-depth strategies. Technical report, Department of Homeland Security, 2009.

[2] B Drury. The control techniques, drives and controls handbook, volume 57 of iet power and energy series. *The Institution of Engineering and Technology, Stevenage, United Kingdom,*, 2009.

[3] Ken Curtis. A dnp3 protocol primer. *DNP User Group*, 2005.

[4] Jingcheng Gao, Jing Liu, Bharat Rajan, Rahul Nori, Bo Fu, Yang Xiao, Wei Liang, and CL Philip Chen. Scada communication and security issues. *Security and Communication Networks*, 7(1):175–194, 2014.

[5] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. Technical report, DTIC Document, 2006.

[6] Andrew Hodges. Alan turing and the turing machine. In *The Universal Turing Machine A Half-Century Survey*, pages 3–14. Springer, 1995.

[7] Gary McGraw and Greg Morrisett. Attacking malicious code: A report to the infosec research council. *Software, IEEE*, 17(5):33–41, 2000.

[8] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 2011.

[9] Bruce Schneier. The story behind the stuxnet virus. *Forbes. com*, 2010.

[10] Aleksandr Matrosov, Eugene Rodionov, David Harley, and Juraj Malcho. Stuxnet under the microscope. *ESET LLC (September 2010)*, 2010.

[11] L Todd Heberlein and Matt Bishop. Attack class: Address spoofing. In *Proceedings of the 19th National Information Systems Security Conference*, pages 371–377, 1996.

[12] Bonnie Zhu, Anthony Joseph, and Shankar Sastry. A taxonomy of cyber attacks on scada systems. In *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 380–388. IEEE, 2011.

[13] Keith Stouffer, Joe Falco, and Karen Scarfone. Guide to industrial control systems (ics) security. *NIST Special Publication*, pages 800–82, 2011.

[14] Siddharth Sridhar, Adam Hahn, and Manimaran Govindarasu. Cyber–physical system security for the electric power grid. *Proceedings of the IEEE*, 100(1):210–224, 2012.

[15] Yilin Mo, Sean Weerakkody, and Bruno Sinopoli. Physical authentication of control systems: Designing watermarked control inputs to detect counterfeit sensor outputs. *Control Systems, IEEE*, 35(1):93–109, 2015.

[16] Juan Lopez. Personal Communication, December 2013.

[17] Modbus Application Protocol Specification Modbus-IDA. V. 1.1 b. *Hopkinton, Massachusetts (www. modbus. org/docs/Modbus Application Proto col V1 1b. pdf)*, 2006.

[18] Andy Swales. Open modbus/tcp specification.

[19] Igor Nai Fovino, Andrea Carcano, T De Lacheze Murel, Alberto Trombetta, and Marcelo Masera. Modbus/dnp3 state-based intrusion detection system. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 729–736. IEEE, 2010.

[20] Andrey Belenky and Nirwan Ansari. Ip traceback with deterministic packet marking. *IEEE Communications Letters*, 7(4):162–164, 2003.

[21] Yong Sheng, Keren Tan, Guanling Chen, David Kotz, and Andrew Campbell. Detecting 802.11 mac layer spoofing using received signal strength. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1768–1776. IEEE, 2008.

[22] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in ssh: provably fixing the ssh binary packet protocol. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 1–11. ACM, 2002.

[23] Klaas Apostol. *Brute-force Attack.* SaluPress, 2012.

[24] Yinqian Zhang, Fabian Monrose, and Michael K Reiter. The security of modern password expiration: an algorithmic framework and empirical analysis. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 176–186. ACM, 2010.

[25] Naoum Sayegh, Ali Chehab, Imad H Elhajj, and Ayman Kayssi. Internal security attacks on scada systems. In *Communications and Information Technology (ICCIT), 2013 Third International Conference on*, pages 22–27. IEEE, 2013.

[26] Graham Clifford Goodwin, Stefan F Graebe, and Mario E Salgado. *Control system design*, volume 240. Prentice Hall New Jersey, 2001.

[27] Thomas Morris, Rayford Vaughn, and Yoginder Dandass. A retrofit network intrusion detection system for modbus rtu and ascii industrial control systems. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 2338–2345. IEEE, 2012.

[28] Symantec. Symantec internet security threat report. 2013.

[29] Dillon Beresford. Exploiting siemens simatic s7 plcs. *Black Hat USA*, 2011.

[30] Dong Wei, Livio Dalloro, and Yan Lu. Application layer security proxy for automation and control system networks, June 17 2014. US Patent 8,756,411.

[31] Andrea Carcano, Alessio Coletta, Michele Guglielmi, Marcelo Masera, Igor Nai Fovino, and Alberto Trombetta. A multidimensional critical state analysis for detecting intrusions in scada systems. *Industrial Informatics, IEEE Transactions on*, 7(2):179–186, 2011.

[32] Steven Cheung, Bruno Dutertre, Martin Fong, Ulf Lindqvist, Keith Skinner, and Alfonso Valdes. Using model-based intrusion detection for scada networks. In *Proceedings of the SCADA security scientific symposium*, pages 1–12, 2007.

[33] Men Long, Chwan-Hwa John Wu, and John Y Hung. Denial of service attacks on network-based control systems: impact and mitigation. *Industrial Informatics, IEEE Transactions on*, 1(2):85–96, 2005.

[34] Sangkyo Oh, Hyunji Chung, Sangjin Lee, Kyungho Lee, Su-Hyun Kim, Im-Yeong Lee, Shinsaku Kiyomoto, Yutaka Miyake, Hee Bong Choi, Hyuk Joong Yoon, et al. Advanced protocol to prevent man-in-the-middle attack in scada system. *International Journal of Security and Its Applications*, 8(2):1–8, 2014.

[35] Igor Nai Fovino, Alessio Coletta, Andrea Carcano, and Marcelo Masera. Critical state-based filtering system for securing scada network protocols. *Industrial Electronics, IEEE Transactions on*, 59(10):3943–3950, 2012.

[36] Niv Goldenberg and Avishai Wool. Accurate modeling of modbus/tcp for intrusion detection in scada systems. *International Journal of Critical Infrastructure Protection*, 6(2):63–75, 2013.

[37] GD Kurundkar, NA Naik, and SD Khamitkar. Network intrusion detection using snort. *International Journal of Engineering Research and Applications*, 2(2):1288–1296, 2012.

[38] Thomas H Morris, Bryan A Jones, Rayford B Vaughn, and Yoginder S Dandass. Deterministic intrusion detection rules for modbus protocols. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 1773–1781. IEEE, 2013.

[39] Masood Parvania, Georgia Koutsandria, Vishak Muthukumary, Sean Peisert, Chuck McParland, and Anna Scaglione. Hybrid control network intrusion detection systems for automated power distribution systems. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 774–779. IEEE, 2014.

[40] Nathan Wallace, Stanislav Ponomarev, and Travis Atkison. A dimensional transformation scheme for power grid cyber event detection. In *Proceedings of the CIRSC conference*, pages 1–12, 2014.

[41] Nathan Wallace, Sean Semple, and Travis Atkison. Identification of state parameters for stealthy cyber-events in the power grid using pca. In *PES General Meeting— Conference & Exposition, 2014 IEEE*, pages 1–5. IEEE, 2014.

[42] Christopher Zimmer, Balasubramany Bhat, Frank Mueller, and Sibin Mohan. Intrusion detection for cps real-time controllers. In *Cyber Physical Systems Approach to Smart Electric Power Grid*, pages 329–358. Springer, 2015.

[43] Jorge Valenzuela, Jianhui Wang, and Nancy Bissinger. Real-time intrusion detection in power system operations. *Power Systems, IEEE Transactions on*, 28(2):1052–1062, 2013.

[44] Matti Mantere, Ilkka Uusitalo, Mirko Sailio, and Sami Noponen. Challenges of machine learning based monitoring for industrial control system networks. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 968–972. IEEE, 2012.

[45] Matti Mantere, Mirko Sailio, and Sami Noponen. Network traffic features for anomaly detection in specific industrial control system network. *Future Internet*, 5(4):460–473, 2013.

[46] Matti Mantere, Mirko Sailio, and Sami Noponen. A module for anomaly detection in ics networks. In *Proceedings of the 3rd International Conference on High Confidence Networked Systems*, pages 49–56. ACM, 2014.

[47] Wei Gao, Thomas Morris, Bradley Reaves, and Drew Richey. On scada control system command and response injection and intrusion detection. In *eCrime Researchers Summit (eCrime), 2010*, pages 1–9. IEEE, 2010.

[48] Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Jung-Eun Kim, and Lui Sha. Securecore: A multicore-based intrusion detection architecture for real-time embedded systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pages 21–32. IEEE, 2013.

[49] J Visumathi, KL Shanmuganathan, and KA Muhamed Junaid. Misuse and anomaly-based network intrusion detection system using fuzzy and genetic classification algorithms. *Fuzzy Systems*, 4(4):137–141, 2012.

[50] Fanglu Guo and Tzi-cker Chiueh. Sequence number-based mac address spoof detection. In *Recent Advances in Intrusion Detection*, pages 309–329. Springer, 2006.

[51] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[52] Daniele Apiletti, Elena Baralis, Tania Cerquitelli, and Vincenzo DElia. Characterizing network traffic by means of the netmine framework. *Computer Networks*, 53(6):774–789, 2009.

[53] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining Network Data*, pages 281–286. ACM, 2006.

[54] Otis Alexander, Hagen Lauer, Nicolai Kuntze, and Michael Jager. Enhancing intrusion detection in substation networks. 2014.

[55] Min Wei and Keecheon Kim. Intrusion detection scheme using traffic prediction for wireless industrial networks. *Communications and Networks, Journal of*, 14(3):310–318, 2012.

[56] Naoum Sayegh, Imad H Elhajj, Ayman Kayssi, and Ali Chehab. Scada intrusion detection system based on temporal behavior of frequent patterns. In *Mediterranean Electrotechnical Conference (MELECON), 2014 17th IEEE*, pages 432–438. IEEE, 2014.

[57] Nathan Wallace and Travis Atkison. Observing industrial control system attacks launched via metasploit framework. In *Proceedings of the 51st ACM Southeast Conference*, ACMSE '13, pages 22:1–22:4, New York, NY, USA, 2013. ACM.

[58] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit.* Syngress, 2006.

[59] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

[60] Pallavi Asrodia and Hemlata Patel. Analysis of various packet sniffing tools for network monitoring and analysis. *International Journal of Electrical, Electronics and Computer Engineering*, 1(1):55–58, 2012.

[61] Lukas Rist, Johnny Vestergaard, Daniel Haslinger, and John Smith. Conpot ics/scada honeypot.

[62] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling tcp latency. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1742–1751. IEEE, 2000.

[63] Florian Hisch. Performance evaluation of tcp flows. *Network*, 79, 2014.

[64] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Springer, 2013.

[65] David D Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Machine learning: ECML-98*, pages 4–15. Springer, 1998.

[66] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[67] David W Hosmer, Stanley Lemeshow, and Rodney X Sturdivant. *Introduction to the logistic regression model*. Wiley Online Library, 2000.

[68] C Lakshmi Devasena, T Sumathi, VV Gomathi, and M Hemalatha. Effectiveness evaluation of rule based classifiers for the classification of iris data set. *Bonfring International Journal of Man Machine Interface*, 1(Special Issue Inaugural Special Issue):05–09, 2011.

[69] Ian H Witten, Eibe Frank, Leonard E Trigg, Mark A Hall, Geoffrey Holmes, and Sally Jo Cunningham. Weka: Practical machine learning tools and techniques with java implementations. 1999.

[70] John Ross Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan Kaufmann, 1993.

[71] Luis Martin Garcia. Programming with libpcap±sniffing the network from our own application. *Hakin9-Computer Security Magazine*, pages 2–2008, 2008.

[72] Stanislav Ponomarev, Nathan Wallace, and Travis Atkison. Detection of ssh host spoofing in control systems through network telemetry analysis. In *Proceedings of the CIRSC conference*, pages 1–12, 2014.

[73] Robert J Kauffman and Bin Wang. *Bid together, buy together: On the efficacy of group-buying business models in Internet-based selling.* CRC Press Boca Raton, FL, 2002.

[74] Nathan Wallace and Travis Atkison. Observing industrial control system attacks launched via metasploit framework. In *Proceedings of the 51st ACM Southeast Conference*, page 22. ACM, 2013.