8-2017

INTRUSION DETECTION OF A SIMULATED SCADA SYSTEM USING A DATA-DRIVEN MODELING APPROACH

Brien Alen Jeffries
University of Tennessee, Knoxville, bjeffri1@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Brien Alen Jeffries entitled "INTRUSION DETECTION OF A SIMULATED SCADA SYSTEM USING A DATA-DRIVEN MODELING APPROACH." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Nuclear Engineering.

J. Wesley Hines, Major Professor

We have read this dissertation and recommend its acceptance:

Jamie B. Coble, Richard R. Wood, Mark Dean, Kenny C. Gross

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

INTRUSION DETECTION OF A SIMULATED SCADA SYSTEM USING A DATA-DRIVEN MODELING APPROACH

A Dissertation Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Brien Alen Jeffries
August 2017

ACKNOWLEDGEMENTS

I would first like to thank my committee members for taking the time to review this work: Dr. J. Wesley Hines, Dr. Jamie Coble, Dr. Richard Wood, Dr. Mark Dean and Dr. Kenny Gross. Next, I would like to thank Dr. Hines for his support and guidance throughout my graduate studies. Last, I would like to thank Dr. Gross from Oracle for the generous financial support and invaluable advice during this research.

From the UTNE support staff, I would especially like to thank Ashly Pearson for all her help with the technology side of this cyber-security research. She was instrumental in helping set up the experiment, guiding me through Linux, and was also a patient teacher.

Last, but not least, I would like to thank one member of my family for all of his support during my studies. Michael, thank you so much for believing in me and listening to me complain about one thing or the other related to my studies over these long years. Finally, thank you for helping me move and the awesome rubies!

ABSTRACT

Supervisory Control and Data Acquisition (SCADA) are large, geographically distributed systems that regulate help processes in industries such as nuclear power, transportation or manufacturing. SCADA is a combination of physical, sensing, and communications equipment that is used for monitoring, control and telemetry acquisition actions. Because SCADA often control the distribution of vital resources such as electricity and water, there is a need to protect these cyber-physical systems from those with possible malicious intent. To this end, an Intrusion Detection System (IDS) is utilized to monitor telemetry sources in order to detect unwanted activities and maintain overall system integrity.

This dissertation presents the results in developing a behavior-based approach to intrusion detection using a simulated SCADA test bed. Empirical modeling techniques known as Auto Associative Kernel Regression (AAKR) and Auto Associative Multivariate State Estimation Technique (AAMSET) are used to learn the normal behavior of the test bed. The test bed was then subjected to repeated intrusion injection experiments using penetration testing software and exploit codes. Residuals generated from these experiments are then supplied to an anomaly detection algorithm known as the Sequential Probability Ratio Test (SPRT). This approach is considered novel in that the AAKR and AAMSET, combined with the SPRT, have not been utilized previously in industry for cyber-security purposes.

Also presented in this dissertation is a newly developed variable grouping algorithm that is based on the Auto Correlation Function (ACF) for a given set of input data. Variable grouping is needed for these modeling methods to arrive at a suitable set of predictors that return the lowest error in model performance.

The developed behavior-based techniques were able to successfully detect many types of intrusions that include network reconnaissance, DoS, unauthorized access, and information theft. These methods would then be useful in detecting unwanted activities of intruders from both inside and outside of the monitored network. These developed methods would also serve to add an additional layer of security. When compared with two separate variable grouping methods, the newly developed grouping method presented in this dissertation was shown to extract similar groups or groups with lower average model prediction errors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS AND SYMBOLS

AAKR	Auto Associative Kernel Regression
AAMSET	Auto Associative Multivariate State Estimation Technique
ACF	Auto Correlation Function
CPS	Cyber Physical System
CSTH	Continuous System Telemetry Harness
CV	Cross Validation
CVE	Common Vulnerabilities and Exposures
DoS	Denial of Service
FAP	False Alarm Probability
FNR	False Negative Rate
FPR	False Positive Rate
HIDS	Host Intrusion Detection System
HMI	Human Machine Interface
HRSG	Heat Recovery Steam Generator
I&C	Instrumentation and Control
IDS	Intrusion Detection System
LAN	Local Area Network
MAP	Missed Alarm Probability
MTU	Master Terminal Unit
NIDS	Network Intrusion Detection System
NLNP	Non Linear Non Parametric
NPP	Nuclear Power Plant
OS	Operating System
PLC	Programmable Logic Controller
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
SPRT	Sequential Probability Ratio Test
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TNR	True Negative Rate
TPR	True Positive Rate
VGM	Variable Grouping Method
VLAN	Virtual Local Area Network
WAN	Wide Area Network

1 INTRODUCTION

The advancements in computer technology over the past few decades have also seen a trend in the implementation of Cyber-Physical Systems (CPS) for all manner of process control. The CPS is typically a large scale, geographically distributed system that consists of physical components such as process equipment, sensors, and actuators. There is also a so-called cyber component that uses proprietary communication protocols to monitor, send data or issue commands from servers and/or operators in a distant control center. This combination of sensing and control technologies has seen use in a variety of industries that include nuclear, oil/gas, hydroelectric, automotive and railway transportation, to name a few. These systems are used where automation is needed or to control equipment in remote environments where having dedicated staff would not be beneficial. The Supervisory Control and Data Acquisition (SCADA) system combines data sensing, acquisition, and proprietary communications protocols to monitor and control equipment and processes. Nowadays, SCADA is synonymous with CPS.

Given that SCADA systems control the distribution of many vital resources such as energy and water, combined with the recent trend in many industries to shift to a digital Instrumentation and Control (I&C) culture, there is a need to protect the integrity and safety of these systems from malicious, unauthorized or even authorized users. Damage or interruption of access to these systems could lead to loss of power for customers, disruption of irrigation for crops, identity theft, or at worst, loss of life if the SCADA system controls passenger transportation. Though major incidents involving SCADA systems like Stuxnet are thankfully few or not reported, there are some well known examples recently and in the past that will serve to illustrate the aforementioned losses. In the nuclear industry, the Slammer worm attacked the Davis-Bessie power plant in 2003 [1]. The main effect of this attack was that several monitoring stations were unavailable for several hours. However, the plant was not operating at the time, so there was no immediate threat to the safety systems or to the general public. In 2007, an engineer disrupted a SCADA system that regulated water flow from the Sacramento River that was used for consumers and irrigation purposes [2]. This disruption led again to lost revenue and a lengthy prison sentence for the engineer. Last, in 2016 two passenger trains in Germany collided, resulting in injury and loss of life [3]. The final cause of the accident was determined to be the operator at the main facility. Incorrect inputs were given to the conductors and the system alerts were overridden even though two trains were traveling towards each other on the same track. This example shows that not all threats to the integrity of the SCADA system are from outsiders, and that authorized users are able to disrupt the system. These and other security incidents related to SCADA systems will be discussed in more detail later.

With these examples in mind, the threats to these systems will become more sophisticated and numerous as more industries shift to digital I&C. A threat to the security of these systems can be defined as misuse to gain privileges, data or control aspects of the system, such as valve actuation [4]. These security threats can also be termed zero-day, short, and long duration. Zero-day attacks are those that are previously unknown and can escape detection from many detection techniques. They are also typically fast acting, like short duration attacks. Short attacks are those that can cause an immediate disruption to the system, while long term attacks typically inject code that masquerades the intruder as an authorized user in order to steal privileged data or disrupt system processes. To catch the majority of these threats, a detection system needs to be in place. An Intrusion Detection System (IDS) can be thought of as any combination of hardware and/or software that monitors various system telemetry and states in an attempt to detect any unauthorized users or actions. The IDS may examine telemetry or logs for attack signatures of known intrusions, deviations from defined normal behavior, or combinations of these techniques. These systems can be passive or reactive, which means that either an alarm is simply reported when some rule is broken or action is taken to evict the intruder and/or isolate the affected system. To address the security needs of SCADA systems, there are several varieties of IDS that have been developed in open literature. In the broadest terms, the IDS are classified by what audit or telemetry data is examined for intrusion events, this then drives the detection technique. The three main categories of detection systems are termed knowledge, behavior or behavior-specification-based [5].

In knowledge-based systems, the method only looks for specific signatures that correspond to previously known and mitigated attacks. These signatures are contained in what is known as an attack dictionary. Any observed signatures that are not contained in the dictionary are treated as normal; this can be considered a strength and weakness [6]. The strength is that this type is very easy to implement and has a low rate of false alarms. The main weakness is that this method can never detect zero-day attacks.

In behavior-based systems, models are trained on data reflecting normal operating conditions, any deviations from this learned behavior can then be considered as anomalous [7]. This type requires normal operational data and time for training, more system overhead to operate than knowledge-based systems, and have often yielded high false alarm rates. This particular challenge is addressed later in this thesis by utilizing the sequential probability ration test. Despite these apparent setbacks of the behavior-based IDS, this type can detect zero-day attacks, which is a major security benefit when compared with a system that only utilizes a knowledge-based system. Also, models do not need to be pre-enumerated with this type of system.

In behavior-specification-based systems, formal behaviors and specifications for the system are preprogrammed by a human expert [8]. Alarms are only given when these rules are violated. The main strength of this type is a low false alarm rate since only attacks corresponding to the defined specifications will trigger an alarm. The main weakness is that a human must define all rules, which can be error prone.

Next, depending on what type of IDS is employed, a high rate of false and missed alarms is often observed. These rates can also be thought of as false and missed alarm probabilities. The False Alarm Probability (FAP) is the probability of the current observation being labeled as showing attack signatures when it is in fact in a normal state. The Missed Alarm Probability (MAP) is the probability of labeling an observation showing attack signatures as being in a healthy state when it is in fact in an abnormal state. The danger here is that if high rates for these probabilities are reported by the IDS, then operators will again distrust the alarm outputs and ignore them.

In general, the difficulty in developing IDS for SCADA systems is that changes in technology often design out certain threats and also introduce completely new threats. For example, a known vulnerability in a wireless sensor that allows full system access to any user can be designed out with better software and threat mitigation procedures. Because this vulnerability was known, the IDS could easily be programmed to look at certain log files and blacklist unauthorized users. However, after redesign, this functionality of the IDS may have become obsolete. This means that the IDS would then also need to be updated to keep pace with this technology change. The situation worsens if the newly designed software is found to be exploitable to new zero-day attacks. If the IDS employed for this example sensor is not a behavior-based system, then once again the sensor is vulnerable to attacks and the IDS will report this new, unwanted activity as normal. Because of this constant change in technology, the skills of hackers must and do also change to meet this demand, which means that no system can be considered totally secure.

The methods used in this dissertation attempt to alleviate some of these concerns by developing a behavior-based detection method that employs non-parametric models known as Auto Associative Kernel Regression (AAKR) and Multivariate State Estimation Technique (AAMSET) to learn the normal system behavior. These methods have seen commercial success in industrial applications for sensor degradation and process anomaly detection. These models are combined with an anomaly detection algorithm known as the Sequential Probability Ratio Test (SPRT), which is a binary statistical hypothesis test that is known to have the lowest mathematically possible FAP and MAP and requires the fewest number of observations to arrive at a decision. This means that the SPRT servers to alleviate issues in timeliness of detection.

1.1 Problem Statement

The primary goals in developing a detection system for cyber attack mitigation should include the ability to detect unknown attacks, minimize the occurrence of false/missed alarm rates, and timeliness in detection of malicious activities. Behavior-based methods have the ability to detect unknown attacks without the need of a dedicated attack dictionary, though several of these have been developed. These behavior or model-based IDS include Neural Networks (NN), Genetic Algorithms (GA), classifiers, and various types of regression analysis [9-12]. While each of these methods has their own strengths, the main weakness is either complexity of implementation or high overhead cost for simple models. The focus of this dissertation is on the development of behavior-based IDS that use the AAKR and AAMSET techniques as the main empirical models. These modeling techniques are used to learn the normal behavior of a system from supplied data. These models are easy to implement, train, and have low overhead compute cost. Also, when new operating conditions such as a change in normal system dynamics or the inclusion of additional sensors occur, the models can be easily retrained to accommodate these new conditions with no loss in performance.

Related to these modeling methods is how to select relevant variables from a larger pool of variables that will return the low model prediction errors. The subject of Variable Grouping Methods (VGM) is extensive and can be as complex as some empirical modeling techniques. To this end, a newly developed VGM that is based on the Auto Correlation Function (ACF) is shown. This novel method, termed ACFgroup, utilizes properties of the ACF for a given set of input data to arrive at groups of variables that have similar changes in system dynamics. The developed method is robust to most data sets, regardless of the number of observations, signals, or sampling rates.

Next, many IDS suffer from high FAP and MAP. This issue can lead to alerts being ignored or sensitive systems being compromised. A high FAP will make even the best designed IDS useless because this will reduce monitored component lifetime and lead to an increase of operating costs for needless maintenance. A high FAP can cause administrators to relax the IDS parameters, which can lead to missed attacks or a loss in confidence in the employed detection system. Alternatively, a high MAP can allow outside intruders or unauthorized users to steal proprietary information or disrupt the system, all without being detected. This alternative can also cause operators to distrust the implemented detection system. Several SPRTs are employed in this work to reduce the occurrence of false and missed alarms. The SPRT was chosen because it is known to have the lowest mathematically FAP and MAP. The SPRTs include tests for shifts in the mean and variance of observed telemetry.

The last issue that can arise in any IDS is that monitored data streams are out of phase or have vastly different sampling rates. This problem arises in all systems that use electrical circuits and is due to differences is developed software or hardware. In order for the control processes and detection system to be effective, the data streams need to be synchronized so that all telemetry sources reflect the current state of the system for timely actuation and detection. Time synchronous averaging methods can be used to resample phased telemetry streams. This dissertation describes such a method that matches overlapping times in data streams and interpolates un-sampled regions in the data to representative values.

Finally, a separate consideration with all IDS and their related techniques is that they need to be usable on many different types of operating systems. If a detection system is developed in a specific programming language such as MATLAB, then the IDS is only useful on systems with that specific language. To this end, the algorithms used to develop the AAKR and AAMSET models and all SPRTs have been ported from MATLAB to Java. This means that the developed methods can be used on any computer architecture that supports Java.

1.2 Original Contributions

The research presented in this dissertation describes several original contributions to the field of cyber-security and empirical-based modeling. The developed behavior-based methods have been validated on several data sets taken from the SCADA test bed, while the newly developed VGM has been validated on several different real-world data sets. The original contributions described in this dissertation can be summarized as follows:

- Development of behavior-based intrusion detection methods which use AAKR or AAMSET models to learn normal system behavior from various computer system telemetry sources
- Development of a novel VGM known as ACFgroup that selects relevant variables for empirical-based modeling based on statistical properties of the ACF. The algorithm typically returns groups of variables with lower average model prediction errors when compared with other VGM
- Development of time synchronous averaging technique that resample telemetry sources which have uneven sampling rates
- Implemented the developed IDS methods in Java. This allows these methods to be used on any computer architecture that supports Java

1.3 Organization of Document

In the next chapter, a comprehensive literature review is provided that is related to the major contributions of this work. This includes a full description of SCADA systems and related security incidents, intrusion detection methods, Variable Grouping Methods (VGM), and time averaging techniques.

With this background provided, Chapter 3 presents a description of the related methodologies required for this research. Time synchronous averaging techniques are described first, which are needed to resample time phased telemetry sources. This is followed by a complete description of the newly developed variable selection method termed ACFgroup. Next, the methodology and techniques related to the empirical based modeling used in this dissertation are discussed. The chapter concludes with a description of the Sequential Probability Ratio Test (SPRT), which is used for anomaly detection purposes.

Chapter 4 presents the simulated SCADA test bed and equipment information. This is followed by a discussion of the various penetration testing software and codes used for this research. The chapter concludes with examples of which tested intrusions were successfully detected using this simulation and developed methods.

Chapter 5 first provides a comparison of the developed time synchronous averaging technique with one developed using regression methods. This is followed by several case studies that display the efficacy and usefulness of the newly developed ACFgroup algorithm for variable grouping purposes. These results are compared with two other VGM that uses correlation coefficients to arrive at final variable groups. This chapter concludes by providing the intrusion detection results for two different data sets. Each data set was obtained by subjecting the simulated SCADA test bed to repeated, successive injections of several different types of exploits. This was done to determine which exploits could be detected using the developed methods.

The last chapter provides a summary and conclusions of this dissertation, followed by recommendations for future work that are outside the scope of this presented work.

2 LITERATURE REVIEW

This chapter provides a comprehensive literature review related to SCADA, IDS, VGM and time averaging techniques. The history and current terminology for SCADA systems are provided first, followed by an investigation into the various security vulnerabilities, threats, and incidents. Next, IDS types and methods to reduce false/missed alarms are given. VGM are then discussed, concluding with a survey of time averaging methods.

2.1 SCADA System Overview

Modern SCADA systems emerged in the 1970s by using technology that matured in the railway, radio communications and computer industries [13]. The first SCADA-type system emerged in the early days of the railway industry to solve the problem of how to monitor current traffic and send commands to change track configurations when needed. The initial systems used electric pressure plates attached to track switches. When a train was present, the pressure plates would induce an electrical charge. Using electrical repeaters, the status of the track could then be wired to a central facility, where an operator would then monitor traffic status and issue commands. If the track configurations needed to be switched, commands could be sent via telegraph lines to stations that were connected to the central facility. In those days, this type of information exchange was termed telemetry. In Figure 2-1, the schematic of what can be considered the first precursor to modern control systems is shown.

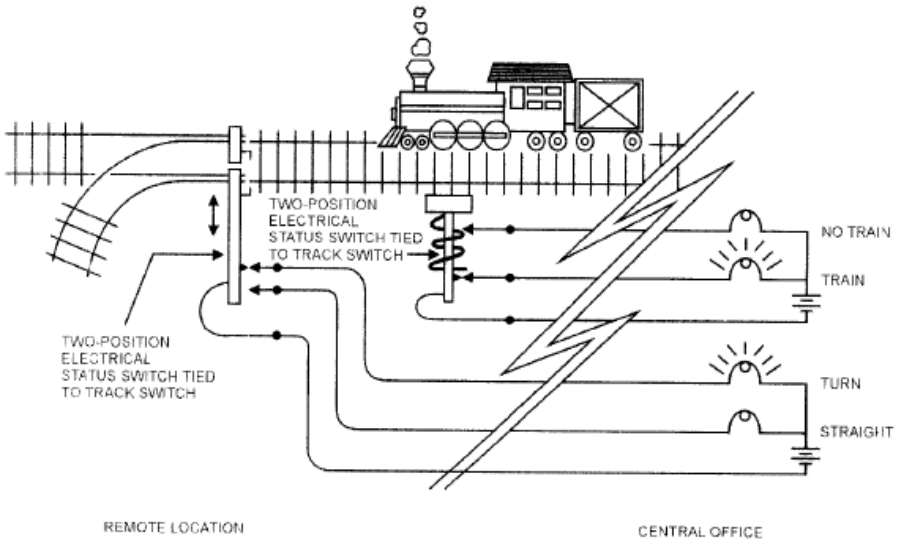


Figure 2-1: Early Railway Control System [13]

This type of monitoring system was at first limited to areas where telegraph wires could be installed and by current technology. Later, radio communications were slowly maturing so that transmitters could be set at remote locations with a simple battery for power needs. Once radio communications could be sent and received, industries such as oil, gas, and railway developed means of controlling simple processes such as valve actuation from a central facility. With the advent of digital computers, larger amounts of data could be collected from the field and the number of processes controlled could be increased as well. As computer technology matured along with the emergence of the Internet, these control systems became more widespread and used in many industries to form the modern SCADA system. The typical components of a modern SCADA system are listed next [14]:

- Human Machine Interface (HMI), interface for engineer to monitor processes and issue override commands
- Master Terminal Unit (MTU) and data historians, the MTU initiates commands and sends processed data to data historians
- Communication equipment for link from the MTU to the field devices
- Remote Terminal Unit (RTU) or Programmable Logic Controllers (PLC), retrieve data from sensors/actuators and issue commands to actuators
- Sensors and actuators, equipment to monitor field devices such as pumps and change configurations of valves
- Field devices, equipment such as pumps or motors that control and maintain monitored physical processes, such as water flow

In Figure 2-2, the components and setup of a generalized modern SCADA system are shown [15].

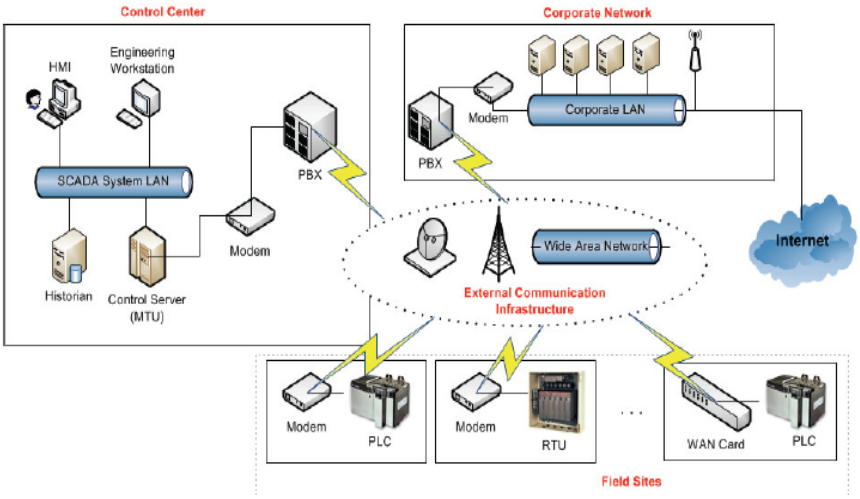


Figure 2-2: Modern SCADA System Design [15]

The generalized SCADA system shown in the previous figure has been used in many industries such as nuclear, oil/gas, electric power generation, chemical, and water distribution [16]. From the above figure, a SCADA system can be further divided into several major groups or layers [17]: control center, corporate network, communication network, field sites and information layer. Each of these different sub layers will be briefly discussed next.

Control Center Layer: This layer consists of the MTU that issues commands and polls field devices for data. This data is processed by the MTU and stored to data historians. The HMI allows the operators to view current system states, issue new commands to the field or reprogram the RTUs or PLCs.

Corporate Network Layer: This layer is seen in the current generation of SCADA systems, it consists of a network at corporate headquarters that can poll the field devices or control center for system states and request data.

Communication Network Layer: Enables the connection and communication between the control center and the field site layer through a Wide Area Network (WAN) by using private connection lines assigned to a particular company. The RTUs commonly use Industrial Ethernet or proprietary communications protocols such as MODBUS, DNP3, Seimens-7 and RS-232.

Field Site Layer: This layer consists of the remote stations or locations that are controlled and monitored by the control center. This layer contains all physical process equipment, sensors, actuators and RTUs used for data sensing and transmission actions.

Information Layer: This last layer is an abstraction but pervades the entire control system; it has two main types of information that pass between each layer. The first is sensor information that is polled from the RTU by the MTU. The other is command information that flows from the control center to the field sites to control process equipment. This information can be an attack target.

The main distinction in SCADA systems is that there is always a monitoring and control element of some process. Depending on the nature of the system, the control actions and telemetry sources can exhibit periodic components. The other distinction is that these systems can be geographically dispersed, often with the control center located hundreds of miles away from the actual physical equipment. This large scale system with many components is desirable because it reduces operating and maintenance costs, and centralizes many control operations. However, because of the nature of these systems, there are several vulnerabilities and threats that can be exploited in all of the previously mentioned integrated layers, discussed in the next section.

2.1.1 SCADA Security Vulnerabilities and Threats

The implementation of SCADA systems also brings some unique vulnerabilities and threats to the system. *Vulnerability* can be defined as a flaw in the design or environment of the system that could allow access to an intruder. Vulnerabilities can arise in developed software, design flaws in equipment or a poorly configured network. They can also be classified as internal or external in nature. A *threat* can be defined as an intruder or agent with the purpose of causing harm to the system in some way, the main avenue to intrusion is usually through the vulnerabilities found in the system. Examples of threats are malicious software that attempts to gain access to privileges and information, criminal intruders or disgruntled employees that attack the system from the inside [18].

Many of the external vulnerabilities found in this type of system are inherent in the design [19]. Because these systems are large in scale and are geographically dispersed; field devices are often located in remote places where human supervision does not exist. This can allow intruders to damage field devices or gain access to sensors to modify telemetry. An example is an intruder that drains the batteries in several RTUs, thus leading to DoS. If field sites are located in remote areas, then repair times can lead to large revenue losses. The field devices may also suffer random failures if located in harsh environments.

Internal vulnerabilities can be caused inadvertently or intentionally by operators. This can arise from inputting incorrect parameters or commands to the system. Intentional actions are caused by disgruntled employees who wish to cause damage to the system. Other internal vulnerabilities include design flaws, bugs in software and in the proprietary communication protocols. These can be exploited by a resourceful intruder. Common cause failures of the equipment or monitoring system can also be considered as a vulnerability of the system. Finally, there can be failures or interruptions in power, network communications, and other unexpected failures that can limit the functionality of the system [20]. The main point here is that vulnerabilities exist in all systems that cannot be designed out; they must be considered when designing the system and when choosing what type of intrusion monitoring system to implement. A summary of these and other vulnerabilities for SCADA systems is given in Table 2-1.

Table 2-1: Common Vulnerabilities of SCADA Systems

External Types	Internal Types
Geographically Dispersed	Operator errors
Large in Scale	Design flaws in software/hardware
Environmental Conditions	Communication/Power Interrupts
Equipment Resources	Internal (disgruntled employees)
Minimal human supervision	Poor System Design

The remainder of this subsection will focus on the various types of security threats that would make a SCADA system vulnerable to intruders. Since SCADA systems are a combination of hardware and software, the security threats will be discussed separately for each of these main component types.

Hardware Security Threats

For this discussion, the term hardware will mean the sensors, actuators, and physical process equipment. The threats to the hardware or physical side of the system first consist of destruction of sensors, equipment, and severing of power and communication lines. These brute force methods are difficult to prevent and can potentially cause the most damage to the system. These threats can also cause large losses of revenue for a resource or business-critical network. Next, the intruder can gain access to sensors or monitored equipment and change reported values. This can have the effect of the wrong command actions being sent due to misinformation. This can potentially lead to information theft and also cause a large amount of damage to the entire system [21]. Other types of sensor related threats are to cut or drain the power supply, thus rendering the sensor and remaining part of the feedback control loop inoperative. Finally, the sensors themselves have security in place, but attackers can break into the security keys by brute force methods, monitoring of access or by dictionary attack [22]. This last type of threat is related to intrusions of the communication protocols of the RTU and will be discussed more fully in the next section.

Software Security Threats

Software threats constitute the majority of attacks against SCADA systems because proprietary communications protocols and other software packages are used to transfer data along the information layer [23, 24].These systems use software packages in sensors, RTU/PLC, all communications, MTU, and data historians. The data historians also contain process and privileged data that would be of interest to an intruder. Many of the software packages used in these systems are written in C, which has its own set of vulnerabilities that can be exploited by a resourceful intruder. The most common flaw seen with software packages is buffer overflows, which can disrupt process actuation. These can include resetting of passwords, malware, spyware or other malicious code injection [25]. However, buffer overflows in these systems can also occur in the field devices or sensor themselves. The field devices typically have low amounts of memory or time allocation settings to complete required tasks. Excessive command actions or DoS can overflow the memory and cause fragmentation. This is a problem because many devices located in remote areas are often not rebooted or updated for years, which can lead to severe memory fragmentation issues [26]. If this occurs, there may be no response to command actions.

Another type of software threat arises from Web applications that use Structure Query Language (SQL). SQL is a programming language that has the ability to store, retrieve or manipulate data on a database. The main components of SQL are queries to store or obtain data and result sets. Internal and external intruders can gain access of the SQL server and inject or manipulate the data stream by inputting unrecognized or unexpected SQL statements. This can allow an intruder to gain complete access to the database and all stored privileged information. Figure 2-3 shows a diagram of how an SQL injection attack can be performed against a SCADA system [27].

When considering the communication layer, the communications protocols used in SCADA systems are also targets for hackers that can lead to severe consequences if an attack is successful [28]. Most of the communications protocols used by these types of systems do not have any means to verify that the data or commands being received are authentic. An intelligent hacker can manipulate these commands for their own ends. Encryption of these protocols does not guarantee security since the keys can stolen or even eventually cracked. These systems are also vulnerable to many DoS attacks such as packet storming, ping flooding, and spoofing of IP packets. These types of attacks send large numbers of packets to the control layer, but at a faster rate than they can be processed. This has the result of a majority of the system resources being utilized to handle the large increase of incoming packets.

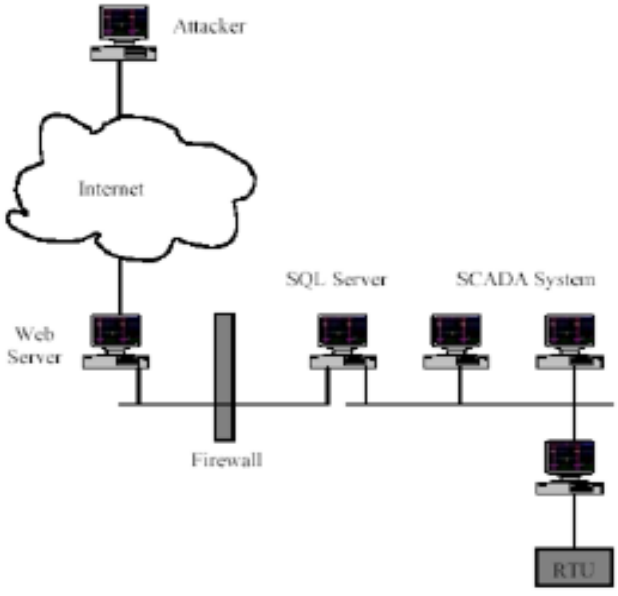


Figure 2-3: SQL Injection Intrusion of SCADA System [27]

The other issues that can arise with these industrial protocols include eavesdropping, which is where an intruder listens in on the data being transferred and remains undetected by the system. Another is the man-in-the-middle attack, which is similar to eavesdropping, but here the hacker can steal, transmit or modify any of the protocols, packets or data being transmitted. These protocols can also be used to trick the system by an intruder by copying a legitimate response and then replaying this response so that the system is fooled into thinking that the intruder is a legitimate user. The intruder can download malicious software to disrupt or even gain complete control of the SCADA system or delete data and files from the control and field networks [29]. There are also cases where the intruder might be an authorized user that has made a mistake, such as a bad command input. A disgruntled employee might also plan an inside attack by disruption of the system, stealing sensitive information or planting malicious code that will disrupt the system at a later date. Of course these particular insider attacks are the hardest to detect since those with the greatest knowledge of the system will know how to circumvent all security and detection techniques [30]. Table 2-2 lists these and other threats for the both the hardware and software levels of the system.

Table 2-2: SCADA Security Threats

Hardware Threats	Software Threats
Damage equipment	DoS/Buffer Overflow
Tap power/communication lines	Gain Privilege/access/steal data
Drain sensors/RTU/PLC of power	Crack security keys & change set points
Gain control of equipment/sensors	Disrupt or reroute traffic
	Desynchronize data streams

Many of the hardware threats shown in the previous table have little defense when it comes to physical damage. At best, the process equipment can be fenced, guarded or locked to prevent damage. Most of the software threats can be combated with security that is up to date. Designing a system that is robust to these threats and detection of these threats is a key research challenge. To conclude this section, a description of some reported successful attacks, inadvertent or otherwise, against SCADA systems are given.

2.1.2 SCADA Related Security Incidents

As mentioned earlier, these systems are used in a wide variety of industries which control vital processes such as power and water distribution. It makes sense that malicious intruders would want to compromise these types of systems to cause the greatest amount of damage. To support this statement, in a recent survey of power utilities around the world, over 80% reported that the

plant had experienced a DoS attack and 85% reported that some part of the communication layer or related components had been attacked [31]. In the United States, Homeland Security reported in 2013 that nearly half of the reported security incidents for CPS were from the energy sector [32]. This high percentage of reported intrusion attempts shows that systems in the energy sector are prime intrusion targets [33]. While many of these intrusions never reach the public due to the insignificance of the attack or were not reported, there are a few well known attacks that have proven to be quite effective.

The first security incident of intrusion against SCADA systems to be discussed is the well known Stuxnet attack that appeared in 2010 [34]. The attack used several previously unknown software exploits, PLC root kits that were designed to gain privilege to a PLC, and several malware evasion techniques. The end goal was to find PLCs and RTU systems that used the Siemens 7 communications protocol, which is used in about 30% of SCADA systems worldwide [35]. It was later discovered that the main target of attack was the control systems and centrifuges in Iran's enrichment facilities and that Stuxnet was designed to specifically disrupt these systems. The attack started as a worm on a flash drive that was ported to the facility, where it propagated through the system. Once the appropriate equipment was discovered, the worm disrupted the centrifuges. In Figure 2-4, the various infection routes that Stuxnet took to complete the intrusion are shown [36].

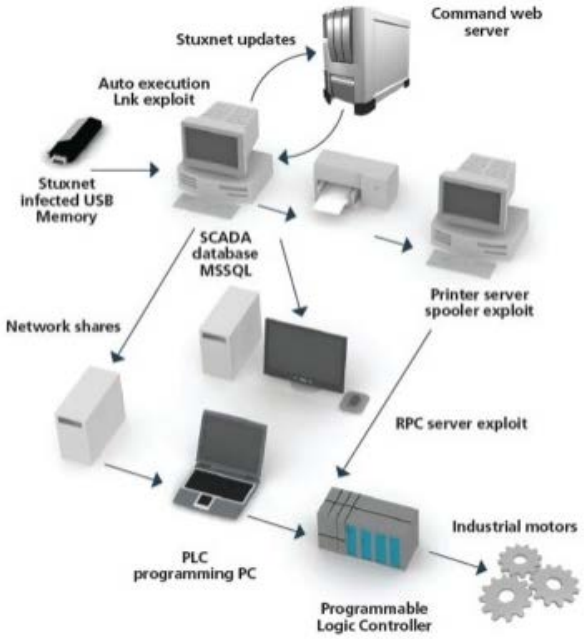


Figure 2-4: Stuxnet Attack Outline [36]

Another example in the nuclear power industry was the Slammer worm that infected the Davis-Bessie power plant [37]. The process monitoring systems of this plant employed SQL along with their SCADA systems. The worm was able to bypass the firewalls between the external and internal networks and infect the system because the software had not been updated for over six months. The attack caused several plant monitoring computers to go offline for several hours, though there was no permanent damage to the facility or infrastructure. Also, the plant was already in a controlled shutdown, so there was no danger to the plant or to the general public. However, if the plant had been operating at the time, there would have been no way to monitor system processes. This would then be an effective means to disrupt and damage a facility of this type.

In the oil and gas industry, a major disruption to a gasoline pumping system in Bellingham, Washington occurred in 1999 [38]. At the time, construction workers had installed water lines over the gas lines and had damaged the lines in the process. Also, the gas company had installed new valves in the system. These were poorly configured and caused a large rise in system pressure soon afterwards. While the MTU polled the RTUs regularly, the information was not updated to the HMI for several minutes. During the accident, an operator had also been updating the live SCADA system with untested programs. This action caused the system to be unresponsive to operator commands and they were unable to alleviate the pressure buildup. A rupture in the gas lines caused a large explosion that had severe environmental impacts, but most importantly caused three deaths. This incident displays the vulnerability of cyber-physical systems from authorized users and poorly configured monitoring and actuation systems.

An additional intrusion incident in the gas and oil industry occurred in San Bruno, California in 2010 [39]. This incident involved an explosion of a natural gas pipeline. The cause of the accident involved a SCADA system that was receiving incorrect data, along with pressure sensors that were not reporting back to the control center. The initiating event was a power failure that caused an imbalance in the pipeline pressure; attempts to mitigate the pressure build based on incorrect data only served to cause an explosion. The final result was environmental damage, over 60 injuries and eight deaths. This incident shows how issues in system availability are a serious vulnerability.

In the water sector, an attack against the SCADA system occurred in Australia, today it is known as the Maroochy attack [40]. The city had recently installed a process system to manage the nearly 900 kilometers of sewer lines and 142 pumping stations. After the system was installed, operators noticed that the system would lag, not respond to commands, and communications would be periodically lost. The culprit was one of the contractors that had installed the system and was resentful after being refused a position on the city council. Using

wireless connections and driving from pumping station to station, the intruder was able to disrupt the system enough that over 800,000 liters of sewage were pumped in parks and residential areas. This type of attack shows how these systems can be easily disrupted by internal threats.

Another disruption of a control system in the water sector happened in California in 2007 [41]. The incident involved a water canal system located on the Sacramento River. It was discovered that an electrical engineer had installed unauthorized software on the system. This had the effect of diverting a large amount of water from the canal to other areas and caused a large loss in revenue for residential customers and threatened the local agriculture industry. The engineer had installed the software on the same day that he was fired after a 17 year career. For this disruption of the system he received 10 years in prison. This shows that SCADA systems are just as vulnerable to insider attacks as they are from outside intrusions.

When considering the electric power grid and related equipment, a major vulnerability was found in a brand of diesel generators called Aurora, reported in a US accountability report [42].This intrusion was called the Aurora Generator Test. These generators cost approximately $1 million and are one of the major brands used in the US and in the power grid. The intrusion was performed remotely against a generator at Idaho National Laboratory (INL). The end effect was the generator being destroyed quite easily. This type of attack caused a great deal of concern because it showed the vulnerability of the US power grid and related equipment to a simple intrusion

2.2 Intrusion Detection Systems and Methods

Given that SCADA systems exhibit many vulnerabilities and threats that can be exploited and lead to significant damage, methods of securing these systems against intruders need to be addressed. One of the main reasons for this given earlier is that SCADA systems control vital processes and disruption can in some cases lead to disastrous results. Intrusion detection can be termed any process that monitors current events in the system telemetry to determine if any use or security policies have been violated [43]. Any combination of software or hardware that actively monitors the communications, process signals, or other computer processes for unwanted activity is termed an Intrusion Detection System (IDS). The main functions that the system must have are real time availability, means to collect data from different sources, low compute cost, alert operators to anomalous activity and if implemented, take reactive measures to evict the intruder from the system before losses can occur [44]. The IDS can be broken into several layers that perform these tasks, an example of this is shown next in Figure 2-5 [45].

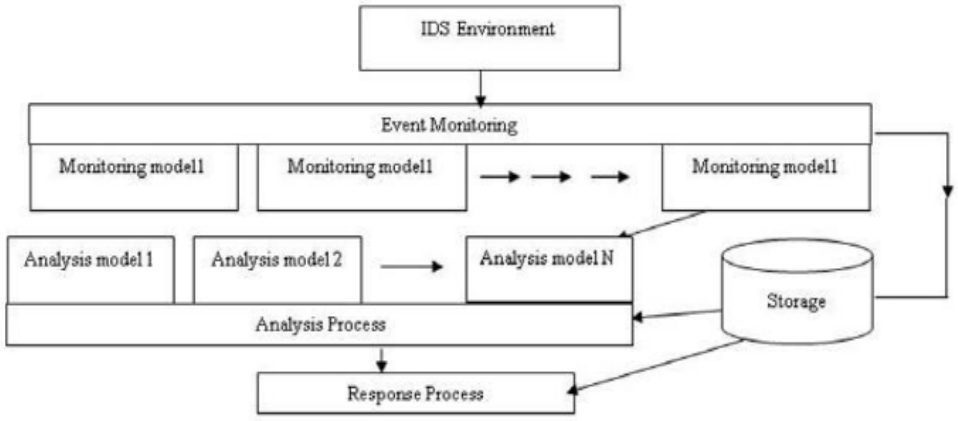


Figure 2-5: Generalized IDS Architecture [45]

The detection processes shown in the previous figure consist of monitoring current processes or operations, analysis of events, response actions to raised alarms, and data storage. The monitoring event layer consists of the IDS platform itself, sensors and data from monitored processes; such as network traffic or pressure readings; and one to several monitoring models. The analysis layer contains one to several models or methods that determine if the monitored event is actually an intrusion or if the output from the monitoring model is normal behavior. This layer can then generate alarms to alert operators. The response layer is a combination of either programmed actions or operator responses to stop any intrusion activities that are detected to evict the attacker. The last layer is for data storage. This stores the process data and output from the monitoring models. This data can be used for further examination offline. In the next section, some of the distinguishing characteristics of various IDS that have been developed will be discussed. The differences in these detection systems are due to deployment, the different data sources that are monitored, the type of monitoring model used, and what anomaly detection technique is employed.

2.2.1 IDS Classification – Audit Data Sources

The first classification of IDS will be based on the audit data that is collected for intrusion detection. Audit data can be extracted from several sources. These sources include the OS, running applications, network traffic, software/hardware measurements or system logs. The two main forms discussed in literature that will be examined are termed Host Intrusion Detection System (HIDS) and Network Intrusion Detection System (NIDS). Each of these distinctions naturally arises from the different audit data sources collected and configurations of the systems that are being monitored for intruders.

The first to be discussed are HIDS, in which a single host such as a computer or server is monitored to determine if there are any events that are show specific signatures of misuse [46, 47]. Even though several hosts may be connected in a network, if host monitoring is employed then only single hosts on the network are monitored and not the network as a whole. Each host would need its own IDS installed to monitor the various audit logs. The audit data used is primarily collected through OS, application, or keystroke logs. OS specific audit data are logs of events in the system. These consist of the event, what user was related to the event, and any commands or programs that were used during the event. Application audit data generates logs on specific applications and how they were used. Keystroke audit data logs every key that was pressed during operations; this can be used to determine how an intrusion occurred and how the system responded. The main idea behind this method is that the intruder will leave some sort of fingerprint in the audit logs. This is because well defined rules for the host can be developed. Anomalous activity would then be different from these specified rules [48]. In Figure 2-6, a generalized HIDS is shown [49].

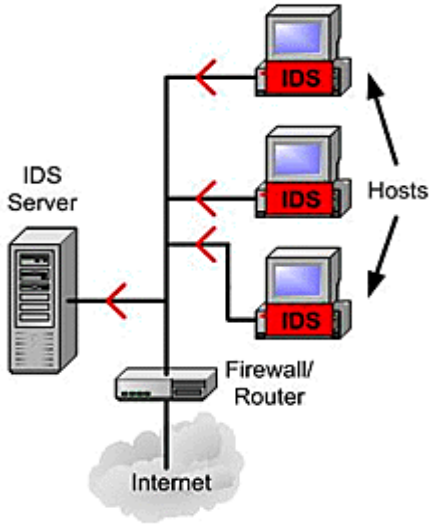


Figure 2-6: Host Based Intrusion Detection System [49]

Some of the disadvantages that can arise when employing HIDS are that each of the hosts must use their own resources to monitor and analyze all of the relevant audit data. Next, the system can be limited by the specific OS that is used by the host. Many vendors and companies would be unwilling to purchase IDS software that is specific to an OS they don't currently use. Also, the software may be application specific. Some applications or log generating capabilities may not be installed or available across all vendors and companies. Finally, a clever intruder or well designed malware can mask the content of the audit files to masquerade their activities.

In contrast, NIDS examines the signals in the communication layer and between different hosts on a network [50]. The audit data can contain TCP/IP traffic, memory or CPU usage, and system process measurements. Other data or monitoring specific to NIDS are the various packet inspection methods. These examine incoming packet contents to determine intrusion activities. They are also used more extensively in SCADA. Because these are already large networks of interconnected systems, it makes sense to monitor the packets across this system for signs of attacks [51]. It is noted that most of these detection systems are passive, in that once an alert is given there is no other action taken besides sending an alert to an operator [52]. Also, host based methods have been found to be suited to internal rather than external threats, since the system is looking at any unauthorized actions on one host [53]. Network based methods are more suited for use in detecting external threats since they monitor the outside communication lines that connect several hosts together.

One of the main advantages in using network detection is that resources are better utilized than in the previous type of system. This method can monitor several hosts at once and the monitored hosts do not have their resources constrained by having to analyze their own audit files. They are also more cost beneficial in this respect in that only one network monitor needs to be installed to protect the network. In HIDS, every component that needs intrusion detection must have this system installed [54]. Another advantage is that it is difficult to gain access to host OS and applications through the network, but is easier if the host is attacked. One disadvantage is that nodes collecting audit data can be viewed by intruders. Also, it is difficult at times to obtain a full view of all network activity. It has been found that different audit trails from several different hosts on a network can be correlated to gain a better picture of the network activity, but this must be implemented in the infrastructure to work properly [55]. Other issues that can arise with these systems are that networks on a switch are more difficult to monitor since the network is subdivided on a dedicated port. Also, most switches do not allow for global monitoring. Next, NIDS usually cannot analyze information that is encrypted unless given the proper encryption keys. This is not usually done because this creates an additional vulnerability in the system and can serve to increase resources used on resource constrained systems. Finally, a large amount of packets can be dropped during observation. This means an intrusion event or signature might be missed in the dropped packets.

2.2.2 IDS Classification – Detection Techniques

The other main classification and function of the IDS that needs to be discussed is the technique that is used to detect intruder attacks. In the open literature, there are three primary detection means that are used by host or network based systems. These are termed Knowledge Based, Behavior-Based, and Behavior-Specification-Based. Before each of these techniques is discussed,

some performance metrics related to IDS will be defined. Typically, performance metrics of these systems would be measured in terms of resource usage, installation cost, and speed. In the past decade, more focus has been placed on performance of the IDS in terms of detection accuracy and the number of missed or false alarms during observation [56]. There are four outcomes that are possible when observing events with the IDS. These can be classified into a confusion matrix for actual and predicted states, shown next in Table 2-3 [57].

Table 2-3: Confusion Matrix of Actual vs. Predicted Outcomes

Actual	**Predicted**	
	Normal	Intrusion
Normal	*True Negative (TN)*	*False Positive (FP)*
Intrusion	*False Negative (FN)*	*True Positive (TP)*

In the previous table, a TN and TP indicate that the detection system correctly identified the current system state; respectively these are normal or attack states. The reverse of these is the FN and FP. These measures indicate that the system incorrectly identified normal behavior as faulted and vice versa. As the outcomes shown in the table are for a single event, several equations that give a numerical value of alarm rates are used to further evaluate the performance of the IDS, shown next in Equations 1-4.

$$\text{True Negative Rate } TNR = \frac{TN}{TN + FP} \quad (1)$$

$$\text{True Positive Rate } TPR = \frac{TP}{TP + FN} \quad (2)$$

$$\text{False Negative Rate } FNR = \frac{FN}{TP + FN} \quad (3)$$

$$\text{False Positive Rate } FPR = \frac{FP}{FP + TN} \quad (4)$$

The FPR provides a measure of the proportion of normal observed data that is classified as an intrusion. If this measure is too high, the system will face down time for needless inspection and users will lose confidence in the IDS. On the other hand, a high FNR will introduce vulnerability into the system because many actual intrusions will be labeled as normal. Intruder actions will then be missed. An effective detection system must have low or zero FNR/FPR measures and also maintain high TPR/TNR functionality.

The first detection technique is termed Knowledge-Based detection. This method examines the audit material for specific signatures or features of an intrusion contained in an attack dictionary. If the specific signature matches one that is contained in the attack dictionary, then an intrusion is said to be detected [58]. This type of detection is also called misuse, pattern-based or supervised detection. The signatures used in the attack dictionary can be known patterns of an attack, system log deviations or other sensor measurements that display a particular signature. Since these systems only respond when a pattern in the audit data matches a specific pattern of misuse, they have a low FPR [59]. The main drawback for this detection technique is that the system will only respond to a known signature. If an intrusion attack is underway that has an unknown signature, then it will be ignored and treated as normal behavior [60]. This has the effect of increasing the FNR, which means that too much trust can be placed in a system that employs this type of detection technique. The other drawback is that a library of signatures will need to be developed and continuously updated since the intruder can come up with a new attack pattern that does not match known signatures. Finally, the key challenge to using this technique is defining an effective attack signature library and keeping it up to date [61].

In contrast, Behavior-Based detection techniques look at any behavior in monitored audit data that is anomalous to normal behavior [62]. In this technique, normal behavior consists of data from the system that is operating without any intrusions present. This is usually termed training data. Using the training data, an empirical model is developed for the system that is used to generate predictions of current observations. Some machine learning approaches that have been employed in this type of IDS are NN, GA, clustering methods or Bayesian classifiers [63, 7, and 10]. The training data used can be either multivariate system data such as memory usage, packet information, and hardware measurements. Statistical techniques can also be included in this detection scheme. These use statistical features of signals such as Root Mean Square (RMS) or if an actuator sensor value is a set number of standard deviations away from normal behavior. The main advantage in using this type of detection technique is that there is no signature library that needs to be developed or updated. Any behavior that deviates from what has been defined as normal is flagged as anomalous. This means that zero-day attacks can be detected, which is impossible with the previous detection method. A drawback to this method is the high FPR seen in many developed model predictions. This issue may be alleviated with an optimized model and will also be addressed later in this dissertation with the implementation of the SPRT. Another possible disadvantage with this method is that the system may be vulnerable to attackers during retraining stages if the IDS had been taken offline [64]. However, given that most critical systems have added layers of security nowadays, this would only be an issue if all layers of security failed.

The last intrusion detection technique is termed Behavior-Specification-Based. This technique is similar to knowledge and behavior based methods in that a specific behavior of the signal or system behavior is defined beforehand and the system only looks for these specific behaviors [8]. In contrast to knowledge-based methods, the specific behavior is defined, formalized, and programmed by human operators. Also, known signatures are not used since these are susceptible to future attacks. Some advantages to using this type of method are that there is a low FPR since only the specified behaviors or rules are examined for intrusion. However, this can again lead to a high FNR since the system cannot distinguish some unknown attacks. Another advantage of this method is that there is no training or user profiling stage, the IDS is developed before the system is brought online. The main drawbacks in utilizing this type of detection technique is the determination and formalization of system behaviors and attack signatures, this can be quite difficult in practice. Also, since a human must define all specifications, the process can be time consuming and prone to a large amount of errors. To summarize the ideas presented in this section, a listing of the advantages and disadvantages of the three detection techniques and audit material sources that were discussed are provided next.

Intrusion Detection Technique & Audit Data Advantages:

- Knowledge-based IDS have low FPR
- Behavior-based IDS have the ability to detect unknown or zero-day intrusions
- Behavior-Specification-based IDS have the ability to detect unknown intrusions and have a low FPR
- Using network based audit data helps reduce loading of systems or nodes that have resource constraints
- Using host based audit data allows for distribution of control and can ease detection of host-based intrusions

Intrusion Detection Technique & Audit Data Disadvantages:

- Knowledge-based IDS will not detect any unknown intrusions and attack signature dictionary must be developed and continually updated
- Behavior-based IDS have traditionally suffered from unreasonably high FPR, a challenge that will be addressed later in this thesis.
- Behavior-Specification-based IDS must be implemented by human operators and formal specification development is costly and error prone
- Using network based audit data can leave nodes or networks visible to intruders
- Using host based audit data can lead to high resource utilization of hosts, visibility of host, IDS and data to intruders and are difficult to generalize to other systems

2.2.3 IDS Developed for SCADA

This section will provide a review of major detection systems that have been specifically designed or reported for SCADA system security. Initially, security concerns for SCADA systems were relatively nonexistent because there were few who had the capabilities or drive to attack these types of systems. Also, the standard practice of plant network isolation made intrusion into these systems somewhat difficult. It has only been in the past decade that serious attacks like Stuxnet, combined with the proliferation of technology in all aspects of daily life, has there been interest in developing detection software for these systems. For this review, nine SCADA specific IDS found in the open literature will be discussed. The review will focus on what audit material is used, what detection technique or method was applied, different intrusion scenarios considered and reported results.

In [65], the researchers developed a Java based HIDS called Middleware-based Intrusion Detection for Embedded System (MIDES), this was used in conjunction with MicroQoSCORBA middleware framework. Middleware is typically installed in the low levels of a SCADA system; this is software that bridges applications to databases and OS on a network. The MIDES system was not actually used for intrusion attacks or detection in the results, rather several different applications and sensor configurations were employed to test the system usage. This was performed on two PCs running a Linux based OS and using Java Tiny Network Interface (TINI) boards. The system usage statistics included memory usage for increased number of applications, latencies in running programs and scalability. The main goal of this research was to evaluate the overhead cost of running this type of detection system in the middleware level of a SCADA system.

Oman and Phillips [66] developed a knowledge-based HIDS that considered power systems. The audit material used for the study consisted of typical host based data, such as login attempts, commands issued, password identification, frequency and installation of new firmware, and keystroke logging, to name a few. To simulate intrusions, the authors used ping flood attacks, change of settings and Snort to monitor if the system is online and if it is being sent legal commands. The authors also develop an intrusion system that in the current results only monitors the RTUs on the test bed; future work would extend the system to other devices on the network. There are no numerical results of system performance given and the intrusion detection model is not fully described. The main focus of this research was to develop a detection system that would examine host based audit material for signs of attack. Signatures used for intrusion detection purposes were simulated in a SCADA power system laboratory test bed at the University of Idaho.

The authors in [67] develop a network-based IDS based on ant colony clustering. This multi-agent system has several agents that monitor, perform learning of behavior and have either passive or reactive actions to evict intruders. This actual system is based on previous work that develops models based on mathematical behaviors of ants [68]. The clustering is further exemplified by deriving mathematical models of pheromone reactions. The authors use the well known KDD-Cup99 dataset to validate their model. They consider several attack vectors; including probing, DoS, U2R and R2L. The results provided are an average detection rate of 92% and 1.5% FPR. The main goal of this research was to develop a multi-agent detection system to reduce the high FPR seen in network-based detection systems.

Next, the researchers in [69] developed a behavior-specification-based detection system for SCADA network systems. The main communication protocol that was studied for the research was Modbus, which uses a master-slave technique to monitor and control process systems. The authors develop several models for normal behavior that include protocol-level models, expected communication pattern models and server availability models. The protocol model examines Modbus headers to determine if bad protocols have been used, the pattern model learns the communications patterns of servers and the sever availability model learns service patterns of a server to detect intrusions. Audit data consisted of process sensor data. Some of the signatures used were unauthorized read and write attempts to the Modbus server and unsupported function codes. The authors do not present any numerical results or any performance metric results. The focus of this research was to detect intrusions in the communication layer.

In [70], the authors developed a network-based detection system that checks if incoming data content has been altered. Several modules are placed in between two SCADA devices; the data from one device is sent to a transmitter that encrypts the data. After encryption, the information is then sent across potentially unsecure lines that could be targets of internal or external attackers. The encrypted data is then sent to a receiver that decrypts the data and sends it to the required target SCADA device. The modules consider Type-I and Type-II protocols, the first compares octets in each packets checksum value for deviations, while the second compares packet header information and length for attacks. The research solution consists of appending specific encrypted information to each packet, if the received packet does not contain this information it is rejected. The authors do not present any numerical results of the experiment and provide no performance metric results. The main goal of this research was to provide a lightweight and secure intrusion prevention method. This method is known as a Bump-In-The-Wire (BITW) security solution since the modules are placed between devices; the schematic of this type of security is shown next in Figure 2-8.

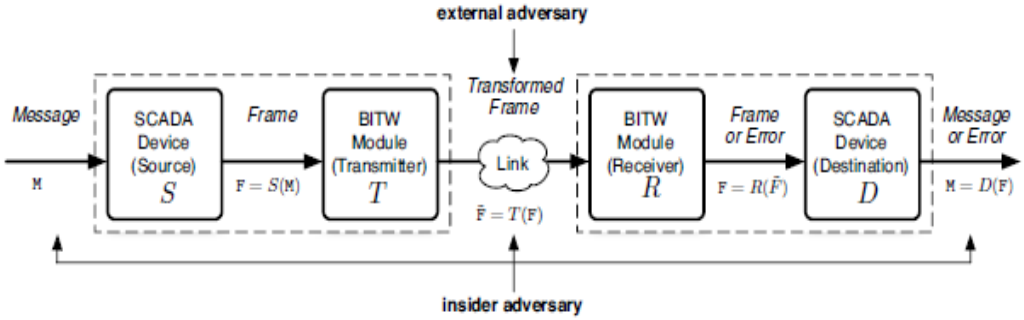


Figure 2-7: Bump-In-The-Wire Security Approach [70]

In [71], the authors investigate a behavior-based detection system for SCADA that uses network audit material. The audit data consists of TCP/IP traffic from a SCADA system and HTTP traffic taken from the perimeter network of an institution. The intrusion tools employed were Metasploit, Nessus, securityfocus.com and remote-exploit.org. Other exploits considered were taken from the US Common Vulnerabilities and Events (CVE) database [72]. For the HTTP data, overflow and application based exploits were considered, the results were an 88% detection rate and .2% FPR. For the TCP/IP data, overflow attacks were also considered; the results were a 92% detection rate and .2% FPR. The authors also supplied performance metric evaluation of their detection system, showing that with deeper levels of packet inspection there is a marked decrease in performance. The main goal of this research was to develop behavior-based IDS that can detect zero-day attacks with a low false alarm rate.

Last, the authors in [73] developed a Java host-based IDS termed SCADA-Hawk. The audit material that is used comprises non-functional events, such as logins, file access, firmware/software updates and other I/O records. The idea behind SCADA-Hawk is to place security perimeters around sensitive communication lines or equipment, the isolated equipment is then monitored for behavior that is abnormal. The authors also develop taxonomy of common events that relate to events in the SCADA system. The actual detection of intrusions is a misuse detection method; that is any preprogrammed signatures that match any in the audit data will raise an alarm. There are minimal results shown and no evaluation of actual performance of the system. The main goal of was to develop a detection system based on isolation of sensitive equipment.

While there are many other simple and complex IDS that have been developed for SCADA systems, the best approach for nuclear and other critical infrastructures is plant network isolation. Though network isolation won't stop insider threats, the vast majority of outside threats can be mitigated with this particular strategy. For other types of SCADA systems, a dedicated detection system is required to stop outsider threats.

2.2.4 IDS False/Missed Alarm Reduction Techniques

As stated earlier in the IDS detection technique section, one of the main drawbacks in using a behavior-based detection system is that there is often an unreasonably high level of false and missed alarms. This is a problem because if a system has a high FPR, then operators won't trust the system and there will be resources lost to investigate all spurious alarms. If the system has a high FNR, then there will be many intrusions missed because the system will never detect them.

A network behavior-based IDS called POSEIDON, which is based on using network traffic and self organizing maps, is developed in [74]. The idea is to correlate incoming and outgoing data so that false alarm rates are reduced. The authors only use three features to characterize the audit material: host IP address, service port and payload length. They also used a modified Mahalanobis distance function for similarity measurements. Using the DARPA 1999 benchmark data, the developed system gave FPR of 0 – 6%, in contrast with the .07 – 11.4% FPR obtained when using Snort. The main drawback in implementing this method is that authors do not fully describe the process used to reduce FPR, but only report the detection results.

Next, probabilistic models are formulated in [75] that uses Snort and university network traffic. The detection system uses a set of definitions to arrive at what is considered a true alarm. These definitions are total alarms generated, total alarms that match or partially match signature library, total that exactly matched and alarms from spoofed IP address. Using these definitions, the detection system calculates a formula to reduce false alarms. The main drawbacks with this paper are that there are no results presented and only a basic description of definitions used.

Last, in [76] the authors develop a signature based detection system that also uses Snort and network data supplied from university servers. The main idea behind this research is that tuning of rules and suppression of alarms will reduce the FPR. Indeed, the authors claim an 89% reduction in FPR before and after tuning, but simply removing all alarms is not actually a reduction in FPR. The main drawback in applying this work is that the detection system is not behavior based, and the main FPR reduction method was simply suppressing repeated alarms.

There were other papers considered, such as [77 and 78], however both of these systems also used a tuning algorithm that effectively suppresses repeated attacks signatures after they are noticed by the detection system. Some issues are that if the attack is long in duration, the system might never realize the intrusion and never creates an updated rule.

2.3 Variable Grouping Methods

Traditional variable grouping methods are employed to arrive at an optimal subset of predictors with the overall goal of improving predictive performance. Variable grouping is also used to reduce data dimensionality, storage needs, model training and computation times [79]. Some popular techniques used include methods such as cross correlation criteria, forward/backward selection, wrappers, and embedded methods. Each of these listed techniques have their own strengths and weaknesses, which will be discussed briefly next.

2.3.1 Correlation Coefficient Criteria

The first variable selection method to be discussed is the use of the correlation coefficient. The correlation coefficient is a measure of the linear relationship between a variable X and Y. The standard Pearson correlation coefficient in MATLAB is used in this work, the equation for the correlation coefficient between variables X and Y is shown next in Equation 5:

$$R(i) = \frac{cov(Xi,Y)}{\sqrt{var(Xi)var(Y)}} \qquad (5)$$

In the previous equation, R is the cross correlation coefficient for variable i, *cov* is the covariance between X and Y, and the denominator is the standard deviations of X and Y. The main strength of using this method for variable grouping is that a metric or score can be obtained for variables from different sources. That is, the same data set may have variables that deal with velocity, power, or mass-flow rates. Thus, the correlation coefficient is a metric that is used to determine linear relationships among different data sources. The main weakness of this method is that the coefficient can only provide a unitless quantity about linear dependence between X and Y and nothing else about the physical dynamics of the data set. This means that it is left to the user to decide if the coefficients for a particular data set are useful for further modeling.

2.3.2 Backward/Forward Selection

Forward and Backward variable selection processes are basically the same idea, just reverse implementations [80]. In forward selection, p-values are typically used to score incoming variables for model development. The basic steps for this method are to score all variables and then add them to the model one at a time until no new variables can be added. In backwards selection, all variables are used to build a model and are progressively eliminated by a scoring function until the subset that returns the least model error is found. The main

strengths of these two methods are that they are very easy to implement, require no special software and with today's high performance computer technology, are computationally efficient. One of the main drawbacks for these two methods is that it is possible to exclude relevant variables during the selection process. Also, these methods are criticized for being computationally intensive and that the full relationships between all variables are not explored since one variable is continuously being removed or added to the initial model.

2.3.3 Wrapper Methods

In much the same vein as forward/backward selection, the wrapper method instead attempts to find a subset of predictors from a particular data set to be used in a specific Machine Learning (ML) algorithm [81]. Here, the ML method is used to access and arrive at suitable subsets of variables based on the performance of the ML model. Subsets that offer the best performance in the particular ML model are then chosen as the best subset. For practical purposes, the user must define some way to search all the possible subsets for a given data set and how to stop the search once a suitable subset is found. If there are several hundred or thousand variables to consider, this type of procedure can be very computationally intensive. The issue of removing and adding variables to a subset to reduce overall model error is criticized as not taking into account the full relationships in the input data.

2.3.4 Embedded Methods

Embedded methods perform the variable selection process as part of the model training phase; hence the variable selection is embedded in the ML model development. This method arrives at subsets of variables by attempting to minimize the cost of an objective function as variables are excluded or included during the selection process [82]. Examples of these objective functions include finite difference, quadratic approximations or taking derivatives of the objective function. Other parameters may be optimized, such as model error, instead of using an objective function. The main strengths of embedded methods are that they are often easy to implement and the process can occur during model training. The embedded method faces the same criticism as wrapper methods, in that the search algorithm can be computationally inefficient for larger data sets.

In conclusion, the previous techniques constitute only a small subset of available variable selection methods. One or more of these techniques are often combined and/or favored over more mathematically complex or computationally intensive search strategies. For more information on several other selection methods, see the paper described in [79] and supplied references.

2.4 Time Synchronous Averaging Techniques

In this last section, the issues of time-phased data streams, or those with a clock mismatch, are discussed. A difference in sample rates of data streams or those with mismatched timestamps that are used in SCADA or IDS is a major issue that can lead to severe consequences. This was shown earlier in the section on SCADA security issues, where a time delay of data polled by the MTU was not seen by the HMI for several minutes. Untimely control commands by operators caused a large gas explosion and several deaths. This type of issue arises in almost any system or process that uses circuitry, in that several components in a system can use different software and hardware to record and transmit telemetry sources [83].

When considering this issue for intrusion detection, the problem that can arise is that several different monitored data streams can have vastly different sampling rates. For example, hardware signals in a server can collected at one sample per minute, while software signals can be collected several times a minute. If models for IDS are developed with these phase shifted signals, then the detection results can be quite poor. To alleviate this problem, the simplest solution is to set all data collection at the same sampling frequency. However, this may not be possible in practice due to resource constrained systems or if there are different software and hardware telemetry collection tools.

One method developed matches specific strings in both data streams and only considers those observations for intrusion detection [84]. The model works by attaching specific headers to normal data sources and to incoming data sources. Those that have matching strings are then considered for additional analysis or intrusion detection purposes. The main strength of this method is that only telemetry with specific properties is considered for intrusion detection purposes. The main weakness of this method is that it is possible for much of the telemetry to be lost if it does not contain the specific strings. This means that if intrusion signatures are in the telemetry not considered, then attackers can gain access to the system.

Another method developed is known as an Analytical Resampling Process (ARP), which has been developed by Oracle [85]. The resampling process involves the use of several variants of regression functions that generate data streams that are evenly sampled from data streams that have different sampling rates. This is a benefit because all telemetry sources being monitored arrive in a timely manner. This allows operators to make effective decisions about monitored processes. The main strength of this method is that telemetry can be resampled from sources that have vastly different sampling rates. A possible disadvantage of this method is an attacker has substituted incoming telemetry with false telemetry that would be resampled as well.

3 METHODOLOGY

Now that SCADA and intrusion detection systems have been introduced, the focus of this chapter will be on the different methodologies used for this research. Since the process of intrusion detection as a whole can seem complicated and confusing for the layperson, Section 3.1 will first provide an overview of this research and why specific methods were employed. After this, the following sections will cover contributions related to empirical modeling and intrusion detection. The first of these topics is the development of a time synchronization method that is used to resample time-phased data streams. Next, a complete description of the newly developed variable grouping algorithm is provided. This is followed by a description of the empirical-based modeling techniques used for this research. The chapter concludes with a full description of the Sequential Probability Ratio Test (SPRT), which was used for intrusion detection purposes.

3.1 Project Description & Related Methodologies

This section will provide a basic overview of the project and introduce various methodologies that were used to fulfill the overall research goals. As stated earlier, the main goal of this research project was to develop behavior-based IDS that employed specific empirical modeling techniques for intrusion detection of a simulated SCADA system. The empirical modeling techniques were chosen because they, along with the SPRT for anomaly detection, have not been previously employed by industry for intrusion detection purposes. It was unknown at the start of the research project if any of the intrusions tested could be detected using the developed methods. Then the overall goal was to determine which types of exploits that were targeted against a simulated SCADA system could be detected using the developed behavior-based IDS.

The first step in this research was to develop a simulated SCADA test bed. This step is required because no industries that use SCADA systems will allow researchers access to their facilities to test the efficacy of a developed method during real or simulated attacks. The rational behind this is that industries need to maintain the integrity and security of their systems. Combined with this is the reluctance to share sanitized data sources for research, which again is a security concern. The concern is that researchers will be able to learn certain dynamics about the processes in the system. This knowledge could then be used to learn proprietary methods or to even bypass system security. The test bed allows the researchers to collect normal operational data that will be used for model development. Anomaly data is collected during intrusion testing activities and is used in the developed models to quantify intrusions. A full description of the test bed and the various penetration testing tools is provided in Chapter 4.

Once normal operational data is collected from the test bed, the development of the empirical based models can begin. First, the data collected from the test bed must be processed. The data processing step entails removing signals that are unsuitable for further modeling. Here, unsuitable means that the original input signals were constant or near-constant valued, white noise, stuck data, and so on. Because these signals offer little information about the original process, they are excluded from further modeling efforts. In this stage, time synchronization methods can also be employed to resample all data sources to the same sampling rate, if required. This ensures that all available telemetry can be utilized and will be discussed in more detail in the following section.

As discussed in the previous chapter, variable grouping is employed to arrive at a suitable set of well correlated predictors that return low model prediction errors. This is important because if there is a large amount of error in the predictions, this would indicate a poorly selected set of predictors or poorly optimized model. Also, for the data-driven models that are utilized for this research, subsets of well correlated predictors are required for these modeling techniques. Discussed in greater detail later is a newly developed variable grouping method that is used to select a suitable set of predictors.

After the data is processed and suitable sets of predictors have been extracted, the development of data driven models can begin. Figure 3-1 provides a schematic of the following described steps for monitoring purposes. Starting with the input data, an AAKR or AAMSET model is developed. In this step, models are initialized, optimized and validated. Once this step is complete, data can be run through the models to generate predictions. Subtracting the predictions from the input data results in what are known as residuals. Once this step is complete, anomaly data generated from intrusion testing activities can be input into the developed models. The anomaly residuals are then supplied to the fault detection algorithm, here it is the SPRT. The SPRT is used to arrive at a decision of the current input being in an unfaulted or faulted state. These states are returned as hypothesis values, where 1 is a faulted state and 0 is an unfaulted state. The final step is to input these hypothesis values to a diagnostic system. Here, this can be an algorithm or human that gives a final decision.

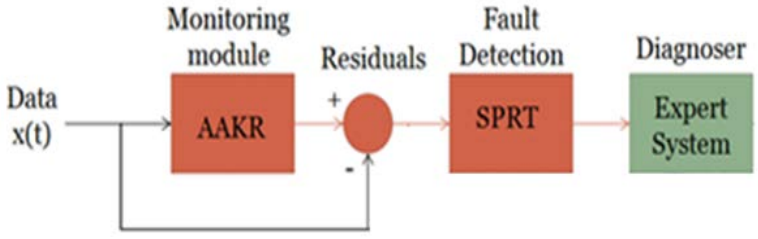


Figure 3-1: Data Driven Modeling Example

3.2 Java-based Time Synchronous Averaging Technique

This section describes the methodology used to develop the Java-based Time Synchronous Averaging Technique (JTSAT). As stated previously, JTSAT is based on a similar technique developed in [85]. It is noted that this method does not use the same resampling algorithms used by the method developed by Oracle. Also, the JTSAT algorithm in its current form is an off-line procedure. This algorithm was developed for this research because the hardware and software variables of the test bed servers have very different sampling rates and need resampling so both telemetry sources can be utilized for intrusion detection.

To begin, this method is used for two or more telemetry sources that have been collected from the same system. However, due to differences in circuitry, software or hardware components, the telemetry may have different sampling rates. It is known that processes in computers tend to speed up or slow down based on the number of applications running or during high memory usage periods. When this occurs, some of the clocks for some processes may start to shift. That is, one process may report a time stamp that is several seconds behind or ahead of the timestamp reported by a different process. If a detection system attempts to use this time shifted input telemetry, then the results may be unreliable if this time-phase becomes more pronounced. To use JTSAT, all that is required is unevenly sampled input data and the new desired sampling rate.

Using the asynchronous input data, JTSAT first examines each input file for overlapping time stamps. This is important because it doesn't make sense to resample data that is outside of the observed time ranges for each telemetry source. Once the overlapping time ranges are determined, the algorithm then generates new timestamps based on these ranges and the desired new sampling rate. For example, if a new sampling rate of one per minute is desired, then the new time stamps will start using the first overlapping range. Then, new timestamps which occur every once per minute are generated, ending with the timestamp corresponding to the last overlapping range.

The last step is to use these new timestamps and the resampling algorithms to generate the new, time synchronous telemetry. The Oracle algorithm uses several regression algorithms to arrive at the final resampled telemetry. In the JTSAT algorithm, linear interpolation is used to generate the newly sampled telemetry. The newly resampled data is then written to text files to be used for further data analysis or in this case, intrusion detection purposes. Because linear interpolation is used to generate the resampled data and not regression methods, there are of course slight differences between the Oracle method and JTSAT. In keeping with the outlined data processing methods described earlier, the next section introduces a newly developed variable grouping method for empirical based modeling.

3.3 ACFgroup Variable Grouping Description

In this section, the newly developed variable grouping algorithm that is based on the Auto Correlation Function (ACF) is discussed. This algorithm is hereafter termed ACFgroup. All equations shown in this section can be found in most standard textbook on signal processing and waveform analysis [86]. In simple terms, the ACF provides a measure of the correlation of a signal with a delayed copy of the same signal. This means that the ACF relates how similar or dissimilar the observations in a signal are with time-lagged versions of those same observations. The standard equation for the ACF is shown next:

$$Rx(t) = \lim (1/2To) * \text{Integral } x(t)\, x^*(t-t)dt \qquad (6)$$

In Equation 6, Rx is the ACF of x(t), tau is the number of time lags used to calculate the ACF, and To is one period. Also, x* represents the complex conjugate of the time-lagged signal x(t). The ACF can be thought of as a measure of change in x(t) with time. Because the ACF is a measure of change of the original input signal x(t), then the resulting shapes and various statistical properties of the ACF can be used for variable grouping. The basic idea is that variables that show similar changes in their respective ACFs must have responded in similar ways to the same system inputs. Then as a selection criterion, variables with similar system dynamic responses should be grouped together and variables that do not show these changes in system dynamic responses should be excluded. Finally, because the ACFs for variables with similar dynamics will also have similar trends and properties, variable grouping can then be based on these observed ACF properties.

3.3.1 ACF Shapes & Persistence

In this first section, the shape of the ACF for several different time series are examined and then shown how these shapes can be used for variable grouping. To facilitate this, a special property of the ACF known as persistence is also employed to develop several selection criteria. For this algorithm, the *autocorr* and k-means clustering functions contained in MATLAB are used. The ACFs generated by *autocorr* are normalized between -1 and 1 and are centered at 0. Also, the ACFs generated by this function can be considered as a sum of slowly decaying exponential functions. Due to the symmetric property of the ACFs, the developed method only considers ACF shapes for positive tau. Examples of specific time series that are considered in this work include trend, periodic, white noise, constant, near-constant, quantized/discrete, and combinations of these time series. Each of these time series and what the expected ACF shapes indicate about the changes in the original input signal are discussed next.

Trend Time Series:

The first of the time series to be considered is what is known as a trend time series. Though it is difficult to define explicitly what is meant by a trend time series, in this work trend time series are those that do not contain periodic components. Also, time series that are monotonically increasing or decreasing will not be considered as a trend time series for this work. Examples of trend type data can include temperature, flow rates or differential pressure measurements. It is noted that the previously mentioned list can in many processes contain periodic components. This is not a problem for this work as time series that contain trend and periodic components can easily be handled by the developed algorithm. In Figure 3-2, an example trend time series is shown along with the respective ACF for 100 time lags.

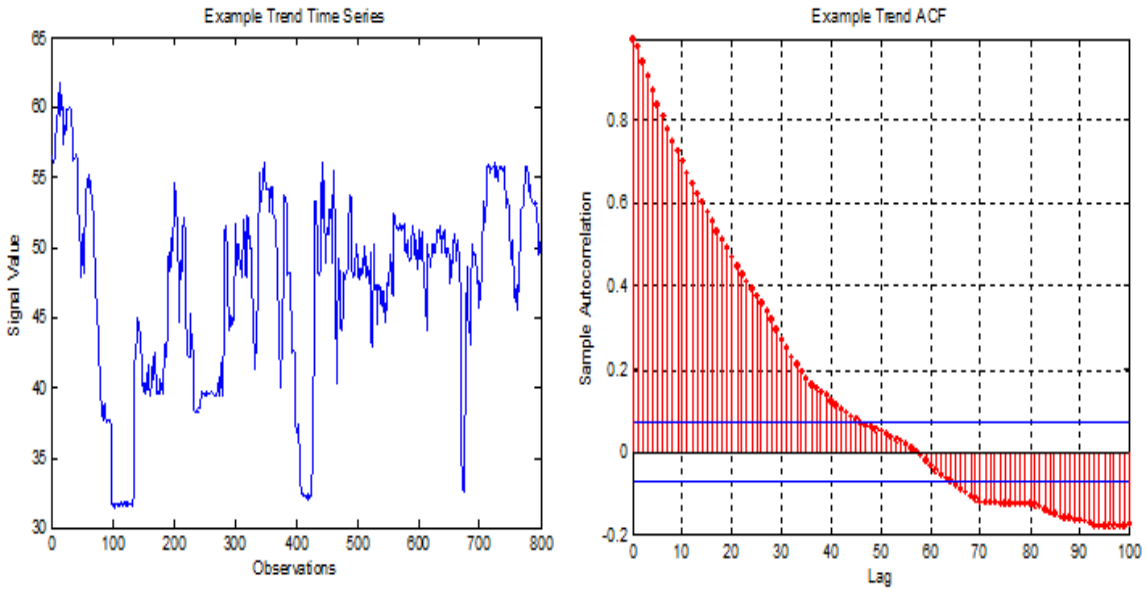


Figure 3-2: Example Trend Time Series & ACF

In the previous figure, the trend time series in the left plot is a pressure measurement taken from a nuclear power plant. The steady decline to negative values seen in the ACF in the right plot is important because this indicates that the input data has changed over time. This means that current observations are dissimilar to future observations in the time series. Also, because the time series considered was taken from a real system, then the changes seen in the time series indicate that the system responded to some form of input. For a given set of input data, trend time series that experienced similar changes over time or respond in kind to similar system inputs will also have similar ACFs.

Periodic Time Series:

In contrast to trend time series, periodic time series contain weak to strong sinusoidal components. As expected, ACFs generated from periodic time series are also periodic. More importantly, the period for these types of ACFs is the same as the input data. Also, because the information contained in a typical sinusoidal signal x(t) merely repeats with each new period, the time series for x(t) can then be restricted to one period. This also means that the information in a periodic ACF merely repeats with each new period. Then, because of this property, only one period is needed to generate ACFs for this type of time series. Shown next in Figure 3-3 is an example of a typical periodic time series with the respective ACF for 100 time lags.

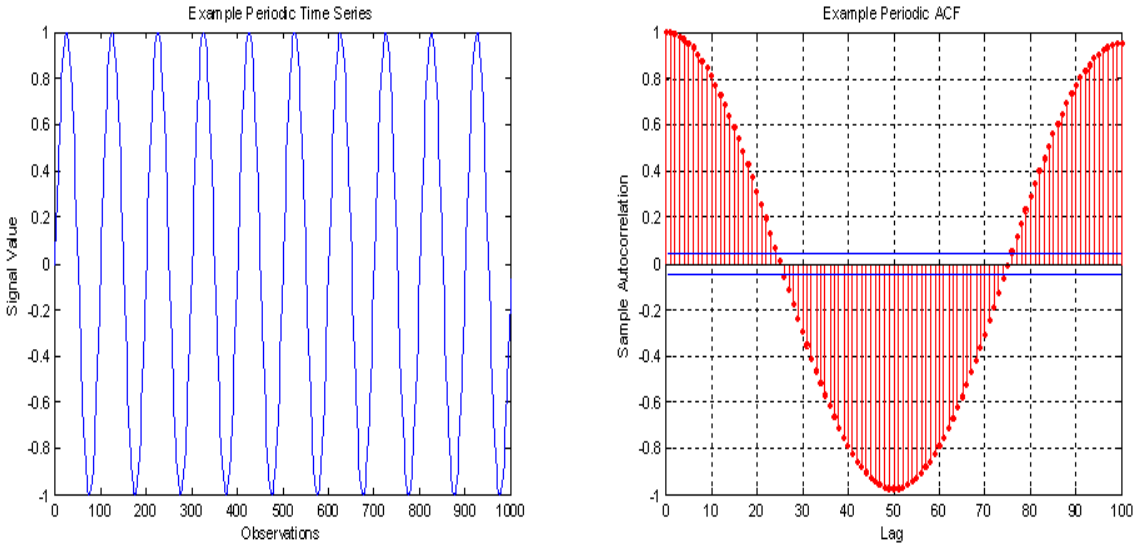


Figure 3-3: Example Periodic Time Series & ACF

The left plot in the previous figure shows a typical periodic time series, while the right plot is the respective ACF. It is noted that the time series shown in the left plot can be considered a "perfect" sinusoid; however, this makes no difference on the outcome of the algorithm. This means that even if the periodicity is not observed visually in the time series, the resulting ACF will be similar to that shown in the right plot. As stated earlier, because a periodic ACF has the same period as the original time series, then we are allowed to restrict the ACF to this one period. This result indicates that we do not need to use the entire time series for variable grouping. This also allows us to perform variable grouping with the developed methodology for data with large sampling rates because the period will be visible in the generated ACF. Periodic input data that show similar changes over time will also generate similar periodic ACFs.

White Noise Time Series:

White noise time series contain unrelated, random components. One of the original uses for the ACF was to detect what is known as serial correlation in data, such as periodic components. In this light, the ACF can also be used to detect serially uncorrelated data or white noise time series. For a pure white noise time series, the ACF is zero for all tau > 1. For time series that contain strong white noise components or white noise time series that have been multiplied by a scalar, the ACF will show very small oscillations around zero for all tau > 1. Shown next in Figure 3-4 is a typical white noise time series with the respective ACF for 100 lags.

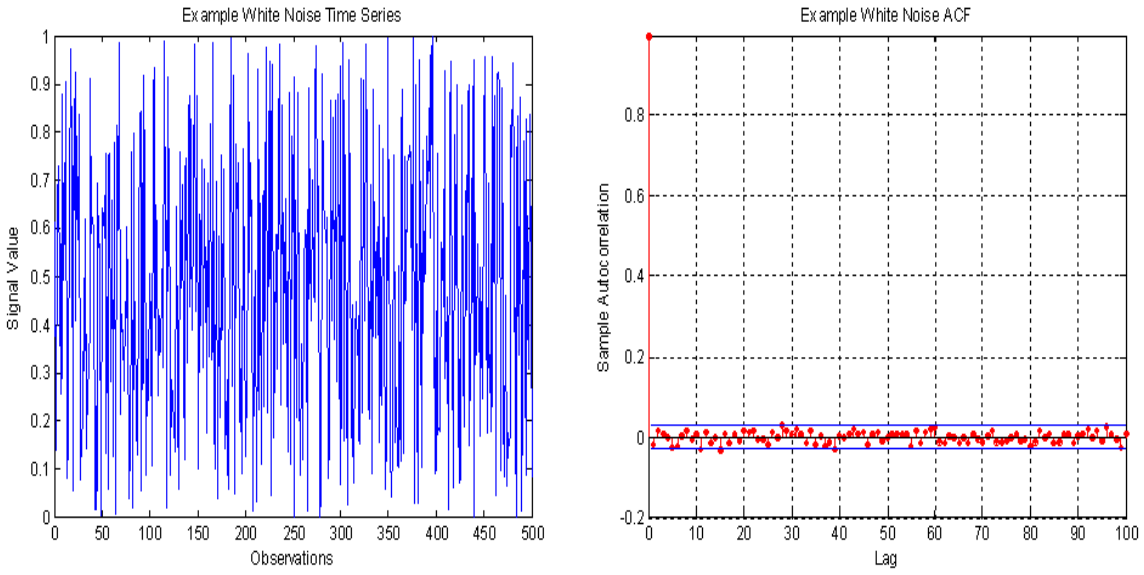


Figure 3-4: Example White Noise Time Series & ACF

In the previous figure, the left plot is a typical white noise time series generated in MATLAB. In the right plot is the respective ACF for this particular time series. The ACF takes on a maximum value at 0 and shows small oscillations around zero for all other time lags. Furthermore, the blue limits in the ACF plot indicate whether a particular time series contains strong white noise components. If the values of the ACF for tau > 1 are all between these two limits, then the series is considered as white noise. Conversely, if the ACF values for tau > 1 are outside these limits, then the series is most likely a trend, periodic or a combination of these. As seen in the ACF plot, all values for tau > 1 are between the limits, which prove that the input data was indeed a white noise time series. Because white noise time series have minimal changes with time over the range of the data, then this property can be used to reject these types of signals for further model development.

Constant Time Series:

As the name implies, constant time series never change value. For this type of time series, the ACF shows extremely slow exponential decay and is *always* positive for all tau. With this in mind, the properties of the ACFs for constant time series can used to reject these series during variable selection. The reason the ACFs for constants and other time series exhibit exponential behaviors is that all ACFs can be considered as a sum of slowly decaying exponential functions. This is also the standard output when using the *autocorr* function contained in MATLAB. Shown next in Figure 3-5 is an example of a constant valued time series and the respective ACF for 100 time lags.

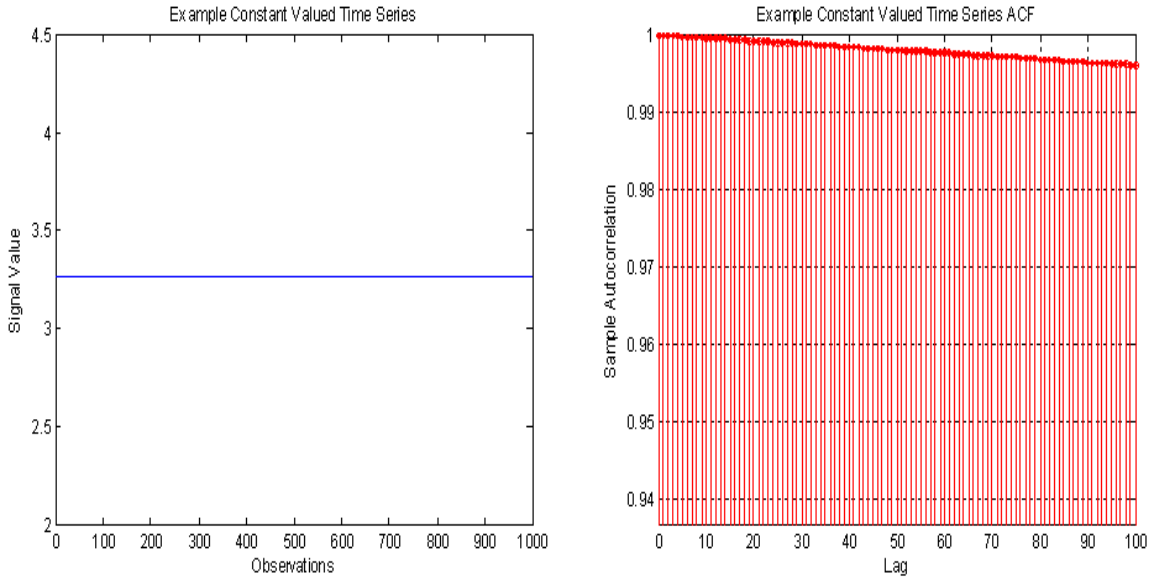


Figure 3-5: Example Constant Valued Time Series & ACF

In the previous figure, the left plot shows a constant valued signal, while the right plot shows the respective ACF. What is interesting about the generated ACF for this particular time series is the very slow change in ACF values for all tau. Because the input time series never changes value, the respective ACF will also have very slow changes. Examination of the ACF in the previous figure shows that for 100 time lags, all of the ACF values remained above .99. This indicates that there were very minor changes in the original data. Another property of the ACF generated from constant time series is that the ACF will always be positive for all tau. This means that this property and the very slow change in ACF values can also be used to remove these types of time series during variable grouping. The benefit is that these particular time series do not need to be removed before variable grouping, as in most other VGM.

Near-Constant Time Series:

For this work, near-constant time series indicate signal dynamics that show very few changes over the entire observation range. An example is a temperature sensor that recorded few temperature changes over the course of one day. Depending on the nature of this type of time series, the ACFs will either take on the behaviors of a constant valued or white noise time series. For example, if the input time series only has one change in signal values over the entire range of the data, then the ACF will take on the behavior of a constant valued time series. If the time series has very small oscillations around a constant value, then the resulting ACF will take on the behavior as that seen previously for white noise time series. Both of these cases indicate that the changes in the original time series do not change with time. Shown next in Figure 3-6 is an example of a time series that is near-constant with the respective ACF for 100 time lags.

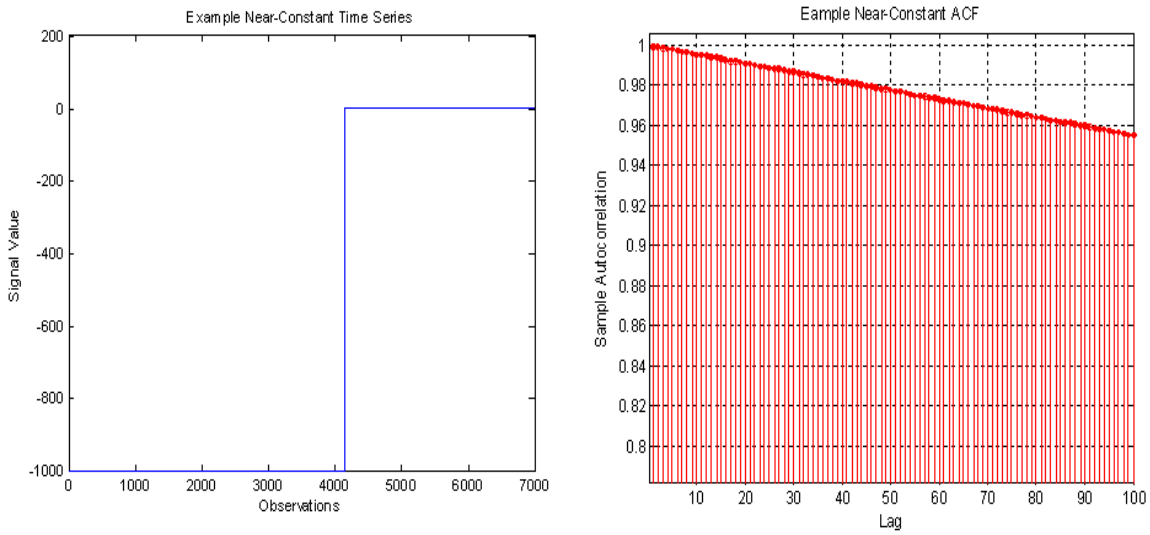


Figure 3-6: Example Near-Constant Time Series & ACF

In the previous figure, the left plot shows a time series that has a shift in value near observation 4000. In the right plot, the ACF behavior and values indicate that the input time series is not changing much with time. The behavior of the ACF for this particular time series is very similar to that seen for the constant valued time series in Figure 3-5. The difference in the near-constant time series ACF shows a slightly more rapid decrease in ACF values over all tau, though these values are still all positive. For near-constant time series that show small oscillations around a constant value, the ACF would take on the behavior seen in Figure 3-4. Due to the small changes in ACF values for this time series, they are removed by the algorithm as these signals are unsuitable for modeling.

Quantized/Discrete Time Series:

Quantized or discrete time series are data that take on, in general, a very specific range of values. For example, quantized voltage data might only take on values of 12.1, 12.2 and 12.3 V over the entire observation range. If the signal dynamics reflect this type of behavior, then the dynamics are said to be quantized. It is noted that quantized time series can contain components of trend, periodic, white noise and near-constant time series. Depending on the range of values for each input variable, the resulting ACF shapes can vary greatly. In general, quantized time series that take on 2 – 5 distinct values will generate ACFs typically seen for near-constant time series. If the input variable also has periodic components, then the resulting ACF will be similar to those seen for periodic time series. However, due to the quantized nature of this type of periodic data, the ACF also contains unique behavior that can be used for variable selection. A typical quantized variable that contains periodic and white noise components is shown next in Figure 3-7 with the ACF for 100 time lags.

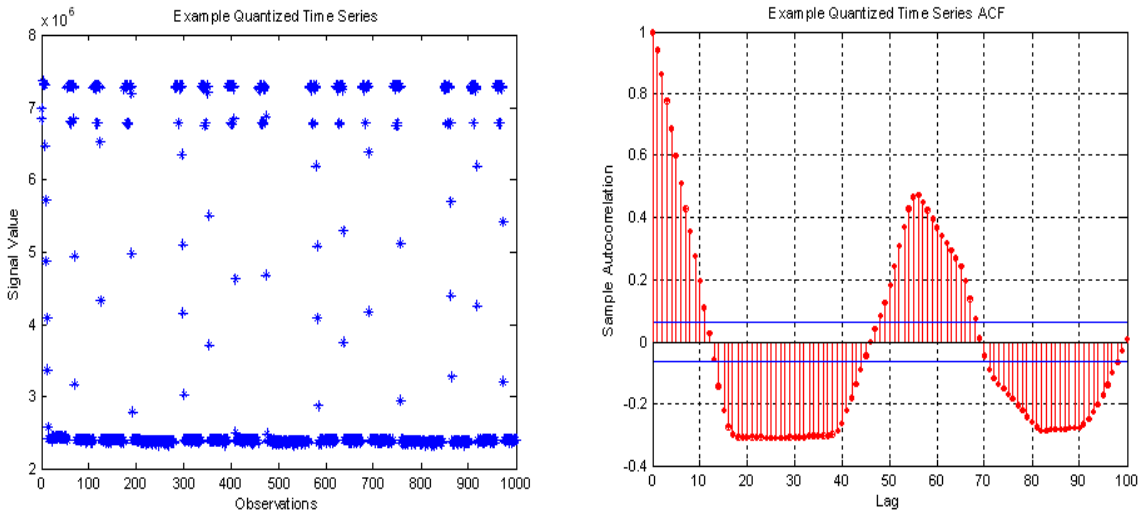


Figure 3-7: Example Quantized Time Series & ACF

In Figure 3-7, the left plot shows distinct quantized behavior at three different values. Also, there appears to be sufficient randomness in some of the variable dynamics that are between these three different quantized values. The ACF shown indicates that the input variable was indeed periodic, but some of the peaks have flat sections for several time lags. These flat sections in the ACF indicate that the input signal is stuck or not changing with time. This behavior can also indicate that the time series for this process will never exceed these quantized levels. Even though the majority of these signals will give small errors in model predictions, they were found to not be useful for intrusion detection purposes since many exploit dynamics are lost during data collection.

Persistence:

Now that the behaviors and expected ACF shapes for specific time series have been discussed, it remains to show how these ideas can be employed for variable grouping. A special property of the ACF that can bridge these two ideas is known as persistence. This property has been used to describe changes in geophysical processes, many of which do not show significant changes over long periods of time [88]. For this work, we adopt this same notion and define *persistence* as the physical property of a variable to remain at the same state or have minor state changes with time. Then because this method was developed using real data from a variety of real systems, the changes in state by the system in response to various inputs can be related as varying degrees of persistence. Finally, because the ACF shapes have been previously defined as responding in expected ways to certain time series, then the ACF shapes themselves are indicators of persistence.

This property can manifest in several different ways for observed ACF shapes. The first is that the ACF values are all positive or negative for all tau > 1. This was seen in the discussion of constant and near-constant time series, where the ACF values remained positive for all tau and had values close to 1. This indicates that the system state is not changing or changing very little with time. Next, the ACF can have combinations of positive and negative values over a large portion of the ACF. When this occurs, it indicates that the system has remained at the same state for long time periods, shifts to a different state and then remains at that new state for long periods of time. When considering white noise or certain near-constant time series, the ACF values can also exhibit small oscillations around 0 for all tau > 1. This indicates again that the system state has not changed greatly with time. In other words, the system dynamics persist at the same state. This type of behavior is also seen for certain quantized signals. Last, if the ACF values have a steady decline to negative values over small tau, then this indicates a small degree of persistence. In this case, the system has had meaningful state changes. This type of behavior was seen for trend type time series, which are useful for empirical modeling. If the ACF takes on a periodic shape, then this also indicates a small degree of persistence. The periodic behavior only arises because the system responds in this way to various inputs. In summary, trend and periodic time series show persistent behavior that indicates a meaningful change in the state of the system. In contrast, constant, near-constant, white noise, and quantized variables indicate undesired levels of persistence.

Based on the definition of persistence and the observed ACF behaviors, several selection criteria can now be defined. This means that the developed algorithm does not simply group variables together because several might have the same ACF shape. As will be discussed next, the developed algorithm is

actually a two stage filter that removes variables based on certain observed persistence values before final variable grouping. This means that the selection criteria for both filters are solely based on observed properties of the calculated ACFs for a given set of input data.

The first set of developed selection criteria is based on the time spent by the calculated ACFs for various levels of persistence. This persistence time is labeled Ψ and can be thought of as an indicator of time spent at the same state for each ACF. The calculation of this first selection parameter is shown next in Equation 7.

$$\Psi = (\Sigma ACF/tau) \qquad (7)$$

In Equation 7, the Psi parameter is simply the sum of all ACF values divided by the number of time lags tau. Equation 7 can be considered as the mean of the ACF that is normalized between 0 and 1 by tau. Variables with large values of Ψ are mainly either constant or certain quantized time series. Conversely, variables with small values of Ψ are mainly either white noise or near-constant time series. Each case indicates that the system state does not change greatly and that these signals should be excluded. Shown next Equation 8, lower and upper bounds are used to exclude variables with levels of unwanted persistence to arrive at a first set that is used for further selection procedures.

$$\Psi_F(m) = .01 < \Psi_I(I) < .7 \qquad (8)$$

In Equation 8, $\Psi_F(m)$ is a [1 x m] vector for the first set of selected variables. Here, m is the final number of variables selected from $\Psi_I(I)$, where I is the original number of variables. Variables placed in $\Psi_F(m)$ had Ψ values that were between the upper bound of .7 and lower bound of .01. The upper bound indicates that the signal spent more than 70% of the observed range at the same state. These signals are constants, near-constant or certain quantized variables. The lower bound indicates that the signal spent less than 1% of the observed range at the same state. These signals are mainly white noise or certain near-constant signals. Variables not selected for $\Psi_F(m)$ are permanently excluded.

In the second filter, the variables contained in $\Psi_F(m)$ are used as inputs for the next selection process. This final metric will be termed Ω and is the dispersion of persistence for each remaining ACF, shown next in Equation 9.

$$\Omega = \sigma^2(\Psi_F(m)) \qquad (9)$$

In Equation 9, Omega is simply the variance of each remaining ACF in $\Psi_F(m)$. It is noted that Ω is also the final metric that is used to develop the number of groups selected by the user. A critical value for Ω is selected, termed

Ω_C, which is used as a final rejection criterion. Shown next in Equation 10, only variables with Ω values above this critical value are selected for final variable grouping.

$$\Omega_F(n) = \Omega(m) > .01 \quad (10)$$

In Equation 10, $\Omega_F(n)$ is a [1 x n] vector, which are the final n remaining variables that had Ω values greater than Ω_C. $\Omega(m)$ is a [1 x m] vector, where m is the number of variables in $\Psi_F(m)$. The limit in Equation 10 indicates that the signal had less than 1% persistence over the observed range. The final set $\Omega_F(n)$ is used as the input to the k-means clustering algorithm. This remaining set can contain trend, periodic or combinations of these time series. What is important is that the values calculated for $\Omega_F(n)$ represent variables whose dynamics responded in kind to similar system inputs. This then satisfies the goal of grouping variables that show similar changes in persistence level.

Before the developed algorithm procedure is discussed in the next section, this large section will be summarized. First, the ACF was defined and shown that it was related as a measure of change in a signal with time. Next, various types of time series and the expected ACF shapes were shown. The persistence property was then defined and was also found to be related to the observed ACF shapes. Based on the observed ACF shapes and persistence relationships, several selection criteria were developed that first related the time spent by the ACF in relation to persistence. This first filter removes variables that were found to show small state changes or those that remained at the same state for long time periods. Next, a mathematical property of the ACF was used to quantify the observed persistence changes in each remaining variable. This final metric is used as a second rejection filter to remove any remaining signals that show low levels of persistence. The final group of remaining variables can then be used together or placed in smaller groups by use of the k-means clustering algorithm.

3.3.2 Developed Algorithm Procedure

The basic procedures used by the developed algorithm for variable selection purposes are discussed in this section. First, the required inputs and expected outputs are provided. Next, a generalized procedure for the developed algorithm is given. Also provided are flow diagrams that outline the procedure.

The first item to be discussed is exactly what inputs are required to use the developed method. For this method, the only three items required by the user are a time series data set, final number of desired groups, and whether the time series is average or fast sampled.

Time Series Input:

The time series data set can first contain any number of observations and variables. That being said, data sets with few observations and/or variables (e.g. 50 observations and 3 variables) should be used in a different variable selection technique. Data sets can contain variables that are a combination of any of the previously discussed time series or that are all characters. This means that the user is not required to preprocess the data set to remove potentially troublesome variables, such as characters or constant valued variables. The developed method is capable of identifying and removing these types of data. The developed method has been tested on data sets with observation and variable sizes ranging from 100 – 10.24×10^6 and 4 – 710, respectively.

Number of Final Groups:

The number of final groups is selected by the user. This option is used as the input to the k-means clustering algorithm, which is nested in ACFgroup. The k-means algorithm attempts to minimize the error between each of the given inputs and the center of each selected number of groups to determine which group each input belongs. Currently, the developed method can return 2 – 8 groups, if applicable. Based on the values obtained for $\Omega_F(n)$ discussed earlier, it was found during development that rarely are more than 6 groups required for data sets with even a large number of variables.

Fast Sample Rate Option:

The last required input for the user gives the option of performing variable selection on fast sampled data sets or not. For this work, "fast sampled" can mean data sampled in the KHz or MHz range. The only real limiting factor is the computer processing capabilities running the algorithm. An important point is that the user is not required to enter a sampling rate. The developed method is capable of arriving at expected results without this information. However, as more information is needed to develop the required ACFs, the fast sample option has a longer run time. The method has been tested on data sets with sampling rates ranging from 3×10^{-4} Hz – 102.4 KHz.

Expected Outputs:

The expected outputs returned to the user first include the final number of groups and all removed variables, in matrix format. Next, the method returns all variables contained in $\Omega_F(n)$ and their respective ACFs, in matrix format. Finally, the indices for all of these listed outputs, along with the number of time lags selected by the algorithm, are also returned.

ACFgroup Procedure Outline:

Using notation defined in previous sections, the generalized outline of the ACFgroup algorithm for time series data set is as follows:

- Begin Stage 1: Obtain user inputs – data, number of groups and Fast Sample (FS) option

- Is FS option set to NO?
  - Obtain data size N, N is number of observations
  - If $N < S_1$, where S_1 is size limit for small data sets, then lag τ for ACF = τ_1
  - If $N > S_1$, then proceed to Proprietary Algorithm 1 (PA1), then $\tau = \tau_2$

- Is FS option set to YES?
  - Obtain data size N
  - Proceed to PA2, then $\tau = \tau_3$
  - Note that this option increases computation time

- Based on value obtained for τ_1, τ_2 or τ_3, begin Stage 2
  - Calculate ACFs for each input variable *I*
  - Proceed to PA3 to obtain $\Psi_I(I)$

- Using $\Psi_I(I)$, proceed to PA4, this is first described filter.
  - Generate $\Psi_F(m)$ and first set of removed variables R1

- Using $\Psi_F(m)$, proceed to PA5, this is second described filter.
  - Generate Ω values for each variable of $\Psi_F(m)$ to obtain $\Omega(m)$

- Using $\Omega(m)$, proceed to PA6, this is still part of second described filter.
  - Generate $\Omega_F(n)$ and second set of removed variables R2

- Use $\Omega_F(n)$ as inputs to k-means clustering algorithm

- Return number of selected groups, removed variables, variables corresponding to those in $\Omega_F(n)$ and their respective ACFs. Return relevant indices for these data sets and selected τ

This concludes the generalized outline for the developed ACFgroup algorithm. To conclude this section on the ACFgroup, two flow diagrams are provided next in Figures 3-8 and 3-9. These diagrams provide the basic flow starting from the input data to the selection of final groups. In the figures, the acronym PA stands for Proprietary Algorithm.

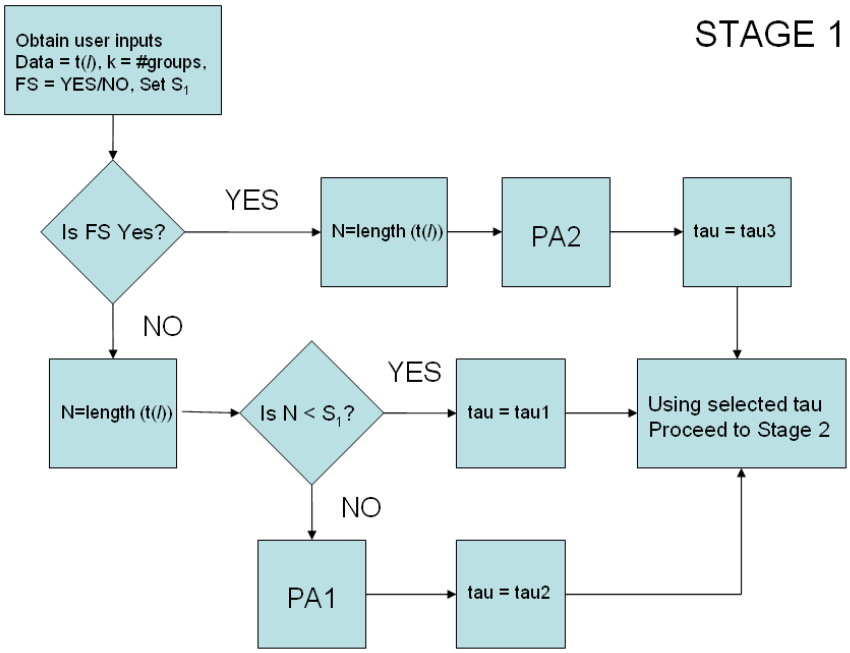


Figure 3-8: Stage 1 of ACFgroup Algorithm

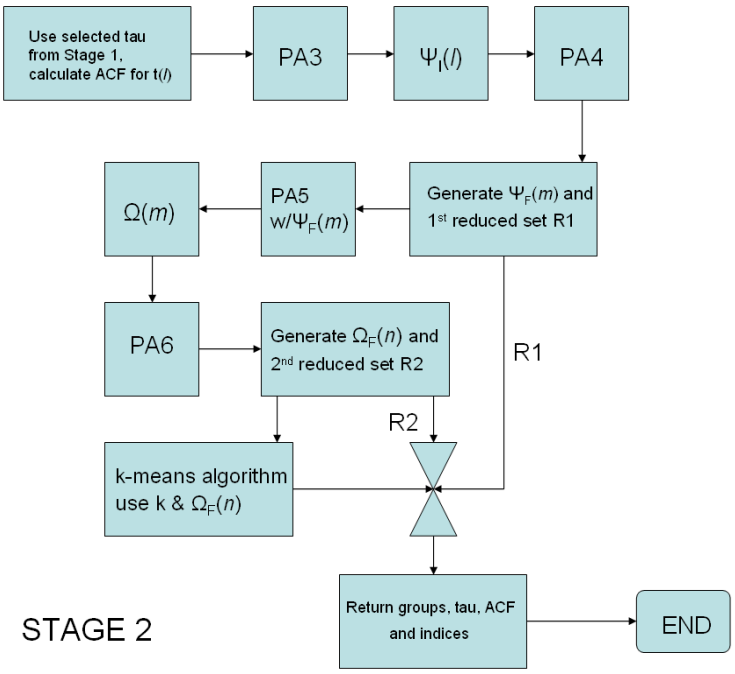


Figure 3-9: Stage 2 of ACFgroup Algorithm

3.4 Empirical-based Modeling

The core algorithms used for the data-driven models will now be discussed. Recall from Section 3.1 that these machine learning methods include the Auto Associative Kernel Regression (AAKR) and Auto Associative Multivariate State Estimation Technique (AAMSET). These and other machine learning methods can be broken down into several steps. First, data is collected from the system during normal operations. This data is assumed to be fault free and covers the entire range of operating conditions. After this data has been processed to remove unwanted time series, it is supplied to the grouping algorithms discussed in previous sections. After variable grouping, this data is used to develop and train the initial models. Next, the model parameters are tuned and optimized to ensure that there is minimal error in the model predictions. The models are tested for performance by running fault-free data that the model has not been exposed to before, this is termed the validation stage. Last, anomaly data or current telemetry is analyzed by the optimized model to generate predictions and anomaly detection results. For all time-series signatures under surveillance, the predictions are subtracted from the actual observations to generate residuals. These residuals can then be used in detection algorithms to arrive at a decision of normal versus anomalous behavior. The algorithms needed to generate model predictions and performance assessment will be discussed more fully in the next section.

The main idea behind the use of machine learning for behavior-based intrusion detection is that for some types of malicious exploits that escape detection by conventional knowledge-based IDS, the activities of an intruder may generate patterns in monitored time-series telemetry that are different from those that were modeled as normal behavior. The benefit in using a behavior-based approach is that new or zero-day attacks can be detected using these methods. Recall that the knowledge-based approach is unable to detect new or zero-day attacks. As will be discussed later, there are many intrusion activities that will not show up in time-series telemetry due to the rapid nature of the intrusion or small size of the payload injection. Many of these exploits will thus escape detection by the behavior-based approach because there may not be enough data to cause alarms. For these types of intrusions, a knowledge-based approach or some form of log analytics is required for detection.

3.4.1 Auto Associative Kernel Regression

The AAKR model is a form of nonlinear, non-parametric (NLNP) regression used to generate predictions of input data by computing the similarity of an input observation to known training exemplars [89 and 90]. The auto associative name convention means that predictor and response variables are

identical. This is opposed to “inferencing” wherein a subset of variables is used to predict the response of a different variable. All that is required to develop an auto-associative model is a set of normal, fault free data of the system, hereinafter termed training data. It is assumed that the training data accurately reflects the full range observed during normal system operations. This assumption is required because these model types cannot predict outside of the range of the data that was used for training. A subset of observations is selected from each signal of the training data set that is assumed to accurately reflect all possible behaviors observed in the set. This subset of observations will be termed the memory matrix $\mathbf{X}_m$, shown next in Equation 11:

$$\mathbf{X}_m = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \qquad (11)$$

In the previous equation, the columns represent observations for each signal considered, *n* is the number of memory observations selected and *p* is the number of signals considered. Query vectors can then be defined as a 1 x *p* vector of inputs. The query vectors are data that is run through the developed model to generate predictions. After the memory matrix has been defined, the distance or similarity between these vectors and the query vectors is calculated. For this work, the Euclidean distance measure is used to calculate the similarities. Equation 12 defines this distance measure, where *d* is the distance calculated between the new input *x* and memory observation X.

$$d_j = \sqrt{\sum_{i-1}^{n} (X_{j,i} - x_i)} \qquad (12)$$

Once these distance measures are calculated, a kernel function converts these values to weights. These weights will then be used to determine how similar or dissimilar the new input is to the values contained in the memory matrix. In this research, a Gaussian kernel function is used and is shown next in Equation 13.

$$K_j = \frac{1}{\sqrt{2\pi h^2}} \exp\left(\frac{-d^2}{2h^2} \right) \qquad (13)$$

In the preceding equation, *d* are the distance values calculated from (12) and *h* is termed the kernel bandwidth. The resulting weights are multiplied by the query vectors to yield the model predictions. The bandwidth parameter in (13) needs to be optimized to avoid under or over fitting of model predictions.

Optimization routines are used in order to minimize the error between the model predictions and input data. The tradeoff here is between an increase or decrease in the model accuracy and uncertainty. Regularization of the bandwidth can either increase the accuracy but introduce variance, or reduce the uncertainty but increase model bias. For this work, a grid search optimization routine is used to find the bandwidth that offers the least error between model predictions and input data.

3.4.2 Auto Associative Multivariate State Estimation Technique

The AAMSET modeling technique is based upon the same class of nonlinear nonparametric (NLNP) mathematics as the well-known MSET algorithm developed by Argonne National Laboratory (ANL) and commercialized by SmartSignal, but does not use the same proprietary kernel as the ANL MSET [90 and 91]. Instead, the AAMSET algorithm uses the same Gaussian kernel function that was shown in Equation 13. The same training data used to develop the models discussed in the previous section can also be used to develop this model type. This modeling method also uses less memory vectors than the AAKR model, though the predictions generated by this method are very similar to those obtained using the modeling methods described in the previous section. This means that using less memory vectors can reduce the overall compute cost when building and running these types of models. Aside from the similarity in using the same input data to build an AAKR model, there are differences in how predictions are calculated from input data.

As in the AAKR method, the distance between the memory matrix and query data is calculated using the same distance measure shown in (12). These distances are hereafter termed the similarity matrix. To obtain the model predictions, a pseudo-inverse method is first used to normalize the similarity matrix. The equation for the pseudo-inverse solution used to obtain the normalized similarity matrix "Nsimx" is shown next in Equation 14:

$$Nsimx = (\mathbf{X}^{\mathbf{T}} * \mathbf{X})^{-1} * \mathbf{X}^{T} \qquad (14)$$

In the preceding equation, **X** is the similarity matrix of distances. In keeping with the previous example, Nsimx would then be a 40 x 40 matrix. Once this normalized matrix is developed, the weights and predictions can be calculated for query vectors. The weights are obtained by using the Euclidean distance shown in (12). These are then multiplied by the query vector and Nsimx to generate the predictions, Equation 15 shows this combined result:

$$\mathbf{Y_p} = w' * ((\mathbf{X}^{\mathbf{T}} * \mathbf{X})^{-1} * \mathbf{X}^{T}) * Q \qquad (15)$$

In Equation 15, $\mathbf{Y_p}$ are the model predictions, w are the transposed weights multiplied by the normalized similarity matrix and the current query vector Q. The resulting residuals can then be used in the exact same way as those generated by the AAKR model. One drawback that can arise when using this method is that the pseudo-inverse solution can result in poor model predictions if the input data is ill-conditioned. Regularization of the similarity matrix during the inversion process mitigates and avoids this complication [92].

3.5 Sequential Probability Ratio Tests

The last core algorithm to be discussed is also the sole anomaly detection engine for this study. The SPRT is a binary hypothesis test that determines if the current observation being tested is in an unfaulted or faulted state [93]. The one assumption in using this test is that the data used to develop the model and residuals tested are Gaussian distributed, which in practice may not always be true. As the name implies, each observation in the query data is examined in sequence to determine the current operating state. The normal operating state is assigned to the null hypothesis H_o, while the faulted operating state is assigned to an alternative hypothesis H_1. For each observation, a test statistic is calculated as the natural log of a ratio of probabilities that either H_1 or H_o is true. This test statistic is known as the likelihood ratio L_N, and is shown next in Equation 16.

$$L_N = \frac{p(\{x_n\}) / H_1}{p(\{x_n\}) / H_0} \qquad (16)$$

To apply the likelihood ratio shown in (16) for anomaly detection, a decision criterion is used to calculate a lower and upper test bounds, A and B. For each observation, the likelihood ratio will be compared with these test bounds to determine if the observation is in a nominal or faulted state. In Equation 17, the A and B limits are defined using false and missed alarm probabilities.

$$A = \ln\left(\frac{\beta}{1-\alpha}\right), \quad B = \ln\left(\frac{1-\beta}{\alpha}\right) \qquad (17)$$

In the previous equation, α is the false alarm probability and β is the missed alarm probability. These two parameters are the probability of making a Type I or Type II error, respectively. If the value of the likelihood ratio is less than the lower bound A, then the observation is still in a nominal state. If the likelihood ratio is greater than the upper bound B, then the observation will be considered to be in a faulted state. If the likelihood ratio lies between these two bounds, then no decision can be made due to insufficient information and testing continues until the one of the test bounds is crossed.

It is important to mention that a cumulative sum of the likelihood ratio is calculated as testing continues. This sum is to account for the increasing probability that the test will eventually terminate. An example of this procedure for an unfaulted data set is shown next in Figure 3-10.

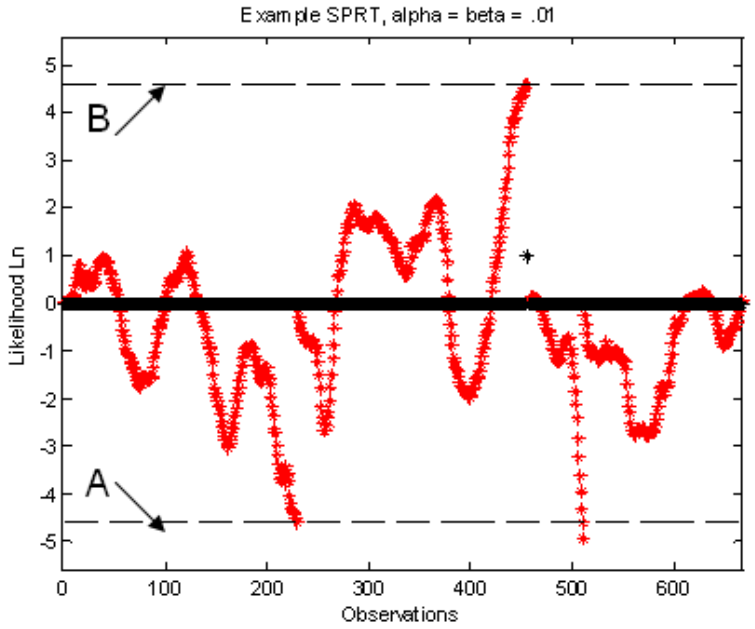


Figure 3-10: Example SPRT – Unfaulted Data

In the previous figure, the test bounds A and B are represented as the dashed lines and have values of -4.6 and 4.6, respectively. A and B are also termed the lower and upper test bounds of the SPRT, respectively. The black plot is the fault hypothesis, which assigns 0 for unfaulted and 1 for faulted states. The red plot is the likelihood ratio per observation. The oscillations in this plot are due to the small sum discussed earlier being added to each cumulative likelihood value, and from the sign of the current likelihood ratio value. The first point to mention is that the value of L_N was less than the lower limit A on two occasions. It is seen that the next observation after these two occasions that the running sum has been reset to 0, but no alarm indicator is given. This is because we only consider a 1-sided SPRT, in that we are only interested when the value of L_N exceeds the upper limit B. The likelihood ratio exceeds the upper limit B on one occasion. The sum is reset to 0 and the observation is flagged with a 1 as anomalous. Testing then continues until all observations in the data are tested.

If the reader has not been exposed to the SPRT outputs, Figure 3-11 on the following page provides an example output. This will be followed by a basic discussion on how to interpret the results and color scheme for the fault scores.

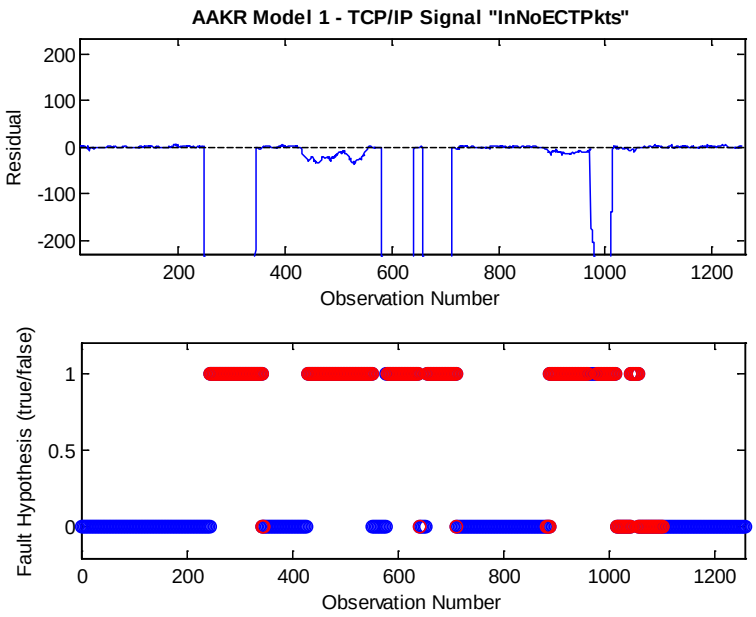


Figure 3-11: Example SPRT Output

In the previous figure, the expected output for a typical SPRT using anomaly data is shown. For the discussion on how to interpret the output, the reader is advised to review the previously discussed definitions for the True Positive Rate (TPR), True Negative Rate (TNR), False Positive Rate (FPR), and False Negative Rate (FNR) shown on page 20. The top subplot in Figure 3-11 is the residual generated by the particular model employed. Recall that the residual is the model prediction subtracted from the input data. For this case, actual anomaly data is used to generate the residual and fault scores. The x-observation ranges for each anomaly are 245:345, 430-550, 580:640, 655:710, 880:960, and 975:1100. That is, these regions are where an actual fault in the system is occurring. The bottom subplot is the fault hypothesis scores discussed earlier that is generated by the SPRT. Here, a 1 means that the SPRT flagged the particular observation as being in a faulted state and a 0 means the observation is unfaulted. Next, a red 1 indicates that the SPRT has correctly identified the observation as being in a faulted state, or a True Positive. A blue 0 indicates that the SPRT has correctly identified the observation as being in an unfaulted state, or a True Negative. Examination of the figure with the supplied anomaly ranges will confirm these previous statements. Next, a red 0 indicates that the SPRT incorrectly identified a faulted observation as unfaulted, or a False Negative. In the figure, there are a few red 0 values in the first half of the data and a lager segment from about 1050 – 1100. This means that the observations should have been labeled as faulted but were not. Last, a blue 1 would indicate that the SPRT incorrectly labeled an unfaulted observation as faulted, or a False Positive. Though slightly masked, there are a few FP near observation 980.

There are four possible probability states, or alternative hypothesis, tested by the SPRT in this work. The first two test whether there is a positive or negative shift in the mean, with constant variance, of the query data when compared to the mean of H_o. This type of test would indicate degradation of a monitored process. The other two hypotheses examine an increase or decrease in the variance of the query data, with constant mean, when compared to the variance of H_o. This type of test would indicate changes in the variability of a monitored process and could indicate the presence of a fault. Figure 3-12 provides a graphical representation of these alternative hypotheses.

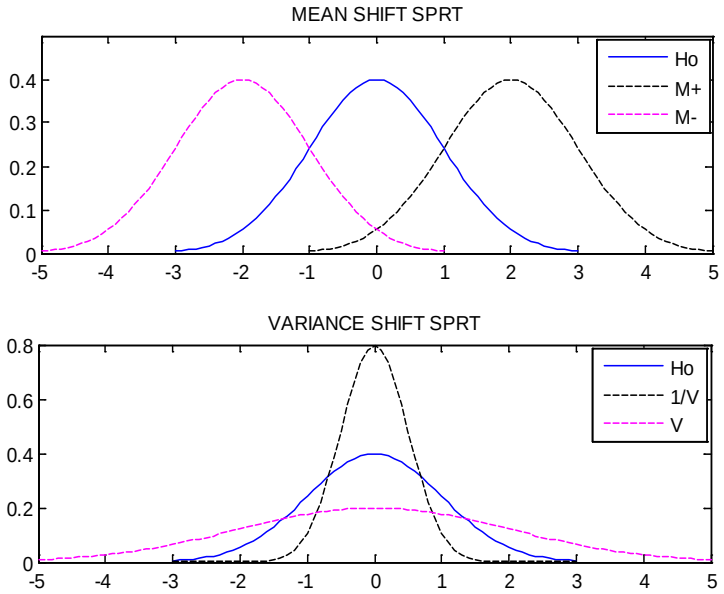


Figure 3-12: Mean and Variance Shift SPRT Distributions

The respective increase or decrease in the mean or given as M+ and M-, while the respective increase or decrease in the variance are given as V and 1/V. A shift in the mean would indicate that there is degradation of a process. A shift in the variance would indicate that there is an introduction of variability in a monitored process. The nominal distribution Ho is a normal distribution with zero mean and unit variance. Again, the nominal distribution is the distribution that the SPRT compares each succession with to determine if there is a shift in mean or variance. Each of these tests is run separately for all residuals of the query data. The alarm results for the mean shift tests are then combined to obtain the final fault hypothesis values. The same is true for the variance shift tests. Another consideration for the SPRT is alarm consolidation, which is used to remove spurious alarms. Per the original Wald theorem, spurious alarms occur randomly for even unfaulted data due to the cumulative sum discussed earlier.

For completeness, the equations for the mean and variance shift SPRTs are provided. More information on the derivation of the mean and variance shift SPRTs can be found in [93]. In the following equations, x_k indicates the current observation being tested, while M and V are defined as disturbance magnitudes for the mean and variance tests, respectively. Again, the mean shift SPRT tests if there is some form of change in a monitored process, while the variance shifts SPRT tests if there is some introduction of variability in a monitored process. All of these tests can be run simultaneously for each observation tested in a data set. The test for a positive mean shift is shown first in Equation 18.

$$SPRT_{M+} = \frac{M}{\sigma^2}\sum_{k-1}^{n} x_k - \frac{M}{2} \tag{18}$$

The negative mean shift SPRT is obtained by replacing the x_k term in (18) with $-x_k$, shown next in Equation 19.

$$SPRT_{M-} = \frac{M}{\sigma^2}\sum_{k-1}^{n} -x_k - \frac{M}{2} \tag{19}$$

The variance shift SPRTs consider the current observation faulted if the variance of the observation shifts by V or the inverse of V. Next, Equation 20 gives the variance shift SPRT for a change by a factor of V.

$$SPRT_V = \frac{1}{2\sigma^2}\left(\frac{V-1}{V}\right)\sum_{k-1}^{n}\left(x_k^2 - \frac{n}{2}\ln(V)\right) \tag{20}$$

Last, the inverse variance SPRT tests for a change in the distribution by a factor of 1/V, shown next in Equation 21.

$$SPRT_{1/V} = \frac{1}{2\sigma^2}(1-V)\sum_{k-1}^{n}\left(x_k^2 + \frac{n}{2}\ln(V)\right) \tag{21}$$

4 EXPERIMENT SETUP & INTRUSION TESTING

This chapter will provide a description of the components and software used in the development of the simulated SCADA test bed. Also included is a description of the various assumptions and task scripts that were developed to generate the simulated SCADA dynamics. Next, a description of the different intrusion testing software and codes that were used in this research is given.

4.1 SCADA Test Bed Description

The simulated SCADA test bed utilizes a set of Linux based enterprise servers that were isolated from the larger UT Nuclear Engineering Research Cluster. Isolation was required to ensure that activities on the test bed and on the larger cluster would not be reflected in the dynamics of each system. Isolation also ensures that intrusion-injection experiments would not impair the performance of the larger cluster. One server in the test bed acts as the Master Terminal Unit (MTU) in a typical SCADA system. This component is tasked with monitoring field equipment, issuing command actions and a variety of data processing actions. The remaining isolated servers act as Remote Terminal Units (RTUs), this equipment is tasked with maintaining field equipment operations and data transfer activities. Also included in the setup is a separate server that acts as a Human Machine Interface (HMI) or engineering workstation. In a typical SCADA system, this station serves to monitor the system and to issue control actions to the MTU. In the test bed, the HMI is used to monitor the simulation and to perform intrusion-injection experiments. Shown next in Figure 4-1 is a basic diagram of this setup.

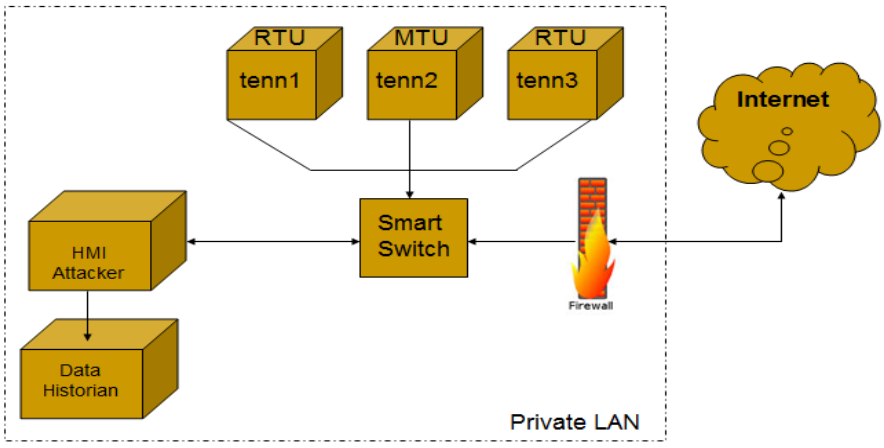


Figure 4-1: SCADA Test Bed Diagram

As mentioned previously, the experimental test bed is a simulation of a typical SCADA system. This is necessary because, for security reasons, sanitized process data from a SCADA or business-critical network is very difficult to obtain. A simulation does not devalue the research or results, as many of the advances made in the nuclear industry began with small scale simulations and test bed experiments that were then shown to have scalability to larger systems. With this in mind, the simulation is based on several assumptions. First, the dynamics of the MTU in a typical industry or asset manufacturing setting can exhibit periodic components. This is valid due to the fact that the MTU performs specific tasks such as data polling and processing on a regular basis. However, it is noted that some industrial systems will not show periodic components in monitored telemetry. This is not a problem as having periodic data is not a requisite to use any of the empirical-based models and SPRT algorithms described in this dissertation. It is also assumed that there will be a large amount of network traffic for systems with hundreds of components and sensors.

Task scripts were developed to first generate a baseline working profile of the MTU. These tasks perform what can be called short and long duration core loading activities. The short task loads all cores to around 95% of capacity. This simulates data polling and collection activities, as well as simple control actions. The MTU rests for a short period and then begins the long duration activities. Here, all cores are first loaded to simulate the data processing and other activities performed by the MTU. After a certain time period, three cores are loaded and must complete the same activities as all four cores. This continues until the long duration activities end. After this, the MTU rests for a short time period and begins the short duration activities. While these task scripts are running, the MTU is also collecting data from several sources and writing to a data store. This simulates additional data polling and storage activities of this component. Additionally, separate task scripts are run by both MTU and RTUs to simulate command actions and other network related activities. Combined, these additional task scripts generate sufficiently randomized and large amounts of network traffic.

All of these particular network actions are sent over SSH. The combination of MTU and network actions results in telemetry with periodic and random components that would be observed in the dynamics of a typical SCADA system. Monitored telemetry included resource and network related statistics taken from the Linux kernel. This telemetry includes measurements such as memory usage, packet numbers, page swaps and other network traffic related measurements. Also, information is collected that is related to the MTU internal hardware power usage and tachometer measurements. The software and hardware telemetry sources were chosen for this study because these are collected and recorded in typical industry networks already. This means that no additional telemetry sources need to be collected to implement these methods.

4.2 Intrusion Testing

As stated earlier in this dissertation, the goal of this research is to determine which types of intrusion activities can be detected using the methodologies presented. Also, there have been no reported uses of these methodologies in industry for cyber-security purposes. Then, successful detection of intrusion activities using these methods proves the efficacy of this study. Due to the ease of implementation of these modeling methods, and that telemetry already recorded by SCADA systems can be used, scaling of these techniques to a larger network seen in a typical Nuclear Power Plant (NPP) would be feasible with current technology. The one assumption in performing all of the intrusion testing is that an attacker has already gained access in some way to the private network. In practice, gaining access to a NPP may be more difficult due to the plant network isolation and various firewalls in the network.

Intrusion testing software primarily utilized Kali Linux and Metasploit [95, 96]. Kali is a well-known OS that contains several different penetration testing software packages. These software packages in Kali can be used to test for vulnerabilities on private networks or web-based applications, crack passwords, install backdoors and other post exploitation activities. Metasploit is another well-known penetration testing software package. This package contains hundreds of exploits that can be used against various types of OS, computer systems, processes and services. Metasploit was primarily used though a Graphical User Interface (GUI) called Armitage. This GUI allows for easier penetration testing as all of the codes and vulnerabilities are contained in modules with various options for the user to test. It is noted that many of the software packages in Kali and many of the Linux based exploits contained in Metasploit are not applicable to our system. For example, Kali contains several software tools related to web based or wireless attacks. Our system does not use web applications and is also connected to a private, wired network.

Intrusion testing codes utilized for this research next included past and current exploit codes related to Linux based systems taken from the Common Vulnerabilities and Exposures (CVE) database and Exploit Data Base (EDB) [72, 94]. These two sites provide references for hundreds of exploits that have been discovered for various systems. The CVE database lists past and current vulnerabilities discovered for many systems, software, and applications. This site also has exploit code to test. The EDB solely provides exploit codes developed by several users, though these are also based on past and current vulnerabilities. It is noted that many of these codes aren't applicable to our system or showed no perceivable effects on the system or dynamics after successful exploitation. This means the exploit was successfully implemented but the system showed no change in behavior. The last section of Chapter 5 will provide a complete list of what exploits were successfully detected and why others were not detected.

5 RESULTS AND APPLICATIONS

This chapter presents the results and applications related to the methodologies presented earlier in this dissertation. First, Section 5.1 presents the results of the developed Java-based time synchronous averaging technique called JTSAT. Time synchronous averaging techniques are used to resample telemetry sources that have different sampling rates. Next, Section 5.2 provides several case studies to prove the efficacy of the newly developed variable grouping technique called ACFgroup. Recall that variable grouping methods are needed to extract sets or subsets of variables that give the lowest model prediction errors. Last, Section 5.3 presents AAKR and AAMSET model results for several successfully detected intrusion activities from two different data sets. The models utilized four different clusters of telemetry taken from the Linux kernel to determine which class of telemetry was more effective for intrusion detection. The section concludes with a description of all intrusion activities that were performed during this research. This summary will discuss why each class of exploit was detectable or why not.

5.1 Java Time Synchronous Averaging Results

In this first results section, a comparison of the Oracle Analytical Resampling Process (ARP) algorithm with the developed Java Time Synchronous Averaging Technique (JTSAT) algorithm is given. These types of algorithms are required for intrusion detection and basic monitoring purposes because telemetry sources in most computer systems can tend to speed up or slow down depending on the number of processes running or other loading patterns. Also recall in Chapter 2 that one of the reported SCADA accidents which led to a loss of life was due to unevenly sample monitored telemetry. This led to inappropriate control actions that severely damaged the system in question. Related to this cyber-security research, the test bed collects two different sources of telemetry. These are software measurements taken from the Linux kernel and hardware measurements, such as power usage by various components of the server, that are taken from a separate piece of hardware/software embedded in the blade servers. The software measurement sampling rate is set by the Linux kernel and is 1 sample every 15 seconds. The sampling rate for the server hardware measurements is roughly 1 sample per minute. Then, to use both of these telemetry sources for intrusion detection purposes, the software and hardware data sources must be sampled to the same rate. For this comparison, resampled outputs from both algorithms are provided, followed by a comparison of the Auto Correlation Functions (ACFs) for each of the algorithm outputs. The ACFs are calculated because this will show if the JTSAT outputs captured the same signal dynamics of the resampled data as what was provided in the ARP algorithm.

To begin, these methods use two data sets with different sampling rates, hereafter termed D1 and D2. D1 is a slow sampled data set that records one sample every minute. This set also contains 1,000 observations and one signal. D2 is an irregular sampled set that records a sample roughly every second. This set also has 65,536 observations and four signals. Also provided with each data set are the corresponding time stamps. The main difference between the ARP and JTSAT methods is that the ARP method uses a regression type of interpolation to resample data streams, while JTSAT uses linear interpolation. This was done so that the JTSAT algorithm would not be identical to the ARP algorithm. This means that there will be observable differences in the resampled time series. The first result that will be shown is a comparison of the ARP and JTSAT algorithms when resampling data set D1, shown next in Figure 5-1.

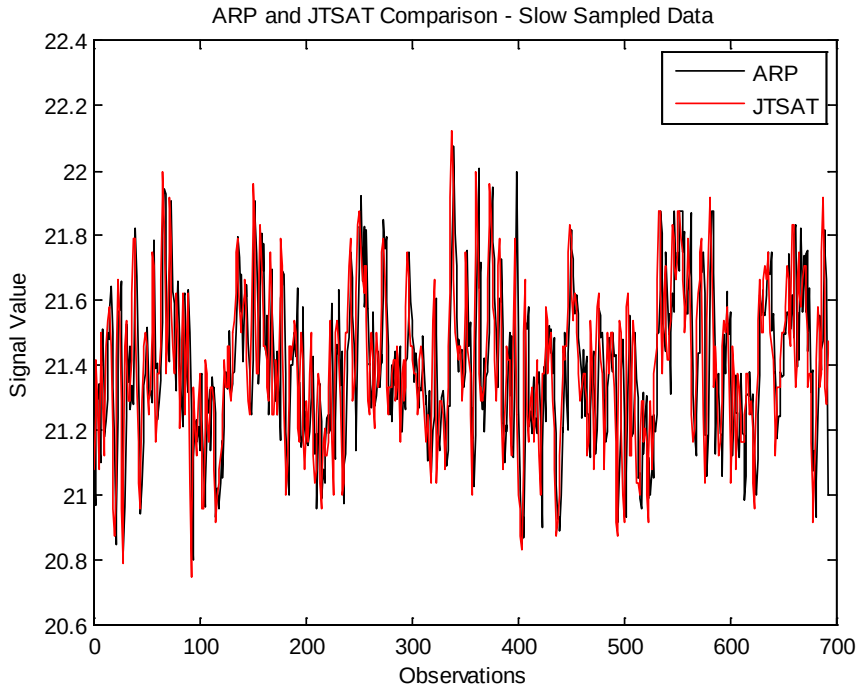


Figure 5-1: Comparison of ARP and JTSAT Output – Slow Sampled Data

In the previous figure, the first point to mention is that D1 has been reduced to about 700 observations, which were the regions of overlapping time stamps discussed previously. It is seen that the JTSAT output closely matches the ARP algorithm output. Again, these differences arise from the different resampling techniques employed by the two methods. Next, to prove that JTSAT was able to correctly capture the same signal dynamics as the ARP method when resampling data, Figure 5-2 shows the developed ACFs using 100 time lags for the outputs of each method.

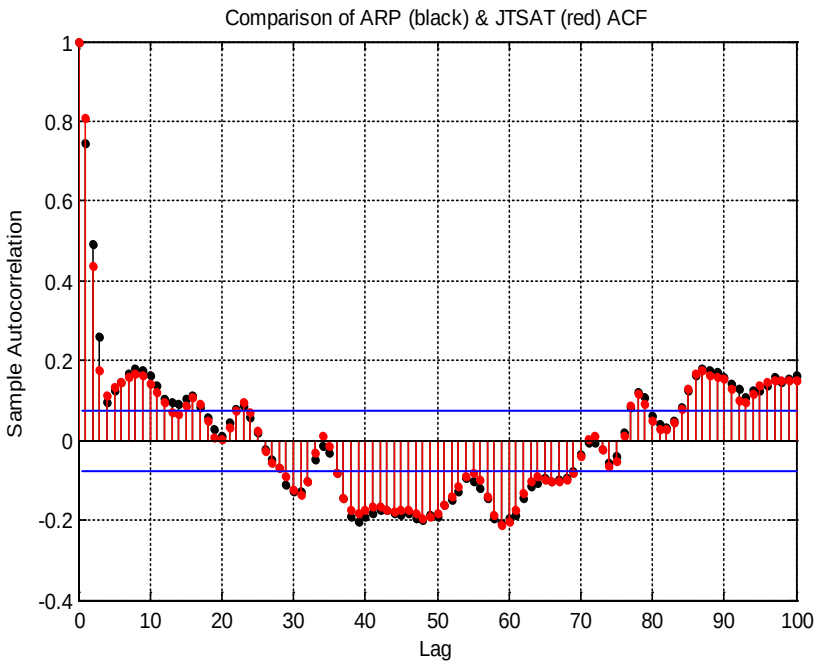


Figure 5-2: Comparison of ARP and JTSAT ACFs – Slow Sampled Data

In Figure 5-2, the black ACF is for the ARP method and the red ACF is for the JTSAT method. The point of showing the ACFs for these resampled signals is to prove that similar signal dynamics were captured by JTSAT as in the ARP method. While there are slight differences observed between both ACFs, it is easily seen in the figure that the JTSAT method was able to correctly capture many of the same signal dynamics as the ARP method. If the ACF for JTSAT was vastly different, then this would indicate that the signal dynamics were not captured during the resampling process.

On the following page, Figures 5-3 and 5-4 provide similar results when resampling data set D2. For these results, this large data set is resampled to the same size as the previous results, which results in about 700 observations. In Figure 5-3, it is seen that JTSAT was able to obtain similar dynamics as the ARP method. The main difference is that JTSAT did not fully capture the full range of the large downward spikes, though these spikes occur in the same location. This behavior is due to the linear interpolation used by JTSAT. The important point is that JTSAT was able to capture the same period in the data as the ARP method. In Figure 5-4, the ACFs for both techniques are again calculated using 100 time lags. Both ACFs in the figure are very similar, which means that JTSAT was again able to correctly capture similar signal dynamics when compared to the ARP method. This then proves that the JTSAT algorithm is as useful as the ARP method for slow and uneven sampled data sets.

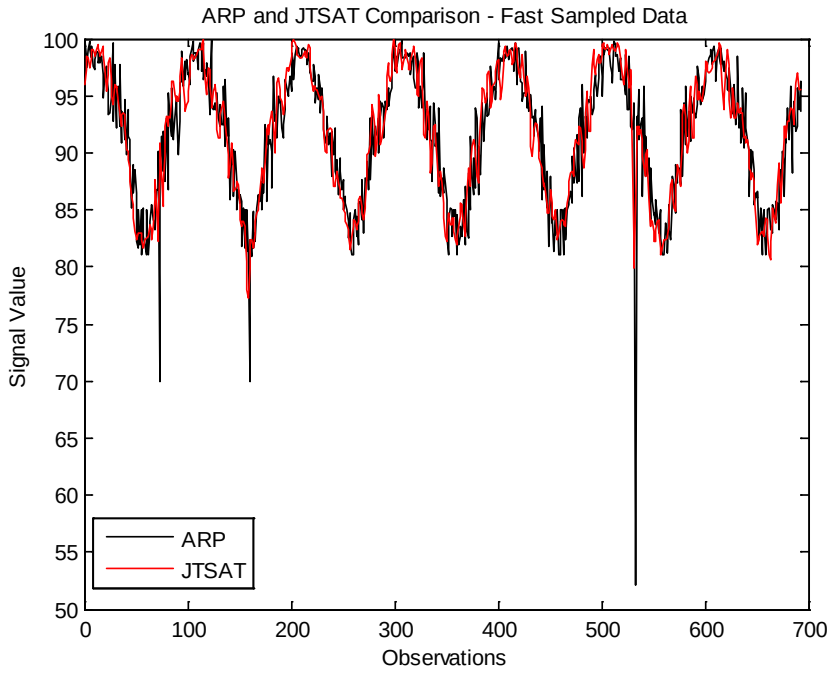


Figure 5-3: Comparison of ARP and JTSAT Output – Fast Sampled Data

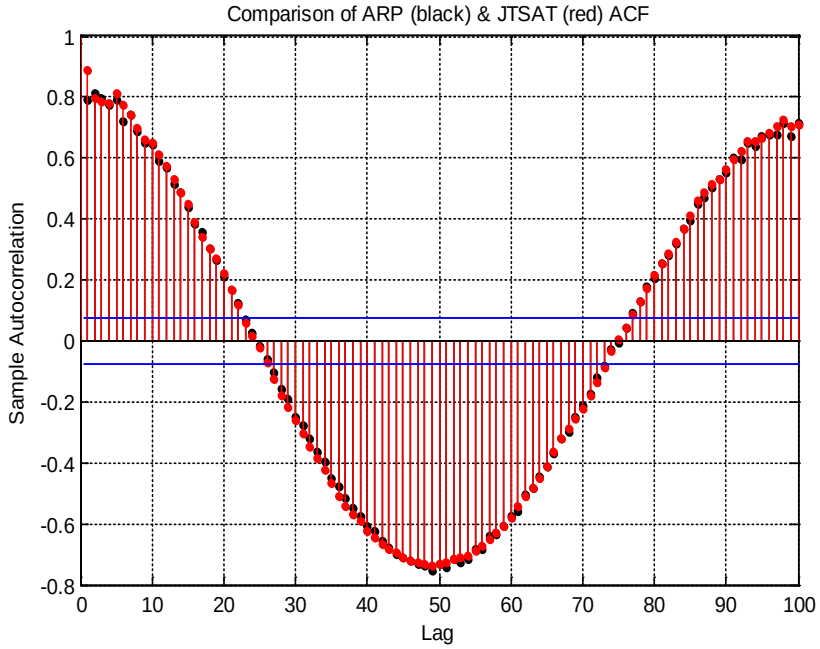


Figure 5-4: Comparison of ARP and JTSAT ACFs – Fast Sampled Data

5.2 ACFgroup Applications

In this section, the proof of principle for the ACFgroup algorithm is provided using seven real-world data sets. For these studies, several functions contained in the Process and Equipment Monitoring (PEM) toolbox, which is a MATLab based set of tools developed at the University of Tennessee, are used for variable grouping and model development [86, 87]. For all studies, the AAKR model is developed to quantify overall error. By necessity, an AAMSET or NN model will also return equivalent average model errors when using the same input data and signals. For these studies, the performance of ACFgroup is compared against two other correlation coefficient variable grouping functions contained in the PEM toolbox called "*roughgroup*" and "*autogroup*". While both of theses functions use correlation coefficients to arrive at final variable groups, each of these algorithms has vastly different functionality. In both of these algorithms, signals with an absolute correlation coefficient value of .7 and above indicate strong linear relationships, values between .3 and .7 indicate moderate linear relationships, and values below .3 indicate little to no linear relationships.

To start, roughgroup is a very simplistic approach to variable grouping. This function calculates the correlation coefficients for the input data and places variables in groups based on how many other variables have similar correlation coefficients. This function will at most return only three groups, if applicable: a strong and moderately correlated group, and a group of removed variables. In contrast, autogroup is much more sophisticated than roughgroup. This particular function still uses the correlation coefficients to group variables, but also combines what is known as the Symmetric Reverse Cuthill-McKee algorithm with many other nested sub-functions. These sub-functions attempt to merge smaller groups into larger groups or place single variables into larger groups. Autogroup can return several final groups of variables, along with a removed set of variables.

For each case study, the outputs from each algorithm are treated with complete trust. That is, the user does not check that each group returned by the functions contains variables appropriate for further model development. In five of these studies, potentially troublesome variables were removed beforehand. Troublesome variables include constant, near-constant, quantized and those with minor changes over the entire range of the data. In two case studies, no variables were removed before using the grouping algorithms for a so-called "double blind" test. This means that the input data for all grouping functions in these two cases could contain variables that were all characters, constants, or near-constant. This was done to access the overall performance of all variable selection techniques when exposed to unknown data sets that can contain any type of variable. These two studies will be called out for the reader.

To quantify overall performance, the average error in the model predictions is shown. Each case study will provide available information about the data set being used, such as size, algorithm runtime and any other special notes. Also provided for each case study is a correlation coefficient matrix plot of the input data. This plot is provided so the user can compare the signals selected by the PEM functions and compare these with the variable groups obtained from ACFgroup. Using the correlation coefficient ranges described earlier and the color bar provided for each plot, strongly correlated variables will be red, moderately correlated variables will be yellow and those with little correlation will be blue. Also, if a variable is constant valued, these will be white. All models are trained, optimized and validated using the same data sets. The only difference in the results will be what signals were selected for each model.

5.2.1 Case Study 1: Blade Server Hardware Data

This first case study uses hardware measurements for a blade server used in the SCADA test bed described in this dissertation. This data set has 22,868 observations and 28 signals. Based on the correlation coefficient plot shown in Figure 5-5, Signals [1:4, 9 & 14] are motherboard temperature measurements, Signals [8, 12:13 & 15:28] are fan speeds and related system voltages. The remaining variables are current measurements for specific small components of the blade server.

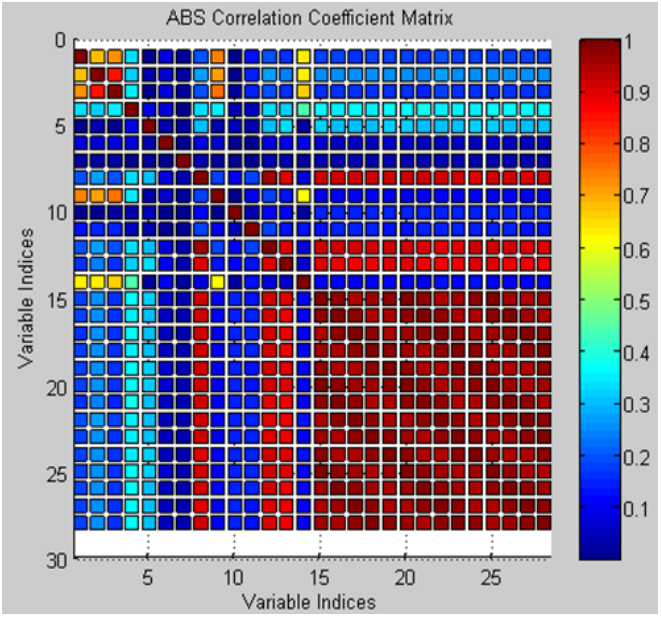


Figure 5-5: Case Study 1 Correlation Coefficient Matrix

The described data is then input through each of the previously described grouping algorithms. The signals selected by each of these grouping algorithms, along with the average percent model error, are shown next in Table 5-1.

Table 5-1: Case Study 1 Grouping Results

Method/Groups	**Auto**	**Rough**	**ACF**
Gr1	1:5, 8, 9, 12:28	3, 8, 9, 12, 13, 15:28	8, 12, 13, 15:28
Gr2	[]	1, 2, 4, 5, 14	1:4, 9
Removed	6, 7, 10, 11	6, 7, 10, 11	5:7, 10, 11, 14
Model Error (%)	**7.30**	**5.43** **5.61**	**3.77** **1.39**

In Table 5-1, it is first seen that autogroup extracted one group of variables, which were all the strongly correlated variables shown in the previous figure. The roughgroup algorithm extracted two groups of variables and also removed the same variables as autogroup. ACFgroup extracted two groups of variables, however these groups were slightly different than those obtained using the correlation coefficient algorithms. Also, ACFgroup removed a slightly different group of variables than the other two algorithms. When examining the average model errors in the table, it is seen that ACFgroup had the lowest error measurements for both extracted groups. The question to be answered now is why the model error values for ACFgroup are lower than the other two grouping functions.

First, the group extracted by the autogroup algorithm used all strongly correlated variables of the input data. This can be confirmed by examining the correlation coefficient values for these variables in the previous figure. Because this and the roughgroup function select variables based solely on the resulting correlation coefficients, then if a variable that is unsuitable for modeling; such as a near-constant time series; has a strong correlation with others in a date set, it will be included. Then the resulting model error values for the group obtained by the autogroup function indicate that one or more variables included in this group was in actuality not suited for modeling. Next, the model errors for both groups obtained by the roughgroup function were slightly less than that seen for autogroup. However, what is interesting is that Gr1 obtained by ACFgroup had an even lower model error than Gr1 of autogroup. The difference in Gr1 of ACFgroup was only two signals, which were numbers 3 and 9. This indicates that maybe these two signals have some dynamics that will increase the overall model error. Shown in Figure 5-6 on the following page is a comparison of the ACFs for the variables in Gr1 that were obtained by the roughgroup and ACFgroup algorithms.

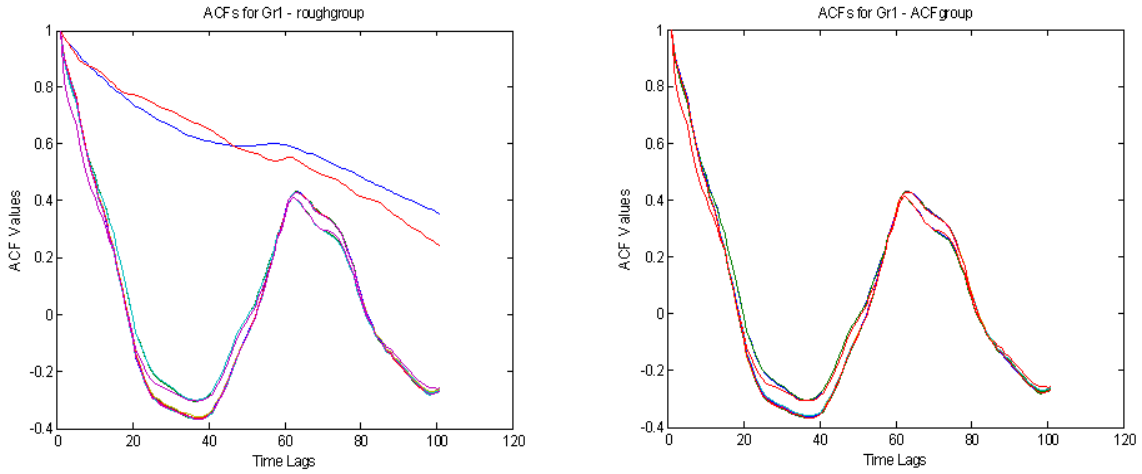


Figure 5-6: Comparison of ACFs for Gr1 - roughgroup & ACFgroup

In the previous figure, it is seen in the left hand plot that there the ACFs for two of the signals are very different in shape or behavior than the others in this group. These two signals are indeed numbers 3 and 9 that were called out earlier. The ACFs for these two signals are examples of what was termed a trend time series that show a moderate degree of persistence because the ACF values for these signals are always positive. However, these two signals are still suitable for modeling as both have a slowly decaying trend towards negative ACF values. Inclusion of these signals by roughgroup has the effect of increasing the average model error. It is also seen in the right hand plot that ACFgroup did not include these signals. Next, Figure 5-7 shows Gr2 extracted by these two algorithms.

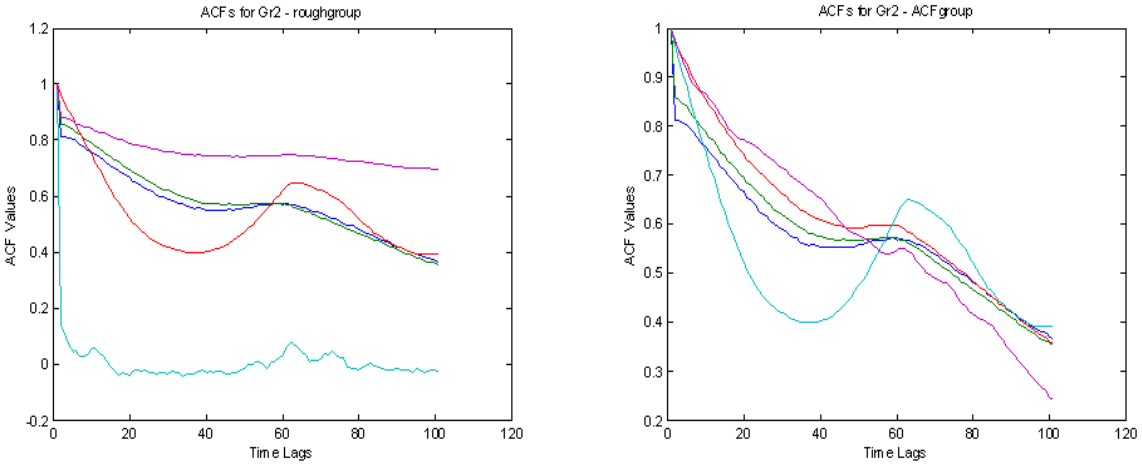


Figure 5-7: Comparison of ACFs for Gr1 - roughgroup & ACFgroup

In the previous figure, it is seen that the ACFs obtained by roughgroup in the left hand plot have two signals that are problematic. These are the teal and purple signals, which correspond to original signals 5 and 14. The ACF for signal 5 is an example of a near-constant time series, which shows minor oscillations around zero. Signal 14 is an example of a highly persistent ACF, the ACF values for this signal have a very slow decay and most values are around .8. This means that the original signal is unsuitable for modeling, even though it had moderate to strong correlations with other signals of the input data. The ACF shape also indicates that the original signal had very few changes over the entire range of the data. The ACFs in the right hand plot of Figure 5-7 are examples of trend time series; this is indicated by the rapid decline of the ACF values. Also of interest in this figure is the teal ACF, which is an example of a periodic time series. This signal was included in this group by ACFgroup because the ACF had similar statistical values as the ACFs for other signals of this group. Recall that ACFgroup can place many different types of time series in a group so long as the statistical measures were similar. This means that the original signals experienced the same changes in dynamics for the same system inputs. For completeness, the ACFs for Gr1 extracted by autogroup are shown in Figure 5-8. In this figure, the ACFs show that there are many time series that show persistent behavior or slow changes over the entire range of the original data. There are also trend and periodic signals that are included in this group. However, the ACFs again indicate that many of these signals did not show the same changes in dynamics for the same system inputs.

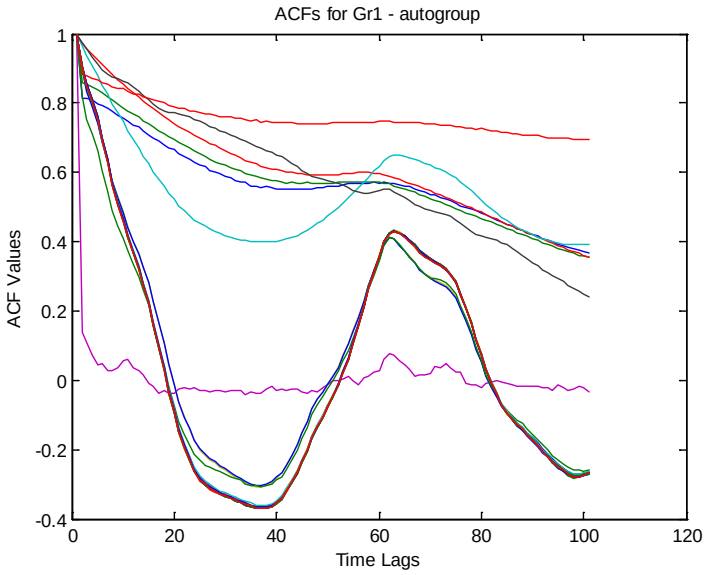


Figure 5-8: ACFs for Gr1 - autogroup

5.2.2 Case Study 2: Anonymous Nuclear Power Plant Data

This second case study uses a data set that is contained in the PEM toolbox and was taken from a nuclear power plant. The set comprises signals from three different, unrelated plant systems. Signals 1:2 belong in System 1, 3:7 in System 2 and 8:19 belong to System 3. Not other information was provided about this data. This set has 800 observations and 19 signals. Figure 5-9 shows the correlation coefficient plot for this data set.

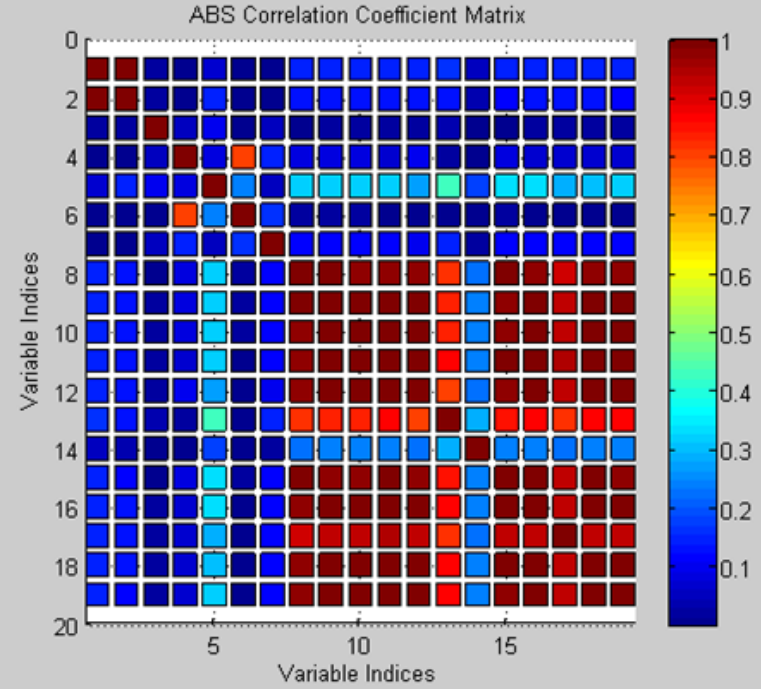


Figure 5-9: Case Study 2 Correlation Coefficient Matrix

The groups obtained by each of the grouping algorithms using this data set and the average model errors are presented next in Table 5-2.

Table 5-2: Case Study 2 Grouping Results

Method/Groups	**Auto**	**Rough**	**ACF**
Gr1	1,2,4:6, 8:19	8:13, 15:19	8:13, 15:19
[]	[]	[]	[]
Removed	3, 7	1:7, 14	1:7, 14
Model Error (%)	**15.39**	**1.46**	**1.46**

In Table 5-2, it is first noted that all grouping algorithms were able to extract only one group from this data set. Another interesting item is that roughgroup and ACFgroup removed the same set of variables and thus obtained the same variables for Gr1. The variables for this group were all from taken from System 3. While it may not seem impressive that ACFgroup arrived at the same set of variables as roughgroup, what is impressive is that this was accomplished by using a completely different approach to variable selection than correlation coefficient criteria. The average percent model error for roughgroup and ACFgroup was 1.46%. Next, the autogroup function placed many variables from all listed systems into one group. Because these variables had so little in common, the resulting model error of 15.39% for the autogroup model was substantially larger than what was obtained by the other two algorithms.

5.2.3 Case Study 3: SCADA Test Bed Telemetry Data

For this case study, the data set used the software variables for the SCADA test bed. All of these signals are taken from the Linux kernel. This data set has 22868 observations and 54 signals. In Figure 5-10, Signals 1:15 are memory usage metrics, Signals16:35 are network or TCP/IP related, Signals 36:45 are disk usage measurements, and Signals 46:54 are CPU usage measurements.

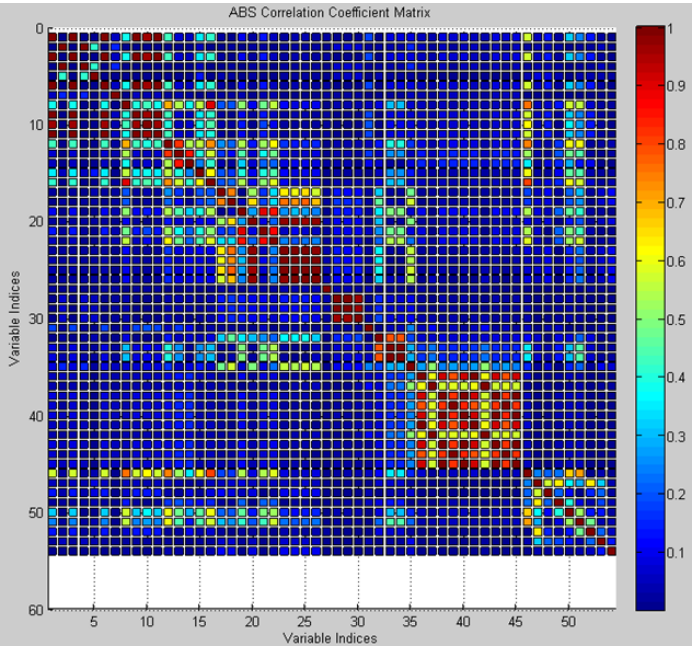


Figure 5-10: Case Study 3 Correlation Coefficient Matrix

Table 5-3 shows the groups selected by all grouping methods. In the table, it is first noticed that autogroup was able to extract five different groups, while the other two algorithms extracted only two groups. When considering autogroup, it is seen that Gr2 has a very small model error of 0.70%. All of the signals in this group were highly correlated and highly quantized. Thus, autogroup placed them into their own group based on their high degree of cross correlation. Also note that ACFgroup removed all signals seen in this group. The remaining four groups selected by autogroup had small to large model errors, many of the signals contained in these groups were selected or removed by the other two grouping algorithms.

Table 5-3: Case Study 3 Grouping Results

Method/Groups	**Auto**	**Rough**	**ACF**
Gr1	8, 12:16, 46:53	3, 6, 9:12, 16, 18, 20, 23:26, 36, 38:41, 43:46	8, 15:18, 20, 23:30, 32:34, 36, 38:41, 43:46, 50, 51
Gr2	1, 3, 6, 9:11	2, 4, 5, 8, 13:15, 17, 19, 21, 22, 28:30, 32:37, 42, 47, 50, 51	7, 12:14, 19, 21:22
Gr3	17, 18, 20, 24:26, 28:30, 35	[]	[]
Gr4	19, 21, 22, 32:34	[]	[]
Gr5	2, 4, 5, 36:45	[]	[]
Removed	7, 27, 31, 54	1, 7, 27, 31, 48, 49, 52:54	1:6, 9:11, 31, 35, 37, 42, 47:49, 52:54
Model Error (%)	**7.43, 0.70, 4.20, 3.08, 39.9**	**5.77, 28.5**	**8.30, 4.91**

When considering the two groups extracted by roughgroup, Gr1 had a low model error. It is noted that all but one variable in autogroup Gr2 is also contained in this group. Due to the extremely low error of autogroup Gr2, then roughgroup Gr1 will also have a lower overall error. Many of the signals in this group were also selected by ACFgroup Gr1. However, ACFgroup removed all signals from autogroup Gr2. Because these are not included in any of the ACFgroup models, Gr1 of ACFgroup has a slightly larger model error. Gr2 selected by roughgroup and Gr5 of autogroup also had a very large model errors due to problematic time series. It is noted that many of the variables in these two mentioned groups were removed by ACFgroup.

5.2.4 Case Study 4: HRSG Boiler Leakage Data

This data set was supplied by Alstom Power and was taken from a natural gas power plant that utilized a Heat Recovery Steam Generator (HRSG). The process measurements contained in this set are a mix of plant temperatures, pressures, flow rates, turbine speed and related power measurements. This data set also has a very slow sampling rate. The information provided states that this data set was average over a 1 year period. It is noted that several signals of this data set are constant or near-constant valued, indicated as white rows/columns in the correlation coefficient plot shown in Figure 5-11. This particular data set has 7,000 observations and 56 signals.

NOTE: For this study, no signal preprocessing is performed before using the three grouping algorithms. That is, no constants, near-constant or other troublesome types of signals are removed to improve the performance of all grouping methods. This blind study was done to assess ACFgroup performance for these types of unknown inputs, and to compare performance with the other correlation grouping methods. On the following page, Table 5-4 lists all of the groups extracted in this blind case study.

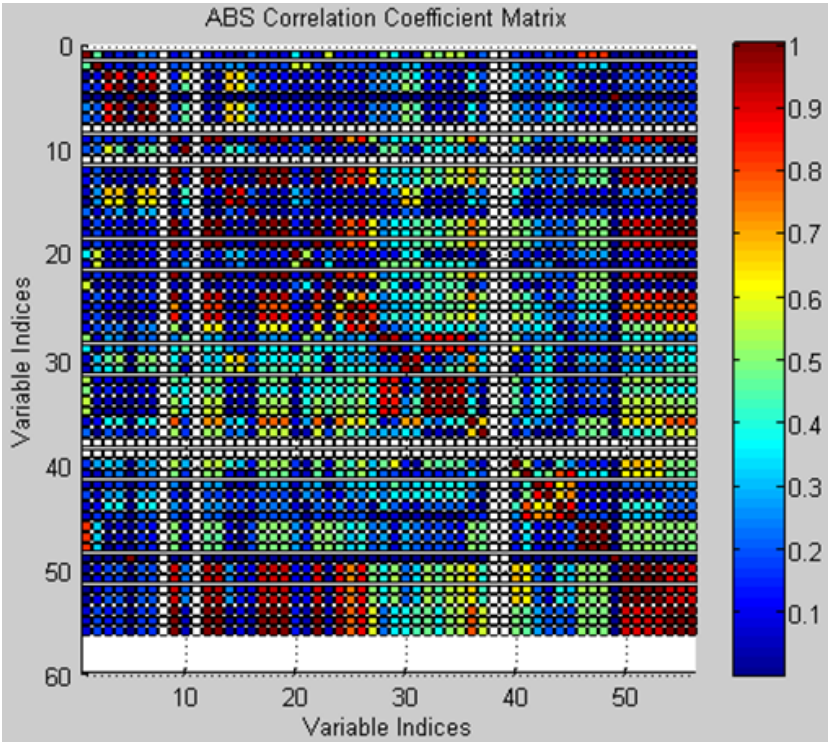


Figure 5-11: Case Study 4 Correlation Coefficient Matrix

Table 5-4: Case Study 4 Grouping Results

Method/Groups	Auto	Rough	ACF
Gr1	21, 28, 29, 32:35	1, 3, 4, 6, 7, 9, 12, 13, 17:19, 22, 24, 28, 29, 32:35, 45:48, 50:56	9, 12, 13, 17:19, 22, 24, 26, 40, 41, 44, 45, 50:56
Gr2	1, 2, 9, 12, 13, 17:20, 22:31, 36,37, 40:48, 50:56	2, 10, 14, 15, 20, 21, 23, 25:27, 30, 31, 36, 37, 40:44	16, 25, 27, 36, 37, 42, 43
Gr3	3, 4, 6, 7, 10, 14:16	[]	[]
Removed	5, 8, 11, 38, 39, 49	5, 8, 11, 16, 38, 39, 49	1:8, 10, 11, 14, 15, 20, 21, 23, 28:35, 38, 39, 46:49
Model Error (%)	**Inf, Inf, 13.38**	**10.88, Inf**	**3.24, 5.18**

In Table 5-4, the "Inf" values seen for several of the models indicate infinite model error. This unreasonable error metric means that some time series type has been included in the final groups whose dynamics will cause mathematical issues. These mathematical issues are mainly division by zero or division by very large or small values. Because the PEM toolbox utilizes statistical measures of the predictions to arrive at the final error metric, a signal with few changes over the entire range of data will cause this very large error.

In this data set, Signals 20, 21 and 23 are near-constant time series that cause this large error. These three variables were included by auto and roughgroup because they had some degree of strong cross correlation with other variables in this set. However, the ACFgroup algorithm has correctly removed these and other troublesome signals due to their insignificant signal changes. The groups extracted by ACFgroup also had the lowest model error values for all extracted groups shown. Next, all three algorithms were able to correctly remove Signals 8, 11, 38 and 39. These were all constant valued time series and are indicated by the white rows/columns seen in the previous figure. This means that the cross correlation algorithms could identify constant valued signals as unwanted signals, but they could not identify signals that have very few changes over the entire range as unwanted signals. The results shown in the table indicate that the ACFgroup method is more robust to data sets that contain many different types of time series than correlation coefficient grouping methods. This means that examining observed changes in signals of a data set can be more useful for grouping applications.

5.2.5 Case Study 5: Fossil Power Plant Turbine Blade Failure Data

This data set was also provided by Alstom Power, but is taken from a different type of plant than what was shown in the previous case study. Also, the information provided states that this data set was averaged over a 1.3 year period. This set contains 9,790 observations and 30 signals. In Figure 5-12, the two large clusters of red variables are process temperature and pressure measurements; while the blue are turbine speeds and other related power variables. Also on this page is Table 5-5, which provides all extracted groups and error metrics.

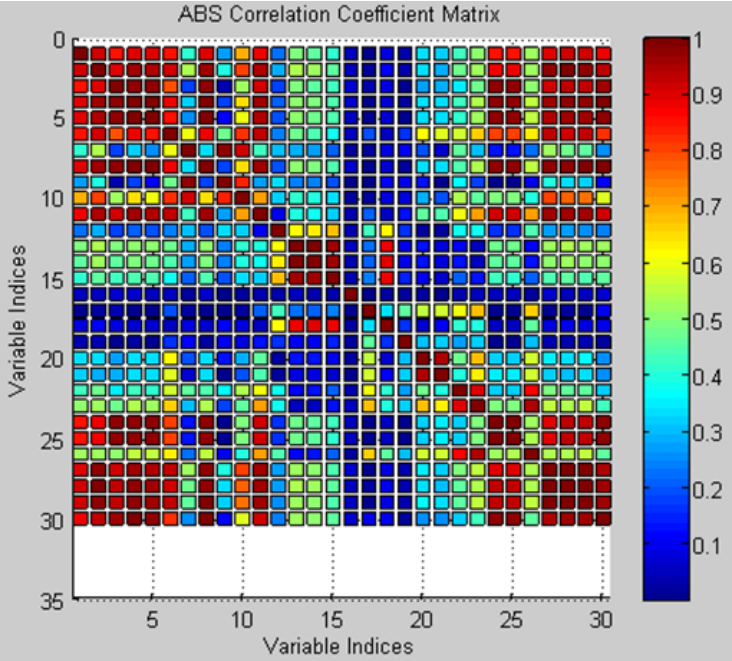


Figure 5-12: Case Study 5 Correlation Coefficient Matrix

Table 5-5: Case Study 5 Grouping Results

Method/Groups	**Auto**	**Rough**	**ACF**
Gr1	1:15, 17:30	1:6, 8, 11, 13:15, 18, 23:30	1, 3, 9, 24, 25,30
Gr2	[]	7, 9, 10, 12, 17, 20:22	16, 17, 20:23, 26
Gr3	[]	[]	2, 4:8, 10, 11, 19, 27:29
Removed	16	16, 19	12:15, 18
Model Error (%)	**7.56**	**6.98, 4.49**	**4.58, 4.21, 3.17**

In the previous table, autogroup was able to extract one group of variables and removed one variable. This is the main turbine RPM speed. Roughgroup also removed this signal, along with the variable for generated power. ACFgroup includes these two important variables in two of the three extracted groups and removes an entirely different set of variables than the other two methods. The turbine RPM and generated power are important measurements for a steam powered plants and should have some relationship with the other plant measurements. When considering the model errors, Gr1 for auto and roughgroup were both larger than the similar group extracted by the ACF method. In fact, all of the model errors for the groups extracted by ACFgroup were lower than the other two methods. These results again show how grouping variables based on observed signal changes can result in models with lower average error.

5.2.6 Case Study 6: Heat Exchanger Fouling Data

The data set for this case study was taken from a heat exchanger fouling test bed experiment performed at UT. This set has 1,000 observations and 13 signals. In Figure 5-13, Signals 1:4 and 9:13 are temperature related, while Signals 5:8 are mass flow rates. On the following page, Table 5-6 provides the grouping results for this data set.

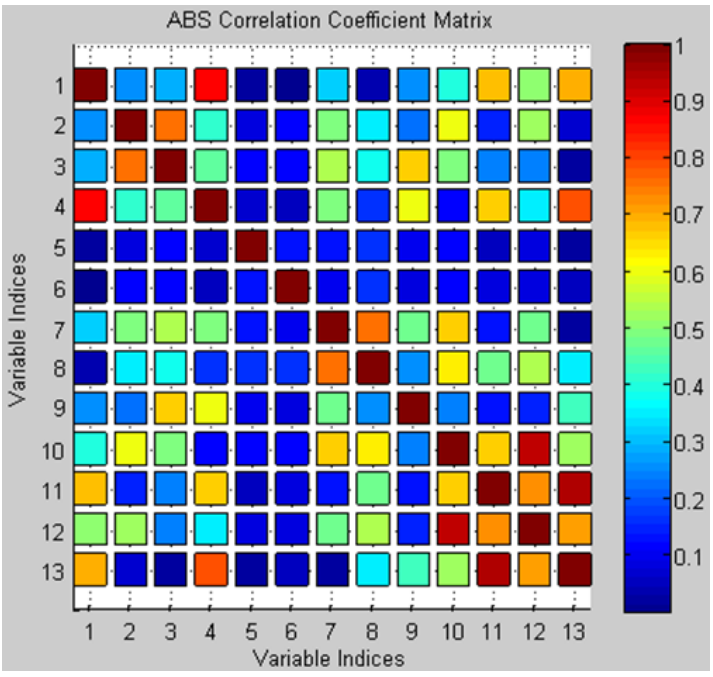


Figure 5-13: Case Study 6 Correlation Coefficient Matrix

Table 5-6: Case Study 6 Grouping Results

Method/Groups	Auto	Rough	ACF
Gr1	1, 4, 10:13	12, 13	1, 4, 11, 13
Gr2	2, 3, 7:9	1:4, 7, 8, 10, 11	2, 3, 7:10, 12
Removed	5, 6	5, 6, 9	5, 6
Model Error (%)	**8.97, 14.76**	**(9.31), 15.66**	**7.08, 13.65**

In Table 5-6, it is seen that all algorithms removed only mass flow related variables, which were variables 5 and 6. Autogroup placed many of the temperature measurements into one group and several temperature variables and mass flow rates into a separate group. The second group extracted by roughgroup also used signals from both subsets. While autogroup and ACFgroup were able to arrive at two somewhat similar groups, roughgroup could only arrive at one final group. Gr1 extracted by roughgroup only had two variables; in reality it makes no sense to develop any type of model with only two variables. However, for completeness, the error value for this insignificant model is also provided. Gr2 of roughgroup is similar to Gr1 of the other two methods; however, roughgroup termed this group as a set of moderately correlated variables. When comparing all average error results, it is seen that ACFgroup was again able to arrive at final model groups that have lower model errors.

5.2.7 Case Study 7: CMAPSSData: Motor Failure Data

The last data set that will be described in this section is a motor degradation data set that was taken from the NASA Prognostics Data Repository [98]. This data set contains normal operational and anomaly data from an accelerated aging motor test bed. In this data set, Signals 5, 23 and 24 are constant valued. This is shown as the white rows/columns in Figure 5-14 on the following page. The remaining signals in this set are either temperature or vibration related. This data set contained 20,631 observations and 36 signals.

NOTE: As in Case Study 4, a blind case study is performed on this data set. This means that no additional preprocessing of this data set is performed to remove potentially problematic variables. Again, problematic variables include constant, near-constant, quantized or variables that show very few changes over the entire range of the data. These can cause sever mathematical issues that can potentially lead to very poor model predictions and errors. This blind study was done to assess ACFgroup performance for these types of unknown inputs, and to compare performance with the other correlation grouping methods. On the following page, Table 5-7 lists all of the groups extracted and the average model errors obtained in this second blind case study.

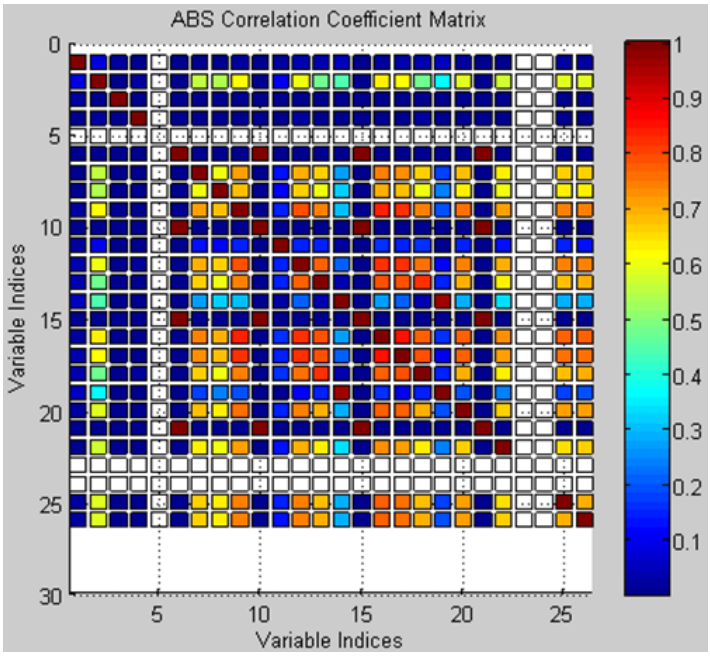


Figure 5-14: Case Study 7 Correlation Coefficient Matrix

Table 5-7: Case Study 7 Grouping Results

Method/Groups	**Auto**	**Rough**	**ACF**
Gr1	2, 7:9, 12:14, 16:20, 22, 25,26	6, 9, 10, 12, 13, 15:18, 20, 21	2, 9, 14, 16, 17
Gr2	[]	2, 7, 8, 14, 19, 22, 25, 26	7, 8, 12, 13, 18:20, 22, 25, 26
Removed	1, 3:6, 10, 11, 15, 21, 23,24	1, 3:5, 11, 23,24	1, 3:6, 10, 11, 15, 21, 23, 24
Model Error (%)	**5.51**	**Inf, 4.45**	**3.78, 4.65**

In Table 5-7, the first item of interest is Gr1 for roughgroup had an "Inf" error value. This means that some type of time series was included in this group and not the other groups that cause this very large error value. Next, it is noted that autogroup and ACFgroup removed exactly the same variables under a blind case study. When considering the model error results, ACFgroup arrives at models with better error values for all but one group. In this case, the increase in error for ACFgroup is only 0.2% over a similar group that was extracted by roughgroup. These results again show that the ACFgroup algorithm is more robust to data sets with a wide range of time series.

5.2.8 Summary of ACFgroup Results

The proof of principle for the ACFgroup algorithm was shown using seven different data sets. These data sets were taken from a NPP, two different fossil fuel plants, the SCADA test bed simulation, and several accelerated aging test beds. The algorithm was compared against two other VGM that are based on correlation coefficient criteria. Recall that the engineering basis of the ACFgroup algorithm utilizes the observed signal changes among all input variables of a data set to group variables. For this study, AAKR models were developed using the final variables selected by each of the grouping algorithms. To compare overall effectiveness of each algorithm, the average percent model error for each developed model is provided. This error metric is the obtained by summing the error metric for each signal, dividing by the total number of signals, finally multiplying this value by 100%. The average percent error for each case study, model and VGM is shown next in Table 5.8.

Table 5-8: Summary of Average Model Errors for Each Case Study

Method/Case	Autogroup	Roughgroup	ACFgroup
1	7.30	5.43, 5.61	3.77, 1.39
2	15.39	1.46	1.46
3	7.43, 0.70, 4.20, 3.08, 39.9	5.77, 28.5	8.30, 4.91
4	Inf, Inf, 13.38	10.88, Inf	3.24, 5.18
5	7.56	6.98, 4.49	4.58, 4.21, 3.17
6	8.97, 14.76	9.31, 15.66	7.08, 13.65
7	5.51	Inf, 4.45	3.78, 4.65

In the previous table, the "Inf" score indicates that the respective model had infinite error. This result is due to the grouping algorithms selecting final variables that were unsuitable for modeling. These unsuitable signals include constant or near-constant valued variables that cause mathematical issues, such as division by zero. It is noted that none of the ACFgroup average errors had an "Inf" value, while several variable groups selected by cross correlation criteria returned this particular error. Next, it is noted in Case 3 that ACFgroup has larger average errors than the other two algorithms. However, given that the error for the first model of this case in ACFgroup contained variables also seen in models 1, 3 and 4 for autogroup, the increase in error of 0.93% for ACFgroup is acceptable. In the last case study, the increase in average error for the second model of ACFgroup when compared to the second model of roughgroup was only 0.2%. The overall results definitively show that ACFgroup was able to extract superior groups of variables for data-driven modeling.

5.3 SCADA Test Bed Intrusion Testing Results

The last section of this chapter will present the model and intrusion detection results using for two different data sets. The first set was collected in June of 2016 and contains six different intrusions tested in succession. The second set was collected in March of 2017 and contains eight different intrusions tested in succession. During the course of this cyber-security research, there have been several sets of data collected using the SCADA experiment. As many of these data sets show the same types of intrusions with variants in options chosen, for brevity only the results for the previously two mentioned data sets are shown. The methods for specific signal processing of certain telemetry sources and variable grouping is first discussed. These specific signals were all monotonically increasing and had to be pre-processed before being used in for model development. This is then followed by the results of the June and March data sets. The results for each set will describe intrusions tested, correlation coefficient plots for all variables groups, and SPRT plots for two selected signals from each developed model. It is noted in these results that every intrusion tested could be detected in one or more models using the developed methods. Last, this section concludes with a summary of all intrusion classes tested during this research, along with the various intrusion testing software or exploit codes. The discussion will focus what each class of intrusion is used for, how they are implemented, and the reasons a particular tested intrusion could or could not be detected. This will provide the reader with a qualitative assessment of the presented methods for intrusion detection.

5.3.1 Signal Processing and Variable Grouping

In total, there were 665 signals collected from various telemetry sources from the servers of the SCADA test bed. These can be separated into various telemetry classes that include memory usage, TCP/IP related statistics, disk usage, CPU usage, and page swaps. It was found during initial signal processing that a large majority of these signals were either constant valued, had minor changes over the range of the data, or were monotonic. For the modeling methods discussed in this dissertation, all of these signals would be removed by correlation coefficient grouping algorithms as these signals have little to offer for diagnostic purposes. However, as the TCP/IP and disk usage signals were all monotonically increasing, it was decided to transform these particular signals into a time series could be used for modeling. This was done because many of the tested exploits send large numbers of packets or consume disk resources. Then the TCP/IP and disk signals should then show large deviations in the respective monitored telemetry during these intrusion activities. Because of this, the TCP/IP and disk usage signals are valuable sources of telemetry that should not be discarded.

In Figure 5-15, several examples of raw TCP/IP signals are shown before any additional signal processing is performed.

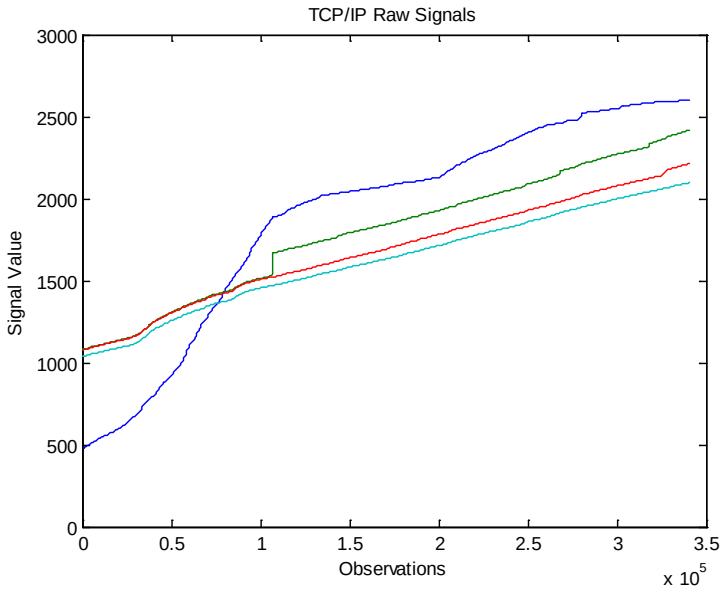


Figure 5-15: Raw TCP/IP Signals

All TCP/IP and disk usage signals that were collected exhibit the same basic shapes and trends seen in the previous figure. The reason monotonically increasing signals are unsuitable for the modeling methods described earlier is twofold. First, signals with a highly varying range will only be correlated with other signals with highly varying ranges. This means that the grouping algorithms would only place these monotonic signals into a group, though these signals may have commonality with other telemetry classes. Second, it is impossible to train a model on these signal types because any values outside of the training range cannot be predicted. When considering anomaly detection, any values outside of the range of the data that the model was trained on will be considered to be in a faulted state, even if the data is unfaulted.

In order to use these signals for modeling, data processing involved two stages. The first stage takes the difference of each of these signals. This step has the effect of lining up each of the step changes in these signals in sequential order. However, many of the values for this transformed signal will have a value of zero, which would be deemed as faulted states during anomaly detection. To resolve this issue, a windowed RMS filter is applied to the transformed signal. The final result is a signal that looks like a typical time series that is suitable for modeling. On the next page, Figures 5-16 and 5-17 show these two data processing steps for an expanded view of the "TW" TCP/IP signal.

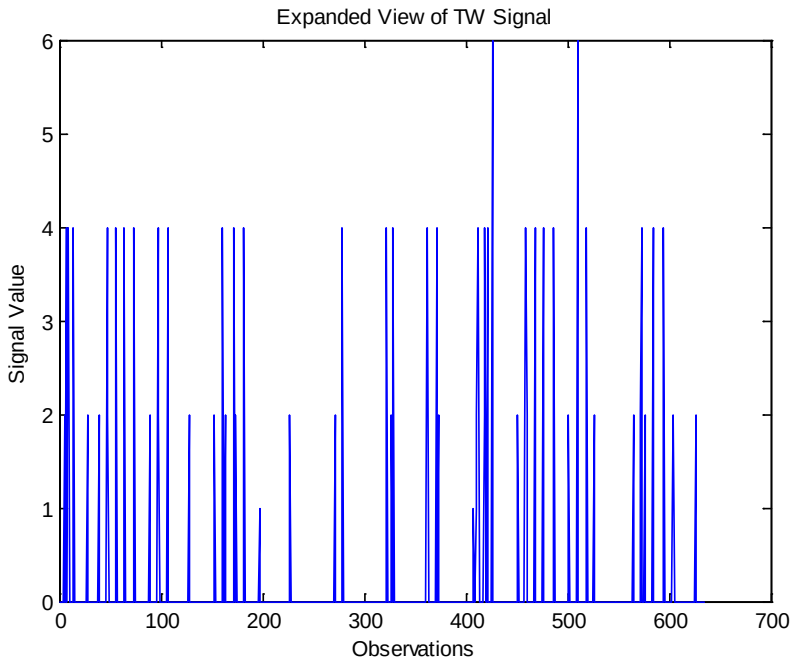


Figure 5-16: Expanded View of TCP/IP Signal "TW" – Difference

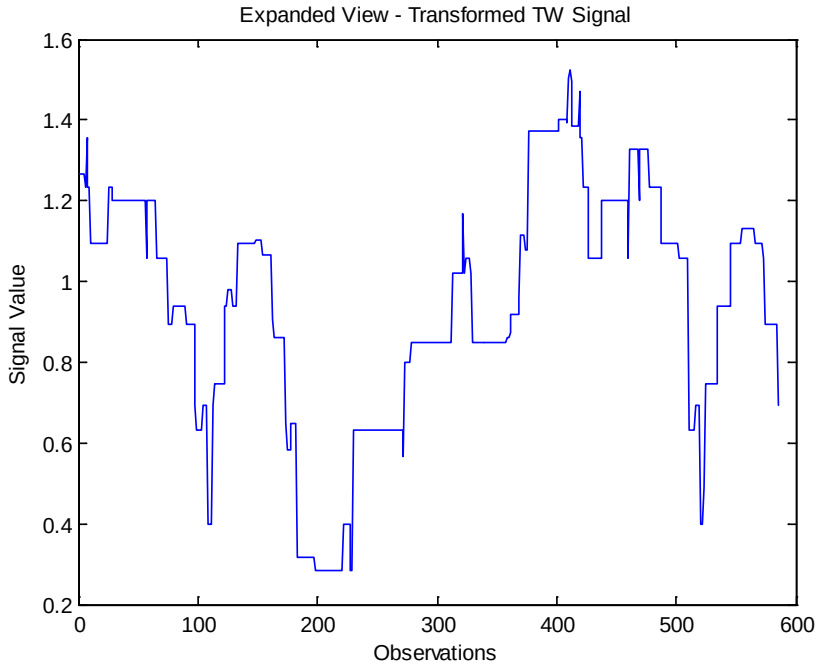


Figure 5-17: Expanded View of TCP/IP Signal "TW" – Windowed RMS

On the previous page, Figure 5-16 shows the TCP/IP after taking the difference of the raw signal. As stated previously, many of the values seen in this figure are zero. This would cause the anomaly detection algorithms to label all of the indices with zero values as being in a faulted state, when in fact this is all normal unfaulted data. In Figure 5-17, a windowed RMS filter is applied to the differenced signal. It is seen after using the filter that the signal now resembles a typical time series. The one item to mention is that in performing these data processing steps is that there was only a loss of 15 data points. This is due to losing one data point when taking the difference of the raw signal and losing points when using the windowed RMS filter that utilized a window size of 14. Other than these minor losses, there is no other loss of information when compared with the original raw signal. Also, other windowed filters were employed, but it was found that the RMS filter provided the desired results in generating the final time series signal. This two stage processing was performed on all TCP/IP and disk usage signals.

Before the selected model results are shown, the last item to be discussed is variable rejection and grouping. As stated earlier, there were 665 total signals. After removing constant valued or near-constant time series, 219 signals remained for consideration. Of these 219, many were TCP/IP and disk usage signals that were processed before variable grouping. It is also noted that some of the memory usage, CPU usage and page swap signals were quantized time series that were discussed in previous chapters. These particular signals were also removed by the grouping algorithms. Last, a consideration for this work was to determine if using small clusters of signals for intrusion detection purposes would be preferred over using hundreds or thousands of signals. Using large numbers of signals to develop models for intrusion detection purposes is a common practice in computer science applications. The idea for this work is that using smaller clusters of well correlated signals would be more beneficial as many signals in computing equipment are redundant and offer the same results when performing intrusion detection. This would then have the effect of reducing overall compute cost for IDS.

For this work, the correlation coefficient grouping algorithms discussed previously extracted four separate groups from the remaining 219 signals discussed in the previous paragraph. The first group contained strongly correlated variables selected from all available telemetry classes. The majority of signals in this group are TCP/IP and disk related. The second group used a combination of memory and CPU resource usage telemetry. The third and fourth groups contained only disk usage and TCP/IP related statistics, respectively. Also, these groups are small in size, ranging from 9 to 21 total signals. Each of the previously listed variable groups is used to develop both AARR and AAMSET models for both considered data sets. All signals for each of the extracted variable groups are contained in the Appendices for reference.

5.3.2 Selected Model Results – June Intrusion Tests

In this first data set, six different intrusion types were tested in succession after collecting normal operational data. For this set, the intrusion classes tested were network/host discovery, DoS, brute force password attack, elevation of privilege and information theft from the victim machine. The software packages used are, respectively, SPARTA, Network Time Protocol (NTP) fuzzer, SSH fuzzer, Samba2 fuzzer, followed by a SSH brute force password attack. Last, Metasploit is used to elevate privilege to root user and then download several files from the victim desktop. The main assumption made in testing these particular intrusions is that an attacker has already gained access to the network. SPARTA is first used to obtain information about all ports, processes and services for each device on the network, followed by the three fuzzers. The fuzzers are then followed by a brute force password attack to gain access. Once access is gained, the guest user privilege is upgraded to root user. For this test, the entire victim desktop was downloaded. This theft included files, pictures and folders. Each of these intrusion activities is followed by a short rest period to allow the system dynamics to recover.

As stated previously, there were four separate groups that were used during model development and intrusion detection activities. These groups used, in order, signals from all telemetry classes, memory/CPU usage signals, disk usage signals and TCP/IP signals. These groups will be termed Models 1 - 4, respectively. For each of these groups, a selected SPRT plot from the developed AAKR and AAMSET models will be shown. For reference in the upcoming figures, the observation ranges for the intrusions presented are listed:

- SPARTA – 245:345
- NTP Fuzzer – 430:550
- SSH Fuzzer – 580:640
- Smb2 Fuzzer – 655:710
- SSH Brute Force/Privilege Escalation – 880:960
- Information Theft – 975:1100

In the upcoming figures, the SPRT plot is shown for a selected signal in each model. Recall from Chapter 3, page 51 the discussion of how to interpret the results. Briefly, the top subplot is the generated anomaly residual. The bottom subplot is the fault hypothesis scores. A 1 is assigned to a faulted state, while a 0 is an unfaulted state. Next, a red 1 is a True Positive, these all occur in the above listed regions for each tested intrusion. This means the SPRT correctly identified the intrusions as anomalous. A blue 0 indicates a True Negative, which means the unfaulted observations were correctly identified as normal behavior. A red 0 is a faulted state incorrectly identified as unfaulted, or a False Negative. A blue 1 is a False Positive, an unfaulted state incorrectly labeled as faulted.

Set 1 Model 1 Results - All telemetry classes:

Model 1 uses 21 variables that were selected from all available and usable telemetry classes. These classes include memory, disk and CPU usage, as well as network related TCP/IP signals. It is noted that for this research that the hardware signals collected had little to offer in the way of intrusion detection. That is, it was not possible to find any of the tested intrusions in any of the hardware signal residuals. Shown next in Figure 5-18 is the correlation coefficient plot that is generated from the training data for this first group.

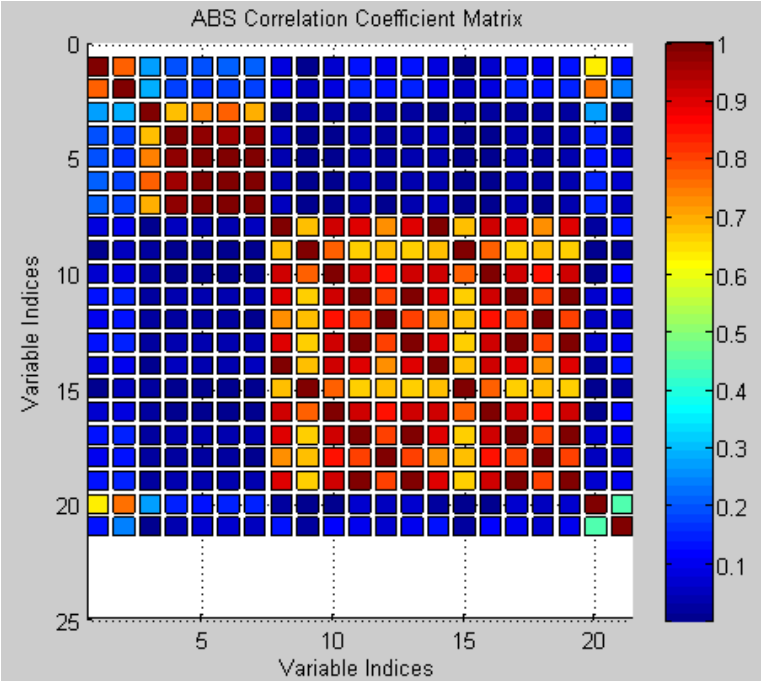


Figure 5-18: Set 1, Model 1 Training Data Correlation Coefficient Plot

In the previous figure, Signals 1-2 are memory usage, Signals 3-7 are TCP/IP, Signals 8-19 are all disk usage and Signals 20-21 are CPU usage related. It is immediately noticed in the plot that only the memory and CPU usage signals have any degree of correlation. Also, the TCP/IP and disk usage signals have nothing in common with any other signals besides TCP/IP or disk usage, respectively. This particular set was used because, for monitoring purposes, a utility might want to utilize a combination of all telemetry sources, regardless of the observed correlations for a particular data set. Later in this chapter, models will be developed using only specific classes of telemetry. This is done to compare if there was any benefit in developing models with specific telemetry classes over a model that contains signals taken from all telemetry sources.

After the models are trained and optimized, the anomaly data discussed previously is run through the developed models. For these results, the residual and SPRT fault hypothesis plot for a TCP/IP signal labeled "InNoECTPkts" is shown for both the AAKR and AAMSET models. This was done to show that both model types can generate the same basic residual and fault hypothesis scores. Shown next in Figure 5-19 is the results generated by using the AAKR model.

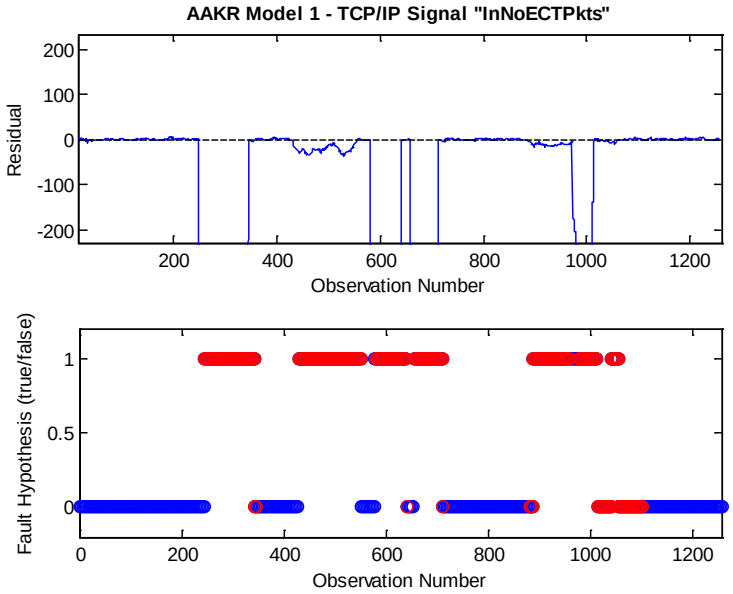


Figure 5-19: Set 1, Model 1: AAKR SPRT Results "InNoECTPkts"

In the previous figure, the magnitude of the large residuals that extend beyond the range of the graph was on the order of -2 x 10^7. These large residuals correspond to the SPARTA, SSH, Smb2 fuzzers, and information theft activities. Compare this to the small magnitude of the NTP fuzzer and brute force attack, both of which were several orders of magnitude smaller. This is important to mention because a simple threshold technique might miss the smaller magnitude intrusions if the threshold was inappropriately chosen. This also shows the ability of the SPRT to correctly identify all of the tested intrusions, even though the resulting residual magnitudes may be quite small. While the above results show that all tested intrusions were detected, there are some false alarms present. First, the red zeros between observations 350 – 850 are false negatives. These are relatively minor when compared with the false negatives after observation 1000. This region was during the information theft attack, the results indicate that while detected, half of the intrusion region was labeled incorrectly. The reason for these false alarms for this attack is that the value of the residuals sometimes falls into regions the model learned as normal. Figure 5-20 show the SPRT results for the same signal using the AAMSET model.

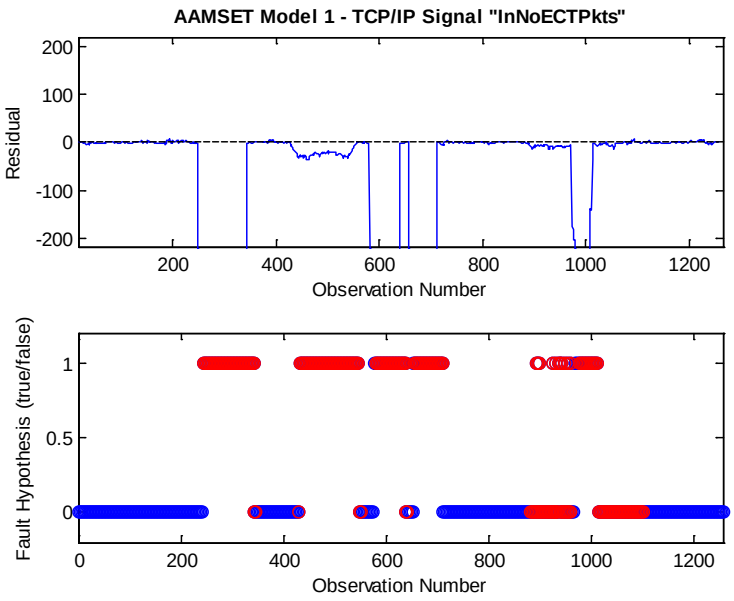


Figure 5-20: Set 1, Model 1: AAMSET SPRT Results "InNoECTPkts"

The AAMSET model results are similar to the AAKR results, though there are a larger number of false negatives for the last two listed intrusions. Recall that these are the password and information theft attacks. These results indicate that the residual generated by the AAMSET for this region had some magnitudes that fell into the range of learned, normal behavior. However, both attacks are detected in spite of the larger number of false negatives. Further model or SPRT optimization could alleviate this unwanted behavior. For this model that uses signals taken from all available telemetry sources, each of the previously listed intrusions was detected. This indicates that a model developed using different telemetry sources can be an effective means to detect a variety of attacks.

Set 1 Model 2 Results – Memory/CPU usage:

The remaining models of this section utilize signals taken from specific telemetry classes. This was done to determine if using subsets of available telemetry to develop models could be just as effective at intrusion detection. This also makes sense for systems that are resource constrained because a smaller set of signals could reduce the overall compute cost of monitoring activities. Model 2 uses a selection of memory and CPU usage measurements for a total of 12 signals. These particular signals were selected for this model because many had some degree of moderate to strong correlations with each other. Shown on the following page in Figure 5-21 is the correlation coefficient plot for this signals of this subset.

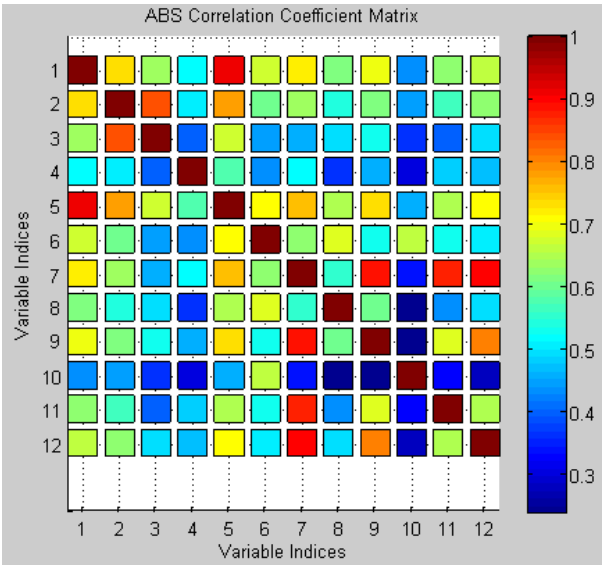


Figure 5-21: Set 1, Model 2 Training Data Correlation Coefficient Plot

In the previous figure, Signals 1-5 are memory usage and Signals 6-12 are CPU usage telemetry selected by cross correlation grouping methods. Many signals show moderate to strong correlations. This makes sense because memory and CPU usage are coupled activities in computing systems. The SPRT results for a memory signal "Min1-load-avg" are shown next in Figure 5-22.

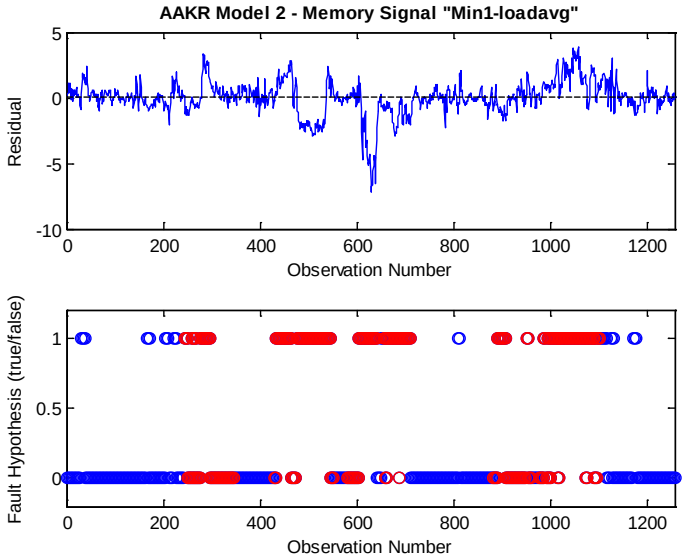


Figure 5-22: Set 1, Model 2 AAKR SPRT Results "Min1-load-avg"

In the previous figure, it is seen that the SSH, Smb2 fuzzer and information theft activities cause the greatest change in the model residual behaviors. The SPARTA intrusion activity also is detected, though there are many false alarms for this attack. The other interesting item to mention about these results is that the information theft activity was more easily detected with this model when compared with the results in the previous section. This particular behavior was seen for all residuals in this data set for both model types. Next, Figure 5-23 shows the AAMSET residuals and SPRT results for this same signal.

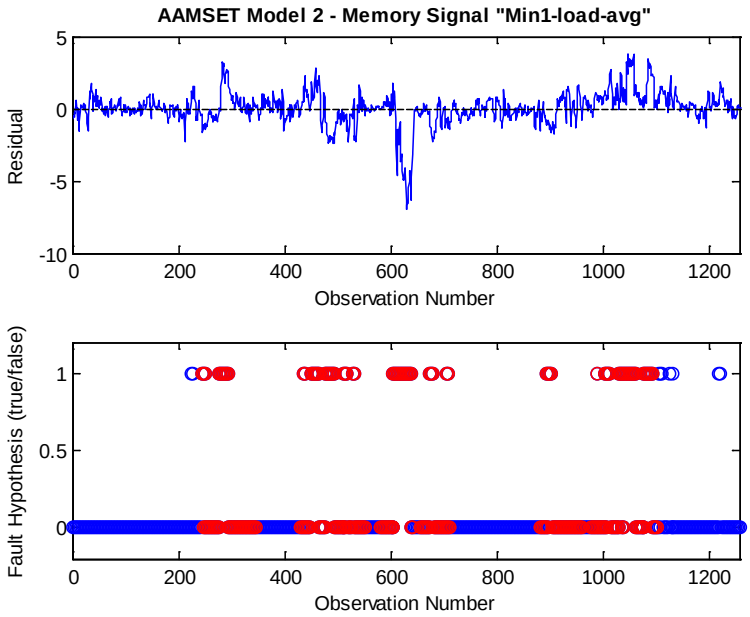


Figure 5-23: Set 1, Model 2 AAMSET SPRT Results "Min1-load-avg"

As in the previous results, the residuals in Figure 5-23 show the same changes for the SSH, Smb2 and theft activities. However, there is a great deal more false alarms seen in these results than displayed for the AAKR model. Also, the SPARTA and brute force attacks are almost completely missed. All signals in this model showed significant delay time to detection. This is seen by the red zeros at the start and end of each listed intrusion range. These results indicate that many of the anomaly residuals in both models had large portions of the signal in the anomaly regions that were similar to learned, normal behavior. This also means that an intruder could possible do damage since there is such a long delay time to detect these attacks. It is noted that all tested intrusions were able to be detected in both models. However, given the large delay times and false alarm rates, it can be concluded that the memory and CPU usage signals would not be ideal candidates to use for intrusion detection purposes.

Set 1 Model 3 Results – Disk usage:

Model 3 contained 12 signals that were all disk usage telemetry. These signals were selected by the grouping algorithms because these only had strong correlations with each other and no other telemetry classes. Next, Figure 5-24 shows the correlation coefficient plot for this data set.

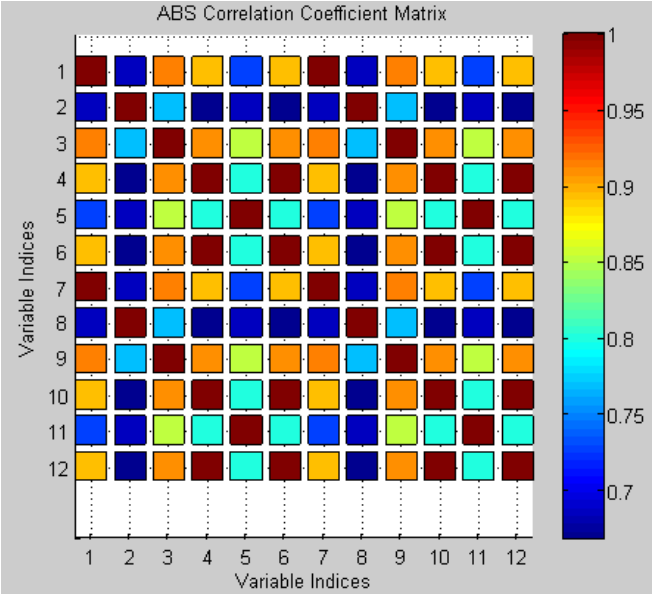


Figure 5-24: Set 1, Model 3 Training Data Correlation Coefficient Plot

In the previous figure, it is seen that all of the selected signals have strong relationships. This makes sense because all of these signals are only disk usage measurements and they all should have some relationships with each other. On the following page, Figures 5-25 and 5-26 show the SPRT results using signal "Disk-sda1-writes-merged" for the AAKR and AAMSET models, respectively. The first interesting point is that in Figure 5-25 all of the intrusions were able to be detected with a minimum of false negatives. This is confirmed by the small number of red zeros. The false negatives are slightly higher between observations 600 – 700, but again the intrusions in this region were detected. Last, there are some false positives in this figure, with the largest number at the end of the test. In contrast, the results seen in Figure 5-26 for the same residual generated using the AAMSET model are almost the exact opposite of the AAKR results. It is noted that the same SPRT missed and false alarm values were used for both models. While all intrusions were detected, the results have a very large and unacceptable amount of false negatives. Operators would not trust the IDS that returned these outputs. These results indicate that the residuals generated using the AAMSET method may be unreliable for intrusion detection.

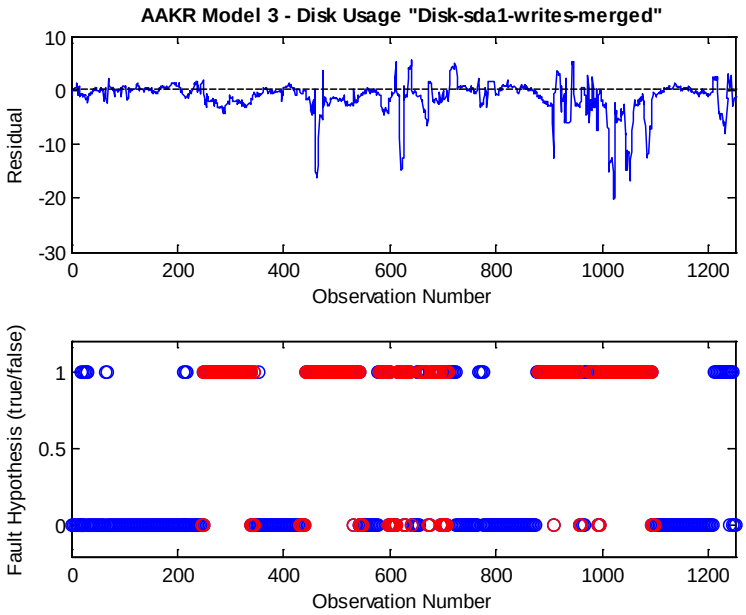


Figure 5-25: Set 1, Model 3 AAKR SPRT Results "Disk-sda1"

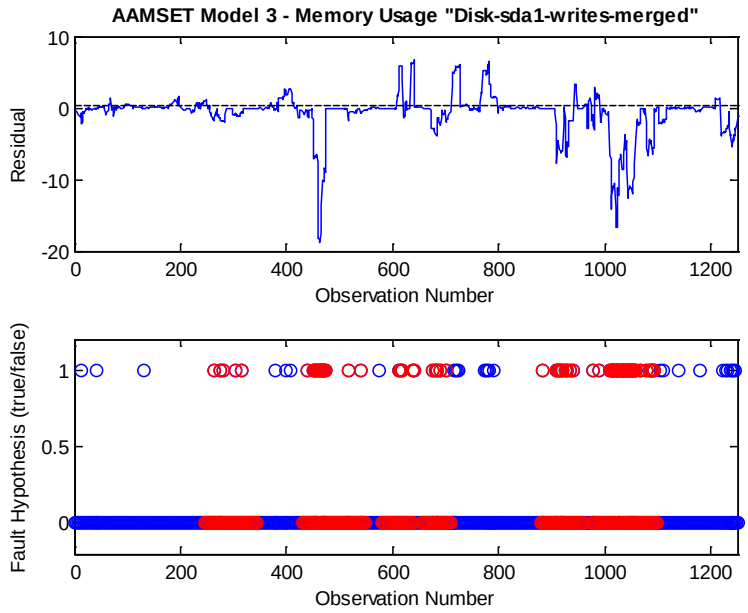


Figure 5-26: Set 1, Model 3 AAMSET SPRT Results "Disk-sda1"

Set 1 Model 4 Results – TCP/IP:

Model 4 used nine signals, which were all TCP/IP measurements. These signals were also selected by the grouping algorithms because these all had strong correlations with each other and no other telemetry classes. It is also noted that many of the TCP/IP signals used for this group were not used in Model 1. Shown next in Figure 5-27 is the correlation coefficient plot for this data.

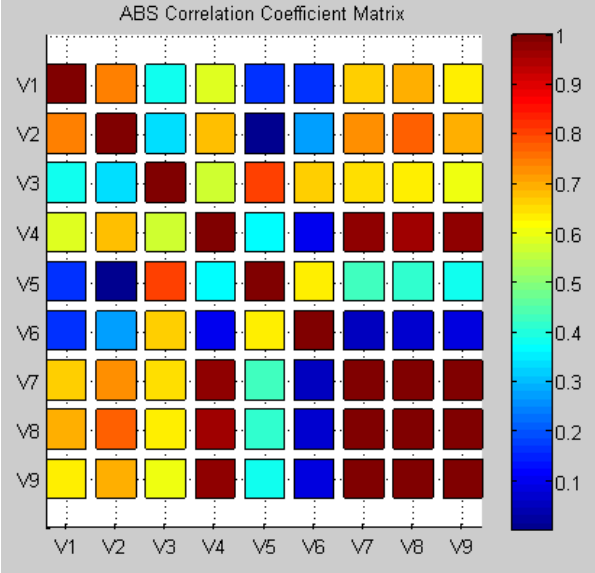


Figure 5-27: Set 1, Model 4 Training Data Correlation Coefficient Plot

In the previous figure it is seen that many of these network related variables have strong correlations. On the following page, Figures 5-28 and 5-29 show the AAKR and AAMSET results for the TCP/IP signal "OutOctets". All tested intrusions were able to be detected in this model. There are a minimal number of total false positives, and a minimal number of false negatives before observation 1000. After this observation, the rate of false negatives increase. These results indicate that much of the residual had magnitudes that were similar to normal behavior. The AAKR results shown are similar to that seen in Model 1. The AAMSET results in Figure 5-29 have more false negatives for the SPARTA, NTP fuzzer, brute force, and information theft attacks. For the NTP fuzzer and password attacks, the residuals are also smaller than those seen in the AAKR model. Due to the smaller size, much of the residuals for these attacks are considered by the SPRT to be in an unfaulted state. Despite the higher rate of false negatives for the AAMSET results, all tested intrusions were able to be detected in this and the AAKR model. Based on these results, we can conclude that a model developed with only TCP/IP telemetry is quite effective for detecting a vide variety of intrusion types.

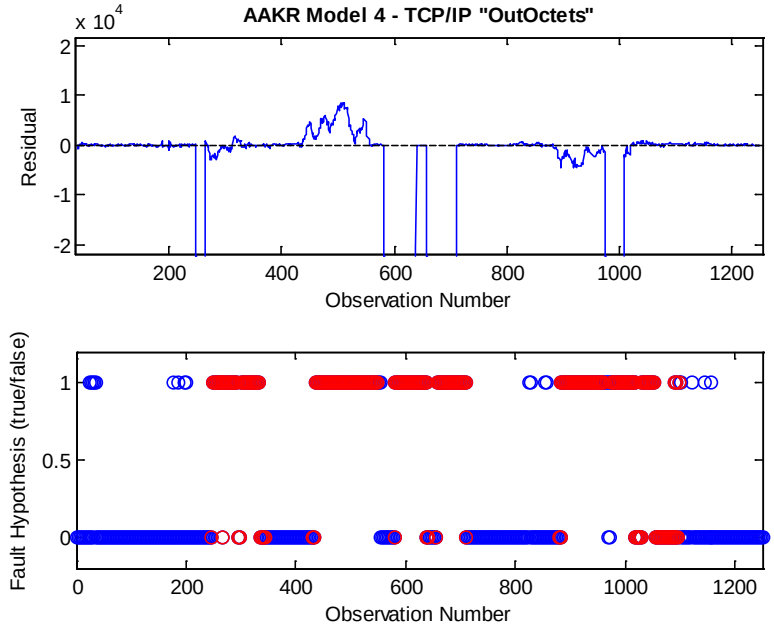


Figure 5-28: Set 1, Model 4 AAKR Results "OutOctets"

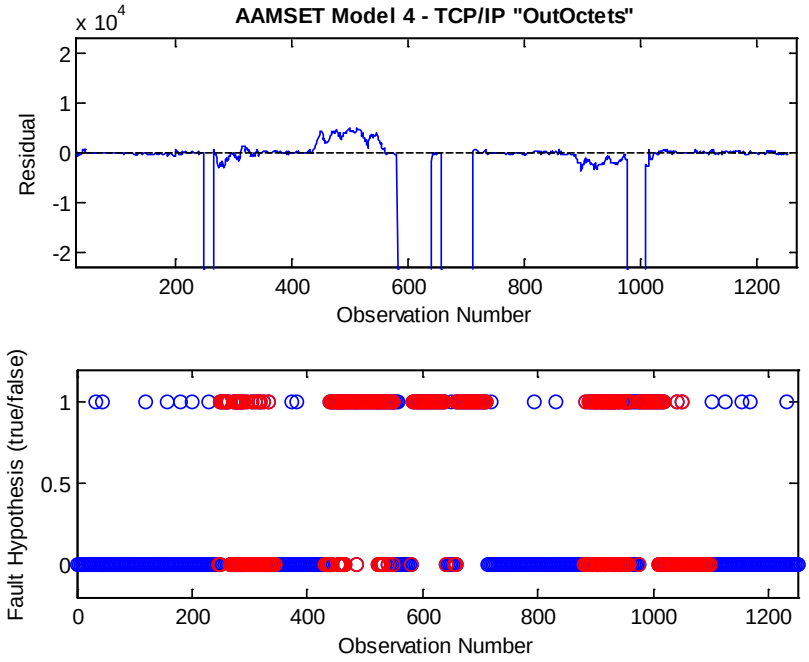


Figure 5-29: Set 1, Model 4 AAMSET Results "OutOctets"

To conclude this section, the results for each of the four models are summarized on the following page in Table 5-9. This summary will list each of the tested intrusions and provide if an attack was detected or not in the each of the models. Here, "Y" indicates that the attack was detected by the model and "N" indicates the attack was not detected by the model.

Table 5-9: Summary of June AAKR/AAMSET Results

Model/Attack	Model 1	Model 2	Model 3	Model 4
AAKR				
SPARTA	Y	N	Y	Y
NTP fuzzer	Y	N	Y	Y
SSH fuzzer	Y	Y	Y	Y
Smb2 fuzzer	Y	N	Y	Y
Password	Y	N	Y	Y
Theft	Y	Y	Y	Y
AAMSET				
SPARTA	Y	N	Y	Y
NTP fuzzer	Y	N	Y	Y
SSH fuzzer	Y	Y	Y	Y
Smb2 fuzzer	Y	N	Y	Y
Password	Y	N	Y	Y
Theft	Y	Y	Y	Y

In the previous table, the reason that Models 1, 3, and 4 detected the tested intrusions is that all of the attacks sufficiently altered the monitored signal dynamics for the SPRT to recognize this behavior as anomalous. These results make sense, as Model 1 used some of the same signals as Models 3 and 4. Thus, if an intrusion is detected in a model developed using signals taken from all telemetry sources, then the intrusion can most likely be detected in models developed on subclasses of these signals.

This is not always the case, as Model 2 had several attacks that were not able to be detected. The main reason for this is that the particular attacks did not alter the monitored signals dynamics for detection to be possible. This means that the residual behavior for the SPARTA, NTP, Smb2, and password attacks looked very similar to regions where no attacks were occurring. The reason that Model 2 was able to detect the SSH fuzzer and information theft attacks is that these two attacks did cause large changes in the signal dynamics. This statement makes sense because all information flowing through the network is done using SSH. The fuzzer and theft attacks then cause the SSH process to use more memory and CPU resources.

5.3.3 Selected Model Results – March Intrusion Tests

In this second data set, several different intrusion types were tested in succession after collecting normal data. The software packages used in this set are, respectively, SSH fuzzer, Zenmap, SPARTA, Dmitry, SSH brute force password, NTP fuzzer, and SSH fuzzer. This is followed by running SPARTA twice at the same time, followed by another application of this package. Running SPARTA twice at the same time was performed to determine if the residual behavior doubled in size or not when compared to just running this software package once. These types of activities can be considered as an attacker gaining access to a network and attempting several different network reconnaissance and DoS attempts. Again, each of these intrusion injection activities is followed by a short rest period to allow the system dynamics to recover.

This data set also uses the same signals that were used to develop Models 1- 4 in the June results shown previously. This was done to validate that the same signals could be used again for intrusion detection purposes. This means that once a suitable signal set is decided on that only those signals need to be used for detection purposes. For each of these groups, a selected signal from both developed AAKR and AAMSET models will be shown. For reference in the upcoming figures, the observation ranges for each of the intrusions presented in this section are listed next:

- SSH Fuzzer – 40:50
- Zenmap – 70:116
- SPARTA – 238:311
- Dmitry – 540:550
- SSH Brute Force – 554:637
- NTP Fuzzer – 836:910
- SSH Fuzzer – 938:948
- SPARTA x 2 – 985:1068
- SPARTA – 1085:1165

In the upcoming figures, the SPRT plot is shown for a selected signal in each model. Recall from Chapter 3, page 51 the discussion of how to interpret the results. Briefly, the top subplot is the generated anomaly residual. The bottom subplot is the fault hypothesis scores. A 1 is assigned to a faulted state, while a 0 is an unfaulted state. Next, a red 1 is a True Positive, these all occur in the above listed regions for each tested intrusion. This means the SPRT correctly identified the intrusions as anomalous. A blue 0 indicates a True Negative, which means the unfaulted observations were correctly identified as normal behavior. A red 0 is a faulted state incorrectly identified as unfaulted, or a False Negative. A blue 1 is a False Positive, an unfaulted state incorrectly labeled as faulted.

Set 2 Model 1 Results - All telemetry classes:

The first results shown for the second data set use the same 21 signals that were used to develop the June results. In fact, for this and all other results shown in this section, the same models that were developed using the June data are also used to develop the results of this second data set. This was done to show that it is possible to use only one or several developed models from a set of normal operational data and then supply these models with new anomaly data at a later date. Shown next in Figure 5-30 is the AAKR SPRT results for the TCP/IP signal labeled "InOctets".

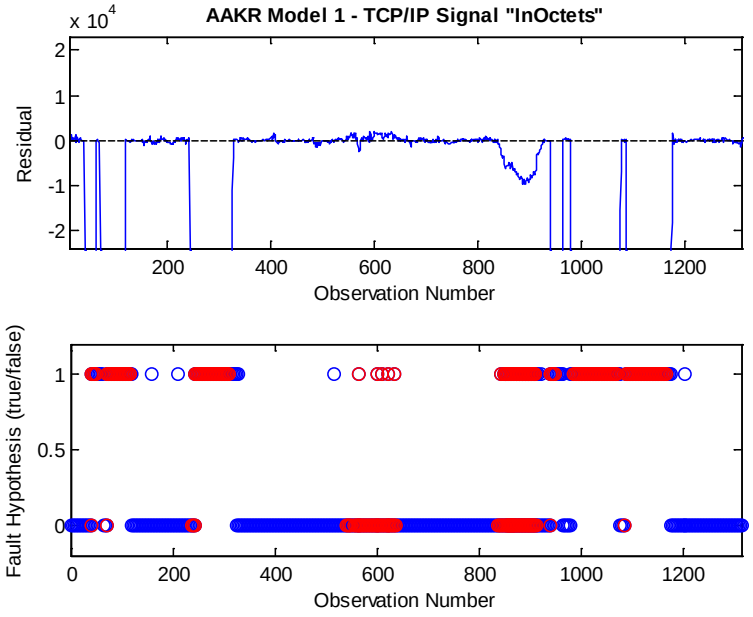


Figure 5-30: Set 2, Model 1 AAKR SPRT Results "InOctets"

In the previous figure, the large residuals that extend off the range of the graph were on the order -4 x 10^7. All of these large residuals correspond to all fuzzers, Zenmap and SPARTA. Of interest is that the network discovery package Dmitry and the brute force password attack in this data set are the smallest in magnitude. Both of these attacks cause very small changes in the residual behavior and Dmitry was completely missed. The password attack and the NTP fuzzer attack that followed were able to be detected, but the rate of false negatives was high. All of the previously listed intrusions tested were able to be detected in this model, many with very small false negative rates. Next, Figure 5-31 shows that AAMSET SPRT results for the same TCP/IP signal.

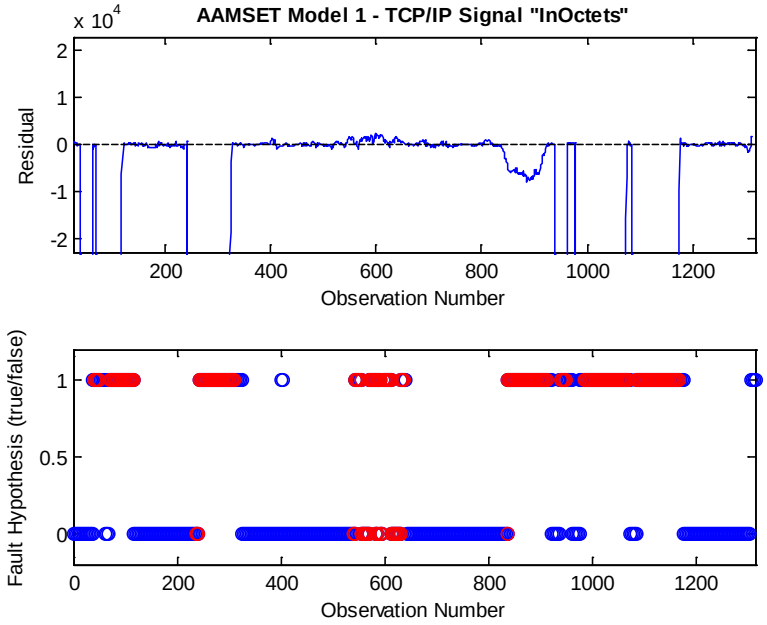


Figure 5-31: Set 2, Model 1 AAMSET SPRT Result "InOctets"

In the previous figure, the main differences in the AAMSET results are that the Dmitry and password attack activities were detected in this model. However, there are some false negatives for both of these attacks seen in both model types. Other than this distinction, the results in Figure 5-31 for all other tested intrusions have minimal or no false negatives present. This means that the time delay for detection was very small for these results. The disk signals used in this and the AAKR model had similar results as seen in the previous two figures. The remaining signals in this model used memory and CPU usage telemetry, all of which had a very large amount of false negatives. However, as all tested intrusions were detected in both the AAKR and AAMSET models, we can conclude that models developed with signals taken from all telemetry classes are an effective means to detect a wide variety of intrusion types.

Set 2 Model 2 Results – Memory/CPU Usage:

Next, the results for Model 2 are shown. Recall that this model only uses twelve memory and CPU usage signals that have strong correlations. Also, it was previously shown that these signals had little to offer for intrusion detection purposes. This particular model is also the same model that was used to develop the June data set results for Model 2; the only difference is the new intrusion data used. On the following page, Figures 5-32 and 5-33 provide the AAKR and AAMSET SPRT results for a memory usage signal labeled "Min-5-load-avg".

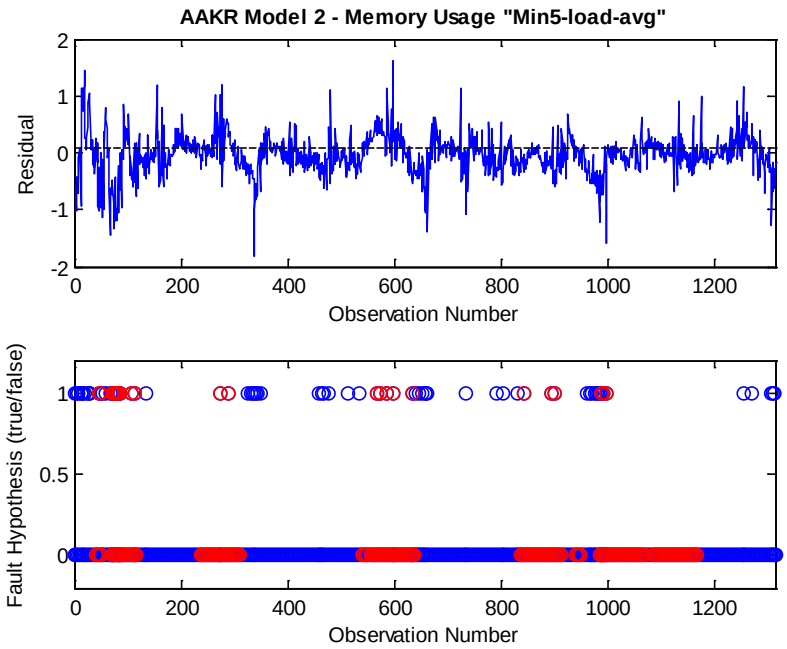


Figure 5-32: Set 2, Model 2 AAKR SPRT Results "Min-5-load-avg"

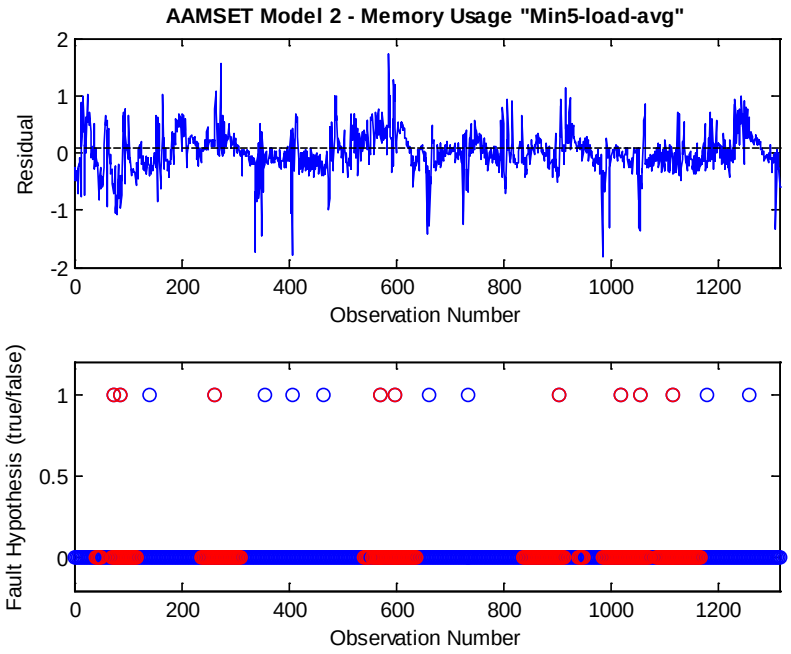


Figure 5-33: Set 2, Model 2 AAMSET SPRT Results "Min-5-load-avg"

The first obvious item of interest seen in both of the previous figures is that none of the intrusions tested induced any clear change in the residual behavior. In the AAKR results, the only intrusion that had a large number of true positives was the Zenmap attack. However, the residuals are quite indistinguishable from the rest of the residual. All other intrusions tested for this model were not detected. In Figure 5-33, the AAMSET results are even worse, with only a minimal number of true positives. If this model was the only one used for intrusion detection purposes, then many attacks would not be detected and an attacker could perform many activities unnoticed. It is noted that all of the memory signal residuals show the same type of detection results. These results are similar to what was seen for the same model of the June data set. This again shows that the memory and CPU usage signals are not useful for general intrusion detection purposes.

Set 2 Model 3 Results – Disk Usage:

Next, the results for the second data set using the disk usage signals are shown. Recall that this model uses twelve strongly correlated disk usage signals that were selected by the grouping algorithms. Shown next in Figure 5-34 is the AAKR SPRT results for a disk signal labeled "Disk-sda-sectors-written".

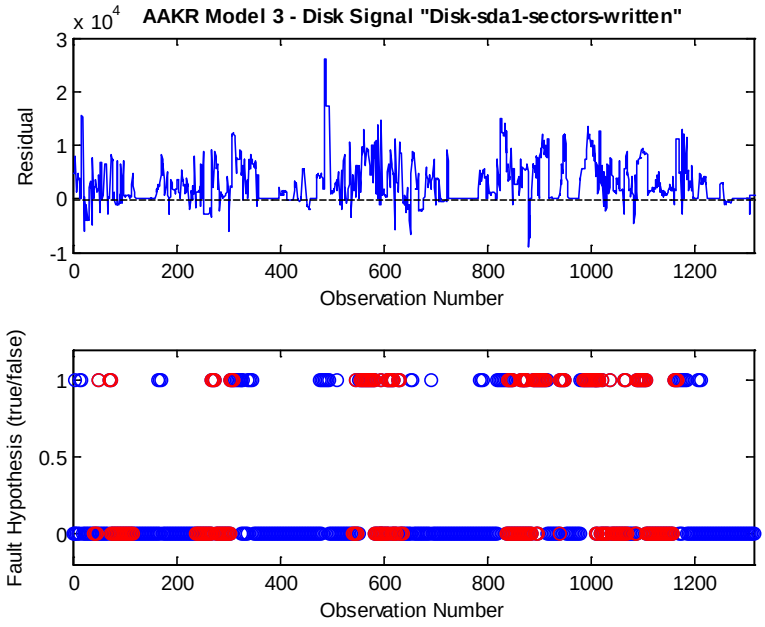


Figure 5-34: Set 2, Model 3 AAKR SPRT Results "Disk-sda-sectors-written"

The first thing to mention is that many of the alarms shown in the previous figure correspond to the actual attack regions. Also, the residuals show clear indications of change during many of the attack regions. For this signal, the SPARTA and brute force password attacks cause the greatest change in residual behavior and thus are easily detected. The remaining fuzzers and Zenmap do not show very large changes during these events. This means that many of the intrusion activities were not recognized by the SPRT as anomalous behavior. These results are different from that seen in the June data set, where the disk usage signals were able to correctly identify most intrusion activities. Next, Figure 5-35 shows the AAMSET SPRT results for the same disk signal.

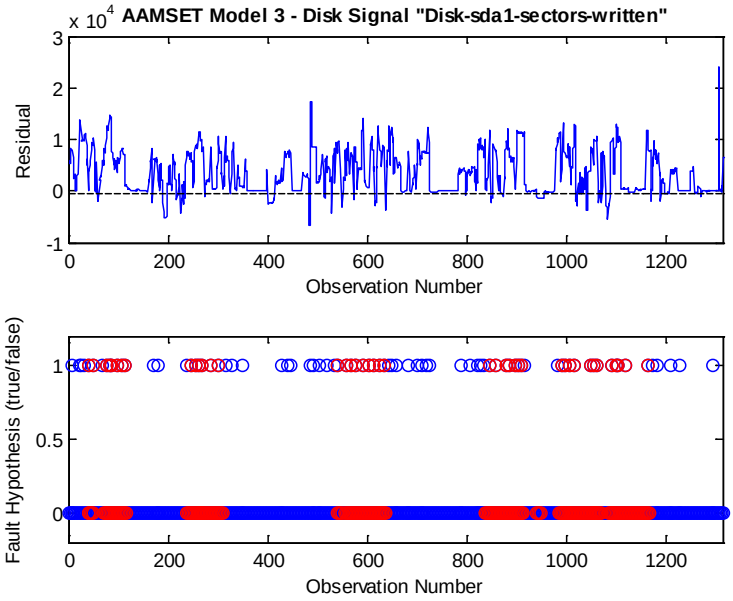


Figure 5-35: Set 2, Model 3 AAMSET SPRT Results "Disk-sda-sectors-written"

In the previous figure, the AAMSET SPRT results are not as conclusive as the AAKR results were. Though there are possible indications in the residuals that anomalous behavior is occurring, the fault hypothesis scores show that many of the listed intrusion activities are missed. However, all of the true positive alarms in the previous figure are in the regions were intrusion activities are taking place. Due to the large number of false negatives, these results indicate that the AAMSET model may need tuning to learn possible new operating conditions. Also, the SPRT would need to be retrained as well. Like the AAKR results, the AAMSET results are in contrast to what was seen in the respective model of the June data set. Recall that there were a dramatically lower number of false negatives and false positives during the intrusion regions.

Set 2 Model 4 Results – TCP/IP:

The last results shown for this second data set are for Model 4. Recall that this model used only nine strongly correlated TCP/IP signals selected by the grouping algorithms. These results also use the same model that was used to develop the Model 4 results for the June data set. This was done to show that models can be developed for monitoring purposes and do not need to be updated to be useful for intrusion detection purposes. In Figure 5-36, the AAKR SPRT results are shown for the TCP/IP signal labeled "TCPHPHits".

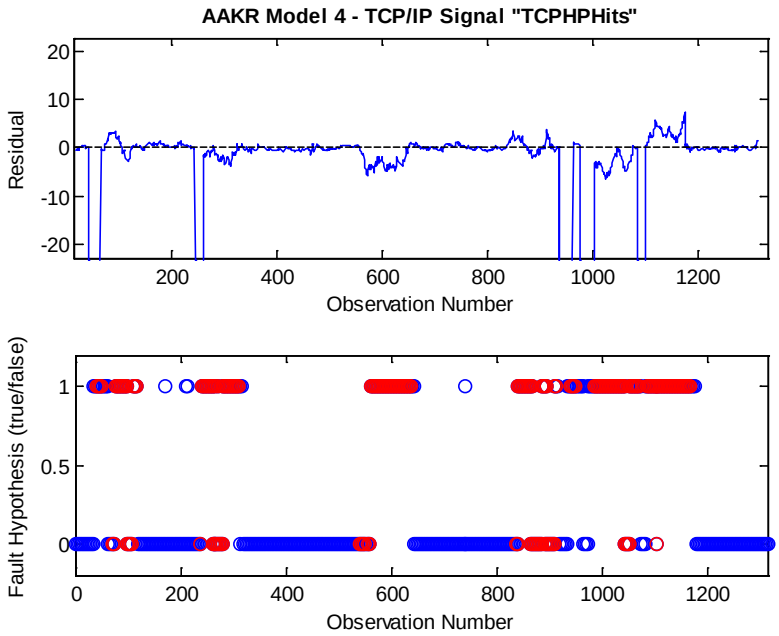


Figure 5-36: Set 2, Model 4 AAKR SPRT Results "TCPHPHits"

The large residuals that extend off the graph in the previous figure are on the order of -4 x 10^7. Again, these large residuals respectively correspond to SPARTA, Zenmap, SSH fuzzer, and two applications of SPARTA. Though there are some false negatives for some of the attacks, all listed intrusions were detected by this model. The intrusion with the largest number of false negatives is the NTP fuzzer. The main reason is portions of the residual during this attack were of similar magnitude as normal behavior. However, there were still enough true positives in succession for this attack that it was detected. These results show the overall effectiveness in using TCP/IP telemetry for intrusion detection. The last result to be shown is in Figure 5-37, which is the AAMSET SPRT result for the same TCP/IP signal.

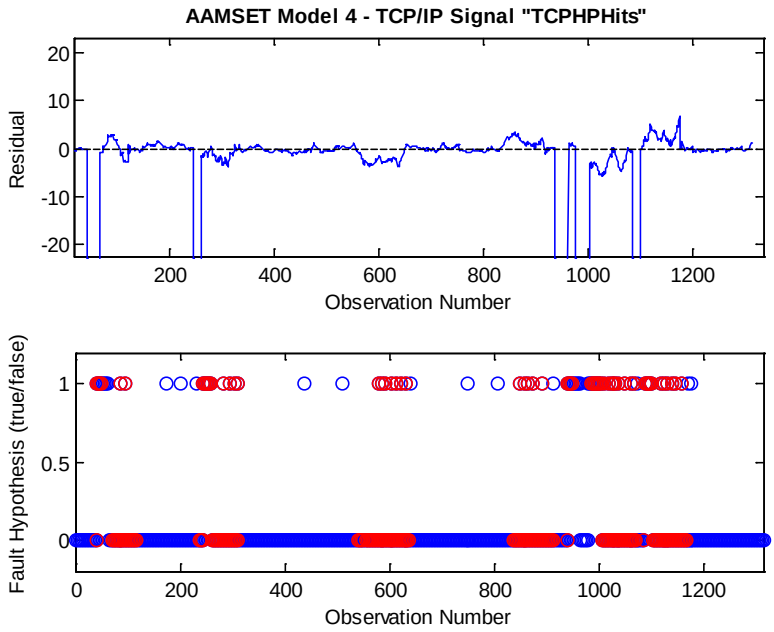


Figure 5-37: Set 2, Model 4 AAMSET SPRT Results "TCPHPHits"

In contrast to the residuals shown in Figure 5-36, the results in Figure 5-37 are different in the ranges for many of the intrusions tested. For example, all of the residuals that extend beyond the range of the graph are the same as those shown in Figure 5-36. However, the residual behavior for these intrusions has much less effect on the system dynamics. This is evidenced by the large number of false negatives; or red zeros; during each of the previously listed ranges for each attack. Though there are a large number of false negatives in this particular signal, all intrusions tested were detected by this model. Based on the results shown in the previous figure, the AAMSET and SPRT parameters may need to be retuned.

As in Table 5.9, the results for the March data set are summarized in Table 5.10 on the following page. Recall that a "Y" means that the tested intrusion was able to be detected in the model. This indicates that the attacks sufficiently altered the monitored signal dynamics for the SPRT to classify these alterations as anomalous. Next, "N" means that the attack did not cause sufficient alterations in the monitored signal dynamics for detection to be possible. This means that the residual behavior during each attack with an "N" classification had behaviors that were indistinguishable from normal behavior. Also, for these results an "N" classification could also indicate that there were true positives during the attack regions, but the number of false negatives was very large. This would indicate that the model or SPRT parameters should be retuned in an attempt to alleviate the large number of false negatives.

Table 5-10: Summary of March AAKR/AAMSET Results

Model/Attack	Model 1	Model 2	Model 3	Model 4
AAKR				
SSH fuzzer	Y	N	N	Y
Zenmap	Y	Y	N	Y
SPARTA	Y	N	Y	Y
Dmitry	Y	N	Y	Y
Password	Y	N	Y	Y
NTP fuzzer	Y	N	Y	Y
SSH fuzzer	Y	N	Y	Y
SPARTAx2	Y	N	Y	Y
SPARTA	Y	N	Y	Y
AAMSET				
SSH fuzzer	Y	N	N	Y
Zenmap	Y	N	N	Y
SPARTA	Y	N	N	Y
Dmitry	Y	N	N	Y
Password	Y	N	N	Y
NTP fuzzer	Y	N	N	Y
SSH fuzzer	Y	N	N	Y
SPARTAx2	Y	N	N	Y
SPARTA	Y	N	N	Y

In the previous table, it is first seen that Models 1 and 4 could detect all of the tested intrusions. This was because the attacks caused significant alterations in the monitored telemetry. In contrast to the previous results seen in Table 5.9, Models 2 and 3 missed many of the tested intrusions. This was first because many of the attack activities did not alter the signal dynamics. Next, many of the attacks did give true positives for both models during the attack regions. However, the number of false negatives during the same regions was excessive. It was then decided to classify the attack as not detected if there were a large number of false negatives observed. The main difference in the March data set results is that all of these results were generated using the same models used to develop the June results. This was done to determine if models that had been previously trained on similar normal data could be used at a later date with new anomaly data. Because all intrusions tested were detected in Models 1 and 4, we can conclude that using previously developed models with new anomaly data could still be useful for intrusion detection purposes. It is noted that if there has been long time periods between these two activities that the model and SPRT parameters should be retuned, if possible. This will ensure that new operating conditions that may have arisen since the development of the models will be included and might reduce the observed rate of false negatives.

5.3.4 Summary of All Intrusion Testing Activities

In this last section, a summary of all intrusion testing results performed in this project are provided. Recall that it was stated at the start of Section 5.3 that there were a large amount of tests and data sets collected. For brevity's sake, only the model results for the June and March data were shown. This was because many of the tests ran the same listed intrusions, but with several options changed for each of the software packages or modules contained in Kali and Metasploit. Also, all of the previously listed intrusions for the June and March data sets, as well as all CVE and EDB exploit codes, were tested in succession. This was done to determine if running the same exploit type several times in a row could be detected. However, there were many intrusion software packages and exploit codes tested that were successfully implemented but did not show any changes in the observed system dynamics. This summary will then provide a listing of all exploit types or classes tested and which ones could or could not be detected with the methods presented in this dissertation.

Stated earlier in Chapter 4, the intrusion testing first utilized the well-known Kali Linux OS. Kali contains several popular penetration testing software packages that can be used to test a variety of systems, software, and applications. Each of these software packages is related to a specific class of intrusion. Their purpose, implementation, and the reasons why they could or could not be detected are discussed next.

The first type of exploit class is known as network reconnaissance. This exploit class attempts to discover information about all components connected to a network. This information can include computers, servers, network hubs/switches, open/closed ports, and identifiable running processes. The software packages listed and tested under this section are called *Dmitry, dnmap-server, ike-scan, maltego, recon-ng, Nmap, Zenmap, p0f, netdiscover, and SPARTA*. Of these, *Dmitry, Nmap, Zenmap, pOf, netdiscover, and SPARTA* caused an alteration in the observed dynamics and were able to be detected using the described methods in this dissertation. The reason these packages could be detected is that they all send a large amount of packets over the network in their attempt to find information. The number of packets sent is usually always larger than what has been seen by the network and learned by the models as normal behavior. The other packages in this list either caused no alterations in the observed signal dynamics or the software package was unable to be initialized. An example of the latter is the *maltego* software package, which would not start due to some issue in the current version of Kali Linux.

The next type of exploit in Kali is termed vulnerability analysis. These packages attempt to determine similar information as the network discovery packages, but also supply the user with additional information about running

processes. For example, a particular version of an application or process may be outdated and vulnerable to exploitation. The packages here are called *Golismero, lynis, Nikto, Nmap, openvas, and unix-privesc-check*. Out of these packages, the last two were not able to be detected as they caused no alterations in the signal dynamics. The first four caused large changes in the observed signal dynamics due to an increase in the network activity and were easily detected.

Next, Kali contains several packages that are related to sniffing and spoofing. These packages attempt to either examine packets flowing across the network for vital information or spoof the identity of a user. These packages included *bdfproxy, driftnet, ettercap, hamster, netsniff, responder, and Wireshark.* Of these packages, only *Wireshark* caused any changes in the signal dynamics because this package must connect to a network to read packet information. Also, Wireshark periodically sends queries to various port and processes. This type of behavior consumes system resources that are larger than what is learned by the models as normal operating behavior. The remaining packages would not initialize, were not related to Linux systems, or did not show any changes in the signal dynamics. Examples of these were *ettercap*, *netsniff,* and *bdfproxy*. The first is only applicable for Windows systems, the second would not initialize, and the last did not cause any changes in the signal dynamics.

Kali also contains several other testing packages that are related to web application analysis, database assessment, offline password cracking, and wireless attacks. Because the SCADA test bed did not utilize web applications, databases or wireless capabilities, these were not applicable to our intrusion testing and none of the tools were employed. Finally, because the password cracking software had to be performed offline, these packages were not used.

The next major penetration testing tool that was employed was Metasploit. This tool contains hundreds of exploits that are related to many different OS, hardware, and software. All of these are implemented through the Armitage GUI. The GUI loads modules for each exploit contained in Metasploit to make testing easier. Each of these modules has several configurable user options. It is noted that many of these exploits are not related to Linux based systems or to any hardware and software utilized in the servers. For example, many exploits were only related to Windows based systems. These and other unsuited exploits are numerous and were not used during this research. For the Linux related exploits, there were several different exploit classes not discussed previously that were able to be successfully detected. These include port binding, Dos, password attack, and privilege escalation. Each of these is discussed next.

There are several modules that allow the attacker to undertake spoofing activities. This exploit class binds ports from the attacker to the target machine so that information can be rerouted or backdoors installed for later access. For this research, this was carried out by utilizing various port binding modules. The modules can bind to specific communications ports such as TCP or utilize flaws in specific running applications to bind other types of ports. In this work, there were six different port binding modules related to TCP/IP, User Datagram Protocol (UDP), executable payloads, and Java. All of these modules work by connecting to an open port on the server and injecting a payload to establish a permanent link. A payload is a set of code that contains specific intrusions for the computer system to perform and often take advantage of a known vulnerability. The reason these port binding modules work is because first a connection must be made over the network and then the payload must be injected. These activities cause changes in most network telemetry and are thus detected.

The next exploit class is termed DoS. This class attempts to disrupt the system by flooding the network with an excessive number of packets so that all system resources are consumed. Other DoS variants attempt to break certain applications or processes by supplying them with nonsensical inputs. This is termed fuzzing. The end result for both is that the system is made unavailable to users or to perform critical tasks such as control actions. For this research, the modules for the TCP SYN flood, NTP, Smb2, and SSH fuzzers were all detected. The flood attack was able to be detected because over the course of 10 minute intervals, billions of packets were sent across the network. This far exceeds the number ever sent by the test bed. All the fuzzers were also able to be detected because they sent large amounts of nonsensical inputs to the respective processes. These inputs must be processed and thus increase the system resources used. The remaining fuzzer modules in Metasploit were not applicable to Linux or the servers did not utilize a specific software package, such as web hosting applications.

Next, the brute force password attack exploit class is discussed. This type of attack is also known as a dictionary attack and attempts to gain access to a system by trying many different usernames and passwords. The idea here is that users often choose poor passwords or leave default passwords installed, so with a well-selected attack dictionary access can often be made. Metasploit contained only one module for a select process on the servers, which was termed brute force SSH. The other password attack modules were either for Windows systems or for different software and applications. For this research, several hundred different usernames and passwords were generated to test if the actions could be detected. It was shown earlier that this activity was detected in several signals. This module was detected because SSH is used to send all information over the network. The actions of trying several hundred different usernames and passwords caused the SSH process to use more resources.

In keeping with the previous paragraph, the last exploit class discussed is termed privilege elevation. This class of attack attempts to circumvent security protocols in order to elevate a guest user to one with administrator privileges. Usually after a successful brute force login, the attacker may be assigned as a guest user and the password must be reentered to gain root access. Metasploit contains privilege elevation software called Meterpreter. This is a payload type exploit that runs over the network and attempts to elevate a user to higher levels of access. Meterpreter also starts an Application Programming Interface (API) that allows the attacker to easily view files and turn off running processes. Because this software uses network resources, it was also detected. This was also the only module in Metasploit for this exploit class.

The last exploits that were tested were taken from the CVE and EDB databases. Recall that the CVE database contains information for new and past exploits or vulnerabilities related to many systems. Often there is code available for testing. The EDB database contains user written code for testing of various vulnerabilities. This code is moth often in Python and contains a small payload of shellcode. Shellcode operates by opening a command window and using machine code to try and exploit the system. Many of these codes must also be compiled and uploaded to the target machine once sufficient access has been gained. For this research, several of these codes related to Linux vulnerabilities were tested. The exploit classes include spoofing, privilege elevation, and egg hunting, among others. While many of the codes taken from these two databases were successful in exploiting the system, none could be detected with the developed methods.

The two main reasons for this are small size of the exploit code and rapid implementation time. Much of the codes taken from these databases had sizes ranging from 1 – 4 kB. In contrast, the size of the packets sent over the test bed network ranged from 5 kB – 125 MB. The number of packets sent over the network was on average 25/s. Then because the exploit code must be uploaded, the small size means that the impact on system resources will be masked by those of normal test bed activities. Next, all of the successful exploit codes took on average 1 second to complete. The server collects data once every 15 seconds and the rapid execution of these codes will not be recorded in monitored telemetry. Finally, because the payload is usually very small, there will be no perceived impact on the system resources. Because the developed methods detect intrusions activities by a shift in mean or variance of the current observation, the uploading and execution of these exploit tools will not cause these effects. The only way to detect the actions of these codes would be if the executed code caused a DoS or similar attack. To detect the activities of these codes, other means must be utilized. This can include auditing of log files or development of specifications that would prohibit uploaded codes from executing. On the following page, Table 5.11 summarized these results.

Table 5-11: Summary of All Exploit Testing Activities

Attack Class (software)	Detected? (Y/N)	Why/Why Not?
Kali Linux		
Recon – Dmitry, Nmap, Zenmap, pOf, netdiscover, SPARTA	Y	Increase of network traffic caused large signal changes
Recon – dnmap, ike-scan, Maltego, recon-ng	N	Not detected because no change in signal dynamics or SW issues
Vulnerability – Golismero, lynis, Nikto, Nmap	Y	Were detected because network traffic was increased
Vulnerability – openvas, unix-privesc-check	N	Not detected because no change in signal dynamics or SW issues
Sniffing – Wireshark	Y	Software increased network traffic
Sniffing – netsniff, bdfproxy	N	Not detected because no change in signal dynamics or SW issues
Metasploit		
Spoofing – Port binding	Y	Detected because module activity increases resource usage
DoS – SYN flood, NTP, Smb2, SSH fuzzers	Y	Detected because attacks caused related processes to consume system resources
DoS – other modules	N	No change in signal dynamics or not applicable to simulation
Password – SSH brute	Y	Detected because of repeated login attempts
Password - Linux related	N	Not detected, no change in signal dynamics or not applicable to simulation
Privilege Elevation/Theft - Meterpreter	Y	Detected because SW & theft consume resources
CVE/EDB codes	N	Not detected due to small payload & execution time

6 CONCLUSIONS

In recent decades, there has been a large shift worldwide in using technology in almost every aspect of our lives. This trend has been termed in many industries as the "Internet of Things" (IoT), where the Internet and cloud-based activities are seeing more and more use for control, monitoring and data storage applications. This trend is also being adopted for the nuclear industry, as the shift to digital I&C is being considered by the Nuclear Regulatory Commission (NRC) and many reactor vendors. In fact, many of the new reactor designs from companies like Westinghouse are implementing digital I&C. These new designs still must be approved by the NRC before they are fully implemented in the US. Because of this new interest in digital I&C, there is also an immediate need to protect these new types of reactor systems and their networks from those with malicious intent. This need can be accomplished first and foremost by plant network isolation, which has seen reasonable success in the US nuclear industry. However, for these newer digital controlled plants, simple network isolation can no longer be relied upon for overall plant security. Any other industry that decides to shift to digital and cloud-based control, or has already, needs to ensure overall system security against intrusion activities.

To address this need, there have been many types of IDS that have been developed over the years. Given the diversity of IDS, these systems can be generalized into three basic types: knowledge, behavior and specification-based. The knowledge-based system is certainly the most numerous and widely used type that uses signatures or features of known attacks for detection purposes. While this type is easy to implement and has a low rate of false/missed alarms, it will always miss the so-called zero-day attacks. Recall that a zero-day attack is a new, unknown intrusion. Because the knowledge-based system has never seen a zero-day intrusion event, it will classify this new behavior as normal. In contrast, the behavior-based system can detect known and zero-day attacks. This type uses data-driven modeling techniques to learn normal system behavior. Once trained, any future monitored telemetry that deviates from this learned behavior will be labeled as an intrusion event. The main weakness seen for this type is that here is often a high rate of false/missed alarms during the monitoring process. This can lead to poor and unreliable detection decisions for an improperly trained model and anomaly detection algorithm. The last type of detection system attempts to use the behavior-based technique along with specification of rules for acceptable system behavior. The rules for the specification-based system are developed and implemented by a security expert, which is one of the main weaknesses seen for this type. Because a human is developing all the rules and specifications for normal system behavior, this can lead to poorly defined rules and errors. The main strength of this type of detection system, which is not often used, is that there is a low rate of false and missed alarms during the monitoring process.

This research focused on the development of a behavior-based detection system that uses two empirical-based modeling techniques along with a well known anomaly detection algorithm. The empirical-based models employed in the dissertation are the AAKR and AAMSET, both of which can be termed nonlinear, nonparametric regression techniques that are used to generate predictions based on model inputs. Both of these modeling techniques have seen success for anomaly and sensor degradation detection in industrial systems. These modeling methods were also chosen as the focus of this research because they have not seen use in industry for computer security prognostic purposes. These modeling methods are combined with a mean and variance based SPRT for intrusion detection. The SPRT was chosen for this work because it is known to have the lowest mathematically possible false and missed alarm probabilities and is also employed to tackle the issue of high false and missed alarm rates.

Based on the results shown, these techniques were able to detect a wide range of intrusion activities in the simulated SCADA test bed using telemetry that included memory, disk, CPU usage, and network related statistics. The exploit tools employed used Kali Linux, Metasploit, and codes taken from two well-known exploit data bases. With a properly tuned SPRT, many of these intrusion events were detected with minimal false and/or missed alarms. In many of the models, there was also minimal delay time to detection. Using these methods, all but the fastest intrusions that were tested were able to be detected. Some exploits have a very rapid implementation time or small payload. Because of this, these activities are not reflected in the system dynamics and cannot be detected using these methods. For these, a knowledge or specification-based system would be required for successful detection. The reason that many of the exploits tested were detected by one or more models is that the exploits caused a large amount of system resources to be consumed. This then caused large changes in much of the monitored telemetry, which was recognized by the SPRT as anomalous.

Next, this dissertation also presented a newly developed VGM termed ACFgroup. This VGM utilizes properties of the ACFs for a set of input data to arrive at a final group or groups of variables that are suitable for use in most data-driven modeling methods. This was proven by using seven different process data sets that were taken from NPPs, fossil plants, and several accelerated aging experiments. The ACFgroup outputs were compared with two other VGM that utilize correlation criteria for variable grouping. In all cases but one, ACFgroup was able to extract groups of variables that had an overall lower average model error when using AAKR. Given that the ACFgroup method performance was superior to the other two mentioned algorithms; we can conclude that this new VGM would be a useful addition to the field of signal processing and empirical-based modeling.

Last, this dissertation also presented a Java-based time synchronous averaging method that is used to resample telemetry sources that have vastly different sampling rates. This method was compared against a different method developed by Oracle called ARP. The difference in these methods is that ARP uses a complex regression algorithm to resample telemetry sources, while the Java method uses linear interpolation. Using the provided data sources, the Java based method was able to correctly capture the same signal dynamics seen in the ARP method. Of course, there are slight differences in the outputs of each of these algorithms. The final proof of the efficacy of the Java based method was a comparison of the ACFs for each of the algorithms outputs. The ACFs for the ARP and Java method were remarkably similar, which indicates that the Java method captured the same system dynamics of the resampled signal when compared with the ARP method.

The major and original contributions of this dissertation are summarized:

- Development of behavior-based intrusion detection methods which use AAKR or AAMSET models to learn normal system behavior from various computer system telemetry sources
- Development of a novel VGM known as ACFgroup that selects relevant variables for empirical-based modeling based on statistical properties of the ACF. The algorithm typically returns groups of variables with lower average model prediction errors when compared with other VGM
- Development of time synchronous averaging technique that resample telemetry sources which have uneven sampling rates
- Implemented the developed IDS methods in Java. This allows these methods to be used on any computer architecture that supports Java

7 RECOMMENDATIONS FOR FUTURE WORK

While the behavior-based detection methods shown in this report were successful at detecting many different types of intrusion activities, there are several areas of focus outside of that described in this dissertation that could be implemented to further improve this work.

First, the simulated SCADA test bed is, at the end of the day, a simulation that was based on several assumptions. These assumptions were that many industrial processes and the dynamics of an actual SCADA server can contain periodic components. Also, it was assumed that for a large NPP or related cyber-physical system that contains hundreds or thousands of sensors will show a large amount of network traffic. As stated earlier, these assumptions were made because of the extreme difficulty in obtaining any related telemetry from an actual SCADA system or business-critical network. To improve the validity of the simulation, it would be beneficial to obtain even sanitized telemetry from an actual system. Then, the simulation could be refined to better reflect the dynamics of the signals in a real system.

If actual SCADA system data is unobtainable, then the test by itself can be modified to reflect more processes seen in these systems. First, a small flow loop with sensing and control elements could be constructed and driven by the servers of the test bed. The primary server in the test bed that acts as the SCADA server could then be programmed to poll the sensors, issue commands and collect data. The other servers of the test bed that act as RTUs could then be used to control and monitor the flow loop. If this is not feasible, then a commercial, off the shelf NPP simulator could be purchased and driven by the test bed servers. Using either of these suggestions, intrusion testing could then attack several of the components of this new system to determine which can be detected using the methods presented in this dissertation. Both of these suggestions also have the added benefit of adding additional process telemetry that can then be used in the data-driven models.

Last, it was noted that many exploits tested that had successful implementation were unable to be detected using the described methods. This is because many of these exploits have a rapid execution time and are not reflected in the system dynamics. Here, rapid means that it took only a few seconds for successful exploit implementation and thus these particular exploits are missed during data collection. This issue could be resolved by investigating other audit material sources that provide some indication that these particular types of exploits have occurred. Also, a knowledge-based detection system could be investigated and combined with these behavior-based techniques.

LIST OF REFERENCES

1. https://www.nrc.gov/docs/ML0324/ML032410430.pdf
2. Nicholson, A., Webber, S., Dryer, S., Patel, T. and Janicke, H., "SCADA security in light of Cyber-warfare", Computers & Security, Elsevier, Vol. 31, No. 4, pp.418-436, June 2012.
3. http://www.cnn.com/2016/02/09/europe/germany-train-collision/
4. Anderson, J. P., *Computer Security Threat Monitoring and Surveillance*, Fort Washington, PA, USA, James P. Anderson Company, Apr. 1980.
5. Mitchell, R. and Chen, I., "A Survey of Intrusion Detection Techniques for Cyber Physical Systems", ACM Computing Surveys, Vol. 46, No. 4, March 2014.
6. Ying, L., Yan, Z. and Yang, O., "The design and implementation of host-based intrusion detection system", Proceedings of 3rd International Symposium on Intelligent Information Technology and Security Informatics, Jinggangshan, China, pp.595-598, 2010.
7. Ni, L. and Zheng, H., "An unsupervised intrusion detection method combined clustering with chaos simulated annealing", Proceedings of the International Conference on Machine Learning and Cybernetics, Hong Kong, China, Vol. 6, pp. 3217-3222, 2007.
8. Uppuluri, P. and Sekar, R., "Experience with Specification-Based Intrusion Detection", Recent Advances in Intrusion Detection, Vol. 2212, pp. 172-189, 2001.
9. Gao, W., Morris, T., Reaves, B. and Richey, D., "On SCADA control system command and response injection and intrusion detection", Proceedings of the 5th Annual Anti-Phishing Working Group of eCrime Researchers Summit, Dallas, TX, USA, pp. 1-9, 2010.
10. Gong, W., Mabu, S., Chen, C., Wang, Y. and Hirasawa, K., "Intrusion detection system combining misuse detection and anomaly detection using Genetic Network Programming", Proceedings of the International Conference on Control, Automation and Systems, Fukuoka, Japan, pp. 3463-3467, 2009.
11. Luo, Y., "The research of Bayesian Classifier Algorithm in Intrusion Detection System", Proceedings of the International Conference on E-Business and E-Government, Guangzhou, China, pp. 2174-2178, 2010.
12. Hines, J.W., Yang, D. and Usynin, A., "SCADA System Monitoring and Diagnostics Using Empirical Models", Final Report, Battelle Energy Contract # 00050837, September 2007.
13. Boyer, S., "SCADA – Supervisory Control and Data Acquisition 3rd Edition", ISA, 2004.
14. Cazorla, L., Alcaraz, C. and Lopez, J., "Towards Automatic Critical Infrastructure Protection through Machine Learning", Springer International Publishing, 2013.
15. Ahmed, I., Obermeier, S., Naedele, M. and Richard III, G.G., "SCADA systems: Challenges for forensic investigations", Computer, 2012.

16. Jain, P. and Tripathi, P., "SCADA security: a review and enhancement for DNP3 based systems", CSIT, Issue 1, Vol. 4, pp. 301-308, October, 2013.
17. Han, S., Xie, M., Chen, H. and Ling, Y., "Intrusion Detection in Cyber-Physical Systems: Techniques and Challenges", IEEE Systems Journal, Vol. 8, No. 4, Dec. 2014.
18. Penetration Testing Tools (PTT), 2016, http://www.pen-tests.com/difference-between-threat-vulnerability-and-risk.html
19. Dressler, F., "Self-Organization in Autonomous Sensor and Actuator Networks", New York, NY, USA: Wiley, 2007.
20. Neuman, C., "Challenges in security for cyber-physical systems", Proc. DHS Workshop Future Directions Cyber-Phys. Syst. Security, Newark, NJ, USA, July, 2009.
21. Zhu, B., Joseph, A. and Sastry, S., "A taxonomy of cyber attacks on SCADA systems", IEEE International Conference on Cyber, Physical and Social Computing, Dalian, China, Oct. 2011.
22. Tsang, R., "Cyberthreats, Vulnerabilities and attacks on SCADA Networks", University of California, Berkeley, 2010.
23. Fovino, A., Coletta, A. and Masera, A., "Taxonomy of security solutions for the SCADA sector", 2010.
24. Makhija, M. and Subramanyan, L., "Comparison of protocols used in monitoring: DNP 3.0, IEC 870-5-011 and Modbus", 2003.
25. Beresford, D., January, 2011, http://thesauceofutterpwnage.blogspot.com.
26. Shaor, Z., Zhuge, Q., He, Y., Edwin, H. and Sha, M., "Defending Embedded Systems against Buffer Overflow via Hardware/Software", Proceedings of the 19th Annual Computer Security Applications Conference, 2003.
27. Paukatong, T., "SCADA Security: A New Concerning Issue of an In-house EGAT-SCADA", IEEE/PES Transmission and Distribution Conference & Exhibition, Dalian, China, 2005.
28. Cleveland, D., "IEC TC57 security standards for the power systems info infrastructure: beyond simple encryption", 2005.
29. Kleinmann, A. and Wool, A., "Accurate Modeling of the Siemens S7 SCADA Protocol for Intrusion Detection and Digital Forensics", JDFSL V9N2, 2014.
30. Almalawi, A., Yu, X., Tari, Z., Fahad, A. and Khalil, I., "An unsupervised anomaly based detection approach for integrity attacks on SCADA systems", Computer & Security, Vol. 46, pp. 94-110, 2014.
31. Baker, S., Filipiak, N. and Timlin, K., "In the Dark: Crucial Industries Confront Cyber-attacks", Technical report, McAfee, 2011.
32. ICS-CERT 2013, ICS-CERT Monitor Report between April-June 2013. Department of Homeland Security, June 17, 2013.
33. Bailey, M., Cooke, E., Jahanian, F., Watson, D. and Nazario, J., "The Blaster Worm: Then and Now", IEEE Security and Privacy Magazine, Issue 3, Vol. 4, pp.26-31, July 2005.

34. Kirsher, D., "The Real Story of Stuxnet", Spectrum IEEE, Feb. 26, 2013.
35. Cheminond, M., Durante, L. and Valenzano, A., "Review of Security Issues in Industrial Networks", IEEE Transactions on Industrial Informatics, Vol. 9, No. 1, Feb. 2013.
36. Eandt 2012, http://eandt.theiet.org/magazine/2012/09/new-era-of-safety.cfm
37. Markey, E., "Infection of the Davis Bessie Power Plant by the "Slammer" Worm Computer Virus – Follow-up Questions", 2003, https://www.nrc.gov/docs/ML0329/ML032970134.pdf
38. Abrams, M. and Weiss, J., "Bellingham, Washington, Control System Cyber Security Case Study", NIST, 2007.
39. Parformak, P.W., "Pipeline Cybersecurity: Federal Policy", Congressional Research Service Report for Congress, August 16, 2012.
40. Slay, J. and Miller, M., "Lessons learned from the Maroochy Water breach", Critical Infrastructure Protection, Chapter 6, pp. 73-82, Springer US, 2007.
41. Nicholson, A., Webber, S., Dryer, S., Patel, T. and Janicke, H., "SCADA security in light of Cyber-warfare", Computers & Security, Elsevier, Vol. 31, No. 4, pp.418-436, June 2012.
42. Government Accountability Office, "Critical Infrastructure Protection – Challenges and Efforts in Secure Control Systems (GAO-04-354)", Washington, DC, USA, 2004.
43. Patel, A., Qassim, Q. and Willis, C., "A survey of intrusion detection and prevention systems", Information Management Computer Security, 18(4), pp. 270:277. 2010.
44. Vinchurkar, D. and Reshamwala, A., "A Review of Intrusion Detection System Using Neural Network and Machine Learning Technique", IJESIT, Vol. 1, Issue 2, November 2012.
45. Jyothsna, V. and Prasad, V., "A Review of Anomaly based Intrusion Detection Systems", International Journal of Computer Applications, Vol. 28, No. 7, 2011.
46. Park, K., Lin, Y., Metsis, V., Le, Z. and Makedon, F., "Abnormal human behavioral pattern detection in assisted living environments", Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments, pp. 9.1-9.8, 2010.
47. Carcano, L., "A multidimensional critical state analysis for detecting intrusions in SCADA systems", IEEE Transactions on Industrial Informatics, Vol. 7, No. 2, May 2011.
48. Lauf, A., Peters, R. and Robinson, W., "A distributed intrusion detection system for resource-constrained devices in ad-hoc networks", Ad Hoc Networks, Vol. 8, No. 3, pp. 253-266, 2010.
49. Killourhy, K. and Maxion, R., "Comparing Anomaly Detection Algorithms for Keystroke Dynamics", Carnegie Mellon University, 2010.

50. Shin, S., Kwon, T., Jo, G., Park, Y. and Rhy, H., "An experimental study of hierarchical intrusion detection for wireless industrial sensor networks", IEEE Transactions on Industrial Informatics, Vol. 6, No. 4, Nov. 2010.
51. Garitano, I., Uribeetxeberria, R. and Zurutuza, U., "A Review of SCADA Anomaly Detection", Electronics and Computing Department, Mondragon University, 2011.
52. Tsang, R., "Cyberthreats, Vulnerabilities and attacks on SCADA Networks", University of California, Berkeley, 2010.
53. Zimmer, R., Bhat, B., Mueller, F. and Mohan, S., "Time-based intrusion detection in cyber-physical systems", Proceedings of the 1st International Conference on Cyber-Physical Systems, Sweden, pp.109-118, 2010.
54. Vigna, G. and Kruegel, C., "Host-Based Intrusion Detection", WL041/Bidgoli, June 15, 2005.
55. Snapp, S.R., Brentano, J., Dias, G., Goan, T., Heberlein, T. and Ho, C., "DIDS (distributed intrusion detection system) - Motivation, architecture, and an early prototype", 14th National Computer Security Conference, 1991.
56. Sabri, F.N., Norwawi, N.M. and Seman, K., "Identifying false alarm rates for intrusion detection system with Data Mining", IJCSNS, Vol. 11, 2011.
57. Ho, C. Y., Lin, Y., Lai, Y., Chen, I., Wang, F. and Tai, W., "False Positives and Negatives from Real Traffic with Intrusion Detection/Prevention Systems", International Journal of Future Computer and Communications, Vol. 1, No. 2, August 2012.
58. Han, H., Lu, X. and Ren, L., "Using data mining to discover signatures in network-based intrusion detection", Proceedings of the International Conference on Machine Learning and Cybernetics, Vol. 1, Beijing, China, pp. 13-17, 2002.
59. Ying, L., Yan, Z. and Yang, O., "The design and implementation of host-based intrusion detection system", Proceedings of 3rd International Symposium on Intelligent Information Technology and Security Informatics, Jinggangshan, China, pp.595-598, 2010.
60. Haddadi, F. and Sarram, M., "Wireless intrusion detection system using a lightweight agent", Proceedings of the 2nd International Conference on Computer and Network Technology, Bangkok, Thailand, pp. 84-87, 2010.
61. White, G., Fisch, E. and Pooch, U., "Cooperating security managers: A peer-based intrusion detection system", IEEE Network 10, No. 1, pp.20-23, Feb. 1996.
62. Whitman, M. and Mattord, H., "Principles of Information Security", Course Technology Ptr. 2011.
63. Hinton, G. and Sejnowski, T., "Unsupervised Learning: Foundations of Neural Computation", MIT Press, 1999.
64. Cortes, J. and Vapnik, V., "Support-vector machines", Machine Learning, Vol. 20, Is. 3, pp. 273-297, 1995.

65. Naess, E., "Configurable Middleware-Level Intrusion Detection Support for Embedded Systems", Thesis, Washington State University, May 2004.
66. Oman, P. and Phillips, M., "Intrusion Detection and Event Monitoring in SCADA Networks", Critical Infrastructure Protection, Vol. 253, pp. 3217-3222, 2004.
67. Tsang, S. and Kwong, S., "Multi-Agent Intrusion Detection System in Industrial Network using Ant Colony Clustering Approach and Unsupervised Feature Extraction", IEEE, 2005.
68. Lumer, E. and Faieta, B., "Diversity and adaptation in populations of clustering ants", Third International Conference on Simulation of Adaptive Behavior, MIT Press, pp. 499-508, 1994.
69. Cheung, S., Dutertre, D., Fong, M., Lindqvist, U., Skinner, K. and Valdes, A., "Using Model-based Intrusion Detection for SCADA Networks", SRI International, December 7, 2006.
70. Tsang, P.P. and Smith, S.W., "YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems", Proceedings of the IFIP TC 11 23rd International Information Security Conference, 2008.
71. Dussel, P., Gehl, C., Laskov, P., Buber, J., Stormann, C. and Kastner, J., "Cyber-Critical Infrastructure Protection Using Real-time Payload Anomaly Detection", BMBF, 2010.
72. Common Vulnerabilities and Exposures Database, 2017. https://cve.mitre.org/
73. Sousan, W., Zhu, Q., Gandhi, R. and Mahoney, W., "Smart Grid Tamper Detection Using Learned Event Patterns", Energy Systems, Berlin, 2013.
74. Bolzoni, D. and Zambon, E., "False Positive Reduction through Anomaly Detection", 2006.
75. Kumar, M., Hanumanthappa, M. and Kumar, T., "Intrusion Detection System – False Positive Alert Reduction Technique", ACEEE Int. Journal on Network Security, Vol. 2, No. 3, 2011.
76. Victor, G., Rao, M. and Venkaiah, V., "Intrusion Detection Systems – Analysis and Containment of False Positives Alerts", IJCA, Vol. 5, No. 8, 2010.
77. Gigstad, L., "Reducing false positives in intrusion detection by means of frequent episodes", Thesis, Gjovik University College, 2008.
78. Leitao, P., "Seldom cry wolf: Tuning out false positives on Network Intrusion Detection Systems", SANS, 2004.
79. Guyon, A. and Elisseeff, A., "An Introduction to Variable and Feature Selection", *Journal of Machine Learning Research 3*, ppg. 1157-1182, 2003.
80. Reunanen, J., "Overfitting in making comparisons between variable selection methods", JMLR, 3:1371-1382, 2003.
81. Kohavi, R. and John, G., "Wrappers for feature selection", Artificial Intelligence, 97(1-2):273-324, December 1997.

82. Rivals, I. and Personnaz, L., "MLPs (mono-layer polynomials and multi-layer perceptrons) for non-linear modeling", JMLR, 3:1383-1398, 2003.
83. Nicklet, F. and Prakash, S., "FPGA Implementation of Fully Synchronized Multi-Rate FSM Based Memory Efficient NID System", IJARTET, Vol. 2, No. 5, 2015
84. Kim, H., Hong, H., Kim, H.S. and Kang, S., "A Memory-Efficient Parallel String Matching for Intrusion Detection Systems", IEEE Comm. Letters, Vol. 13, No. 12, pp. 1004-1006, 2009.
85. Gross, K.C. and Vaidyanathan, K., "Improved Energy Monitoring and Prognostics of Servers via High-Accuracy Real-Time Synchronization of Internal Telemetry Signals", Proc. IEEE World Congress in Computer Science, Computer Engineering, and Applied Computing (WorldComp2010), Las Vegas, NV, August 2010.
86. Garvey, D. and Hines, J. W., "The Development of a Process and Equipment Monitoring (PEM) Toolbox and its Application to Sensor Calibration Monitoring", *Quality and Reliability Engineering International*, 22, 1-13, 2007.
87. Gribok, A. V., Hines, J. W., Urmanov, A. and Uhrig, R. E., "Use of Kernel Based Techniques for Sensor Validation in Nuclear Power Plants", *Statistical Data Mining and Knowledge Discovery*, pp. 217-231, Chapman and Hall/CRC Press, 2004.
88. Gross, K. C., Singer, R.M., Wegerich, S. W., Herzog, J. P., VanAlstine, R. and Bockhorst, F. "Application of a Model-based Fault Detection System to Nuclear Plant Signals", Proc. 9th Intnl. Conf. On Intelligent Systems Applications to Power Systems, pp. 66-70, Seoul, Korea (July 6-10, 1997).
89. Singer, R.M., Gross, K. C., Herzog, J. P., King, R.W. and Wegerich, S., "Model-Based Nuclear Power Plant Monitoring and Fault Detection: Theoretical Foundations", Proc. 9th Intnl. Conf. On Intelligent Systems Applications to Power Systems, pp. 60-65, Seoul, Korea (July 6-10, 1997).
90. Wald, A., Sequential Analysis, J. Wiley & Sons, New York, 1947.
91. Gross, K.C., Vaidyanathan, K. and Valiollahzadeh, S., "Advanced Pattern Recogonition for Optimal Bandwidth and Power Utilization for Wireless Intelligent Motes for ioT Applications", World Coumputer Conf., 2016.
92. Ambardar, A., "Analog and Digital Signal Processing", PWS Publishing Company, 1995.
93. Weiner, N., "*Time Series*", M.I.T. Press, Cambridge, 1964.
94. Hines, J. W., Garvey, J., Garvey, D. R. and Siebert, R., "Technical Review of On-Line Monitoring Techniques for Performance Assessment", NUREG/CR-6895, Vol. 3, August, 2008.
95. Paofsky, H. A. and Brier, G. W., "Some applications of statistics to meteorology", College of Earth and Mineral Sciences, Penn. State University, 1968.
96. Exploit Data Base, Accessed 2017-02-27, Copyright 2017, Offensive Security, https://www.exploit-db.com/

97. Kali Linux, Accessed 2017-02-27, Copyright 2017, Kali Linux https://www.kali.org
98. Armitage, Accessed 2017-02-27, Copyright 2017, Offensive Security https://www.offensive-security.com
99. Saxena, A. and Goebel, K., "Turbofan Engine Degradation Simulation Data Set", NASA Ames Prognostics Data Repository (http://ti.arc.nasa.gov/project/prognosti-data-repository), NASA Ames Research Center, Moffett Field, CA, 2008.

APPENDICES

Appendix A: Variables Selected for Models 1 - 4

Signals Selected AAKR/AAMSET Model 1

Signal Number	Signal Name	Software Subset
1.	Min-1-loadavg	Memory
2.	Processes-total	Memory
3.	TCPPrequeued	TCP/IP
4.	TCPPureAcks	TCP/IP
5.	InOctets	TCP/IP
6.	OutOctets	TCP/IP
7.	InNoECTPkts	TCP/IP
8.	Disk-sda-writes-completed	Disk Stats
9.	Disk-sda-writes-merged	Disk Stats
10.	Disk-sda-sectors-written	Disk Stats
11.	Disk-sda-msec-writing	Disk Stats
12.	Disk-sda-msec-IO	Disk Stats
13.	Disk-sda-weighted-msec-IO	Disk Stats
14.	Disk-sda1-writes-completed	Disk Stats
15.	Disk-sda1-writes-merged	Disk Stats
16.	Disk-sda1-sectors-written	Disk Stats
17.	Disk-sda1-msec-writing	Disk Stats
18.	Disk-sda1-msec-IO	Disk Stats
19.	Disk-sda1-weighted-msec-IO	Disk Stats
20.	CPU-all-pct-idle	CPU Usage
21.	CPU-0-pct-user	CPU Usage

Signals Selected for AAKR/AAMSET Model 2

Signal Number	Signal Name	Software Subset
1.	KernelStack	Memory
2.	Min-1-loadavg	Memory
3.	Min-5-loadavg	Memory
4.	Processes-running	Memory
5.	Processes-total	Memory
6.	CPU-all-pct-user	CPU
7.	CPU-all-pct-idle	CPU
8.	CPU-1-pct-user	CPU
9.	CPU-1-pct-idle	CPU
10.	CPU-2-pct-user	CPU
11.	CPU-2-pct-idle	CPU
12.	CPU-3-pct-idle	CPU

Signals Selected for AAKR/AAMSET Model 3

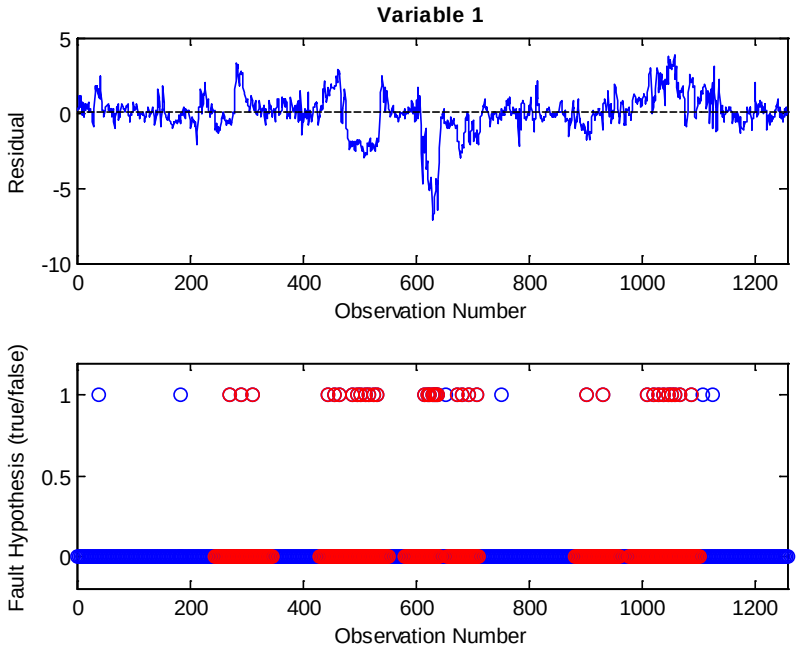
Signal Number	Signal Name	Software Subset
1.	Disk-sda-writes-completed	Disk
2.	Disk-sda-writes-merged	Disk
3.	Disk-sda-sectors-written	Disk
4.	Disk-sda-msec-writing	Disk
5.	Disk-sda-msec-IO	Disk
6.	Disk-sda-weighted-msec-IO	Disk
7.	Disk-sda1-writes-completed	Disk
8.	Disk-sda1-writes-merged	Disk
9.	Disk-sda1-sectors-written	Disk
10.	Disk-sda1-msec-writing	Disk
11.	Disk-sda1-msec-IO	Disk
12.	Disk-sda1-weighted-msec-IO	Disk

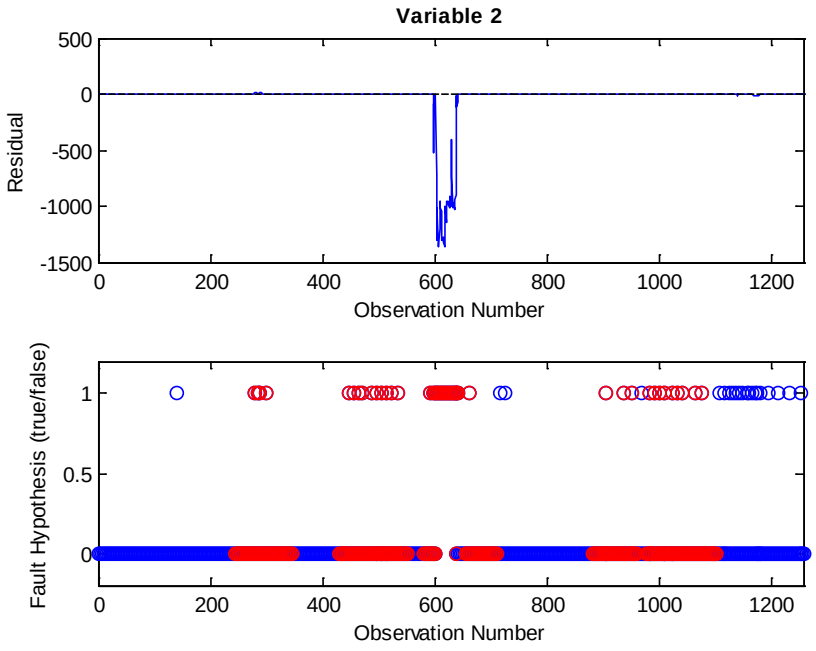
Signals Selected for AAKR/AAMSET Model 4

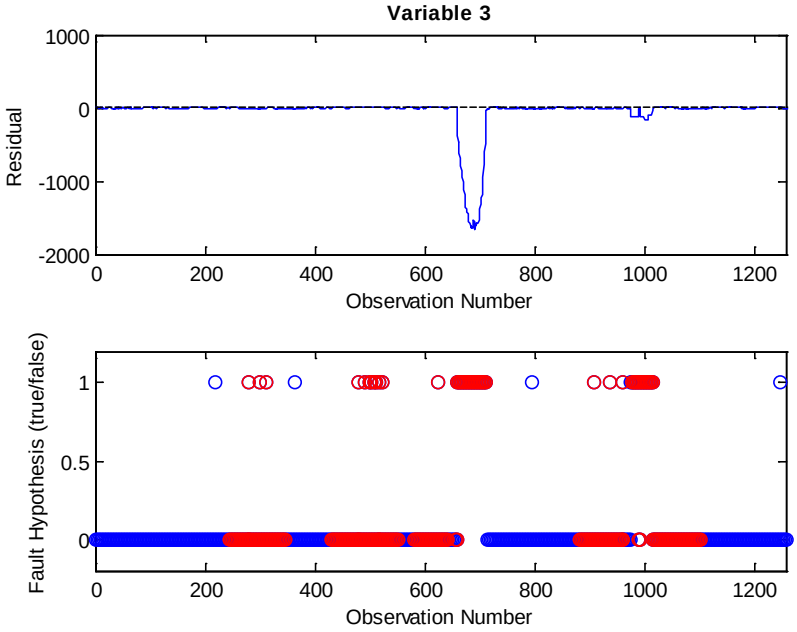
Signal Number	Signal Name	Software Subset
1.	TW	TCP/IP
2.	TCPPrequeued	TCP/IP
3.	TCPHPHits	TCP/IP
4.	TCPPureAcks	TCP/IP
5.	TCPHPAcks	TCP/IP
6.	TCPRcvCoalesce	TCP/IP
7.	InOctets	TCP/IP
8.	OutOctets	TCP/IP
9.	InNoECTPkts	TCP/IP

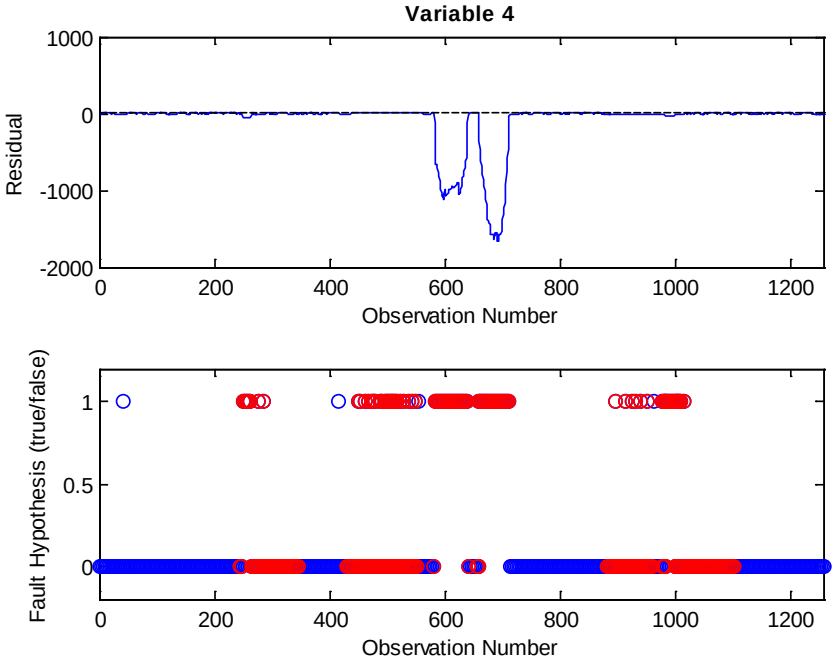
Appendix B: Additional Figures – June Data Set

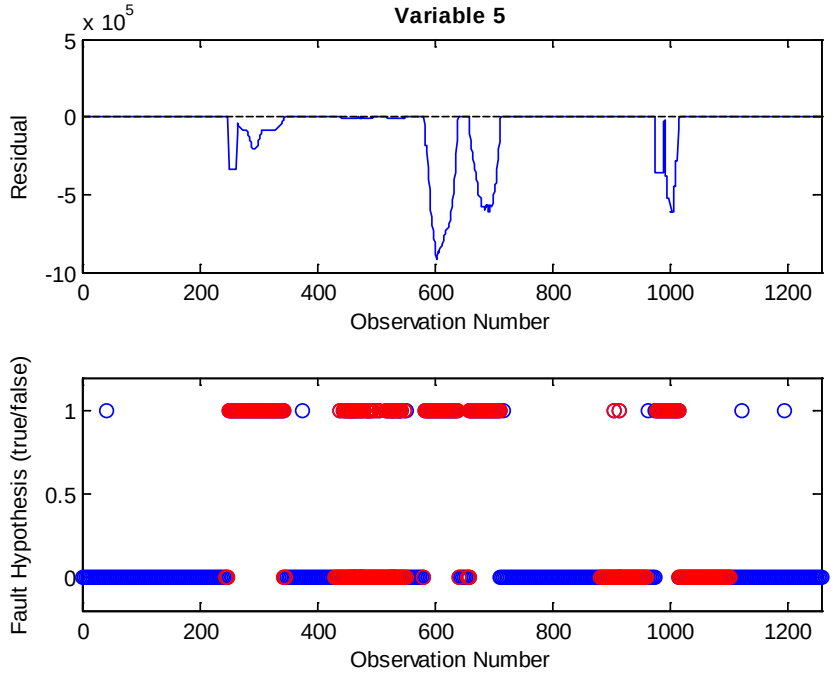
Additional Figures for June Data – Model 1

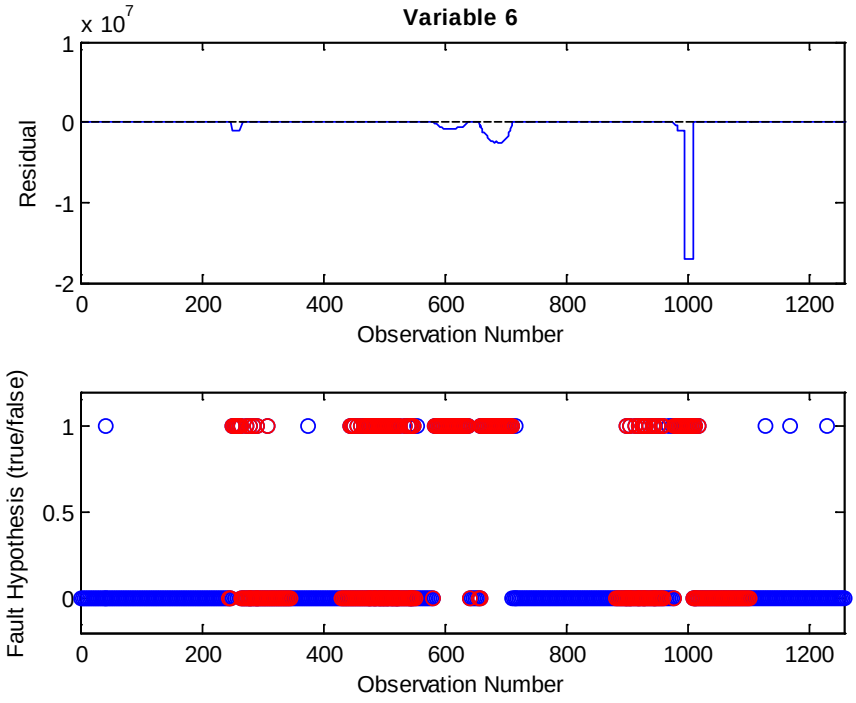


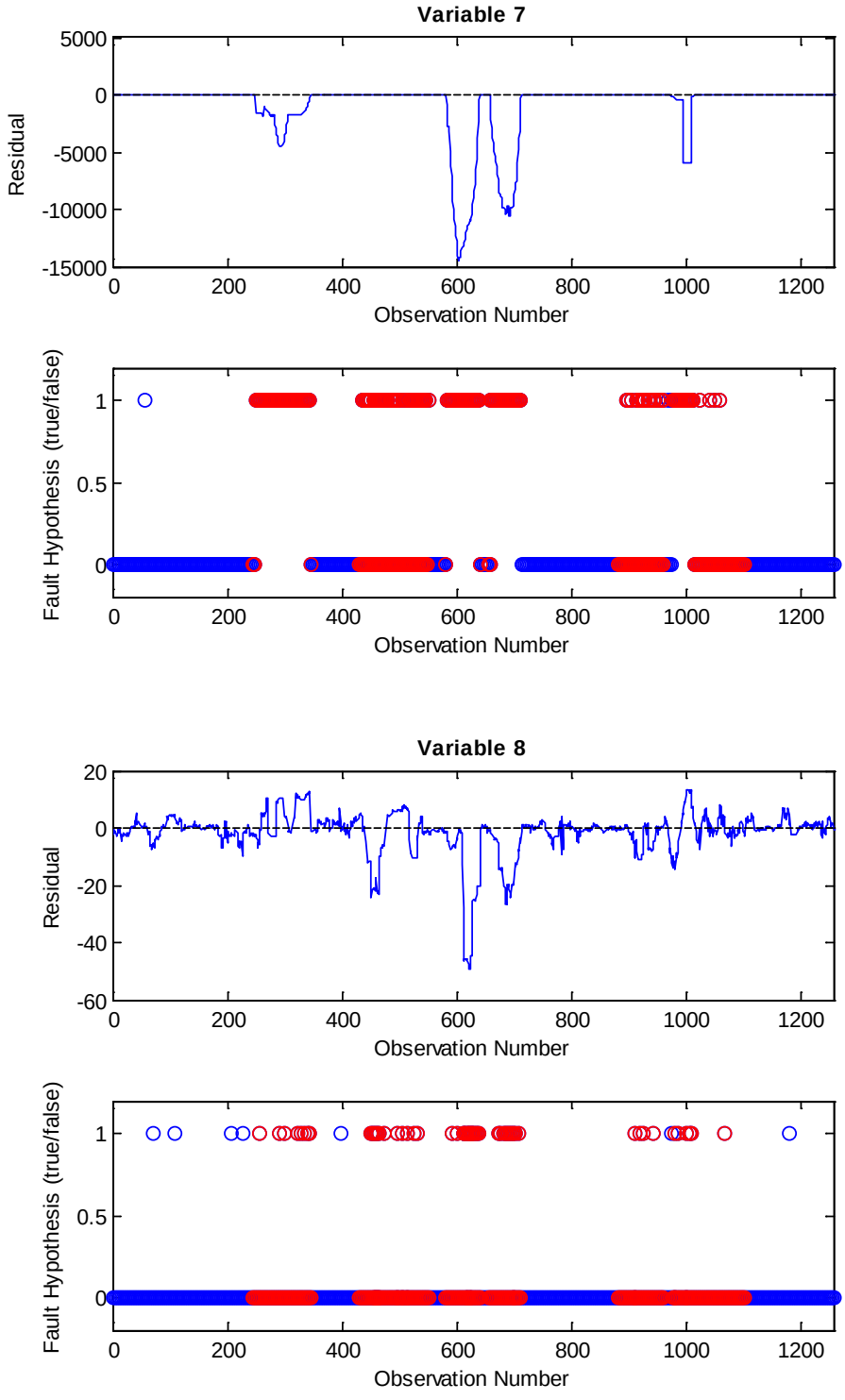


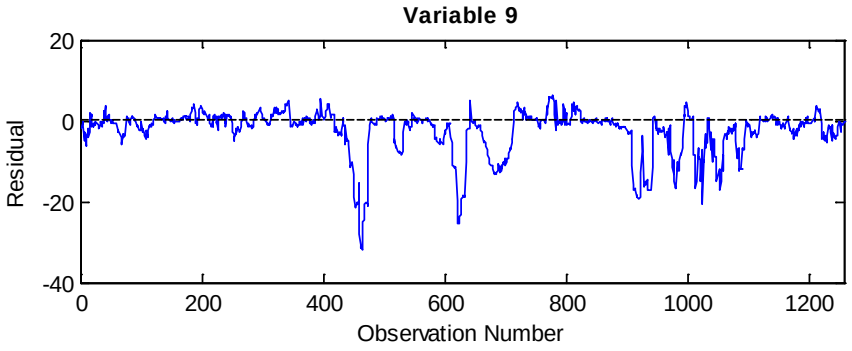

Variable 3
Residual
1000
0
-1000
-2000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

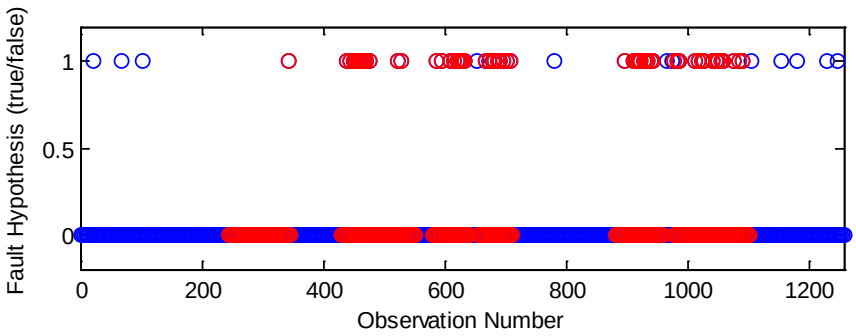

Variable 4
Residual
1000
0
-1000
-2000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

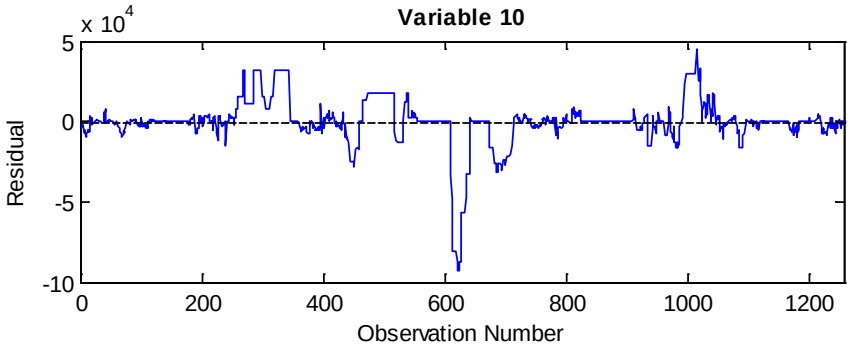

Variable 5
x 10^5
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

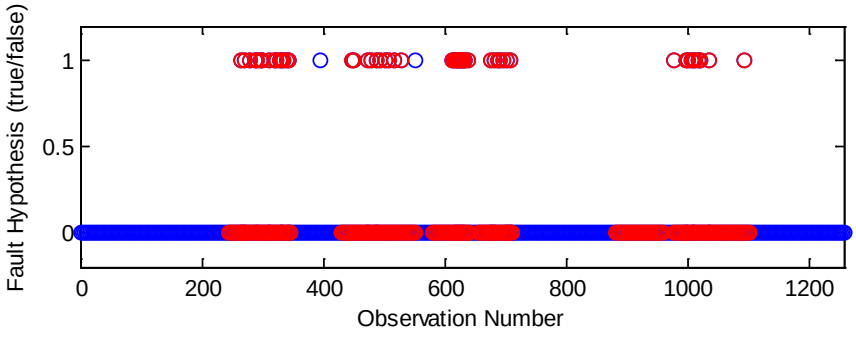

Variable 6
x 10^7
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

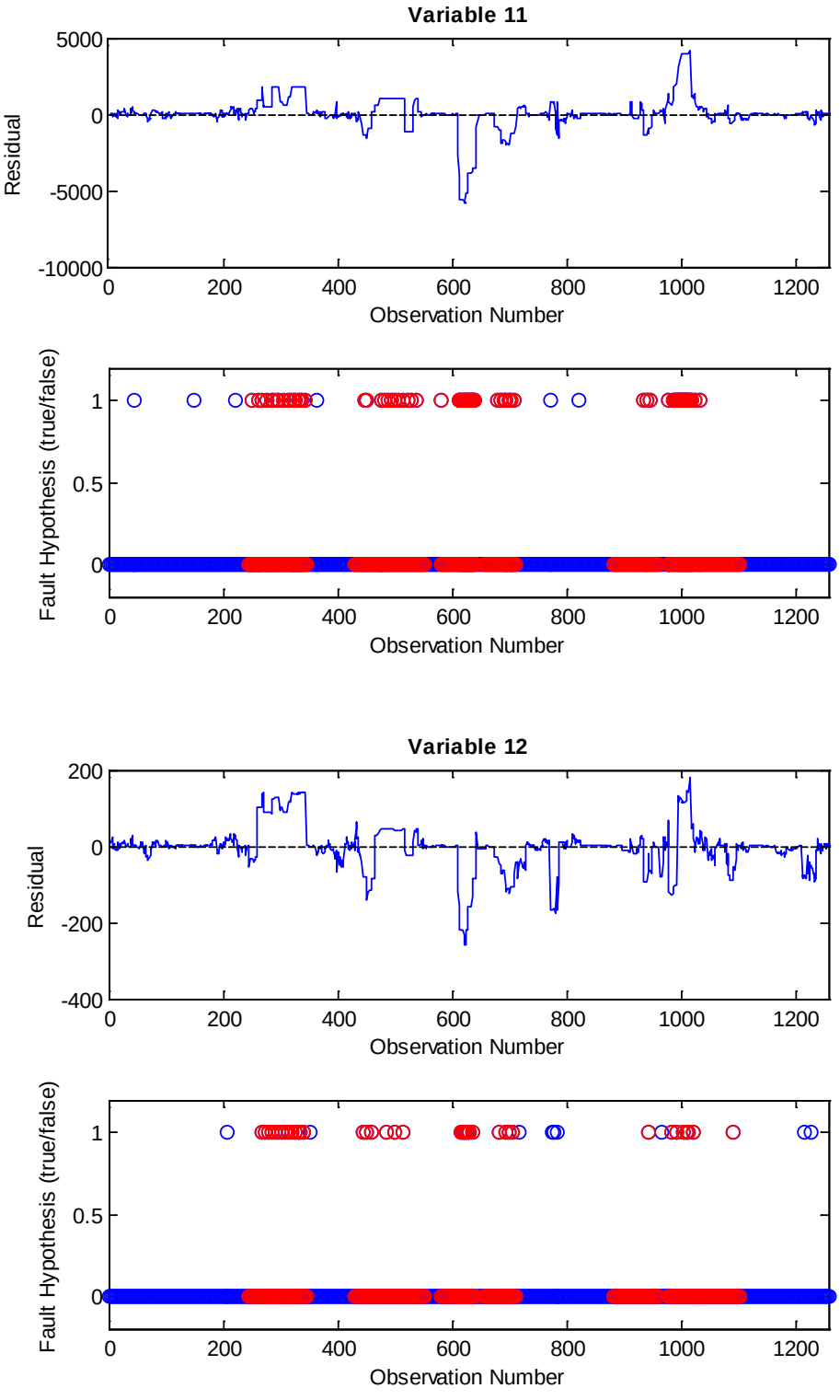

Variable 7
Residual
5000
0
-5000
-10000
-15000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number
Variable 8
Residual
20
0
-20
-40
-60
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

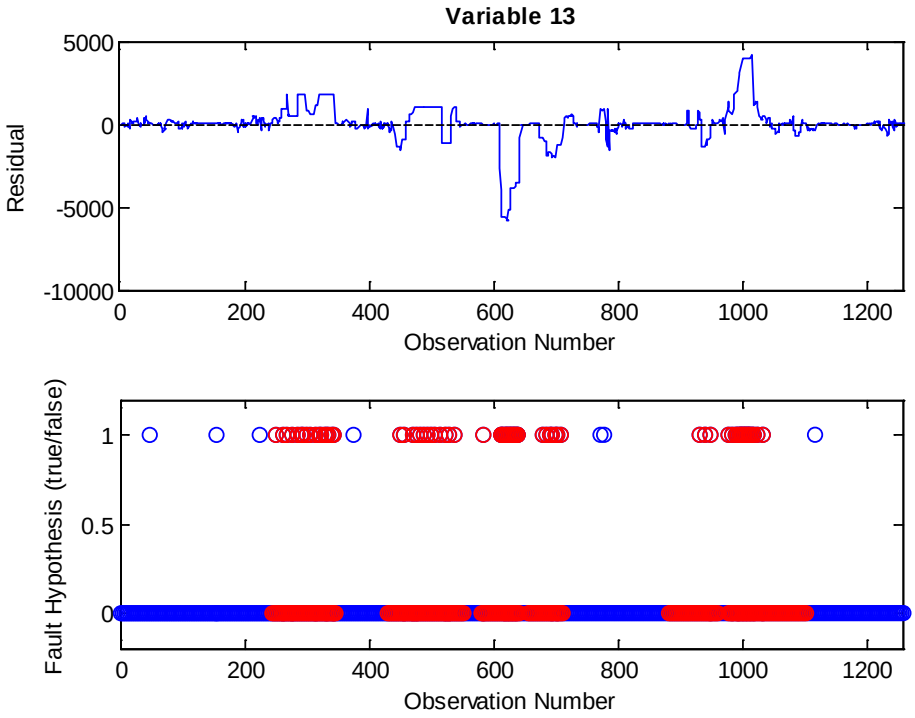

Variable 9
Residual
20
0
-20
-40
0
200
400
600
800
1000
1200
Observation Number

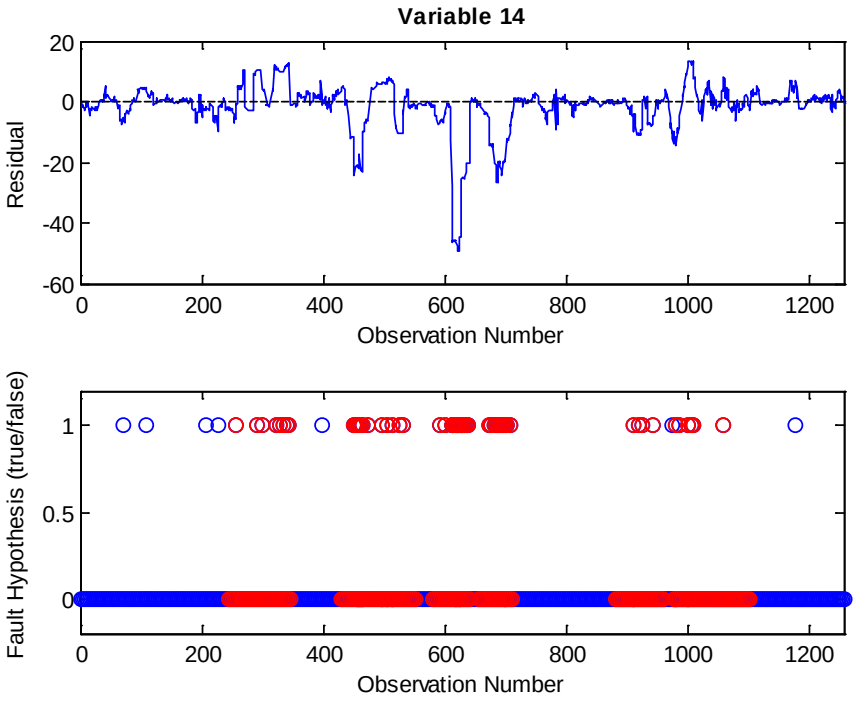

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

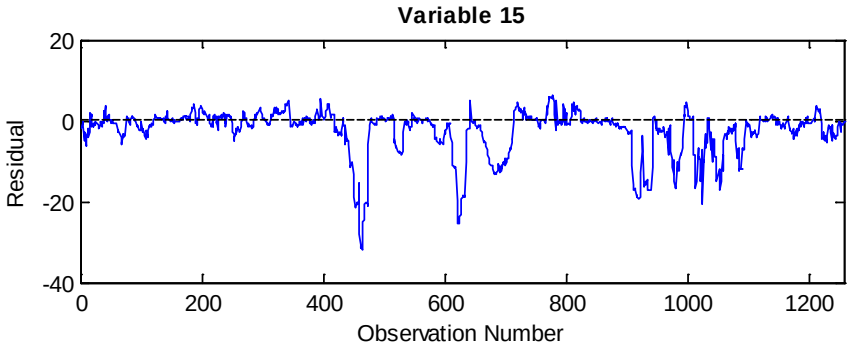

Variable 10
x 10^4
Residual
5
0
-5
-10
0
200
400
600
800
1000
1200
Observation Number

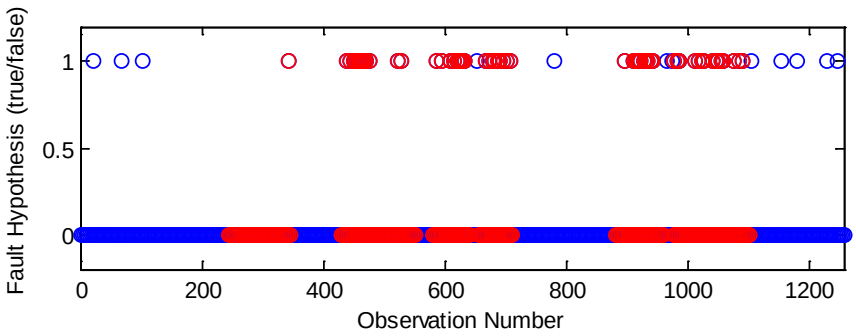

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

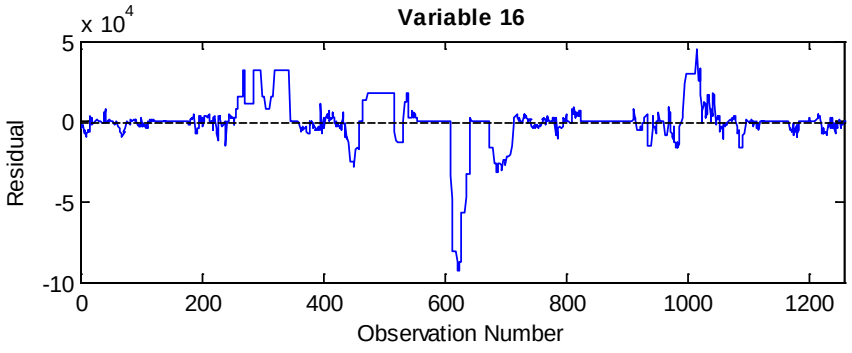

Variable 11
Residual
5000
0
-5000
-10000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number
Variable 12
Residual
200
0
-200
-400
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

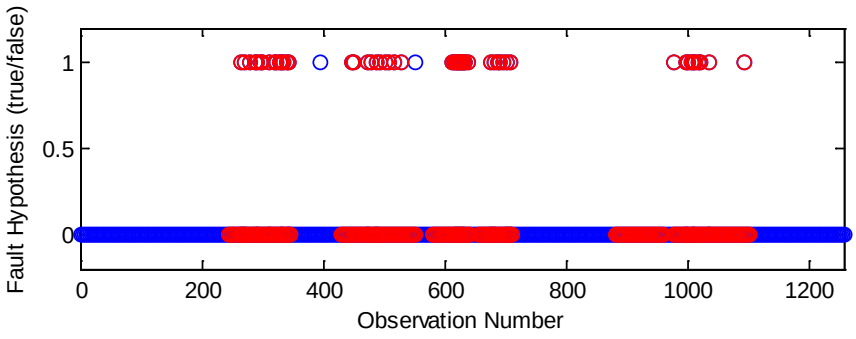

Variable 13
Residual
5000
0
-5000
-10000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

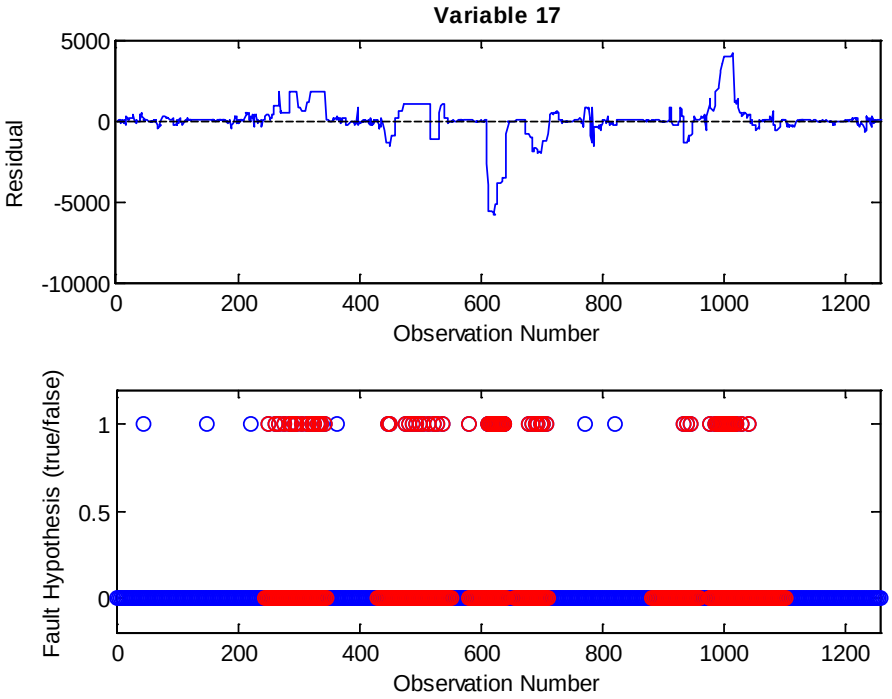

Variable 14
Residual
20
0
-20
-40
-60
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

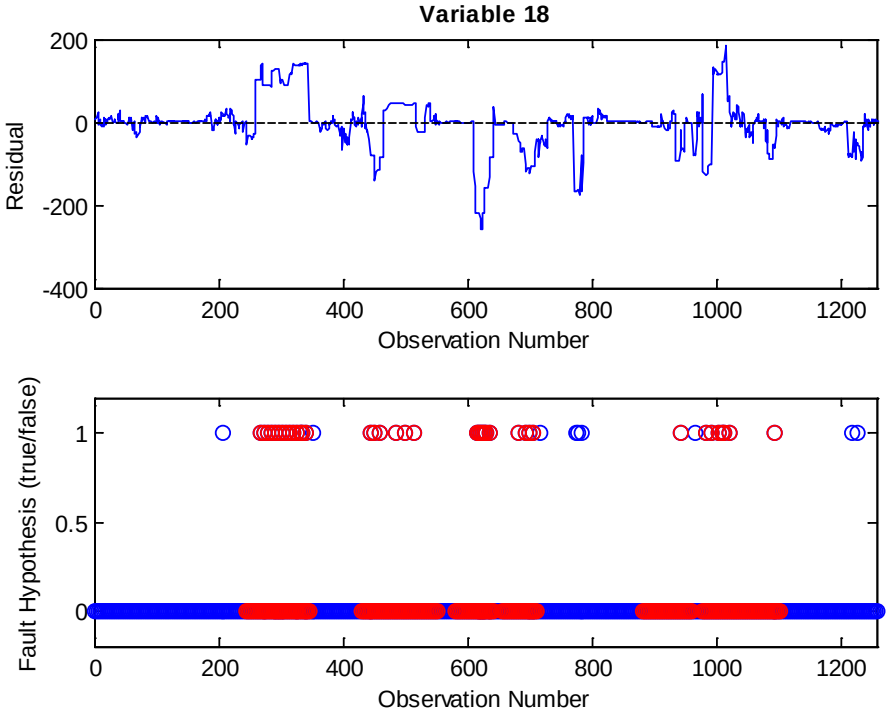

Variable 15
Residual
20
0
-20
-40
0
200
400
600
800
1000
1200
Observation Number

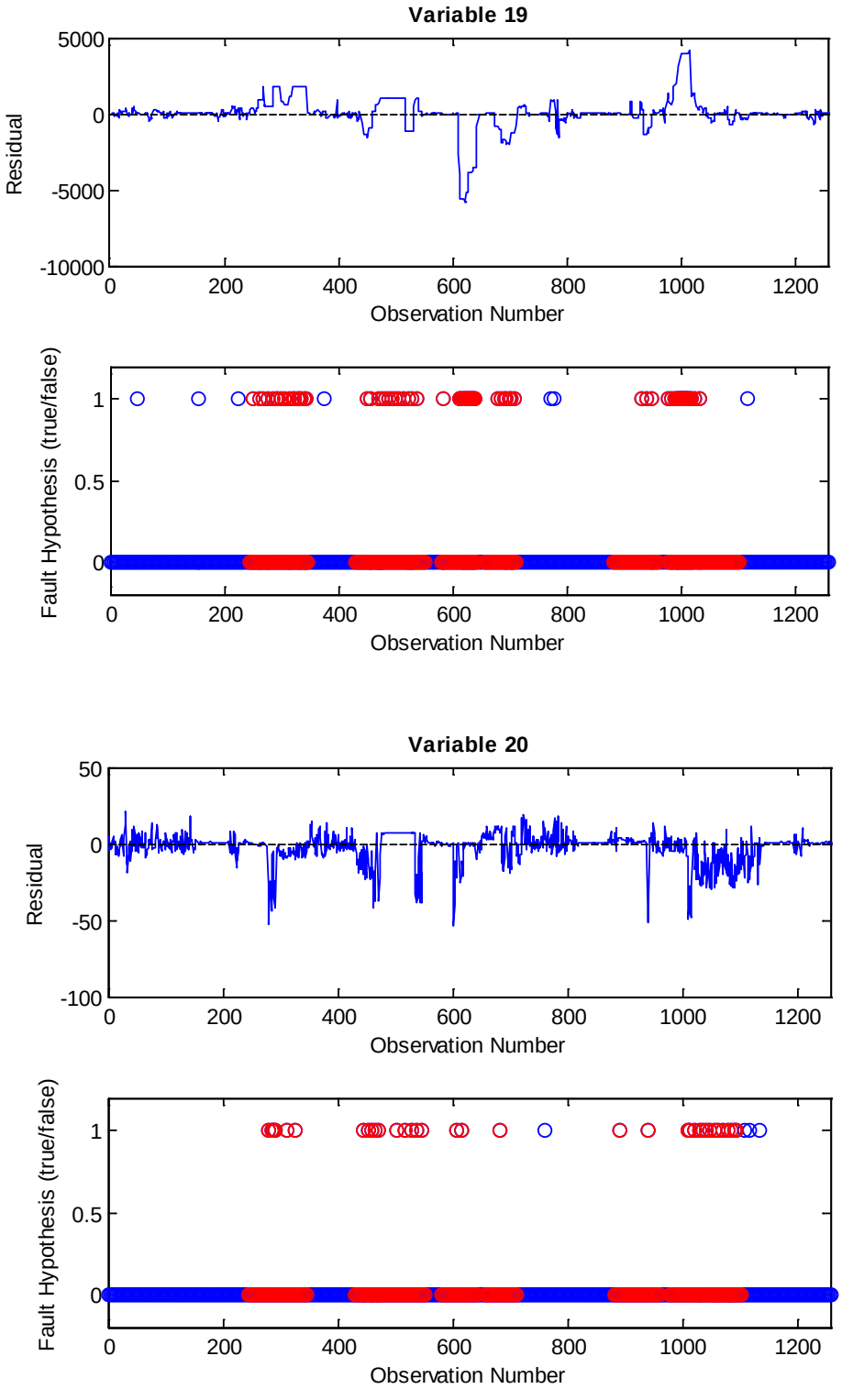

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

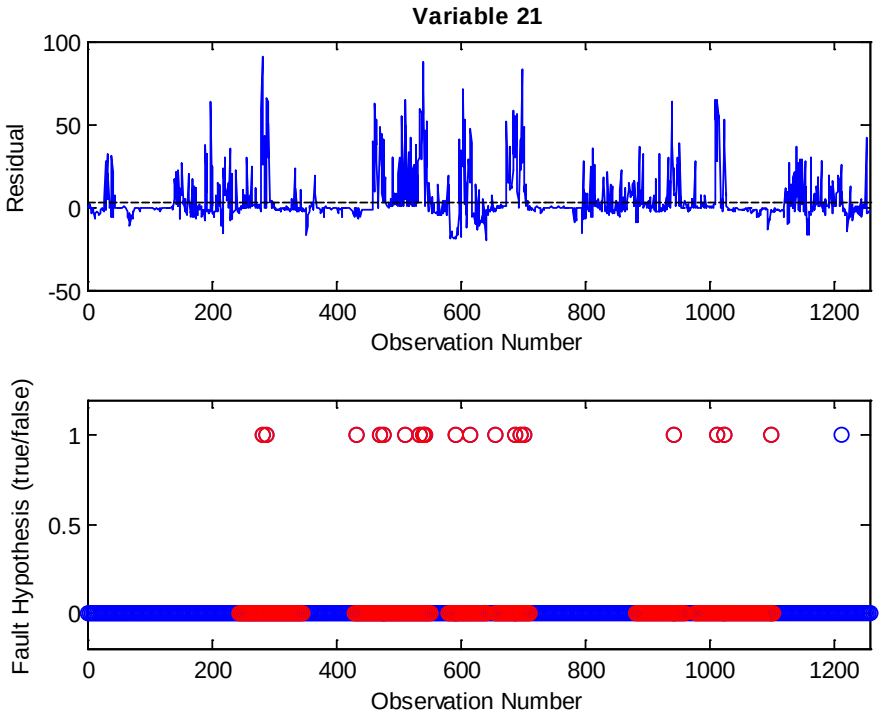

Variable 16
x 10^4
Residual
5
0
-5
-10
0
200
400
600
800
1000
1200
Observation Number


Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

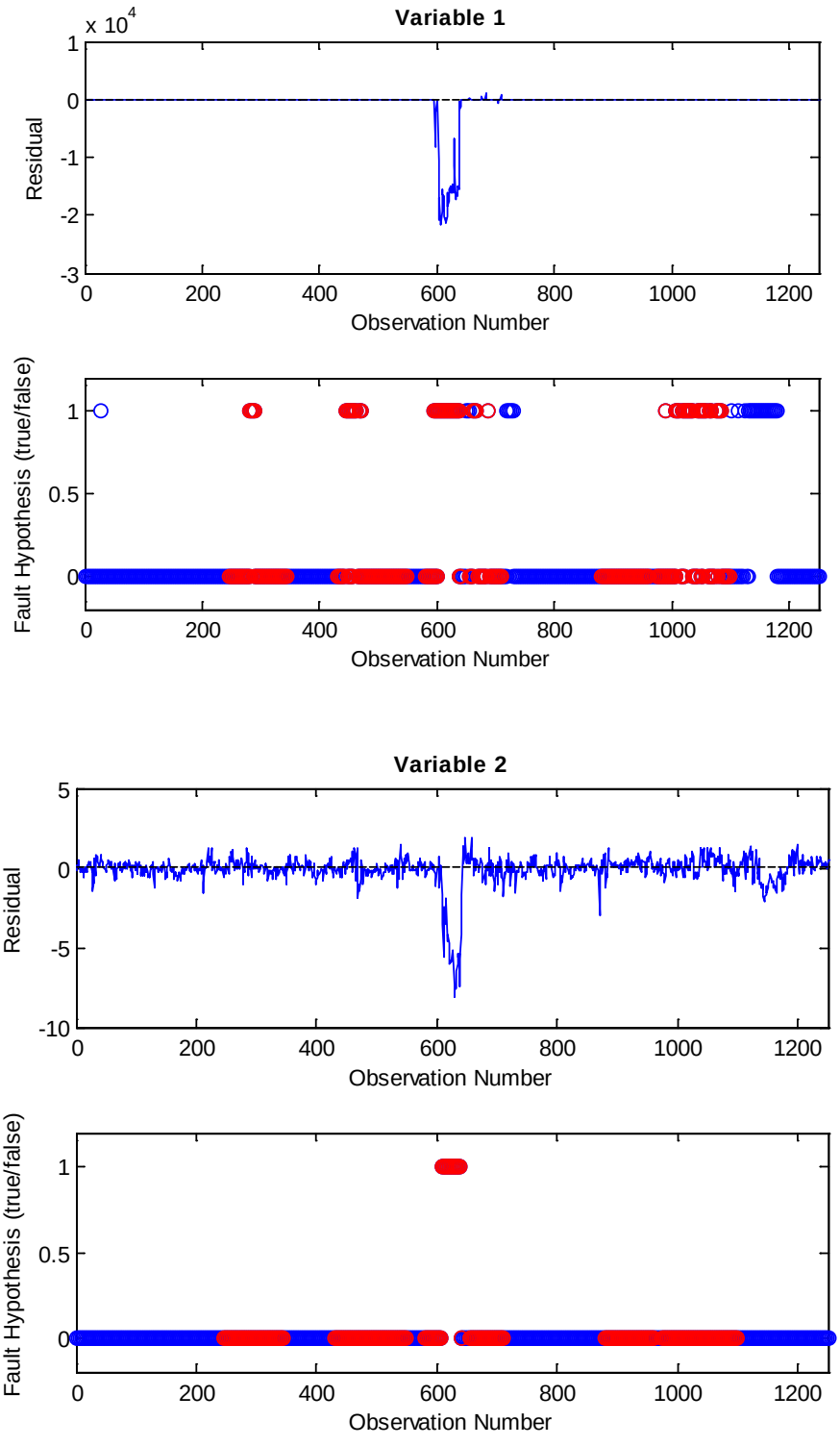

Variable 17
Residual
5000
0
-5000
-10000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

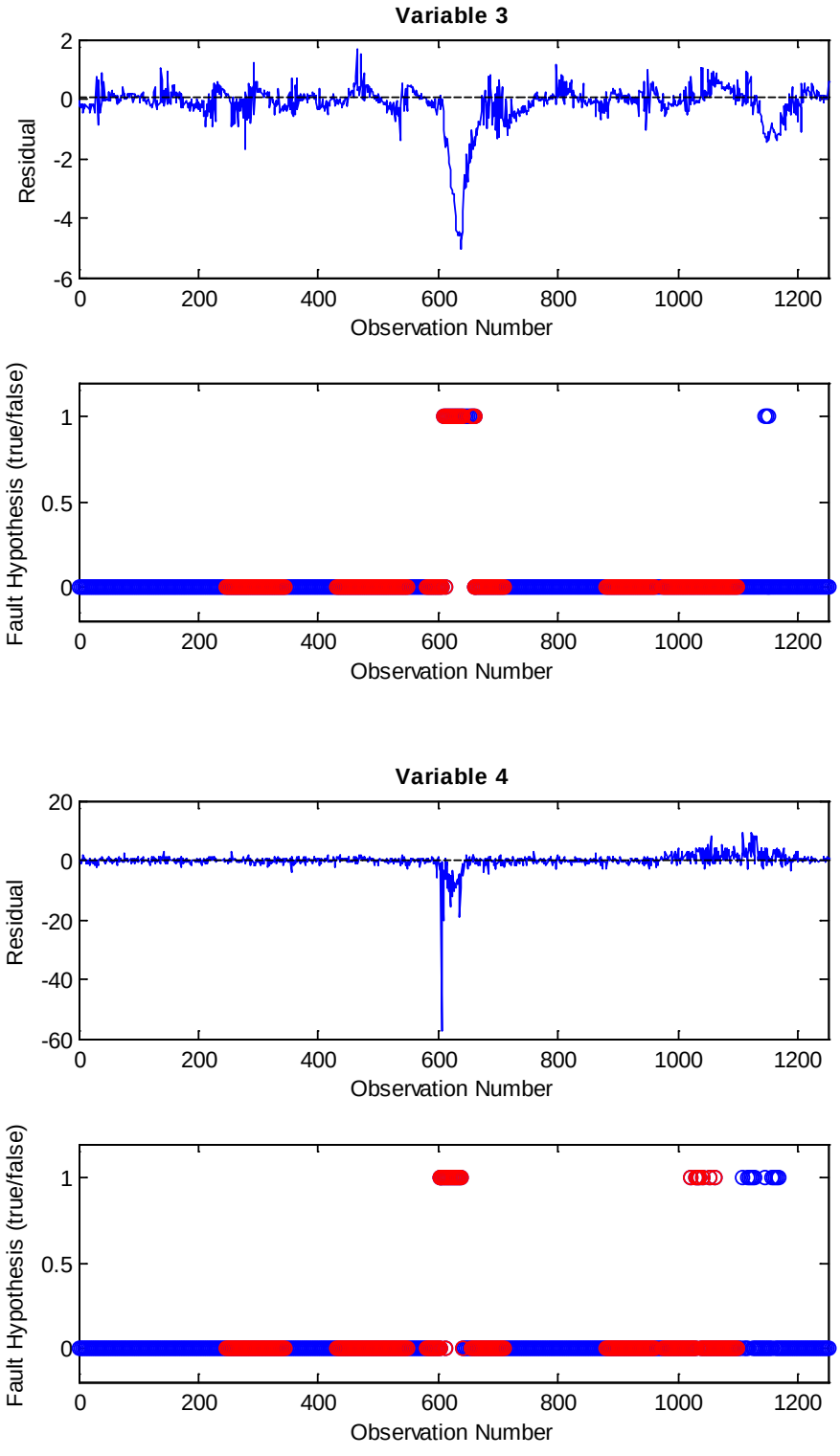

Variable 18
Residual
200
0
-200
-400
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

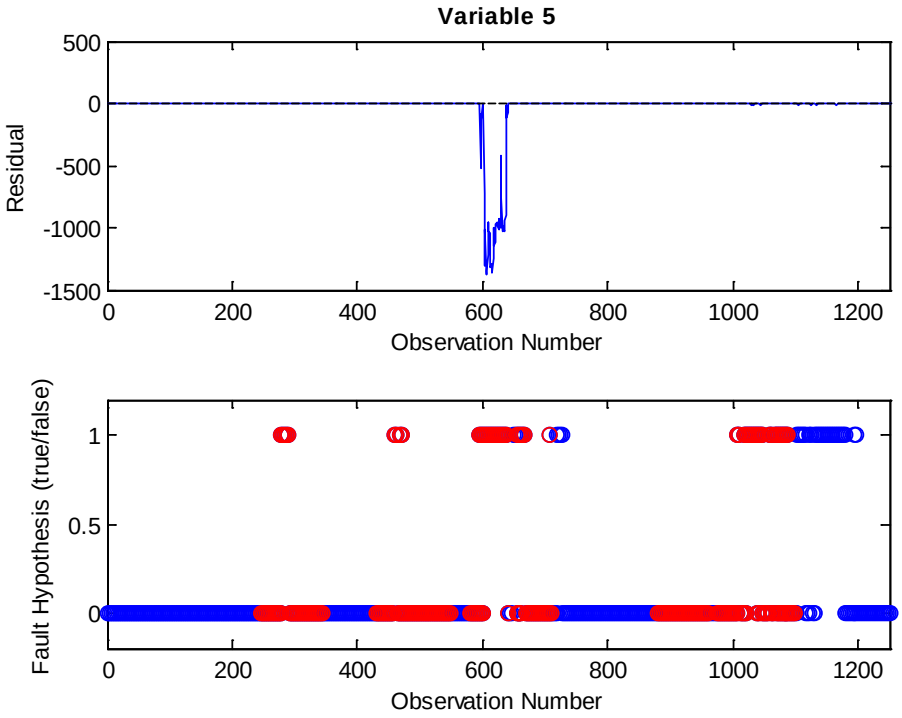

Variable 19
Residual
5000
0
-5000
-10000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number
Variable 20
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

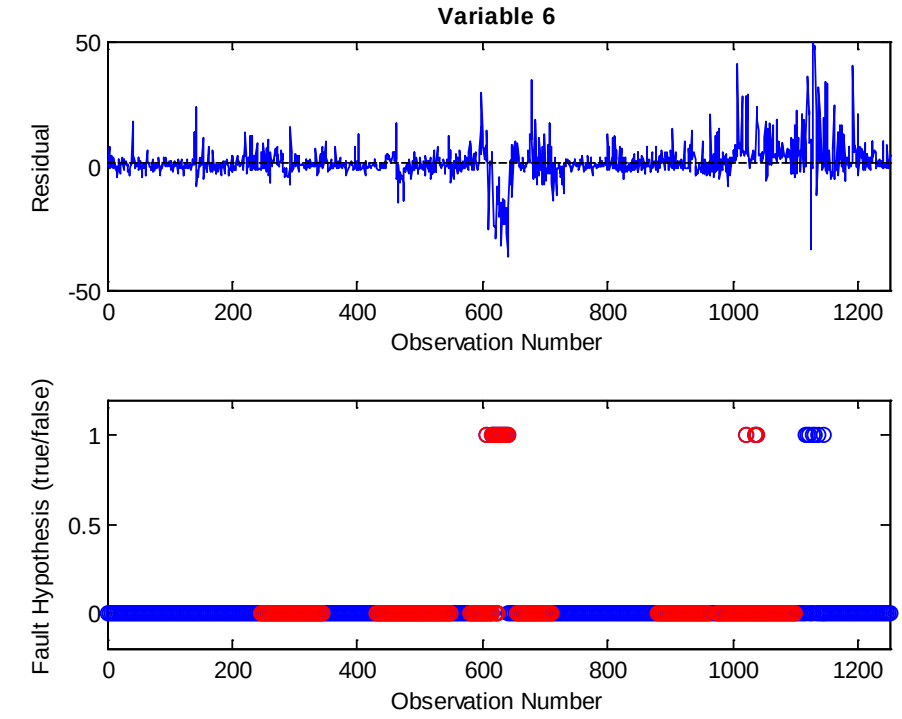

Variable 21
Residual
100
50
0
-50
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

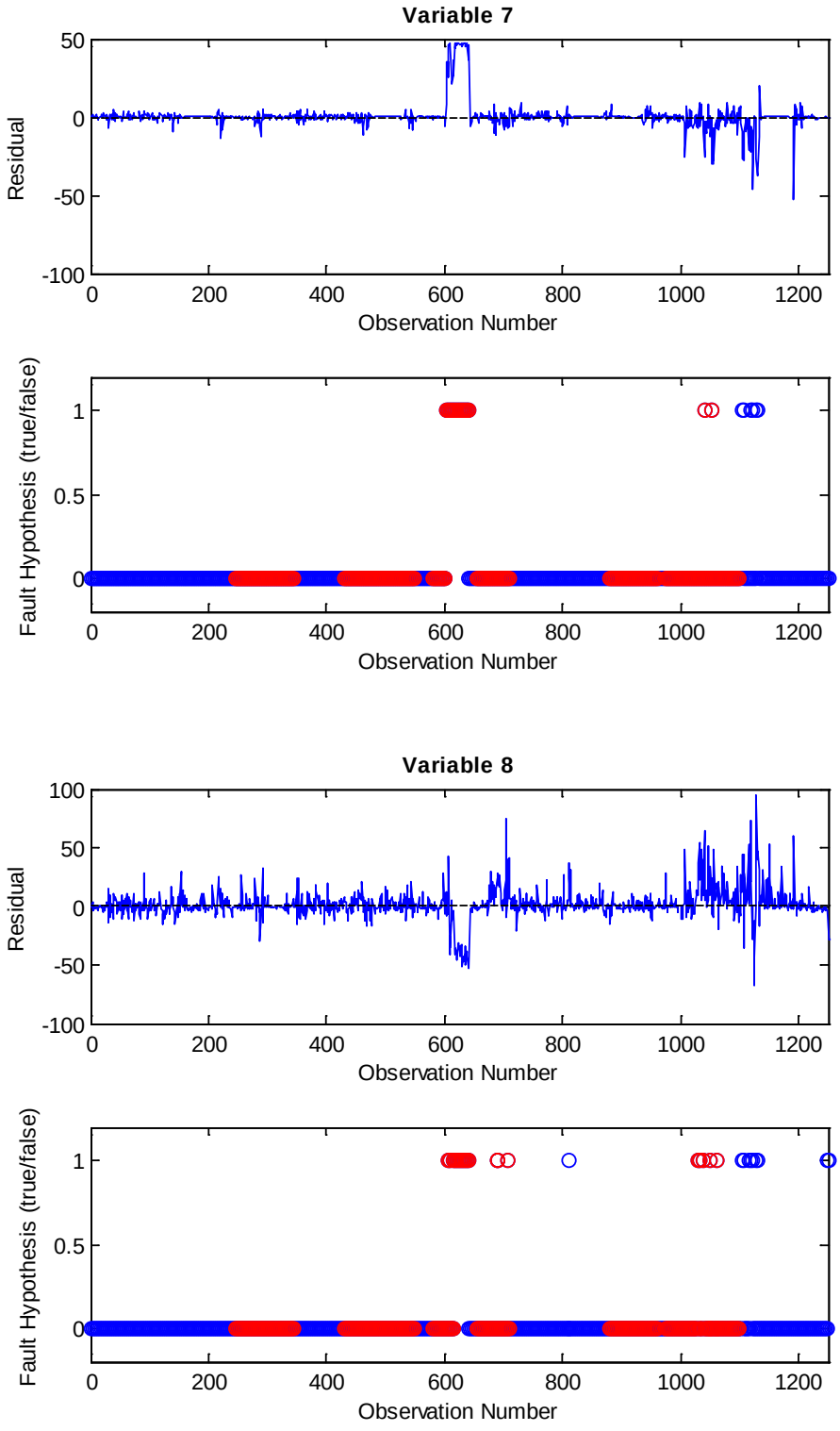
Additional Figures June Set – Model 2

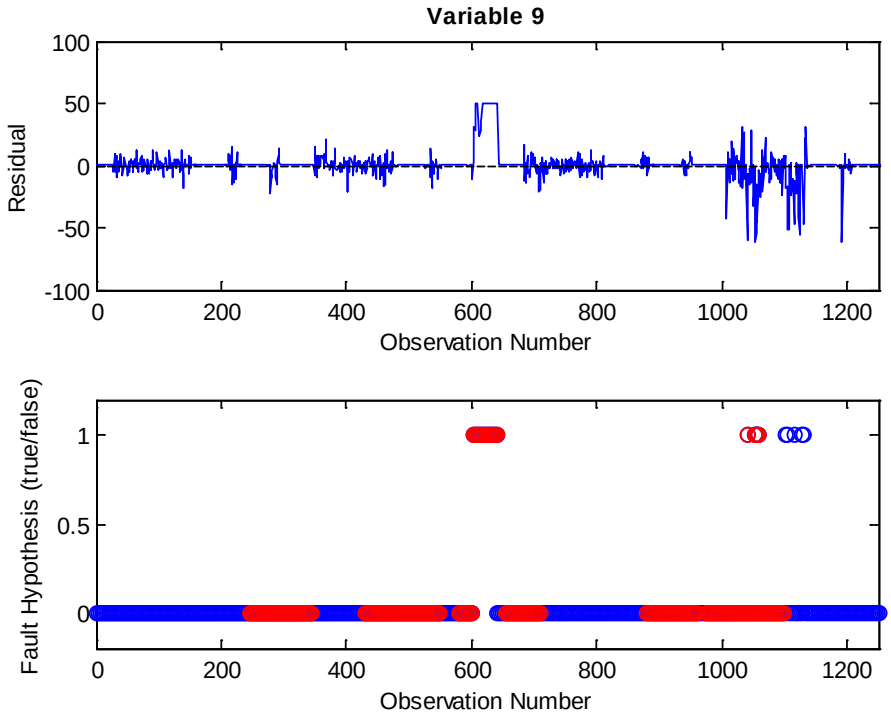


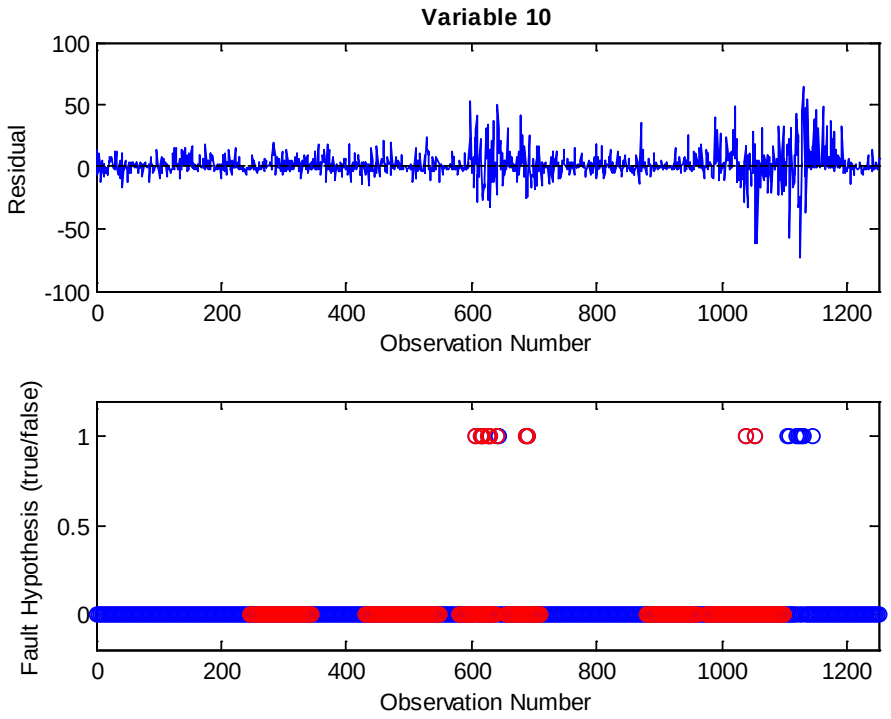

Variable 3
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number
Variable 4
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

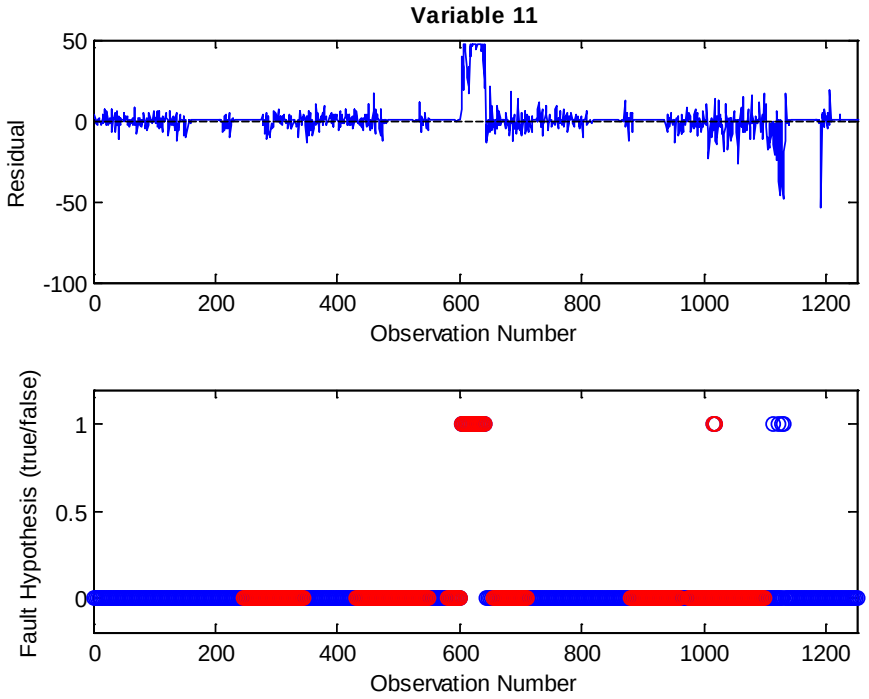

Variable 5
Residual
500
0
-500
-1000
-1500
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

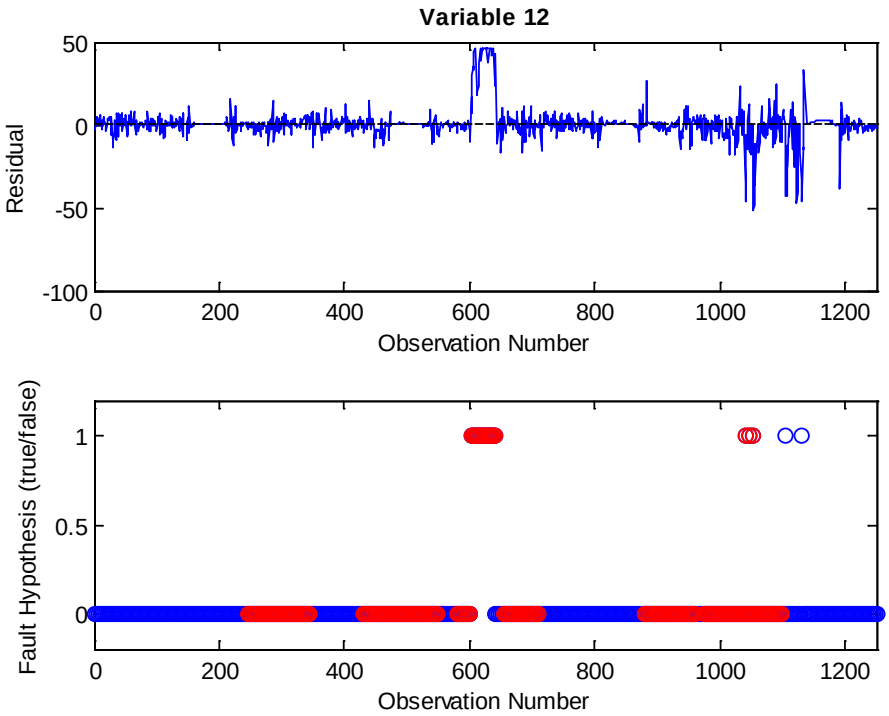

Variable 6
Residual
50
0
-50
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 7
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number
Variable 8
Residual
100
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

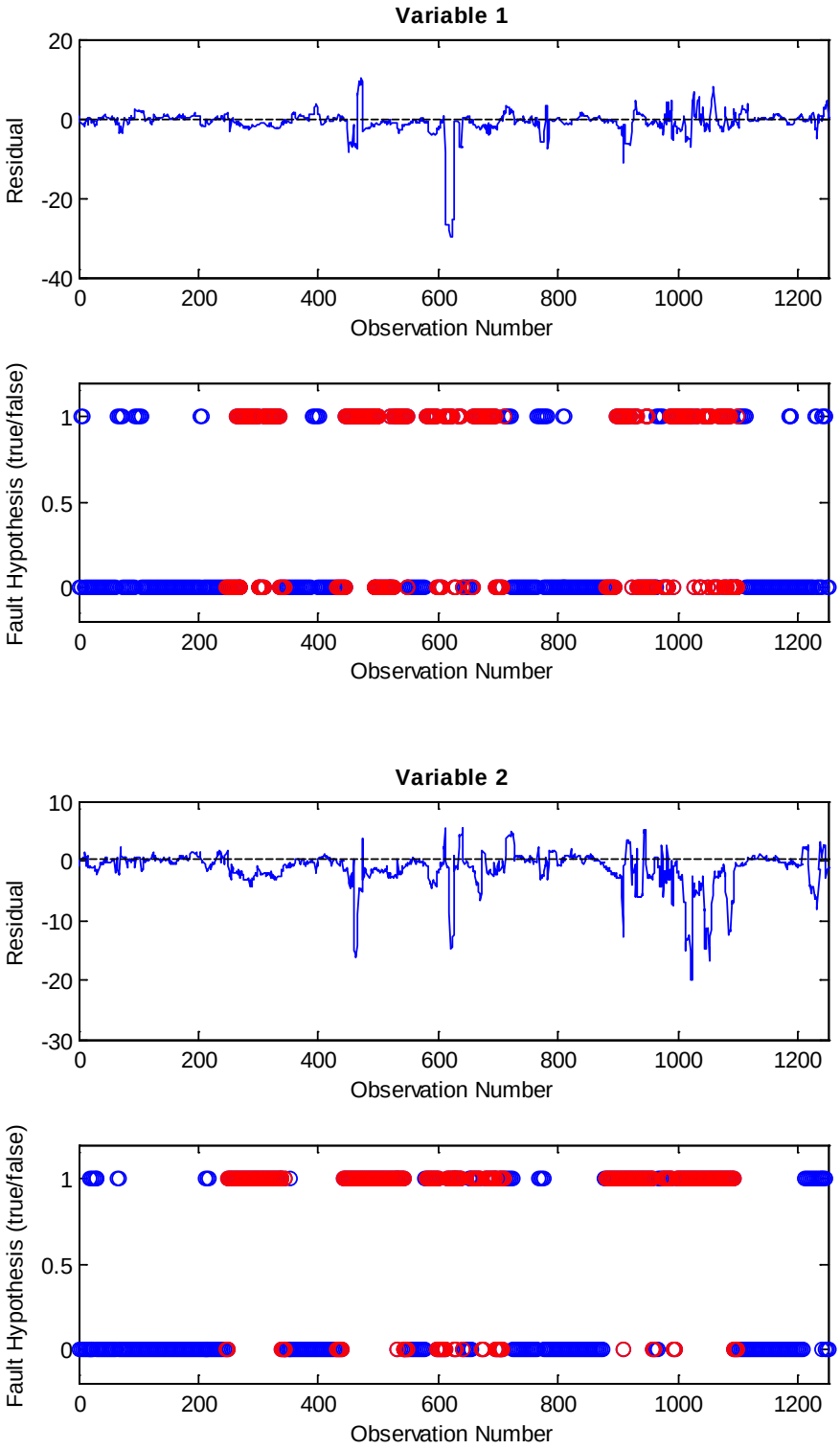

Variable 9
Residual
100
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

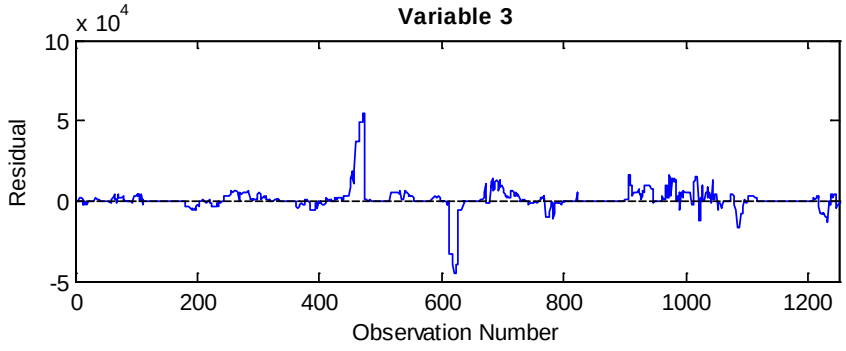

Variable 10
Residual
100
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

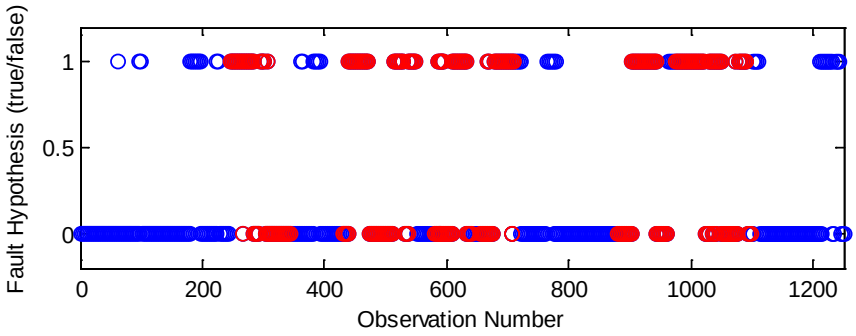

Variable 11
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

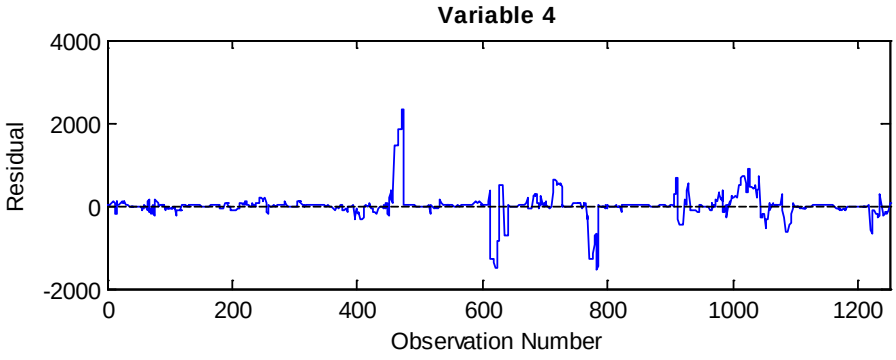

Variable 12
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

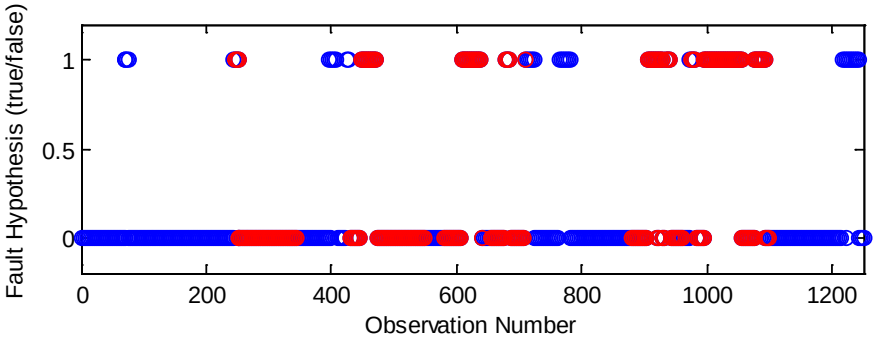
Additional Figures June Set – Model 3

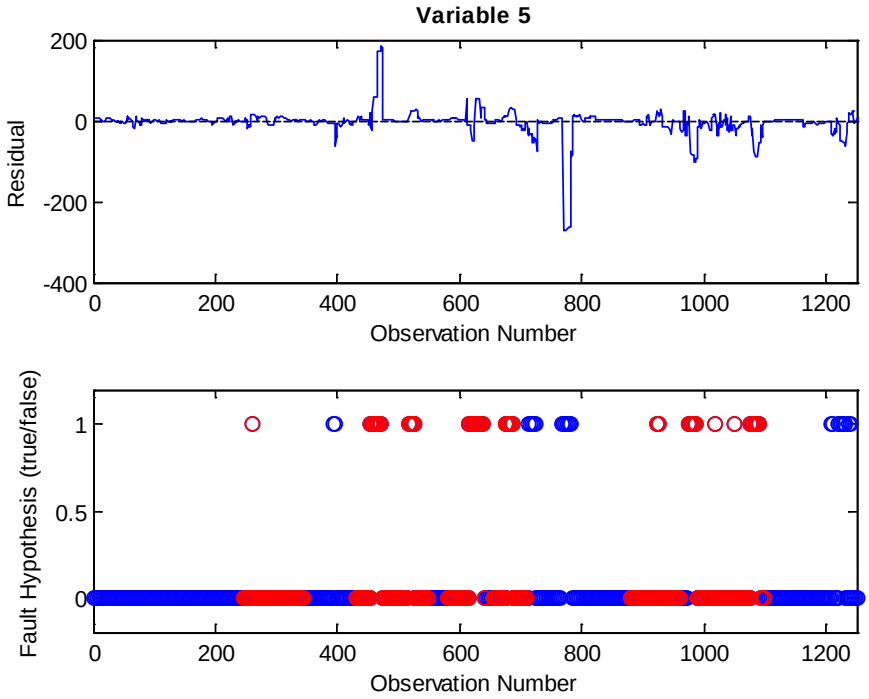


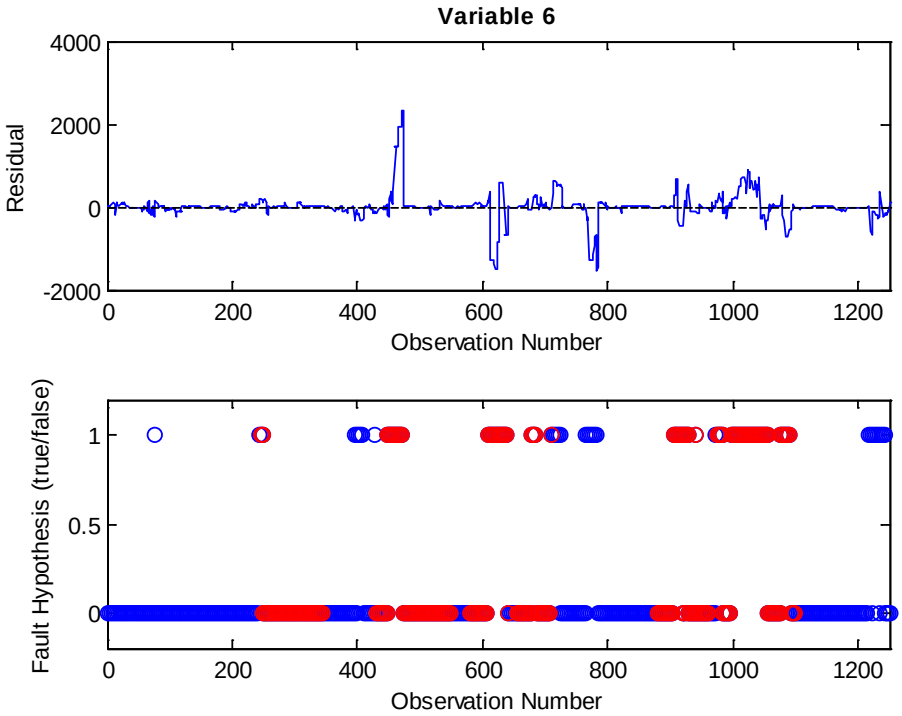

Variable 3
x 10^4
10
5
0
-5
Residual
0
200
400
600
800
1000
1200
Observation Number

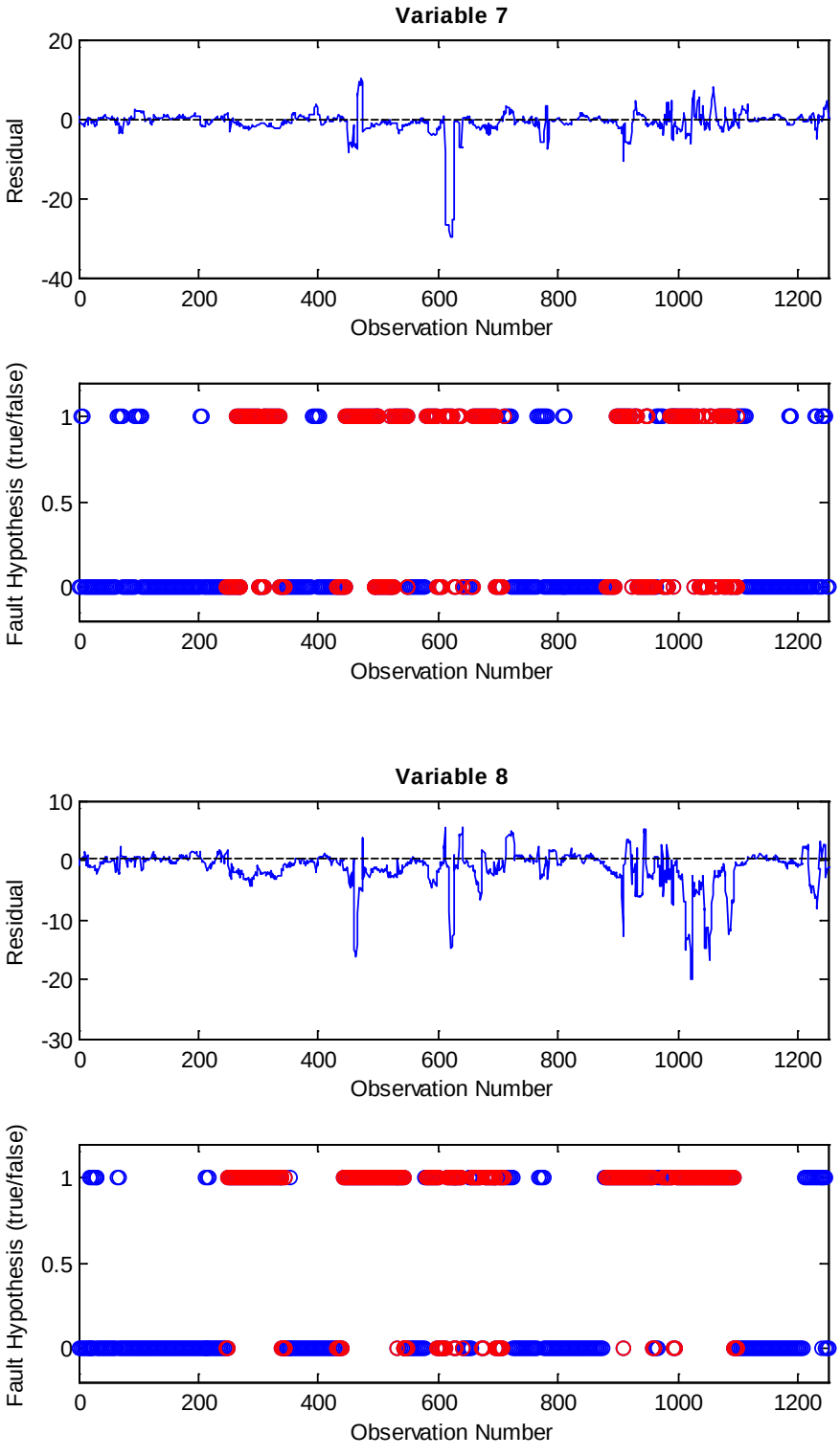

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

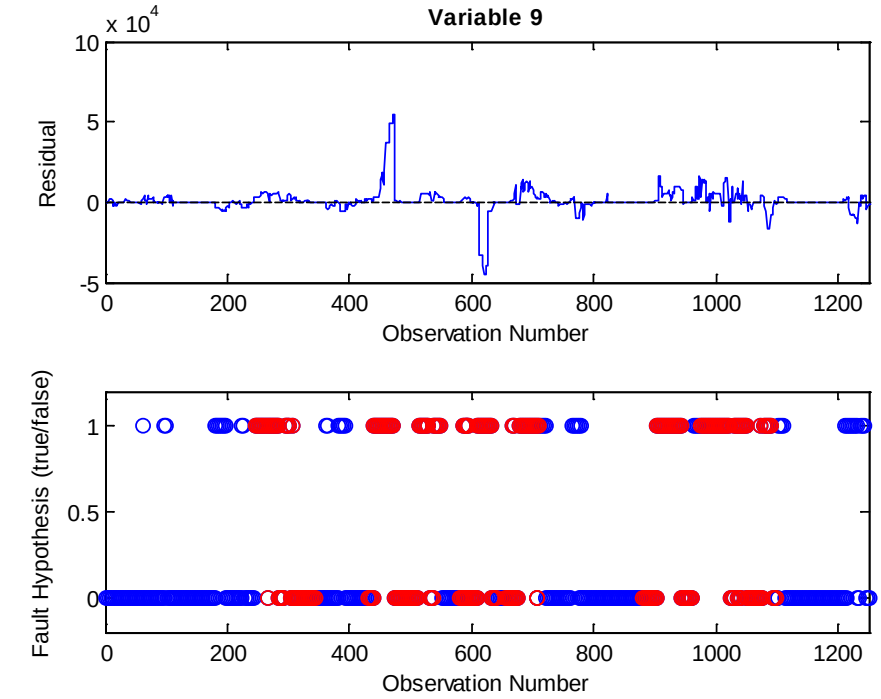

Variable 4
4000
2000
0
-2000
Residual
0
200
400
600
800
1000
1200
Observation Number

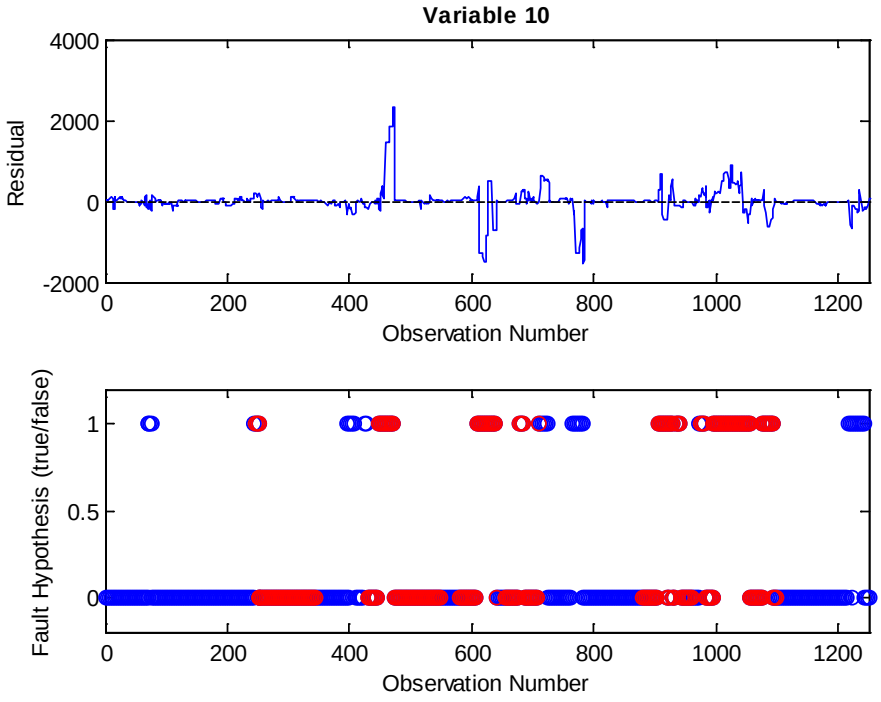

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

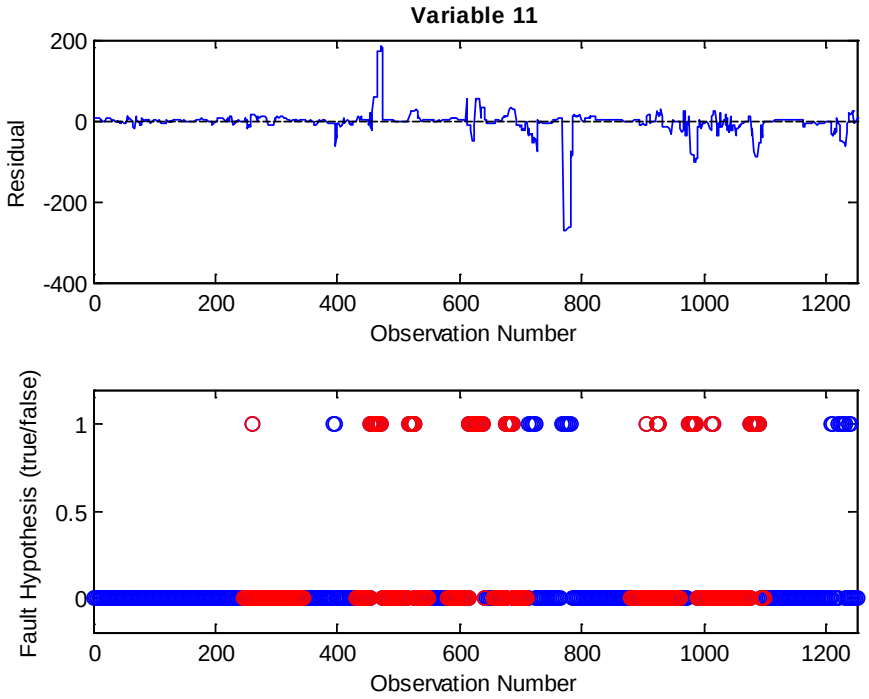

Variable 5
Residual
200
0
-200
-400
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

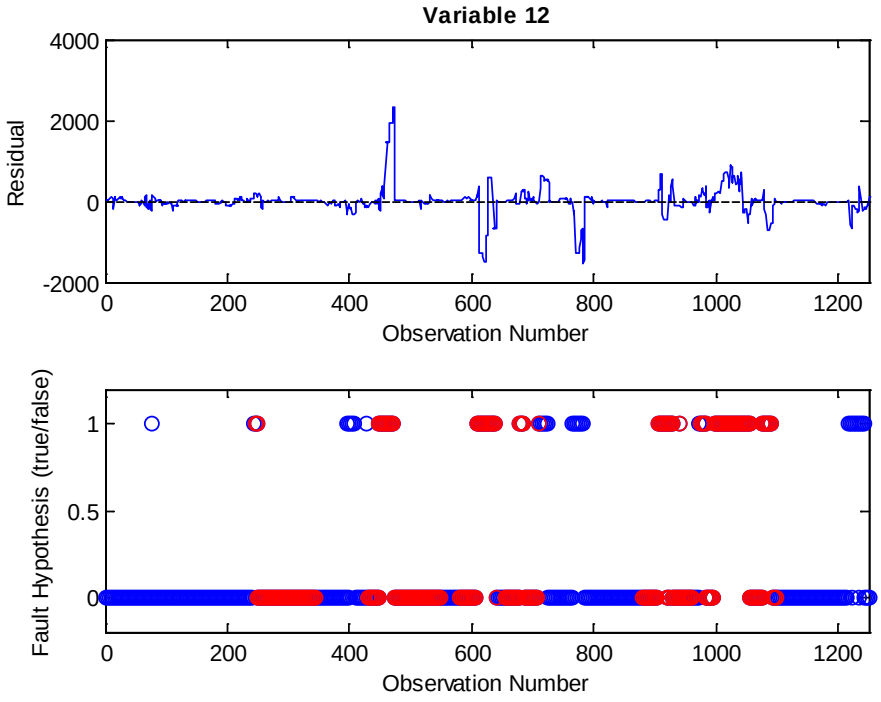

Variable 6
Residual
4000
2000
0
-2000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 7
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number
Variable 8
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

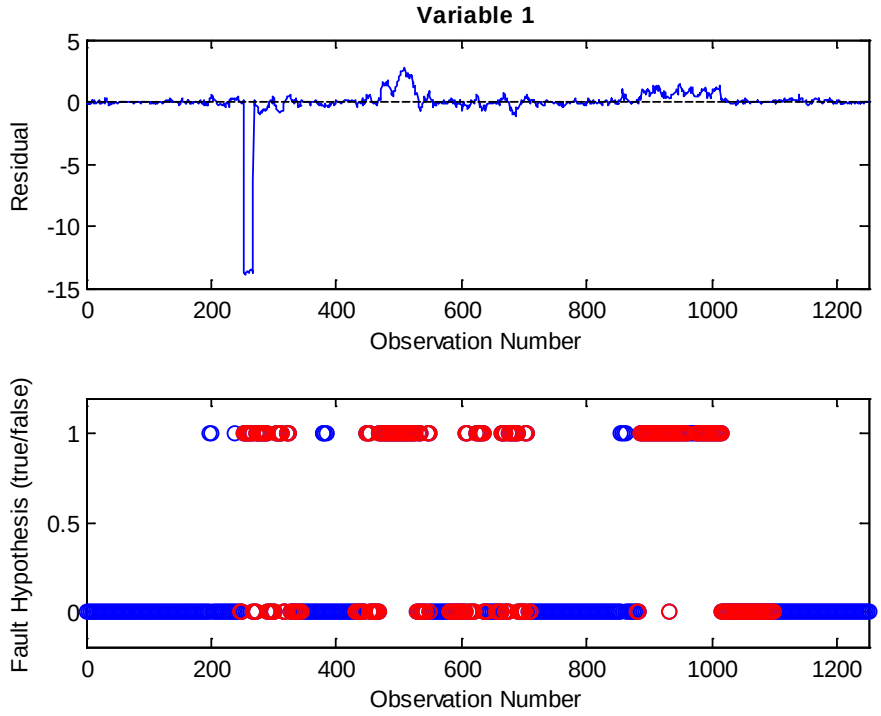

Variable 9
x 10^4
10
5
0
-5
Residual
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

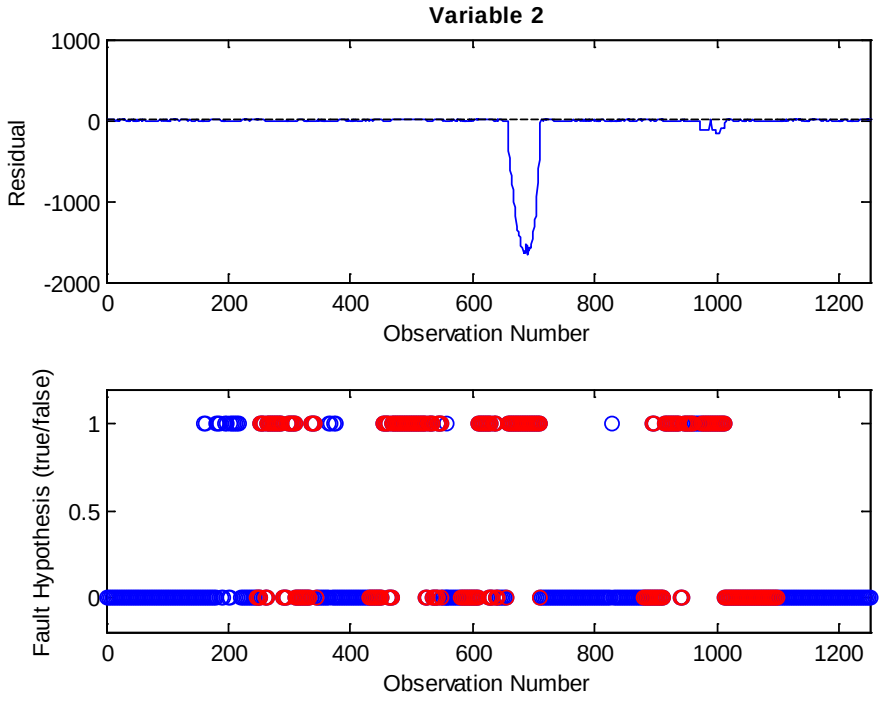

Variable 10
4000
2000
0
-2000
Residual
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

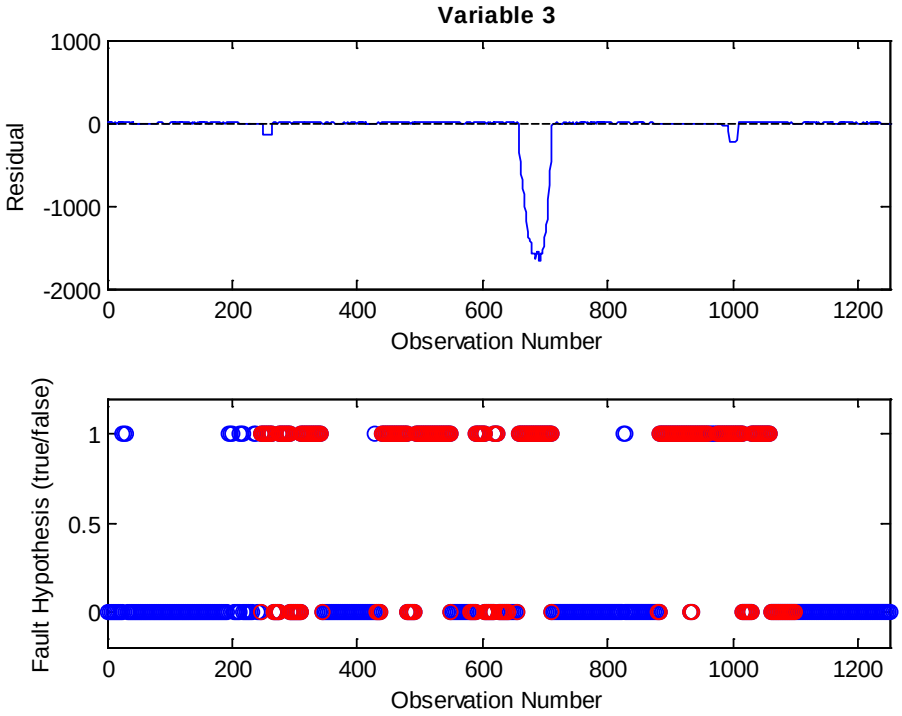

Variable 11
Residual
200
0
-200
-400
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

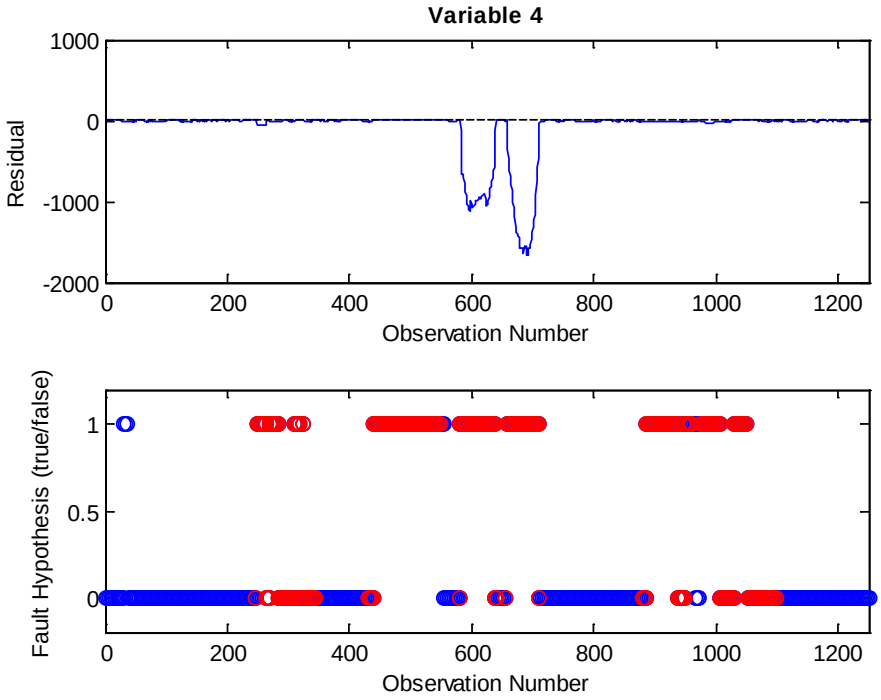

Variable 12
Residual
4000
2000
0
-2000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

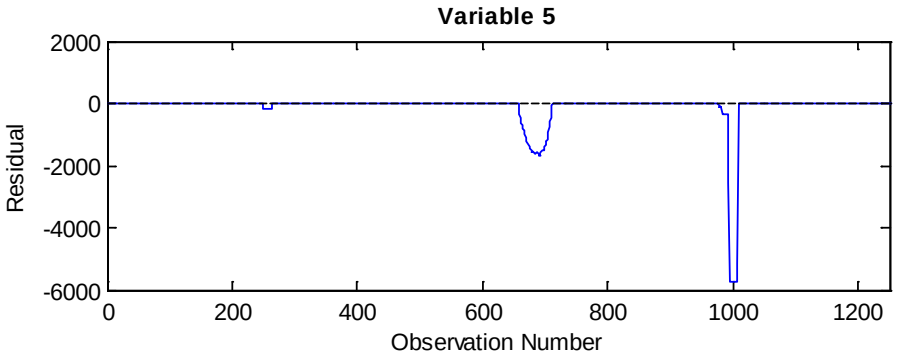
Additional Figures June Set – Model 4

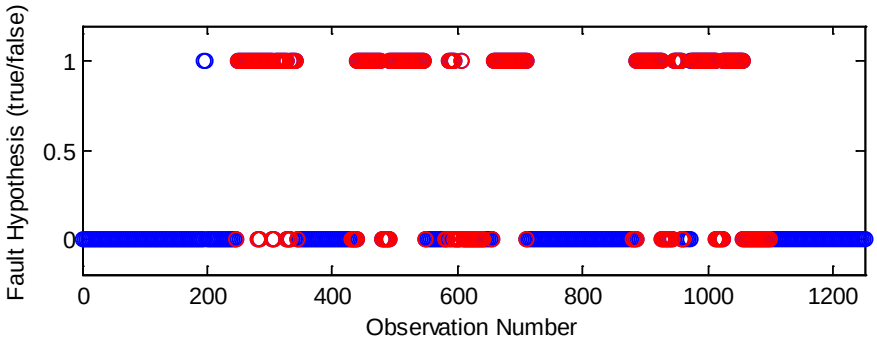


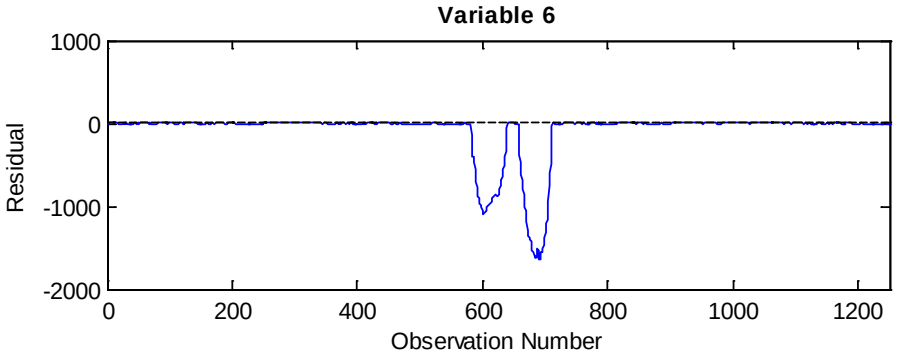


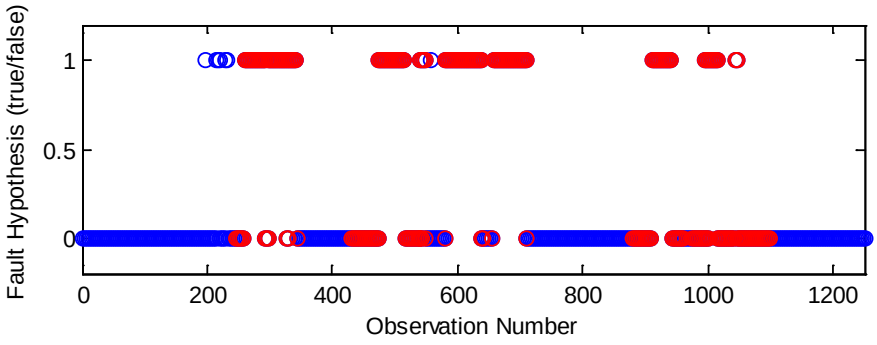

Variable 3
Residual
1000
0
-1000
-2000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

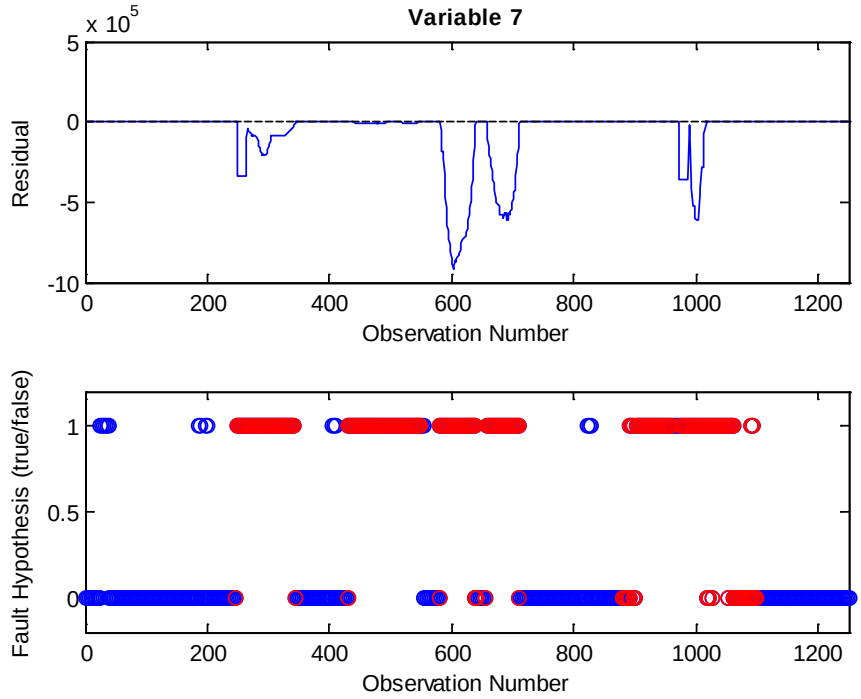

Variable 4
Residual
1000
0
-1000
-2000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

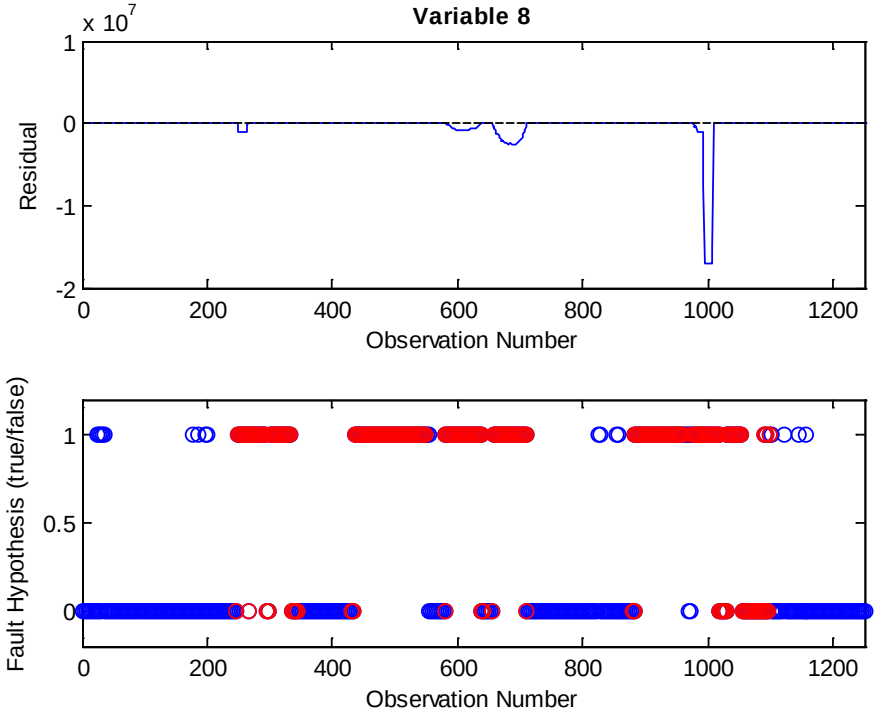

Variable 5
Residual
2000
0
-2000
-4000
-6000
0
200
400
600
800
1000
1200
Observation Number

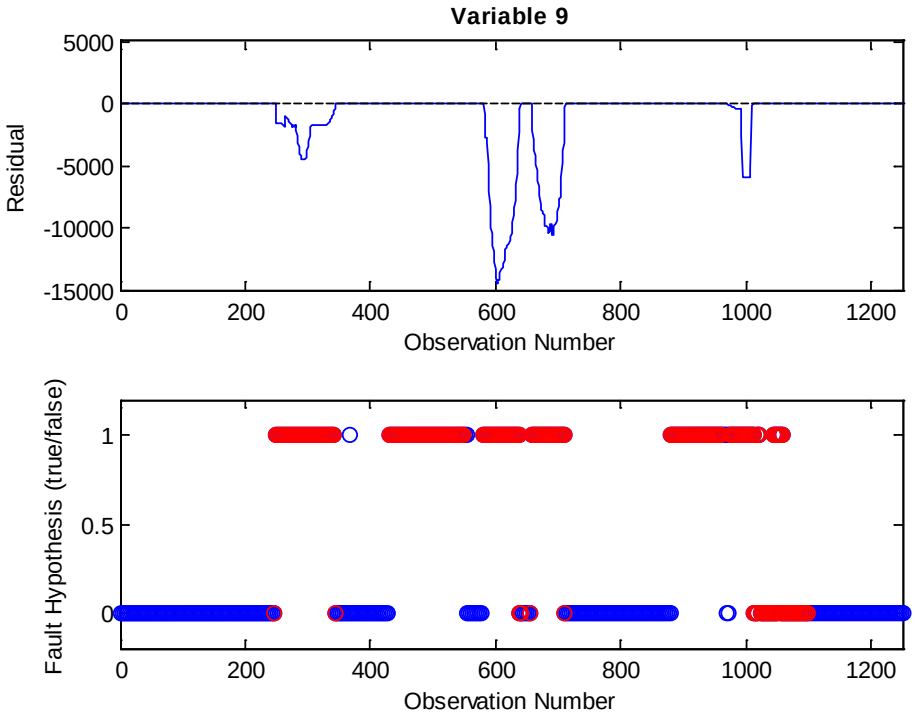

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 6
Residual
1000
0
-1000
-2000
0
200
400
600
800
1000
1200
Observation Number


Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 7
x 10^5
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number


Variable 8
x 10^7
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number


Variable 9
Residual
5000
0
-5000
-10000
-15000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

Appendix C: SPRT Results – June Data Set

Model 1 AAKR SPRT Results

Int. #/Sig. #	1 Sparta	2 NTP	3 SSH	4 Smb2	5 Brute	6 Theft
1.		X	X			X
2.		X	X			X
3.		X		X		X
4.	X	X	X	X	X	X
5.	X	X	X	X		X
6.	X	X	X	X	X	X
7.	X	X	X	X	X	X
8.	X	X	X	X		X
9.		X	X	X	X	X
10.	X	X	X	X		X
11.	X	X	X	X		X
12.	X	X	X	X		X
13.	X	X	X	X		X
14.	X	X	X	X		X
15.		X	X	X	X	X
16.	X	X	X	X		X
17.	X	X	X	X		X
18.	X	X	X	X		X
19.	X	X		X		X
20.	X	X				X
21.			X	X		

Model 1 AAMSET SPRT Results

Int. #/Sig. #	1 Sparta	2 NTP	3 SSH	4 Smb2	5 Brute	6 Theft
1.	X	X	X			X
2.	X	X	X	X		X
3.				X		X
4.	X	X	X	X		X
5.	X	X	X	X	X	X
6.	X	X	X	X	X	X
7.	X	X	X	X	X	X
8.	X	X	X	X	X	X
9.	X	X	X	X	X	X
10.	X	X	X	X	X	X
11.	X	X	X	X	X	X
12.	X	X	X	X	X	X
13.	X	X	X	X	X	X
14.	X	X	X	X	X	X
15.	X	X	X	X	X	X
16.	X	X	X	X	X	X
17.	X	X	X	X	X	X
18.	X	X	X	X	X	X
19.	X	X	X	X	X	X
20.	X	X	X	X		X
21.	X			X		X

Model 2 AAKR SPRT Results

Int. #/Sig. #	1 Sparta	2 NTP	3 SSH	4 Smb2	5 Brute	6 Theft
1.	X	X	X	X	X	X
2.			X			
3.			X			
4.			X			X
5.	X	X	X	X		X
6.		X	X	X		X
7.		X	X	X		X
8.	X	X	X	X		X
9.	X	X	X	X		X
10.			X	X	X	X
11.			X	X		X
12.			X	X		X

Model 2 AAMSET SPRT Results

Int. #/Sig. #	1 Sparta	2 NTP	3 SSH	4 Smb2	5 Brute	6 Theft
1.	X	X	X	X	X	X
2.			X			
3.			X			
4.			X			X
5.	X	X	X			X
6.			X			X
7.			X			X
8.		X	X			X
9.		X	X	X		X
10.			X			X
11.			X		X	X
12.		X	X			X

Model 3 AAKR SPRT Results

Int. #/Sig. #	1 Sparta	2 NTP	3 SSH	4 Smb2	5 Brute	6 Theft
1.		X	X	X	X	X
2.	X	X	X	X	X	X
3.	X	X	X	X	X	X
4.		X	X		X	X
5.		X	X			X
6.		X	X		X	X
7.	X	X	X	X	X	X
8.	X	X	X	X	X	X
9.	X	X	X	X	X	X
10.		X	X		X	X
11.		X	X	X		X
12.		X	X		X	X

Model 3 AAMSET SPRT Results

Int. #/Sig. #	1 Sparta	2 NTP	3 SSH	4 Smb2	5 Brute	6 Theft
1.	X	X	X	X	X	X
2.	X	X	X	X	X	X
3.	X	X	X	X	X	X
4.		X	X	X	X	X
5.		X	X	X	X	X
6.		X	X	X	X	X
7.	X	X	X	X	X	X
8.	X	X	X	X	X	X
9.	X	X	X	X	X	X
10.		X	X	X	X	X
11.		X	X	X	X	X
12.		X	X	X	X	X

Model 4 AAKR SPRT Results

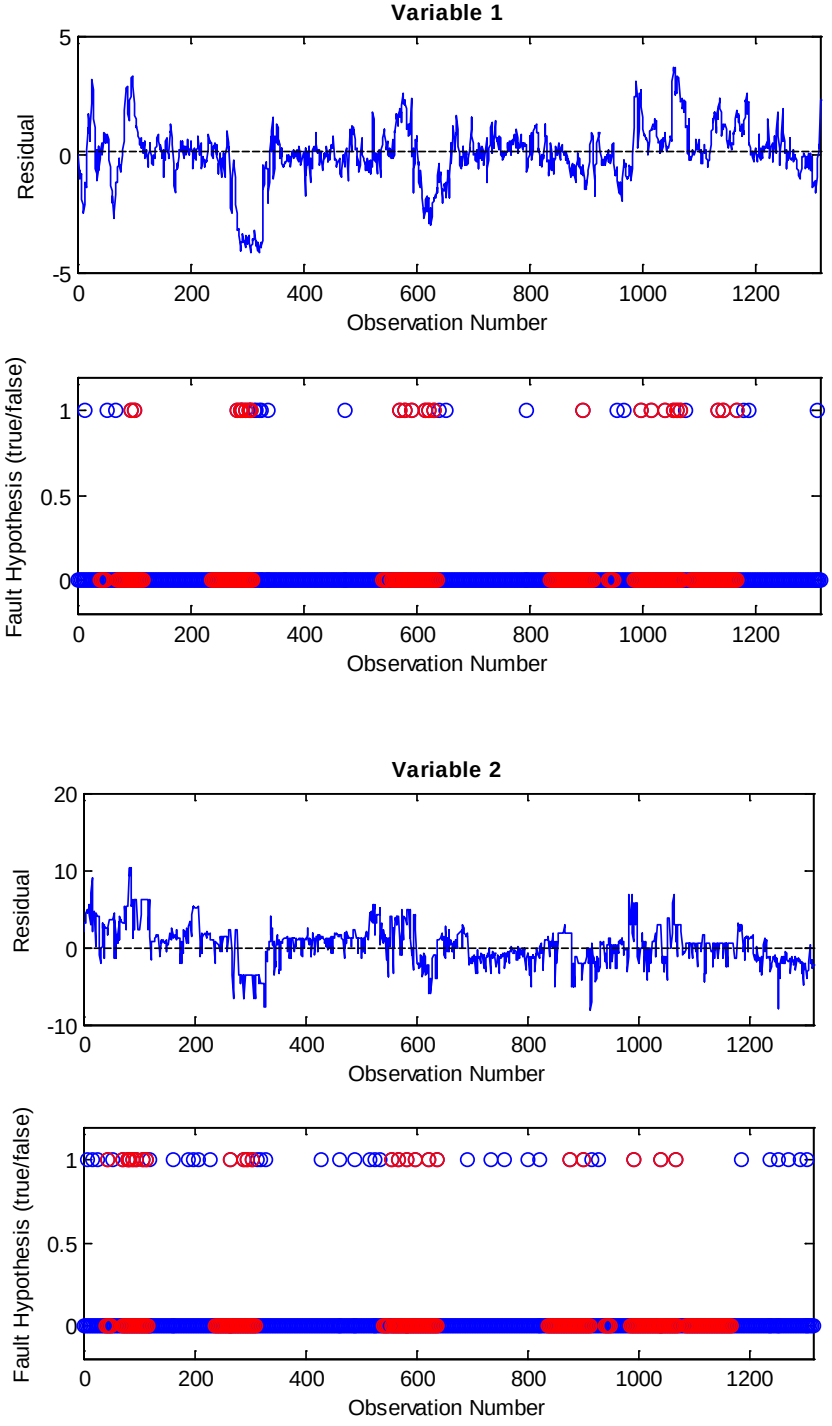
Int. #/Sig. #	1 Sparta	2 NTP	3 SSH	4 Smb2	5 Brute	6 Theft
1.	X	X	X		X	X
2.	X	X	X	X	X	X
3.	X	X	X	X	X	X
4.	X	X	X	X	X	X
5.	X	X	X	X	X	X
6.	X	X	X	X	X	X
7.	X	X	X	X	X	X
8.	X	X	X	X	X	X
9.	X	X	X	X	X	X

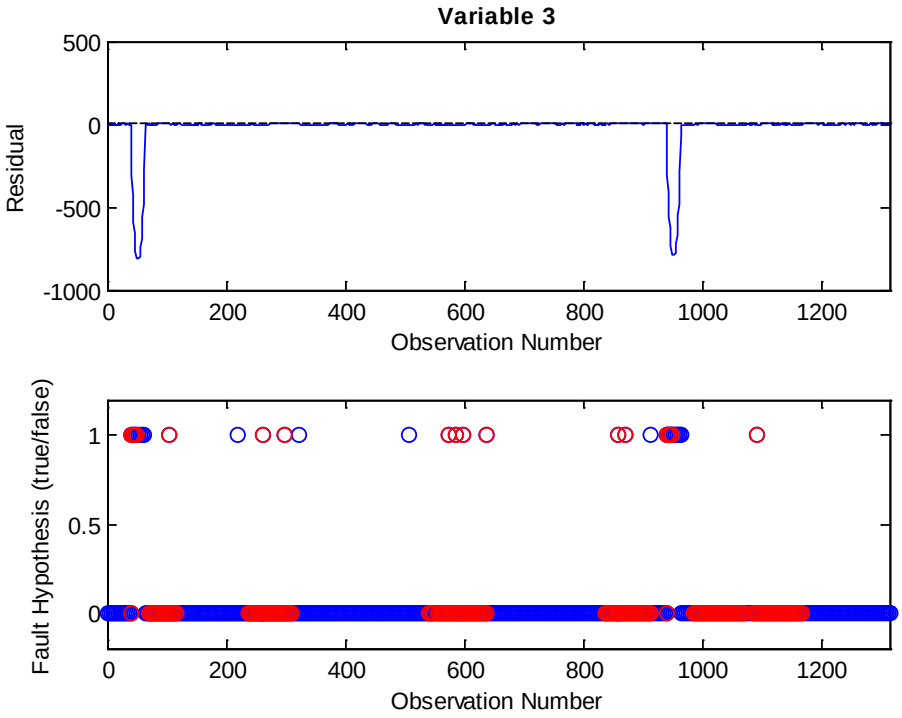
Model 4 AAMSET SPRT Results

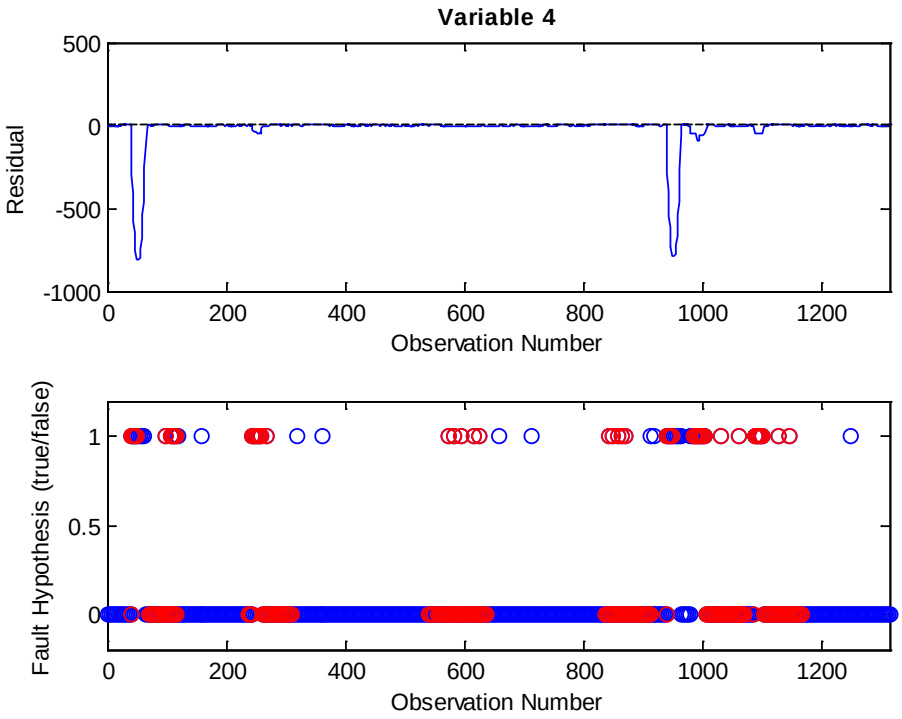
Int. #/Sig. #	1 Sparta	2 NTP	3 SSH	4 Smb2	5 Brute	6 Theft
1.	X					
2.				X		X
3.	X	X		X	X	X
4.	X	X	X	X	X	X
5.	X			X		X
6.	X	X	X	X		X
7.	X	X	X	X	X	X
8.	X	X	X	X	X	X
9.	X	X	X	X	X	X

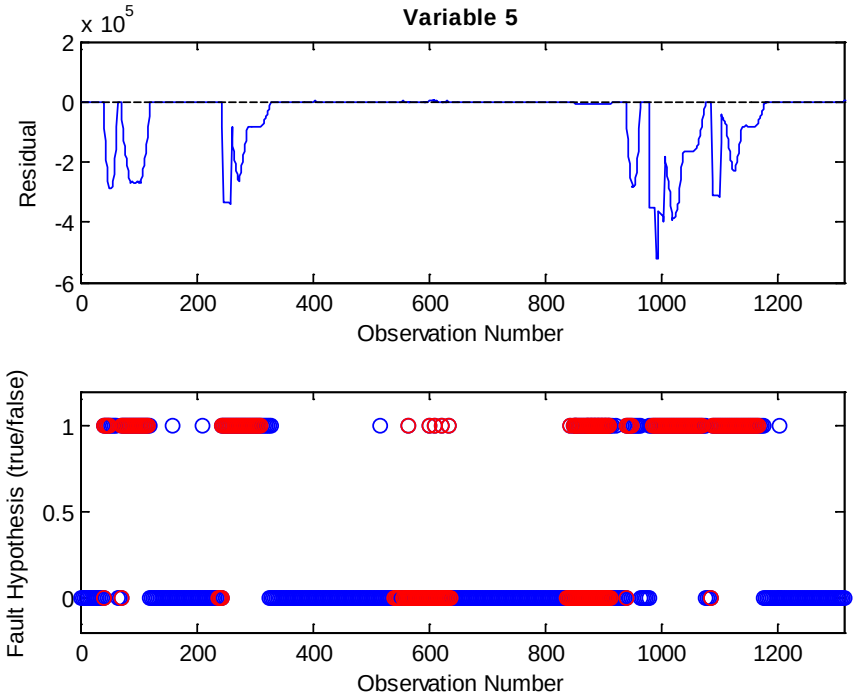
Appendix D: Additional Figures – March Data Set

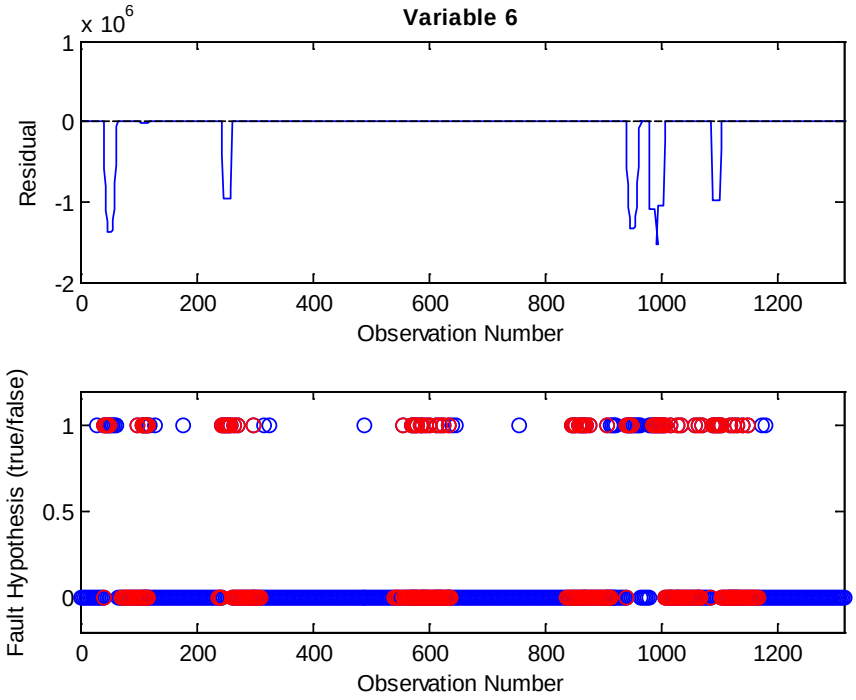
Additional Figures March Set – Model 1

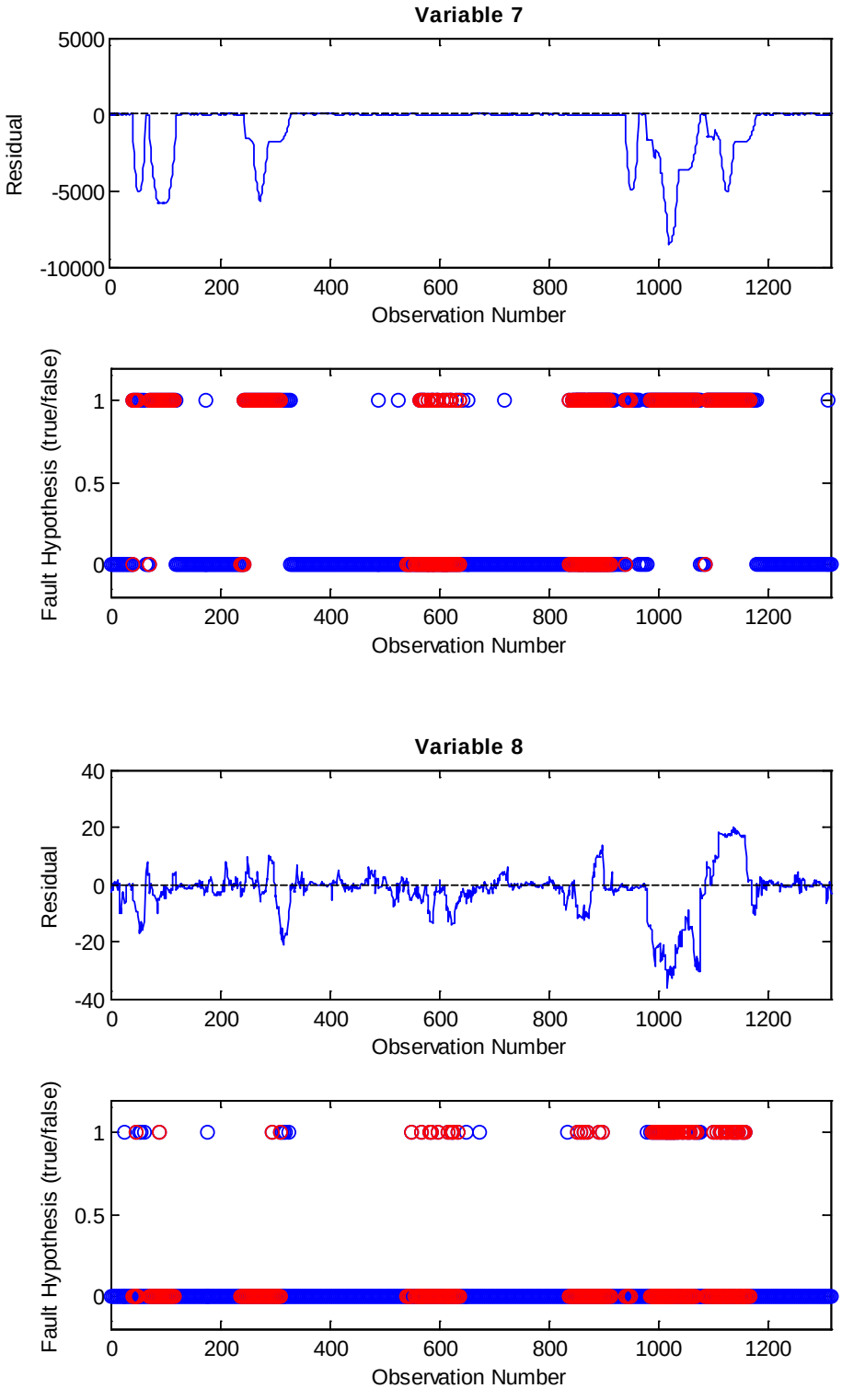


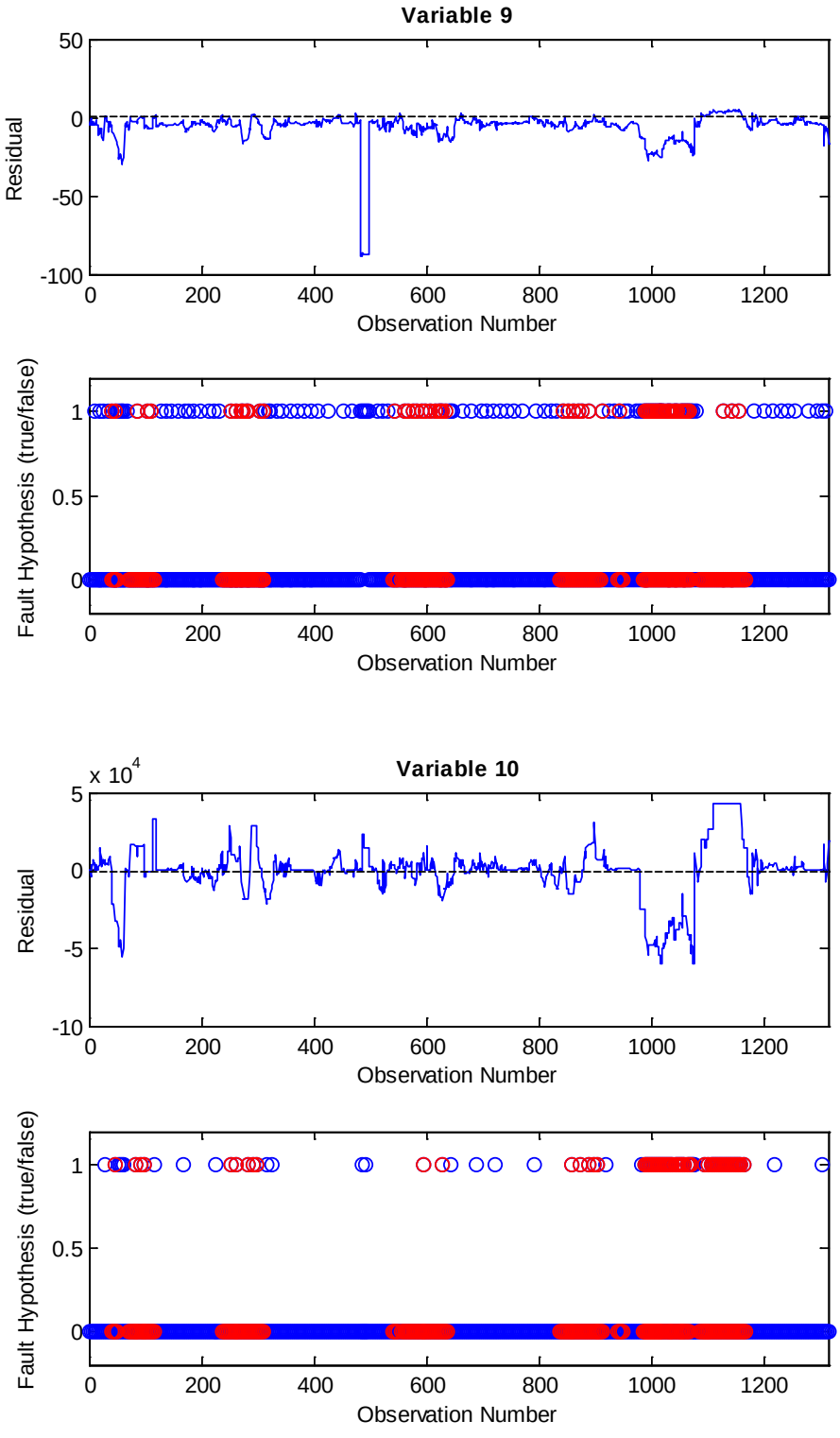

Variable 3
Residual
500
0
-500
-1000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

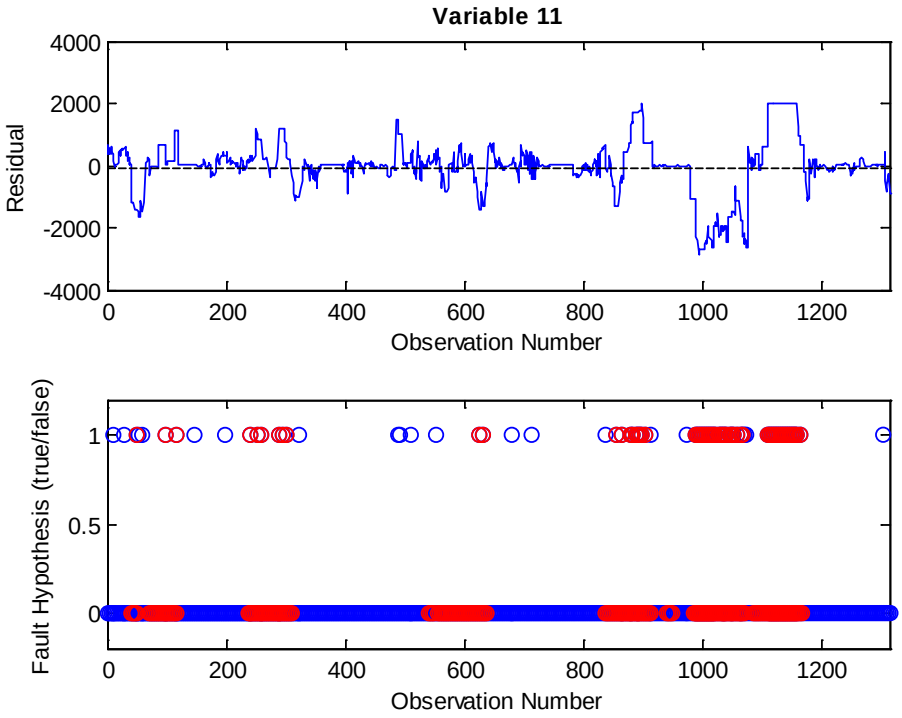

Variable 4
Residual
500
0
-500
-1000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

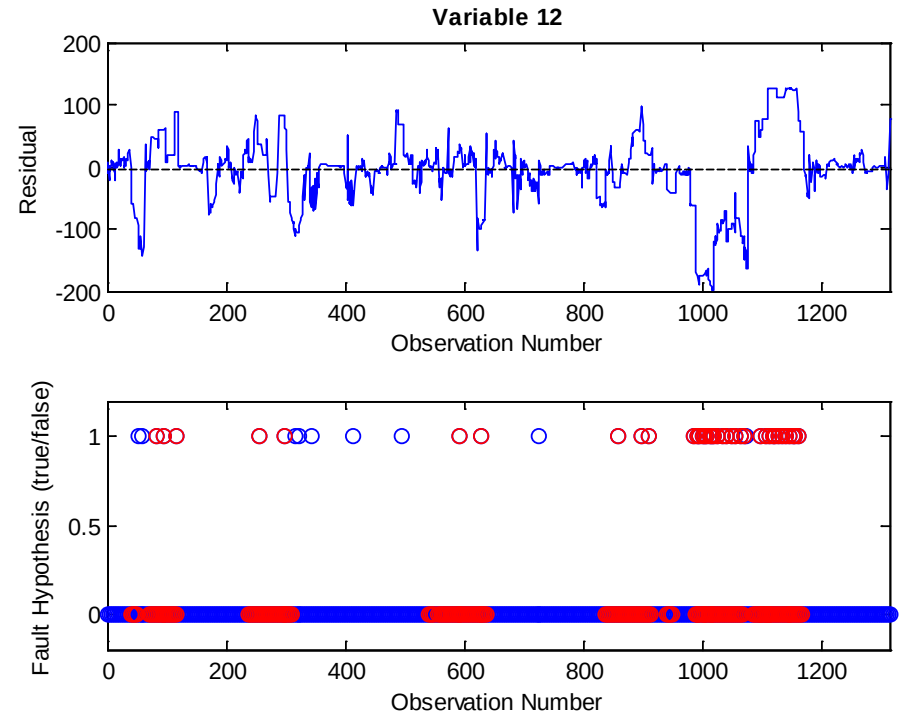

Variable 5
x 10^5
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

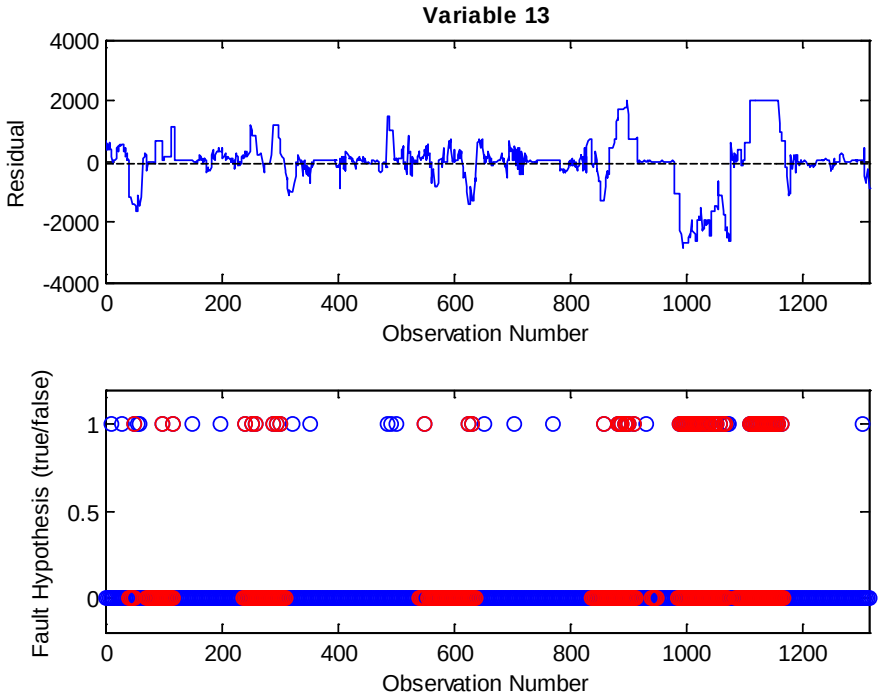

Variable 6
x 10^6
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

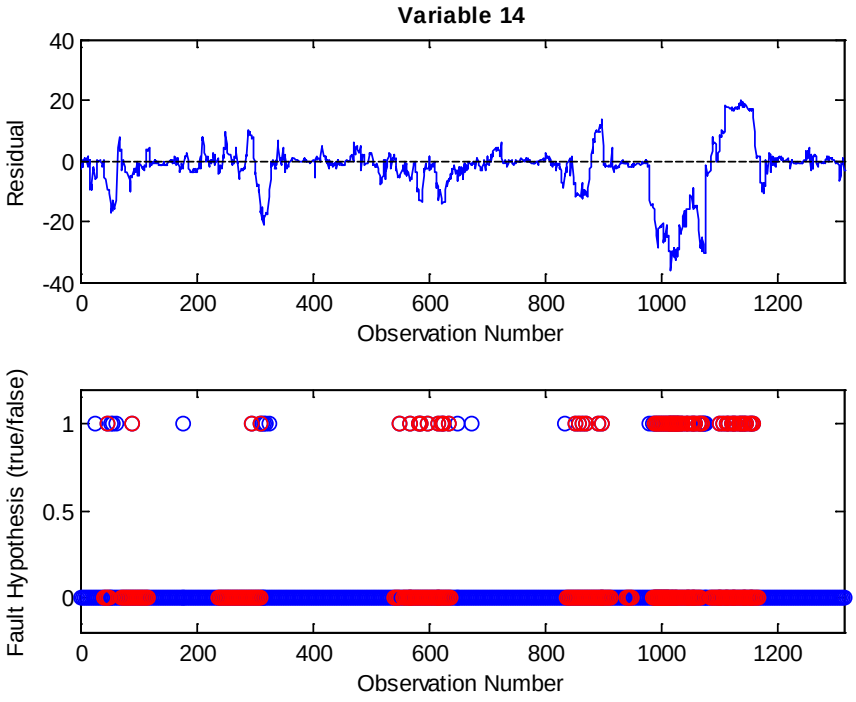

Variable 7
Residual
5000
0
-5000
-10000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number
Variable 8
Residual
40
20
0
-20
-40
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

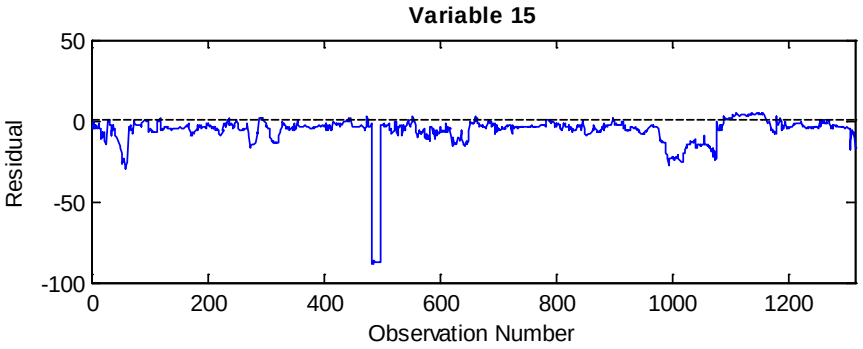

Variable 9
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number
Variable 10
x 10^4
Residual
5
0
-5
-10
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

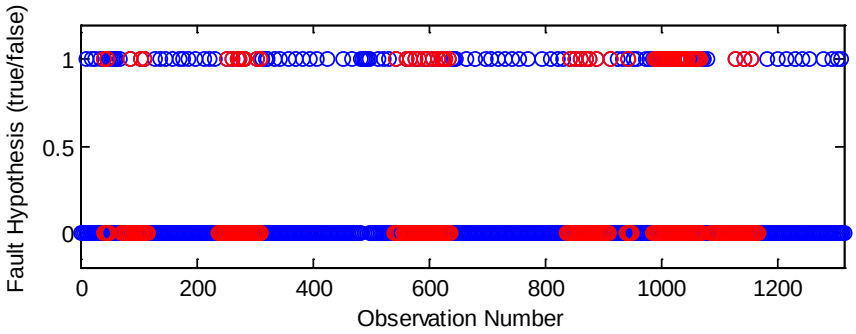

Variable 11
Residual
4000
2000
0
-2000
-4000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

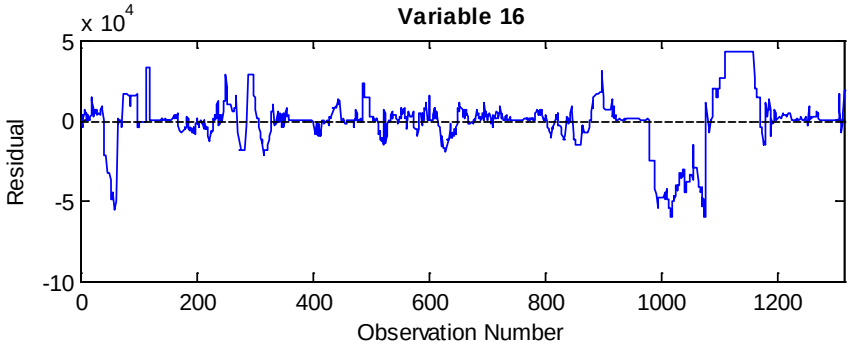

Variable 12
Residual
200
100
0
-100
-200
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

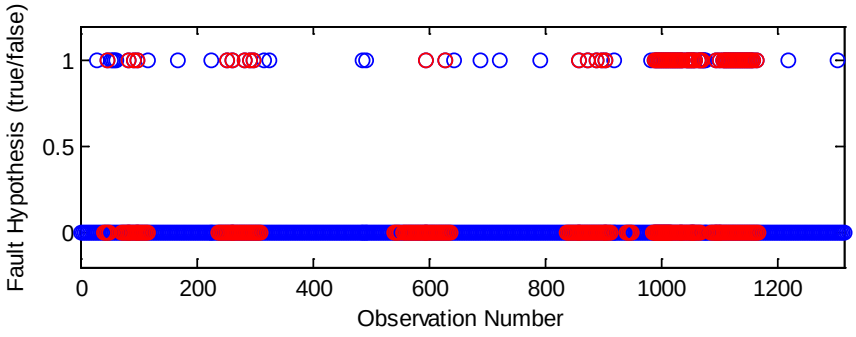

Variable 13
Residual
4000
2000
0
-2000
-4000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

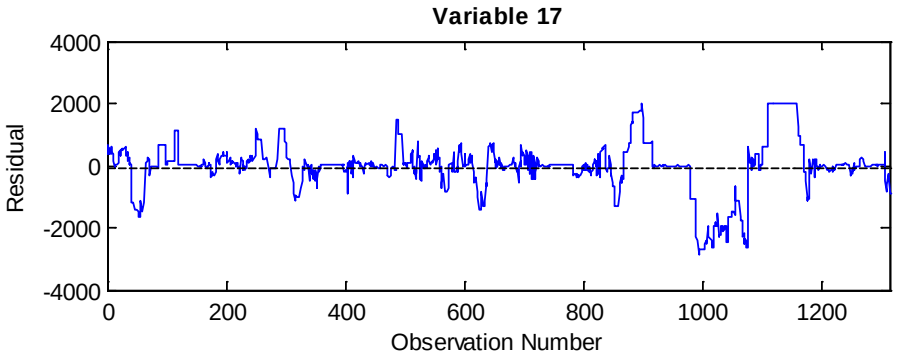

Variable 14
Residual
40
20
0
-20
-40
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

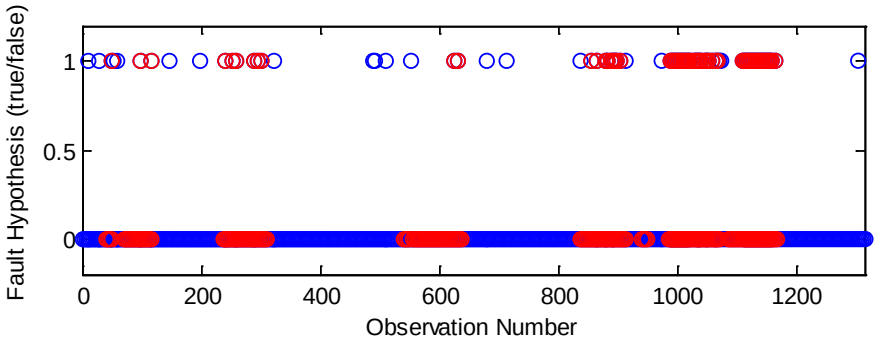

Variable 15
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number

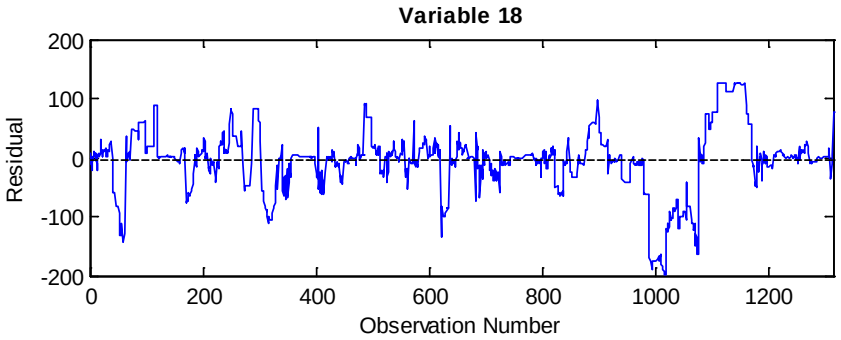

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

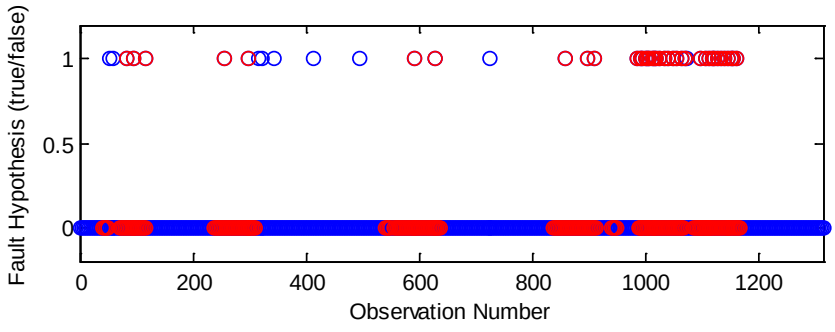

Variable 16
x 10^4
Residual
5
0
-5
-10
0
200
400
600
800
1000
1200
Observation Number

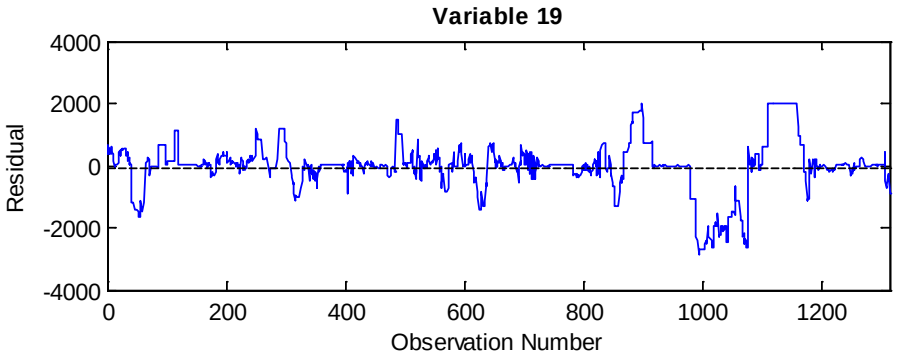

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

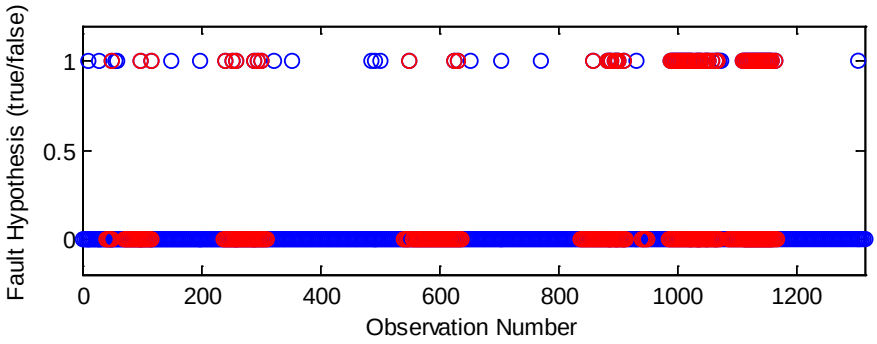

Variable 17
Residual
4000
2000
0
-2000
-4000
0
200
400
600
800
1000
1200
Observation Number

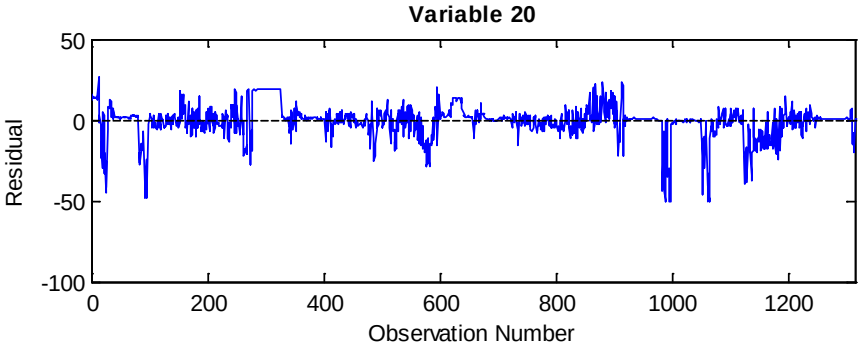

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

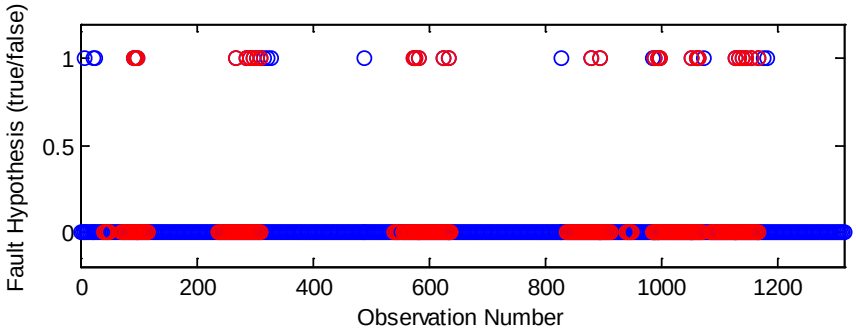

Variable 18
Residual
200
100
0
-100
-200
0
200
400
600
800
1000
1200
Observation Number

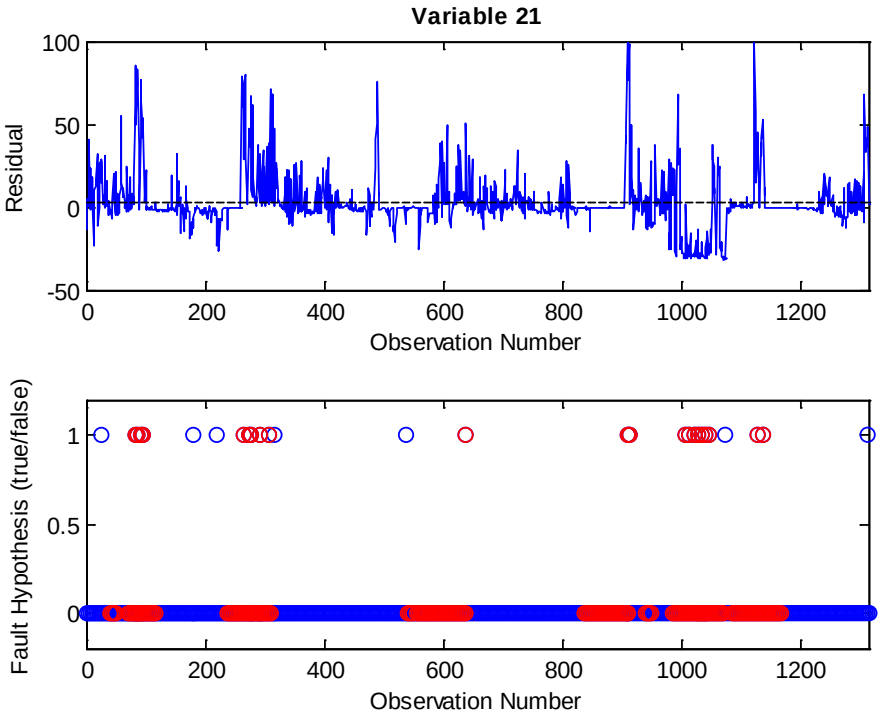

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 19
Residual
4000
2000
0
-2000
-4000
0
200
400
600
800
1000
1200
Observation Number

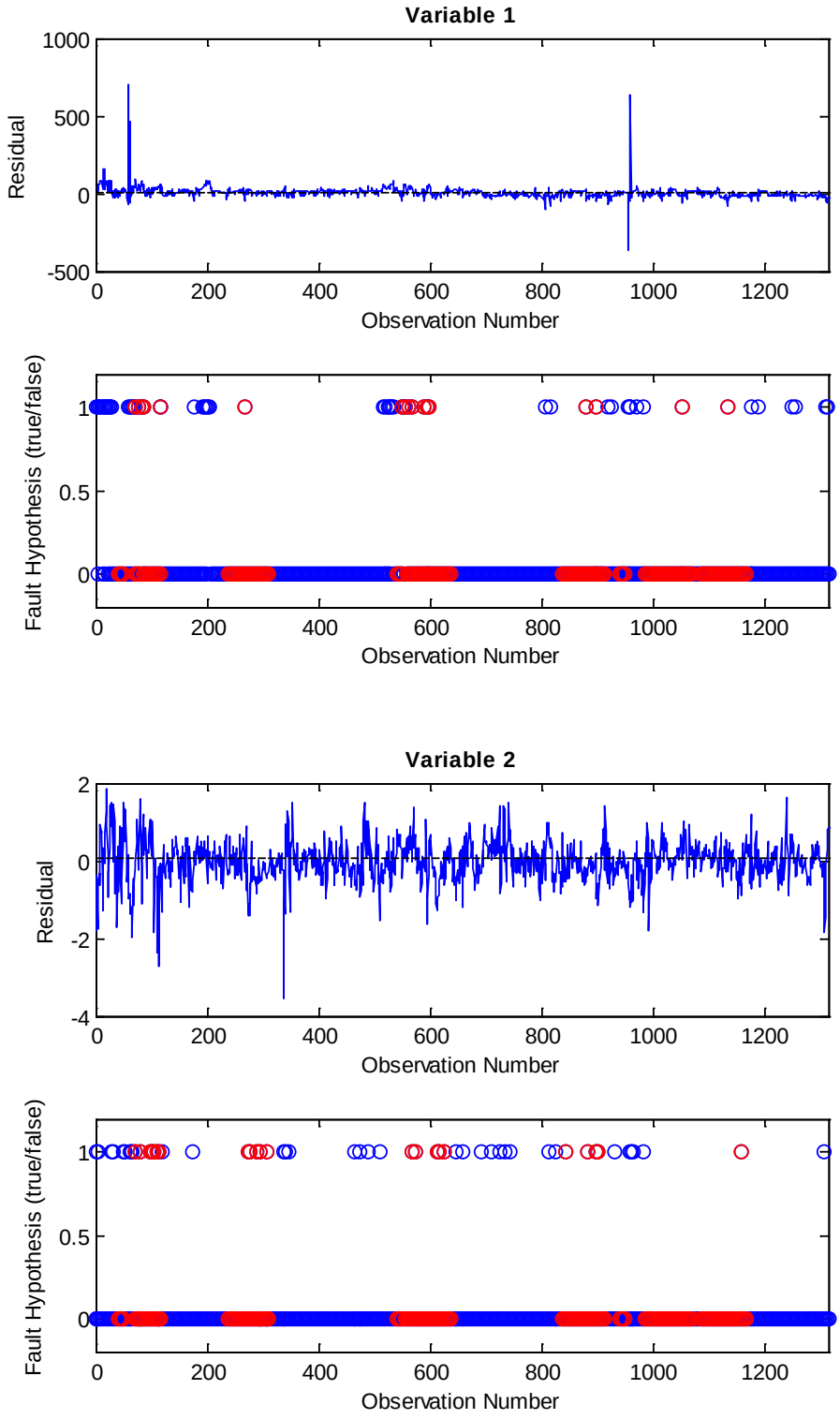

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

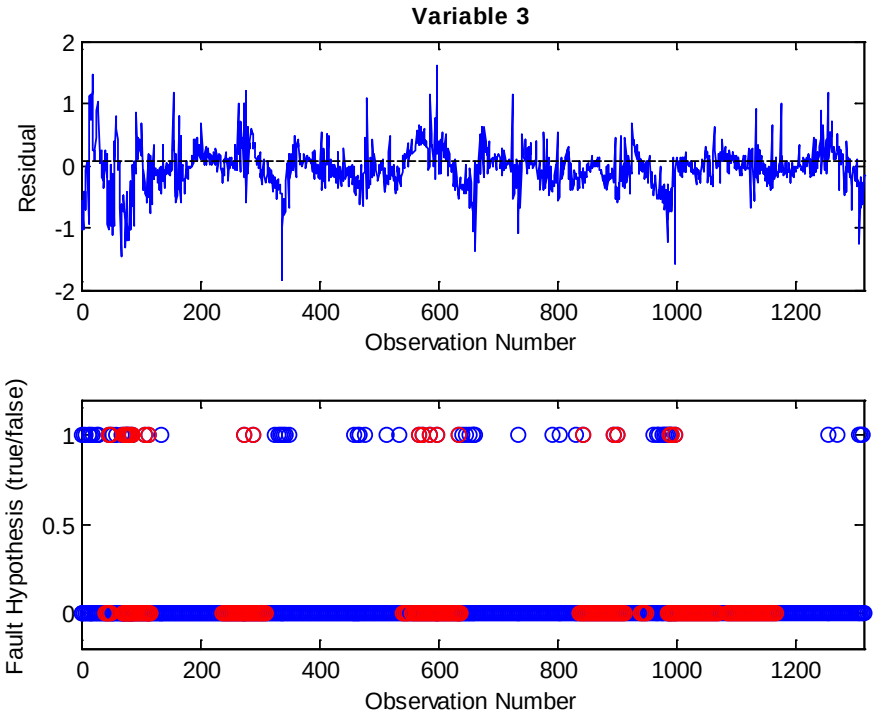

Variable 20
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number

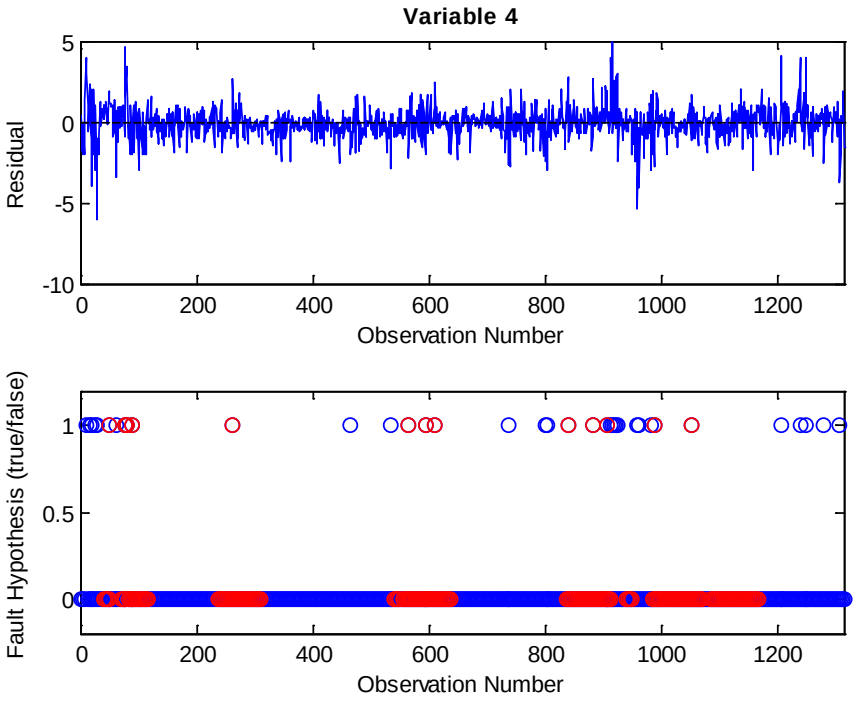

Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

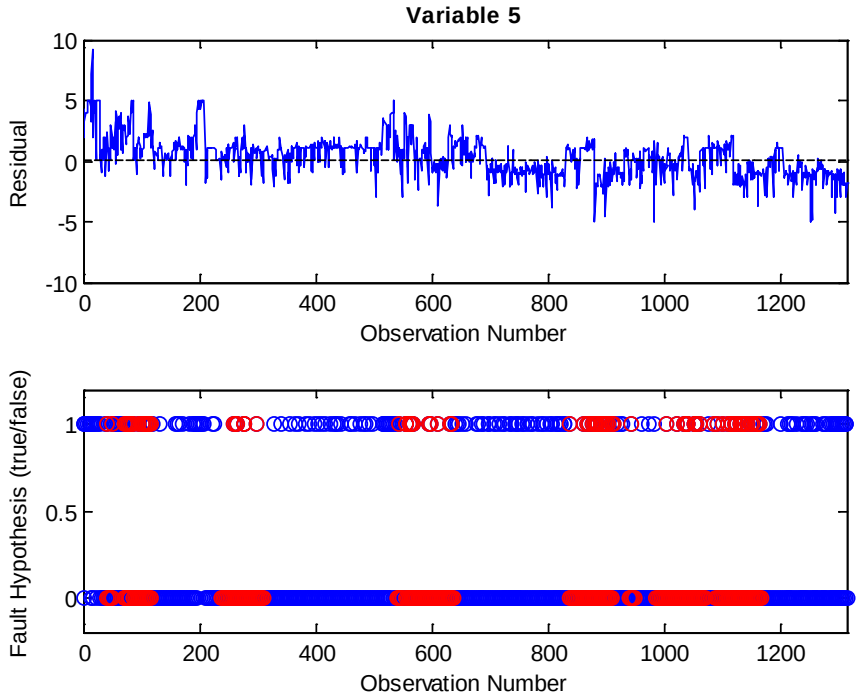

Variable 21
Residual
100
50
0
-50
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

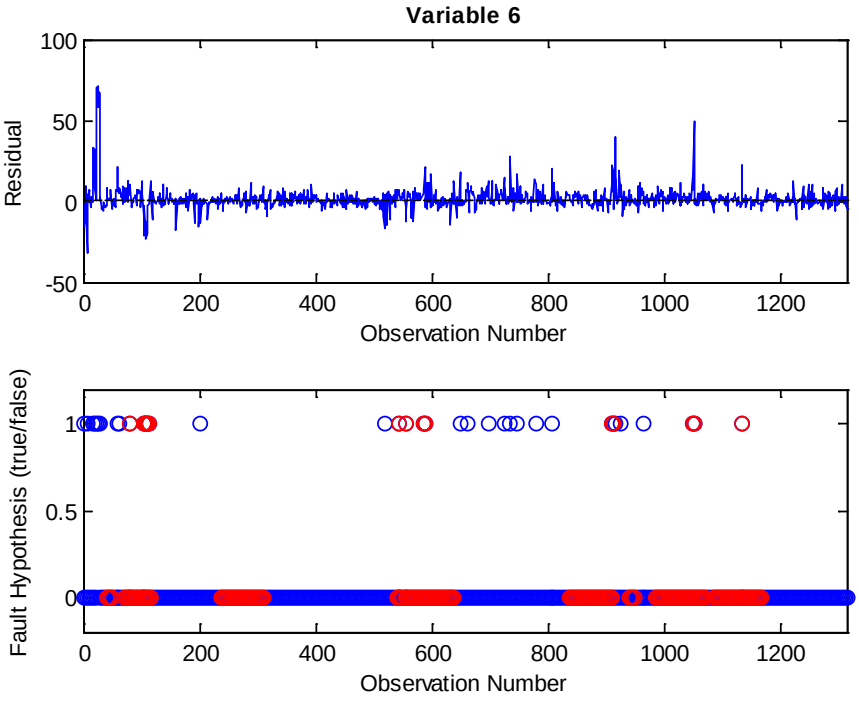
Additional Figures March Set – Model 2

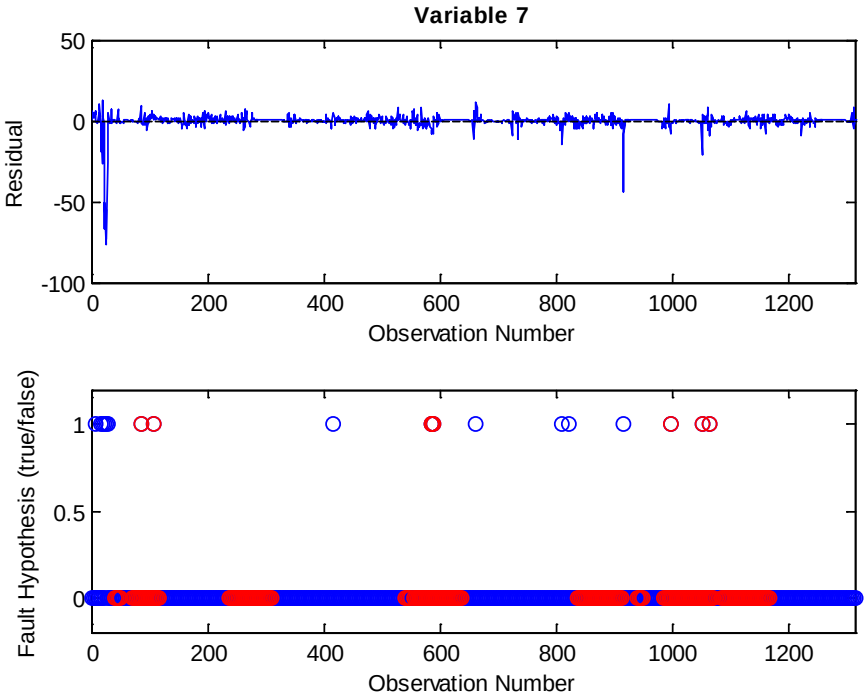


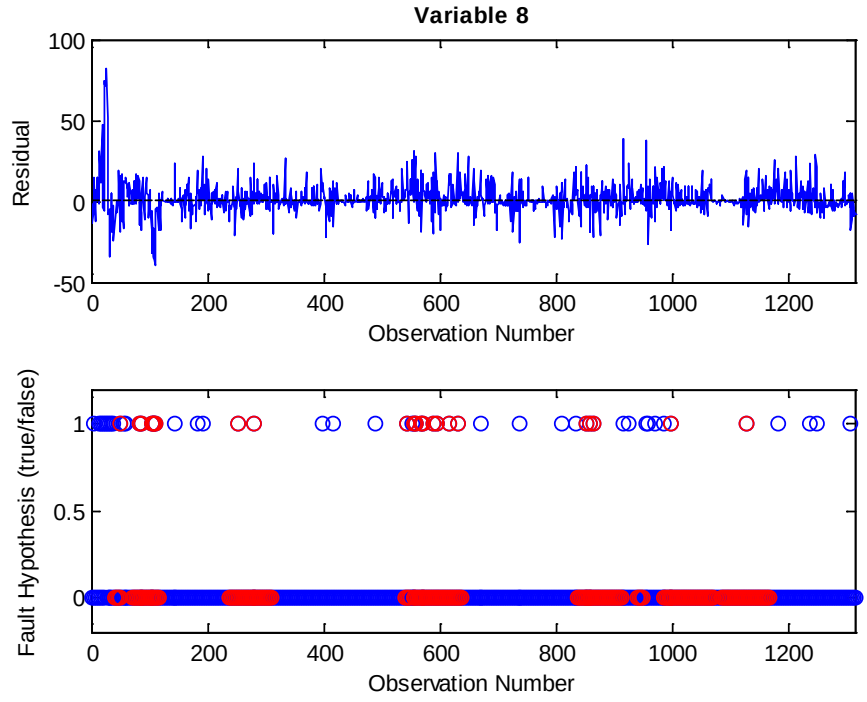

Variable 3
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

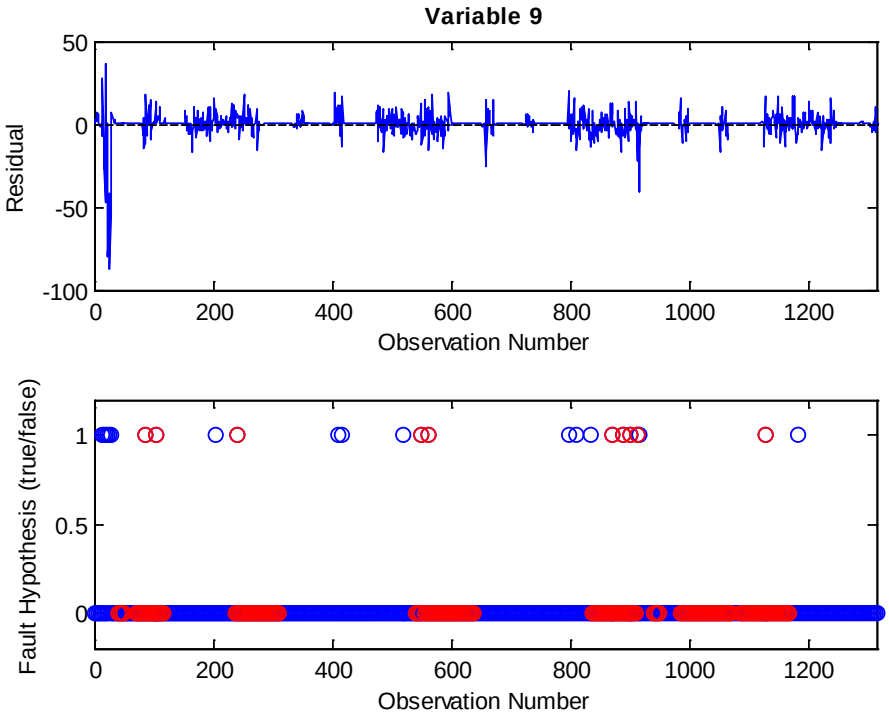

Variable 4
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

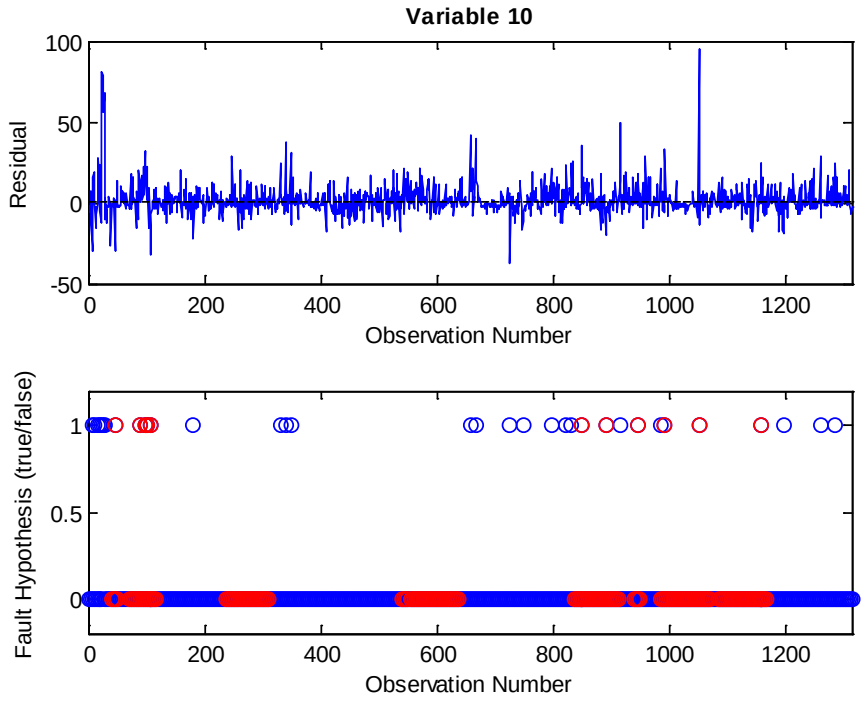

Variable 5
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

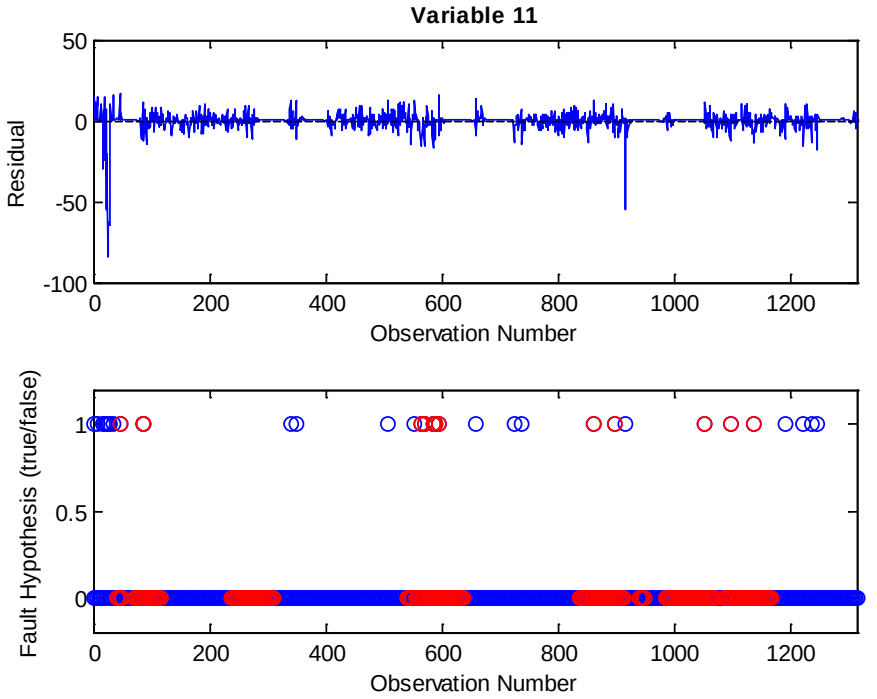

Variable 6
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

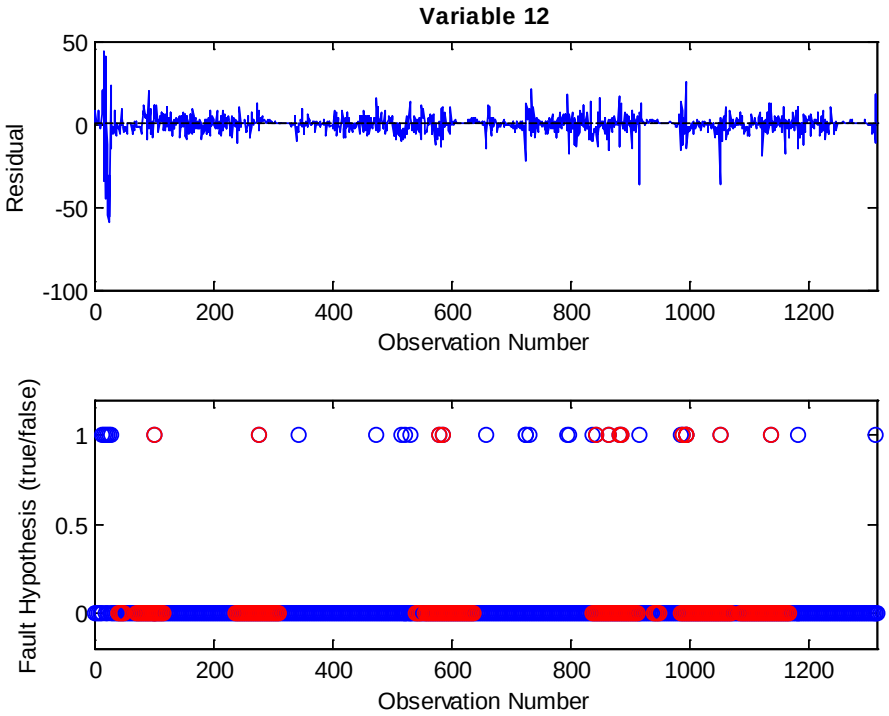

Variable 7
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 8
Residual
100
50
0
-50
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

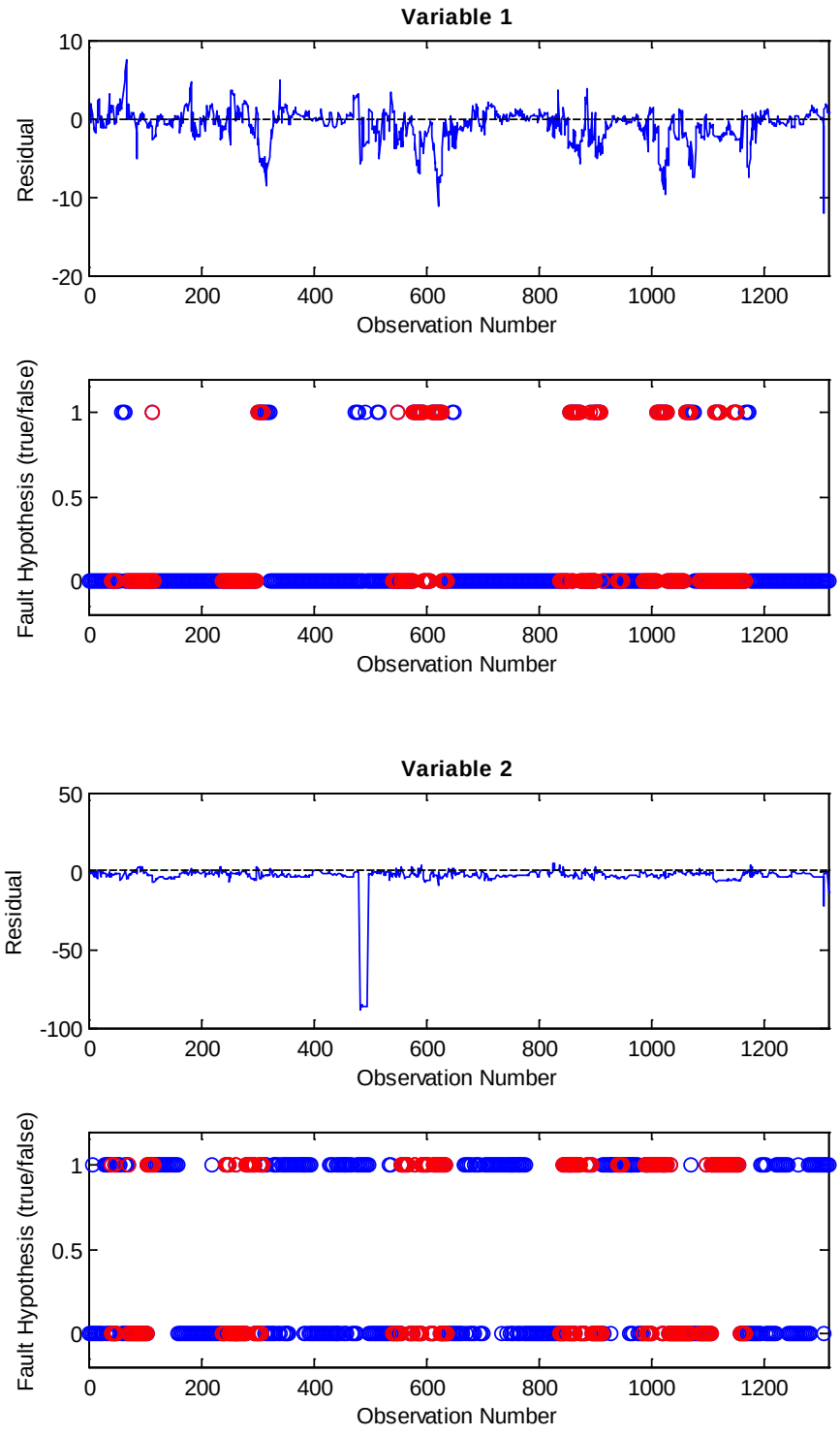

Variable 9
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

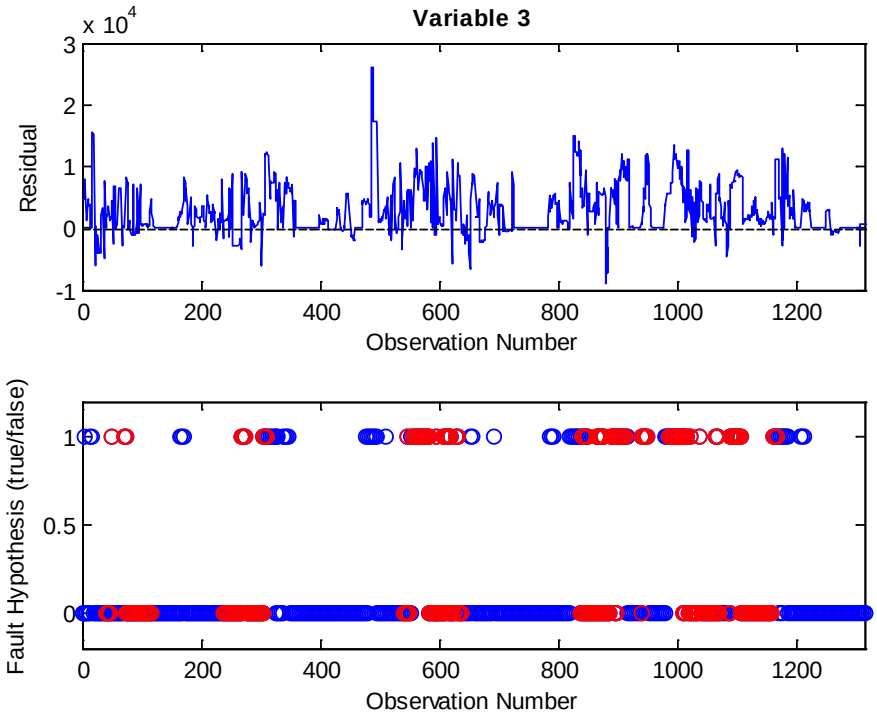

Variable 10
Residual
100
50
0
-50
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

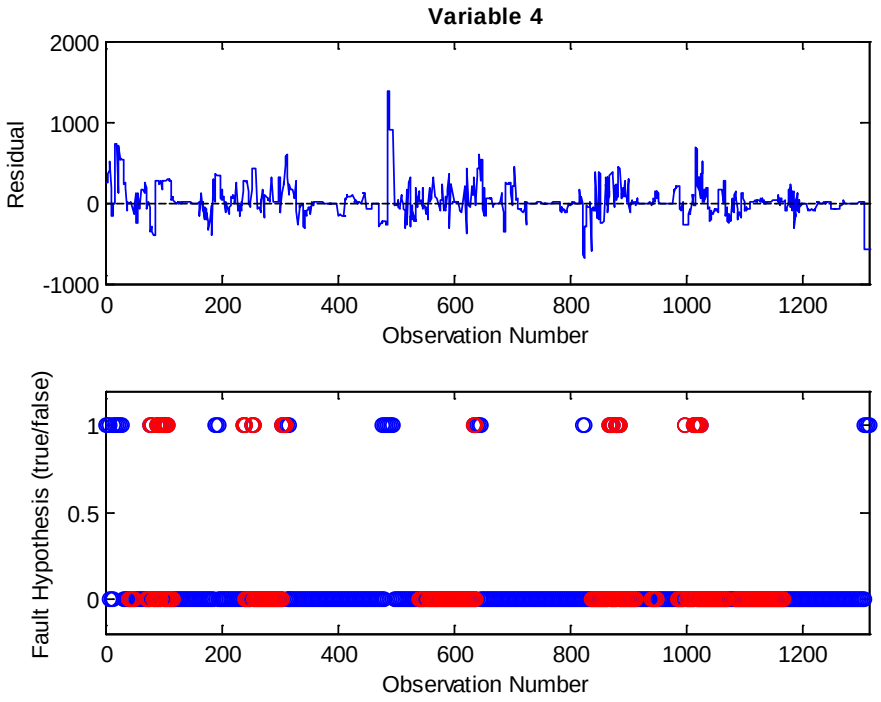

Variable 11
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

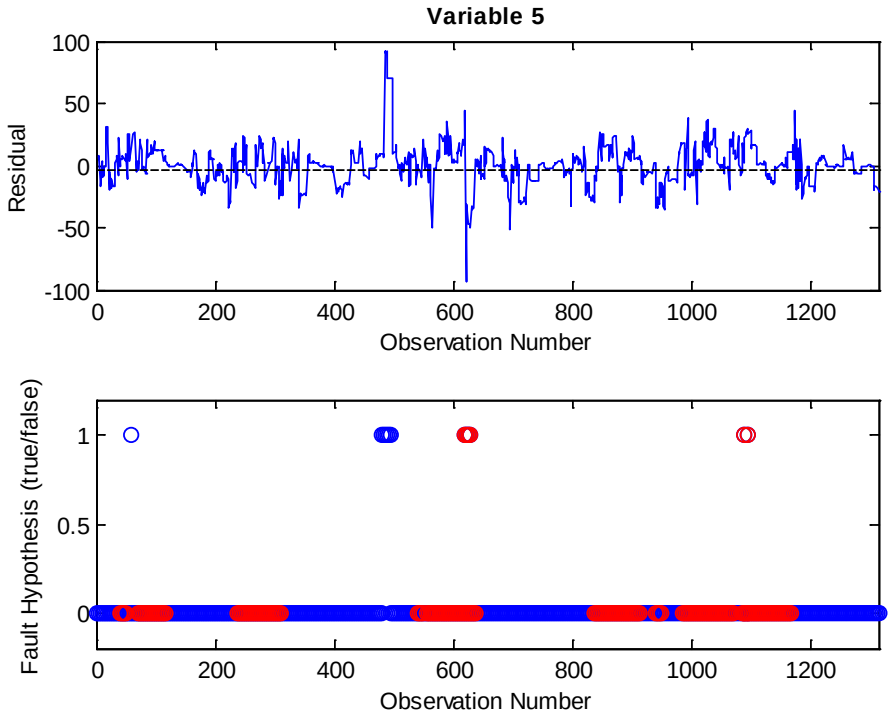

Variable 12
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

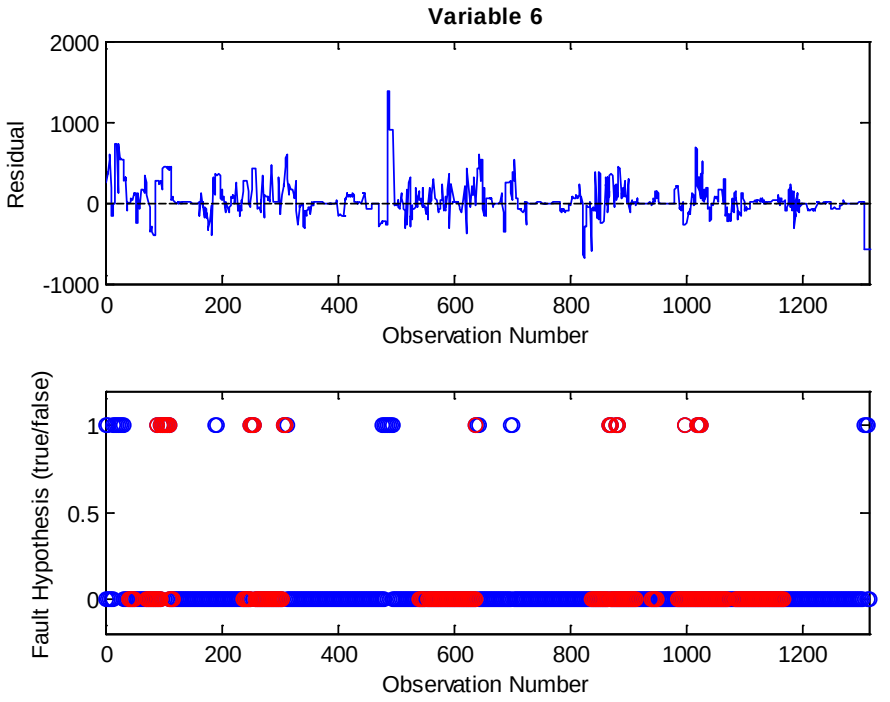
Additional Figures March Set – Model 3

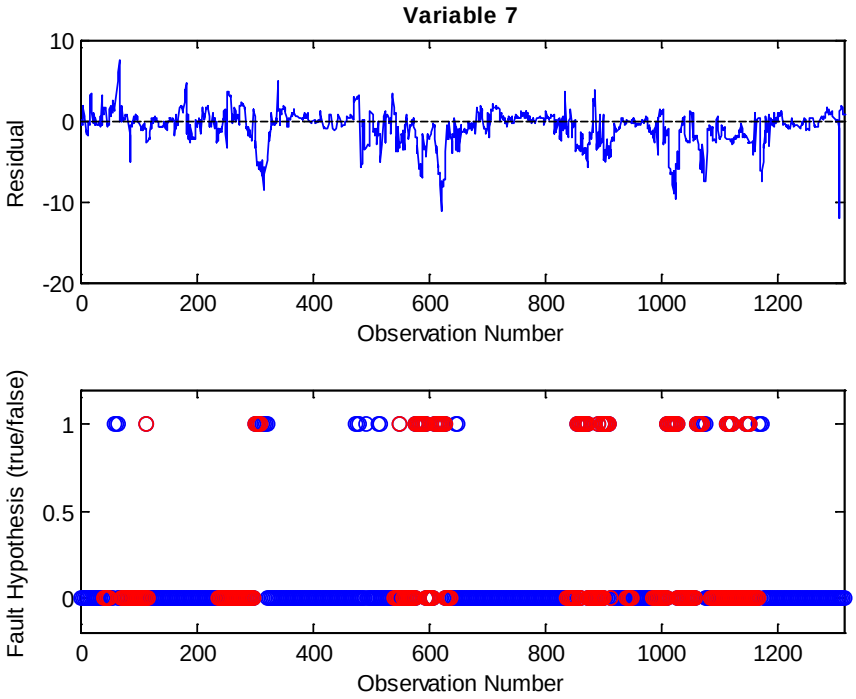


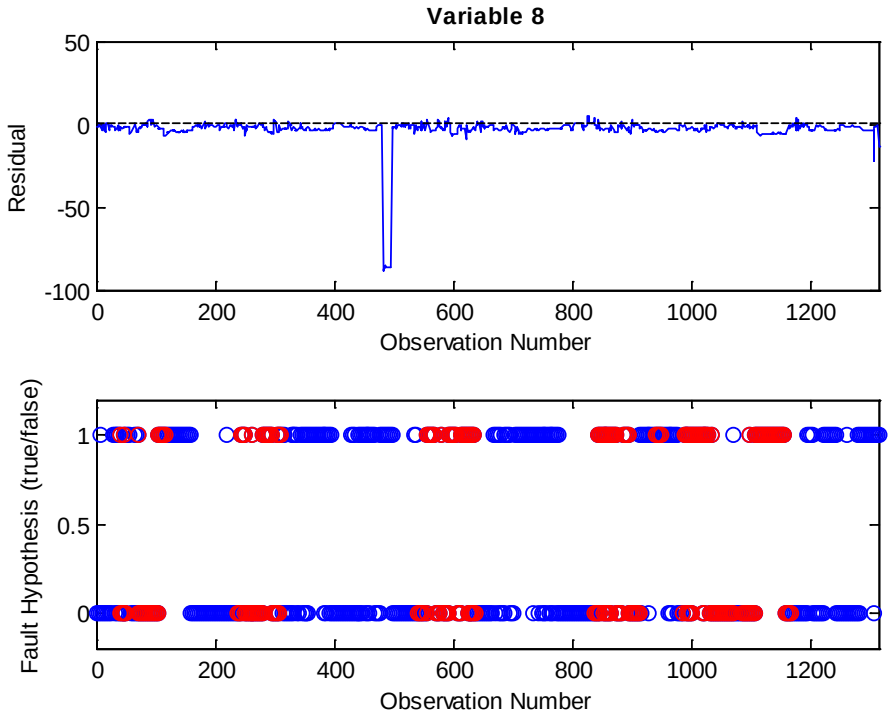

Variable 3
x 10^4
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

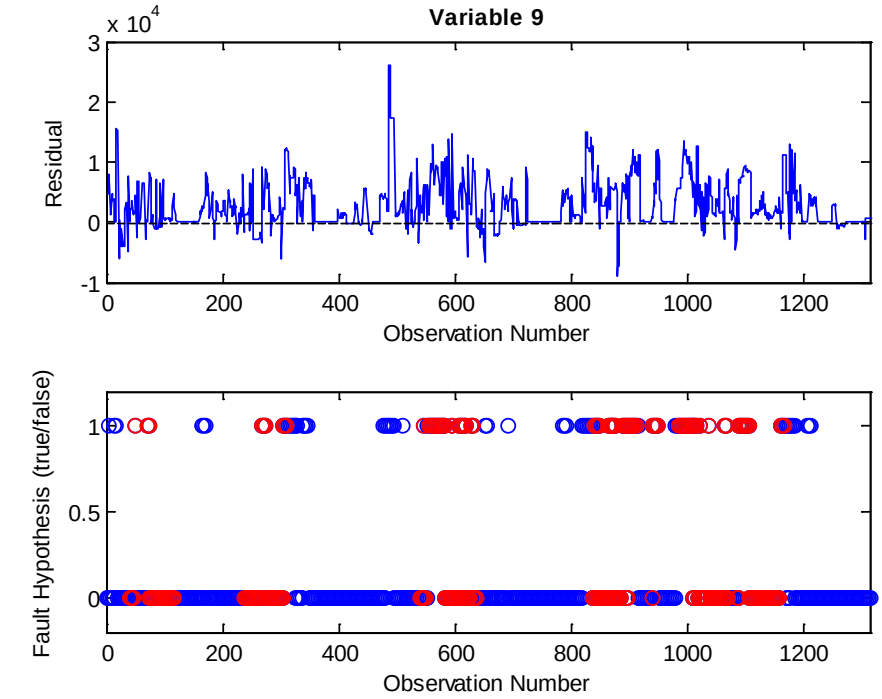

Variable 4
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

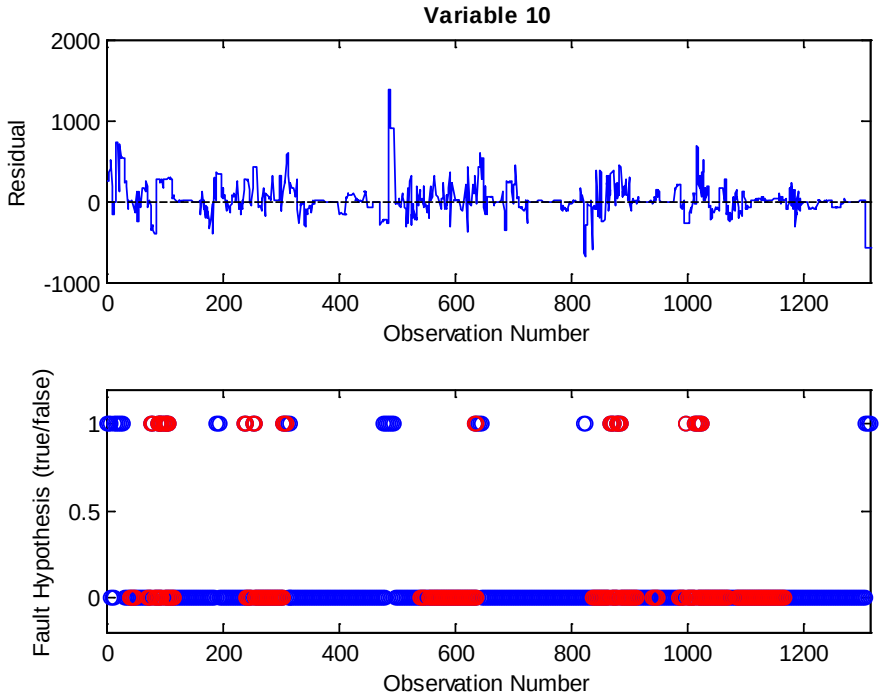

Variable 5
Residual
100
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

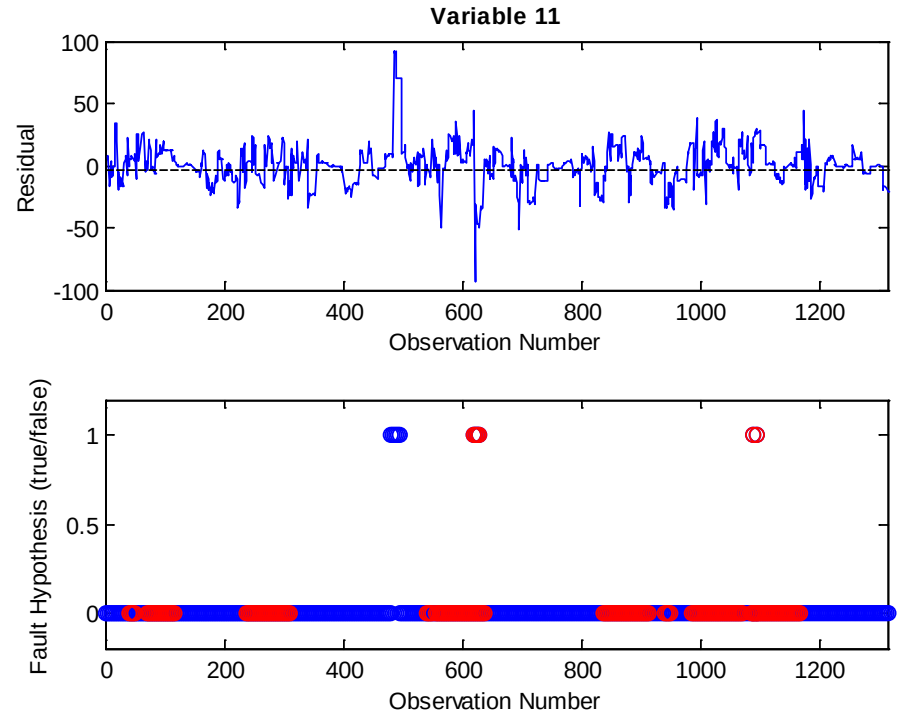

Variable 6
Residual
2000
1000
0
-1000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

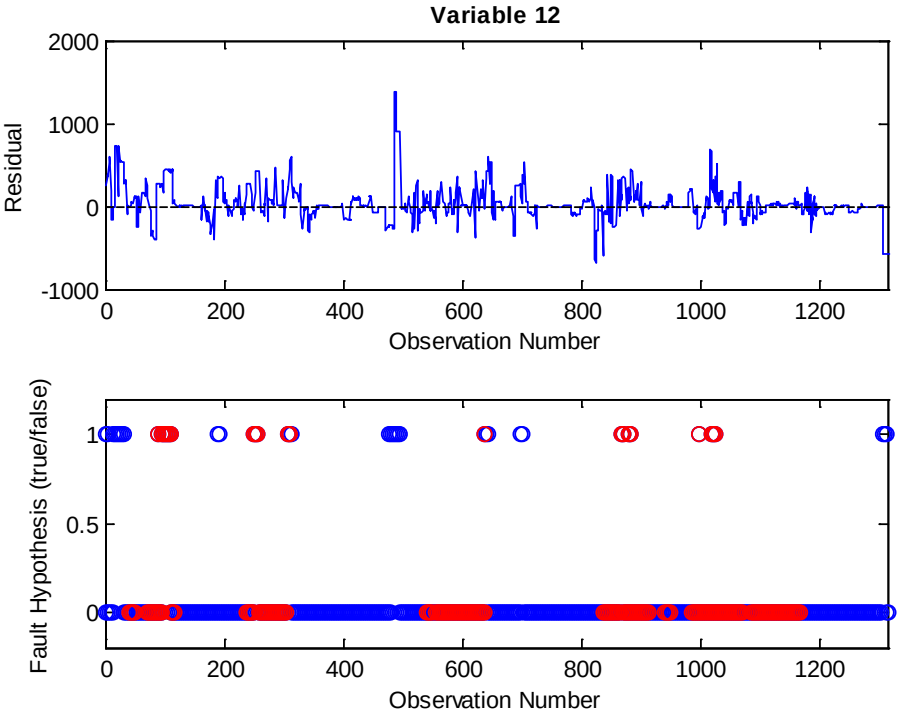

Variable 7
Residual
10
0
-10
-20
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 8
Residual
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

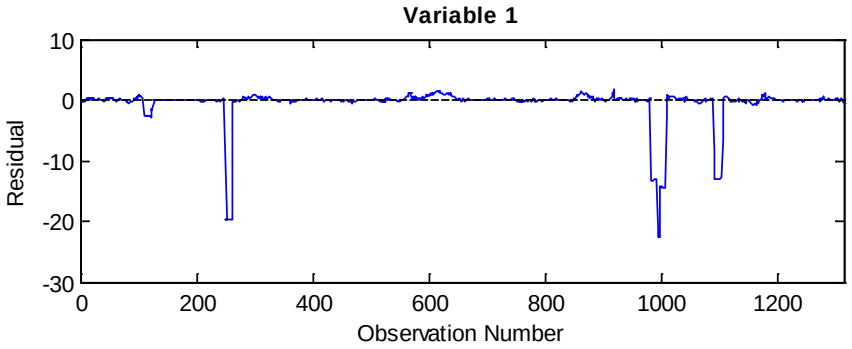

Variable 9
x 10^4
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

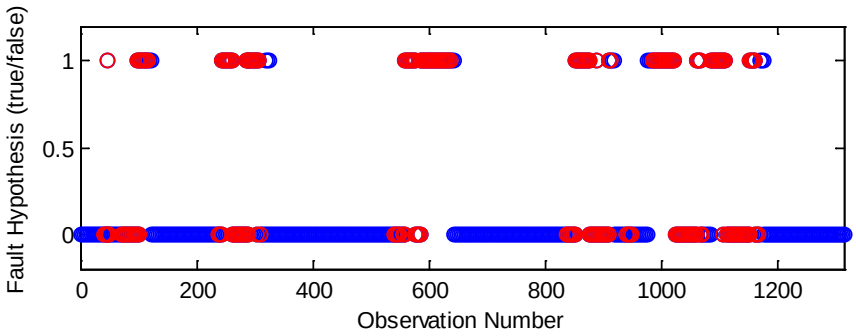

Variable 10
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number

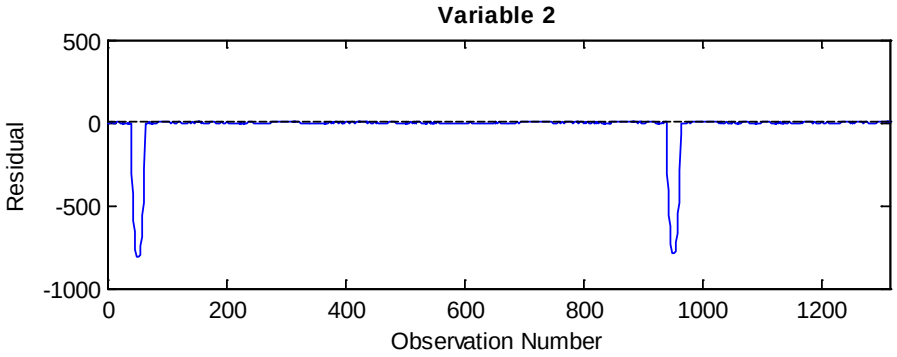

Variable 11
Residual
100
50
0
-50
-100
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

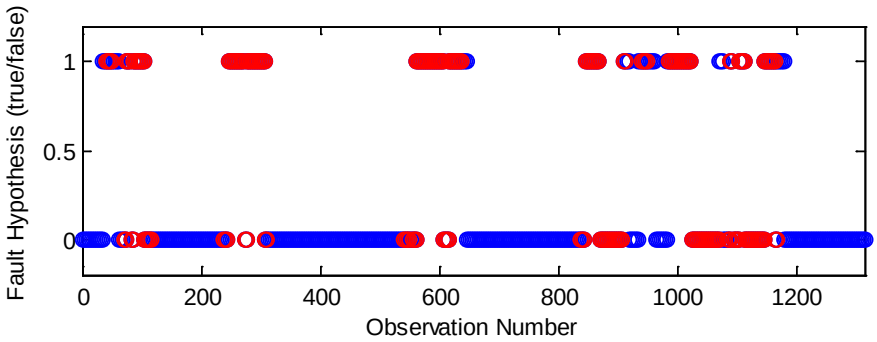

Variable 12
Residual
2000
1000
0
-1000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

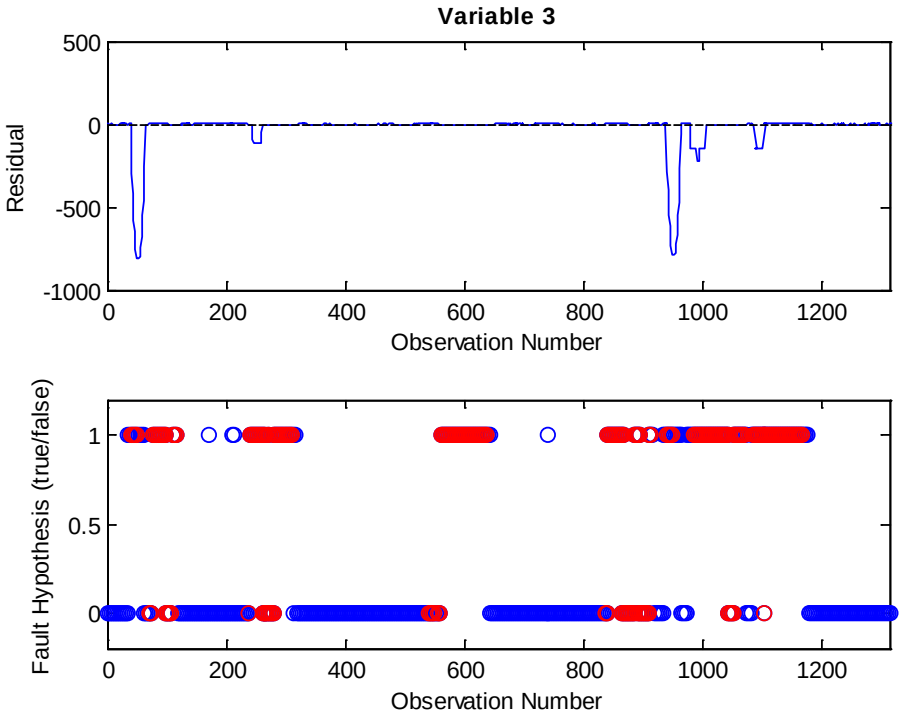
Additional Figures March Set – Model 4

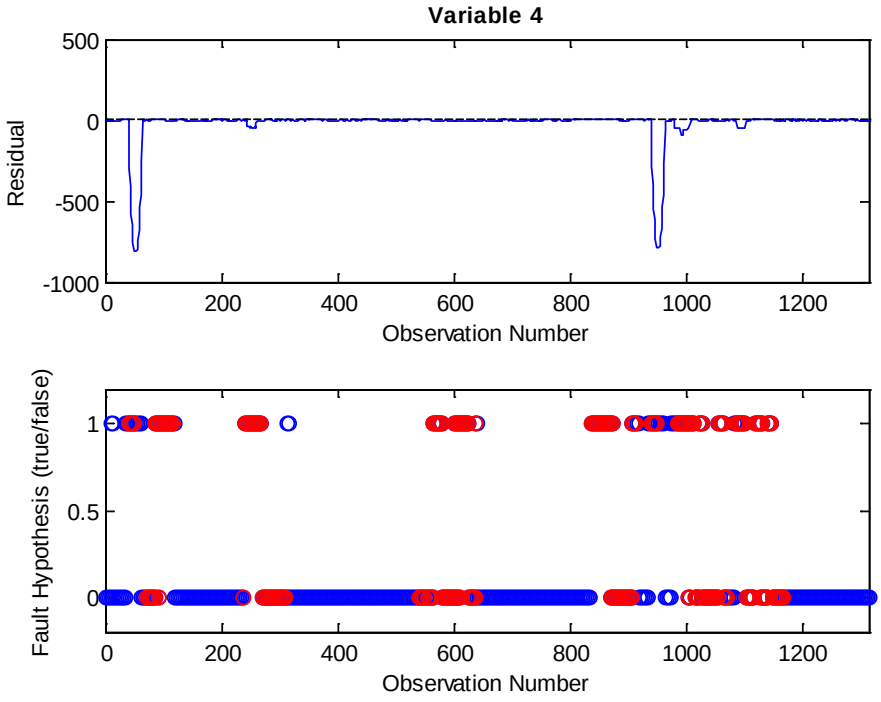


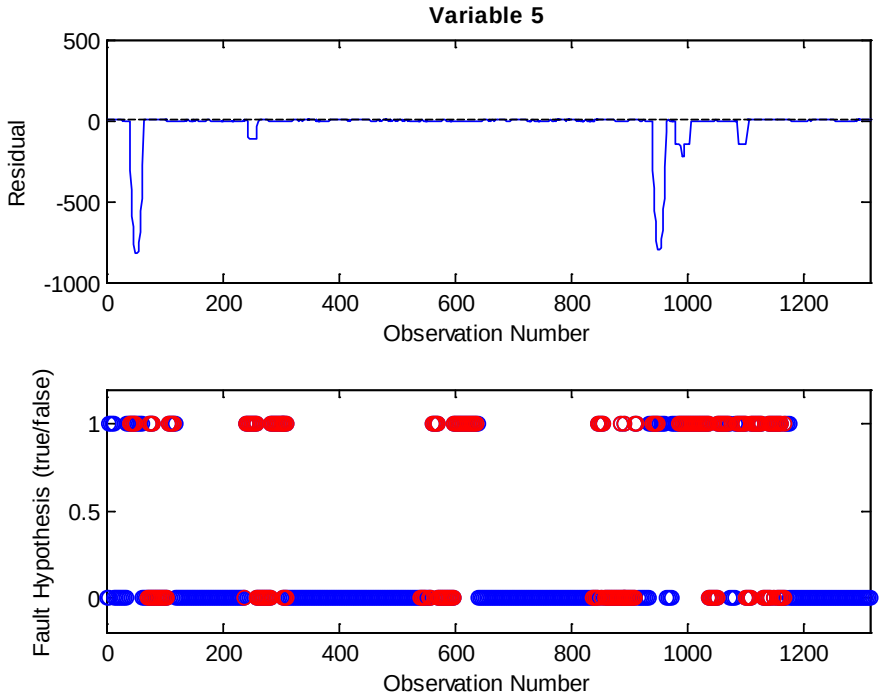


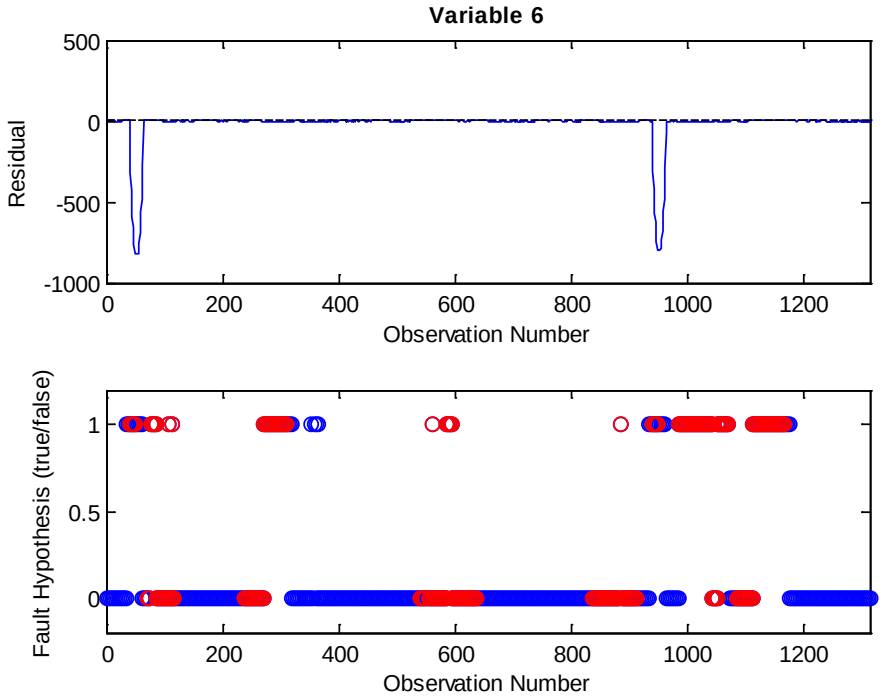


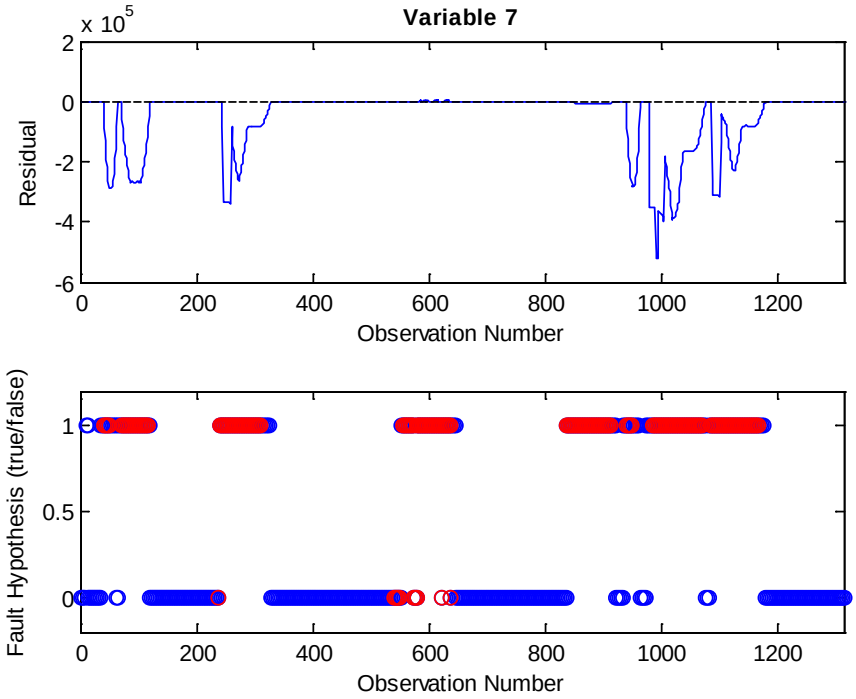


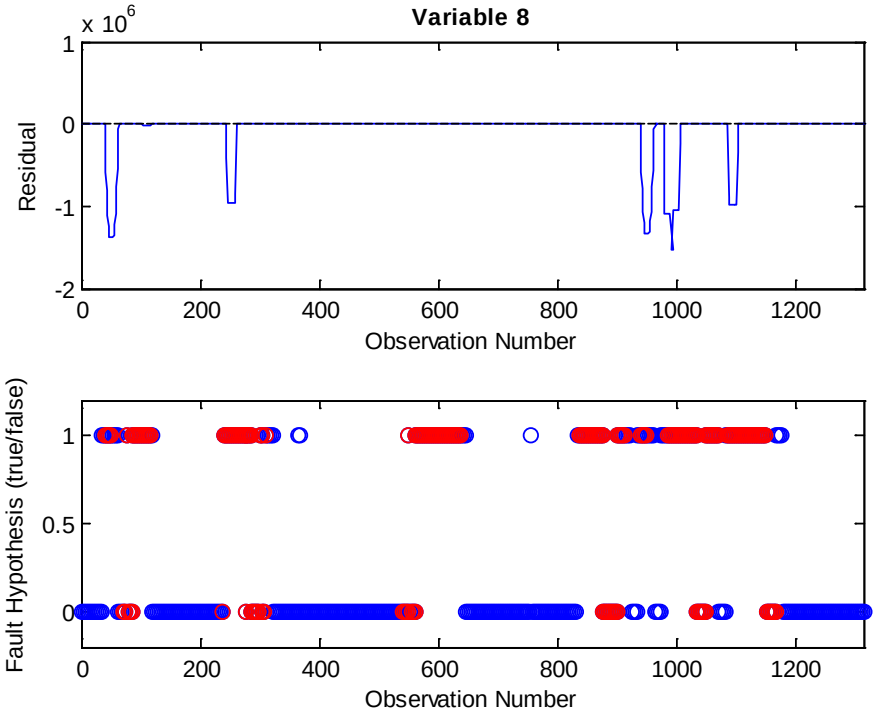

Variable 3
Residual
500
0
-500
-1000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

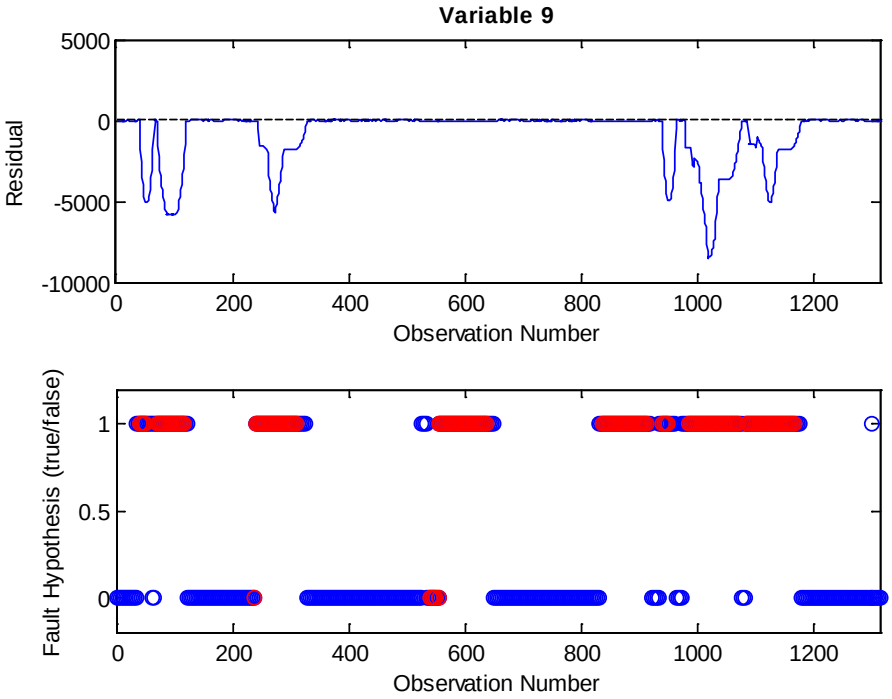

Variable 4
Residual
500
0
-500
-1000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 5
Residual
500
0
-500
-1000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 6
Residual
500
0
-500
-1000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number


Variable 7
x 10^5
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number


Variable 8
x 10^6
Residual
Observation Number
Fault Hypothesis (true/false)
Observation Number


Variable 9
Residual
5000
0
-5000
-10000
0
200
400
600
800
1000
1200
Observation Number
Fault Hypothesis (true/false)
1
0.5
0
0
200
400
600
800
1000
1200
Observation Number

Appendix E: SPRT Results – March Data Set

Model 1 AAKR SPRT Results

Int. #/Sig. #	SSH Fuzzer	Zenmap	SPARTA	Dmitry	Password	NTP Fuzzer	SSH Fuzzer	SPARTA x2	SPAR TA
1.	X	X	X	X	X	X	X	X	X
2.		X	X		X			X	X
3.	X	X	X	X	X	X	X	X	X
4.	X	X	X	X	X	X	X	X	X
5.	X	X	X	X	X	X	X	X	X
6.	X	X	X	X	X	X	X	X	X
7.		X	X			X	X		
8.	X	X	X	X	X	X	X	X	X
9.	X	X	X	X	X	X	X	X	X
10.	X	X	X	X	X	X	X	X	X
11.	X	X	X	X	X	X	X	X	X
12.	X	X	X	X	X	X	X	X	X
13.	X	X	X	X	X	X	X	X	X
14.	X	X	X	X	X	X	X	X	X
15.				X	X		X	X	X
16.		X		X				X	X
17.	X	X	X	X	X	X	X	X	X
18.	X	X	X	X	X	X	X	X	X
19.		X		X		X		X	X
20.		X				X			
21.		X	X		X	X	X		

Model 1 AAMSET SPRT Results

Int. #/Sig. #	SSH Fuzzer	Zenmap	SPARTA	Dmitry	Password	NTP Fuzzer	SSH Fuzzer	SPARTA x2	SPAR TA
1.	X		X	X	X	X	X	X	X
2.		X	X		X			X	X
3.	X	X	X	X	X	X	X	X	X
4.	X	X		X	X	X	X	X	X
5.	X	X	X	X	X	X	X	X	X
6.	X	X	X	X	X	X	X	X	X
7.		X	X			X	X		
8.	X	X	X	X	X	X	X	X	X
9.	X	X	X	X	X	X	X	X	X
10.	X	X	X	X	X	X		X	X
11.	X	X	X	X	X	X	X	X	X
12.	X	X		X	X	X	X	X	X
13.	X	X	X	X	X	X	X	X	X
14.	X	X	X	X	X	X	X	X	X
15.				X	X		X	X	X
16.		X		X				X	X
17.	X	X	X	X	X	X	X	X	X
18.	X	X	X	X	X	X	X	X	X
19.		X		X		X		X	X
20.		X				X			
21.		X			X	X	X		

Model 2 AAKR SPRT Results

Int. #/Sig. #	SSH Fuzzer	Zenmap	SPARTA	Dmitry	Password	NTP Fuzzer	SSH Fuzzer	SPARTA x2	SPAR TA
1.		X	X			X		X	X
2.	X	X					X	X	X
3.		X	X		X			X	X
4.	X	X						X	X
5.		X	X		X			X	X
6.		X						X	X
7.		X	X		X	X		X	X
8.	X	X						X	X
9.		X	X		X	X	X	X	X
10.		X						X	X
11.	X	X		X			X	X	X
12.		X	X	X				X	X

Model 2 AAMSET SPRT Results

Int. #/Sig. #	SSH Fuzzer	Zenmap	SPARTA	Dmitry	Password	NTP Fuzzer	SSH Fuzzer	SPARTA x2	SPAR TA
1.	X	X	X	X		X		X	X
2.	X	X					X	X	X
3.			X	X	X			X	X
4.	X	X		X				X	X
5.		X	X		X		X	X	X
6.				X				X	X
7.		X	X		X	X		X	X
8.	X	X						X	X
9.			X		X		X	X	X
10.		X				X		X	X
11.	X	X		X	X	X	X	X	X
12.		X	X	X	X			X	X

Model 3 AAKR SPRT Results

Int. #/Sig. #	SSH Fuzzer	Zenmap	SPARTA	Dmitry	Password	NTP Fuzzer	SSH Fuzzer	SPARTA x2	SPAR TA
1.	X	X	X	X	X	X	X	X	X
2.	X	X					X	X	X
3.	X	X	X	X	X	X	X	X	X
4.	X	X		X				X	X
5.	X	X	X	X	X	X	X	X	X
6.				X				X	X
7.	X	X	X	X	X	X	X	X	X
8.	X	X						X	X
9.			X		X		X	X	X
10.	X	X	X	X	X	X		X	X
11.	X	X	X	X	X	X	X	X	X
12.	X	X	X	X	X	X	X	X	X

Model 3 AAKR SPRT Results

Int. #/Sig. #	SSH Fuzzer	Zenmap	SPARTA	Dmitry	Password	NTP Fuzzer	SSH Fuzzer	SPARTA x2	SPAR TA
1.	X	X	X	X	X	X	X	X	X
2.	X	X					X	X	X
3.	X	X	X	X	X	X	X	X	X
4.	X	X		X				X	X
5.		X	X		X		X	X	X
6.				X				X	X
7.		X	X		X	X		X	X
8.	X	X						X	X
9.	X	X	X	X	X	X	X	X	X
10.		X				X		X	X
11.	X	X		X	X	X	X	X	X
12.	X	X	X	X	X	X	X	X	X

Model 4 AAKR SPRT Results

Int. #/Sig. #	SSH Fuzzer	Zenmap	SPARTA	Dmitry	Password	NTP Fuzzer	SSH Fuzzer	SPARTA x2	SPAR TA
1.	X	X	X	X	X	X	X	X	X
2.	X	X	X	X	X	X	X	X	X
3.	X	X	X	X	X	X	X	X	X
4.	X	X		X				X	X
5.	X	X	X	X	X	X	X	X	X
6.				X				X	X
7.	X	X	X	X	X	X	X	X	X
8.	X	X	X	X	X	X	X	X	X
9.	X	X	X	X	X	X	X	X	X

Model 4 AAMSET SPRT Results

Int. #/Sig. #	SSH Fuzzer	Zenmap	SPARTA	Dmitry	Password	NTP Fuzzer	SSH Fuzzer	SPARTA x2	SPAR TA
1.	X	X	X	X	X	X	X	X	X
2.	X	X	X	X	X	X	X	X	X
3.	X	X	X	X	X	X	X	X	X
4.	X	X	X	X	X	X	X	X	X
5.	X	X	X	X	X	X	X	X	X
6.				X				X	X
7.		X	X		X	X		X	X
8.	X	X	X	X	X	X	X	X	X
9.	X	X	X	X	X	X	X	X	X

VITA

Brien Alen Jeffries was born and raised in Michigan. He earned a Bachelor's degree in Nuclear Engineering from the University of Tennessee in 2010. After this, he worked with Dr. J Wesley Hines as a graduate student. He spent time working on accelerated aging test beds and simulations of various industrial plants. He earned a Master's degree in Nuclear Engineering in 2012. During this time, he was also able to earn two 6 month merit-based engineering related research contracts from Alstom Power.

In 2017, he earned a PhD in Nuclear Engineering. His dissertation topic focused on developing empirical-based modeling methods for cyber-security applications of industrial systems. During this time, he was also able to earn two 1 year merit-based cyber-security related research contracts from Oracle.