3-22-2012

Towards Quantifying Programmable Logic Controller Resilience Against Intentional Exploits

Henry W. Bushey

TOWARDS QUANTIFYING PROGRAMMABLE LOGIC CONTROLLER RESILIENCE AGAINST INTENTIONAL EXPLOITS

THESIS

Henry W. Bushey, Captain, USAF

AFIT/GCO/ENG/12-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCO/ENG/12-03

TOWARDS QUANTIFYING PROGRAMMABLE LOGIC CONTROLLER RESILIENCE AGAINST INTENTIONAL EXPLOITS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Henry W. Bushey, B.S.E.E.

Captain, USAF

March 2012

AFIT/GCO/ENG/12-03

TOWARDS QUANTIFYING PROGRAMMABLE LOGIC CONTROLLER RESILIENCE AGAINST INTENTIONAL EXPLOITS

Henry W. Bushey, B.S.E.E.
Captain, USAF

Approved:

//SIGNED//_____	___5Mar2012
Maj Jonathan Butts, Ph.D. (Chairman)	Date
//SIGNED//_____	___2Mar2012
Mr. Juan Lopez, Jr. (Member)	Date
//SIGNED//_____	___2Mar2012
Dr. Robert Mills, Ph.D. (Member)	Date

Abstract

Supervisory Control and Data Acquisition (SCADA) systems control and monitor services for the nation's critical infrastructure. Recent cyber induced events (e.g., Stuxnet) provide an example of a targeted, covert cyber attack against a SCADA system that resulted in physical effects. Of particular note is how Stuxnet exploited the trust relationship between the human machine interface (HMI) and programmable logic controllers (PLCs). Current methods for validating system operating parameters rely on message exchange and network communications protocols, generally observed at the HMI. Although sufficient at the macro level, this method does not provide detection of malware that exhibits physical effects via covert manipulation of the PLC, as demonstrated by Stuxnet. In this research, an alternative method that leverages direct analysis of PLC input and output to derive the true state of SCADA end-devices is introduced. The behavioral input-output characteristics are modeled using Petri nets to derive metrics for quantifying resilient properties of systems against malicious exploits. The results yield metrics that are applicable towards quantifying resilience in PLCs and implementing real-time security solutions. These findings enable detecting programming changes that affect input and output relationships, identifying the degree of deviation from a baseline program, and minimizing performance losses against disruptive events.

Acknowledgments

I would like to thank my advisor Maj Jonathan Butts for his insightful guidance throughout the entirety of my thesis development. I would also like to thank my committee members, Lt Col David Robinson, Dr. Robert Mills, and Mr. Juan Lopez for exposing me to their wealth of knowledge during my research. A warm thank you is also extended to my classmates who journeyed through the GCO program alongside me. Finally, I'd like to acknowledge a loving appreciation to my wife and children who support me through everything.

Table of Contents

List of Figures

List of Tables

List of Equations

TOWARDS QUANTIFYING PROGRAMMABLE LOGIC CONTROLLER RESILIENCE AGAINST INTENTIONAL EXPLOITS

I. Introduction

This chapter provides an overview of this research. Section 1.1 introduces the motivation; Section 1.2 provides the research goals; Section 1.3 describes an overview of the research approach; Section 1.4 lists key assumptions and limitations; and Section 1.5 outlines the thesis organization.

1.1 Motivation

Supervisory Control and Data Acquisition (SCADA) systems provide automated control and monitoring for the nation's critical infrastructure. Implemented in many industry sectors as early as the 1960's, security was not initially a priority for SCADA design and development; however, recent intentional and unintentional events have highlighted concerns associated with SCADA security (Stouffer, 2008). Non-intentional events have traditionally been addressed with redundant and fault tolerant architectures. However, current solutions for intentional malicious actions are not sufficient for addressing the threat.

A primary risk factor associated with intentional malicious events is the trend to incorporate business enterprise networks for cost saving purposes. Indeed, interconnecting critical systems via LAN and WAN technologies enables entry points for attacks via the Internet, internal workstations, or communication links between the control center and field sites (Stouffer, 2008). As demonstrated by Stuxnet, an attack can

propagate via the enterprise network to execute code on field devices that results in physical damage to the underlying system (Falliere, 2011).

Stuxnet is a recent example of an intentional malicious cyber event. Stuxnet targeted a specific programmable logic controller (PLC) manufacturer and configuration. PLCs control physical end-devices (e.g., sensors, pumps, motors, valves) at the edge of SCADA systems. Stuxnet functionally alters the PLC's parameters such that specific drive motors were driven beyond nominal specifications (Falliere, 2011). Additionally, Stuxnet masks modification of the PLCs functions from the SCADA system operator. Stuxnet demonstrates a novel threat to SCADA security since it both altered physical parameters to the system and concealed the modifications.

The Stuxnet example demonstrates SCADA systems are vulnerable to rootkit-like exploits. Current methods of validating the functional parameters of a PLC primarily consider the message exchange and network communications protocols, generally observed at the human machine interface (HMI). Although sufficient at the macro level, this method does not provide detection of malware which exhibits physical effects and masks the operations from the HMI or communication channel.

Establishing a resilient SCADA system can help mitigate risks associated with malicious exploits. Resiliency requires that a system be self-aware, robust and adaptive (National Infrastructure Advisory Council, 2009). Additionally, determining the resilience of a system requires that a system's susceptibility to degradation and capability to recover be quantifiable. Establishing a quantifiable measure of resilience for SCADA systems is key to protecting critical infrastructure assets.

1.2 Research Statement

The goal of this research is to provide a method to quantify the identification and absorption of malicious alterations by monitoring and characterizing field device inputs and outputs to PLCs. By focusing on the field device at the micro level, intentional malicious actions can be observed that otherwise would mask effects at the HMI, as was the case in Stuxnet. This research investigates metrics that align with characteristics of resilience. Traits such as self-awareness are a foundational characteristic of resilience and may provide a basis for tangible mechanisms to maintain the integrity of a PLC's nominal functions in the presence of malicious events.

1.3 Research Approach

Establishing a metric to assess a PLC's resilience requires both data that reflects nominal PLC functionality and a definition of resilience which makes the metric applicable. The data used for this research is derived from PLC simulations executed on LogixPro® 500 software. The definition for resilience is taken from the National Infrastructure Advisory Council (2009).

The PLC simulations consist of various programs that emulate instances of a PLC. Each PLC instance is subjected to malicious exploit test cases. The simulated PLC programs are then observed for input-output behavior. The behavior is characterized into formal Petri nets to facilitate analysis of the data and to allow for graphical and mathematical analysis of defined system events (Zurawski, 1994).

This research establishes four program types for each instance. The first two types of programs form two baselines for a PLC instance. The first baseline establishes

the nominal ladder logic to execute a defined set of system processes. The second baseline alters the original baseline's ladder logic to protect against a known malicious exploit. The second pair of programs is formed when *attacks* alter the logic of the first two baselines. The formation of these four PLC programs forms the basis from which equivalent Petri nets are derived.

The method of characterizing a PLC program via a Petri net is by defining the inputs, outputs, and input-output interdependencies of the PLC program. The inputs of the PLC program characterize the transitions between observable process events. The outputs of the PLC program characterize the observable process events. The input-output interdependencies characterize the association between the transitions and observable process events. The resulting Petri nets allow for graphical and mathematical analysis of the emulated PLC instances. These results facilitate identification of metrics which are applicable to assessing resilience.

The Petri nets are created and simulated with PIPEv4.0 software. PIPEv4.0 allows for non-deterministic analysis of the Petri nets (Bonet, 2007). The results establish a set of tangible states and a reachability graph for each Petri net. The tangible states and graphs are combined into a matrix which lists the input-output behavior for each Petri net. Comparative analysis of the matrices provides several metrics that directly address, or indirectly support, the key aspects of resilience.

1.4 Assumptions/Limitations

In this research, the specific attack applied to the baseline program assumes knowledge of the original baseline program. Similarly, the protective baseline program

utilizes knowledge of the attack. These assumptions lead to the creation of four distinctly defined program categories: (i) baseline, (ii) attack baseline, (iii) delta baseline, and (iv) attack delta baseline. This research does not focus on ladder logic programming, but rather seeks to identify measures for differentiating between programs of known quantities. By formulating known programs, the analysis is assured of presenting findings consistent with true input-output behavior for a PLC system under nominal, attack, and protected instances.

The Petri net's simulation software, PIPEv4.0, has limited expressive capabilities. Indeed, the drawing functions are limited to basic places, transitions, and arcs; however, the software performs sufficient simulations and analysis for the instances presented. A useful element not utilized in the experimentation is the presence of inhibitor arcs. Due to the lack of inhibitor arcs, some Petri nets illustrate transitions which have similar, yet unique, properties. For example, a transition labeled *10 to 8* signifies the same PLC input sequence as a transition labeled *10to8*; however, the next output state taken by the PLC is determined by the current place(s) which is enabling the transition *10 to 8* (also *10to8*). Note that a labeled transition (e.g., *10 to 8*) within a Petri net refers to the change in decimal value, from ten to eight, within the PLC's input module.

The assumptions, and limitation, do not alter the applicability of this research or the significance of its findings. Utilizing known programs as a basis for differentiating input-output behavior is necessary and sufficient for this research. Bonet compared PIPE to several other Petri net tools and preferred PIPE's interface and analysis modules (Bonet, 2007). The key analysis modules used are:

- GSPN analysis – Checks for safeness and boundedness, and generates a tangible state table
- Reachability graph – Checks for safeness and boundedness, and generates a graph of all possible firing sequences between reachable states
- Simulation – Performs step-wise and fully automated simulation for a Petri net

Additionally, the Petri net modeling interface analysis fully captures the defined process requirements for each PLC instance. The appropriate enabling states for each transition are representative of the proposed PLC programs.

1.5 Thesis Organization

Chapter 1 provides an introduction for this research. This includes the motivation for this research, research statement, research approach, assumptions, and the organization for this document.

Chapter 2 presents fundamental concepts and related work associated with this research. Background topics include SCADA, resilience, and Petri nets. Related work includes efforts related to SCADA security and resilience.

Chapter 3 describes the experimental methodology. First, a definition of the system boundaries is provided. Second, the factors and workload applied to the system are defined. Finally, details for characterizing the PLC instances into Petri nets and the method for performance evaluation are presented.

Chapter 4 provides the details for the results and analysis of the PLC instances. A corollary analysis of the resulting metrics is performed to identify statistically relevant observations. Then significant findings from the analysis of the metrics and their applicability toward resilience are discussed.

Chapter 5 reviews the key points of this research and provides recommendations for follow-on research. Finally, concluding remarks for this research are provided.

II. Literature Review

This chapter addresses fundamental concepts and related work. Section 2.1 details background topics in SCADA, resilience and Petri nets. Section 2.2 discusses several works closely related to SCADA security and resilience.

2.1 Background

The background topics relevant to this research include SCADA, resilience, and Petri nets. An overview of SCADA architecture is presented along with discussion on security vulnerabilities of associated subcomponents. Additionally, the definition of resilience is discussed, and Petri nets are discussed as a practical means of modeling processes.

2.1.1 Supervisory Control and Data Acquisition (SCADA)

SCADA systems provide an efficient means of monitoring and controlling processes across large geographical regions. SCADA systems are implemented in most modern industrial facilities, such as utilities and manufacturing. Approximately 90 percent of the nation's critical infrastructures are privately owned; a majority of these implementing SCADA as part of their enterprise network (Stouffer, 2008). Indeed, SCADA systems allow industries to streamline operating processes that cover vast geographical regions. To further enhance operating efficiency, industries have now integrated the SCADA system with their business enterprise networks (Stouffer, 2008).

2.1.1.1 Components of SCADA

A SCADA system consists of a control center, communication links, and field sites (Figure 1) (Stouffer, 2008). The control center is comprised of the following:

- Human Machine Interface – displays status of field sites in graphical form.
- Engineering Workstations – allows for operator control of field sites.
- Data Historian – storage and analysis of processed data.
- Control Server or Master Terminal Unit (MTU) – operates SCADA functions, and processes data between control center and field sites.

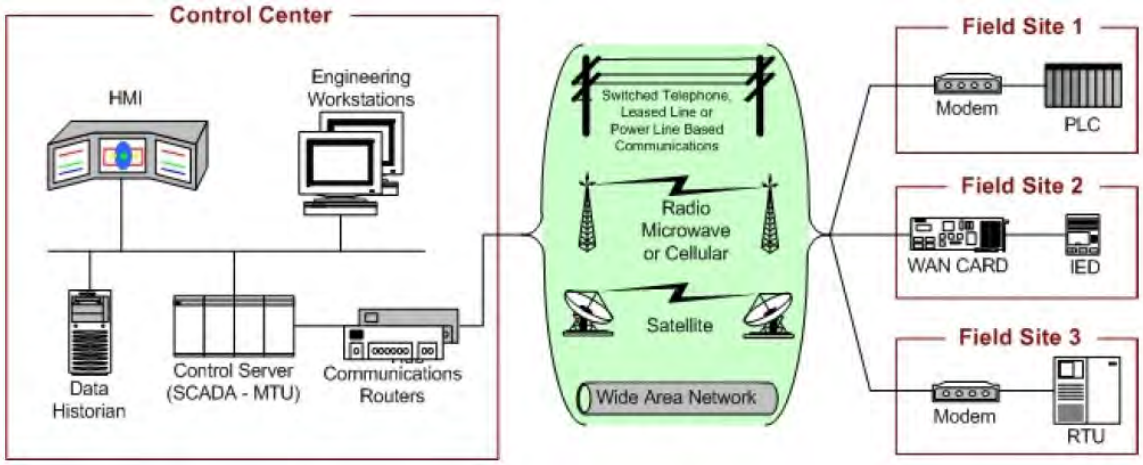

Figure 1: SCADA Components (Stouffer, 2008)

Communication links, routers, and modems relay and convert signals for processing between field sites and the control center. Field sites consist of end-devices that collect data from various sensors (e.g., pressure, flow, or temperature) and transmit the appropriate data to the MTU. The end devices represented in Figure 1 include programmable logic controllers (PLC), intelligent electronic devices (IED), and remote

terminal units (RTU) (Stouffer, 2008). These devices perform similar functions in that they locally control devices (e.g., motors, sensors, valves) and are able to communicate with the control center.

While PLCs and RTUs share similarities, they differ by their interaction to end-devices. RTUs may communicate with other processing units prior to control of an end-device, while PLCs are directly linked to end-devices. PLCs are also more capable of tightly controlling sequential physical processes. The PLC replaces what traditionally were multiple solid state relays, switches, and mechanical timers; however, PLC's flexible programming facilitates configuration changes to physical process requirements.

2.1.1.2 SCADA Security Issues

SCADA systems are designed to provide cost effective control and data acquisition. Security was not initially a priority of SCADA design and development. However, in the past decade focus on SCADA security has grown due to both intentional (Table 1) and unintentional events (Table 2) (Stouffer, 2008) (Falliere, 2011). Non-intentional events have traditionally been addressed with redundant and fault tolerant architectures. Only recently has intentional consequences of malicious events drawn the attention of security experts in the SCADA domain.

Table 1: Intentional SCADA Incidents

Name	Description
Worcester Air Traffic Communications	In March 1997, a teenager in Worcester, Massachusetts disabled part of the public switched telephone network using a dial-up modem connected to the system. This knocked out phone service at the control tower, airport security, the airport fire department, the weather service, and carriers that use the airport. Also, the tower's main radio transmitter and another transmitter that activates runway lights were shut down, as well as a printer that controllers use to monitor flight progress. The attack also knocked out phone service to 600 homes and businesses in the nearby town of Rutland (Thomas, 1998).
MAROOCHY Shire Sewage Spill	In the spring of 2000, a former employee of an Australian organization that develops manufacturing software applied for a job with the local government, but was rejected. Over a two-month period, the disgruntled rejected employee reportedly used a radio transmitter on as many as 46 occasions to remotely break into the controls of a sewage treatment system. He altered electronic data for particular sewerage pumping stations and caused malfunctions in their operations, ultimately releasing about 264,000 gallons of raw sewage into nearby rivers and parks (Smith, 2001).
Stuxnet Siemens Worm	Stuxnet is a threat targeting a specific industrial control system likely in Iran, such as a gas pipeline or power plant. The ultimate goal of Stuxnet is to sabotage that facility by reprogramming PLCs to operate as the attackers intend them to, most likely out of their specified boundaries. Stuxnet was discovered in July 2010, but is confirmed to have existed at least one year prior and likely even before. The majority of infections were found in Iran (Falliere, 2011).

The introduction of business enterprise networks to the SCADA domain has increased vulnerability to malicious attack. Injection points of attack can occur via the Internet, the enterprise network, internal workstations, or communication links between the control center and field sites (Stouffer, 2008). The end goal of a SCADA specific attack may include affecting the physical process by altering the end devices (e.g., motors, sensors, valves); such was the case with the Stuxnet worm. Stuxnet executed code on specific PLCs that caused physical damage to specific drive motors (Falliere, 2011). Stuxnet was not detected by SCADA operators due to a rootkit that masked the deviant behavior.

Table 2: Non-Intentional SCADA Incidents

Name	Description
CSX Train Signaling System	In August 2003, the Sobig computer virus was blamed for shutting down train signaling systems throughout the east coast of the U.S. The virus infected the computer system at CSX Corp.'s Jacksonville, Florida headquarters, shutting down signaling, dispatching, and other systems. According to Amtrak spokesman Dan Stessel, ten Amtrak trains were affected in the morning. Trains between Pittsburgh and Florence, South Carolina were halted because of dark signals, and one regional Amtrak train from Richmond, Virginia to Washington and New York was delayed for more than two hours. Long-distance trains were also delayed between four and six hours (Niland, 2003).
Davis-Besse	In August 2003, the Nuclear Regulatory Commission confirmed that in January 2003, the Microsoft SQL Server worm known as Slammer infected a private computer network at the idled Davis-Besse nuclear power plant in Oak Harbor, Ohio, disabling a safety monitoring system for nearly five hours. In addition, the plant's process computer failed, and it took about six hours for it to become available again. Slammer reportedly also affected communications on the control networks of at least five other utilities by propagating so quickly that control system traffic was blocked (Poulsen, 2003).
Northeast Power Blackout	In August 2003, failure of the alarm processor in First Energy's SCADA system prevented control room operators from having adequate situational awareness of critical operational changes to the electrical grid. Additionally, effective reliability oversight was prevented when the state estimator at the Midwest Independent System Operator failed due to incomplete information on topology changes, preventing contingency analysis. Several key 345kV transmission lines in Northern Ohio trip due to contact with trees. This eventually initiates cascading overloads of additional 345 kV and 138 kV lines, leading to an uncontrolled cascading failure of the grid. A total of 61,800 MW load was lost as 508 generating units at 265 power plants tripped (Minkel, 2008).
Zotob Worm	In August 2005, a round of Internet worm infections knocked 13 of DaimlerChrysler's U.S. automobile manufacturing plants offline for almost an hour; stranding workers as infected Microsoft Windows systems were patched. Plants in Illinois, Indiana, Wisconsin, Ohio, Delaware, and Michigan were knocked offline. While the worm affected primarily Windows 2000 systems, it also affected some early versions of Windows XP. Symptoms include the repeated shutdown and rebooting of a computer. Zotob and its variations caused computer outages at heavy-equipment maker Caterpillar Inc., aircraft-maker Boeing, and several large U.S. news organizations (Roberts, 2005).

2.1.2 Resilience Overview

In general terms, resilience is the ability of a system to continue to operate through disruptions. This notion encompasses a multitude of other terms such as robustness, dependability, and survivability. These characteristics are important to the protection and healing of a system. This section surveys various resilience models from other domains for applicability to SCADA systems (e.g., PLCs).

2.1.2.1 Defining Resilience

Resilience has been researched in other domains in which biological, psychological and community resilient models have been formulated. Biological resilience presents itself in the study of immune systems (VanBreda, 2001). Psychological resilience has been studied in the mental capacity for individuals to perform through adversity (VanBreda, 2001). Community and organizational resilience is demonstrated through the ability of a group or region to recover from catastrophic events (Tierney, 2007; Cutter, 2008). In each of these domains, a common structure of resilience is presented. For example, the components of psychological resilience can be categorized into the following three parts (VanBreda, 2001):

- Inner Self Mechanism - monitoring your physical, meditative, and mental awareness
- Relationship Mechanism - monitoring the taking and giving awareness as well as your self-relationship
- Method - monitoring your habits

This structure of resilience incorporate self awareness and self monitoring mechanisms from the psychological domain in order to initiate the actions required to maintain functional capacity and ability to recover.

Similarly, Tierney (2007) presents resilience in a community or organization as:

- Robustness - the ability of systems, system elements, and other units of analysis to withstand disaster forces without significant degradation or loss of performance
- Redundancy - the extent to which systems, system elements, or other units are substitutable, that is, capable of satisfying functional requirements, if significant degradation or loss of functionality occurs
- Resourcefulness - the ability to diagnose and prioritize problems and to initiate solutions by identifying and mobilizing material, monetary, informational, technological, and human resources
- Rapidity - the capacity to restore functionality in a timely way, containing losses and avoiding disruptions

Trivedi (2009) states that while qualitative descriptions of resilience across domains have been accomplished, applicable quantitative measures are still deficient. This statement is particularly applicable to computer systems. His work attempts to quantify metrics that compare availability, performance, and survivability for computer systems (Trivedi, 2009). Similarly, quantifying resilience of SCADA systems is necessary to measure their ability to perform when perturbations or disruptions to the system occur.

2.1.2.2 Resilience Framework

While there are numerous definitions of resilience across various domains, The National Infrastructure Advisory Council (NIAC) provides perhaps the most fitting definition with respect to SCADA. NIAC (2009) define infrastructure resilience as:

> "the ability to reduce the magnitude and/or duration of disruptive events. The effectiveness of a resilient infrastructure or enterprise depends upon its ability to anticipate, absorb, adapt to, and/or rapidly recover from a potentially disruptive event." (p. 8)

This definition provides a framework for resilience according to the following four characteristics:

1. *The ability to anticipate a potentially disruptive event requires that the system has a self awareness of its baseline and is able to monitor its current state.*
2. *The ability to absorb potentially disruptive events requires that the system has mechanisms in place to minimize the amount, if any, of performance loss.*
3. *The ability to adapt requires that the system have contingencies available that allow for flexible system adjustments to maintain operational availability.*
4. *The ability to recover from a disruptive event requires mechanisms (either automated or manual processes) which allow the system to perform up to its baseline.*

2.1.3 Petri Nets Overview

Petri nets are named after its creator Carl A. Petri in 1962 (Zurawski, 1994). Initial development in 1962 concentrated on the study of communication via automata. Zurawski and Zhou provide a simple definition (Zurawski, 1994):

> "Petri nets as, graphical and mathematical tools, provide a uniform environment for modeling, formal analysis, and design of discrete event systems." (p. 567)

In its graphical form, Petri nets consist of four basic parts: (i) places, (ii) transitions, (iii) arcs, and (iv) tokens. The places and transitions are indicative of nodes within a graph and arcs relate to pairs of places and transitions. The tokens represent places which are active (marked). Table 3 lists examples of places and transitions.

Table 3: Examples of Petri Net Places and Transitions (Abhishek, 2005)

Places	Transitions
pre/post condition	event
input/output data	computational step
input/output signal	signal processor
resources	tasking
buffer	processor

2.1.3.1 Simple Petri Net Example

Figure 2 illustrates three markings of a Petri net modeling a simple traffic light, with one light for red, yellow, and green (Abhishek, 2005). Places are represented by circles, transitions as blocks (or bars), arcs as directed arrows, and tokens as dots. The initial marked graph (M_0) shows that a token is active in the red place, which is interpreted as the red light being active (or illuminated). Note that there is only one token in the red place of the initial marked graph (M_0) to emphasize that only one red light exists for this example. The presence of multiple tokens in one place may be interpreted as the existence of more than one red light in the system.

The basic rules for transitioning a Petri net from one marked graph (M_0) to the next (M_1) involves the action of the transitions (Peterson, 1977). The execution of a transition is called firing. In order for a transition to fire, the transition must be enabled.

A transition is enabled, and may fire, if all its input places contain at least one token. Note that even though a transition is enabled, it is not strictly required to fire. The firing of a transition results in moving a token from the input places to all output places. In the traffic light example, all transitions have only one input and output place, so it is easily shown that the number of tokens in the Petri net remain unchanged (at one). Figure 2 shows all possible states of the Petri net (based on the initial state of M_0 with one token). The sequence of places is limited to one light (red, green, or yellow) illuminated in any specific instance, and limited to one repeating sequence (R → G → Y).

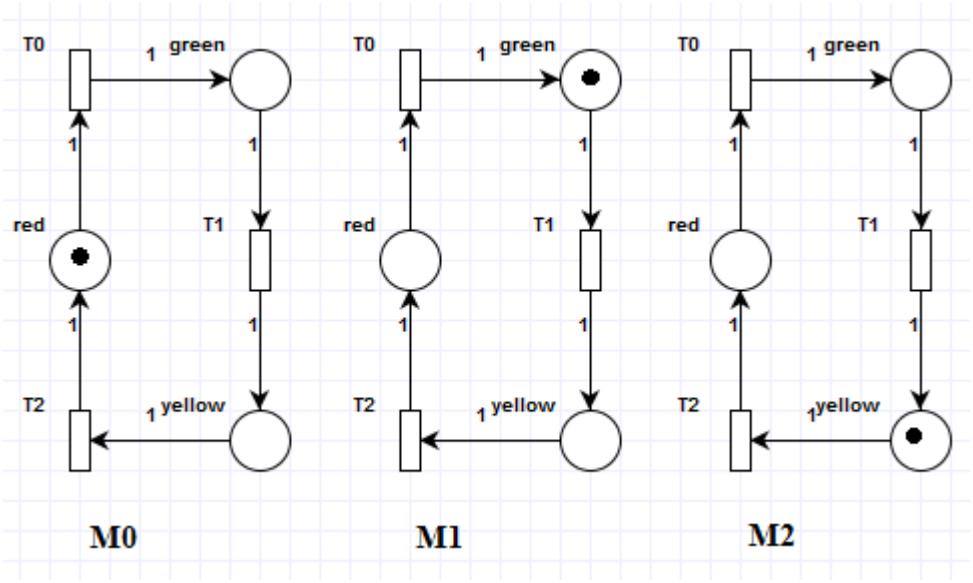

Figure 2: Example Traffic Light Petri Net

2.1.3.2 Formal Definition of a Petri Net

A Petri net C consists of four subsets, $C = \{P, T, I, O\}$, such that P is a set of places, T is a set of transitions, I is a set of input functions for each transition, and O is a

set of output functions for each transition (Peterson , 1981). The formal definition for the traffic light example in Figure 2 is represented as:

- C = {P, T, I, O}
- P = {red, green, yellow}
- T = {T0, T1, T2}
- I(T0) = {red}, I(T1) = {green}, I(T2) = {yellow}
- O(T0) = {green}, O(T1) = {yellow}, O(T2} = {red}

2.1.3.3 Petri nets in Application

Petri nets have been applied to modeling of performance, reliability, fault recovery, and fault tolerance in various systems such as operating systems, queues, traffic control and mathematics (Peterson, 1981). Additionally, modeling of manufacturing processes similar to SCADA applications have also been analyzed (Zurawski, 1994). However, the analysis of these systems focuses primarily on fault tolerance and reliability within the systems' designs. Utilizing Petri nets to analyze systems for intentional exploits (e.g., malware) is significant to increasing the security posture of SCADA systems.

Properties of Petri nets that are practical for analysis of SCADA applications are concurrency, safeness and boundedness (Peterson, 1977). Concurrency allows for the modeling of parallel processes that occur between the multiple devices that interact in a SCADA system. Safeness and boundedness addresses the potential issue of state explosion when analyzing a system. Safeness implies no more than one token may be

present in each place of a Petri net. It follows that if a Petri net is safe, then it is also bounded. This results in a finite set of reachable markings since tokens are not created without bound. These properties well suit the defined configuration and deterministic interdependencies present in SCADA systems. Properly defining places and transitions for a Petri net based on SCADA system processes should result in a finite set of states.

2.2 Related Work

The related works section examines analysis and resiliency concepts relating to SCADA security. Queiroz (2010) and Germanus (2010) present individual models for SCADA security analysis at a macro-level, while Shah (2008) explores SCADA security protocol at a micro-level. Wei (2009) provides an exploratory analysis of resilience metrics that may be utilized to assess industrial control systems.

2.2.1 Survivable SCADA Systems

Queiroz (2010) presents a model to quantify SCADA system performance against a denial of service (DoS) attack. The model focuses on the interplay of four main components of a SCADA system: RTU, MTU, HMI Server, and Data Historian. The availability of each component is modeled as queues that allow each service to handle a specific number of requests. The aggregate output of each component's availability is compiled into a Bayesian table, which incorporates the interdependencies, and then quantifies the survivability of the SCADA system. Figure 3 provides a summary for two sets of sample data. The thresholds for each of the SCADA components (i.e., normal,

degraded, unavailable) and survivability (i.e., yes, no) for the system is pre-determined prior to model analysis.

RTU	MTU	HMI Server	Historian Database	Survivable
Normal	Normal	Normal	Normal	YES
Degraded	Normal	Unavailable	Degraded	NO
...	...	...	...	...

Figure 3: Queiroz's Summary for Sample Data (Queiroz, 2010)

Quieroz's research contributes to part two of the previously defined resilience framework (i.e., the ability to absorb potentially disruptive events requires that the system has mechanisms in place to minimize the amount, if any, of performance loss). Interdependencies of a particular SCADA system may be analyzed to determine if the architecture is survivable against a DoS attack. The result of the analysis can be used to improve the absorptive capacity for the SCADA system. Quieroz's approach is sufficient for system wide analysis of a SCADA system and the timing interdependencies between network nodes. However, it does not account for hardware or software faults. They assume that each node itself is not prone to failure; only that the communication between the nodes is interrupted which causes degradation of node availability, and subsequently system survivability.

The Quieroz (2010) approach ignores traditional fault tolerance or the presence of malware. Hardware faults are traditionally classified in the domain of fault tolerance, while software faults may include malware exploits such as Stuxnet. Additionally,

Quieroz's contribution towards resilience resides at the macro level of the SCADA system. No observations are made to determine the specific behavior of one particular node. This approach does not address part one of the resilience framework (i.e., the ability to anticipate a potentially disruptive event requires that the system has a self awareness of its baseline and is able monitor its current state) and lacks the sensitivity to detect malware. While the model works well in determining susceptibility to DoS attacks and improving a SCADA system's absorptive capacity, it is insufficient in monitoring the current state of the SCADA system to aid real-time monitoring of system behavior at a micro-level (e.g., end-device control via PLC).

2.2.2 Redundant SCADA Network Architecture

Germanus (2010) presents a model in which communication between the RTU and MTU is performed via redundant links throughout the SCADA system. The model implements the redundant paths as middleware that are assumed to be free from security vulnerabilities. This model may improve the SCADA system's resilience against DoS and man-in-the-middle (MITM) attacks. DoS attacks may be mitigated by using the redundant paths available on the network links. MITM attacks may be mitigated by the data integrity checks associated with the middleware. Figure 4 illustrates the middleware model that passively extracts SCADA communication and relays it across the peer-to-peer (P2P) overlay.

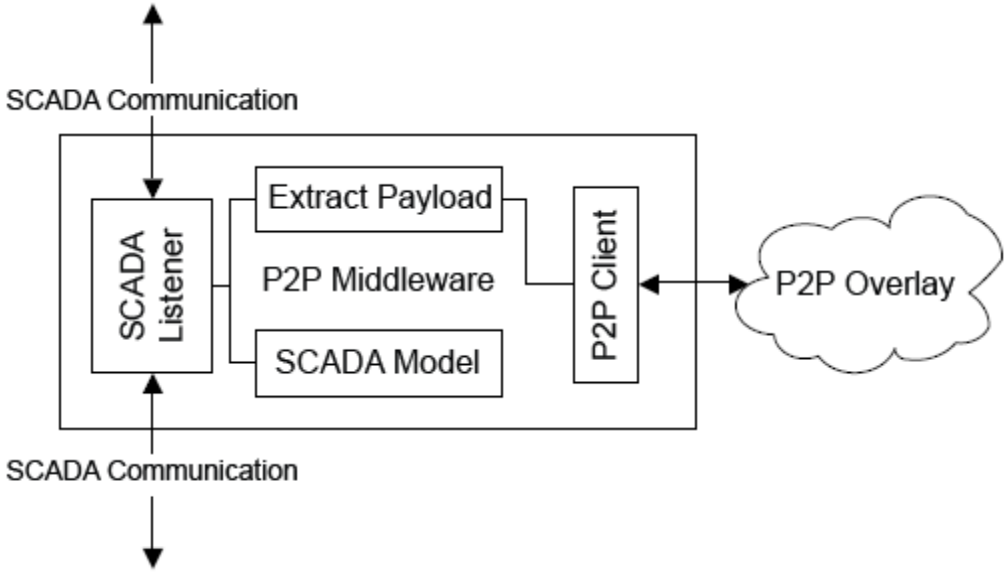

Figure 4: Germanus' Middleware Building Blocks (Germanus, 2010)

The advantages to this model are flexibility, interoperability, and minimal intrusiveness. The flexibility allows the system to withstand link failures which addresses part two of the resilience framework (i.e., the ability to absorb potentially disruptive events requires that the system has mechanisms in place to minimize the amount, if any, of performance loss). Figure 5 illustrates the interoperability and minimal intrusiveness of Germanus' model which facilitates deployment of the model to existing SCADA systems since the P2P overlay uses middleware as an interface between existing RTU and MTU links (Germanus, 2010).

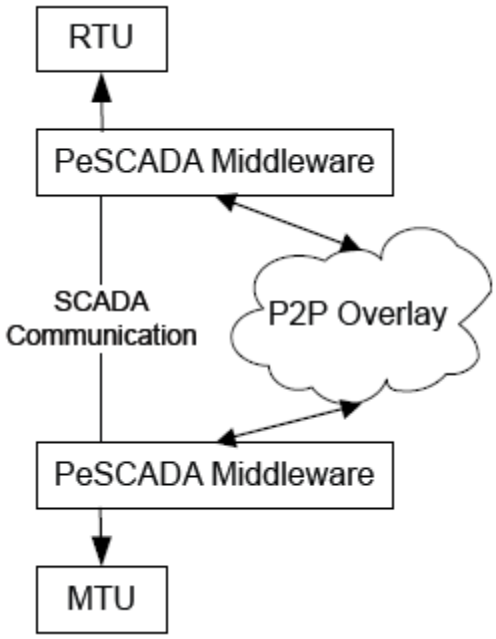

Figure 5: Germanus' (2010) Redundant P2P Model

The model expands on Quieroz's research contribution to SCADA resilience in two ways. First, it can be implemented real-time on existing SCADA system infrastructure. Second, the redundant network paths provide increased node availability, and therefore increase survivability of the SCADA system.

Similar to Quieroz's research, Germanus' analysis resides at the macro-level of the SCADA system. Detection of hardware and software faults local to either the RTU or MTU is undetected since it is isolated to the SCADA system's communication links. Local behavior of any particular RTU is still only monitored through the HMI. However, the P2P overlay is able to provide real-time feedback of link or message abnormalities and partially addresses part one of the resilience framework since it will detect systemic behavior.

2.2.3 Mechanisms to Provide Integrity in SCADA Devices

Shah (2008) presents a method to verify the executable code of a PLC. The method implements a challenge-response protocol between the PLC and an external dispatcher. A verification function resides on both the PLC and dispatcher. Figure 6 summarizes the steps of the verification protocol. Steps one through three assures that the verification function is trustworthy, while steps four and five assure that the executable code of the PLC is untampered (Shah, 2008). Steps one through five of Shah's challenge-response protocol is as follows:

1. The dispatcher sends a random challenge to the PLC.
2. The verification function of the PLC computes a checksum.
3. PLC returns the results to the dispatcher.
4. Verification function of the PLC creates a hash of the executable code.
5. PLC sends the hash result to the dispatcher which compares it against the known hash.

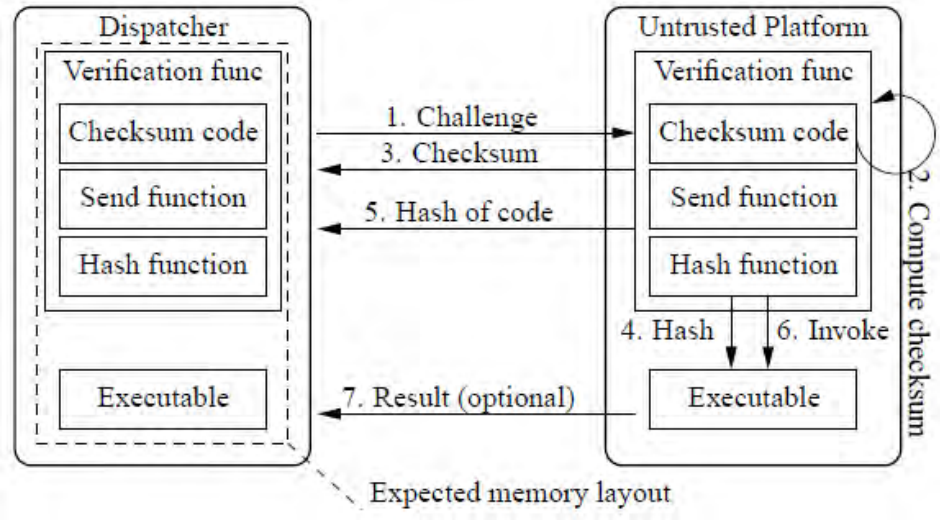

Figure 6: Shah's (2008) Verification Function Overview

This method of verifying executable code on the PLC approaches SCADA resilience from the local node level as opposed to the system-level approaches of Quieroz and Germanus. Shah's method addresses the first part of the resilience framework (i.e., the ability to anticipate a potentially disruptive event requires that the system has a self awareness of its baseline and is able monitor its current state) since it is able to detect changes to the executable code on the PLC. It also provides flexibility since it may be implemented in existing PLCs currently deployed in the field; however, it requires that the verification function be integrated with the PLC.

Shah's method incurs several logistical issues. The paper acknowledges that the verification functions of the PLC and dispatcher are different for each PLC manufacturer (Shah, 2008). While the challenge-response protocol is general across platforms, the verification functions differ based on the PLC architecture. Another logistical issue is that the PLC must be taken off-line to perform the challenge-response protocol between the dispatcher and PLC. This presents operational impacts to most SCADA systems since most PLCs run real-time applications.

Shah (2008) also acknowledges that the verification process only assures that no malicious code is present at the time the verification function is performed between the dispatcher and PLC. It does not prevent timed attacks in which the adversary may execute malicious code on the PLC between verification timelines. Additionally, the method does not address the second part of the resilience framework (i.e., the ability to absorb potentially disruptive events requires that the system has mechanisms in place to minimize the amount, if any, of performance loss) since no processes are in place to reduce the effect of malicious code once detected.

2.2.4 Resilient ICS: Concepts, Formulation, Metrics, and Insights

Industrial control systems (ICS) are deployed in sectors such as agriculture, utilities, and transportation. Wei (2009) presents a set of resilience metrics that may be used to quantify performance of a system. Figure 7 identifies the trigger points for a resilient system across a timeline. The trigger points are utilized to define equations for protection time, degradation time, identification time, recovery time, performance degradation, performance loss, total loss, and overall potential critical loss.

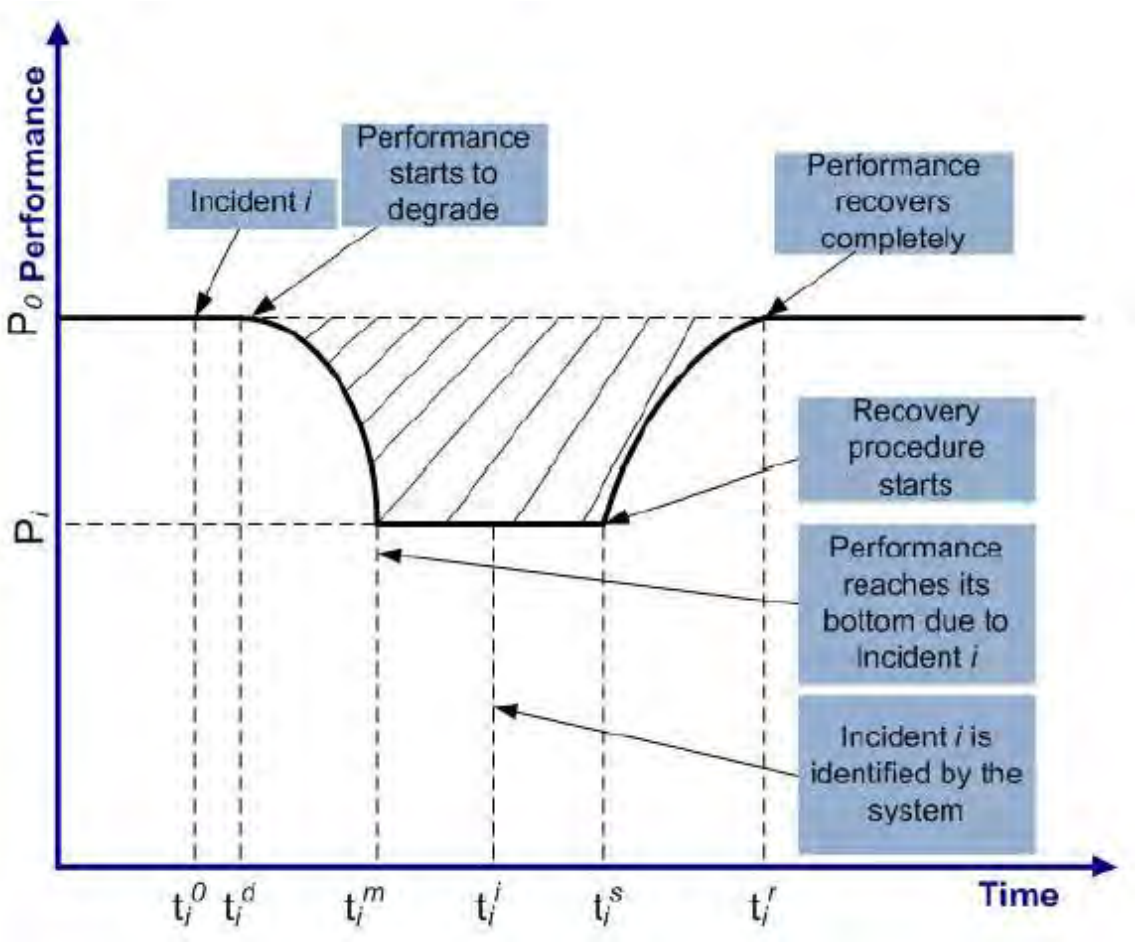

Figure 7: Wei's Resilience Curve (Wei, 2009)

Subsets of the defined equations that apply to the first and second parts of the resilience framework are (Wei, 2009):

- Protection time – time that the system tolerates an incident without degradation
- Degradation time – time that the system incurs to reach its minimum performance level
- Identification time – time from incident occurrence to system identification
- Performance degradation – difference between baseline performance and degraded performance due to incident

The four definitions presented by Wei address the detection of the incident and the level of mitigation the system performs. These definitions, or slight variations of them, may aid in analyzing various models that seek to improve resilience in SCADA systems.

2.3 Summary of Literature

This chapter presented the relevant background and related works associated with quantifying resiliency of PLCs through the use of Petri nets. Knowledge of SCADA architecture and security vulnerabilities is a foundational element. A formal definition of resilience and Petri nets is also relevant to key areas of Chapters 3 and 4 of this research. The various SCADA security analysis techniques presented at both the macro and micro-levels provide a basis of comparison for the proposed methodology of Chapter 3. The related work on resilient metrics for ICS is insightful to the hypothesis of this research and gives relevance to findings in Chapter 4.

III. Methodology

This chapter presents the methodology for characterizing the PLC ladder logic programs into equivalent Petri nets for evaluating metrics to assess resilience. Section 3.1 describes the goal and hypothesis for this research. Section 3.2 identifies the system boundaries. Section 3.3 describes the system services. Section 3.4 lists the parameters of the system. Section 3.5 defines the factors that apply to the system. Section 3.6 describes the workload applied to the system. Section 3.7 details the approach for characterizing the ladder logic into equivalent Petri nets. Section 3.8 identifies the performance metrics derived from the experimentation. Section 3.9 describes the evaluation method used to form resilience metrics. Section 3.10 outlines the experimental design.

3.1 Problem Definition

Improving the resilience of ICS allows critical infrastructures to withstand degrading events, and recover to a nominal functional capability within an acceptable period. However, determining resilience requires that a system's susceptibility to degradation and capability to recover is quantifiable. Narrowing the scope of research to a micro-level component of a SCADA system provides a basis to facilitate evaluation of potential resilience metrics.

3.1.1 Goal

The primary goal of this research is to identify metrics that may assess a PLCs performance with respect to the resilience framework presented in Chapter 2. A

complimentary goal is to identify metrics that are applicable to real-time physical mechanisms. The resilience curve (Figure 7) identifies trigger points that are utilized in evaluating resilience performance; however, the *mechanisms* for the triggers are absent. Achieving both research goals may result in applicable mechanisms which appropriately assess resilience in controlled (e.g., benchmark) and real-time (e.g., operations) environments. This research may reveal comparative metrics that help determine if awareness of the system state is discernible. Self-awareness is a foundational characteristic of the resilient framework and provides a basis for tangible mechanisms to implement trigger points in real-time hardware protection schemes.

3.1.2 Hypothesis

The hypothesis of this research is that a PLC's ability to identify and absorb malicious alterations is quantifiable by monitoring system outputs in response to system inputs. The approach to derive the metrics for resilience assessment uses comparative analysis of various instances of PLC programs.

3.2 System Boundaries

The system under test (SUT) is the PLC processes. Figure 8 illustrates the SUT and associated inputs and outputs. The workload applied to the SUT includes various attack instances as detailed in *Section 3.6.* The parameters applied to the SUT are primarily fixed attributes of the PLC emulation provided by LogixPro® 500; the varying parameter during experimentation is the protection scheme applied to each specific PLC

instance. The metric produced from the SUT is the decimal input and output values produced during program execution.

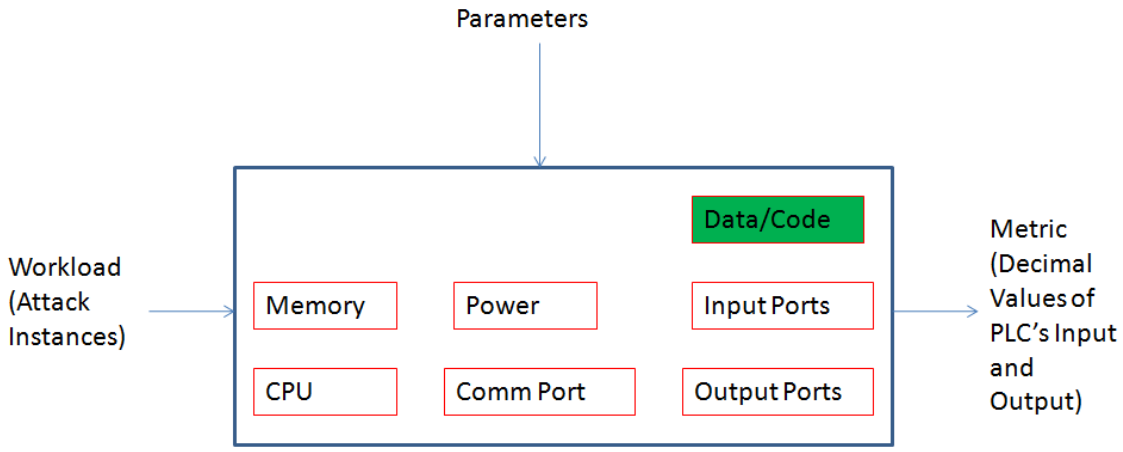

Figure 8: PLC SUT Diagram

System components that comprise the PLC include: Memory, Data/Code, CPU, Input Ports, Output Ports, Power Supply, and Communication Port. The component under test (CUT) is the data/code or programming logic of the system. Data/code is programmed in ladder logic from a laptop with the accompanying LogixPro® 500 software package associated with the PLC. The program is loaded to the PLC which executes the ladder logic and produces observable output signals in response to input signals.

3.3 System Services

A PLC provides four primary services: (i) execution of the ladder logic program, (ii) monitoring of input signals, (iii) production of output signals, and (iv) providing data back to the master device of a SCADA system. It is assumed that accurate data are

transmitted to the master device (i.e., no spoofing of output states), and that input signals are only injected at valid input ports (i.e., no spoofing of input states). It is also assumed that the programs (i.e., baseline and enumerated versions) are not subject to hardware faults or undesired software faults. These assumptions isolate the boundary of the system from external influences, and assure the integrity of the applied inputs and observed outputs.

The primary services monitored are the applied inputs and the behavior of the outputs. The observed PLC output signals are a direct result of the PLC program code execution and the input signal status. Applying inputs to the PLC produces output signal states that affect the end-devices (e.g., motors, lights, actuators). These behavioral responses of the output states in response to the input states are measurable in the observed status of the end-devices. The PLC's interaction with the end-devices fall into one of three observable response categories:

- Valid – Nominal input results in nominal output processes
- Degraded – Nominal input results in deviant but safe output processes. A safe outcome is defined as a non-nominal output response in which the system's interactions with end-devices do not cause catastrophic losses (e.g., minor perturbations)
- Unstable – Nominal input results in deviant and unsafe processes. An unsafe outcome is defined as a non-nominal output response in which the system's interactions with end-devices may cause catastrophic losses (e.g., loss of life or resources)

Table 4 outlines system responses using a traffic signal example. For this example, the input is an automated timed sequence which transitions the light between its

potential outputs. In nominal conditions the sequence is always valid (see Table 4). However, as seen in Table 4, in non-nominal conditions the sequences are degraded (within defined process requirements) or unstable (outside defined process requirements). The output of the end-devices (i.e., lights) are observed, and categorized accordingly.

Table 4: Example Traffic Light System Response

Category	Traffic Signal Output
Valid	Lights transition from green to yellow to red.
Degraded	Lights transition from green to yellow to flashing red.
Unstable	Lights transition from green to yellow to green.

3.4 System Parameters

The following are the system parameters: Power Input, Communication Port Input, Communication Port Output, CPU scanning speed, Memory size, Input module size, Output module size, Data/Code (programmed Ladder Logic). Table 5 describes each parameter.

Table 5: Parameters

Parameter	Description
Power Input	Provides power to the PLC via an AC to DC inverter. This remains at factory default (24VDC).
Communication Input Port	Provides access to write to PLC memory. This is used to download program code to the PLC via serial communications using RS-232 signaling. During testing, the port is not used and is in a closed state.
Communication Output Port	Provides external and remote monitoring of the PLC via an external master unit. During testing, the port is not used and is in a closed state.
CPU Scanning Speed	Adjusts the rate at which the code is read from memory. The experiment will use factory default settings of 44 Kbps.
Memory Size	Memory size is upgradeable depending on the size of programming required. The experiment does not necessitate programs larger than the factory default memory space. The experiment will use the factory default of 1K.
Input Module Size	Modules are upgradeable depending on the number of required inputs signals that are required to connect to the PLC. The experiment does not necessitate a number larger than the factory default input module size (4 input channels).
Output Module Size	Modules are upgradeable depending on number of required outputs signals that are required to connect to the PLC. The testing scenarios do not necessitate a number larger than the factory default output module size (4 output channels).
Data/Code (Ladder Logic)	Ladder logic is the data held in memory which is executed by the CPU. Input channels are scanned, depending on the program logic, and output channels are energized.

3.5 Factors

The factor of interest resides in the ladder logic programs of the PLC. Two variations of a program baseline are applied to the CUT. The research environment consists of a process emulation using LogixPro® 500. LogixPro® 500 provides a graphical user interface to develop, compile, and execute distinct instances of PLC

programs and system operating parameters. The multiple instances demonstrate distinct observable input-output behavioral patterns when subjected to various example malicious attacks. For each instance, two baseline program categories are established:

- Baseline – A program to perform defined process requirements; generates valid input-output responses.
- Delta Baseline – A protection scheme applied to the baseline that generates valid input-output responses. The protection scheme can be considered equivalent to a fail-safe system state (e.g., flashing red lights for a roadway stoplight system).

3.6 Workload

The workload includes ten instances of PLC attacks applied to the CUT. These ten instances were created such that degraded physical operations of the system are readily observable. Table 6 summarizes the ten attack instances. The attacks, in combination with the baseline program categories, form two additional program categories:

- Attack Baseline – A targeted attack to the baseline that generates degraded or unstable input-output responses.
- Attack Delta Baseline – A targeted attack applied to the delta baseline that generates valid, degraded or unstable input-output responses.

Table 6: Ten PLC Attack Instances

Instance	Description
1	remove logic for proximity sensor in rung 3; quickly floods plant unless stopped manually
2	remove ladder logic for level sensor in rung 2; quickly floods plant unless stopped manually
3	remove logic for manual stop in rung 0; plant runs continuously
4	remove logic for full signal in rung 3; slowly floods plant due to lag response of fill valve to close; may stop manually
5	combine attack 2 & 4; quickly floods plant; may stop manually
6	combine attacks 2, 3, and 4; quickly floods plant; manual stop disabled
7	remove logic for proximity sensor and full light signal in rung 1; slowly floods due to containers not stopping at fill station; may stop manually
8	combine attack 6 & 7; quickly floods plant; manual stop disabled
9	remove full light and motor signal in rung 4; slowly floods plant; may stop manually
10	remove logic for proximity sensor and motor signal in rung 4; quickly floods; may stop manually

3.7 Approach

This section describes the methodology for characterizing the input-output relationships for a PLC's programming logic. An initial baseline program is established that incorporates PLC programming for an operational system. Once the baseline is established, modifications are made to emulate a PLC infected with malware. Protective schemes are then applied to mitigate effects of the malware. The enumerated instances of the PLC programs are evaluated to observe deviations of input-output behavior. Petri net

models are then utilized to extract metrics that measure the PLC's security performance with respect to the resiliency framework.

The various PLC instances establish a basis of observable input-output responses that are modeled and analyzed using Petri nets. The observations obtained from the input-output responses are consistent with black-box analysis; however, application of the targeted attacks and protection schemes use the PLC program to facilitate differentiation of observed behavior from the defined nominal process requirements. The following steps describe the methodology for deriving each of the four program categories and equivalent Petri nets.

1. *Establish Baseline Program* – A ladder logic program is developed to perform defined nominal process requirements. The baseline program generates valid system input-output responses.

2. The possible combinations for outputs of the formal ladder logic are abstracted as *places* in a Petri net.

3. The possible combinations for inputs of the formal ladder logic are abstracted as *transitions* in a Petri net.

4. The input and output interdependencies of the formal ladder logic are abstracted as input and output functions for each of the potential transitions of the Petri net.

5. The data obtained in steps 2 through 4 are combined to define a Petri net for stochastic analysis of the input-output behavior.

6. *Establish Delta Baseline Program* – The original ladder logic developed in Step 1 is modified to incorporate a protective scheme that generates valid input-output responses. Steps 2 through 5 are repeated to produce the equivalent Petri net of the delta baseline PLC Program.

7. *Establish Attack Baseline Program* – This step modifies the ladder logic in a manner consistent with a targeted malicious attack. Steps 2 through 5 are repeated to produce the equivalent Petri net of the attack baseline PLC Program.

8. *Establish Attack Delta Baseline Program* – The ladder logic developed in Step 7 is modified with a targeted attack to generate degraded or unstable input-output responses. Steps 2 through 5 are repeated to produce the equivalent Petri net of the attack delta baseline PLC Program.

This research examines ten instances and varying attacks using an example silo process for experimentation. Each PLC instance has a baseline and delta baseline program; each instance also has an attack applied to each baseline. The net result is each PLC instance has four generated programs and corresponding Petri nets. The following provides a step-by-step guide to generate the four programs and corresponding Petri nets for the first PLC instance. The remaining nine instances are derived in a similar fashion; the resulting programs and Petri nets are provided in Appendix A.

3.7.1 Establish Baseline for PLC Instance #1

This phase constructs a ladder logic program which executes a defined set of process requirements. Consider, for example, a system process in a silo plant that fills containers via a conveyer belt and automated sensors. The nominal processes for the silo plant are: bring an empty container into the plant, maneuver the container under the silo valve, fill the container until full, and ship the full container out of the plant. Figure 9 shows a baseline ladder logic program for the process.

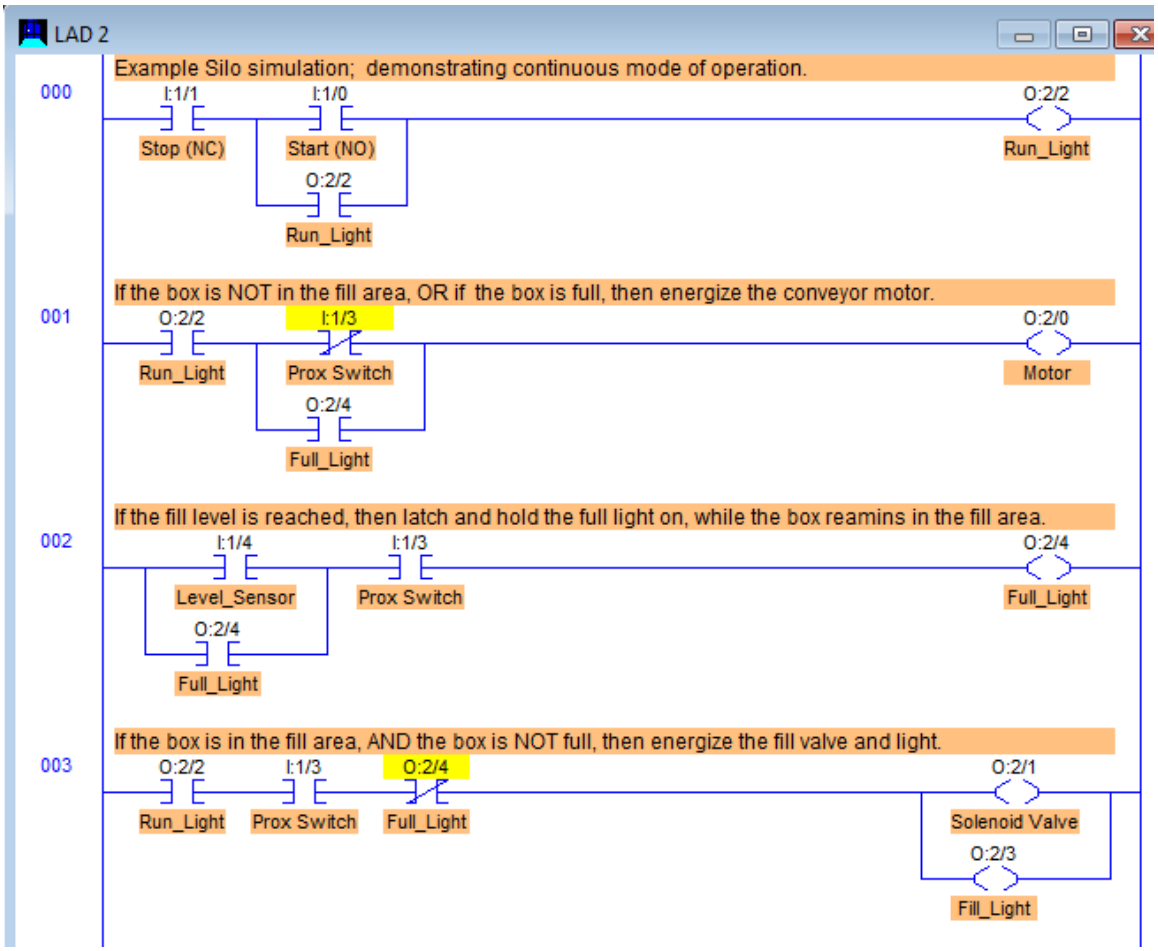

Figure 9: Ladder Logic of Baseline Program

3.7.2 Characterize Baseline Program as Petri Net

This phase translates the ladder logic program into an equivalent Petri net. Potential inputs, outputs, and interdependencies of the program are converted into a Petri net $C = \{P, T, I, O\}$. The formal definition of C is used to generate a graphical Petri net that is simulated to derive analytical data and metrics.

3.7.2.1 List Potential Output Behavior

Output behavior of the program is monitored and recorded during its execution. For this example, output behavior during simulation is described as the following: deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship. Note that observed output behavior of the program closely mirrors the nominal process requirements described in Section 3.7.1; the only additions are the stop intervals during any portion of the program's execution. This is as expected since a PLC directly controls physical devices.

The output behaviors form the set of output places, P, for the Petri net C: P = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}. Note that a place (e.g., *deliver container*) within a Petri net is defined as an observed physical process of the PLC.

3.7.2.2 List Potential Input Transitions

Input transitions of the program that result in changes to output behavior are monitored and recorded. For example, consider the following input transitions: 2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27. These input transitions form the set of transitions for the Petri net: T = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}. Note that a transition (e.g., *2 to 0*) within a Petri net is defined as a change in decimal value from two to zero, within the PLC's input module.

3.7.2.3 Identify Input-Output Interdependencies

The input and output interdependencies of the program during execution are monitored and recorded. This step defines the arcs that interconnect the places and transitions of the Petri net. The process of defining each of the Petri nets focuses strictly on PLC program input transitions that cause a physical output state to change. For example in the baseline program, the act of the user *releasing* the *stop button* causes the input value to transition from *0 to 2*; however it causes no change to the output state. Only the act of *pressing* the *stop button* (changing input value from *2 to 0*) may cause a change to the output state. This simplification to the Petri net models enables PIPEv4.0 to adequately model the input-output behavior of the PLC programs. Note that the output places are annotated with the cumulative decimal value of the PLC's output module for a given observable physical process (i.e., *deliver container* is manifested when the decimal value of the PLC is *5*). The set of *I* consist of the following functions:

- *I*(2 to 0) = {deliver container (5)}
- *I*(2 to 3) = {stop deliver (0)}
- *I*(2 to 10) = {deliver container (5)}
- *I*(10 to 2) = {ship container (21)}
- *I*(10 to 8) = {fill container (14), ship container (21)}
- *I*(10 to 11) = {stop fill (0), stop ship (0)}
- *I*(10 to 26) = {fill container (14)}
- *I*(26 to 10) = {depart silo (21), container full (20)}
- *I*(26 to 24) = {depart silo (21)}
- *I*(26 to 27) = {stop depart (0)}

The set of *O* consist of the following functions:

- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {deliver container (5)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 2) = {deliver container (5)}
- *O*(10 to 8) = {stop fill (0), stop ship (0)}
- *O*(10 to 11) = {fill container (14), ship container (21)}
- *O*(10 to 26) = {depart silo (21), container full (20)}
- *O*(26 to 10) = {ship container (21)}
- *O*(26 to 24) = {stop depart (0)}
- *O*(26 to 27) = {depart silo (21)}

3.7.2.4 Formal Petri Net of Program

The formal Petri net for the baseline program is defined as *C = {P, T, I, O}*. Combining the definitions for *P, T, I and O* from *Section 3.7.2.1* through *Section 3.7.2.3* results in the graphical Petri net presented in Figure 10.

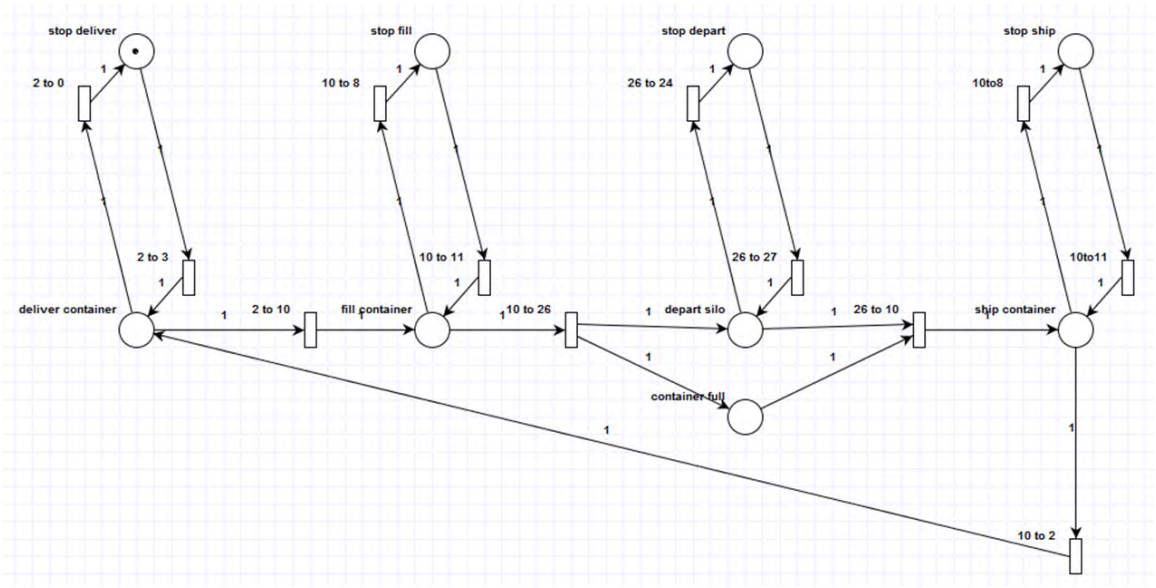

Figure 10: Petri Net of Baseline Program

3.7.3 Establish Delta Baseline Program

Establishing a delta baseline program provides a ladder logic program which emulates the process requirements of the baseline program. The primary difference is that it provides robustness against a targeted attack (Table 6). The formation of the baseline and delta baseline program comprises the two possible factors for each PLC instance. Characterization of the delta baseline program into a Petri net follows the method described in *Section 3.7.2*. The resulting ladder logic and Petri net for the *delta baseline program* is provided in Appendix A.

3.7.4 Apply Workload (Attacks to Baselines)

The applications of attacks to the *baseline* and *delta baseline* programs comprise the workload for the experimentation. The attacks modify the ladder logic of each of the baseline program. The resulting attacks result in two additional PLC programs: *attack*

baseline, and *attack delta baseline*. Characterization of these two programs via a Petri net follows the method described in *Section 3.7.2*. The resulting programs and Petri nets for the *attack baseline*, and *attack delta baseline*, are provided in Appendix A.

In all PLC instances the *attack baseline* programs demonstrate degraded or unstable output; similarly, all *attack delta baseline* programs demonstrate stable or degraded output. This outcome is a product of the assumption that states all attacks are based on internal knowledge of the *baseline program*. This also highlights the fact that the *delta baseline programs* are consequently more robust than the *baseline programs* when similar attacks are applied.

3.8 Performance Metrics

The metric of interest produced from the SUT are the decimal values of the input and output states during the execution of the programs for each PLC instance. The decimal values of the input and output states are measured directly from the input and output modules of the PLC. Observing the input-output behavior during program execution allows for the characterization of an equivalent Petri net. The Petri nets are then analyzed to derive comparative metrics to determine which set(s) of programs provide significant findings towards assessing a PLC's performance with respect to the resilience framework.

Performing a pair-wise comparison between the four possible programs results in six possible pairings:

- (baseline – delta baseline)
- (baseline – attack baseline)

- (baseline – attack delta baseline)
- (attack baseline – delta baseline)
- (attack baseline – attack delta baseline)
- (delta baseline – attack delta baseline)

Analyzing the observed differences between these pairings provides metrics for assessing PLC performance with respect to the resiliency framework. The direct measurements and comparisons of the input and output states of the PLC provide a true representation of the PLCs performance.

3.9 Evaluation Technique

The experiments are performed via two methods: (i) direct measurement on PLC hardware, and (ii) simulated results evaluated with a Petri net model.

3.9.1 Direct Measurement via PLC

The setup for this method utilizes LogixPro® 500 and a laptop with Windows 7 (64-bit) installed. For each PLC instance, four programs are created. The first program is the baseline program which executes defined nominal process requirements. The second is a delta baseline program which also executes defined nominal process requirements, but is more robust against the application of a specific attack. The third program is a modified version of the baseline program to simulate application of a specific attack. The fourth program is a modified version of the delta baseline program to

simulate the application of the same specific attack previously demonstrated in the third program.

Both the baseline and delta baseline programs should produce predictable and valid responses for all input sequences. The third program (*attack baseline*) should exhibit degraded or unstable responses as a consequence of the knowledgeable applied attack. The fourth program, depending on its level of robustness, should exhibit valid or degraded responses.

The input-output behavior during the execution of the four programs is monitored and recorded. The derived metrics provide the basis for characterizing the equivalent Petri nets. Additionally, the number of ladder logic modifications made between each of the four programs is recorded. These metrics provide the basis for quantifying the internal modifications made to the PLC programs.

3.9.2 Petri net Analysis

The setup for analysis of the Petri nets uses Platform Independent Petri Net Editor version 4.0 (PIPEv4.0) and a laptop with Windows 7 installed (32-bit). For each PLC instance, four equivalent Petri nets are created. The purpose of each Petri net follows the four programs described in *Section 3.9.1*. The input-output behavior during the simulation of the four Petri nets is monitored and recorded. These metrics provide the basis for quantifying the external input-output behavior of the PLC programs.

3.10 Experimental Design

Experimental trials consist of full factorial (without replication) configuration for the PLC and equivalent Petri nets. The two baseline programs combined with the attacks of each PLC instance result in four program categories that alter the data/code.

- Data/Code
    1. Baseline – A program to perform defined process requirements and generates valid input-output responses.
    2. Attack Baseline – A targeted attack to the baseline that generates degraded or unstable input-output responses.
    3. Delta Baseline – A protection scheme applied to the baseline that generates valid input-output responses. The protection scheme can be considered equivalent to a fail-safe system state (e.g., flashing red lights for a roadway stoplight system).
    4. Attack Delta Baseline – A targeted attack applied to the delta baseline that generates valid, degraded or unstable input-output responses.
- PLC instances – See Table 6 in Section 3.7 for description of ten PLC instances.
- Methods – Metrics collected from both the PLC and Petri net simulations.

Full factorial experimentation leads to: 4 (program categories) * 10 (PLC instances) * 2 (methods: PLC/Petri net) = 80 trials

3.11 Summary of Methodology

This chapter provided the goals of the experimentation and detailed the boundaries and approach. The goals of this research are to identify metrics that may assess a PLC's resiliency and applicability as trigger points in real-time hardware protection schemes. The boundaries of the SUT are the PLC; the CUT is the ladder logic

program that executes on the PLC. The approach consists of emulating ten PLC instances. Each PLC instance is comprised of four varying program types: *baseline, delta baseline, attack baseline* and *attack delta baseline*. Execution of the PLC instances occurs in both emulated hardware simulations and equivalent Petri net simulations.

The PLC simulations provide delta ladder logic metrics, and the Petri net simulations provide delta input-output behavioral metrics. The comparative analysis performed between all PLC program and Petri net metrics result in six comparative metrics which form the basis for quantitative analysis to achieve the stated goals:

- (baseline – delta baseline)
- (baseline – attack baseline)
- (baseline – attack delta baseline)
- (attack baseline – delta baseline)
- (attack baseline – attack delta baseline)
- (delta baseline – attack delta baseline)

IV. Results and Analysis

The purpose of this chapter is to document the analysis and results derived from the behavioral-based characterization process of PLCs. The primary focus is to determine the applicability of potential metrics that directly, or indirectly support, the four characteristics of the resilience framework as documented in Chapter 2. The metrics are a result of a general stochastic Petri net (GSPN) analysis for each of the Petri nets derived in Chapter 3.

Section 4.1 documents the results of the GSPN analysis for each of the Petri nets. Section 4.2 presents corollary analysis of the resulting metrics to identify statistically relevant observations. Section 4.3 reports the significant findings from the analysis of the metrics and applicability toward the resilience framework.

4.1 Results of Simulation Scenarios

This section describes the collection and organization of data produced from the experimentation. The behavioral-based characterization process yields equivalent Petri nets that facilitate analysis of the PLC input-output behavior. The Petri net simulation application, PIPEv4.0 (Bonet, 2007), is used to execute a GSPN analysis for each of the 40 Petri nets. The results of the GSPN analysis provide the reachability matrices of each Petri net.

4.1.1 Derivation of Tangible State Table

A tangible state table is a direct result of the characterization process and facilitates quantitative analysis. The GSPN analysis module of PIPEv4.0 produces a

table of states and lists the output characteristics for each state. Table 7 illustrates the tangible state table for one example baseline PLC instance. Note that analysis for the ten instances and varying programs are consistent with the example used for discussion.

Table 7: Tangible States for Baseline

	container full	deliver container	depart silo	fill container	ship container	stop deliver	stop depart	stop fill	stop ship
M0	0	0	0	0	0	1	0	0	0
M1	0	1	0	0	0	0	0	0	0
M2	0	0	0	1	0	0	0	0	0
M3	0	0	0	0	0	0	0	1	0
M4	1	0	1	0	0	0	0	0	0
M5	1	0	0	0	0	0	1	0	0
M6	0	0	0	0	1	0	0	0	0
M7	0	0	0	0	0	0	0	0	1

The rows represent the tangible states, and columns represent the places that characterize the output states. The elements of each matrix are marked as 0 or 1, which represent the absence or presence of a token, respectively. For example, state *M0* represents the Petri net marking in which the place *stop deliver* is active. The baseline PLC for this instance comprises eight distinct states.

4.1.2 Derivation of Reachability Graph

A reachability graph identifies all possible states and interactions for a given Petri net. PIPEv4.0 provides an analysis module that creates a reachability graph for each Petri net. Figure 11 illustrates the reachability graph consistent with the baseline PLC instance

referenced in Table 7. Note that analysis for the ten instances and varying programs are consistent with the example used for discussion.

In Figure 11, *S0* through *S7* inherit the output characteristics of *M0* through *M7,* respectively. The arrows pointing towards a state indicate the Petri net transition required to reach that state. For example, to transition from state *S0* to *S1*, the transition *2 to 3* must fire. Note that any given state must have at least one enabling transition; similarly any given state may have more than one enabling transitions.

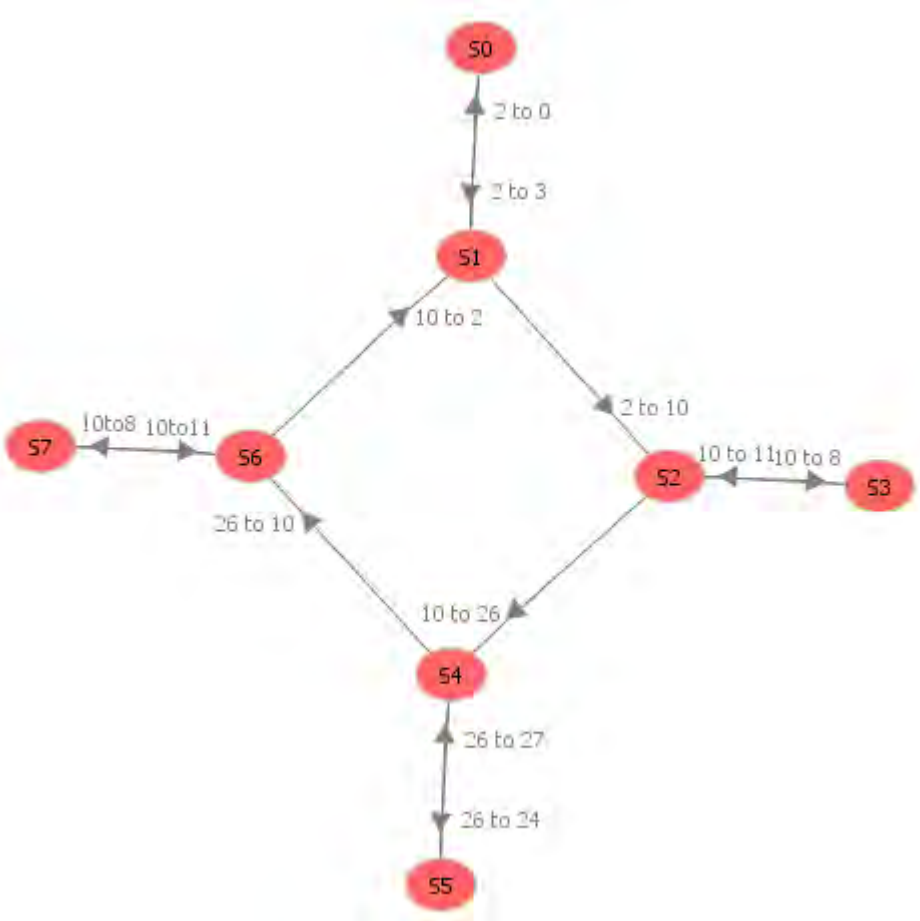

Figure 11: Reachability Graph for Baseline

4.1.3 Derivation of Reachability Matrix

A reachability matrix is a combination of results from the tangible state table and reachability graph. The reachability matrix identifies the output states and enabling input transitions for all potential markings of a given Petri net. Table 8 presents the reachability matrix consistent with tangible state table and reachability graph, presented in Table 7 and Figure 11, respectively. Note that analysis for the ten instances and varying programs are consistent with the example used for discussion.

Table 8: Reachable Markings for Baseline

	PN marking	m0	m1	m2	m3	m4	m5	m6	m7
Output Places	stop deliver	1							
	deliver container		1						
	stop fill				1				
	fill container			1					
	stop depart						1		
	depart silo					1			
	container full					1	1		
	stop ship								1
	ship container							1	
Input Transitions	2 --> 0	1							
	2 --> 3		1						
	2 --> 10			1					
	10 --> 2		1						
	10 --> 8				1				1
	10 --> 11			1				1	
	10 --> 26					1			
	26 --> 10							1	
	26 --> 24						1		
	26 --> 27					1			
	*input								

The reachability matrix organizes the input-output behavior into a numerical model that facilitates quantitative analysis. The columns present the potential markings for a given Petri net. The rows list the potential output behavior and input transitions. A numeral one in the element of the matrix indicates the specific combination of inputs and outputs that characterize any given marking for a Petri net. In Table 8, the Petri net marking *m4* is summarized with the output behavior of *depart silo* and *container full*. Marking *m4* may only be reached with the firing of either transition *10 to 26* or *26 to 27*.

4.1.4 Differentiating Between Reachability Matrices

Differentiating the input-output behaviors between any two PLC programs forms the basis for analysis of one set of metrics. The net difference between any two PLC programs is derived by comparing the number of dissimilar markings between each of their respective reachability matrices. Table 9 presents the reachability matrix for the attack program for PLC instance #1. Note that analyses for the ten instances are consistent with the example used for discussion.

The input transition **input* in Tables 8, and 9, denotes transitions in the attack scenario, which are not represented in the baseline case; **input* is important in differentiating PLC programs from the baseline case. Similarly, as seen in Table 9, **output-place* (e.g., **deliver container*) is important in differentiating specific PLC program cases. Table 9 shows the attack program where **deliver container's* decimal output value is different than the decimal output value produced by the baseline program's *deliver container* in Table 8.

Table 9: Reachable Markings for Attack (Instance #1)

	PN marking	m0	m1	m2	m3	m4	m5	m6	m7
Output Places	stop deliver	1							
	*deliver container		1						
	stop fill				1				
	fill container			1					
	stop depart						1		
	depart silo					1			
	container full					1	1		
	stop ship								1
	ship container							1	
Input Transitions	2 --> 0	1							
	2 --> 3		1						
	2 --> 10			1					
	10 --> 2		1						
	10 --> 8				1				1
	10 --> 11			1				1	
	10 --> 26					1			
	26 --> 10							1	
	26 --> 24						1		
	26 --> 27					1			
	*input								

The net difference is determined via a pair-wise comparison of the programs' potential markings. The following algorithm compares the pair-wise behavioral comparisons between two matrices such as *(baseline – attack baseline)*:

1. Select two matrices, *A* and *B*

2. If the number of potential markings between matrices is unequal, set the matrix with the highest number of potential markings as matrix *A*
3. *count* = the number of potential markings in *A*
4. *maxA* = *count*
5. *maxB* = number of potential markings in *B*
6. Set *X* and *Y* to zero
7. Compare all input-output parameters of *A(mX)* to *B(mY)*
    a) If *A(mX)* == *B(mY)*, *X* = *X* + *1*, *Y* = 0, *count* = *count* - 1; goto 7
    b) If *Y* < *maxB*, *Y* = *Y* + 1; goto 7
    c) If *X* < *maxA*, *X* = *X* + 1, Y = 0, goto 7
    d) Else goto 8
8. Return *count*

The resulting net difference between the baseline matrix and attack baseline matrix for PLC instance #1 is one. Table 10 presents the differences between each of the four program categories. For example, there are no observable differences in the input-output behavior between the following program pairings: (baseline – delta baseline), (baseline – attack delta baseline), and (delta baseline – attack delta baseline). There is exactly one observable difference between the remaining pairings.

Table 10: Net Difference in Input-Output Behavior (Instance #1)

baseline - attack baseline	baseline - delta baseline	baseline - attack delta baseline	attack baseline - delta baseline	attack baseline - attack delta baseline	delta baseline - attack delta baseline
1	0	0	1	1	0

4.1.5 Differentiating Between Ladder Logic

Differentiating the ladder logic between any two PLC programs forms the basis of analysis for a second set of metrics. The net difference between any two PLC programs is derived by comparing the number of dissimilar ladder logic symbols between the PLC programs. Figure 12 illustrates the baseline PLC Ladder Logic for the baseline program for PLC instance #1. Note that analyses for the ten instances are consistent with the example used for discussion.

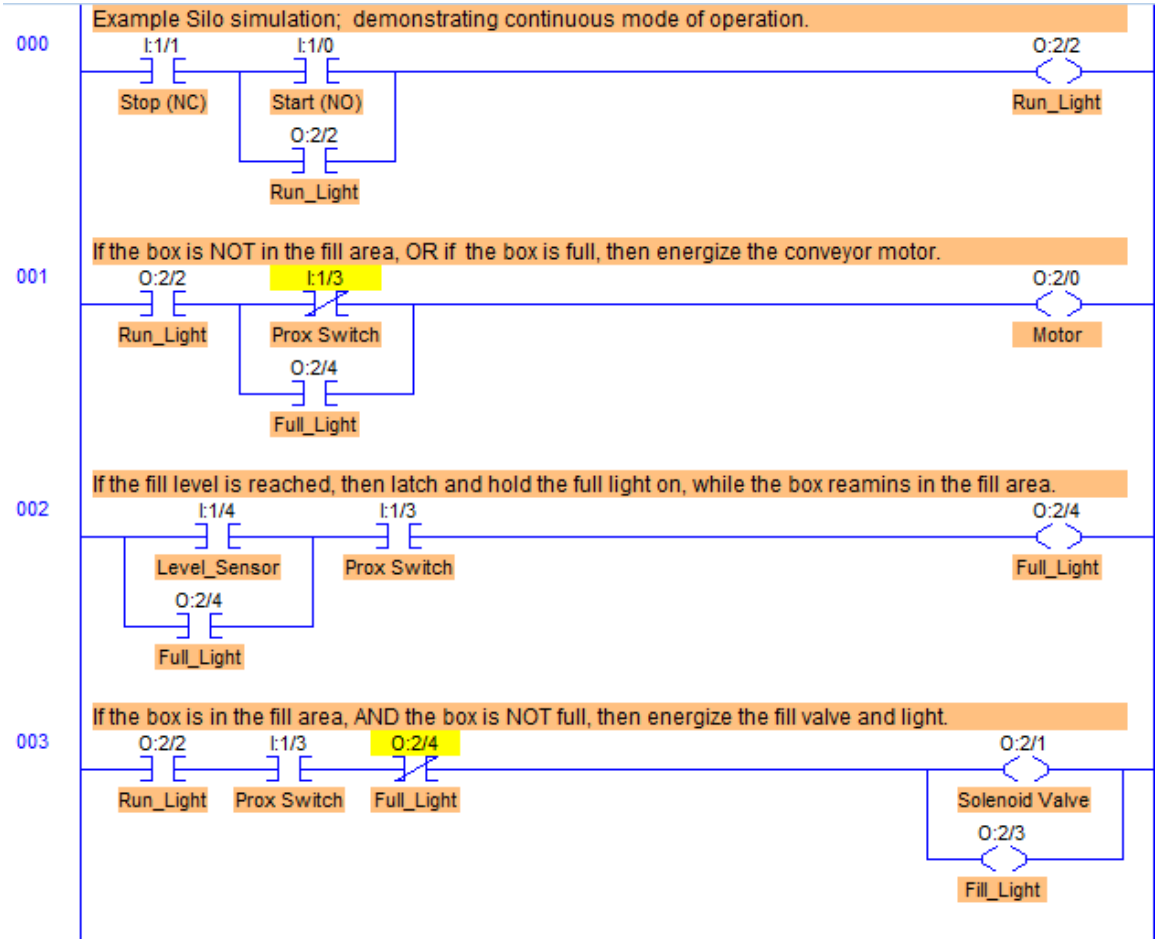

Figure 12: Baseline PLC Ladder Logic

To find the net difference between the *baseline* and *attack baseline* programs for PLC instance #1, the symbolic ladder logic deltas are counted (both the removal and addition of a symbol count as one change). For this example there is only one difference; the symbol for *Prox Switch*, present in rung *003* of the baseline (Figure 12), is removed from rung *003* of the attack baseline program in Figure 13. The converse is also true; the symbol for *Prox Switch*, absent in rung *003* of the attack program, is added to rung *003* of the baseline program.

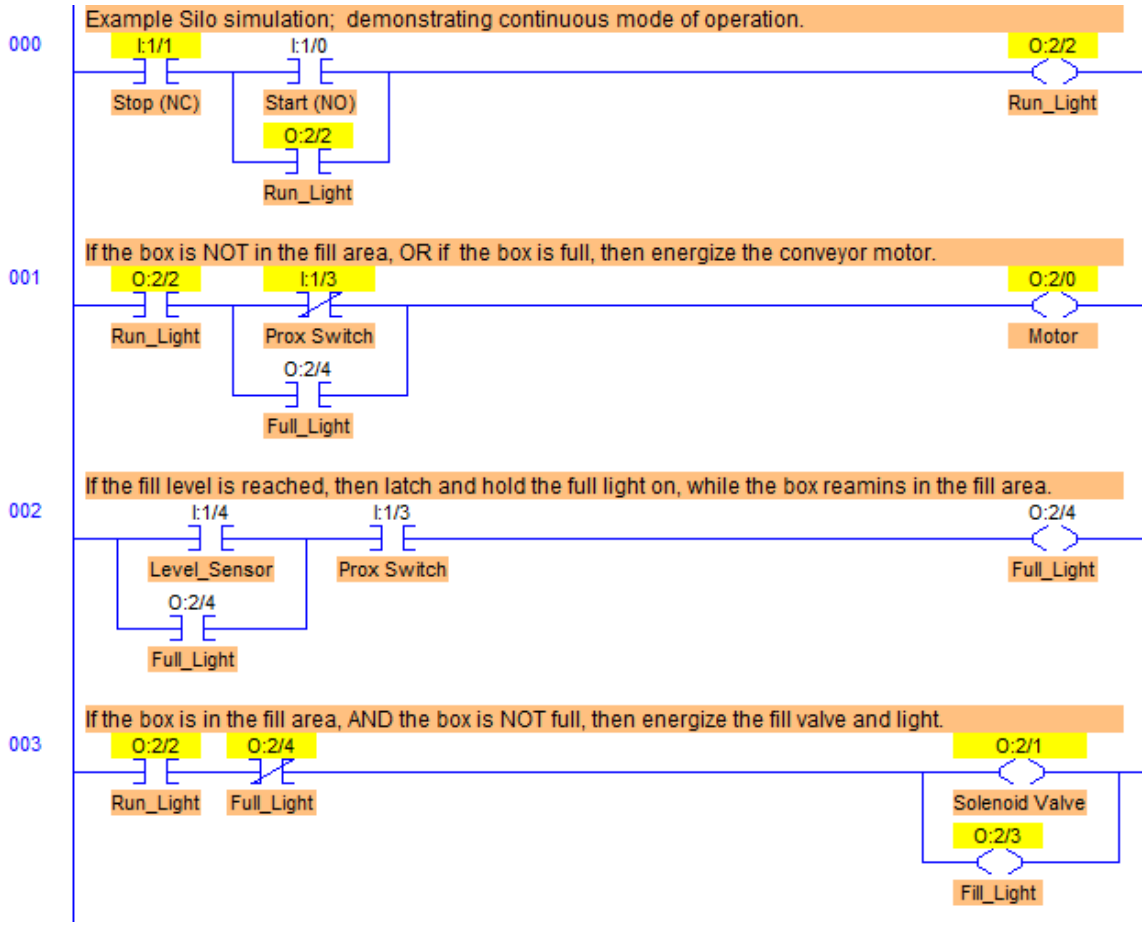

Figure 13: Attack Baseline PLC Ladder Logic for Instance #1

Table 11 presents the differences between each of the four program categories. There are two observable differences in the ladder logic between the program pairings (*baseline – attack delta baseline*) and (*attack baseline – delta baseline*). There is exactly one observable difference between the remaining pairings.

Table 11: Net Difference in Symbolic Ladder Logic (Instance #1)

baseline - attack baseline	baseline - delta baseline	baseline - attack delta baseline	attack baseline - delta baseline	attack baseline - attack delta baseline	delta baseline - attack delta baseline
1	1	2	2	1	1

4.1.6 Summary of Results

Each of the ten PLC instances specified in Chapter 3 result in 6 pair wise differentiations of the ladder logic and input-output behavior. The resulting net differences for each of the 60 cases are presented in Tables 12 and 13. The ladder logic programs for all 60 cases are available in Appendix A. Similarly, the state tables, reachability graphs, and matrices are available in Appendix B.

Table 12: Net Difference in Symbolic Ladder Logic

Instance #	baseline - attack baseline	baseline - delta baseline	baseline - attack delta baseline	attack baseline - delta baseline	attack baseline - attack delta baseline	delta baseline - attack delta baseline
1	1	1	2	2	1	1
2	1	1	0	2	1	1
3	1	2	3	3	2	1
4	1	2	3	3	2	1
5	2	2	4	4	2	2
6	3	4	7	7	4	3
7	2	2	0	4	2	2
8	5	6	7	11	6	5
9	6	10	10	6	4	2
10	10	12	10	4	2	2

Table 13: Net Difference in Input-Output Behavior

Instance #	baseline - attack baseline	baseline - delta baseline	baseline - attack delta baseline	attack baseline - delta baseline	attack baseline - attack delta baseline	delta baseline - attack delta baseline
1	1	0	0	1	1	0
2	6	0	0	6	6	0
3	4	0	0	4	4	0
4	3	0	0	3	3	0
5	4	0	0	4	4	0
6	7	0	0	7	7	0
7	5	0	0	5	5	0
8	8	0	0	8	8	0
9	1	0	4	1	4	4
10	1	0	0	1	1	0

The results of the ladder logic and input-output differentiation provide the basis for identifying quantitative metrics that support the resiliency framework. The results of these comparisons are analyzed for correlation between ladder logic deltas and input-output behavior deltas. Testing for correlation determines the delta ladder logic dictates the outcome of the input-output behavior deltas. The significant findings are then assessed against the resiliency framework to determine their applicability in potential real-time hardware solutions.

4.2 Analysis of Results

The data presented in Tables 12 and 13 represent two sets of metrics that measure the observable differences between PLC programs. The purpose of this section is to identify which, if any, of the metrics is most applicable to the resilience framework. Analysis for correlation is performed between the ladder logic and input-output behavior.

Then observations between any correlation and the differentiation tables are listed. The result is a subset of metrics that are most applicable to the resilience framework.

4.2.1 Scatter Plot of Results

The deltas in ladder logic and input-output behavior are visually tested for correlation in *R*. Figure 14 illustrates a scatter plot of the 60 data points derived from the data in Tables 12 and 13. For example, PLC instance #5, *(baseline – attack)*, generates the point (2, 4) on the scatter plot. Upon visual inspection, no apparent correlation between delta for ladder logic and input-output behavior exist. Note that there are less than 60 data points visible on Figure 14 due to overlap of several data points.

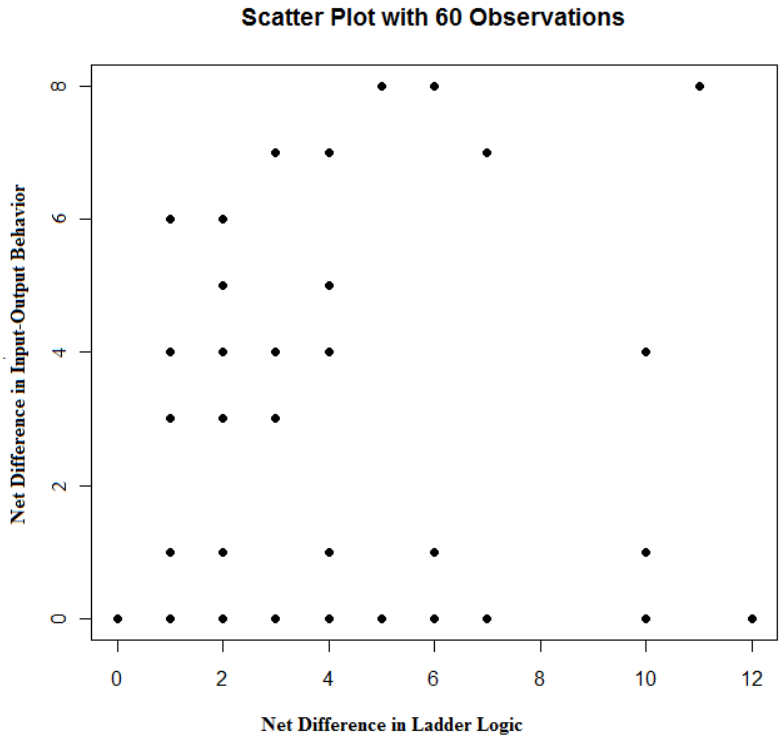

Figure 14: Scatter Plot Between Ladder Logic and I/O Deltas

4.2.2 Smooth Densities Plot of Results

A secondary plot to test for possible correlation is performed with the smooth density function of R. The smooth density function aids in visualizing any potential correlation within the overlapped data points (Figure 15). Upon visual inspection, a dense region exists at around the points (2, 0) and (2, 4). The remainder of the plot is similar to Figure 14 in that no other apparent correlations are visible.

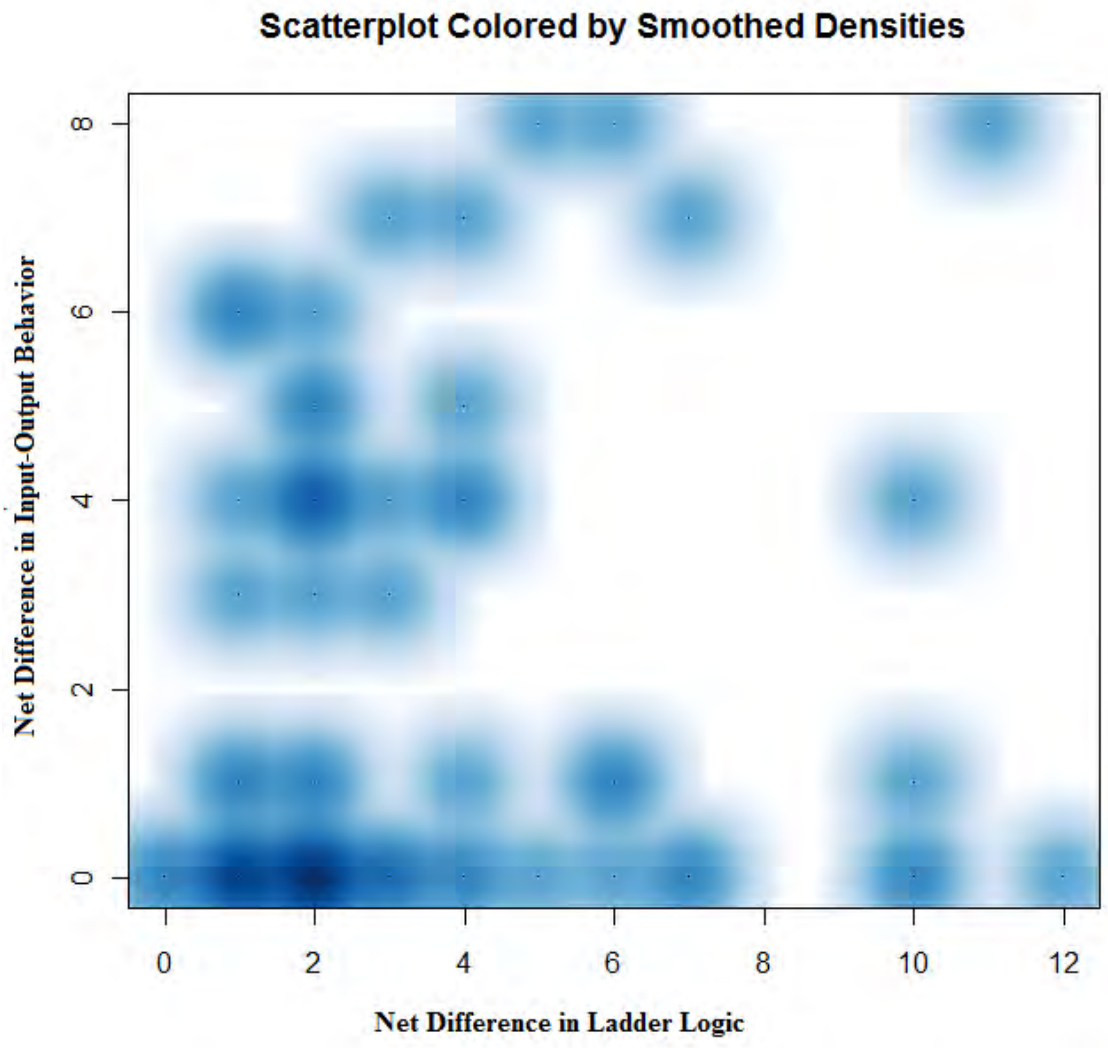

Figure 15: Scatter Plot Revealing Overlap Densities

The dense regions around the points (2, 0) and (2, 4) conflict one another and suggests that the differences of input-output behavior are independent from the differences in the ladder logic. This finding suggests that the delta ladder logic alone is insufficient in determining the input-output deltas of the PLC. The remaining regions of the density plot suggest no other observations to confirm or refute the previous suggestion. Visual inspection is useful in identifying consistent trends, but since this is absent in the plots a mathematical correlation of the data points is executed in *R*.

4.2.3 Correlation Results

A useful method to assess correlation between a set of variables is Spearman's rank order coefficient. Spearman's method of correlation is preferred over other methods, such as Pearson's, due to the non-linear patterns observed in the scatter plots (Bolboaca, 2006). The value for Spearman's rank order coefficient ranges from -1 to 1; values close to zero suggests no correlation exists between the variables and values close to ±1 suggests a corollary relationship exists.

Equation 1: Spearman's Rank Order Coefficient

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

Equation 1 defines Spearman's rank order coefficient, ρ, where:

d_i = difference in paired ranks

n = number of cases

Applying *R*'s Spearman correlation function to the data set resulted in $\rho = 0.14$; p-value = 0.3015. The ρ value suggests there is very little correlation between the deltas in ladder logic and the deltas of input-output behavior; however, the p-value of 0.3015 implies weak confidence in this hypothesis. Therefore the null hypothesis cannot be rejected beyond a confidence threshold of p = 0.30. Investigating the cause of the weak p-value reveals that the sample size of cases, *n*, is the root cause.

Spearman's ρ indicates a statistically weak correlation between the delta ladder logic and delta input-output data. This supports the visual observation of the plots which strongly suggests the relationship between the deltas in ladder logic and input-output behavior is strongly independent of one another. This result supports the assertion that *no correlation exists between the two sets of metrics*. This assertion is applied to additional observations that further refine the applicability of the metrics to the resilience framework.

4.2.4 Observations from Differentiation Tables and Correlation Analysis

Key observations from the correlation analysis and differentiation tables identify the most relevant subset of metrics which are applicable to the resilience framework. The net difference in ladder logic is a derivative of a PLC's internal characteristics which does not consistently quantify physical changes to the PLC's external state. Conversely, the net difference of input-output behavior is a derivative of a PLC's external characteristics which consistently quantifies external physical states. This distinction between the two metrics suggests the most significant metric resulting from the

experiments is the difference observed between the PLC programs' and instances' input-output behavior.

Since the input-output behavior metrics are self-sufficient, a focus to identify a key subset of these metrics is undertaken. Note the assumption asserted in Chapter 3 which states that all instances execute a successful attack that changes the physical input-output behavior of the system. The contribution of the ladder logic metrics as a complimentary metric is pursued in a latter section. The following are key observations of the differentiation tables for the input-output behavior (see Table 13):

- (baseline – delta baseline) is always equal to zero; this is by design of PLC instances/programs such that the baseline and delta baseline I/O behavior are consistent with each other.
- (baseline – attack delta baseline) and (delta baseline – attack delta baseline) are inconclusive; note that these metrics results in zero and non-zero values.
- (attack baseline – attack delta baseline) is inconclusive; note that this metric results in non-zero values for the ten instances; however, an instance can be created such that this metric results in zero, therefore it is inconclusive.
- (attack baseline – baseline) and (attack baseline – delta baseline) are conclusive; note that these metrics are always non-zero in the face of a successful attack and always zero in the face of an unsuccessful attack.

The most important observation is number four. The two metrics (attack baseline – baseline) and (attack baseline – delta baseline) are identified as the most discerning metrics in detecting input-output changes caused by successful attacks to the PLC's program. While the two metrics are equally discerning, the metric (attack baseline – delta baseline), is proposed as being more applicable in real-time hardware solutions to

improve the system's security posture. The metric, (attack baseline – delta baseline), also directly addresses two aspects of resiliency (i.e., detecting a change occurrence, and quantifying the degree of change occurrence) and supports mechanisms to minimize performance losses due to disruptive events.

4.2.5 Summary of Analysis

Corollary analysis between the ladder logic and input-outputs suggests no direct correlation exists; therefore, it can be reasoned that the net change of ladder logic within a PLC program is not a self-sufficient metric to assess a PLCs security performance with respect to the resilience framework. As suggested by the analysis and observations, the most significant metrics resulting from the experiments is the difference observed between the PLC programs' and instances' input-output behavior. The metrics, (attack baseline – baseline) and (attack baseline – delta baseline), exhibit roles both as a self-sufficient metrics and as a complimentary metrics to the deltas in ladder logic. Perhaps significant, is the finding that the metric, (attack baseline – delta baseline), is applicable to the resilience framework and potential real-time hardware solutions.

4.3 Significant Findings

The most significant finding in the analysis of the data is the metric (attack baseline – delta baseline). Indeed, the corollary analysis and differentiation observations suggest that this metric directly addresses two aspects of resiliency and supports mechanisms to minimize performance losses due to disruptive events. As a result of the metric's contribution to the resiliency framework, it may have potential application in real-time hardware solutions to improve a system's security posture. This section presents the applicability to both the resilience framework and real-time hardware solutions.

4.3.1 Applicability to the Resilience Framework

The following subsections summarize the application of the metric, (attack baseline – delta baseline), with respect to the four tenants of the resilience framework.

4.3.1.1 Self Awareness and Monitoring

The first tenant of the resilience framework is *the ability to anticipate a potentially disruptive event requires that the system has self-awareness of its baseline and is able to monitor its current state.*

The proposed metric identifies when physical input-output relationships deviate from its baseline. This metric may support one of two triggering mechanisms:

1. A quantity of deviations exceeding a threshold is identified by *count*
2. A violation against a whitelist (e.g., any matrix component output value of *attack baseline* is deviant)

4.3.1.2 Absorbing Disruptions

The second tenant of the resilience framework is *the ability to absorb potentially disruptive events requires that the system has mechanisms in place to minimize the amount, if any, of performance loss.*

The proposed metric in combination with the difference in ladder logic changes may assess a PLC's ability to absorb disruptive events:

1. If the input-output behavioral difference is zero, then the differences in ladder logic are treated as complimentary metrics to assess the inherent robustness of a PLC's ladder logic program.
2. If the input-output behavioral difference is greater than zero, then the differences seen in input-output behavior is self-sufficient and may be utilized to assess the PLC's overall absorption.

4.3.1.3 Adaptation

The third tenant of the resilience framework, *the ability to adapt, requires that the system has contingencies available that allow for flexible system adjustments to maintain operational availability.*

The proposed metric supports this by providing the triggering mechanisms necessary to initiate adaptive processes. Either of the triggering mechanisms outlined in Section 4.3.1.1 may support initiation of the adaptive process. Note that the adaptive processes may exist external to the PLC (e.g., requiring further coordination with additional hardware/software).

4.3.1.4 Recovery

The fourth tenant of the resilience framework is *the ability to recover from a disruptive event requires mechanisms, either automated or manual, that allow the system to perform functionalities consistent with its baseline.*

The proposed metric supports this by providing the triggering mechanisms necessary to initiate recovery processes. Either of the triggering mechanisms outlined in Section 4.3.1.1 may support initiation of the recovery process. Note that the recovery processes may exist external to the PLC (e.g., requiring further coordination with additional hardware or software).

4.3.2 Applicability to Real-Time Hardware Solutions

The metric's applicability to the resiliency framework cooperates well with potential real-time hardware solutions. This is an important notion because the protection mechanism may be an external, and preferably, parallel process. For example, Figure 16 illustrates a high-level Petri net that utilizes the input-output behavioral metric as the primary means of monitoring and detecting state security. The Petri net also illustrates architecture which supports absorptive, adaptive, and recovery features that are triggered when the metric *(attack baseline – delta baseline)* exceeds a threshold *count* delta or upon detection of deviant matrix values.

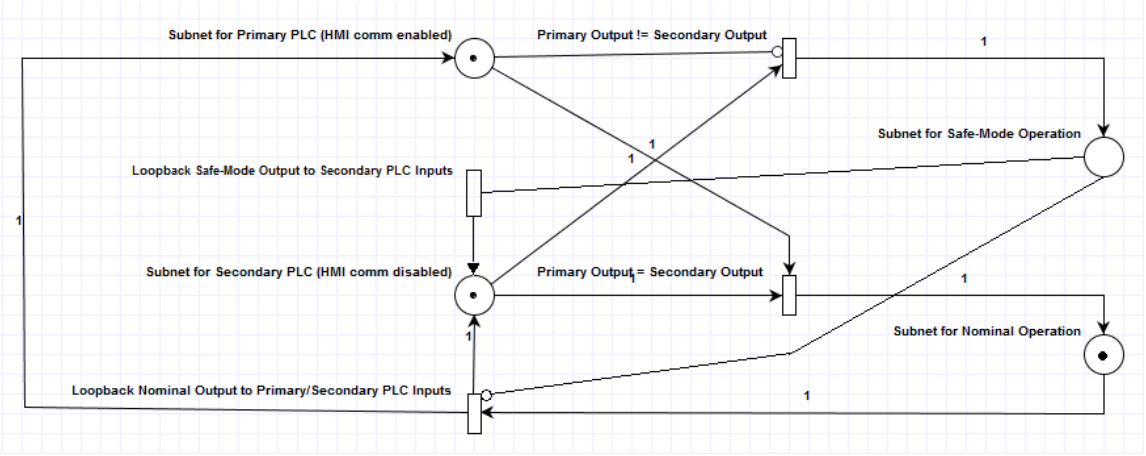

Figure 16: High-Level Petri Net Utilizing I/O Analysis (Nominal Operation)

The primary PLC executes the baseline program; however, the secondary protective PLC executes the delta baseline program and is isolated from direct communication links to the SCADA network. If deviation from expected behavior is detected, the primary PLC is prevented from contributing to the input-output state of the system, and the secondary PLC triggers a fail-safe operation. Figure 16 shows the system in nominal operation where the subnet for the primary PLC controls process flow; however, a deviation of input-output behavior as seen in Figure 17 transfers control of process flow to the subnet of the secondary PLC.

4.3.3 Summary of Findings

The metric (attack baseline – delta baseline) is the most significant result from the analysis of the data. The application to the resilience framework directly addresses two tenants (i.e., monitor and absorb), and supports the remaining two tenants (i.e., adapt and recover). A method of application is as a triggering mechanism which translates well

towards applications for potential real-time hardware solutions. An example high-level Petri net architecture is presented that utilizes the metric as the primary mechanism to transfer between nominal and safe operating modes.

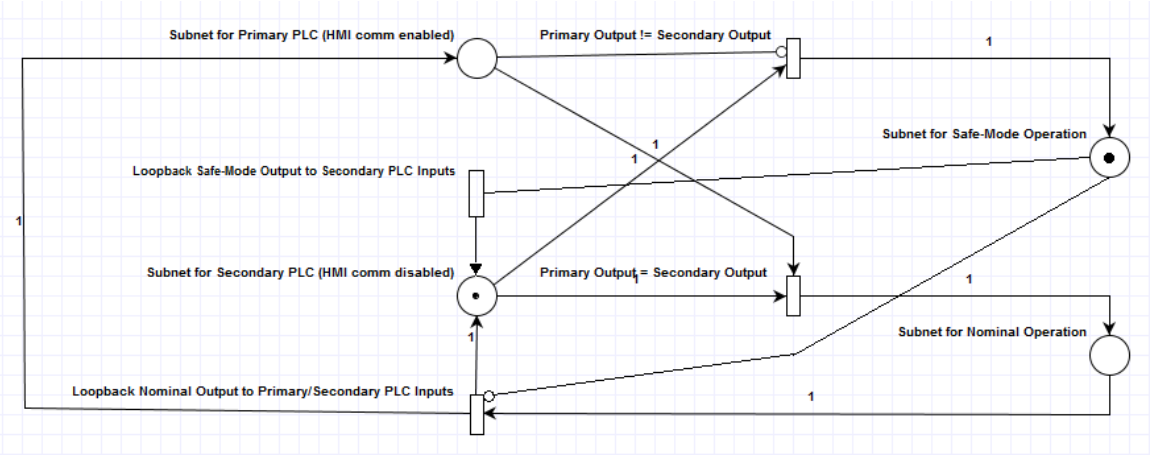

Figure 17: High-Level Petri Net Utilizing I/O Analysis (Safe-Mode Triggered)

4.4 Summary of Results and Analysis

The results in this chapter produced a quantitative means to assess the data generated from the Petri nets characterized from the PLC instances and programs. The formation of the differentiation matrices forms the basis for the set of metrics that can assess the performance of a PLC with respect to the resiliency framework. Analysis of the matrices suggests that the most significant metric is (attack baseline – delta baseline). This metric has direct applications to the resilience framework and subsequently to potential real-time hardware solutions.

V. Conclusions and Recommendations

This chapter summarizes the research effort and proposes considerations for future research. Section 5.1 presents a summary of the research goals and approach. Section 5.2 discusses considerations that progress the significant findings of this research. Section 5.3 provides concluding remarks.

5.1 Research Summary

The primary goal of this research is to identify metrics that may assess a PLC's resilience against malicious exploits. The complimentary goal of this research is to identify metrics that may be applicable as mechanisms for triggers that allow real-time hardware implementation. The experimental method to derive substantial metrics consisted of creating PLC instances, modeling equivalent Petri nets, and comparatively analyzing the data. The following sections discuss the experimental methodology, data analysis and effectiveness of meeting the research goals for this research effort.

5.1.1 Summary of Experimental Methodology

The experimental methodology presented in this research used a set of defined PLC instances that comprised of four versions of a baseline ladder logic program. The purpose is to emulate a variety of attacks that result in effectively altering the baseline performance of the PLC. The four baseline programs include:

- Baseline program – Executes nominal system processes
- Attack baseline program – Modifies baseline ladder logic; alters nominal system process execution

- Delta baseline program – Executes nominal system processes; robust against attack
- Attack delta baseline program – Modifies delta baseline ladder logic; *may* alter nominal system process execution

The input-output behavior of the programs during execution are observed and characterized into Petri nets. Non-deterministic simulation of the Petri nets generated the data necessary for the comparative analysis.

5.1.2 Summary of Analysis

Analysis of the Petri nets resulted in reachability matrices for each program. Differentiation between the matrices provided a summary of observable outcomes between the pair-wise analyses of reachability matrices. The pair-wise comparisons between the four program types comprise the six metrics of interest:

- (baseline – delta baseline)
- (baseline – attack baseline)
- (baseline – attack delta baseline)
- (attack baseline – delta baseline)
- (attack baseline – attack delta baseline)
- (delta baseline – attack delta baseline)

Two metrics, (attack baseline – baseline) and (attack baseline – delta baseline), are the most discerning metrics in detecting input-output modifications caused by successful attacks to a PLC's program. Of these two metrics, (attack baseline – delta

baseline), is proposed as having the most significant contributions to both the resilient framework and to real-time hardware applications.

5.1.3 Summary of Meeting Goals

The metric, (attack baseline – delta baseline), is determined to be most applicable to the research goals. The metric allows for self awareness by enabling the detection of deviations from nominal input-output state behavior. The metric may also quantify the absorptive performance of PLCs. Finally, the metric supports the adaptive, and recovery, qualities of resilience and it may enable potential mechanisms for triggers in real-world hardware applications.

5.2 Future Work

This section proposes three topics that may progress the findings from this research. The proposals include real-time hardware execution, benchmark utilization and an alternate view for the Petri net modeling of processes.

5.2.1 Real-time Application of Metrics in Hardware

The metric, (delta baseline – attack baseline), may be applicable to real-time hardware solutions to improve SCADA security. At the micro-level, it can be applied to PLCs identified as critical nodes of the system. Application of the metric requires that an additional hardware device operate concurrently with the baseline PLC. The additional hardware device would execute nominal processes exactly the same as the baseline; however, it implements additional logic that may trigger protective actions. The trigger

is initiated when a delta between the input-output characteristics of the baseline PLC and additional hardware is detected. Potential issues that may arise in real-world experimentation are related to timing delays between the two devices. This is due to the synchronous differentiation that the metric, (delta baseline – attack baseline), leverages. The potential for false negatives (e.g., attack to baseline not detected properly) is unlikely; however, false positives may result in excessive triggering of the protective actions.

5.2.2 Enhancing Benchmark Tools for Resilience

The metric, (delta baseline – attack baseline), provides a triggering mechanism to quantify changes to a system. This result may be integrated with the resilience curve presented by Wei (2009). The ability to measure aspects of the resilient curve is directly applicable to other work based on benchmarking tools for assessing resilience in systems (Almeida, 2010). The findings from this research for detecting change and quantifying absorptive rates between PLCs may apply to assessing resilience of similar systems at a micro-level benchmark. Finally, the application of the metric as a triggering mechanism may apply to assessing the adaptive and recovery aspects of resilience in a macro-level benchmark.

5.2.3 Alternate Experimentation Method Strictly Utilizing Petri Nets

Petri nets offer a powerful method of modeling and analyzing system processes (Peterson, 1981). SCADA system processes are deterministic in that they exhibit defined state boundaries; however, there exists an infinite sequence of state execution within the

boundaries. Non-deterministic analysis of Petri nets for a defined state space may provide an efficient method of modeling systems that execute discrete input-output states (Peng, 2004).

The method used in this research defined PLC programs prior to characterization into Petri nets. An alternate method is to solely define Petri nets as the basis for representing potential PLC instances. This facilitates the creation of a magnitude (complexity and quantity) of PLC instances which may result in more significant statistical analysis. Automating the generation and analysis of Petri nets would also eliminate potential sources of human error; the method utilized in this research consisted of several manual processes where human error is likely to be introduced.

5.3 Concluding Remarks

The behavioral-based method provides a practical means of assessing the security posture of a PLC against malicious code. The research demonstrates the means to quantify resiliency on the basis of monitoring, detecting, and absorbing intentional malicious actions. The ability to analyze the system in real-time, for nonconforming behavior at the PLC, enables security solutions for detecting and mitigating attacks at the system end points. Indeed, deriving metrics from input-out characterization incorporates a true representation of system state that cannot be deceived via alteration at the HMI or communication channel. This proposed method provides a measure of PLC performance against malicious code and provides a baseline for quantitative analysis of the security posture. Examining security at the micro level by focusing on field device and system functions provides a means for addressing and preparing for future Stuxnet-like attacks.

Appendix A

Baseline Program

Figure 18 illustrates the baseline program ladder logic and Figure 19 shows the baseline Petri net for all ten instances.

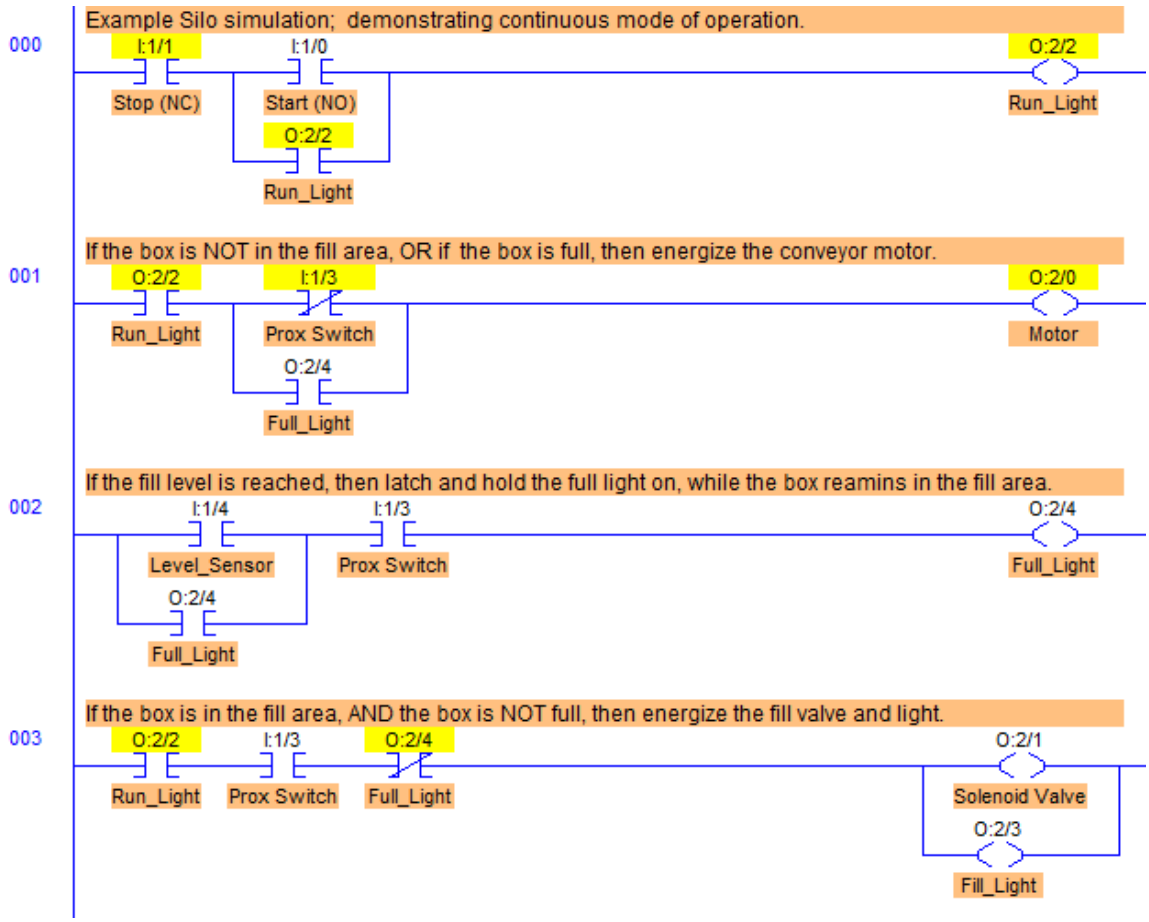

Figure 18: Ladder Logic for Baseline (all)

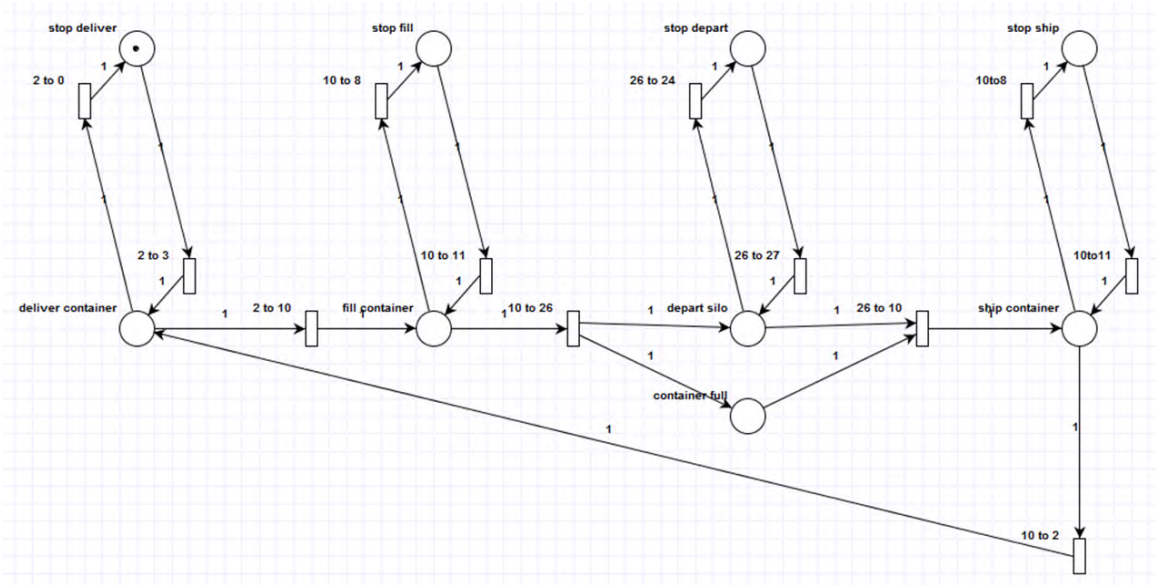

Figure 19: Petri Net for Baseline (all)

The formal definition for the Petri net illustrated in Figure 19 is C = *{P, T, I, O}*, such that:

- *P* = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- *T* = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- *I*(2 to 0) = {deliver container (5)}
- *I*(2 to 3) = {stop deliver (0)}
- *I*(2 to 10) = {deliver container (5)}
- *I*(10 to 2) = {ship container (21)}
- *I*(10 to 8) = {fill container (14), ship container (21)}
- *I*(10 to 11) = {stop fill (0), stop ship (0)}
- *I*(10 to 26) = {fill container (14)}
- *I*(26 to 10) = {depart silo (21), container full (20)}

- *I*(26 to 24) = {depart silo (21)}
- *I*(26 to 27) = {stop depart (0)}
- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {deliver container (5)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 2) = {deliver container (5)}
- *O*(10 to 8) = {stop fill (0), stop ship (0)}
- *O*(10 to 11) = {fill container (14), ship container (21)}
- *O*(10 to 26) = {depart silo (21), container full (20)}
- *O*(26 to 10) = {ship container (21)}
- *O*(26 to 24) = {stop depart (0)}
- *O*(26 to 27) = {depart silo (21)}

Attack Baseline Program for Instance 1

Figure 20 illustrates the attack baseline program ladder logic and Figure 21 shows the Petri net for instance 1.

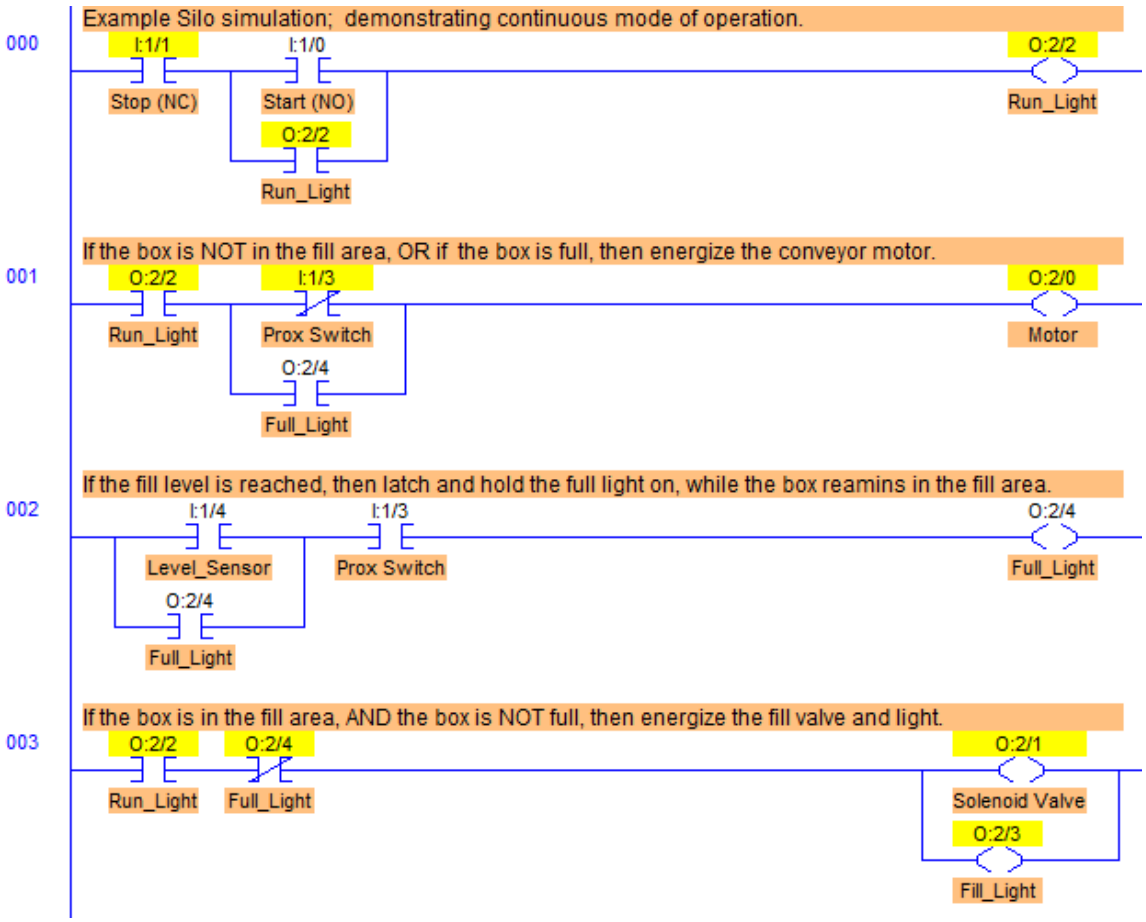

Figure 20: Ladder Logic for Attack Baseline (1)

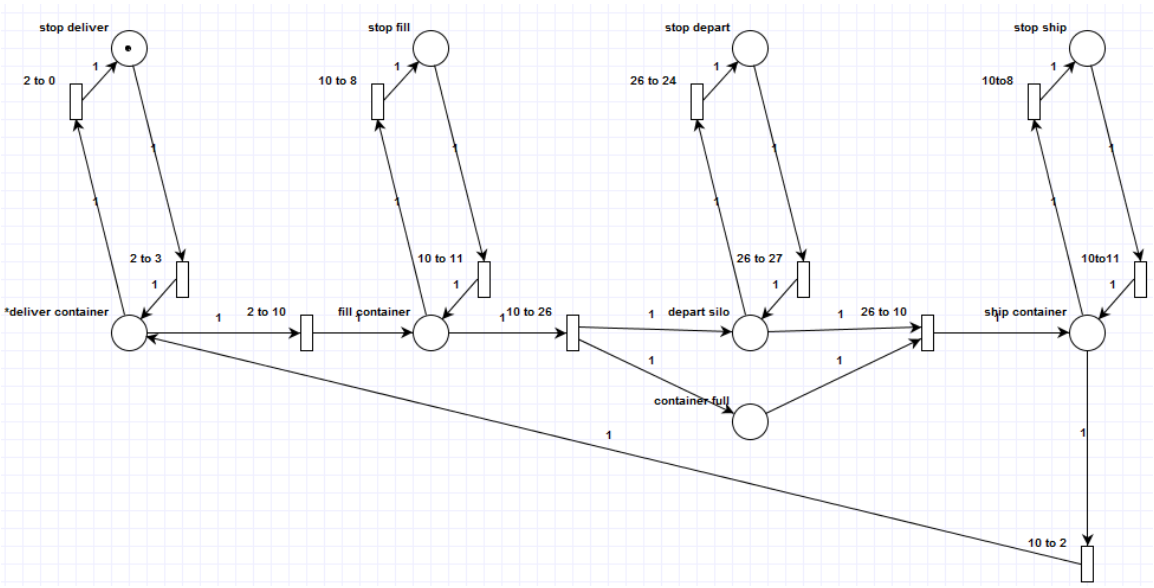

Figure 21: Petri Net for Attack Baseline (1)

The formal definition for the Petri net illustrated in Figure 21 is *C = {P, T, I, O}*, such that:

- *P* = {*deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- *T* = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- *I*(2 to 0) = {*deliver container (15)}
- *I*(2 to 3) = {stop deliver (0)}
- *I*(2 to 10) = {*deliver container (15)}
- *I*(10 to 2) = {ship container (21)}
- *I*(10 to 8) = {fill container (14), ship container (21)}
- *I*(10 to 11) = {stop fill (0), stop ship (0)}
- *I*(10 to 26) = {fill container (14)}

- *I*(26 to 10) = {depart silo (21), container full (20)}
- *I*(26 to 24) = {depart silo (21)}
- *I*(26 to 27) = {stop depart (0)}
- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {*deliver container (15)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 2) = {*deliver container (15)}
- *O*(10 to 8) = {stop fill (0), stop ship (0)}
- *O*(10 to 11) = {fill container (14), ship container (21)}
- *O*(10 to 26) = {depart silo (21), container full (20)}
- *O*(26 to 10) = {ship container (21)}
- *O*(26 to 24) = {stop depart (0)}
- *O*(26 to 27) = {depart silo (21)}

Attack Baseline Program for Instance 2

Figure 22 illustrates the attack baseline program ladder logic and Figure 23 shows the Petri net for instance 2.

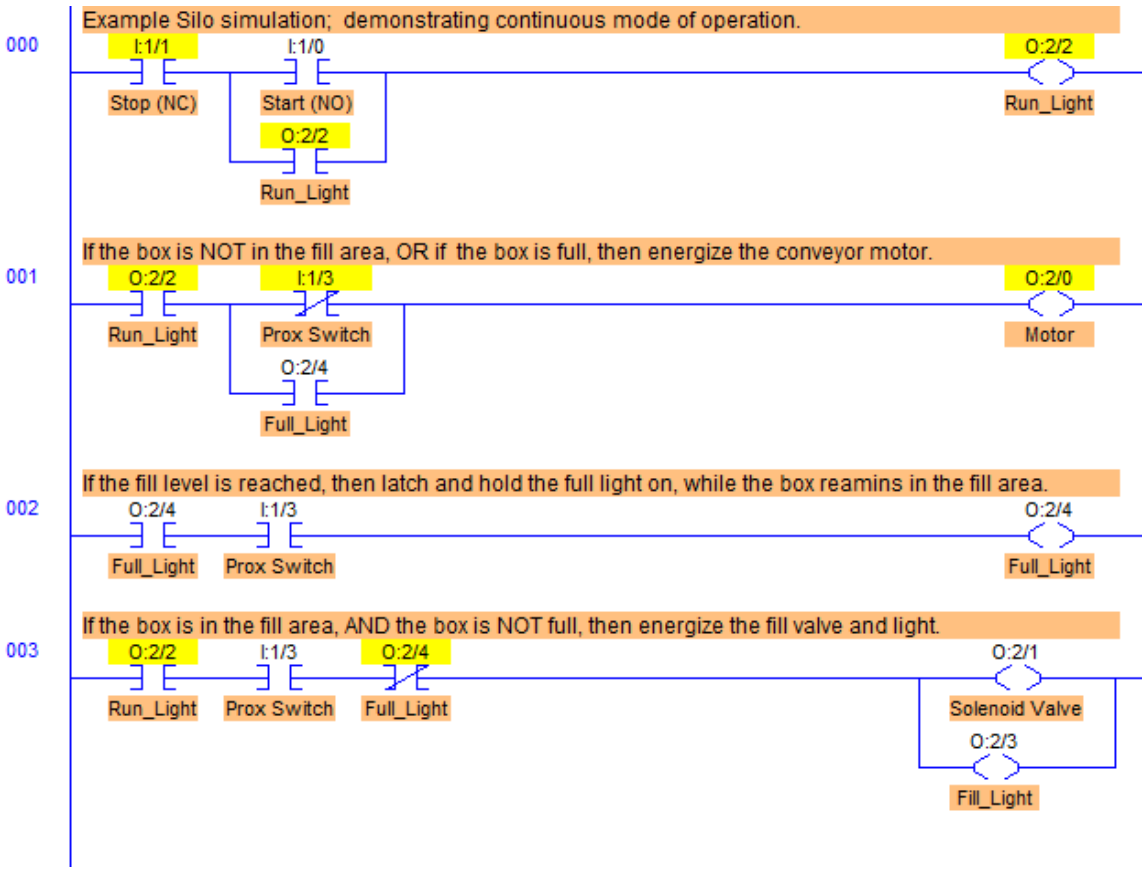

Figure 22: Ladder Logic for Attack Baseline (2)

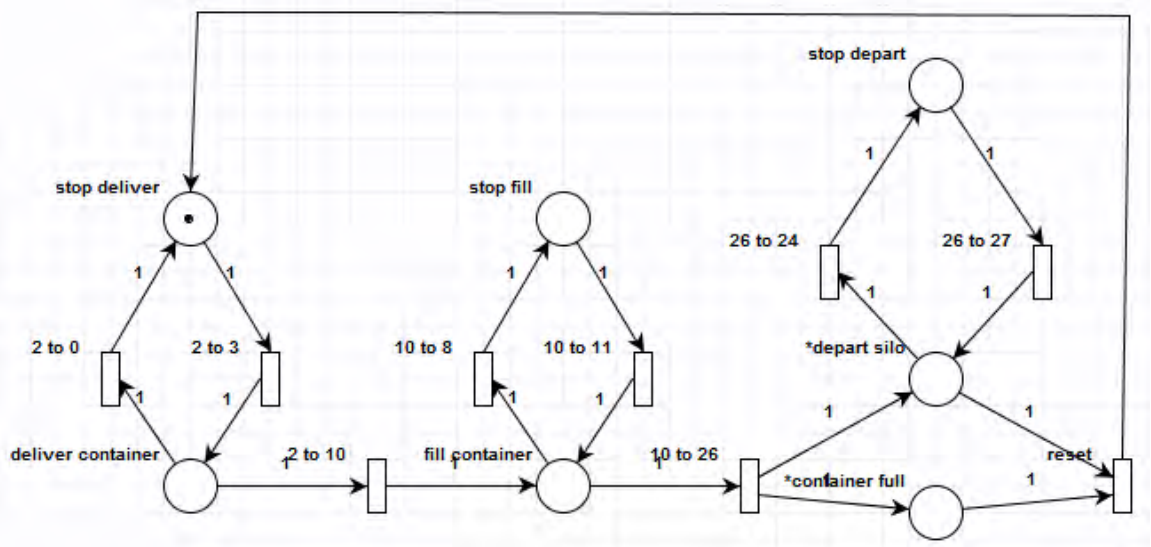

Figure 23: Petri Net for Attack Baseline (2)

The formal definition for the Petri net illustrated in Figure 23 is $C = \{P, T, I, O\}$, such that:

- P = {deliver container, stop deliver, fill container, stop fill, *container full, *depart silo, stop depart}
- T = {2 to 0, 2 to 3, 2 to 10, 10 to 8, 10 to 11, 10 to 26, 26 to 24, 26 to 27, reset}
- I(2 to 0) = {deliver container (5)}
- I(2 to 3) = {stop deliver (0)}
- I(2 to 10) = {deliver container (5)}
- I(10 to 8) = {fill container (14)}
- I(10 to 11) = {stop fill (0)}
- I(10 to 26) = {fill container (14)}
- I(26 to 24) = {*depart silo (14)}
- I(26 to 27) = {stop depart (0)}

- *I*(reset) = {*depart silo (14), *container full (14)}
- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {deliver container (5)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 8) = {stop fill (0)}
- *O*(10 to 11) = {fill container (14)}
- *O*(10 to 26) = {*depart silo (14), *container full (14)}
- *O*(26 to 24) = {stop depart (0)}
- *O*(26 to 27) = {*depart silo (14)}
- *O*(reset) = {stop deliver (0)}

Attack Baseline Program for Instance 3

Figure 24 illustrates the attack baseline program ladder logic and Figure 25 shows the Petri net for instance 3.

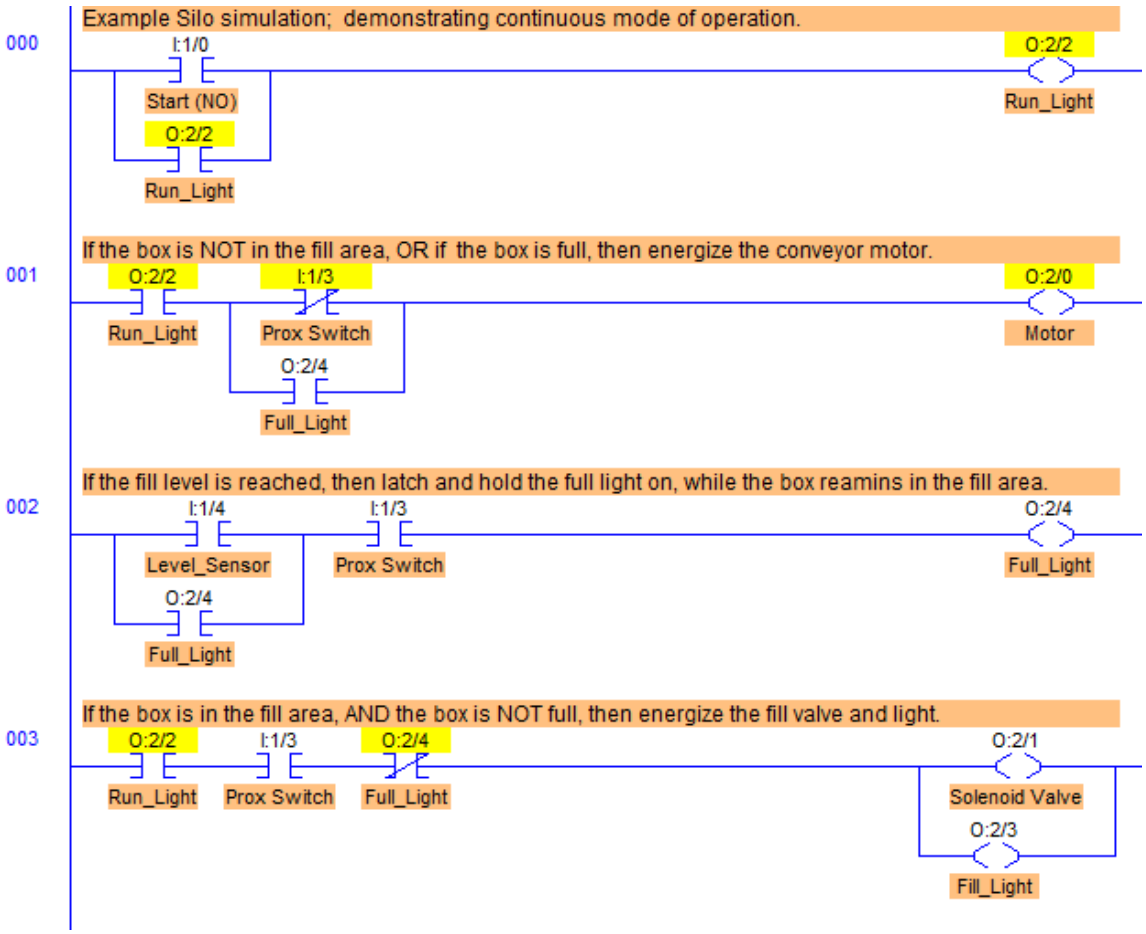

Figure 24: Ladder Logic for Attack Baseline (3)

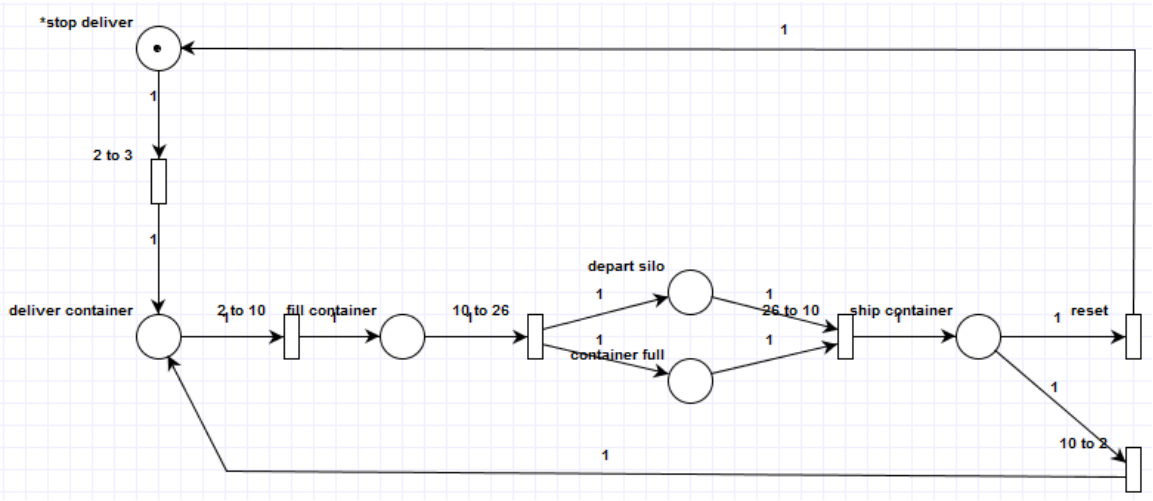

Figure 25: Petri Net for Attack Baseline (3)

The formal definition for the Petri net illustrated in Figure 25 is *C = {P, T, I, O}*, such that:

- *P* = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- *T* = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- *I*(2 to 3) = {*stop deliver (0)}
- *I*(2 to 10) = {deliver container (5)}
- *I*(10 to 2) = {ship container (21)}
- *I*(10 to 26) = {fill container (14)}
- *I*(26 to 10) = {depart silo (21), container full (20)}
- *I*(reset) = {ship container (20)}
- *O*(2 to 3) = {deliver container (5)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 2) = {deliver container (5)}

- *O*(10 to 26) = {depart silo (21), container full (20)}
- *O*(26 to 10) = {ship container (21)}
- *O*(reset) = {*stop deliver (0)}

Attack Baseline Program for Instance 4

Figure 26 illustrates the attack baseline program ladder logic and Figure 27 shows the Petri net for instance 4.

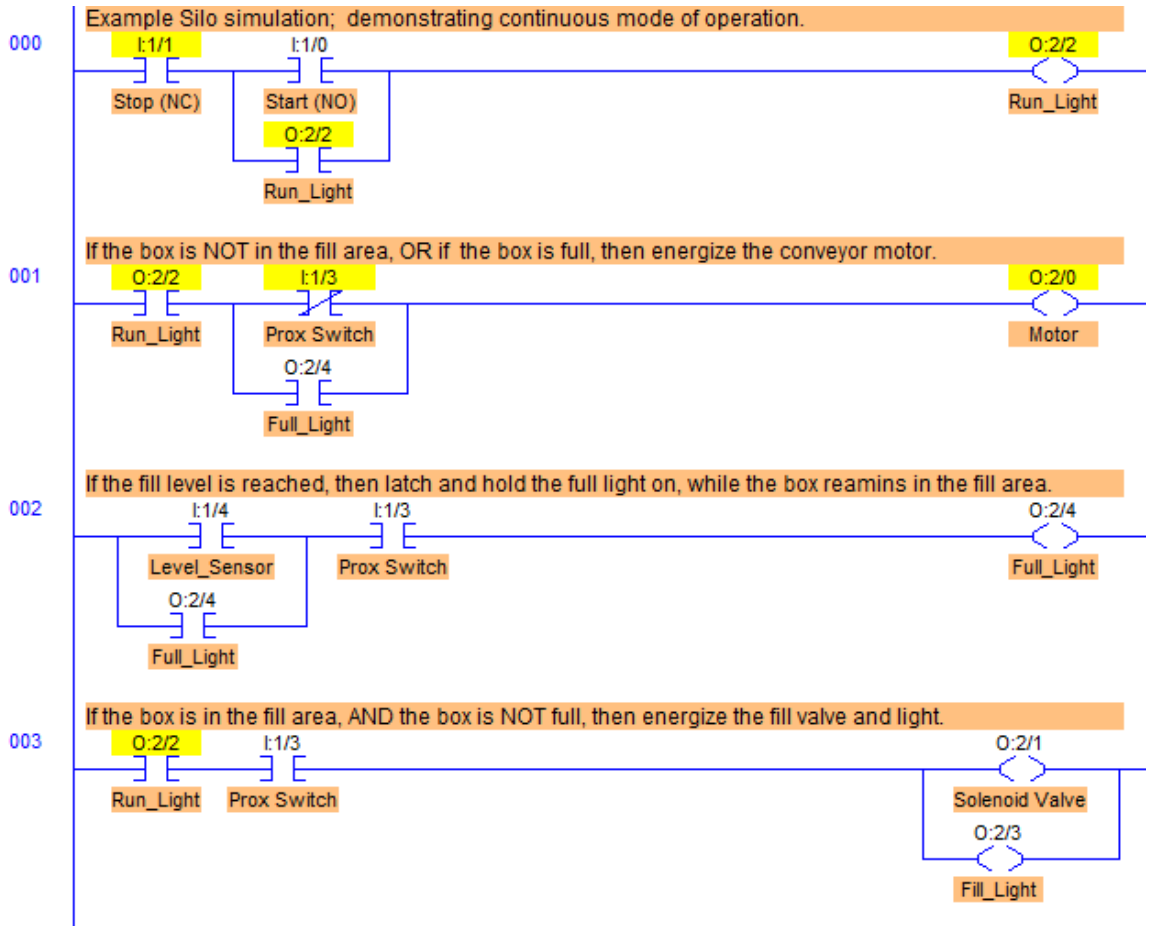

Figure 26: Ladder Logic for Attack Baseline (4)

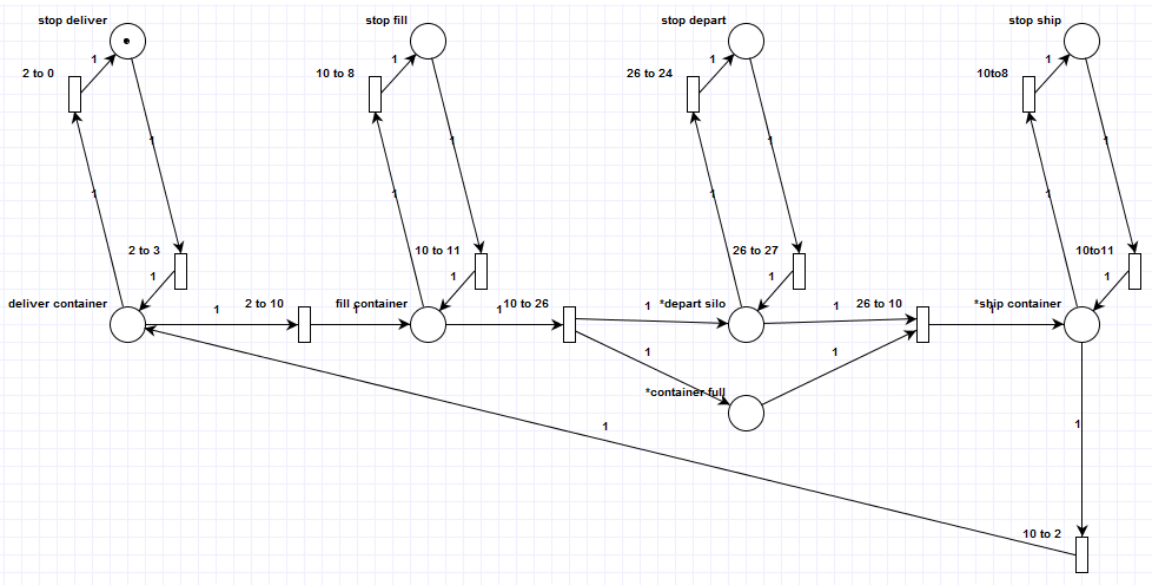

Figure 27: Petri Net for Attack Baseline (4)

The formal definition for the Petri net illustrated in Figure 27 is *C = {P, T, I, O}*, such that:

- *P* = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- *T* = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- *I*(2 to 0) = {deliver container (5)}
- *I*(2 to 3) = {stop deliver (0)}
- *I*(2 to 10) = {deliver container (5)}
- *I*(10 to 2) = {*ship container (31)}
- *I*(10 to 8) = {fill container (14), *ship container (31)}
- *I*(10 to 11) = {stop fill (0), stop ship (0)}
- *I*(10 to 26) = {fill container (14)}

- *I*(26 to 10) = {*depart silo (31), *container full (30)}
- *I*(26 to 24) = {*depart silo (31)}
- *I*(26 to 27) = {stop depart (0)}
- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {deliver container (5)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 2) = {deliver container (5)}
- *O*(10 to 8) = {stop fill (0), stop ship (0)}
- *O*(10 to 11) = {fill container (14), *ship container (31)}
- *O*(10 to 26) = {*depart silo (31), *container full (30)}
- *O*(26 to 10) = {*ship container (31)}
- *O*(26 to 24) = {stop depart (0)}
- *O*(26 to 27) = {*depart silo (31)}

Attack Baseline Program for Instance 5

Figure 28 illustrates the attack baseline program ladder logic and Figure 29 shows the Petri net for instance 5.

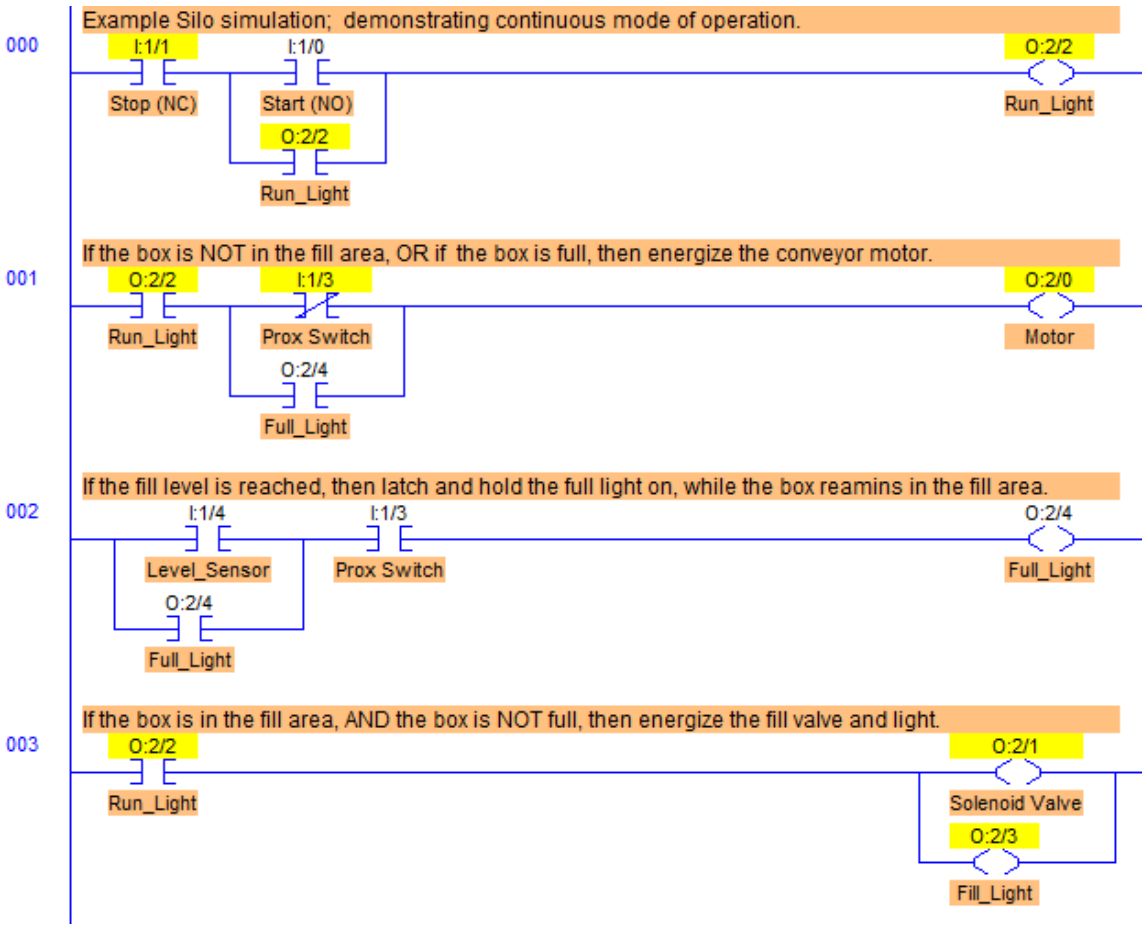

Figure 28: Ladder Logic for Attack Baseline (5)

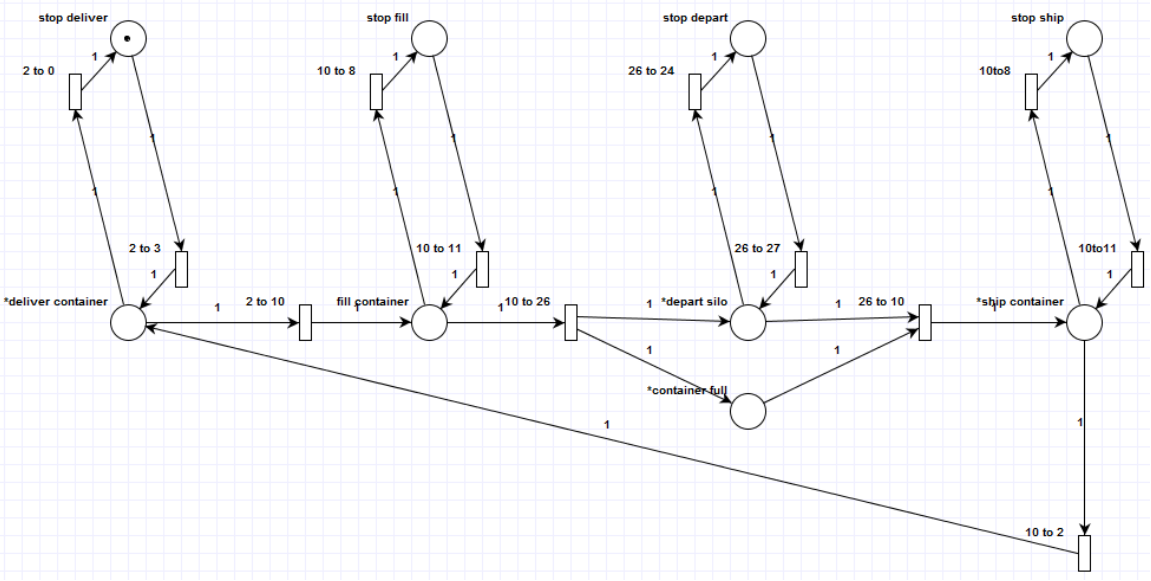

Figure 29: Petri Net for Attack Baseline (5)

The formal definition for the Petri net illustrated in Figure 29 is *C = {P, T, I, O}*, such that:

- *P* = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- *T* = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- *I*(2 to 0) = {*deliver container (15)}
- *I*(2 to 3) = {stop deliver (0)}
- *I*(2 to 10) = {*deliver container (15)}
- *I*(10 to 2) = {*ship container (31)}
- *I*(10 to 8) = {fill container (14), *ship container (31)}
- *I*(10 to 11) = {stop fill (0), stop ship (0)}
- *I*(10 to 26) = {fill container (14)}

- *I*(26 to 10) = {*depart silo (31), *container full (30)}
- I(26 to 24) = {*depart silo (31)}
- *I*(26 to 27) = {stop depart (0)}
- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {*deliver container (15)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 2) = {*deliver container (15)}
- *O*(10 to 8) = {stop fill (0), stop ship (0)}
- *O*(10 to 11) = {fill container (14), *ship container (31)}
- *O*(10 to 26) = {*depart silo (31), *container full (30)}
- *O*(26 to 10) = {*ship container (31)}
- *O*(26 to 24) = {stop depart (0)}
- *O*(26 to 27) = {*depart silo (31)}

Attack Baseline Program for Instance 6

Figure 30 illustrates the attack baseline program ladder logic and Figure 31 shows the Petri net for instance 6.

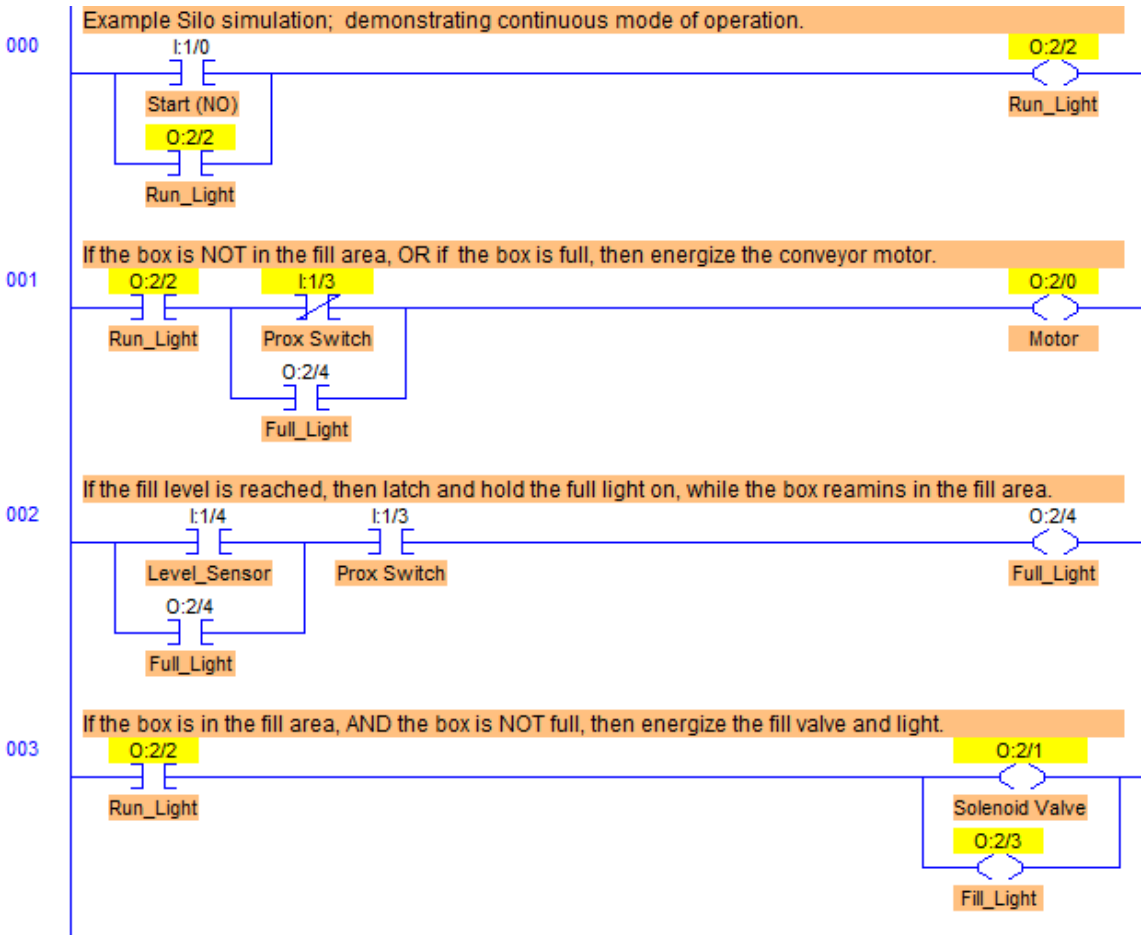

Figure 30: Ladder Logic for Attack Baseline (6)

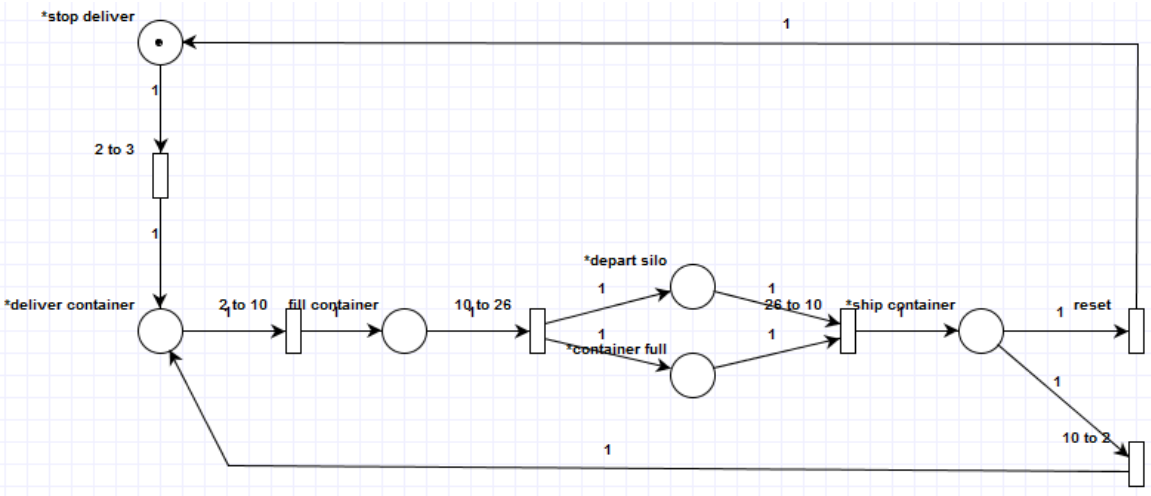

Figure 31: Petri Net for Attack Baseline (6)

The formal definition for the Petri net illustrated in Figure 31 is *C = {P, T, I, O}*, such that:

- *P* = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- *T* = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- *I*(2 to 3) = {stop deliver (0)}
- *I*(2 to 10) = {*deliver container (15)}
- *I*(10 to 2) = {*ship container (31)}
- *I*(10 to 26) = {fill container (14)}
- *I*(26 to 10) = {*depart silo (31), *container full (30)}
- *I*(reset) = {*ship container (31)}
- *O*(2 to 3) = {*deliver container (15)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 2) = {*deliver container (15)}

- *O*(10 to 26) = {*depart silo (31), *container full (30)}
- *O*(26 to 10) = {*ship container (31)}
- *O*(reset) = {*stop deliver (0)}

Attack Baseline Program for Instance 7

Figure 32 illustrates the attack baseline program ladder logic and Figure 33 shows the Petri net for instance 7.

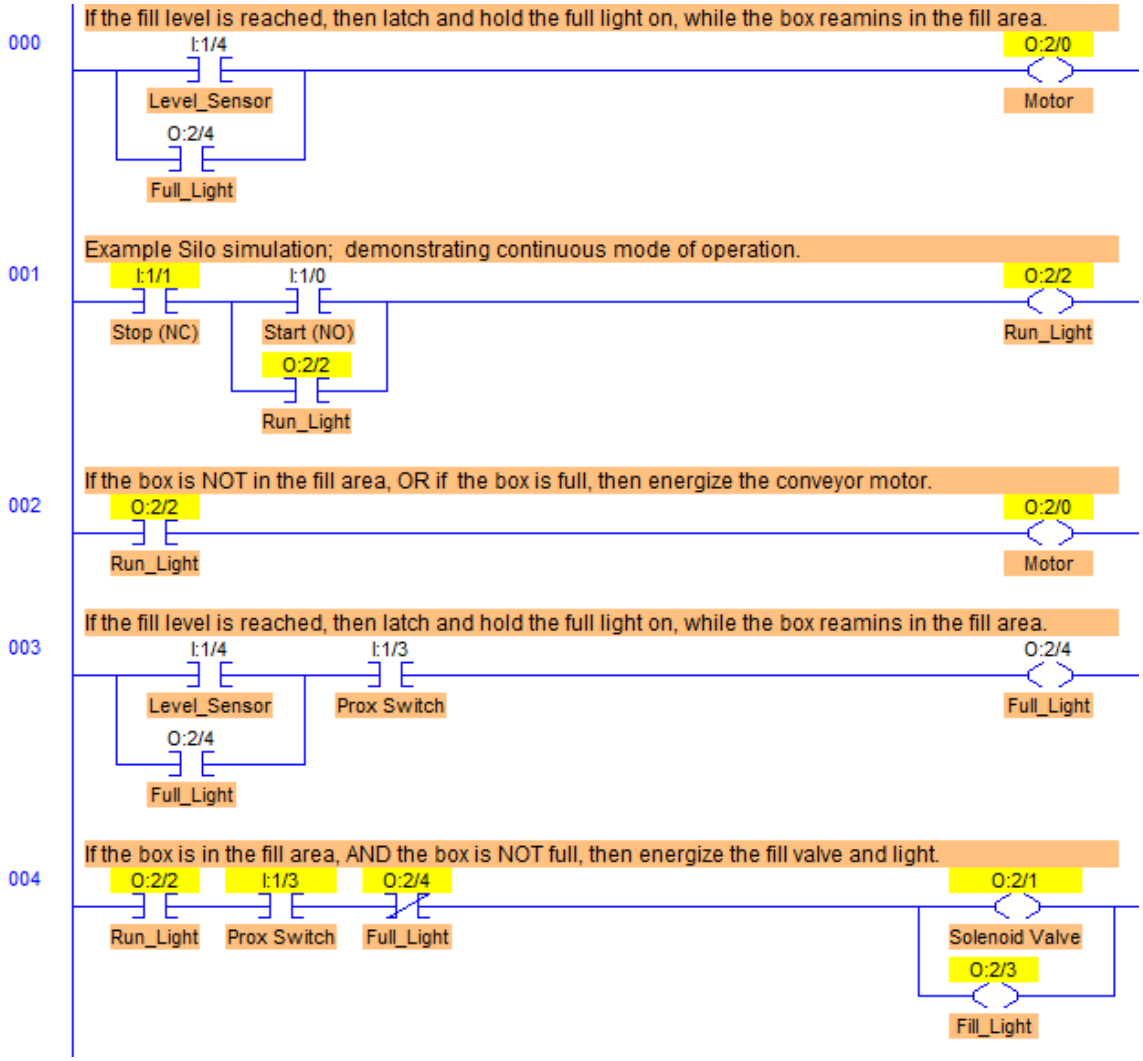

Figure 32: Ladder Logic for Attack Baseline (7)

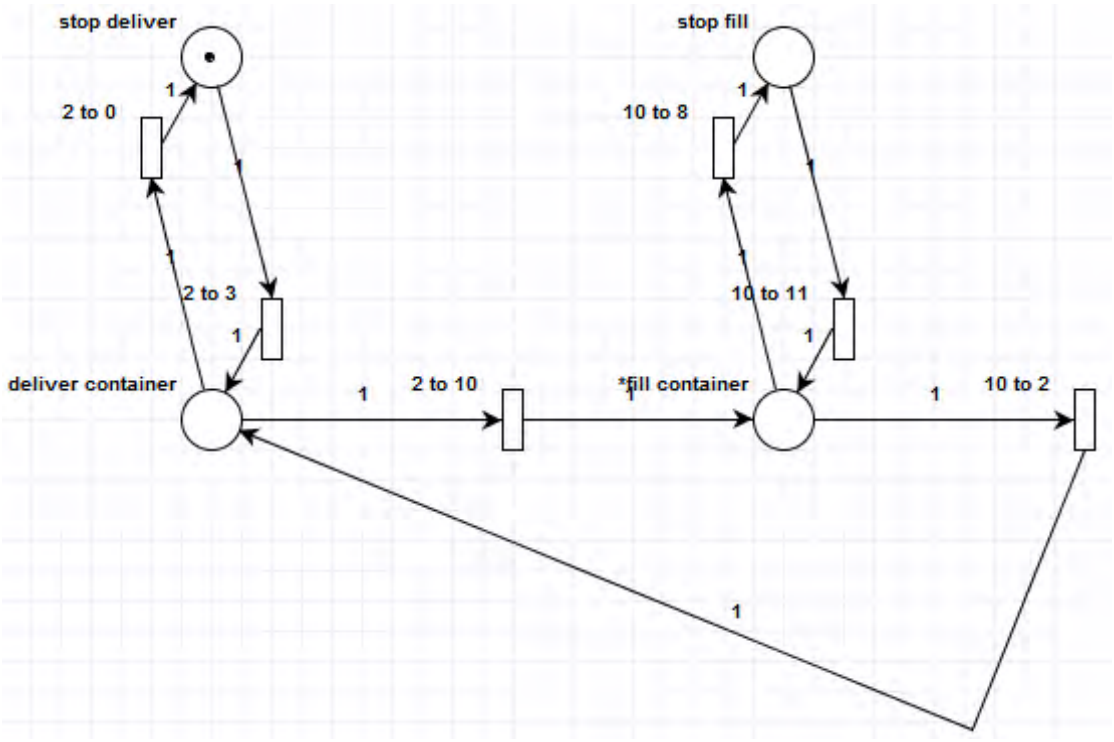

Figure 33: Petri Net for Attack Baseline (7)

The formal definition for the Petri net illustrated in Figure 33 is *C = {P, T, I, O}*, such that:

- *P* = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- *T* = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- *I*(2 to 0) = {deliver container (5)}
- *I*(2 to 3) = {stop deliver (0)}
- *I*(2 to 10) = {deliver container (5)}
- *I*(10 to 2) = {*fill container (15)}
- *I*(10 to 8) = {*fill container (15)}

- *I*(10 to 11) = {stop fill (0)}
- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {deliver container (5)}
- *O*(2 to 10) = {*fill container (15)}
- *O*(10 to 2) = {deliver container (5)}
- *O*(10 to 8) = {stop fill (0)}
- *O*(10 to 11) = {*fill container (15)}

Attack Baseline Program for Instance 8

Figure 34 illustrates the attack baseline program ladder logic and Figure 35 shows the Petri net for instance 8.

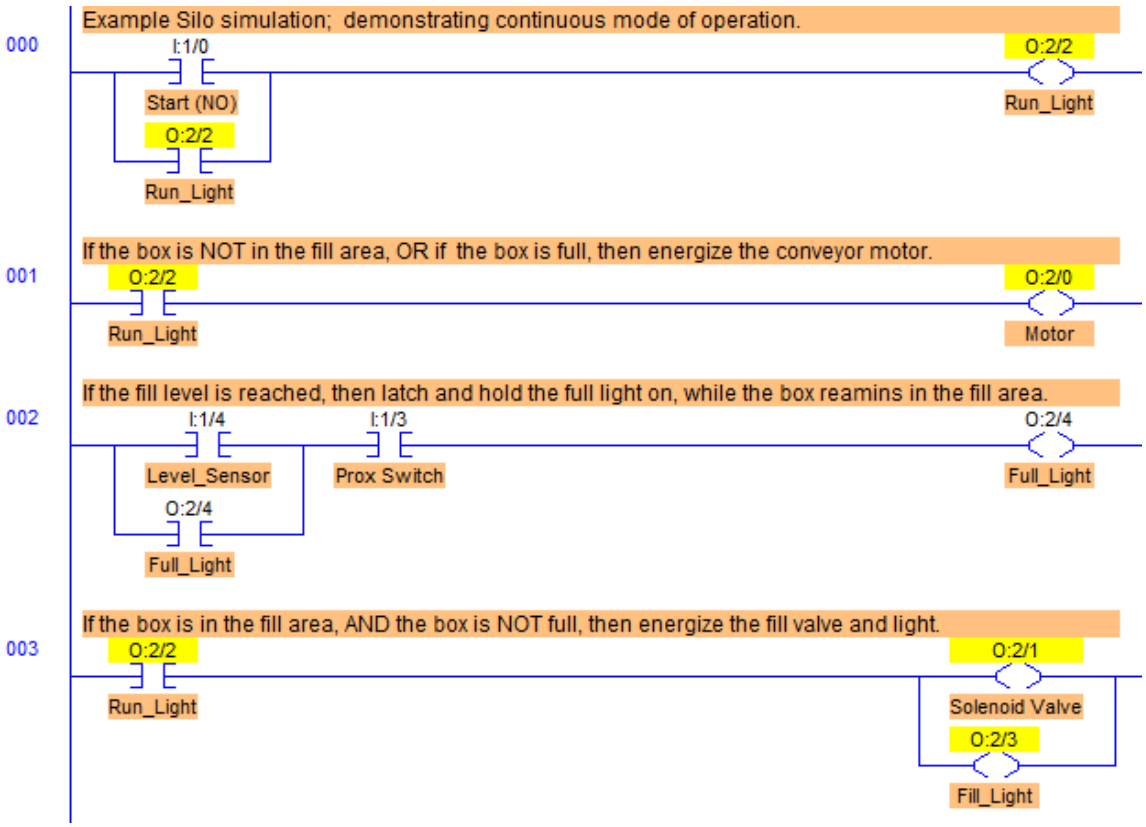

Figure 34: Ladder Logic for Attack Baseline (8)

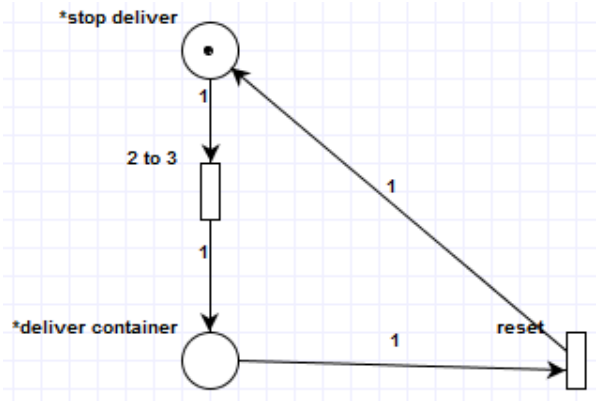

Figure 35: Petri Net for Attack Baseline (8)

The formal definition for the Petri net illustrated in Figure 35 is *C = {P, T, I, O}*, such that:

- *P* = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- *T* = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- *I*(2 to 3) = {*stop deliver (0)}
- *I*(reset) = {*deliver container (15)}
- *O*(2 to 3) = {*deliver container (15)}
- *O*(reset) = {*stop deliver (0)}

Attack Baseline Program for Instance 9

Figure 36 illustrates the attack baseline program ladder logic and Figure 37 shows the Petri net for instance 9.

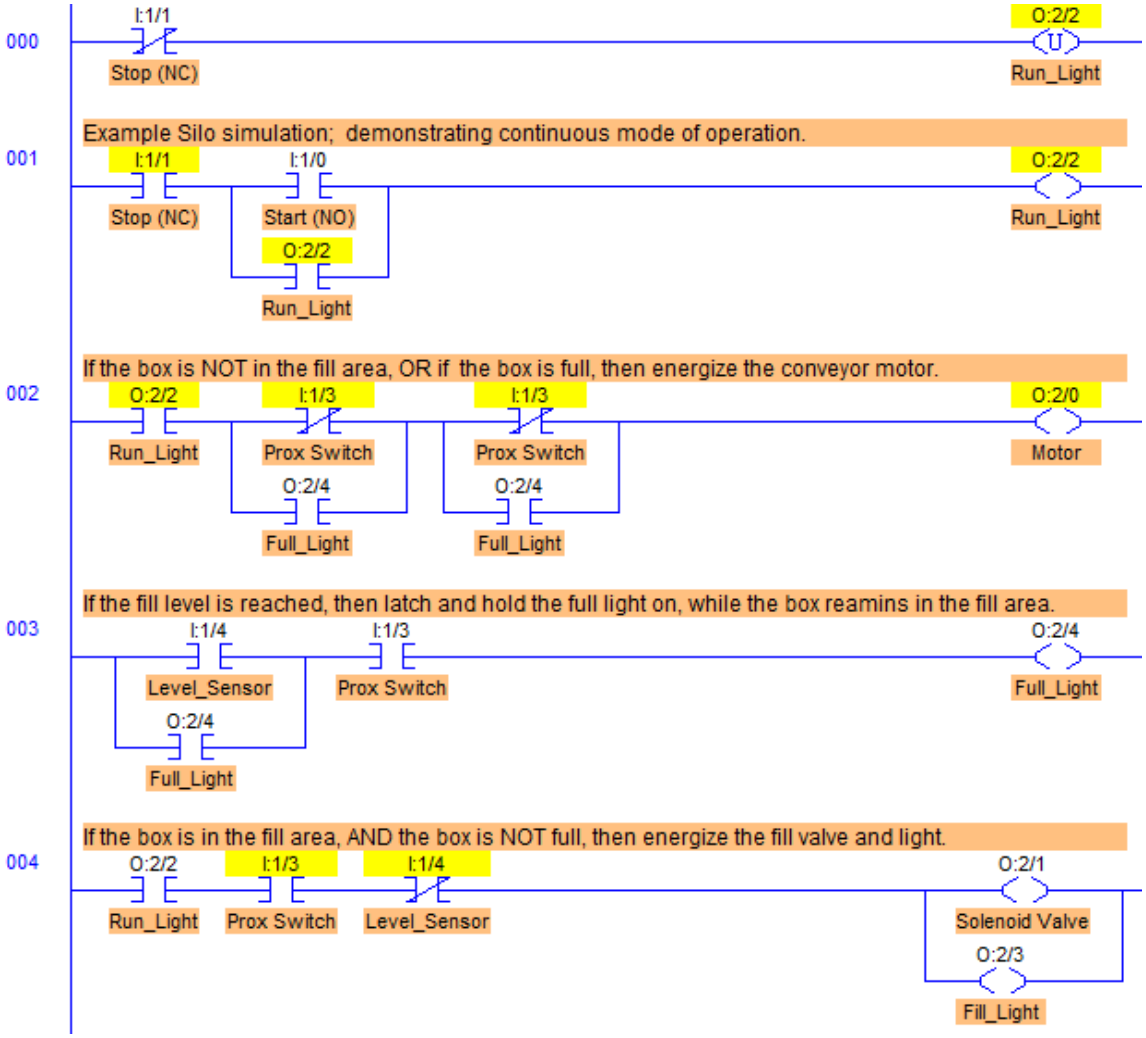

Figure 36: Ladder Logic for Attack Baseline (9)

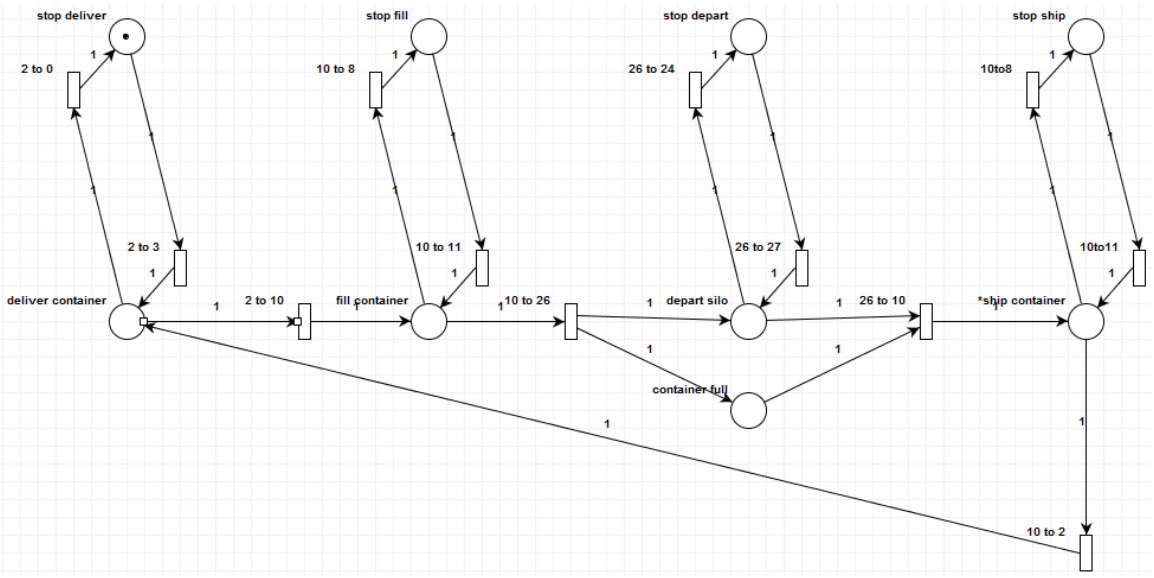

Figure 37: Petri Net for Attack Baseline (9)

The formal definition for the Petri net illustrated in Figure 37 is $C = \{P, T, I, O\}$, such that:

- P = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- T = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- I(2 to 0) = {deliver container (5)}
- I(2 to 3) = {stop deliver (0)}
- I(2 to 10) = {deliver container (5)}
- I(10 to 2) = {*ship container (31)}
- I(10 to 8) = {fill container (14), *ship container (31)}
- I(10 to 11) = {stop fill (0), stop ship (0)}
- I(10 to 26) = {fill container (14)}
- I(26 to 10) = {depart silo (21), container full (20)}

- *I*(26 to 24) = {depart silo (21)}
- *I*(26 to 27) = {stop depart (0)}
- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {deliver container (5)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 2) = {deliver container (5)}
- *O*(10 to 8) = {stop fill (0), stop ship (0)}
- *O*(10 to 11) = {fill container (14), *ship container (31)}
- *O*(10 to 26) = {depart silo (21), container full (20)}
- *O*(26 to 10) = {*ship container (31)}
- *O*(26 to 24) = {stop depart (0)}
- *O*(26 to 27) = {depart silo (21)}

Attack Baseline Program for Instance 10

Figure 38 illustrates the attack baseline program ladder logic and Figure 39 shows the Petri net for instance 10.

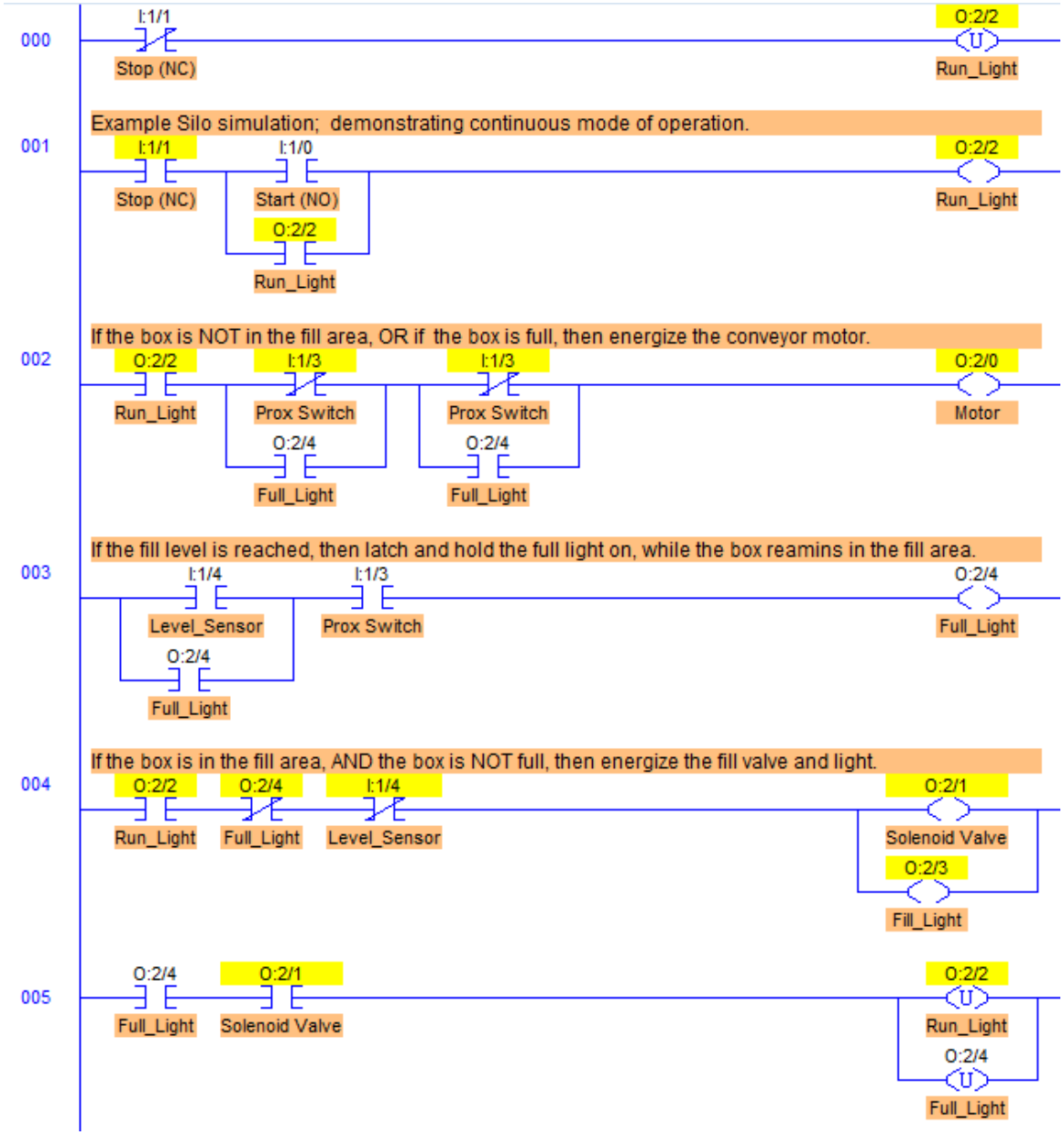

Figure 38: Ladder Logic for Attack Baseline (10)

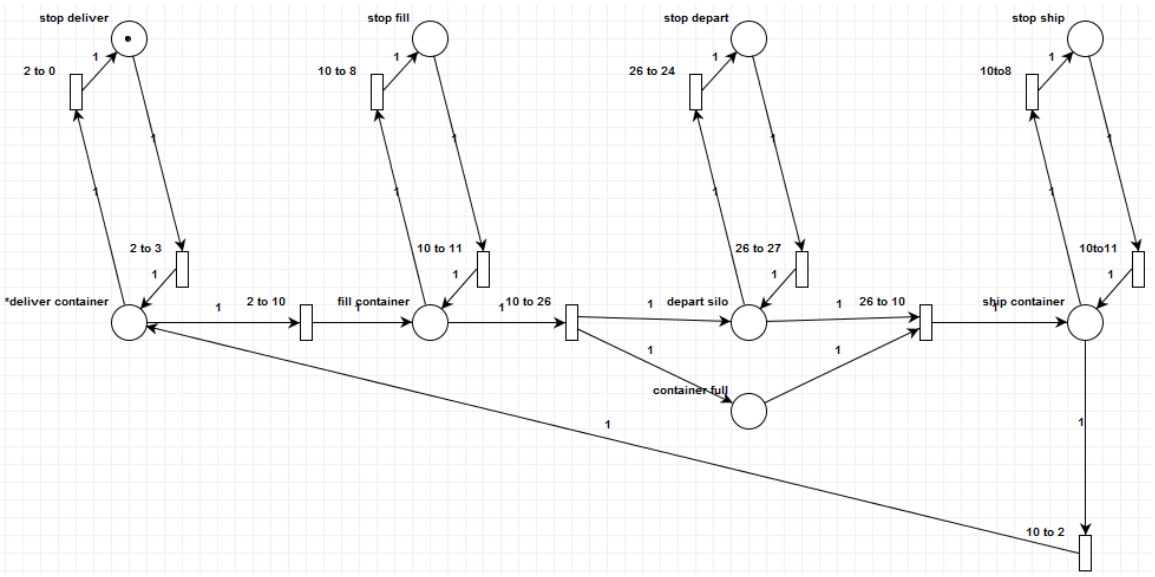

Figure 39: Petri Net for Attack Baseline (10)

The formal definition for the Petri net illustrated in Figure 39 is $C = \{P, T, I, O\}$, such that:

- P = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- T = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- I(2 to 0) = {*deliver container (15)}
- I(2 to 3) = {stop deliver (0)}
- I(2 to 10) = {*deliver container (15)}
- I(10 to 2) = {ship container (21)}
- I(10 to 8) = {fill container (14), ship container (21)}

- *I*(10 to 11) = {stop fill (0), stop ship (0)}
- *I*(10 to 26) = {fill container (14)}
- *I*(26 to 10) = {depart silo (21), container full (20)}
- *I*(26 to 24) = {depart silo (21)}
- *I*(26 to 27) = {stop depart (0)}
- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {*deliver container (15)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 2) = {*deliver container (15)}
- *O*(10 to 8) = {stop fill (0), stop ship (0)}
- *O*(10 to 11) = {fill container (14), ship container (21)}
- *O*(10 to 26) = {depart silo (21), container full (20)}
- *O*(26 to 10) = {ship container (21)}
- *O*(26 to 24) = {stop depart (0)}
- *O*(26 to 27) = {depart silo (21)}

Delta Baseline Program for Instance 1

Figure 40 illustrates the delta baseline program ladder logic for instance 1. The equivalent Petri net is similar to the baseline program shown in Figure 19.

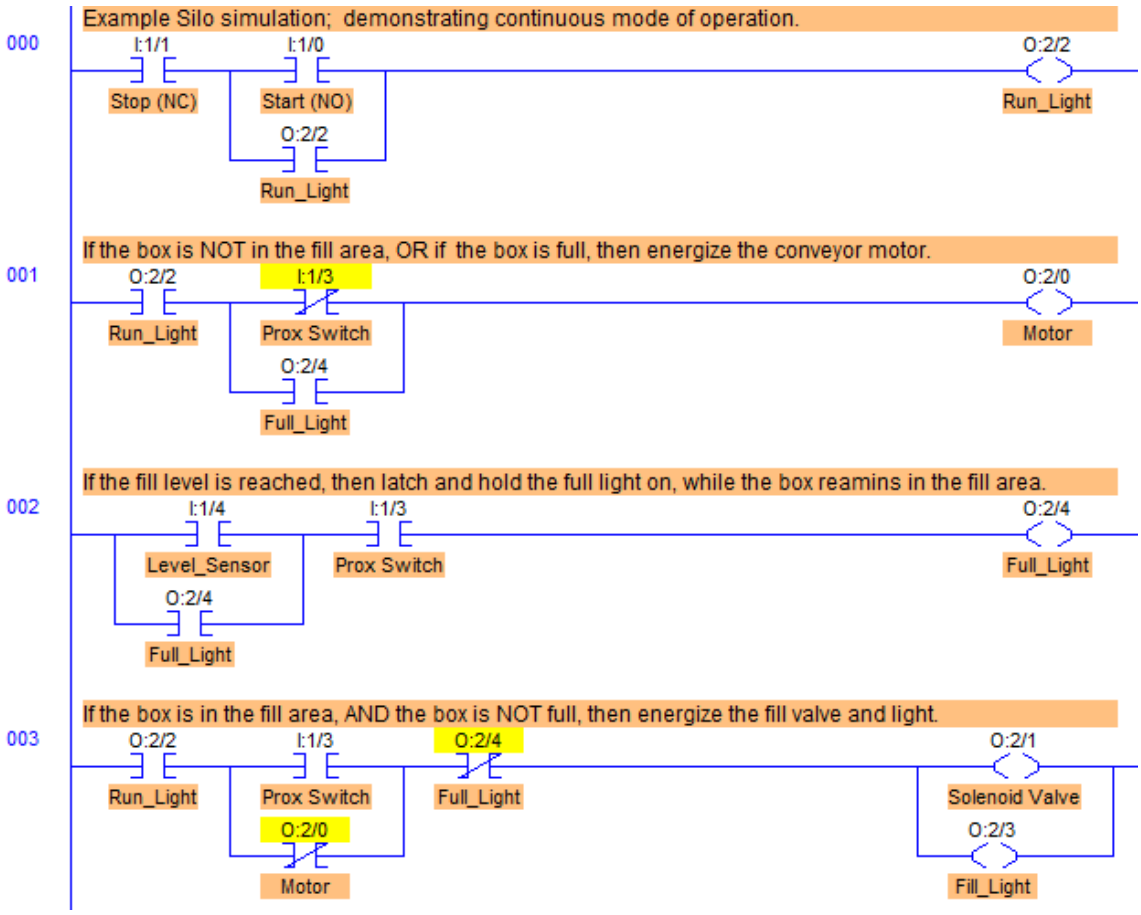

Figure 40: Ladder Logic for Delta Baseline (1)

Delta Baseline Program for Instance 2

Figure 41 illustrates the delta baseline program ladder logic for instance 2. The equivalent Petri net is similar to the baseline program shown in Figure 19.

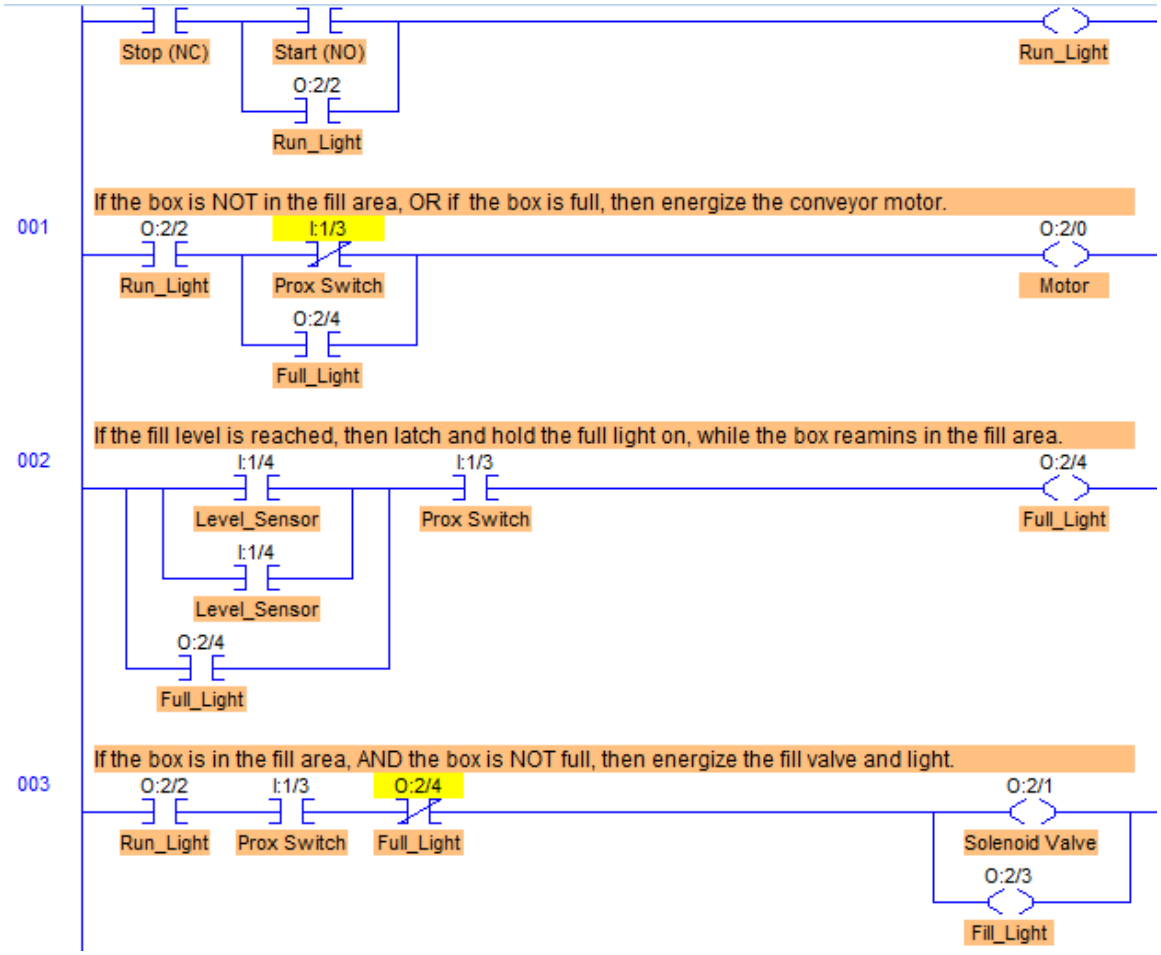

Figure 41: Ladder Logic for Delta Baseline (2)

Delta Baseline Program for Instance 3

Figure 42 illustrates the delta baseline program ladder logic for instance 3. The equivalent Petri net is similar to the baseline program shown in Figure 19.

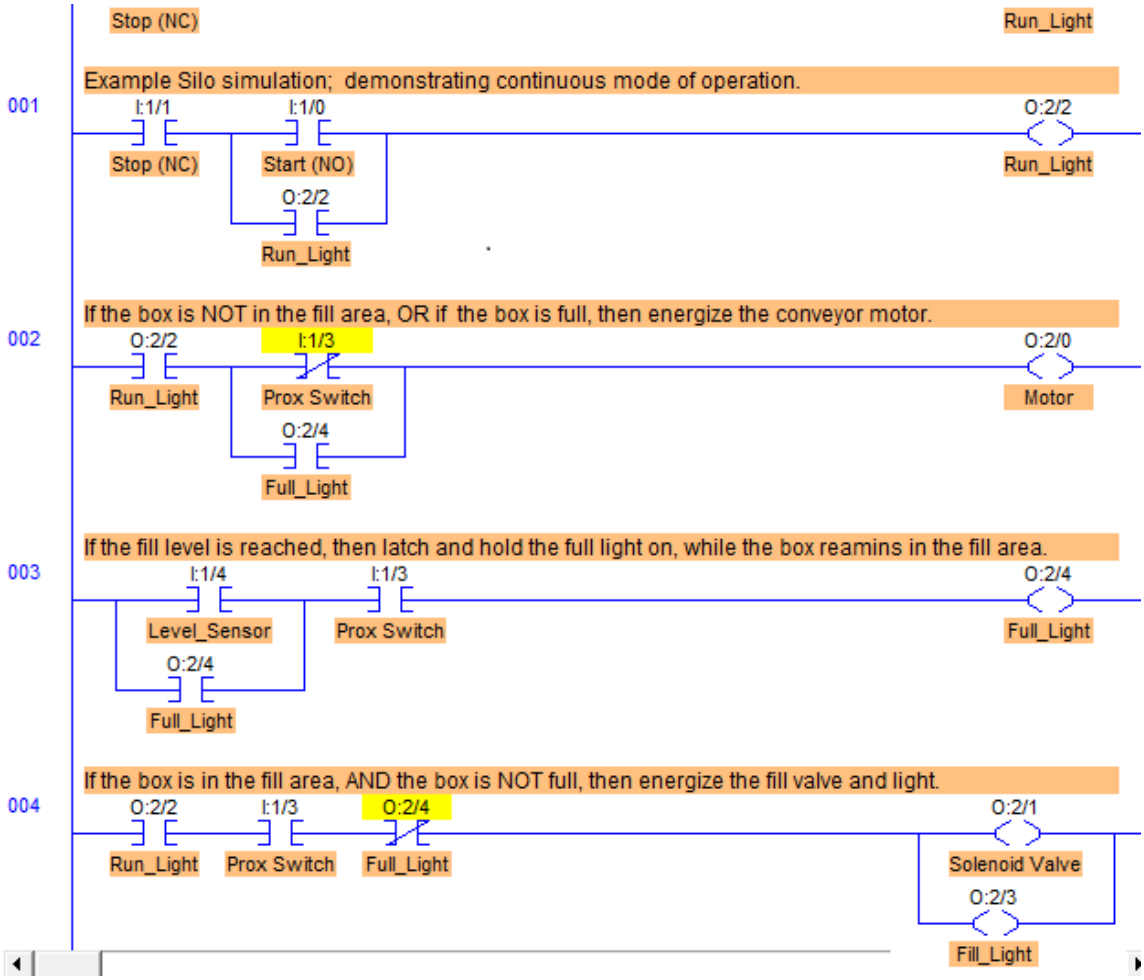

Figure 42: Ladder Logic for Delta Baseline (3)

Delta Baseline Program for Instance 4

Figure 43 illustrates the delta baseline program ladder logic for instance 4. The equivalent Petri net is similar to the baseline program shown in Figure 19.

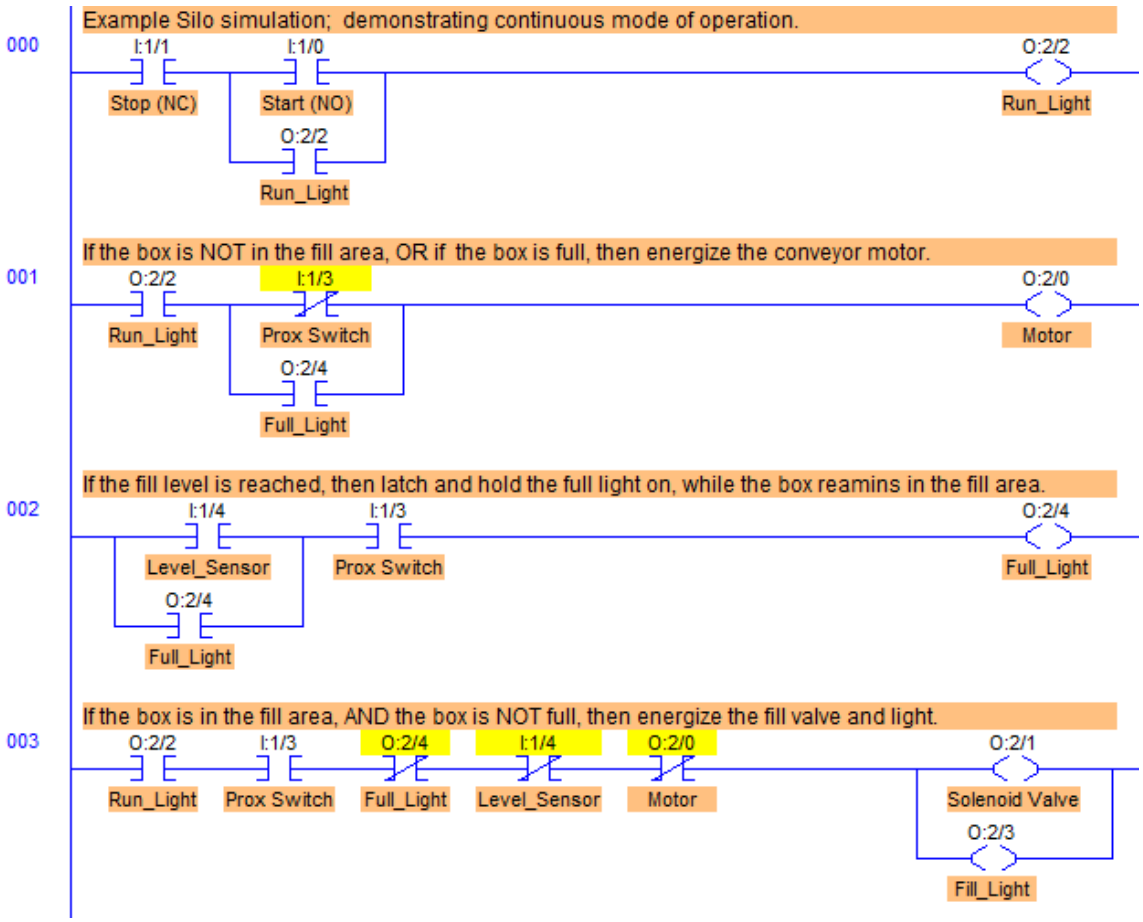

Figure 43: Ladder Logic for Delta Baseline (4)

Delta Baseline Program for Instance 5

Figure 44 illustrates the delta baseline program ladder logic for instance 5. The equivalent Petri net is similar to the baseline program shown in Figure 19.

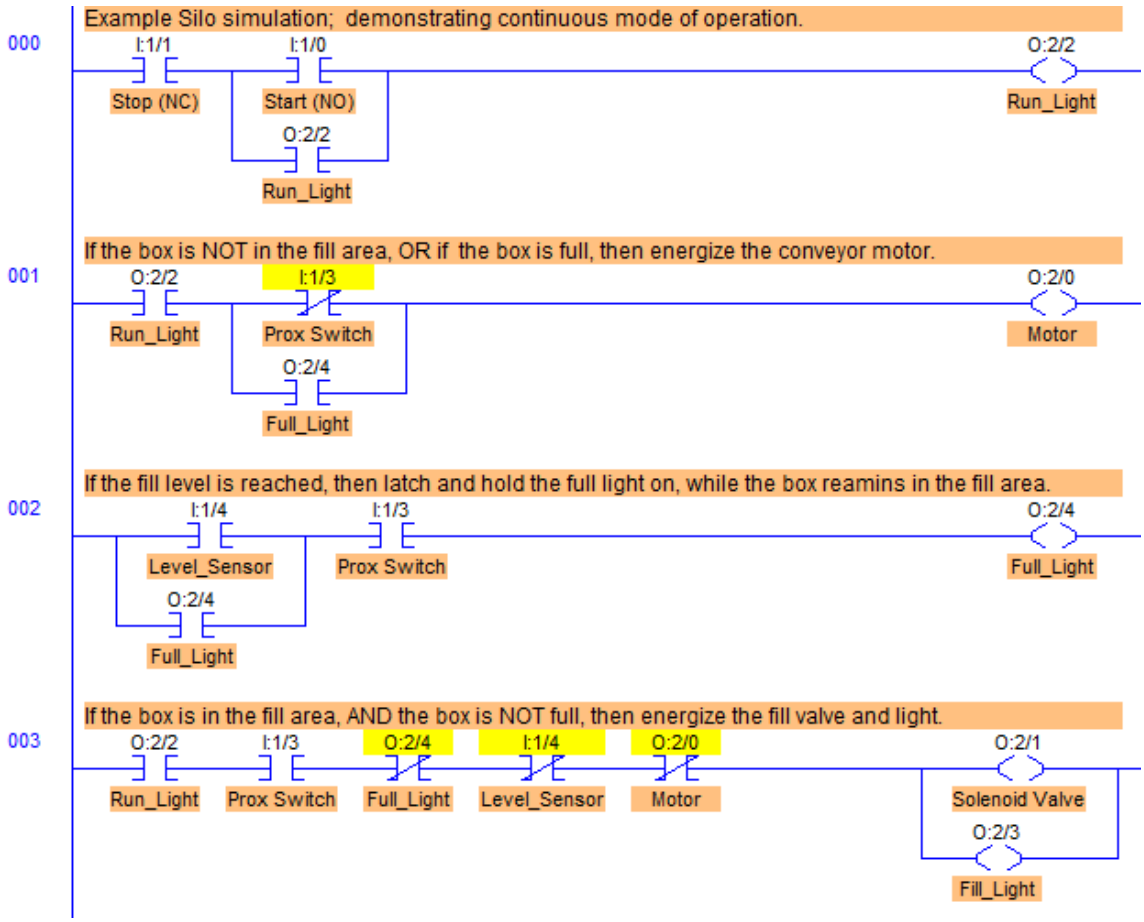

Figure 44: Ladder Logic for Delta Baseline (5)

Delta Baseline Program for Instance 6

Figure 45 illustrates the delta baseline program ladder logic for instance 6. The equivalent Petri net is similar to the baseline program shown in Figure 19.

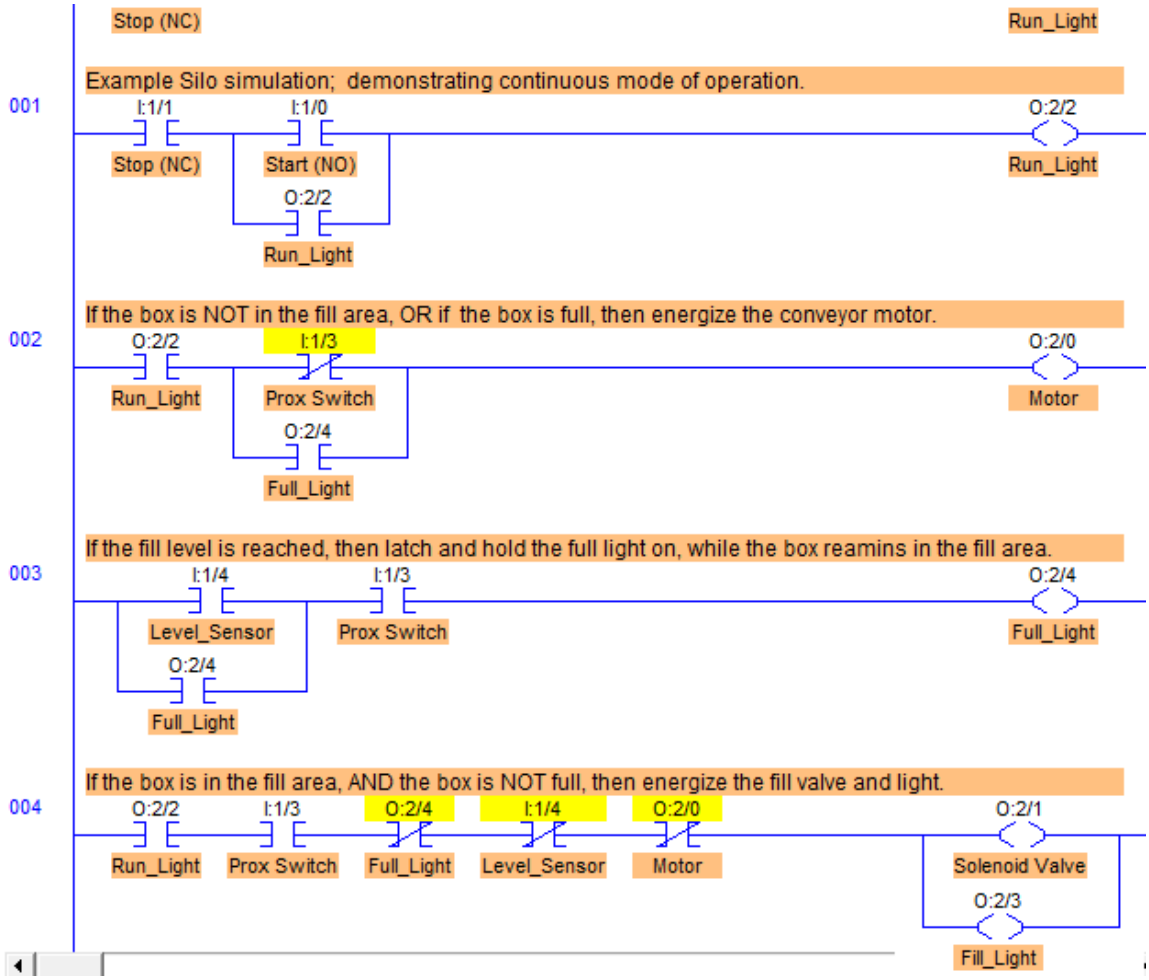

Figure 45: Ladder Logic for Delta Baseline (6)

Delta Baseline Program for Instance 7

Figure 46 illustrates the delta baseline program ladder logic for instance 7. The equivalent Petri net is similar to the baseline program shown in Figure 19.

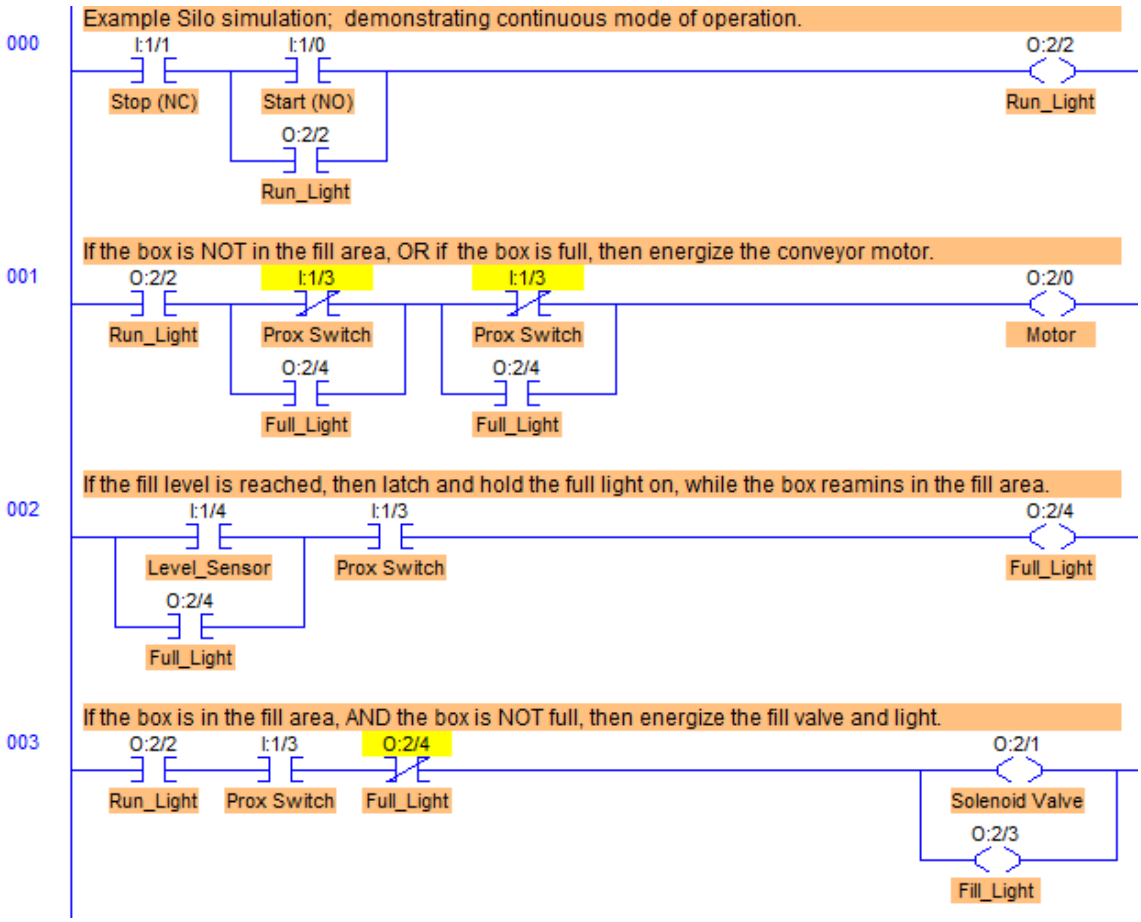

Figure 46: Ladder Logic for Delta Baseline (7)

Delta Baseline Program for Instance 8

Figure 47 illustrates the delta baseline program ladder logic for instance 8. The equivalent Petri net is similar to the baseline program shown in Figure 19.

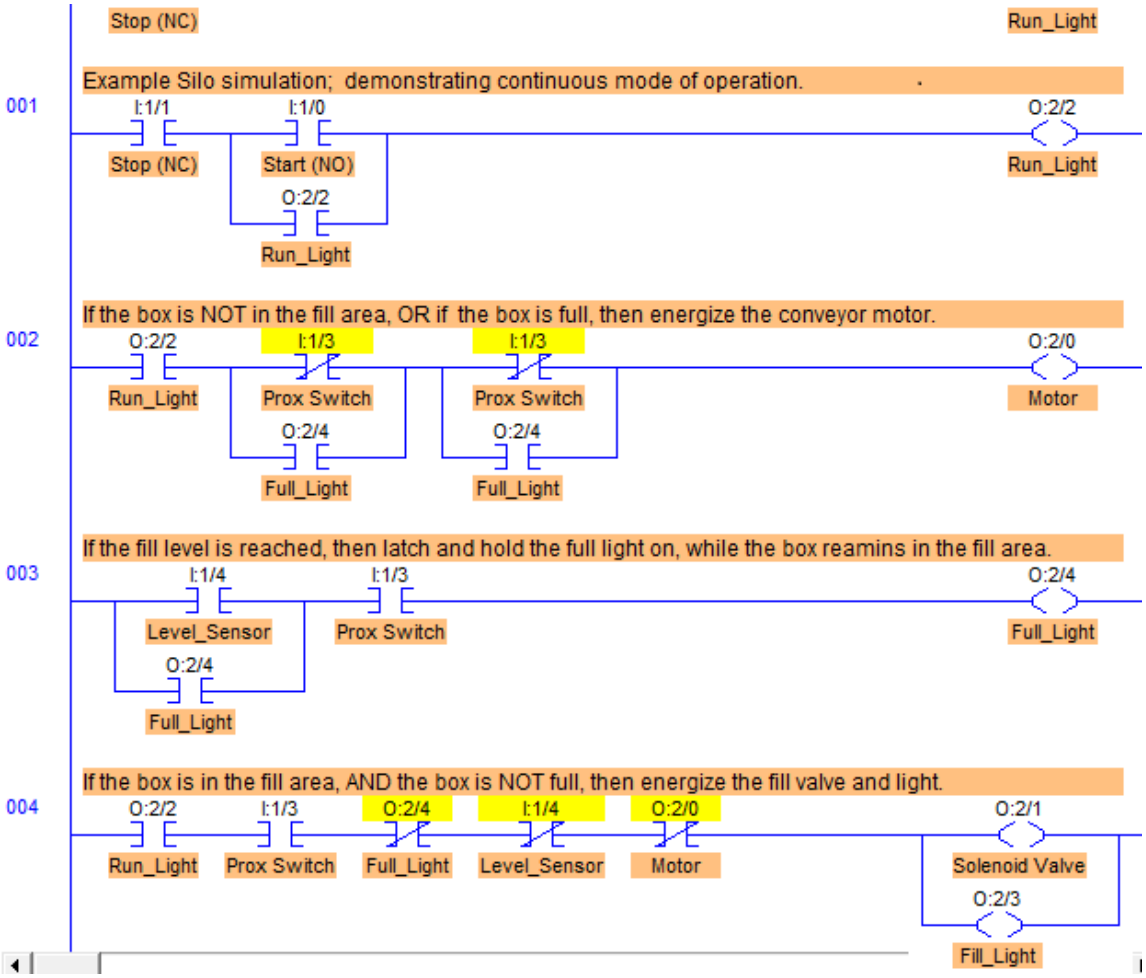

Figure 47: Ladder Logic for Delta Baseline (8)

Delta Baseline Program for Instance 9

Figure 48 illustrates the delta baseline program ladder logic for instance 9. The equivalent Petri net is similar to the baseline program shown in Figure 19.

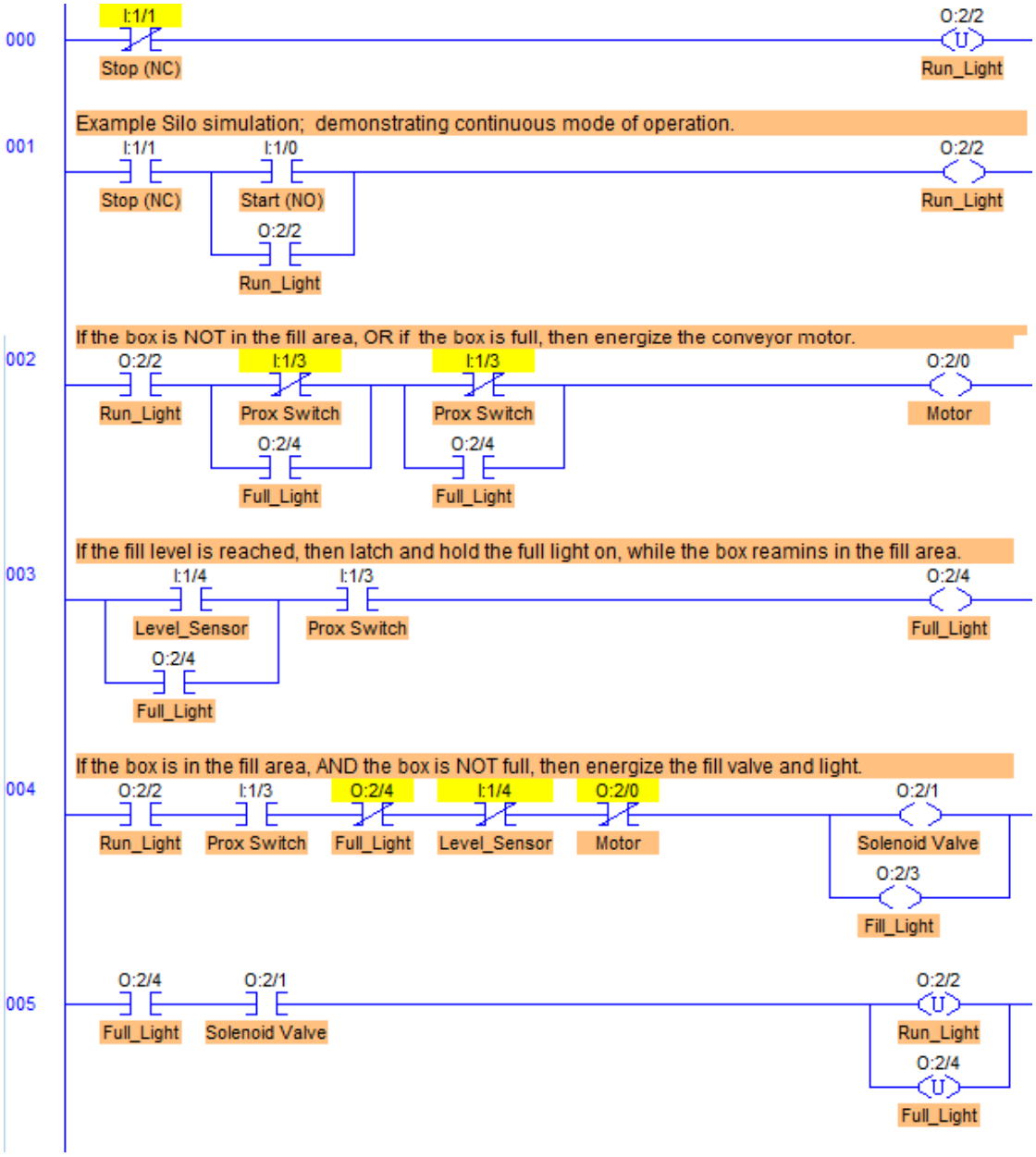

Figure 48: Ladder Logic for Delta Baseline (9)

Delta Baseline Program for Instance 10

Figure 49 illustrates the delta baseline program ladder logic for instance 10. The equivalent Petri net is similar to the baseline program shown in Figure 19.

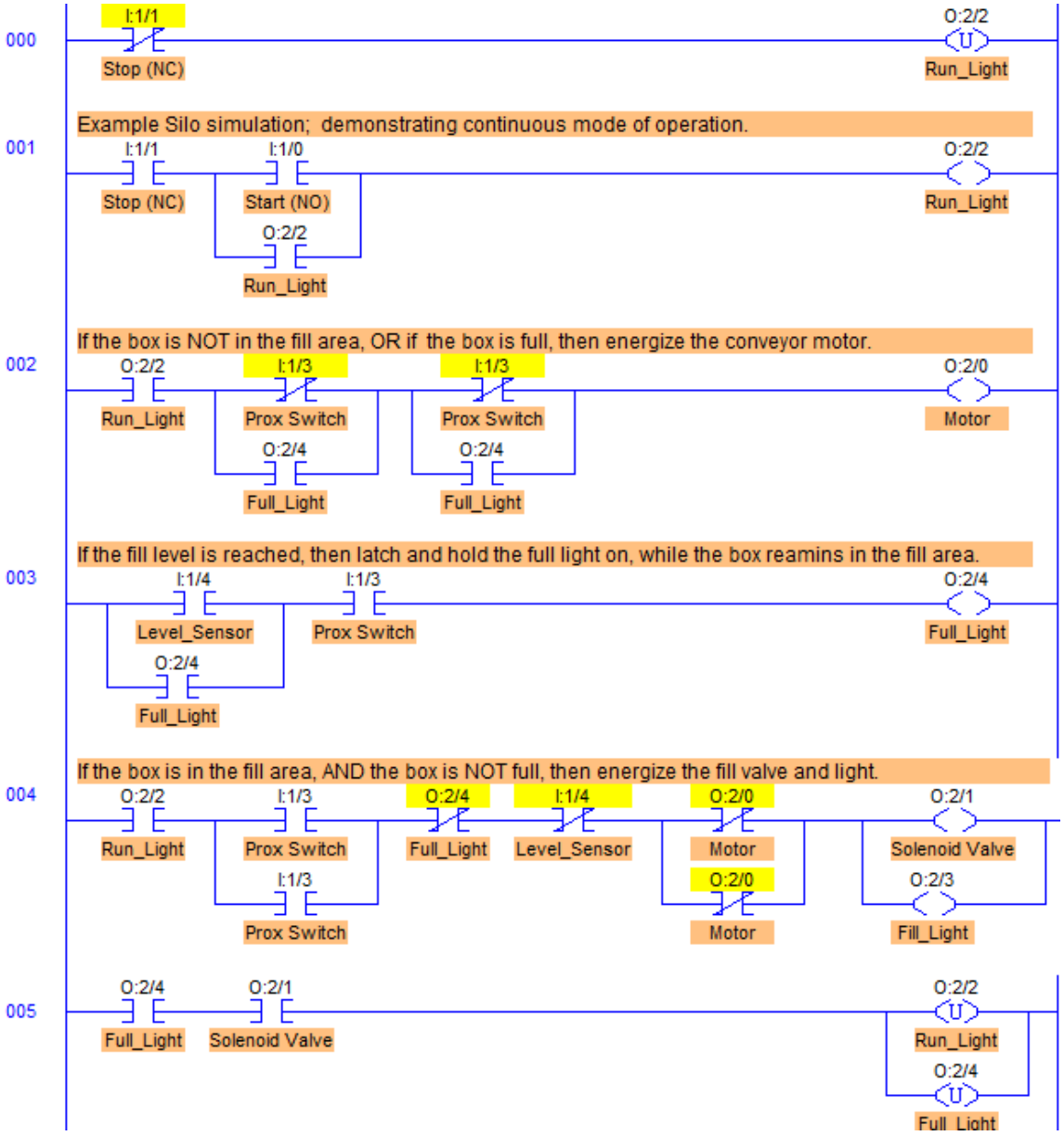

Figure 49: Ladder Logic for Delta Baseline (10)

Attack Delta Baseline Program for Instance 1

Figure 50 illustrates the attack delta baseline program ladder logic for instance 1. The equivalent Petri net is similar to the baseline program shown in Figure 19.

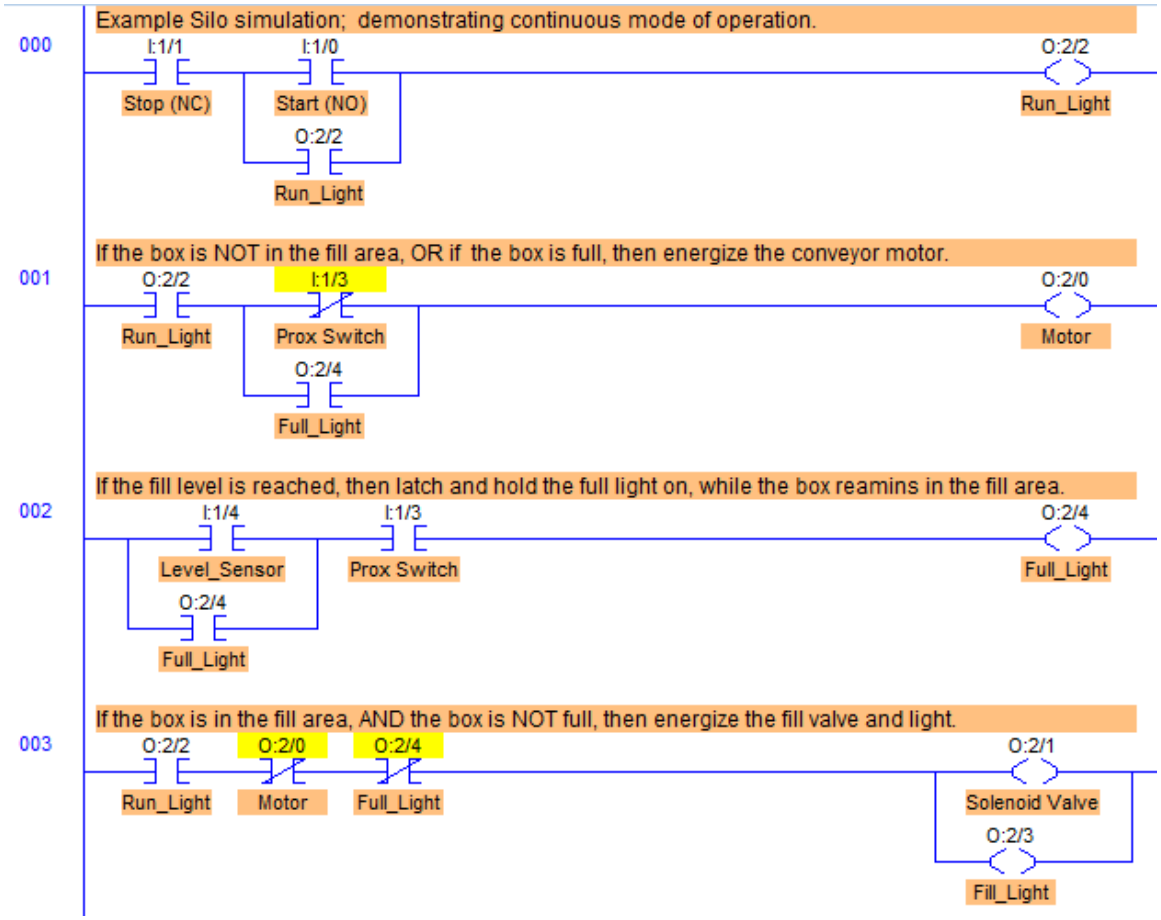

Figure 50: Attack Ladder Logic for Delta Baseline (1)

Attack Delta Baseline Program for Instance 2

Figure 51 illustrates the attack delta baseline program ladder logic for instance 2. The equivalent Petri net is similar to the baseline program shown in Figure 19.

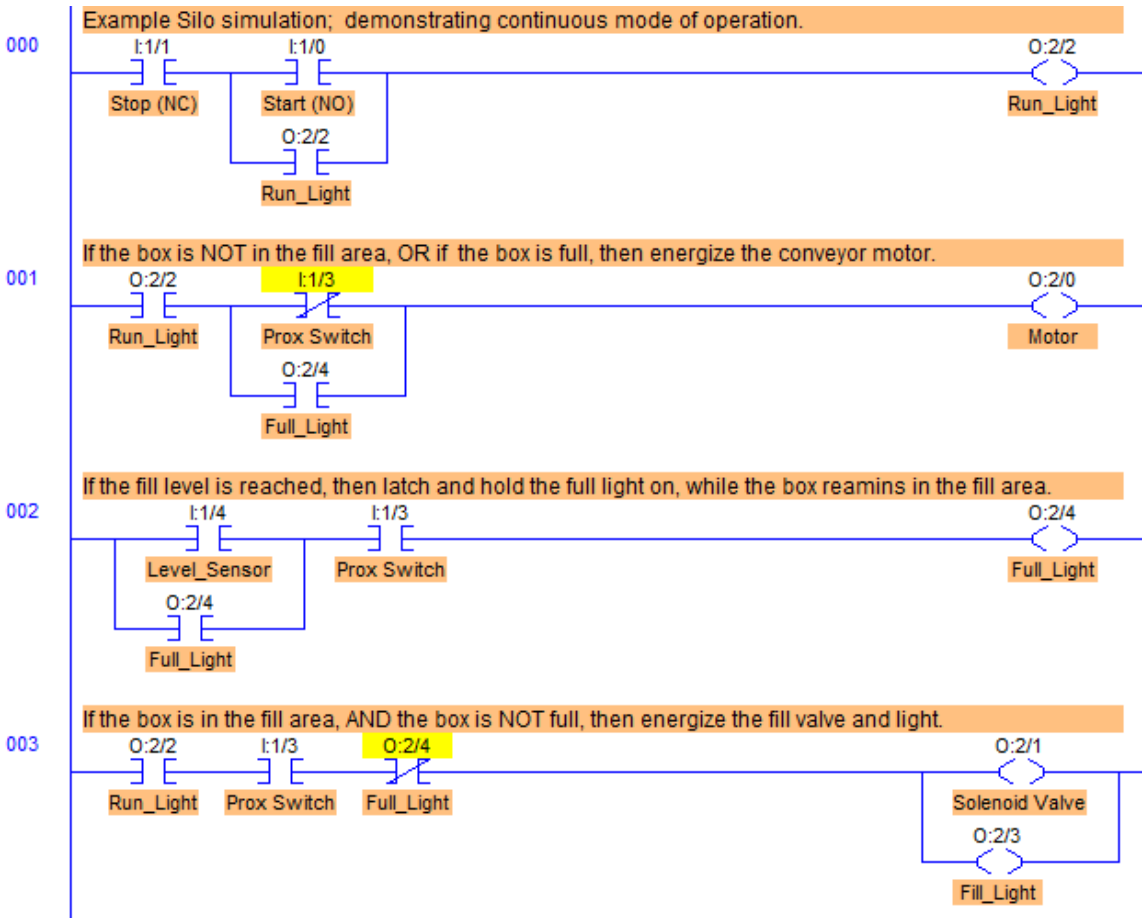

Figure 51: Attack Ladder Logic for Delta Baseline (2)

Attack Delta Baseline Program for Instance 3

Figure 52 illustrates the attack delta baseline program ladder logic for instance 3. The equivalent Petri net is similar to the baseline program shown in Figure 19.

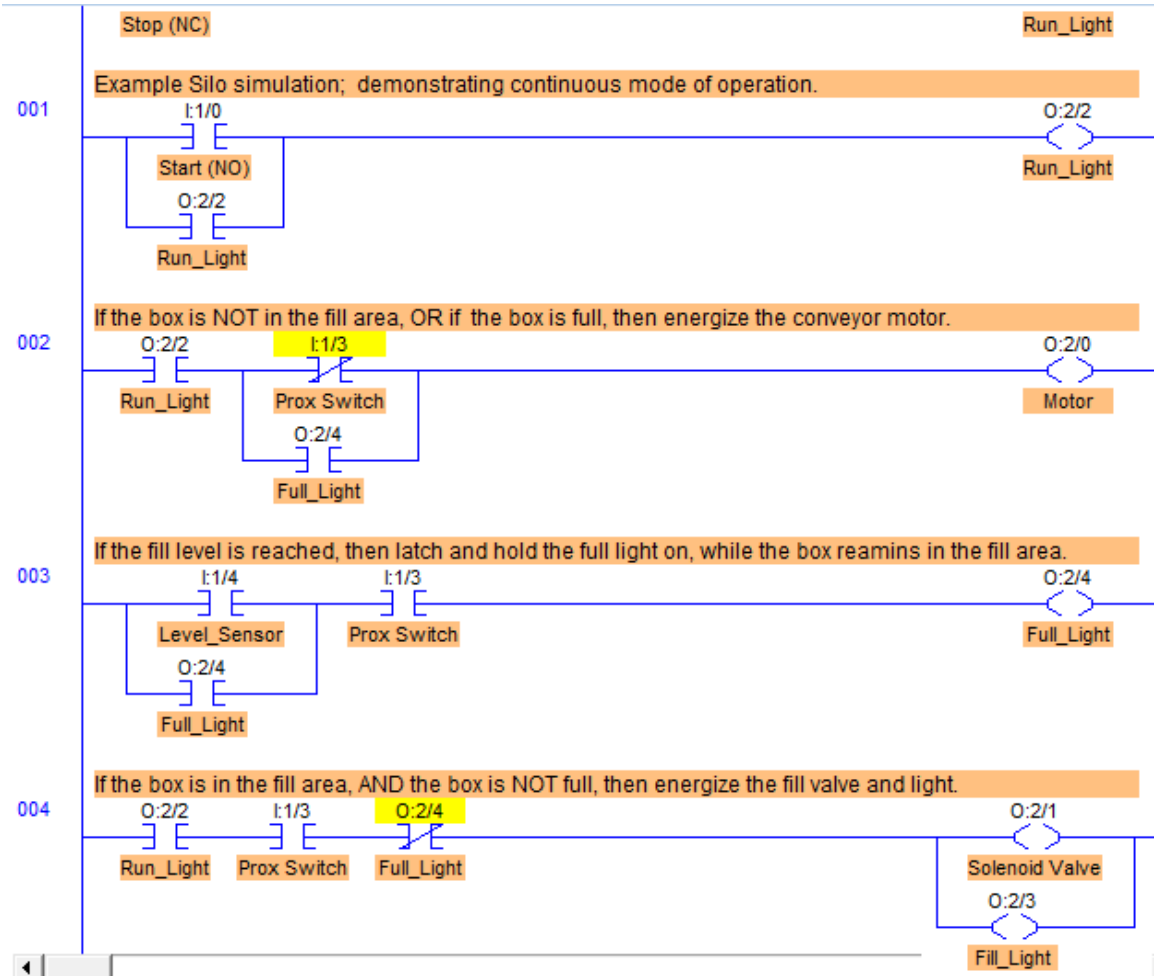

Figure 52: Attack Ladder Logic for Delta Baseline (3)

Attack Delta Baseline Program for Instance 4

Figure 53 illustrates the attack delta baseline program ladder logic for instance 4. The equivalent Petri net is similar to the baseline program shown in Figure 19.

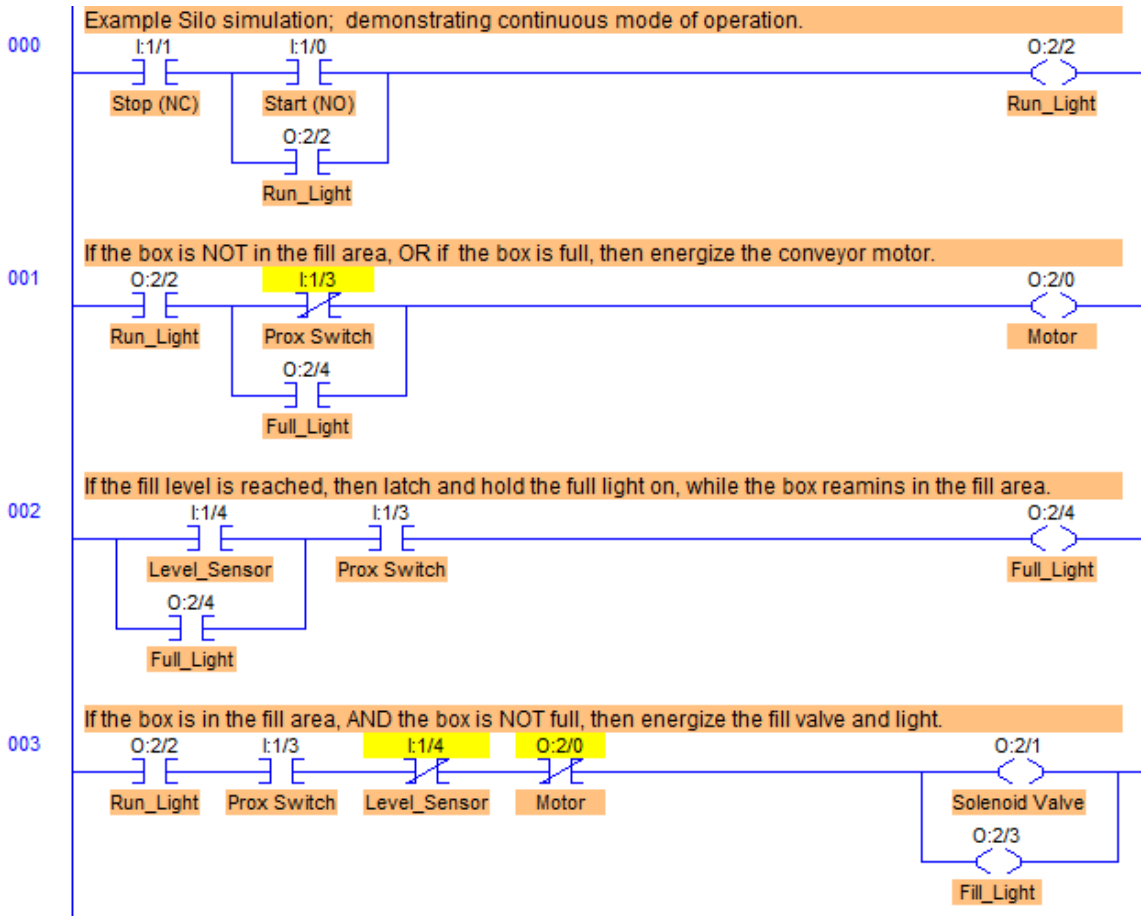

Figure 53: Attack Ladder Logic for Delta Baseline (4)

Attack Delta Baseline Program for Instance 5

Figure 54 illustrates the attack delta baseline program ladder logic for instance 5. The equivalent Petri net is similar to the baseline program shown in Figure 19.

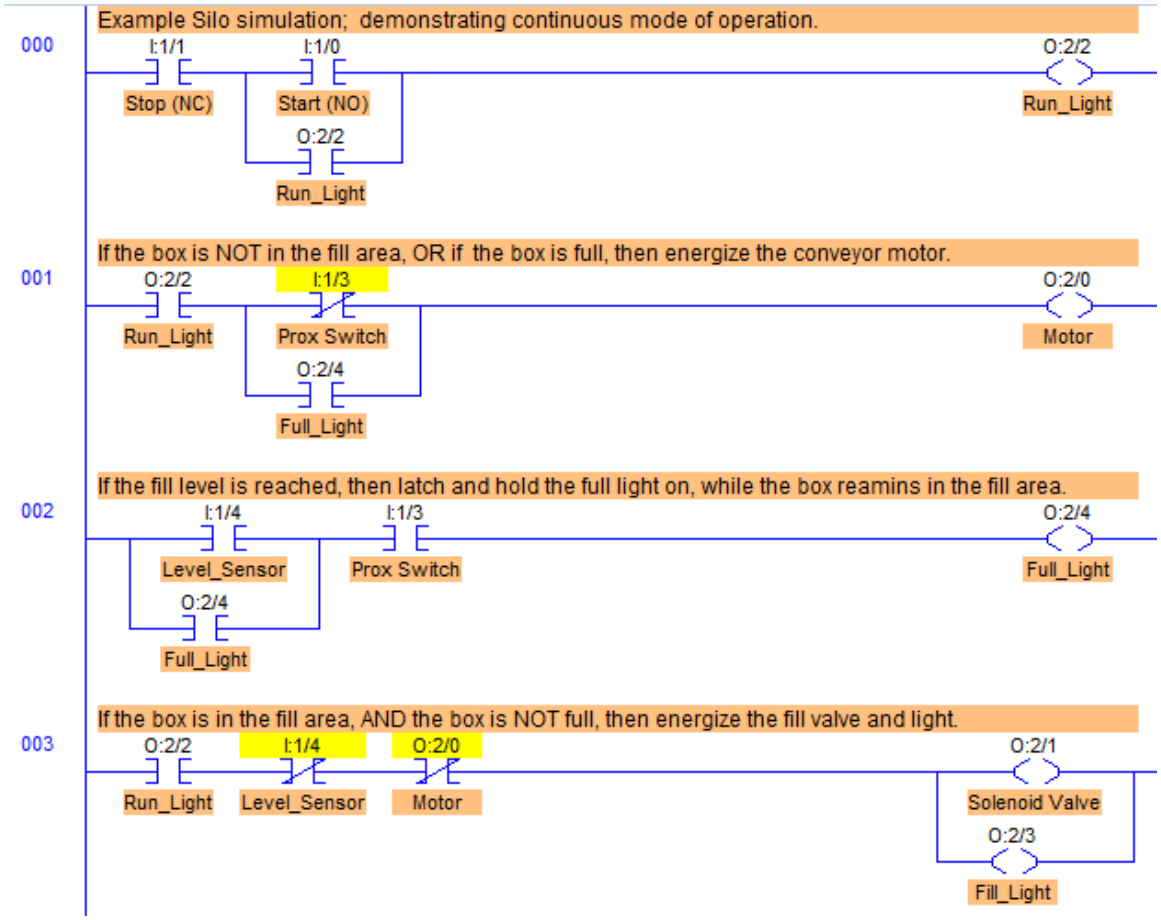

Figure 54: Attack Ladder Logic for Delta Baseline (5)

Attack Delta Baseline Program for Instance 6

Figure 55 illustrates the attack delta baseline program ladder logic for instance 6. The equivalent Petri net is similar to the baseline program shown in Figure 19.

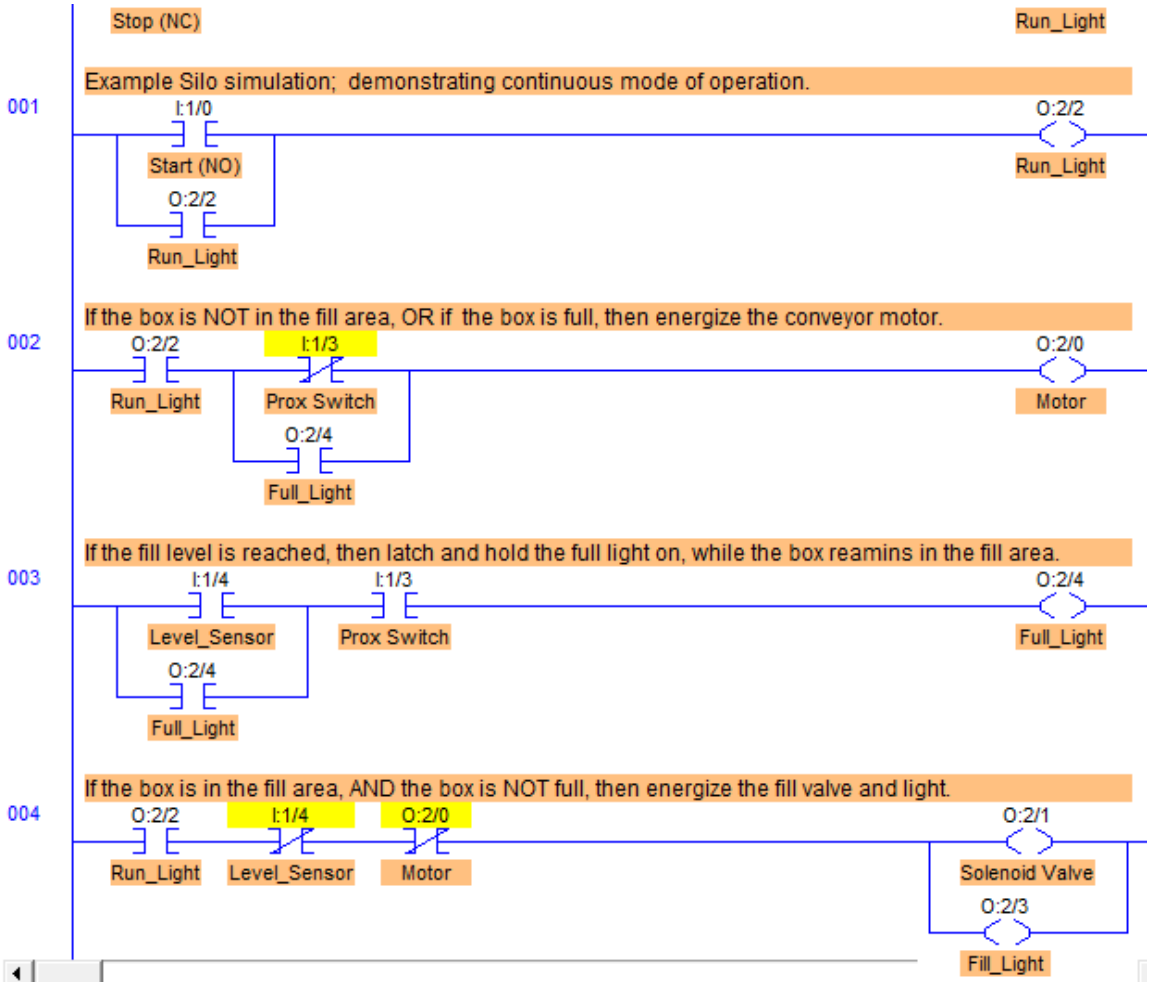

Figure 55: Attack Ladder Logic for Delta Baseline (6)

Attack Delta Baseline Program for Instance 7

Figure 56 illustrates the attack delta baseline program ladder logic for instance 7. The equivalent Petri net is similar to the baseline program shown in Figure 19.

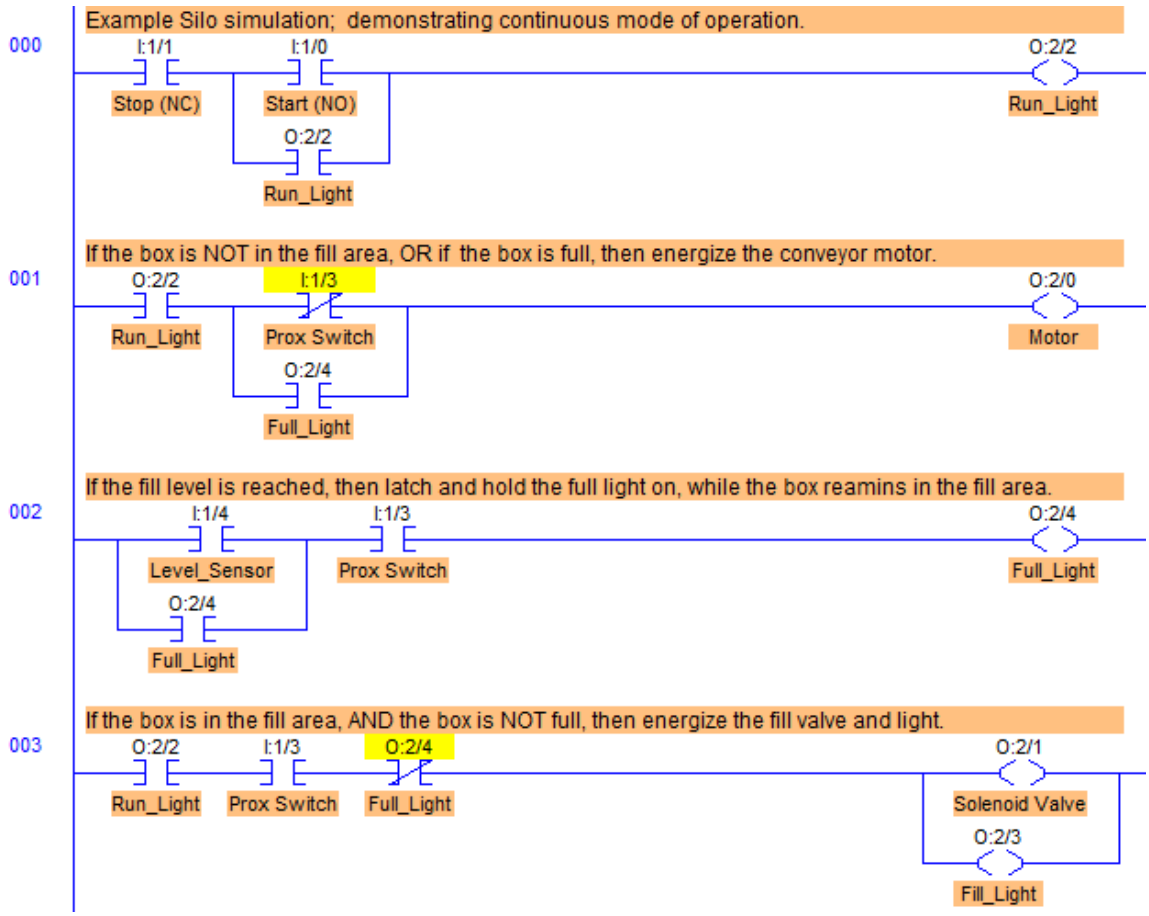

Figure 56: Attack Ladder Logic for Delta Baseline (7)

Attack Delta Baseline Program for Instance 8

Figure 57 illustrates the attack delta baseline program ladder logic for instance 8. The equivalent Petri net is similar to the baseline program shown in Figure 19.

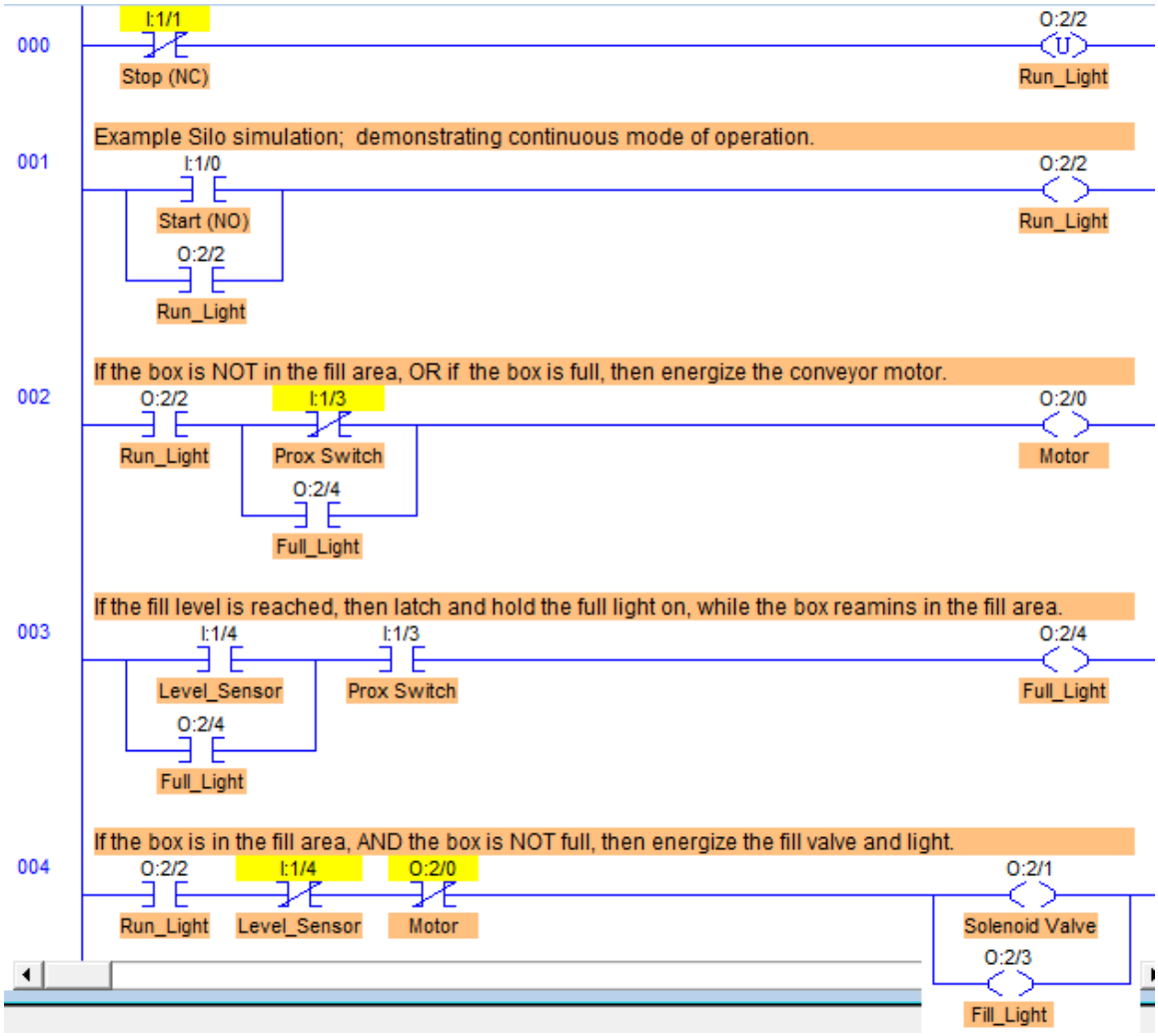

Figure 57: Attack Ladder Logic for Delta Baseline (8)

Attack Delta Baseline Program for Instance 9

Figure 58 illustrates the attack delta baseline program ladder logic and Figure 59 shows the Petri net for instance 9.

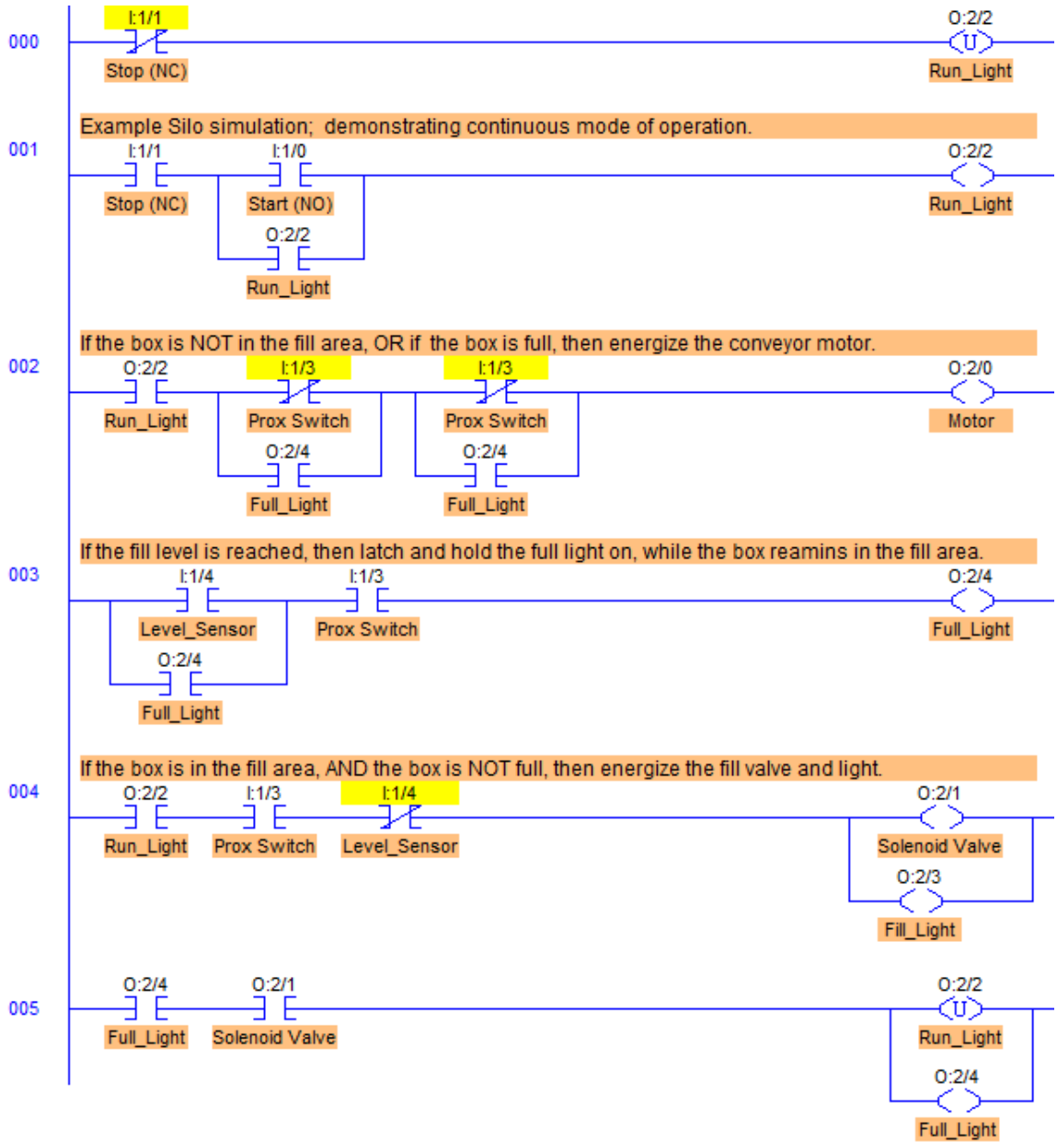

Figure 58: Attack Ladder Logic for Delta Baseline (9)

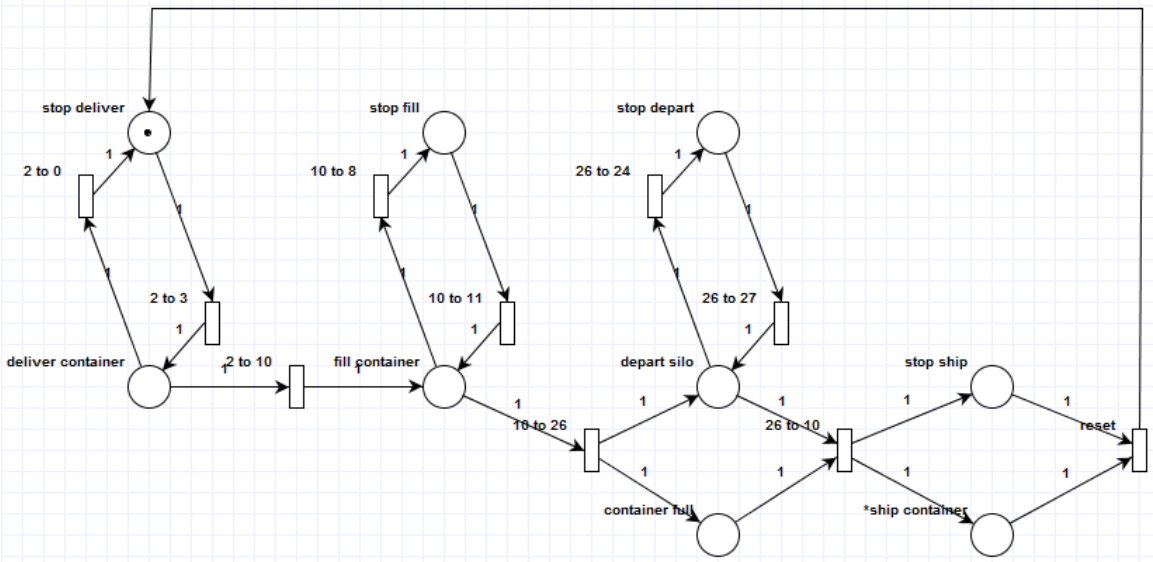

Figure 59: Petri Net for Attack Delta Baseline (9)

The formal definition for the Petri net illustrated in Figure 77 is *C = {P, T, I, O}*, such that:

- *P* = {deliver container, stop deliver, fill container, stop fill, container full, depart silo, stop depart, ship container, stop ship}
- *T* = {2 to 0, 2 to 3, 2 to 10, 10 to 2, 10 to 8, 10 to 11, 10 to 26, 26 to 10, 26 to 24, 26 to 27}
- *I*(2 to 0) = {deliver container (5)}
- *I*(2 to 3) = {stop deliver (0)}
- *I*(2 to 10) = {deliver container (5)}
- *I*(10 to 8) = {fill container (14)}
- *I*(10 to 11) = {stop fill (0)}
- *I*(10 to 26) = {fill container (14)}
- *I*(26 to 10) = {depart silo (21), container full (20)}

- *I*(26 to 24) = {depart silo (21)}
- *I*(26 to 27) = {stop depart (0)}
- *I*(reset) = {stop ship (0), *ship container (11)}
- *O*(2 to 0) = {stop deliver (0)}
- *O*(2 to 3) = {deliver container (5)}
- *O*(2 to 10) = {fill container (14)}
- *O*(10 to 8) = {stop fill (0)}
- *O*(10 to 11) = {fill container (14)}
- *O*(10 to 26) = {depart silo (21), container full (20)}
- *O*(26 to 10) = {stop ship (0), *ship container (11)}
- *O*(26 to 24) = {stop depart (0)}
- *O*(26 to 27) = {depart silo (21)}
- *O*(reset) = {stop deliver (0)}

Attack Delta Baseline Program for Instance 10

Figure 60 illustrates the attack delta baseline program ladder logic for instance 10. The equivalent Petri net is similar to the baseline program shown in Figure 19.

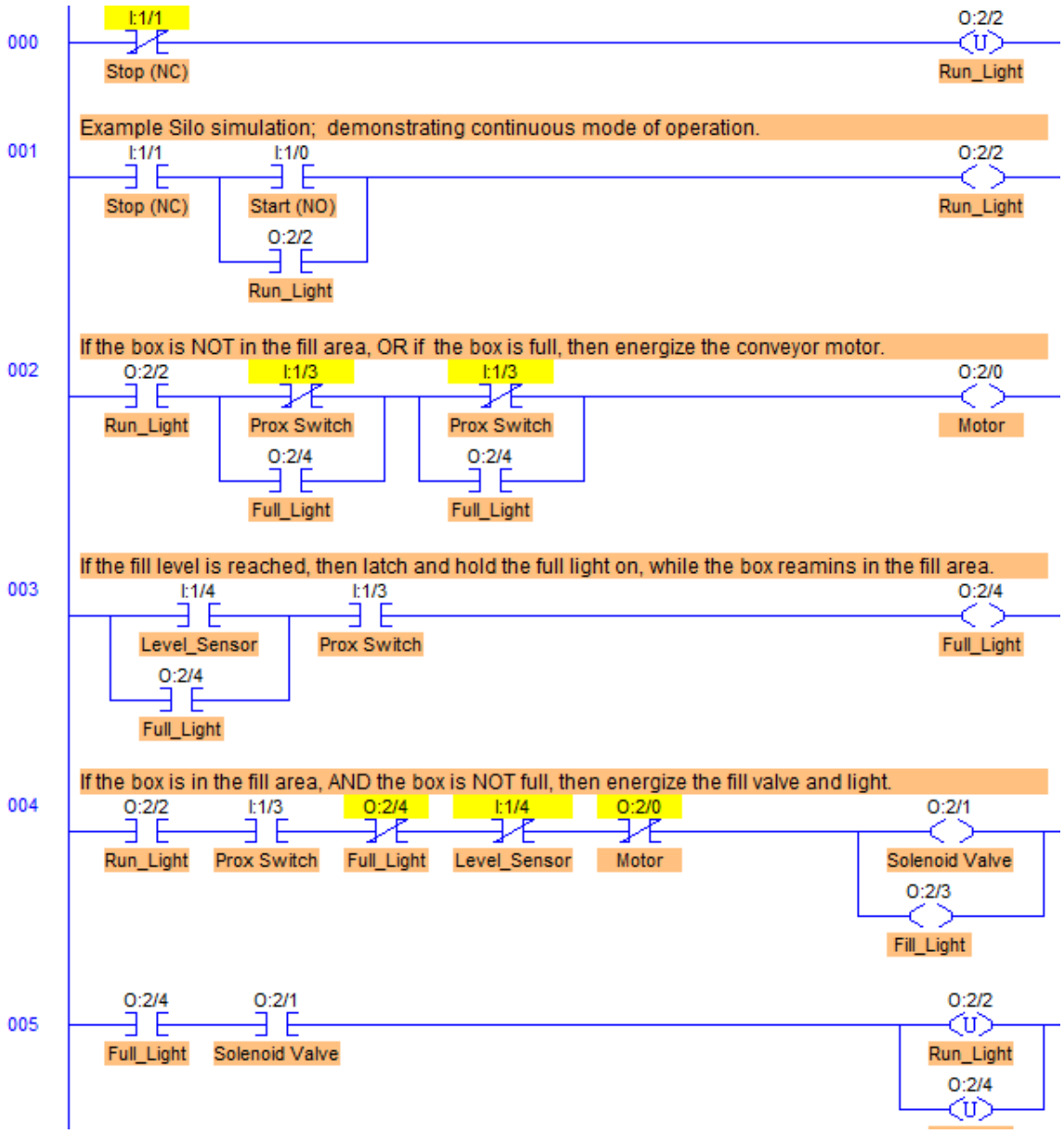

Figure 60: Attack Ladder Logic for Delta Baseline (10)

Appendix B

Baseline Program

The following state table, reachability graph and matrix are representative of:

- The *baseline* program for PLC instances 1 through 10
- The *delta baseline* program for PLC instances 1 through 10
- The *attack delta baseline* program for PLC instances 1 through 8, and 10

Table 14: Tangible States for Baseline (all), Delta Baseline (all) and Attack Delta Baseline (1-8, 10)

	container full	deliver container	depart silo	fill container	ship container	stop deliver	stop depart	stop fill	stop ship
M0	0	0	0	0	0	1	0	0	0
M1	0	1	0	0	0	0	0	0	0
M2	0	0	0	1	0	0	0	0	0
M3	0	0	0	0	0	0	0	1	0
M4	1	0	1	0	0	0	0	0	0
M5	1	0	0	0	0	0	1	0	0
M6	0	0	0	0	1	0	0	0	0
M7	0	0	0	0	0	0	0	0	1

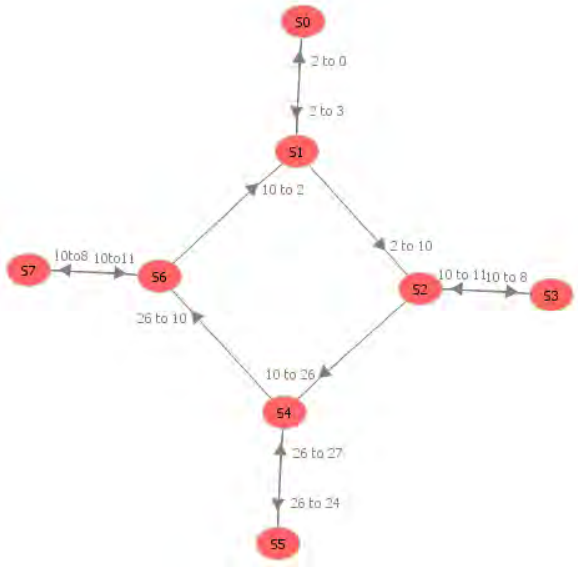

Figure 61: Graph for Baseline (1-10), Delta Baseline (1-10) and Attack Delta Baseline (1-8,10)

Table 15: Matrix for Baseline (1-10), Delta Baseline (1-10) and Attack Delta Baseline (1-8,10)

	PN marking	m0	m1	m2	m3	m4	m5	m6	m7
Output Places	stop deliver	1							
	deliver container		1						
	stop fill				1				
	fill container			1					
	stop depart						1		
	depart silo					1			
	container full					1	1		
	stop ship								1
	ship container							1	
Input Transitions	2 --> 0	1							
	2 --> 3		1						
	2 --> 10			1					
	10 --> 2		1						
	10 --> 8				1				1
	10 --> 11			1				1	
	10 --> 26					1			
	26 --> 10							1	
	26 --> 24						1		
	26 --> 27					1			
	*input								

Attack Delta Baseline Program (for PLC Instance 9)

The following state table, reachability graph and matrix are representative of *attack delta baseline* program for PLC instance 9.

Table 16: Tangible States for Attack Delta Baseline (9)

	container full	deliver container	depart silo	fill container	Ship Container	stop deliver	stop depart	stop fill	Stop Ship
M0	0	0	0	0	0	1	0	0	0
M1	0	1	0	0	0	0	0	0	0
M2	0	0	0	1	0	0	0	0	0
M3	0	0	0	0	0	0	0	1	0
M4	1	0	1	0	0	0	0	0	0
M5	1	0	0	0	0	0	1	0	0
M6	0	0	0	0	1	0	0	0	1

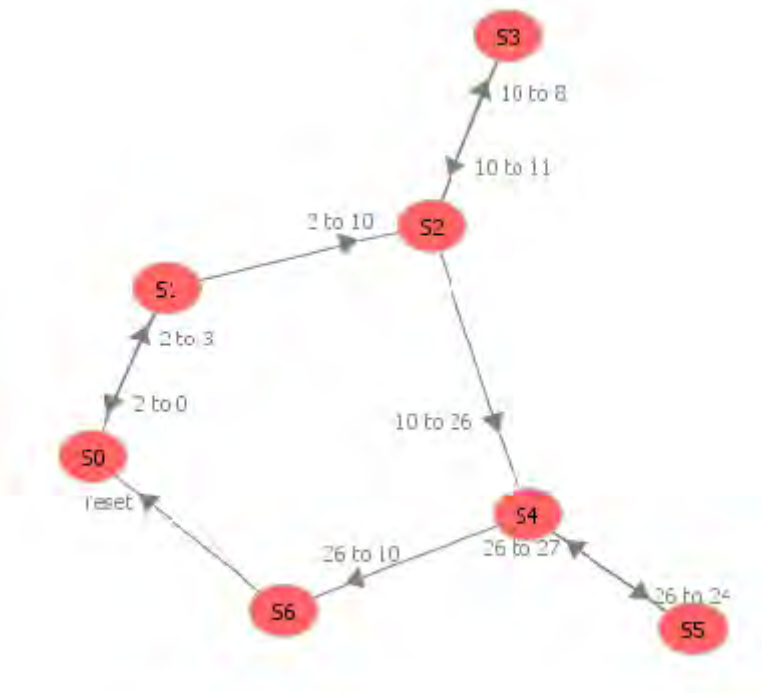

Figure 62: Graph for Attack Delta Baseline (9)

Table 17: Matrix for Attack Delta Baseline (9)

	PN marking	m0	m1	m2	m3	m4	m5	m6
Output Places	stop deliver	1						
	deliver container		1					
	stop fill				1			
	fill container			1				
	stop depart						1	
	depart silo					1		
	container full					1	1	
	stop ship							1
	*ship container							1
Input Transitions	2 --> 0	1						
	2 --> 3		1					
	2 --> 10			1				
	10 --> 2							
	10 --> 8				1			
	10 --> 11			1				
	10 --> 26					1		
	26 --> 10							1
	26 --> 24						1	
	26 --> 27					1		
	*input	1						

Attack Baseline Program (for PLC Instance 1)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 1.

Table 18: Tangible States for Attack Baseline (1)

	*deliver container	container full	depart silo	fill container	ship container	stop deliver	stop depart	stop fill	stop ship
M0	0	0	0	0	0	1	0	0	0
M1	1	0	0	0	0	0	0	0	0
M2	0	0	0	1	0	0	0	0	0
M3	0	0	0	0	0	0	0	1	0
M4	0	1	1	0	0	0	0	0	0
M5	0	1	0	0	0	0	1	0	0
M6	0	0	0	0	1	0	0	0	0
M7	0	0	0	0	0	0	0	0	1

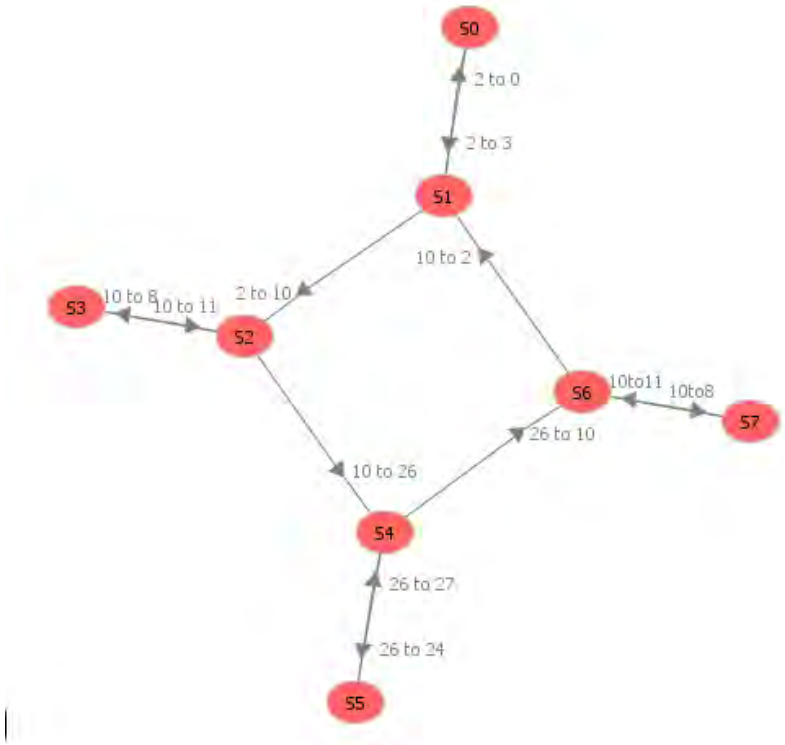

Figure 63: Graph for Attack Baseline (1)

Table 19: Matrix for Attack Baseline (1)

	PN marking	m0	m1	m2	m3	m4	m5	m6	m7
Output Places	stop deliver	1							
	*deliver container		1						
	stop fill				1				
	fill container			1					
	stop depart						1		
	depart silo					1			
	container full					1	1		
	stop ship								1
	ship container							1	
Input Transitions	2 --> 0	1							
	2 --> 3		1						
	2 --> 10			1					
	10 --> 2		1						
	10 --> 8				1				1
	10 --> 11			1				1	
	10 --> 26					1			
	26 --> 10							1	
	26 --> 24						1		
	26 --> 27					1			
	*input								

Attack Baseline Program (for PLC Instance 2)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 2.

Table 20: Tangible States for Attack Baseline (2)

	*depart silo	*container full	deliver container	fill container	stop depart	stop deliver	stop fill
M0	0	0	0	0	0	1	0
M1	0	0	1	0	0	0	0
M2	0	0	0	1	0	0	0
M3	0	0	0	0	0	0	1
M4	1	1	0	0	0	0	0
M5	0	1	0	0	1	0	0

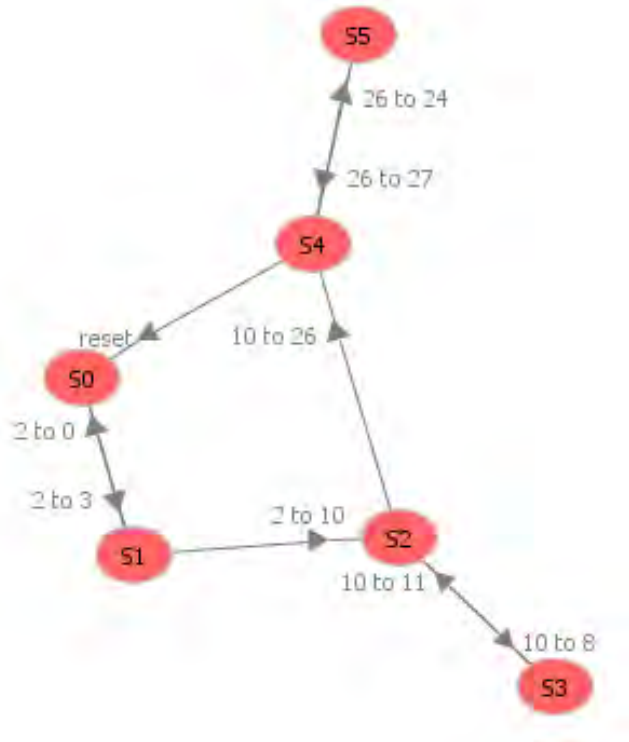

Figure 64: Graph for Attack Baseline (2)

Table 21: Matrix for Attack Baseline (2)

	PN marking	m0	m1	m2	m3	m4	m5
Output Places	stop deliver	1					
	deliver container		1				
	stop fill				1		
	fill container			1			
	stop depart						1
	*depart silo					1	
	*container full					1	1
Input Transitions	2 --> 0	1					
	2 --> 3		1				
	2 --> 10			1			
	10 --> 2						
	10 --> 8				1		
	10 --> 11			1			
	10 --> 26					1	
	26 --> 10						
	26 --> 24						1
	26 --> 27					1	
	*input	1					

Attack Baseline Program (for PLC Instance 3)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 3.

Table 22: Tangible States for Attack Baseline (3)

	*stop deliver	fill container	container full	depart silo	ship container	deliver container
M0	1	0	0	0	0	0
M1	0	0	0	0	0	1
M2	0	1	0	0	0	0
M3	0	0	1	1	0	0
M4	0	0	0	0	1	0

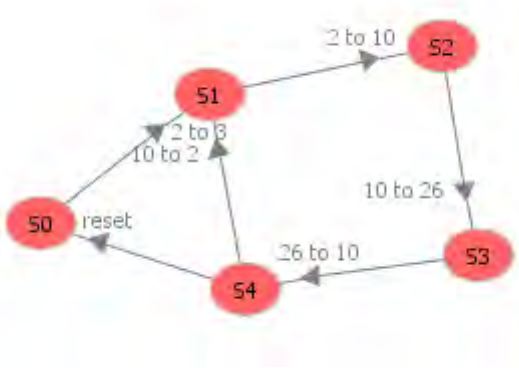

Figure 65: Graph for Attack Baseline (3)

Table 23: Matrix for Attack Baseline (3)

	PN marking	m0	m1	m2	m3	m4
Output Places	*stop deliver	1				
	deliver container		1			
	fill container			1		
	depart silo				1	
	container full				1	
	ship container					1
Input Transitions	2 --> 0					
	2 --> 3		1			
	2 --> 10			1		
	10 --> 2		1			
	10 --> 8					
	10 --> 11			1		1
	10 --> 26				1	
	26 --> 10					1
	26 --> 24					
	26 --> 27				1	
	*input	1				

Attack Baseline Program (for PLC Instance 4)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 4.

Table 24: Tangible States for Attack Baseline (4)

	*container full	*depart silo	*ship container	deliver container	fill container	stop deliver	stop depart	stop fill	stop ship
M0	0	0	0	0	0	1	0	0	0
M1	0	0	0	1	0	0	0	0	0
M2	0	0	0	0	1	0	0	0	0
M3	0	0	0	0	0	0	0	1	0
M4	1	1	0	0	0	0	0	0	0
M5	1	0	0	0	0	0	1	0	0
M6	0	0	1	0	0	0	0	0	0
M7	0	0	0	0	0	0	0	0	1

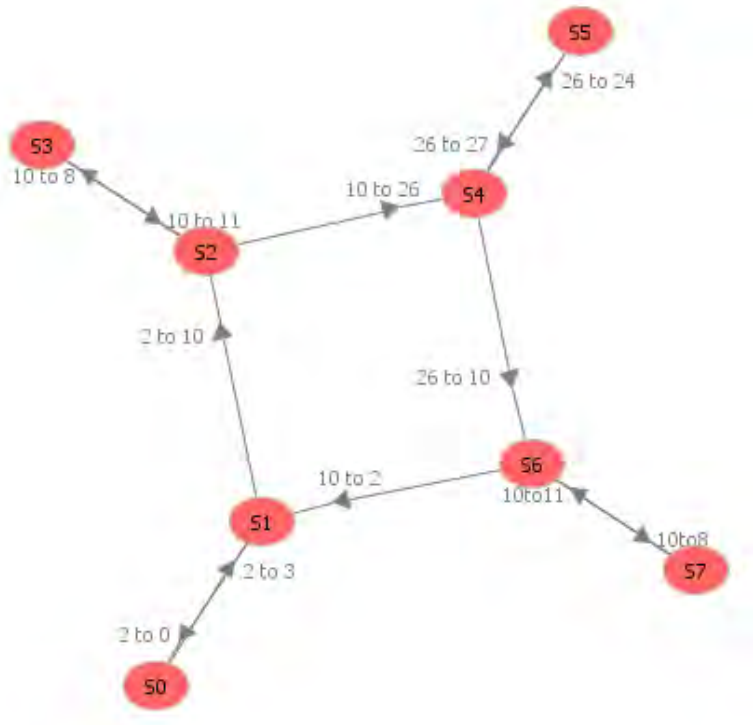

Figure 66: Graph for Attack Baseline (4)

Table 25: Matrix for Attack Baseline (4)

	PN marking	m0	m1	m2	m3	m4	m5	m6	m7
Output Places	stop deliver	1							
	deliver container		1						
	stop fill				1				
	fill container			1					
	stop depart						1		
	*depart silo					1			
	*container full					1	1		
	stop ship								1
	*ship container							1	
Input Transitions	2 --> 0	1							
	2 --> 3		1						
	2 --> 10			1					
	10 --> 2		1						
	10 --> 8				1				1
	10 --> 11			1				1	
	10 --> 26					1			
	26 --> 10							1	
	26 --> 24						1		
	26 --> 27					1			
	*input								

Attack Baseline Program (for PLC Instance 5)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 5.

Table 26: Tangible States for Attack Baseline (5)

	*deliver container	*container full	*depart silo	*ship container	fill container	stop deliver	stop depart	stop fill	stop ship
M0	0	0	0	0	0	1	0	0	0
M1	1	0	0	0	0	0	0	0	0
M2	0	0	0	0	1	0	0	0	0
M3	0	0	0	0	0	0	0	1	0
M4	0	1	1	0	0	0	0	0	0
M5	0	1	0	0	0	0	1	0	0
M6	0	0	0	1	0	0	0	0	0
M7	0	0	0	0	0	0	0	0	1

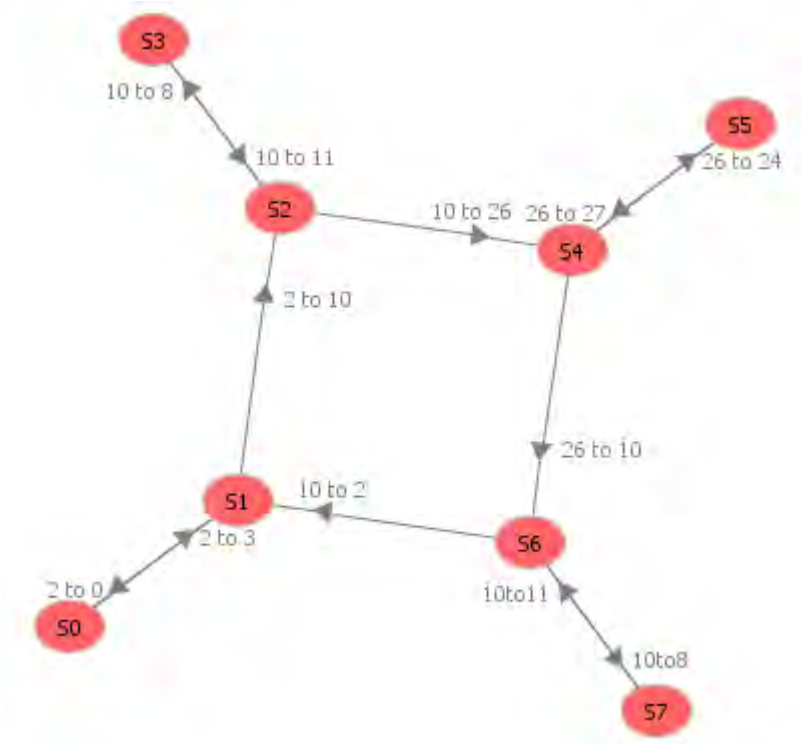

Figure 67: Graph for Attack Baseline (5)

Table 27: Matrix for Attack Baseline (5)

	PN marking	m0	m1	m2	m3	m4	m5	m6	m7
Output Places	stop deliver	1							
	*deliver container		1						
	stop fill				1				
	fill container			1					
	stop depart						1		
	*depart silo					1			
	*container full					1	1		
	stop ship								1
	*ship container							1	
Input Transitions	2 --> 0	1							
	2 --> 3		1						
	2 --> 10			1					
	10 --> 2		1						
	10 --> 8				1				1
	10 --> 11			1				1	
	10 --> 26					1			
	26 --> 10							1	
	26 --> 24						1		
	26 --> 27					1			
	*input								

Attack Baseline Program (for PLC Instance 6)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 6.

Table 28: Tangible States for Attack Baseline (6)

	*stop deliver	fill container	*deliver container	*container full	*depart silo	*ship container
M0	1	0	0	0	0	0
M1	0	0	1	0	0	0
M2	0	1	0	0	0	0
M3	0	0	0	1	1	0
M4	0	0	0	0	0	1

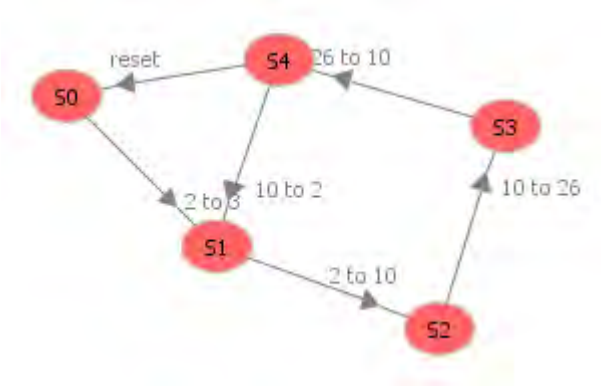

Figure 68: Graph for Attack Baseline (6)

Table 29: Matrix for Attack Baseline (6)

	PN marking	m0	m1	m2	m3	m4
Output Places	*stop deliver	1				
	*deliver container		1			
	fill container			1		
	*depart silo				1	
	*container full				1	
	*ship container					1
Input Transitions	2 --> 0					
	2 --> 3		1			
	2 --> 10			1		
	10 --> 2		1			
	10 --> 8					
	10 --> 11			1		1
	10 --> 26				1	
	26 --> 10					1
	26 --> 24					
	26 --> 27				1	
	*input	1				

Attack Baseline Program (for PLC Instance 7)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 7.

Table 30: Tangible States for Attack Baseline (7)

	*fill container	deliver container	stop deliver	stop fill
M0	0	0	1	0
M1	0	1	0	0
M2	1	0	0	0
M3	0	0	0	1

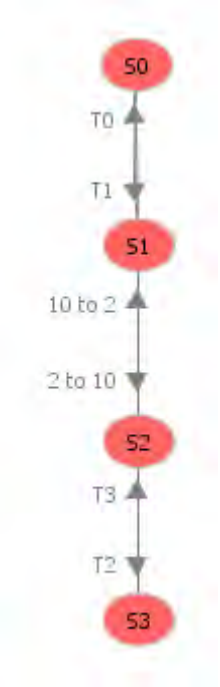

Figure 69: Graph for Attack Baseline (7)

Table 31: Matrix for Attack Baseline (7)

	PN marking	m0	m1	m2	m3
Output Places	stop deliver	1			
	deliver container		1		
	stop fill				1
	*fill container			1	
Input Transitions	2 --> 0	1			
	2 --> 3		1		
	2 --> 10			1	
	10 --> 2		1		
	10 --> 8				1
	10 --> 11			1	
	10 --> 26				
	26 --> 10				
	26 --> 24				
	26 --> 27				
	*input				

Attack Baseline Program (for PLC Instance 8)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 8.

Table 32: Tangible States for Attack Baseline (8)

	*stop deliver	*deliver container
M0	1	0
M1	0	1

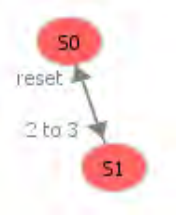

Figure 70: Graph for Attack Baseline (8)

Table 33: Matrix for Attack Baseline (8)

	PN marking	m0	m1
Output Places	*stop deliver	1	
	*deliver container		1
Input Transitions	2 --> 0		
	2 --> 3		1
	2 --> 10		
	10 --> 2		
	10 --> 8		
	10 --> 11		
	10 --> 26		
	26 --> 10		
	26 --> 24		
	26 --> 27		
	*input	1	

Attack Baseline Program (for PLC Instance 9)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 9.

Table 34: Tangible States for Attack Baseline (9)

	*ship container	container full	deliver container	depart silo	fill container	stop deliver	stop depart	stop fill	stop ship
M0	0	0	0	0	0	1	0	0	0
M1	0	0	1	0	0	0	0	0	0
M2	0	0	0	0	1	0	0	0	0
M3	0	0	0	0	0	0	0	1	0
M4	0	1	0	1	0	0	0	0	0
M5	0	1	0	0	0	0	1	0	0
M6	1	0	0	0	0	0	0	0	0
M7	0	0	0	0	0	0	0	0	1

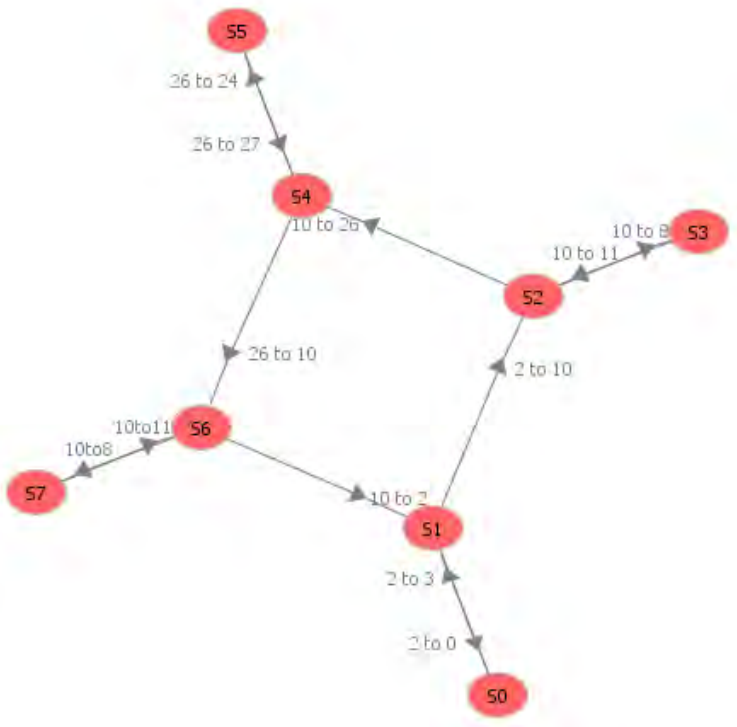

Figure 71: Graph for Attack Baseline (9)

Table 35: Matrix for Attack Baseline (9)

	PN marking	m0	m1	m2	m3	m4	m5	m6	m7
Output Places	stop deliver	1							
	deliver container		1						
	stop fill				1				
	fill container			1					
	stop depart						1		
	depart silo					1			
	container full					1	1		
	stop ship								1
	*ship container							1	
Input Transitions	2 --> 0	1							
	2 --> 3		1						
	2 --> 10			1					
	10 --> 2		1						
	10 --> 8				1				1
	10 --> 11			1				1	
	10 --> 26					1			
	26 --> 10							1	
	26 --> 24						1		
	26 --> 27					1			
	*input								

Attack Baseline Program (for PLC Instance 10)

The following state table, reachability graph and matrix are representative of *attack baseline* program for PLC instance 10.

Table 36: Tangible States for Attack Baseline (10)

	*deliver container	container full	depart silo	fill container	ship container	stop deliver	stop depart	stop fill	stop ship
M0	0	0	0	0	0	1	0	0	0
M1	1	0	0	0	0	0	0	0	0
M2	0	0	0	1	0	0	0	0	0
M3	0	0	0	0	0	0	0	1	0
M4	0	1	1	0	0	0	0	0	0
M5	0	1	0	0	0	0	1	0	0
M6	0	0	0	0	1	0	0	0	0
M7	0	0	0	0	0	0	0	0	1

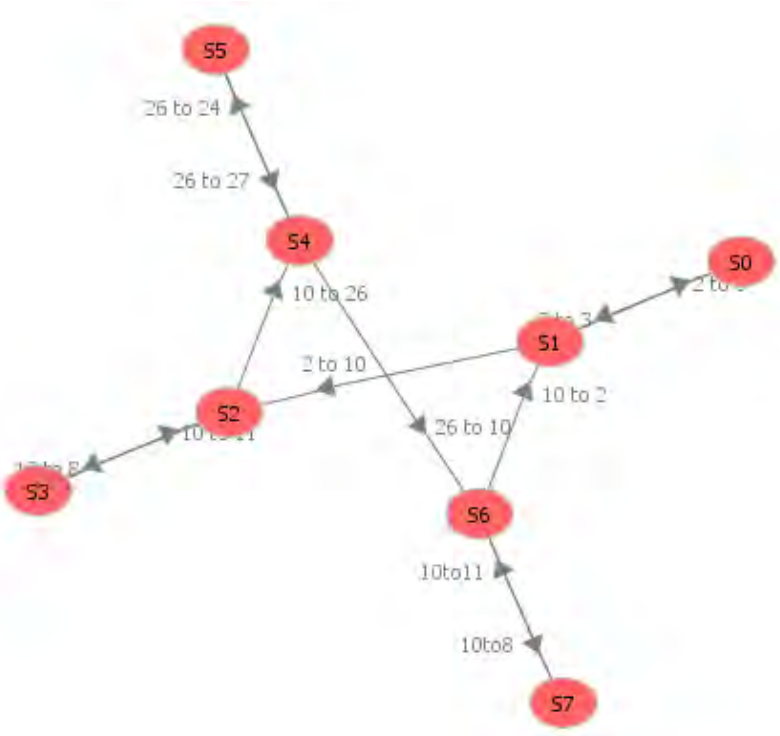

Figure 72: Graph for Attack Baseline (10)

Table 37: Matrix for Attack Baseline (10)

	PN marking	m0	m1	m2	m3	m4	m5	m6	m7
Output Places	stop deliver	1							
	*deliver container		1						
	stop fill				1				
	fill container			1					
	stop depart						1		
	depart silo					1			
	container full					1	1		
	stop ship								1
	ship container							1	
Input Transitions	2 --> 0	1							
	2 --> 3		1						
	2 --> 10			1					
	10 --> 2		1						
	10 --> 8				1				1
	10 --> 11			1				1	
	10 --> 26					1			
	26 --> 10							1	
	26 --> 24						1		
	26 --> 27					1			
	*input								

Bibliography

Abhishek, N. (2005). *Time Augmented Petri Nets for Modeling Discrete Event Dynamic Systems.* Durham NC: Duke University.

Almeida, R. M. (2010). Benchmarking the Resilience of Self-Adaptive Systems: A New Research Challenge. *29th IEEE International Symposium on Reliable Distributed Systems* (pp. 348-352). New Delhi: IEEE Computer Society.

Bolboaca, S. J. (2006). Pearson versus Spearman, Kendall's Tau Correlation Analysis on Structure-Activity Relationships of Biologic Active Compounds. *Leonardo Journal of Sciences*, 179-200.

Bonet, P. L. (2007). PIPE v2.5: a Petri Net Tool for Performance Modeling. *23d Latin American Conference on Informatics.* San Jose, Costa Rica.

Cutter, S. B. (2008). A place-based model for understanding community resilience to natural disasters. *Global Environmental Change*, 598-606.

Falliere, N. M. (2011). *W32.Stuxnet Dossier.* Cupertino CA: Symantec Corporation.

Germanus, D. K. (2010). Increasing the Resilience of Critical SCADA Systems Using Peer-to-Peer Overlays. *Architecting Critical Systems, First International Symposium.* Prague, Czech Republic: ISARCS.

Johnson, A. M. (1988). Survey of Software Tools for Evaluating Reliability, Availability, and Serviceability. *ACM Computing Surveys* , 227-269.

Minkel, J. (2008, Aug 13). *The 2003 Northeast Blackout--Five Years Later*. Retrieved 02 29, 2012, from Scientific American: http://www.scientificamerican.com/article.cfm?id=2003-blackout-five-years-later

National Infrastructure Advisory Council. (2009). *Critical Infrastructure Resilience Final Report and Recommendations.* Washington DC: Department of Homeland Security.

Niland, M. (2009, Feb 11). *Virus Disrupts Train Signals*. Retrieved Feb 29, 2012, from cbsnews: http://www.cbsnews.com/stories/2003/08/21/tech/main569418.shtml

Peng, S. Z. (2004). Ladder Diagram and Petri-Net-Based Discrete-Event Control Design Methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 523-531.

Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems.* New Jersey: Prentice Hall.

Peterson, J. L. (1977). Petri Nets. *ACM Computing Surveys*, 223-252.

Poulsen, K. (2003, Aug 19). *Slammer worm crashed Ohio nuke plant network*. Retrieved Feb 29, 2012, from SecurityFocus: http://www.securityfocus.com/news/6767

Queiroz, C. M. (2010). An analytical framework using performance modeling. *IEEE Globecom 2010.* Melbourne, Australia: IEEE Communication Society.

Reza, H. P. (2009). A Safety Analysis Method Using Fault Tree Analysis and Petri Nets. *Sixth International Conference on Information Technology: New Generations* (pp. 1089-1094). Las Vegas NV: IEEE Computer Society.

Roberts, P. (2005, Aug 18). *Zotob, PnP Worms Slam 13 DaimlerChrysler Plants*. Retrieved 02 29, 2012, from eWEEK: http://www.eweek.com/c/a/Security/Zotob-PnP-Worms-Slam-13-DaimlerChrysler-Plants/

Shah, A. P. (2008). Mechanisms to Provide Integrity in SCADA and PCS devices. *International Conference on distributed computing in sensor systems.* Sontorni Greece.

Smith, T. (2001, Oct 31). *Hacker jailed for revenge sewage attacks*. Retrieved Feb 29, 2012, from The Register: http://www.theregister.co.uk/2001/10/31/hacker_jailed_for_revenge_sewage/

Stouffer, K. F., Falco, J., Scarfone, K. (2008). *Guide to Industrial Control Systems (ICS) Security.* Gaithersburg MD: National Institute of Standards and Technology.

Thomas, P. (1998, Mar 18). *Teen hacker faces federal charges*. Retrieved 02 29, 2012, from CNN: http://www.cnn.com/TECH/computing/9803/18/juvenile.hacker/index.html

Tierney, K. B. (2007). Conceptualizing and Measuring Resilience. *TR News*, 14-17.

Trivedi, K. K. (2009). Resilience in Computer Systems and Networks. *Computer-Aided Design-Digest of Technical Papers* (pp. 74-77). San Jose CA: IEEE.

VanBreda, A. (2001). *Resilience Theory: A Literature Review.* Gezina, South Africa: Military Psychological Institute.

Wei, D. J. (2009). Resilient Industrial Control System (RICS): Concepts, Formulation, Metrics, and Insights. *2d International Symposium on Resilient Control Systems.* Idaho Falls ID: Resilient Control Systems.

Zurawski, R. M. (1994). Petri Nets and Industrial Applications. *IEEE Transactions on Industrial Electronics*, 567-582.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
22 Mar 2012	Master's Thesis	Sept 2010 – Mar 2012

4. TITLE AND SUBTITLE
Towards Quantifying Programmable Logic Controller Resilience Against Intentional Exploits

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Bushey, Henry W., Capt, USAF

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCO/ENG/12-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
POC: Eric Cornelius, Department of Homeland Security Industrial Control Systems Cyber Emergency Response Team Technical Lead
ATTN: NPPD/CS&C/NCSD/US-CERT
Mailstop: 0635, 245 Murray Lane, SW, Bldg 410
Washington, DC 20528
ics-cert@dhs.gov; (877)776-7585

10. SPONSOR/MONITOR'S ACRONYM(S)
DHS ICS-CERT

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Statement A. Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
This material is declared a work of the United States Government and is not subject to copyright protection in the United States.

14. ABSTRACT
Supervisory Control and Data Acquisition (SCADA) systems control and monitor services for the nation's critical infrastructure. Recent cyber induced events (e.g., Stuxnet) provide an example of a targeted, covert cyber attack against a SCADA system that resulted in physical effects. Of particular note is how Stuxnet exploited the trust relationship between the human machine interface (HMI) and programmable logic controllers (PLCs). Current methods for validating system operating parameters rely on message exchange and network communications protocols, generally observed at the HMI. Although sufficient at the macro level, this method does not provide detection of malware that exhibits physical effects via covert manipulation of the PLC, as demonstrated by Stuxnet. In this research, an alternative method that leverages direct analysis of PLC input and output to derive the true state of SCADA end-devices is introduced. The behavioral input-output characteristics are modeled using Petri nets to derive metrics for quantifying resilient properties of systems against malicious exploits. The results yield metrics that are applicable towards quantifying resilience in PLCs and implementing real-time security solutions. These findings enable detecting programming changes that affect input and output relationships, identifying the degree of deviation from a baseline program, and minimizing performance losses against disruptive events.

15. SUBJECT TERMS
Behavioral-based security, resilience, SCADA security, Petri net

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Butts, Jonathan, Maj, Ph.D., USAF
U	U	U	UU	172	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, x 4332 (jonathan.butts@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18