Chapter 6

QUANTIFYING CONTROLLER RESILIENCE USING BEHAVIOR CHARACTERIZATION

Henry Bushey, Juan Lopez and Jonathan Butts

Abstract Supervisory control and data acquisition (SCADA) systems monitor and control major components of the critical infrastructure. Targeted malware such as Stuxnet is an example of a covert cyber attack against a SCADA system that resulted in physical effects. Of particular significance 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, which are generally observed at the HMI. Although sufficient at the macro level, this method does not support the detection of malware that causes physical effects via the covert manipulation of a PLC. This paper introduces an alternative method that leverages the direct analysis of PLC inputs and outputs to derive the true state of SCADA devices. The input-output behavior characteristics are modeled using Petri nets to derive metrics for quantifying the resilience of PLCs against malicious exploits. The method enables the detection of programming changes that affect input-output relationships, the identification of the degree of deviation from a baseline program and the minimization of performance losses due to disruptive events.

Keywords: Programmable logic controllers, behavior characterization, resilience

1. Introduction

Supervisory control and data acquisition (SCADA) systems provide for the automated monitoring and control of major components of the critical infrastructure [1]. SCADA systems were implemented in many industry sectors as early as the 1960s, but security was not a priority at the time. However, recent events – intentional and unintentional – have raised concerns regarding SCADA security [10]. Non-intentional actions have traditionally been addressed using

redundant and fault tolerant architectures. Unfortunately, current solutions for dealing with intentional malicious actions are woefully inadequate [12].

A primary risk factor associated with intentional malicious actions is the trend to integrate SCADA systems and enterprise networks to save costs. Indeed, interconnecting critical SCADA systems via LAN and WAN technologies enables numerous attack entry points – from the Internet, internal workstations and communications links between the control center and field sites [10]. As demonstrated by Stuxnet, an attack that propagates from an enterprise network can execute code on field devices such as programmable logic controllers (PLCs) that ultimately results in physical damage to the process system [2].

Current methods for validating the functional parameters of a PLC primarily consider message exchange and network communications protocols, which are generally observed at the human machine interface (HMI). Although this method is sufficient at the macro level, it does not allow for the detection of malware that causes negative physical effects while employing deception techniques to mask the effects from the HMI. Additionally, quantifying the resilience of a PLC requires metrics for assessing its susceptibility to degradation and its ability to recover after an attack. This paper describes a method for detecting programming changes to PLCs by monitoring and characterizing PLC inputs and outputs. Focusing on PLCs at the micro level enables the effects of malicious actions to be observed despite efforts to mask the effects at the HMI, as was the case with Stuxnet.

2. Background

The National Infrastructure Advisory Council [4] defines resilience as the ability to reduce the magnitude and/or duration of disruptive events. An infrastructure is resilient if it can anticipate, absorb, adapt to and/or rapidly recover from a disruptive event. This definition provides a resilience framework with the following four characteristics:

- The ability to monitor the current state to identify deviations from an accepted baseline and anticipate potentially disruptive events.
- The ability to absorb potentially disruptive events by incorporating mechanisms that minimize the amount, if any, of performance loss.
- The ability to adapt by ensuring that contingencies are available that allow for flexible system adjustments to maintain operational availability.
- The ability to recover from disruptive events by incorporating automated or manual mechanisms that allow the system to provide functionality consistent with its baseline.

The method presented in this paper addresses the first two characteristics of the resilience framework and supports mechanisms that help address the last two characteristics. Effectively monitoring and absorbing disruptive events caused by malicious manipulations of PLC functionality requires examination

at the micro level. This level of inspection isolates the PLC and its effects on the external end devices from potentially deceptive reporting to the HMI. The monitoring process is, thus, able to detect the true output states in response to PLC inputs. Analyzing the true output behavior of a PLC provides an accurate method for measuring the absorptive capability of the PLC against disruptive events. Furthermore, the results support the evaluation of contingencies and mechanisms to examine the ability of a system to adapt to and recover from disruptive events.

Other research efforts focus primarily at the macro level and involve static analysis. Queiroz, *et al.* [7] present a model for quantifying SCADA system performance against denial-of-service attacks by determining the probabilistic failures of interdependent nodes of a SCADA system. Germanus, *et al.* [3] present a model in which SCADA system communications use redundant links. Both these methods approach their analysis at a macro level view of the system and are dependent on a trusted HMI for system status.

Shah, *et al.* [9] present a method for verifying PLC executable code. This method utilizes a challenge-response protocol between the verification functions of an untrusted PLC and an external device. While this method approaches the problem at the micro level, the authors have noted that certain logistical issues arise when taking a PLC offline. Note, however, that the method does not protect against timed attacks in which malware is inactive during the verification process. Also, the method does not mitigate the negative effects that occur after malicious code is detected.

Our analysis uses Petri nets derived from PLC inputs and outputs. Petri nets provide a powerful analysis method by abstracting the observed behavior in terms of its graphical and mathematical equivalents. A Petri net C is a four-tuple $C = (T, P, I, O)$ where T is a set of transitions, P is a set of places, I is a set of input functions for each transition and O is a set of output functions for each transition [5]. The Petri nets engaged in our research have been shown to be safe and bounded [6]. This yields finite reachability sets that provide quantifiable data to measure PLC performance during malicious events with respect to the resilience framework.

3. Behavior Characterization Methodology

This section describes the methodology for characterizing the input-output relationships of a PLC. An initial baseline program is established that incorporates PLC programming corresponding to an operational system. After the baseline is established, modifications are made to emulate a PLC infected with malware. Protective schemes are then applied to mitigate the effects of the malware. The enumerated instances of the PLC programs are evaluated to observe deviations in input-output behavior. Petri net models are then utilized to extract metrics that measure PLC security performance with respect to the resilience framework.

3.1 PLC Behavior Characterization

The two primary parameters monitored are: (i) the system input to the PLC; and (ii) the resulting output that translates directly to the physical system end devices (e.g., state of motors, lights and actuators). Because the output behavioral responses are based on the actual status of the physical end devices, the observations can be considered to represent the true state of the system. In order to categorize the observations, PLC interactions with the physical end devices are classified into three input-output 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 system interactions with end devices do not cause catastrophic losses.
- **Unstable:** Nominal input results in deviant and unsafe processes. An unsafe outcome is defined as a non-nominal output response in which system interactions with end devices cause catastrophic losses.

The research environment employed the LogixPro 500 programming software [8] for process emulation and the ProSim II tool [11] for simulation. LogixPro 500 provides a graphical user interface to develop, compile and execute instances of PLC programs with various system operating parameters. The multiple instances demonstrate distinct observable input-output behavior patterns when subjected to malicious attacks. For each instance, four program categories are established:

- **Baseline:** A program that meets the defined process requirements and generates valid input-output responses.
- **Attack Baseline:** A targeted attack applied to the baseline that generates degraded or unstable input-output responses.
- **Protection Baseline:** A protection scheme applied to the baseline that is intended to generate valid input-output responses. The protection scheme produces a fail-safe system state (e.g., flashing red lights at a traffic intersection).
- **Attack Protection Baseline:** A targeted attack applied to the protection baseline that generates valid, degraded or unstable input-output responses.

3.2 PLC Program Development

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, using the PLC program in conjunction with the targeted attacks

and protection schemes helps differentiate between the observed behavior and defined nominal process requirements.

A PLC is programmed with a baseline program that is analyzed indirectly by applying baseline input signals and observing the output. The program is analyzed for nominal output behavior (i.e., valid inputs resulting in the nominal output) to demonstrate the PLC baseline interactions with its environment; this establishes the baseline behavior patterns for the model. Next, a series of targeted changes to the baseline program code are implemented that are indicative of alterations by malware. These enumerated versions of the PLC baseline program are also analyzed indirectly by applying the baseline inputs to each enumerated version. The subsequent outputs, which may include inappropriate behavior (e.g., invalid output states), demonstrate the altered interactions of the PLC with its environment; these form the deviated behavior patterns for the model. The purpose of the deviated behavior patterns is to quantify the level of behavioral deviation that is observable in the baseline program. A protection scheme is then applied to the PLC baseline program code. The established protection scheme represents a resilient program that enters a safe state to counter degraded or unstable states. Finally, the targeted malware is applied to the protection scheme in order to evaluate the resilient behavior patterns (i.e., inputs resulting in safe or unsafe outputs). The purpose of the resilient behavior patterns is to quantify the level of behavioral deviation that is observable in the resilient program.

Petri net analysis is performed on the results; this mirrors the states and transitions in the PLC. The absorptive capacity of the protection scheme applied to the PLC is a derivative of the behavioral differences between the baseline and resilient programs. In theory, a fully resilient protection scheme would maximize these measured differences and provide full absorptive capacity against malicious code.

3.3 Petri Net Derivation

This section outlines the methodology for deriving each of the four program categories and the equivalent Petri nets.

1. Establish Baseline Program

(a) Develop a ladder logic program that implements the defined nominal process requirements. The baseline program generates valid system input-output responses.

Consider, for example, a silo plant that fills containers using a conveyer belt and automated sensors. The nominal processes are: bring an empty container into the plant, maneuver the container under the silo valve, fill the container until it is full, and ship the full container from the plant. Figure 1 shows the baseline program for the silo plant.

(b) Abstract the possible combinations of the inputs of the formal ladder logic as transitions T in a Petri net.

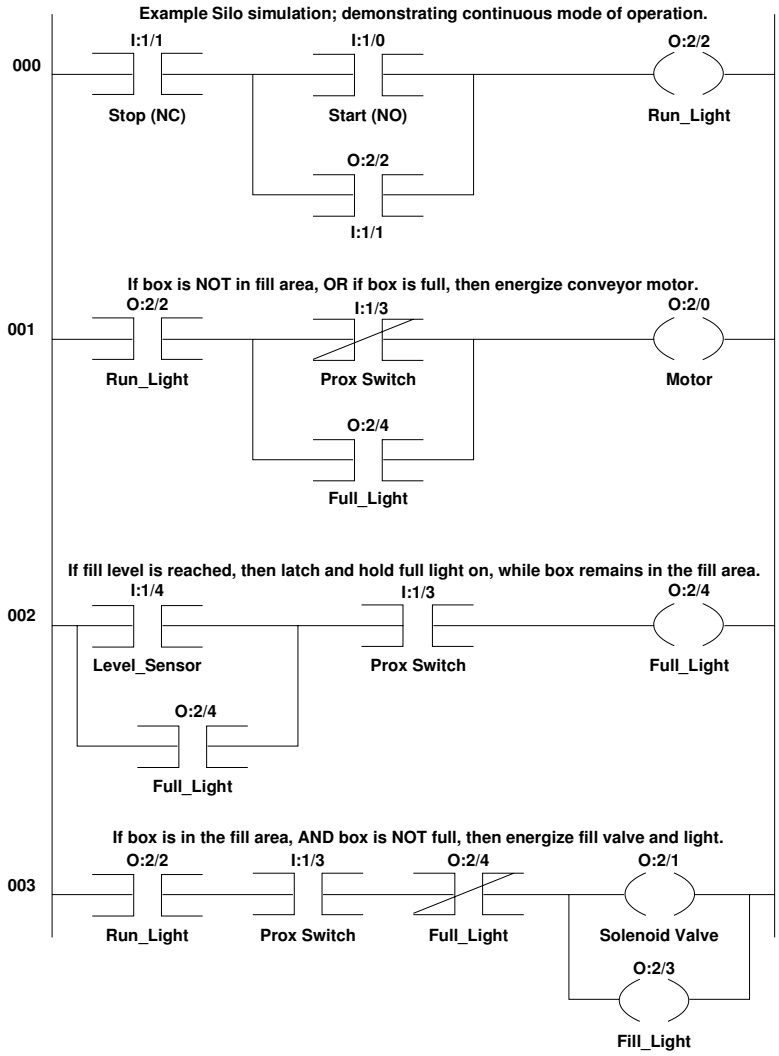

Figure 1. Example baseline ladder logic program for a silo plant.

A transition in a Petri net is defined as a change occurrence in the real-time input of the PLC instance. In this example, four input parameters contribute to the possible transitions in the Petri net: *start, stop, prox switch* and *level sensor* corresponding to I:1/0, I:1/1, I:1/3 and I:1/4, respectively, in Figure 1.

(c) Abstract the possible combinations of the outputs of the formal ladder logic as places P in a Petri net.

A place in a Petri net is defined as a physical state. In the silo plant, five output parameters contribute to the potential places in

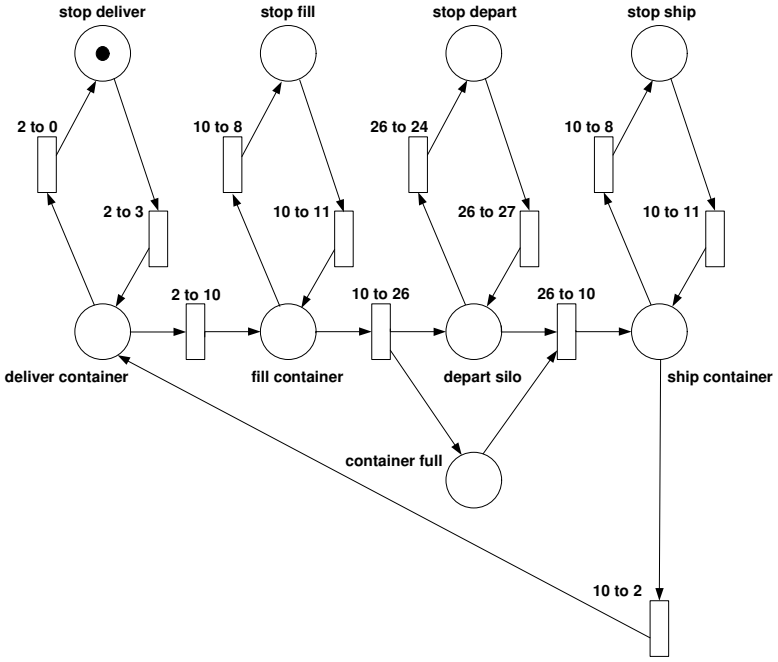

Figure 2. Initial baseline capture configuration.

the Petri net: *motor, solenoid valve, run light, fill light* and *full light* corresponding to O:2/0, O:2/1, O:2/2, O:2/3 and O:2/4, respectively in Figure 1.

(d) Abstract the input and output interdependencies of the formal ladder logic as input and output functions, I and O, for each of the potential transitions in the Petri net.

(e) Combine the subsets derived in Steps 1(a) through 1(d) to define the Petri net $C = (T, P, I, O)$ for analyzing the input-output behavior.

Figure 2 shows the Petri net derived from the example baseline PLC program. The token (black dot) in "stop deliver" denotes that the place is actively manifested as a physical state. From this state, the only transition enabled is Transition 2 to 3. Note that the label "2 to 3" represents the combined decimal input value for the PLC (i.e., Transition 2 to 3 indicates that the system input value changes from decimal value 2 to decimal value 3). Firing Transition 2 to 3 changes the marking of the Petri net and "deliver container" becomes the active physical state. Two transitions are enabled from the active state "deliver container." In cases where there are two or more transitions enabled, the Petri net non-deterministically selects the transition that fires. The resulting Petri net for the baseline

program identifies the possible states that the PLC can reach with valid operations.

2. Establish Attack Baseline

(a) Modify the baseline program ladder logic in a manner consistent with a targeted malicious attack. The attack baseline program generates degraded or unstable system input-output responses.

(b) Repeat Steps 1(a) through 1(e) to produce the equivalent Petri net for the attack baseline PLC program.

An example attack in this scenario targets the proximity sensor so that the silo valve is deceived into remaining open despite a container not being in close proximity to the fill station. The resulting attack manipulates the process to continually dispense product regardless of whether or not a container is in position.

3. Establish Protection Baseline

(a) Modify the baseline program ladder logic to incorporate a protective scheme that mitigates the effects of the attack baseline program. The protection baseline program, which implements fail-safe operation, is intended to generate valid system input-output responses.

(b) Repeat Steps 1(a) through 1(e) to produce the equivalent Petri net for the protection baseline PLC program.

The protection scheme implemented for this example utilizes additional ladder logic to provide a secondary fail-safe check for the silo valve to prevent it from opening when the conveyor belt is in motion.

4. Establish Attack Protection Baseline

(a) Modify the protection baseline program ladder logic to incorporate targeted attacks derived in the attack baseline. The attack protection baseline program generates degraded or unstable input-output responses.

(b) Repeat Steps 1(a) through 1(e) to produce the equivalent Petri net for the attack protection baseline PLC program.

In the case of the baseline attack example, the protection scheme provides fail-safe procedures for safeguarding against the observable effects of the targeted attack that manipulates the proximity sensor. The attack protection baseline program represents the attacks applied to the system with the implemented protection schemes.

In our experiments, we reviewed ten instances and several attacks on the silo process. Each PLC instance generated four programs and the corresponding Petri nets. A total of 40 Petri nets were utilized to mirror the physical states and transitions resulting from the different PLC programs. The quantitative analysis of the 40 Petri nets revealed several consistent findings for assessing PLCs with respect to the resilience framework.

Table 1. Set of tangible states (baseline).

	Container Full	Deliver Container	Depart Silo	Fill Container	Ship Container	Stop Deliver	Stop Depart	Stop Fill	Stop Skip
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

4. Analysis and Findings

The analysis of the Petri nets yielded several metrics that directly address or indirectly support the four characteristics of the resilience framework. The Petri net simulation application PIPE v4.0 was used to evaluate the Petri nets. The Petri net analysis produced a reachability matrix for each Petri net. These matrices were analyzed to identify comparative metrics that assess true input-output performance with respect to the resilience framework.

4.1 Reachability Matrix

A reachability matrix defines all the possible states in a given Petri net [6]. Table 1 shows the reachability matrix for an example baseline PLC instance. Note that the analysis of the ten instances and the various attacks is consistent with the example used in the discussion. The rows in the table represent the tangible states and the columns represent the places that characterize the states. The elements of each matrix are marked "0" or "1," representing the absence or presence of a token, respectively. For example, State M0 represents the Petri net marking in Figure 2 in which the place "stop deliver" is active. The baseline PLC presented for this instance has eight distinct states.

Table 2. Set of tangible states (attack baseline).

	Container Full	Deliver Container	Depart Silo	Fill Container	Ship Container	Stop Deliver	Stop Depart	Stop Fill	Stop Skip
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	1	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

Table 2 shows an example attack baseline matrix for the PLC instance. Note that State M4 has been altered to incorporate a targeted attack. The attack implemented in this instance causes the silo valve to remain open even though a full container has left the fill station. This state represents a change in the PLC that results in a degraded or unstable operating process, which is not defined in the Petri net corresponding to the baseline program.

Table 3. Comparison of baseline and protection programs.

Attack Instance	Baseline Program	Protection Baseline
1	d/u	fs
2	d/u	fs
3	d/u	fs
4	d/u	fs
5	d/u	fs
6	d/u	fs
7	d/u	fs
8	d/u	d/u
9	d/u	d/u
10	d/u	d/u

4.2 Metrics

The differences between the four program categories form the basis for deriving quantitative metrics. Table 3 summarizes the results of applying the attack instances against the baseline program and the protection baseline program. Note that "d/u" denotes degraded or unstable impact while "fs" denotes system transition to the fail-safe or resilient state.

Attack Instances 1 through 4 alter one state by manipulating one element (e.g., modifying the silo sensor for State M4). Attack Instances 5 through 7 alter one state by manipulating two elements. Attack Instances 8 through 10 alter two states by manipulating two elements for each state. As shown in Table 3, the protection baseline resulted in fail-safe operations for Attack Instances 1 through 7; however, the protection baseline was insufficient for attacks targeting multiple states and multiple elements (Attack Instances 8 through 10).

Comparing these observations with the pairwise differences observed for the ladder logic suggests a correlation between the net changes to the output behavior and the net changes to the ladder logic. As indicated by the results, the most significant metric resulting from the evaluation is the difference observed between the PLC program and the input-output behavior corresponding to the instance. With respect to the resilience framework, this evaluation serves as a self-sufficient metric as well as a complementary metric. Indeed, comparative analysis directly addresses two aspects of resilience (i.e., detecting a change occurrence and quantifying the degree of change occurrence) and supports mechanisms to minimize performance losses due to disruptive events.

The following list summarizes the implications with respect to the resilience framework:

- The ability to anticipate a potentially disruptive event requires the system to be self-aware of its baseline and monitor its current state.

The proposed evaluation identifies when physical input-output relationships deviate from the baseline. This metric may support one of two triggering mechanisms: (i) the number of identified deviations exceeds a threshold; and (ii) a violation against a whitelist of expected outcomes.

- The ability to absorb potentially disruptive events requires the system to have mechanisms in place that minimize the amount, if any, of performance loss.

  The proposed evaluation in combination with the difference in ladder logic changes can help assess the ability of a PLC to absorb disruptive events: (i) if the input-output behavioral difference is zero, then the differences in ladder logic may be used to assess the inherent robustness of the PLC ladder logic program; and (ii) if the input-output behavioral difference is greater than zero, then the differences observed in the input-output behavior may be utilized to assess the overall absorption by the PLC.

- The ability to adapt requires the system to have contingencies that permit flexible system adjustments to maintain operational availability.

  The proposed evaluation supports this feature by providing triggering mechanisms that initiate adaptive processes. Note that the adaptive processes may exist external to the PLC (e.g., requiring coordination with additional hardware and/or software).

- The ability to recover from a disruptive event requires mechanisms, either automated or manual, that enable the system to operate in a manner consistent with its baseline.

  The proposed evaluation supports this feature by providing triggering mechanisms that initiate recovery processes. Note that the recovery processes may exist external to the PLC (e.g., requiring coordination with additional hardware and/or software).

4.3 Extending the Implementation

Although the analysis presented here focuses on an individual PLC program implementation, the methodology can be applied to a more general protection scheme. This is important because the protection mechanism may be an external, and preferably, parallel process. Figure 3 illustrates an example Petri net that models the input-output behavior metric as the primary means for monitoring and detecting state security.

In the example, a monitoring system is incorporated to signal the need to transition to a backup PLC when unstable or degraded operations are detected. The primary PLC executes the baseline program; however, the secondary protective PLC is isolated from direct communications with the SCADA network. If a deviation from the expected behavior is detected, then the secondary PLC triggers a fail-safe operation. During nominal operations, the subnet of the

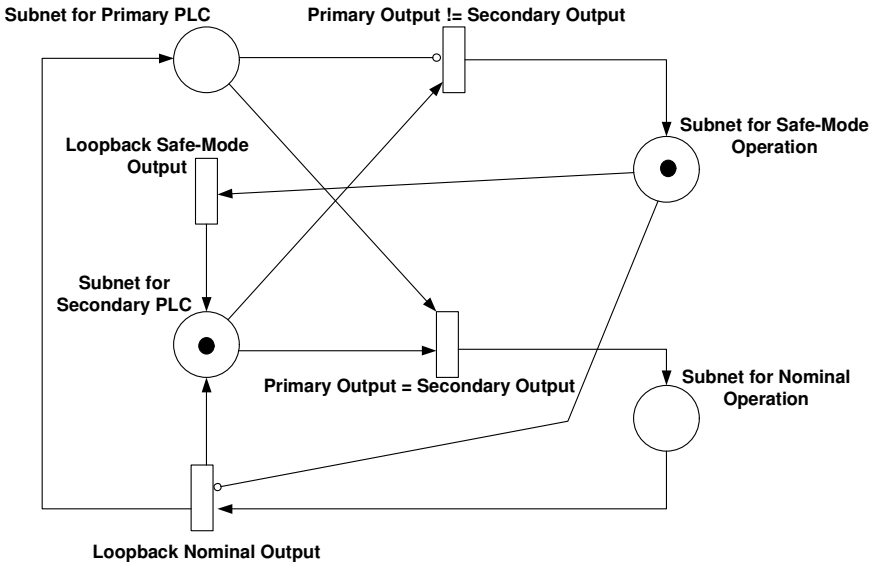

Figure 3. Petri net model for monitoring and detecting security in a fail-safe state.

primary PLC controls process flow. In the case of an input-output behavior deviation, process control is transferred to the subnet of the secondary PLC as represented by the current state in the Petri net. The Petri net illustrates an architecture that supports absorptive, adaptive and recovery features that are triggered primarily by an analysis of the input-output metric.

5. Conclusions

The behavior-based method described in this paper provides a practical means for assessing the security posture of a PLC against malicious code. The resulting framework helps quantify PLC resilience in terms of its ability to monitor, detect and absorb intentional malicious actions. Analyzing the system in real time for nonconforming behavior by the PLC supports attack detection and mitigation. Indeed, metrics from input-output behavior characterization constitute true representations of system state that cannot be deceived by alterations at an HMI or communications channel. Thus, the behavior-based method provides a measure of PLC resilience against malicious code and provides a baseline for quantitatively assessing the security posture. Indeed, analyzing security at the micro level by focusing on field devices and system functions can help prepare for and address future Stuxnet-like attacks.

Note that the views expressed in this article are those of the authors and do not reflect the official policy or position of the U.S. Air Force, Department of Defense or the U.S. Government.

References

[1] Department of Homeland Security, National Infrastructure Protection Plan, Washington, DC, 2009.

[2] N. Falliere, L. O'Murchu and E. Chien, W32.Stuxnet Dossier, Symantec Corporation, Cupertino, California, 2011.

[3] D. Germanus, A. Khelil and N. Suri, Increasing the resilience of critical SCADA systems using peer-to-peer overlays, *Proceedings of the First International Symposium on Architecting Critical Systems*, pp. 161–178, 2010.

[4] National Infrastructure Advisory Council, Critical Infrastructure Resilience Final Report and Recommendations, Department of Homeland Security, Washington, DC, 2009.

[5] J. Peterson, Petri Nets, *ACM Computing Surveys*, vol. 9(3), pp. 223–252, 1977.

[6] J. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Upper Saddle River, New Jersey, 1981.

[7] C. Queiroz, A. Mahmood and Z. Tari, Survivable SCADA systems: An analytical framework using performance modeling, *Proceedings of the IEEE Global Communications Conference*, 2010.

[8] Rockwell Automation, RSLogix 500, Milwaukee, Wisconsin.

[9] A. Shah, A. Perrig and B. Sinopoli, Mechanisms to provide integrity in SCADA and PCS devices, *Proceedings of the International Workshop on Cyber-Physical Systems Challenges and Applications*, 2008.

[10] K. Stouffer, J. Falco and K. Kent, Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security, NIST Special Publication 800-82, National Institute of Standards and Technology, Gaithersburg, Maryland, 2006.

[11] The Learning Pit, ProSim II, Whitby, Ontario, Canada.

[12] G. Wilshusen, Cybersecurity: Continued Attention Needed to Protect Our Nation's Critical Infrastructure, GAO Report GAO-11-865T, Government Accountability Office, Washington, DC, 2011.