PNNL-38717

Resilience Through Data-Driven, Intelligent Designed Control

A Formal Methods Approach

November 2025

Nawaf Nazir
Veronica Adetola
Samuel L. Litchfield

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

PNNL-38717

Resilience Through Data-Driven, Intelligent Designed Control

A Formal Methods Approach

November 2025

Nawaf Nazir
Veronica Adetola
Samuel L. Litchfield

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Abstract

The PNNL and GTRI team developed a strategy to integrate temporal logic rule specification for detection of cyber-intrusion in the source code and control algorithms of CPS using advanced cyber-data. The GTRI team utilized its capabilities in rule synthesis and temporal logic specifications for software assurance and verification to detect and predict impact of cyber-intrusions and malware in the computational and control algorithms of cyber-physical systems. The team also developed a testing and verification approach that could be used to validate the suggested approach against a realistic use-case CPS showcasing improvements in system impact prediction performance. Temporal logic offers a compact expression of events in absolute and relative time and has a formalized translation to state machines. As such, temporal logic rules can feasibly be synthesized to any system as a rule engine, with the process being formally verified to be correct. The goal here is to utilize temporal logic rules to detect cyber-attacks and manipulations in the computational algorithms and provide real-time software assurance and verification guarantees.

Summary

To address the proposed work, the team targeted the detection of cyber-intrusions in source code and network communication of Programmable Logic Controllers (PLCs), as these digital systems are responsible for the control of CPS within several sectors of Critical Infrastructure across the United States. PLCs are commonly programmed through a set of standardized configuration languages. These are commonly known as Ladder Logic, Structured Text (ST), or Instruction List (IL). Ladder Logic is a pictorial programming scheme where engineers connect vertical lines with horizontal lines of computation to sense input states, make decisions, and control actuator states. Structured Text and Instruction List perform similar functions in a programming style more akin to Assembly Language or hardware RTL. To integrate temporal logic rule specification and other AI/ML based methods, the team augmented and analyzed the Structured Text representation of PLC configuration. This augmentation could provide assurance guarantees on the physical behavior of the PLC in the case of Temporal Logic, and detection of network data manipulation or configuration changes through other AI/ML based methods.

Acknowledgments

We would like to acknowledge the significant contribution of Georgia Tech Research Institute (GTRI) team partners in this project, including William Stuckey, Samul Litchfield, Travis Ayres, and Jacob Ashmore. We would also like to acknowledge the RD2C initiative leadership for their support especially Marisa DeCillis.

Acronyms and Abbreviations

1. PNNL → Pacific Northwest National Laboratory
2. GTRI → Georgia Tech Research Institute
3. AI → Artificial Intelligence
4. ML → Machine Learning
5. CPS → Cyber Physical Systems
6. PLC → Programmable Logic Controller
7. ST → Structured Text
8. IL → Instruction List
9. LTL → Linear Temporal Logic
10. RCE → Remote Code Execution
11. APT → Advanced Persistent Threats
12. SMV → Symbolic Model Verification
13. GBA → Generalized Bucci Automata
14. HDL → Hardware Description Language
15. FSM → Finite State Machine

Contents

Figures

Tables

1.0 Introduction

The objective of this project is to develop an approach to predict cyber-intrusions in the computational layer of CPS. We utilize rule synthesis and temporal logic specifications for software assurance and verification to detect and predict impact of cyber-intrusions and malware in the computational and control algorithms of CPS. Temporal logic offers a compact expression of events in absolute and relative time and also has a formalized translation to state machines. As such, temporal logic rules can feasibly be synthesized to any system as a rule engine, with the process being formally verified to be correct. The goal here is to utilize temporal logic rules to detect cyber-attacks and manipulations in the computational algorithms and provide real-time software assurance and verification guarantees. The GTRI/PNNL team developed a scenario to demonstrate the approach. This includes a simulation environment and applicable threat models.

2.0 Technical Approach

2.1 Problem Scenario

The GTRI/PNNL team developed a hydropower plant scenario to demonstrate the approach. This includes a simulation environment and applicable threat models.

2.1.1 Environment

The simulation environment scenario is kept simple for initial testing. The environment consists of a hydroelectric turbine, penstock, and controlling PLC. The tools and details of the environmental setup are described in further detail in Section 2.3.1 Co-Simulation.

2.1.2 Threat Model

The GTRI team identified two different threat models in conjunction with PNNL partners. The first threat model is data-modeling focused and generally represents a compromise external to the local station. The second threat model is more severe and covers the plausible scenario of compromise within a local station. These two threat models are explored below.

2.1.2.1 Network-based PLC Data Poisoning

Network-based data poisoning is explored as the first threat-model because the viable attack surface is higher and skill needed to perform the attack is relatively low. This makes the threat more likely to occur. The threat model considers the compromise of a remote sensor or data link connecting a sensor to a PLC. A few common factors compound to make these sensors and links more vulnerable: the links are often unencrypted (at least for some part of the link), and they may cross sites increasing the number of entry points, particularly remotely accessible entry points. It is also common for a failed sensor, or failed link, to be part of a redundant sensor network. In these situations, detecting intrusion vs failure is of particular interest as well as formal proofs that an uncompromised PLC correctly handles the failure. This attack is less powerful than others, as compromised data streams can be detected though formally specified traditional approaches, like network timing and packet analysis. This threat model is shown in Figure 1.

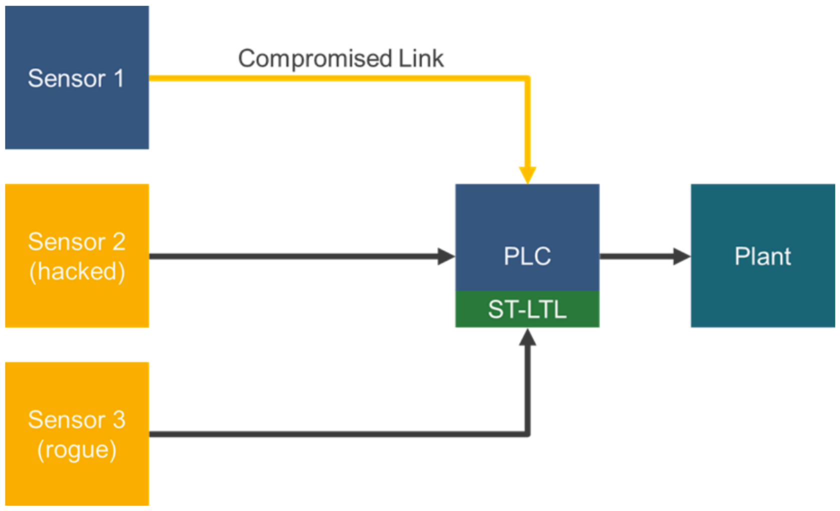


Figure 1: Data Poisoning Threat Model

2.1.2.2 Runtime PLC Compromise

Runtime PLC Compromise is explored as the second threat-model because it forms a realistic and necessary comparison to Network-based Data Poisoning. This threat model assumes an adversary has gained malicious access to a PLC and is capable of manipulating or rewriting either its configuration or it's Structured Text configuration/program.

There are a variety of ways adversaries could accomplish this. Remote Code Execution (RCE) vulnerabilities would provide attackers remotely accessible compromises but are reliant on network connectivity and vulnerable combinations of PLC models and currently active firmware versions.

Physical access is another possible compromise method, which has the obvious and necessary requirement of physically being able to manipulate the hardware, but potentially requires less technical acumen to achieve.

In either case, the ability of adversaries to directly manipulate firmware or ST configuration/programming provides them a greater ability to affect local system behavior. This leads to a higher average consequence to the system, with adversaries potentially being able to subtly manipulate sensor values and trigger malicious implants only under certain timing or system states, exhibiting common characteristics of Advanced Persistent Threats (APT).

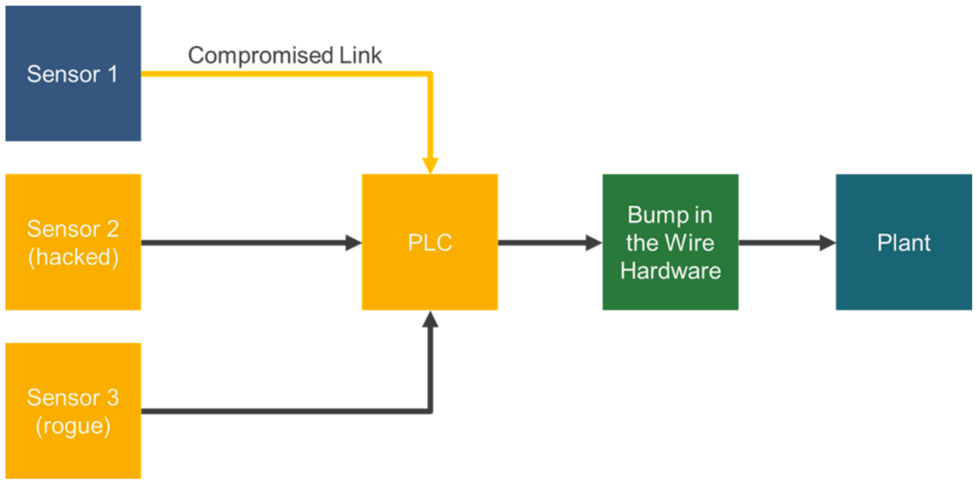


Figure 2: Runtime PLC Compromise Threat Model

2.2 Proposed Solution

The central research thesis of the approach is to use temporal logic as a common mathematical binder for intrusion detection and functional correctness specification. This enables engineers to make higher-level behavioral descriptions, without having to spend as much effort on verifying implementations for correctness by hand. This is done through the use of (semi-)automated translation steps between representations. The translation steps can be seen in Figure 3.

There are four main steps in the proposed approach:

1. **Specification** – system designers and engineers use temporal logic to specify the system, focusing on correct behaviors and intrusion metrics. These specifications are represented

by the orange boxes. The engineers also design the key physical behaviors and program code in structured text, represented by the top blue box. In the traditional flow, the specification is skipped and at this point the system begins simulated and in-situ testing.

2. **Synthesis** – In the proposed approach the higher-level specifications and structured text are (semi-)automatically translated to NuSMV, which can perform symbolic model checking. This is more suitable to formal analysis and proving. After translation, all key aspects of the system are preserved, but re-represented.
3. **Combination** – desired system behaviors (like correctness and intrusion detection specifications) are combined with the system definition now that both are represented in a common mathematical domain. Proofs of behavior are discharged and results reported.
4. **Translation** – the NuSMV domain artifacts need to be re-lowered to ST or another language that can actually be deployed. This process is also (semi-)automated.
5. **Deployment** – The verified artifacts are compiled and deployed, including any needed automatic generation of runtime monitors for intrusion detection.

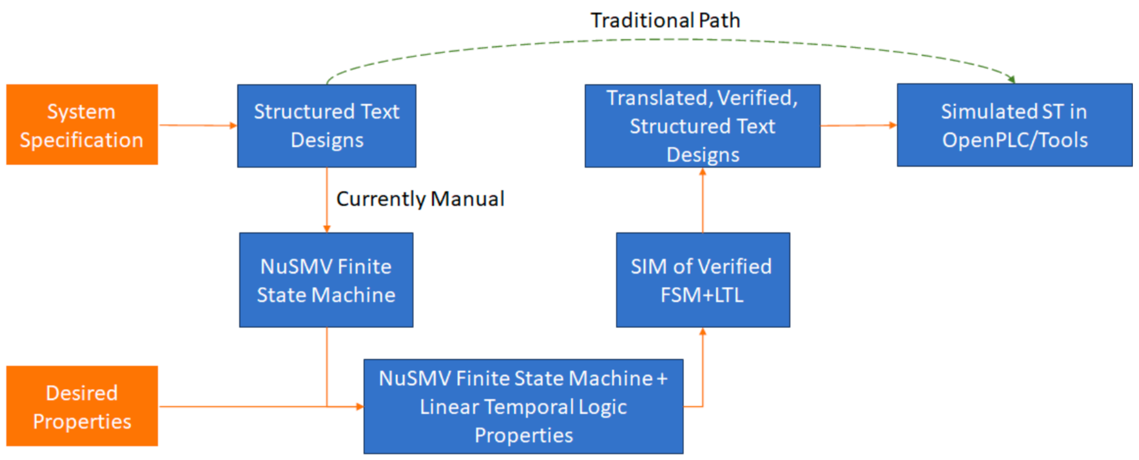

Traditional Path
System Specification
Structured Text Designs
Translated, Verified, Structured Text Designs
Simulated ST in OpenPLC/Tools
Currently Manual
NuSMV Finite State Machine
SIM of Verified FSM+LTL
Desired Properties
NuSMV Finite State Machine + Linear Temporal Logic Properties

Figure 3: Top Level Approach

2.2.1 Temporal Logic Rule Specifications

Temporal logic is well suited to decision making specification because it combines Boolean logic decision making, with aspects of time. Often security specifications aren't just of the nature of allowing or disallowing something based on observed patterns, but the timing of those patterns matters greatly. Linear Temporal Logic (LTL), the temporal logic of choice for this project, creates a grammar by adding temporal operators to the standard Boolean operators. A specification engineer or end user can then describe time-based relations with ease as compared to using manual buffers and conditional logic in a programming language. Auditors can also then much more easily understand what exactly was specified compared to parsing and testing code.

The key enabling transformation is from LTL to Generalized Bucci Automata (GBA), a graph adjacent representation closely related to graph representations of data and control flow used by programming language and HDL compilers, including ST, HDL, C, and assembly. The translation from LTL to GBA has existing tooling available, such as Spot, and can be formally characterized and proven for correctness. This is the base step that automatically transforms LTL to a lower-level abstract representation. Then, domain specific programs (as proposed by this effort) can complete the translation from the GBA description into formats suitable for functional correctness proofs, or runtime intrusion monitors. These artifacts can then be incorporated into any standard testing environment including unit test and simulations to help determine if the specifications completely covered the desired behaviors.

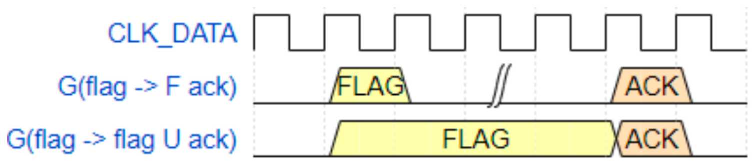


Figure 4: Example of LTL Trace in HDL Waveform Viewer

One advantage to this approach is that most or all of the computationally expensive steps are done prior to software deployment. This division is shown in Figure 5. The synthesized runtime artifacts are then highly assured and if deployed in hardware are capable of responding within micro or nano seconds. An example of LTL behaviors synthesized in hardware can be seen in Figure 4.

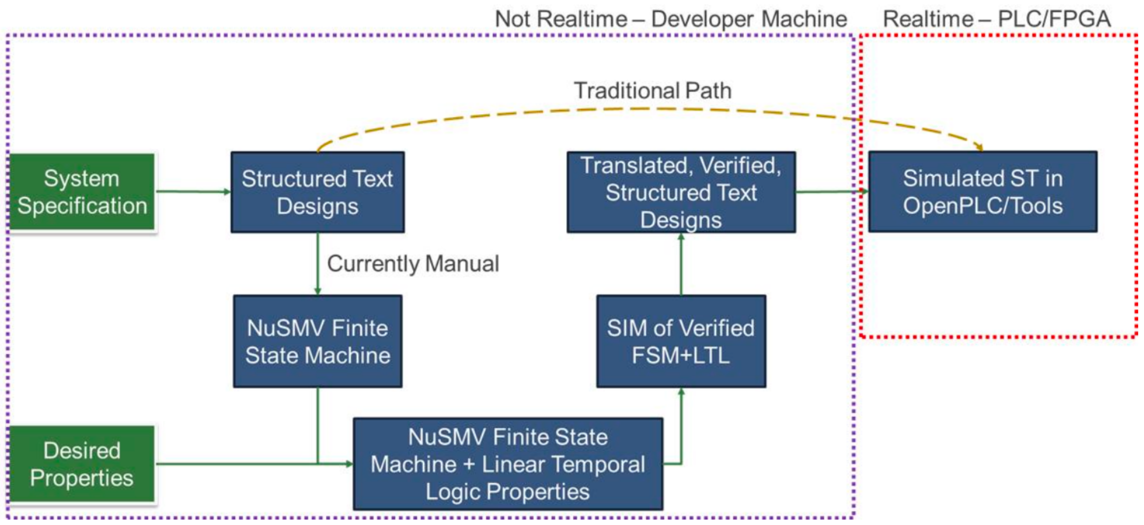

Not Realtime – Developer Machine
Realtime – PLC/FPGA
Traditional Path
System Specification
Structured Text Designs
Translated, Verified, Structured Text Designs
Simulated ST in OpenPLC/Tools
Currently Manual
NuSMV Finite State Machine
SIM of Verified FSM+LTL
Desired Properties
NuSMV Finite State Machine + Linear Temporal Logic Properties

Figure 5: Processing Time Description

2.2.2 Research Methods for Software Assurance

Formal methods suffer from low adoption due to perceived complexity and having a relatively small number of practitioners in the field. Automated translation of formal specifications can bridge the gap, reducing the burden of applying leading-edge software assurance methods. Having the ability to extract LTL specifications directly from Structured Text in a “bottom-up” way gives system designers another tool to increase overall system assurance while minimizing friction.

A “bottom-up” approach of extracting LTL specifications from extant code would provide an avenue for increased assurance for already fielded systems. When combined with “top-down” approaches like SysMLv2, the potential for end-to-end formally specified systems with greatly reduced adoption costs can be realized.

2.3 Testing and Verification

To test and validate our methods in this project, the team decided to go with a co-simulation approach for a model hydropower plant. This provides a playground to test all manner of generated artifacts in a realistic but approachable scenario.

2.3.1 Co-Simulation

The Co-simulation is composed of three parts. A block diagram showing the different components can be seen in Figure 6.

1. OpenPLC Python Sub Module
2. OpenModelica OMPython
3. OpenHPL

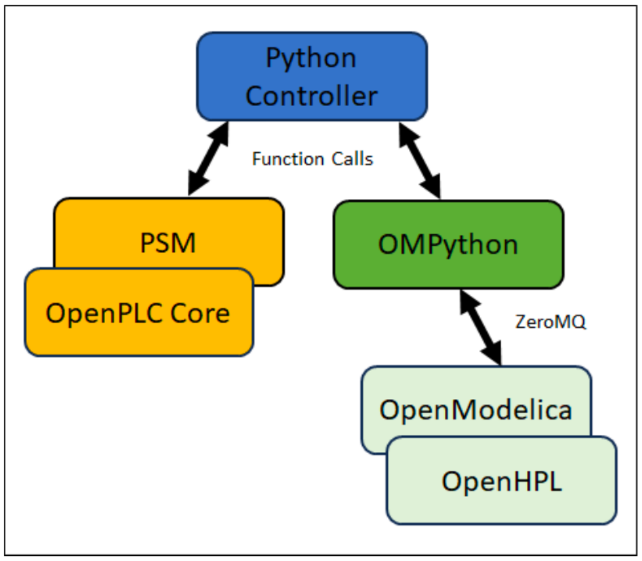


Figure 6: Architecture of the Co-Simulation Environment

OpenPLC is an open-source Programmable Logic Controller. The OpenPLC project was created in accordance with the IEC 61131-3 standard, which defines the basic software architecture and programming languages for PLCs. OpenPLC supports a wide variety of targets. This includes hardware like microcontrollers, PLCs and software only. The OpenPLC Python SubModule (PSM) a software only target that serves as the bridge connecting the OpenPLC core to Python programs. PSM allows you to directly interface OpenPLC IO using Python as well as write drivers for expansion boards using just regular Python. This interface allows you to interact with OpenPLC "hardware" and IO [1].

OMPython is the OpenModelica Python scripting interface. It is a free, open source, highly portable Python based interactive session handler for Modelica scripting. It provides the modeler with components for creating a complete Modelica modeling, compilation and simulation environment based on the latest OpenModelica library standard available. OMPython is architected to combine both the solving strategy and model building. It requires a full installation of the OpenModelica software to function [2].

OpenHPL is an open-source hydropower library that consists of hydropower unit models and is modelled using OpenModelica. OpenHPL makes it possible to model hydropower systems of different complexity and connect them with models from other libraries, e.g., with models of the power system or other power generating sources [3].

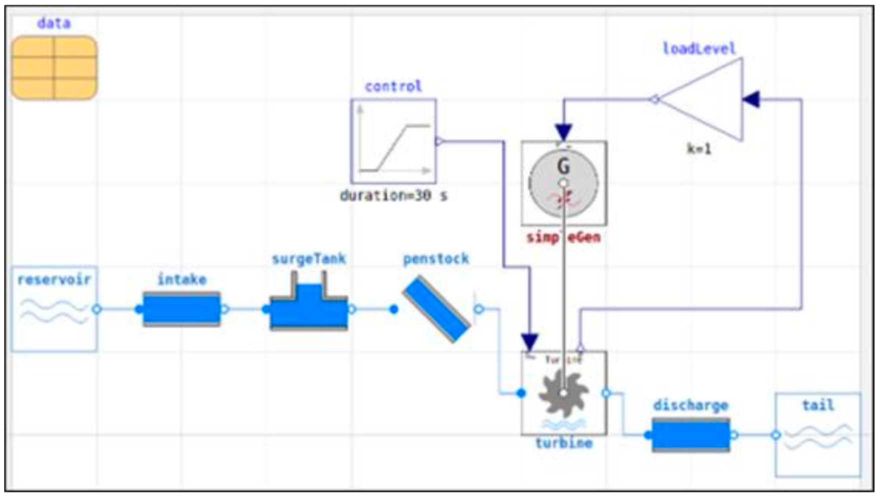
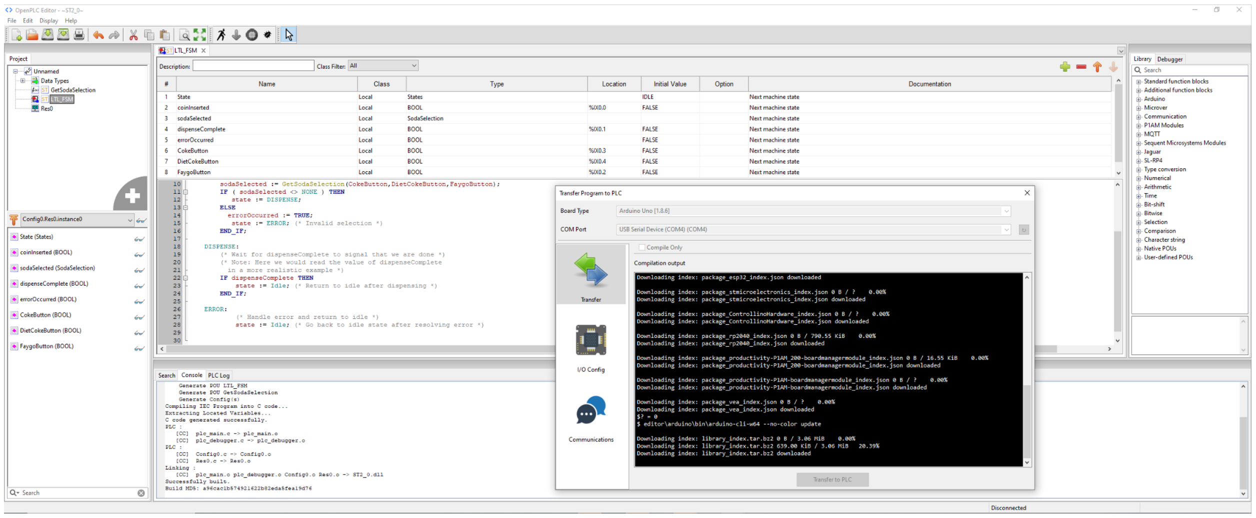


Figure 7: OpenHPL block diagram for a Simple Power Generation Model

To build out a simple test environment, the team built a simple OpenPLC application and ran it via the PSM soft-hardware interface. The PSM is set up as a Python virtual environment in the root directory when the OpenPLC runtime is installed; by activating this environment from the command line, OMPython library is installed and activated. From OMPython, the desired model is initialized, along with any dependencies. A PSM script interfaces between the two components, by querying the model, stepping through the simulation and responding when an input from the PLC is activated. As the team continues to refine the OpenPLC model, there will be a button to control the penstock in the OpenHPL simulation. Utilizing this system will allows the construction arbitrarily complex models that can test and validate the other work performed.



#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	State	Local	States		IDLE		Next machine state
2	coinInserted	Local	BOOL	%IX0.0	FALSE		Next machine state
3	sodaSelected	Local	SodaSelection				Next machine state
4	dispenseComplete	Local	BOOL	%IX0.1	FALSE		Next machine state
5	errorOccurred	Local	BOOL		FALSE		Next machine state
6	CokeButton	Local	BOOL	%IX0.3	FALSE		Next machine state
7	DietCokeButton	Local	BOOL	%IX0.4	FALSE		Next machine state
8	FaygoButton	Local	BOOL	%IX0.2	FALSE		Next machine state

```
        sodaSelected := GetSodaSelection(CokeButton,DietCokeButton,FaygoButton);
        IF ( sodaSelected <> NONE ) THEN
          state := DISPENSE;
        ELSE
         errorOccurred := TRUE;
          state := ERROR; (* Invalid selection *)
        END_IF;

    DISPENSE:
        (* Wait for dispenseComplete to signal that we are done *)
        (* Note: Here we would read the value of dispenseComplete
          in a more realistic example *)
        IF dispenseComplete THEN
           state := Idle; (* Return to idle after dispensing *)
        END_IF;

    ERROR:
            (* Handle error and return to idle *)
           state := Idle; (* Go back to idle state after resolving error *)
```

Board Type: Arduino Uno [1.8.6]

COM Port: USB Serial Device (COM4) (COM4)

Compile Only [ ]

```
Downloading index: package_esp32_index.json downloaded

Downloading index: package_stmicroelectronics_index.json 0 B / ?    0.00%
Downloading index: package_stmicroelectronics_index.json downloaded

Downloading index: package_ControllinoHardware_index.json 0 B / ?    0.00%
Downloading index: package_ControllinoHardware_index.json downloaded

Downloading index: package_rp2040_index.json 0 B / 790.55 KiB    0.00%
Downloading index: package_rp2040_index.json downloaded

Downloading index: package_productivity-P1AM_200-boardmanagermodule_index.json 0 B / 16.55 KiB    0.00%
Downloading index: package_productivity-P1AM_200-boardmanagermodule_index.json downloaded

Downloading index: package_productivity-P1AM-boardmanagermodule_index.json 0 B / ?    0.00%
Downloading index: package_productivity-P1AM-boardmanagermodule_index.json downloaded

Downloading index: package_vea_index.json 0 B / ?    0.00%
Downloading index: package_vea_index.json downloaded
$? = 0
$ editor\arduino\bin\arduino-cli-w64 --no-color update

Downloading index: library_index.tar.bz2 0 B / 3.06 MiB    0.00%
Downloading index: library_index.tar.bz2 639.00 KiB / 3.06 MiB   20.39%
Downloading index: library_index.tar.bz2 downloaded
```

```
    Generate POU LTL_FSM
    Generate POU GetSodaSelection
    Generate Config(s)
Compiling IEC Program into C code...
Extracting Located Variables...
C code generated successfully.
PLC :
   [CC]  plc_main.c -> plc_main.o
   [CC]  plc_debugger.c -> plc_debugger.o
PLC :
   [CC]  Config0.c -> Config0.o
   [CC]  Res0.c -> Res0.o
Linking :
   [CC]  plc_main.o plc_debugger.o Config0.o Res0.o -> ST2_0.dll
Successfully built.
Build MD5: a96cac1b574921622b82eda5fea19d76
```

Figure 8: Exemplar Design Deployed on OpenPLC

2.3.2 Network-based PLC Data Poisoning

With an environment, threat model, and technical approach defined, threats can be modeled by injecting data in the co-simulation environment. The first scenarios will focus on the threat model described in 2.1.2.1 Network-based PLC Data Poisoning. The simulation will begin with injecting faulty and malicious turbine data and demonstrated response by a runtime detection monitor. More advanced detections will include periods of correct data and intentionally crafted errant data design to confuse the PLC. Future scenarios will consider more complex threat models, and physical plant configurations.

3.0 Future Work

There are multiple opportunities to extend the concepts in this project; this section will highlight those which we see as having the largest potential impact. The project team has in-house capabilities necessary to expand the work with the following:

3.1 Integrated Demo

GTRI can support a fully integrated demo on real hardware, including PLCs, microcontrollers, and bump-in-the-wire reconfigurable hardware.

3.2 Direct Online Monitor Synthesis

With the possibility of extracting LTL specifications mechanically (either from a Structured Text representation directly or created manually by system designers) comes the possibility of creating a “self-checking” functionality directly on-target. This would allow real-time feedback from fielded PLC system.

3.3 Probabilistic Integration

Extending the proposed LTL-based assurance mechanisms, GTRI has the capability of adding an additional layer of manipulation detection through the use of probabilistic approaches. There would be two primary applications of this: 1) providing additional detection mechanisms on stochastic system measurements, or 2) providing a confidence score to detection mechanisms to quantify security states.

There are two additional detection mechanisms the team could explore that detect Runtime manipulations. The simpler of the two would be to leverage the relatively stable nature of PLC scan cycle execution times to detect deviations. These scan cycle times are approximately normal, but PLCs are designed for each cycle’s execution to be predictable and bounded. In the simplest form, observing the mean and standard deviation of scan cycle times for a given configuration over a long period of time would form a baseline, and deviations from that baseline would indicate a change in system behavior, potentially pointing to malicious configuration manipulation or hardware failure. More sophisticated versions of this approach could include AI/ML applications for similar approaches to system states.

The more complicated type of the two additional mechanisms would be to identify empirical probabilities to state transitions in the LTL specification of the system. When the system arrives in a terminal state, the state path taken to arrive in that final arrangement includes the set of transitions and transition probabilities taken to get there. The transition probabilities along these paths could form a metric for confidence in the final state or system inputs, as a transition path of primarily low probability transitions indicates a combined series of extremely unlikely events. This could arise naturally through particularly aberrant system inputs, but this metric, when combined with other proposed assurance mechanisms, could be an indicator of malicious input manipulation. A similar approach could be implemented in the control flow graph of the PLC’s firmware directly, which would provide insight into commonly utilized firmware paths or code. While this would be extremely interesting to study with respect to the runtime manipulation threat model, the work required to implement and verify this would be significantly greater.

4.0 Conclusion

The GTRI/PNNL team has outlined an approach for the unified specification of PLC firmware correctness and runtime monitoring. The proposed solution can address multiple threat models and when deployed in hardware, provide microsecond response times to errant system behavior. The team's development flow and proposed translation steps minimize the formal methods knowledge needed by system specification and development engineers. The team has created a viable co-simulation environment to test the approach as the pieces are fully developed and to experiment with different threat models and response specifications.

The team looks forward to engaging in future collaborative work to address the needs of national energy security requirements.

5.0 References

[1] T. R. Alves, M. Buratto, F. M. de Souza and T. V. Rodrigues, "OpenPLC: An open source alternative to automation," *IEEE Global Humanitarian Technology Conference (GHTC 2014)*, San Jose, CA, USA, 2014, pp. 585-589, doi: 10.1109/GHTC.2014.6970342.

[2] “OpenModelica Python Interface and PySimulator — OpenModelica User’s Guide v1.25.0-dev-68-g61a4aba1a5 documentation,” *Openmodelica.org*, 2023. https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/ompython.html

[3] K. Tuszynski, J. Tuszyński, and K. Slättorp, “The Modelica Association Modelica,” 2006. Accessed: Mar. 18, 2025. [Online]. Available: https://modelica.org/events/modelica2006/Proceedings/sessions/Session3a3.pdf