Hierarchical Software Quality Assurance Deliverable 2.2.2 - Month 18

Project: 70RSAT22CB0000005
Task ID: 2.2.2
Scheduled Delivery Date: Month 18 (March 15, 2024)
Delivery Date: March 15, 2024
Owners: Idaho State University (ISU), Montana State University (MSU)
Context: HSQA Statement of Work (SoW)
Research Management Plan (RMP)

Task

TASK TWO. Measure Source Code Quality and Maturity of Industrial Control Systems (ICS) and Cloud-Based Software

The contractor shall generate models to measure the Total Quality Index (TQI) value of software in ICS and cloud-based software. The models conduct analysis between phases of the Software Development Life Cycle (SDLC) and shall identify vulnerabilities before they cascade into the next phase of the SDLC. Models shall be tailored to specific transition activities within the SDLC to improve vulnerability and threat identification. The contractor shall select technologies for evaluation from the microservices domain with input from DHS S&T.

Activities to assess ICS code quality shall include exploratory research on methods for adopting secure and safe Programmable Logic Controller (PLC) programming, development of a taxonomy of PLC programming weaknesses, obtaining an appropriate body of PLC logic from real-world deployment for analysis and comparison with the taxonomy, and exploration of automated tools to detect PLC programming weaknesses.

Subtask 2.2.2: PLC Programming Weaknesses Taxonomy

As an output of initial investigation, research, and evaluation for the development of a hierarchical model for PLC programming, the contractor shall develop a taxonomy of PLC programming weaknesses. The contractor shall update the taxonomy over the period of performance to incorporate new and revised data and findings, as appropriate. The initial draft taxonomy shall be developed by Month 9 ACA. The contractor shall provide an updated taxonomy by Month 18 ACA, and a subsequent update by Month 27 ACA. The final taxonomy shall be delivered by Month 35 ACA.

Investigation

Background

At the completion of Month 9, we stated, “We anticipate extending and refining the Valentine taxonomy. This will first involve:

- Comparing it with the concepts described in the Top 20 Secure PLC Programming Practices list described in the Task 2.2.1 Investigation Report.
- Investigating existing tools for analyzing user-developed PLC logic and incorporating tool output or tool organizing concepts into the taxonomy.”

Methodology

Use cases

In refining the taxonomy, our initial focus was on aligning it with the project's objectives. We identified five distinct use cases for the PLC code scanning tool:

Table 1. Use cases for envisioned PLC Code Scanning Tool

Case	User	Use
1	PLC Programmers	Check own code for errors
2	Engineering Managers	Review the work of code developers or processes associated with development of PLCs
3	Education/Training Providers	Quickly highlight poor programming practice for students
4	PLC Programmers	Identify where PLC code could be attacked to cause physical damage
5	Incident Responders	Identify where/how attackers may have manipulated PLC code to cause physical damage

Use cases 1-3 are preventative in nature; use case 4 is offensive; and, use case 5 is reactive.

For the preventative use cases (1-3), a potential PLC code scanning tool that feeds into a PIQUE model that operationalizes the taxonomy, would be of greatest value in attack scenarios where the adversary does not have the capability (i.e., access, expertise, tools, and time) to modify PLC code.

From a preventative perspective, a tool would be of use assuming that the adversary has access to the SCADA, the local HMI, or transmitters, but not access to the engineering workstation or some other tool with the capability of examining and overwriting the PLC code.

Use case 4 would be helpful for PLC programmers to recognize that adversaries could use similar tools to plan attacks against their code to cause specific kinetic effects. This would help PLC programmers to recognize why they need to consider safe/secure PLC programming practices.

Use case 5 could help discover/identify/show how the code actually facilitated physical damage in the aftermath of an attack that involved malicious PLC code.

Taxonomy Organizing Principle

A key observation from the use case analysis exercise is that *the PLC code scanning tool will be most useful if it can link code structure to the possibility of specific kinetic effects*. This is what makes PLC code different from typical computer code.

In revisiting the Valentine Taxonomy [1], we recognized that it is not organized around the concept of specific kinetic effect (process context). Instead, it is organized by types of errors and locations of errors within the programming context.

We hypothesize that it is possible to connect the programming context to the process context in order to assess the possibility of causing a specific kinetic effect via cyber attack (process context). This hypothesis now forms a central thrust of our effort.

Safe/Secure PLC Programming Practices

To investigate our hypothesis, we turned to the “Top 20 Secure PLC Programming Practices” [2]. This document, released in 2021, advances 20 things PLC programmers can do to enhance the security/safety of their code. Each practice includes a title, a short description, guidance on how to implement the practice, an example of implementation, a statement of rationale, and informative references.

The document provided inspiration for the Safety portion of the 2.2.1 PLC code software hierarchy deliverable at month 9.

Our initial observation was that the Top 20 Practices paper does not have an organizing principle: the practices are not presented alphabetically, by priority, or by logical grouping. We reviewed the practices to determine whether 1) they could be placed into intuitive categories, and 2) they could be observed/identified in PLC code per se.

Our analysis is found in Table 2, below. The column on the left presents five characteristics of the safety/security attribute. The second column displays which of the top 20 PLC secure programming practices are included within each characteristic, and the column on the right represents our initial, non-conclusive, assessment of whether the corresponding top 20 practices can be identified by examining the PLC code file alone. The characters “N/I” in the middle column indicates that the characteristic is not included in the Top 20 practices. In the right-most column, “Y” represents yes, “M” represents maybe,

Table 2. Proposed characteristics of "Safety" attribute with alignment to Top 20 secure PLC programming practices.

Category: Safety/Security		*Top 20*	*Perceivable in code?*
1	Data Validity	12,9,6	Y,M,M
2	Process State Awareness	7,15,20	M,Y,Y
3	Exclusivity	7,15,20	M,Y,Y
4	Configuration Awareness	5,16,17,18,19	M,Y,Y,M,Y
5	Conservativity	N/I	M

Perceiving Safety/Security PLC Coding Practices

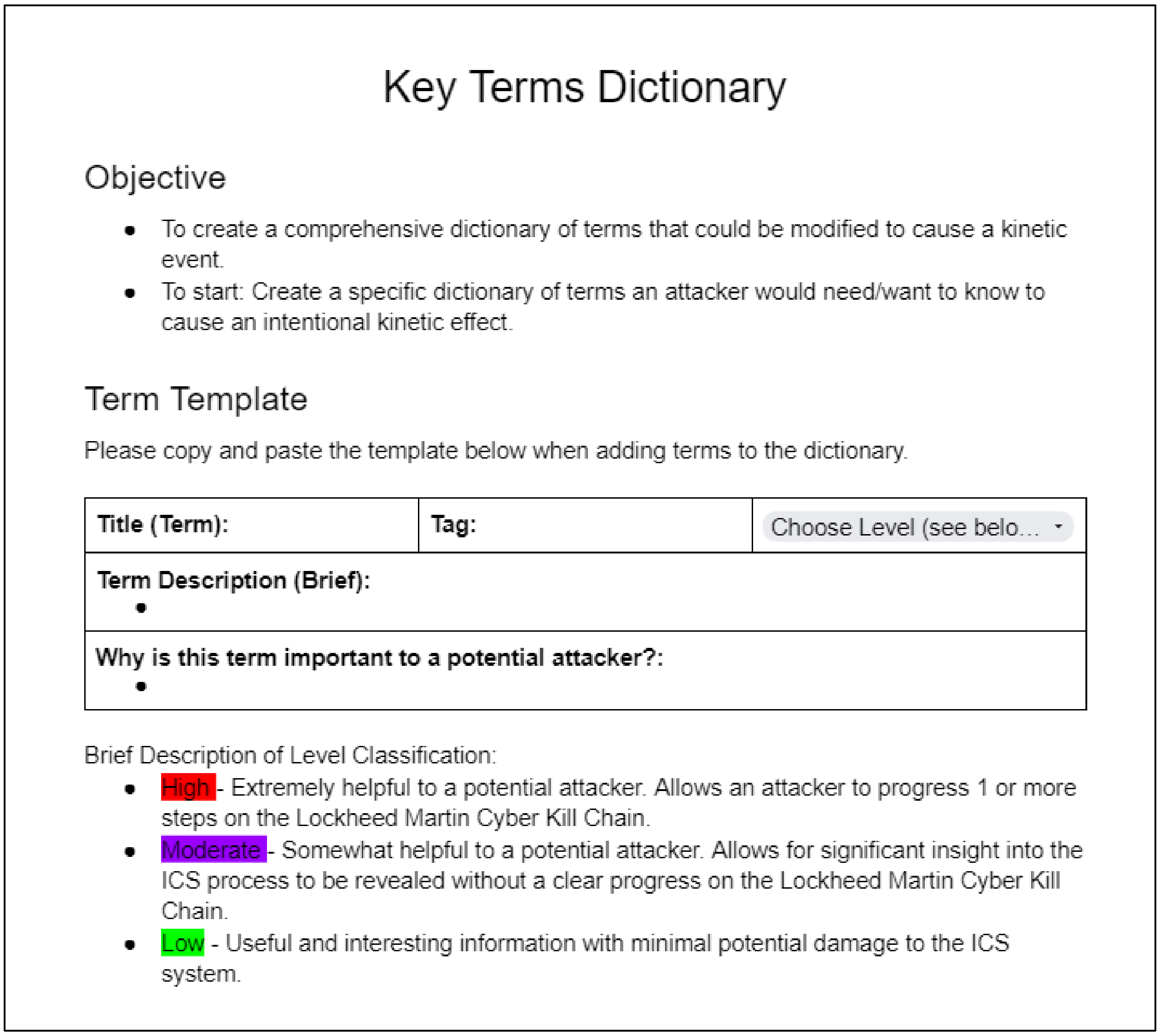
To further investigate whether PLC coding practices relative to Safety/Security can be perceived in code, we identified two approaches:

1. Review PLC code for comments or variable names that an attacker would need/want to know to cause an intentional physical, kinetic effect.

2. Conduct experiments ranging from simplistic to more complex about whether failure to implement a given secure programming practice actually would cause physical damage.

PLC Code Review

Our initial concept was to create a dictionary of key terms that could be found within the code to give clues about how the code could be modified to cause physical effects. Figure 1 below displays the proposed dictionary. The dictionary currently includes 19 terms.An analysis engine could then be created to help identify and prioritize high sensitivity code areas. The dictionary and the analysis engine combine to form what we have termed a “Kinetic Effect Navigator Tool”. Figure 2 presents this concept. Figure 3 shows a first draft wire frame for the tool.



Key Terms Dictionary

Objective

- To create a comprehensive dictionary of terms that could be modified to cause a kinetic event.
- To start: Create a specific dictionary of terms an attacker would need/want to know to cause an intentional kinetic effect.

Term Template

Please copy and paste the template below when adding terms to the dictionary.

Title (Term):	**Tag:**	Choose Level (see belo…
Term Description (Brief): •		
Why is this term important to a potential attacker?: •		

Brief Description of Level Classification:

- High - Extremely helpful to a potential attacker. Allows an attacker to progress 1 or more steps on the Lockheed Martin Cyber Kill Chain.
- Moderate - Somewhat helpful to a potential attacker. Allows for significant insight into the ICS process to be revealed without a clear progress on the Lockheed Martin Cyber Kill Chain.
- Low - Useful and interesting information with minimal potential damage to the ICS system.

Figure 1. Key terms dictionary template

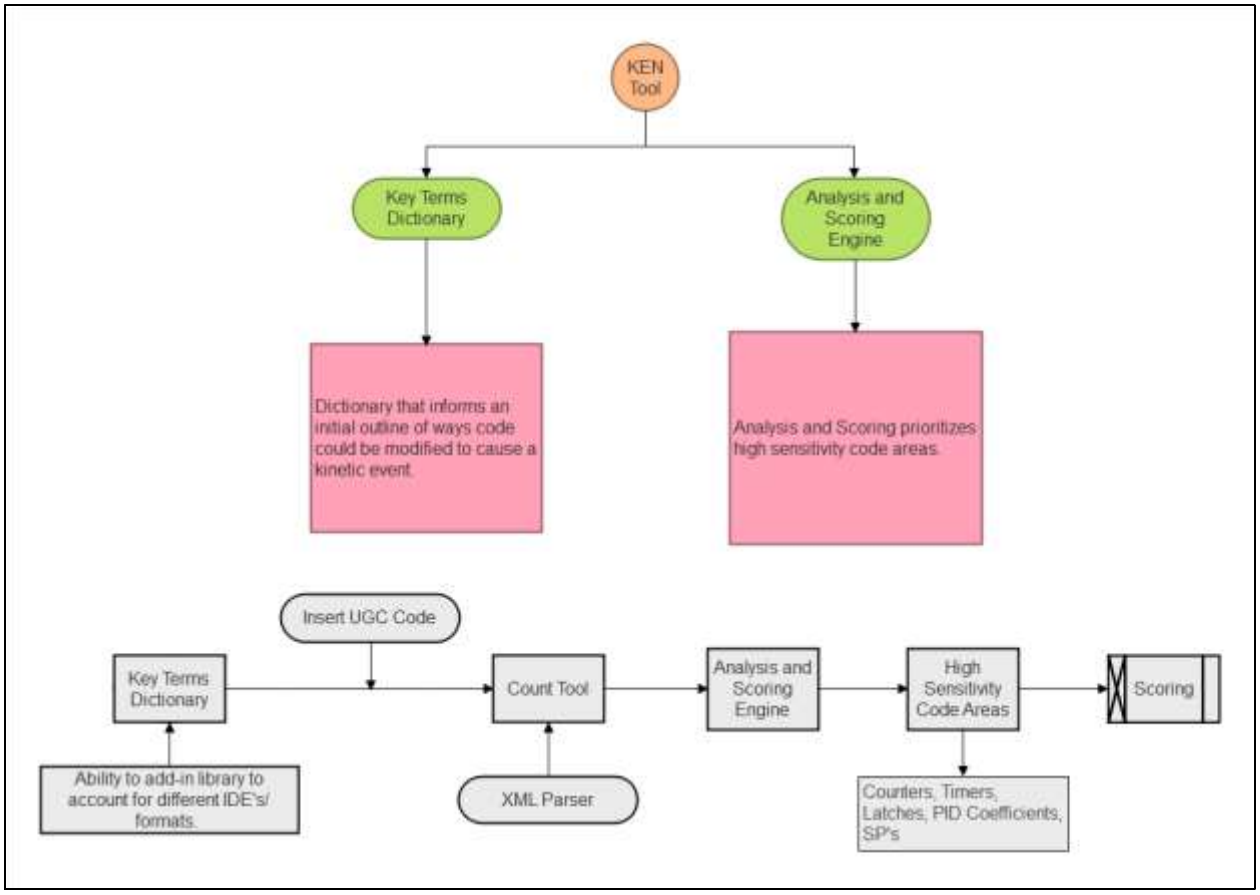

KEN Tool
Key Terms Dictionary
Analysis and Scoring Engine
Dictionary that informs an initial outline of ways code could be modified to cause a kinetic event.
Analysis and Scoring prioritizes high sensitivity code areas.
Insert UGC Code
Key Terms Dictionary
Count Tool
Analysis and Scoring Engine
High Sensitivity Code Areas
Scoring
Ability to add-in library to account for different IDE's/ formats.
XML Parser
Counters, Timers, Latches, PID Coefficients, SP's

Figure 2. Proposed flow diagram of KEN tool.

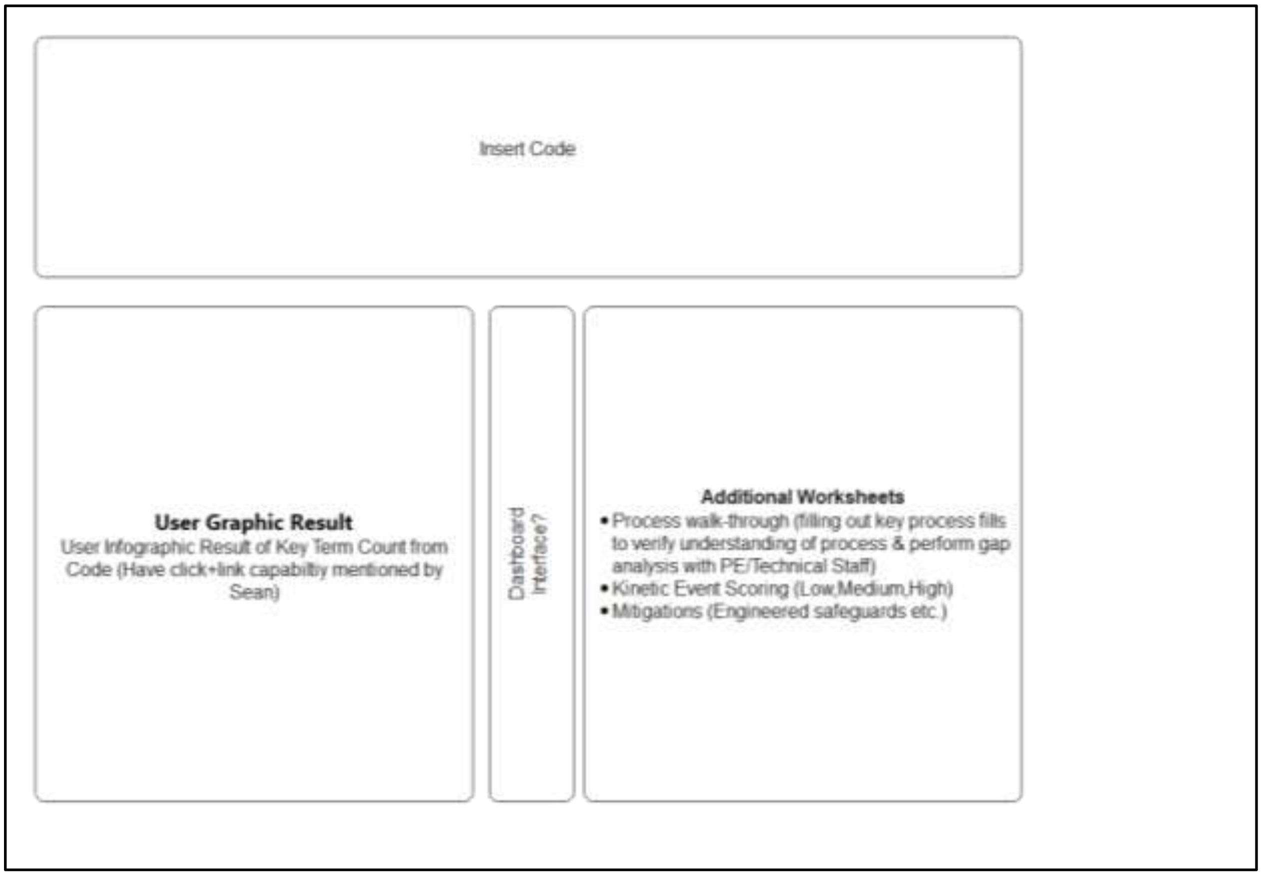

Insert Code
User Graphic Result
User Infographic Result of Key Term Count from Code (Have click+link capabiltiy mentioned by Sean)
Dashboard Interface?
Additional Worksheets
• Process walk-through (filling out key process fills to verify understanding of process & perform gap analysis with PE/Technical Staff)
• Kinetic Event Scoring (Low,Medium,High)
• Mitigations (Engineered safeguards etc.)

Figure 3. Proposed wire frame of KEN tool user interface.

Conduct experiments about whether a coding practice would actually cause physical damage

For this approach we identified a four-pronged methodology:

1) Ultra-simplistic process model
2) Relatively simplistic but real-world physical controlled process
3) A slightly more complex simulated model
4) Highly complex code without simulation of physics

For 1, and 2, we are developing real control system scenarios and examples of good and bad coding practice. The images below show PLC code (ladder diagram) for a heat exchange process in which the identified safety characteristic of Exclusivity is included, and PLC code in which Exclusivity is not included.

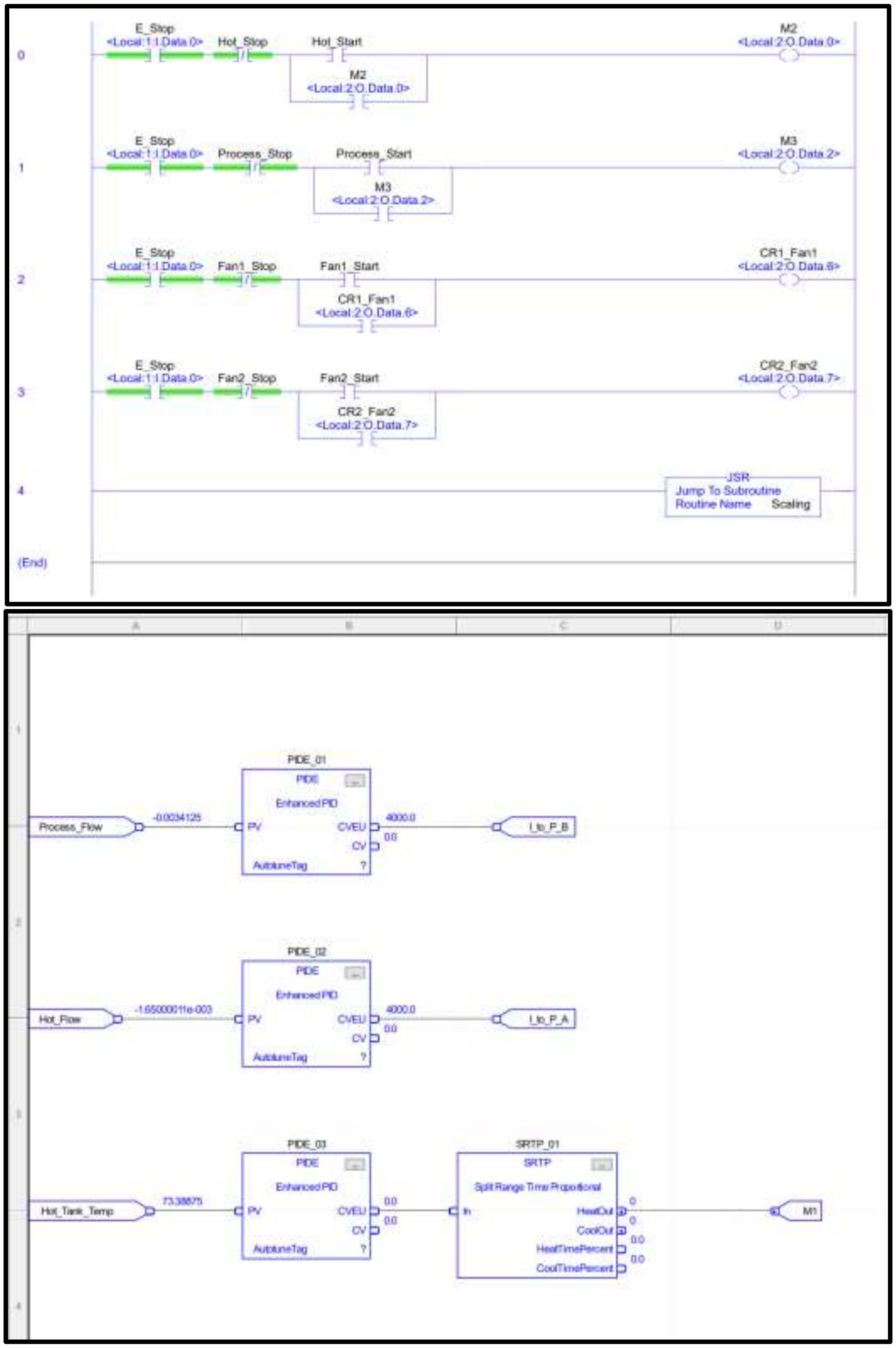
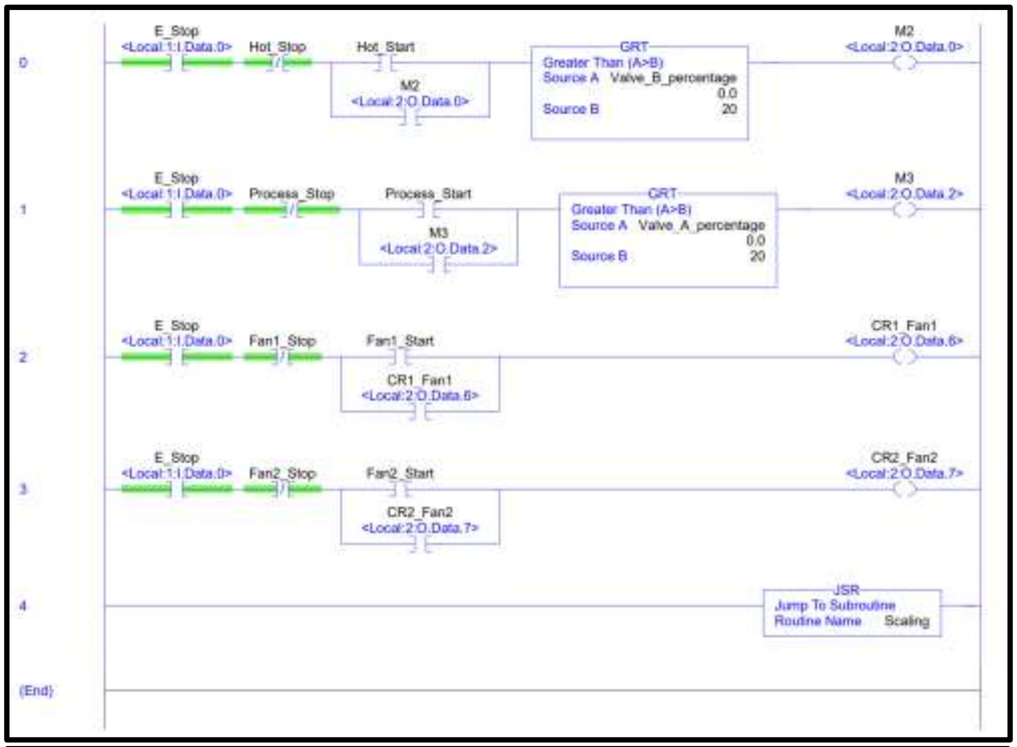


Figure 4. Code without Exclusivity characteristic allows pumps (M2 and M3) to be on and the corresponding valves to be closed, damaging the pumps.



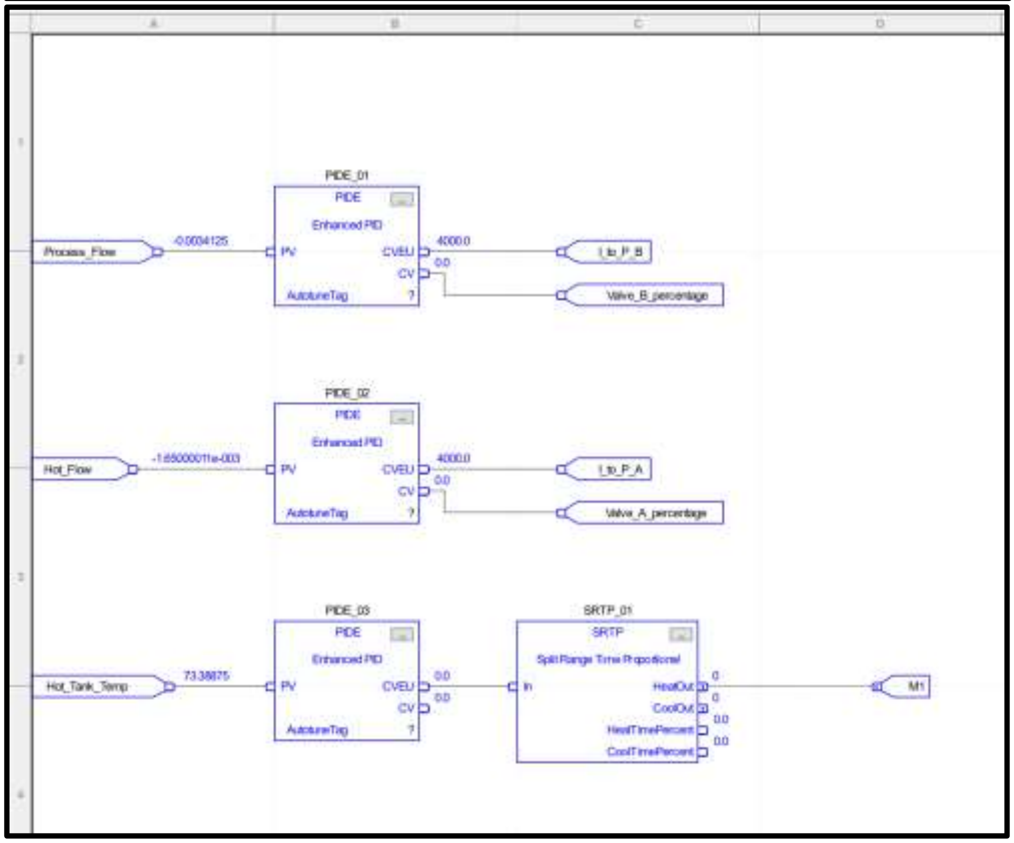
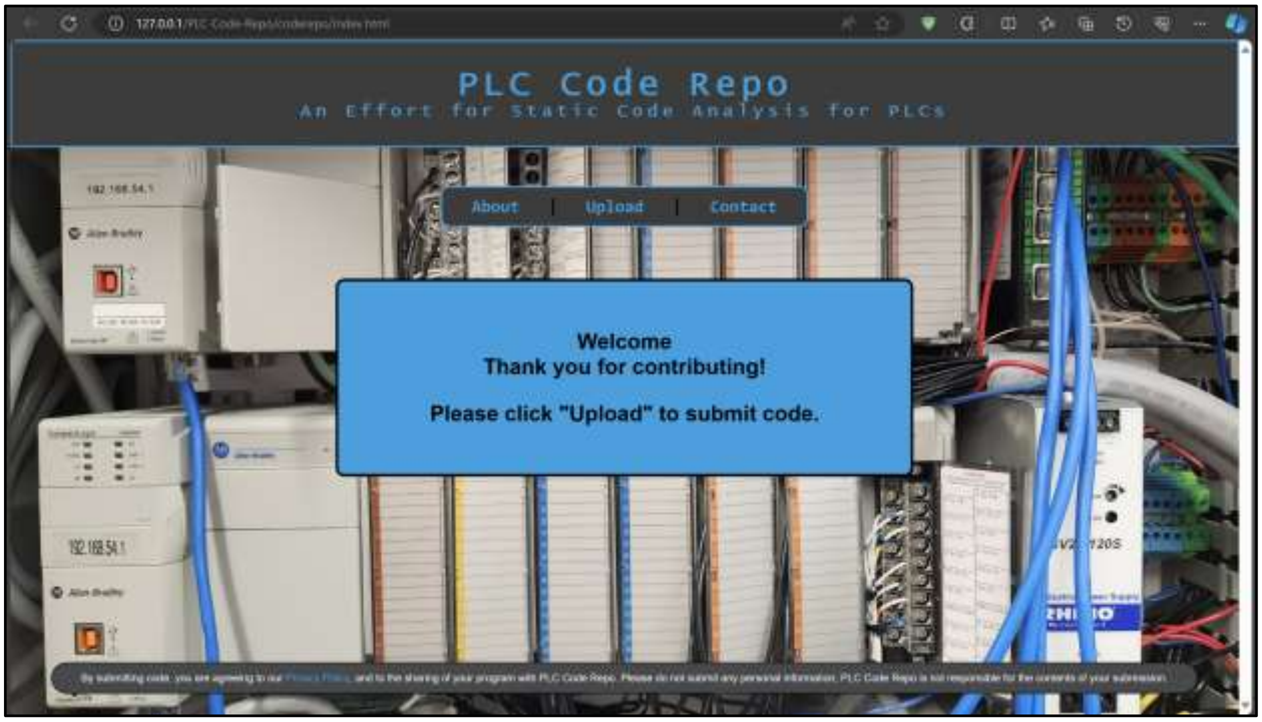


Figure 5. Code with Exclusivity characteristic requires valve to be open in order for pumps (M2 and M3) to be on.

To conduct approach 3, we are exploring several process and PLC simulation technologies, including CoDeSys Simulator, MapleSim, FactoryIO, Famic AutomationStudio, and Unity. The key consideration is whether the simulation software can simulate not just the PLC program, but the process physics.

To conduct approach 4, we are building a web-based PLC code repository and a web-based application to collect submissions of PLC code for analysis. See the following screen snippet of the submission page.


127.0.0.1/PLC-Code-Repo/coderepo/index.html
PLC Code Repo
An Effort for Static Code Analysis for PLCs
About
Upload
Contact

Welcome
Thank you for contributing!

Please click "Upload" to submit code.

By submitting code, you are agreeing to our Privacy Policy, and to the sharing of your program with PLC Code Repo. Please do not submit any personal information. PLC Code Repo is not responsible for the contents of your submission.

Figure 6. Web page for submitting PLC code.

Comparison of Static Analysis Tools

In addition to aligning the taxonomy with use cases and categorizing secure coding practices, we set out to identify and explore the capabilities of static analysis tools for user-generated PLC code. We reviewed the following tools:

Table 3. Comparison of PLC code static analysis tools.

Tool	*Support*	*Ease of use*	*Code Quality Metrics*	*Cost*	*Languages*	*Interface*
Arcade.PLC	Poor	Medium	Not supported	Subscription	61311-3 structured text	Web app (code submitted to another

						server)
EcoStruxture (Control Engineering - Verification)	?	No response	Supported. Safety not included	?	61131-3 from various leading vendors	Web Browser interface
EcoStruxture (Machine Advisor)	End of Support	No response	End of Support	?	?	?
attkfinder	Open-source project	Hard to configure use	Attack map, but not traditional code quality metrics	Free	61131-3 XML or structured text	Text-based
CoDeSys Static Analysis Tool	High	Easy to use	Traditional code quality metrics. Safety not included	660 USD/yr	61131-3 structured text	Graphical – embedded in IDE

Of these tools, the CoDeSys Static Analysis Professional Developer tool was the simplest to obtain, and produced appropriate code quality metrics. We procured the tool and used it as the baseline for subtask 2.2.3.

Results

In the past 9 months the MSU/ISU team has made the following refinements to the PLC code security taxonomy:

- Created use cases for the eventual creation of a PLC code scanning tool
- Adopted safety/security as the key organizing principle of the PLC secure code research effort
- Proposed alignment of the taxonomy with Top 20 PLC Secure Coding Practices
- Advanced experimental designs for perceiving safety/security practices within the PLC code
- Compared PLC code static analysis tools

Next steps

Over the next periods of performance we anticipate advancing by:

1. Creating additional examples of code that does and does not follow secure coding practices
2. Developing heuristics (such as pattern matching) for identifying the presence or absence of secure coding practices
3. Developing a stand-alone code PLC safety scanning tool for 61131-3 languages that implements the heuristics (from 2) and produces associated metrics
4. Creating examples of malicious PLC code (code known to cause physical consequences) and testing whether the PLC code scanning tool can identify it
5. Expanding the number of PLC programs within the repository
6. Testing the performance of the PLC code scanning tool with reliance on simulated physics of more-complex industrial processes
7. Creating an open-source PLC code scanning tool for IEC 61131-3 languages that covers a range of code quality measures beyond safety
8. Advancing the functionality of the KEN tool for IEC 61131-3 compliant XML

Risks

Risks are organized by next step number as identified in the previous section.

Table 4. Risks by proposed next steps

Step Number	Risk Description	Estimated Likelihood
1	None identified.	
2	Researchers get stuck on developing heuristics and pattern matching. *Impact:* Medium *Possible causes:* Difficult for all IEC 61131-3 languages *Response:* Focus on single language, Focus on KEN Tool Impact of this risk would flow to other steps.	20%
3	Researchers cannot implement heuristics or patterns identified in 2 in code as an automated technique. *Impact:* Medium *Possible causes:* Heuristic for LD does not translate to programming skills of team	30%

	Response: Focus on structured text. Release heuristic as "manual check" advise	
4	Validation tests cannot reliably identify known unsafe code. *Impact:* Medium *Possible causes:* Heuristic only applies to edge cases *Response:* Note potential weaknesses/inaccuracies in release notes/report. Focus on KEN Tool	10%
5	Researchers fail to convince individuals to submit code. *Impact:* Medium *Possible causes:* Researchers fail to reach the right audience. PLC programmers do not trust the effort. PLC programmers perceive this is giving away intellectual property *Response*: Focus on key relationships. Refine the message.	40%
6.	Researchers cannot reliably simulate more-complex processes *Impact:* Low *Possible causes:* Learning curve for simulation software is too high. Simulation software cannot incorporate accurate physics of the entire process. *Response:* Attempt to engage true experts in modeling. Rely on simplified graphical simulations rather than accurate physical simulations. Focus on the other approaches.	60%
7.	Researchers cannot create their own open source tool *Impact:* Medium *Possible causes:* ISU team lacks code development capabilities. MSU team attention drawn to other tasks. XML code difficult to parse. *Response:* Focus on structured text only. Rely on CoDeSys. Ask request scripting feature be added to CoDeSys.	30%

8.	None identified.	

REFERENCES
[1] S. Valentine, “PLC Code Vulnerabilities Through SCADA Systems” https://scholarcommons.sc.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1804&context=etd
[2] Top 20 Secure PLC Programming Practices https://www.plc-security.com/content/Top_20_Secure_PLC_Coding_Practices_V1.0.pdf