On Ladder Logic Bombs in Industrial Control Systems

Naman Govil (Georgia Tech), Anand Agrawal,
Nils Ole Tippenhauer (SUTD)

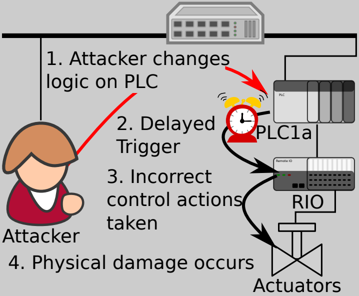
Challenge: Malicious Logic

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- We consider industrial devices with reprogramable logic
  - In this context: Programable Logic Controllers (PLC)
- Assume attacker is able to modify the logic
- Goal
  - Physical damage in future
  - Extract data, exfiltrate
  - Remain hidden from manual inspection
- How could the attacker achieve those goals?
- How could we detect such manipulations?

Use Case

- PLC controls physical process chemical pump
- Malicious contractor on site
- Able to connect to local plant or field network
  - Tools available to download, modify, upload logic
- Attacker goal: Increase chemical dosing
  - Reduce water quality, damage components
- Attacker now manipulates PLC control logic slightly, which causes triggers for safety measure to be ignored
- *How can engineers detect this change?*



Ladder Logic and PLCs

- PLCs are programmed on two different levels
  - Firmware
  - Control Logic
- Firmware is nowadays often cryptographically signed
  - For our Rockwell PLCs, we did not find direct way to manipulate
  - Related work suggests to use JTAG
- Control Logic can be uploaded by anyone
  - Device might need to be set into programming mode with physical switch
  - We argue that this will be the case in practise

Logic Flavors

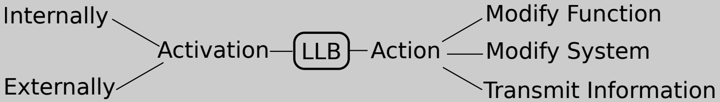
- Logic for PLCs can have different flavors according to IEC 61131-3
  - Ladder logic (visual, mimicking logic circuits)
  - Functional Block Diagrams (visual, wiring between blocks of logic)
  - *Sequential Text* defines small sequential functions (C-like)
- Overall logic somewhat similar to hardware description languages (VHDL, Verilog)
- Control logic defined based on input signals (e.g. sensor values)
- Output values can be commands for actuators, and processed sensor values

Programming a PLC

- To program a PLC, a suitable software is required
  - Typically, by the vendor of the PLC
  - Using the software, current logic code can be read from PLC over network
  - Logic can be modified in the software, and then be re-uploaded
- Uploading logic can require a switch to be enabled on PLC
  - But we observed that in practise, engineers are leaving it active
- *How to verify that correct code is running on PLC?*
  - Manual inspection of logic by engineers?

Ladder Logic Bombs

- We call malicious logic hidden in PLC logic *ladder logic bombs*
- Classification based on *activation* and *action* of LLB



Triggering LLBs

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Externally

- Trigger based on particular single input
  - Trigger could be directly sent by attacker, or could occur naturally
- Triggering based on particular input sequence
  - To make detection and accidental triggering unlikely, a specific sequence of inputs could be required

Internally

- Triggering based on Timer
  - For example, to ensure that attacker can leave premises before payload is deployed
- Triggering based on specific internal condition
  - For example, if error conditions or states are caused

Payload of LLBs

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- Modify Function
  - Change existing control logic, e.g. thresholds, invert control signals
  - Denial of Service
- Modify System
  - For example, modify local time
- Transmit Information
  - Exfiltrate control states, current state of sensors, etc.
  - Potentially hidden, e.g. using steganography in sensor values

Hiding the LLBs

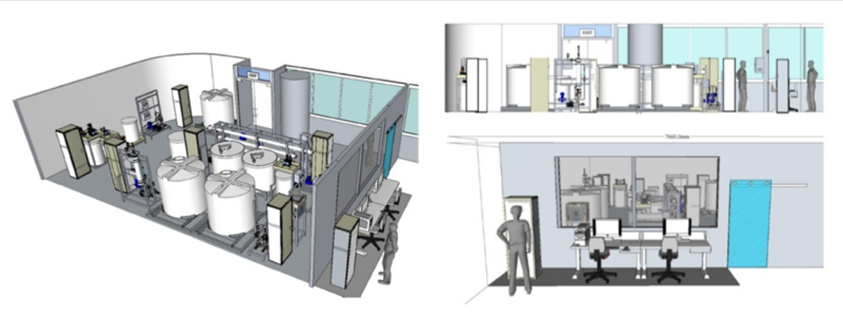
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- Assuming manual inspection is used, hiding the LLB can be achieve in different ways
- In Ladder Logic, additional functional blocks can be added
  - Visually close to common blocks (I/O pins, naming)
  - Inside the block, malicious code is hidden
  - Malicious code is limited to signals on I/O pins
- Similar attacks are possible for sequential text and functional blocks
- To test difficulty of creating and finding LLBs, we implemented a number of prototypes

The Secure Water Treatment testbed

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- Testbed designed for security research & education
- Full system with physical process, control, SCADA
- Overall system cost: > 750k SGD



SWaT planning stage rendering. Source: iTrust

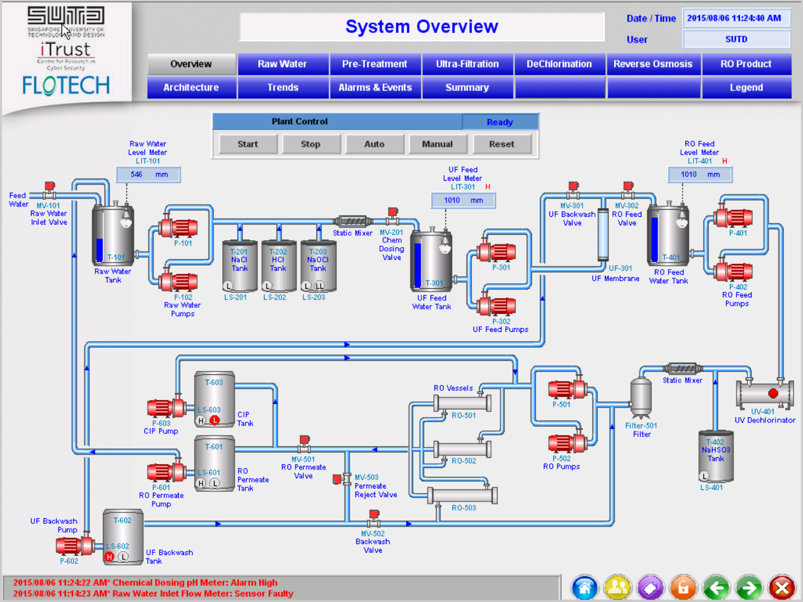
The Secure Water Treatment testbed II

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- Started operations in March'15
- Used by about ~15 researchers, guests are welcome!

SWaT view on ultra-filtration (left), reverse osmosis process (right)

The Secure Water Treatment testbed III

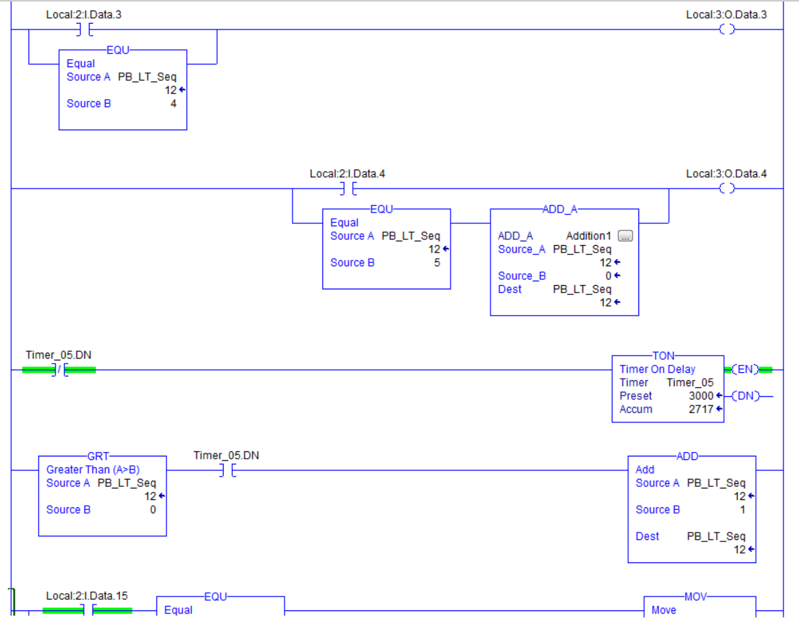


SWaT HMI page

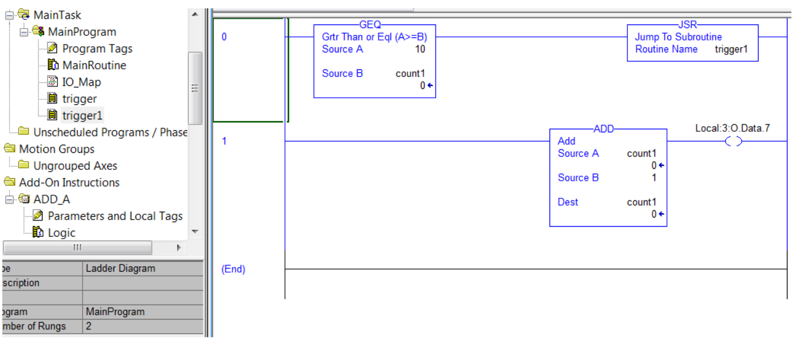
Implementation

- We implemented four main LLBs
- DoS using Add On Instructions
  - An infinite loop can be triggered remote
- Manipulation of Sensor data using Subroutines
  - Constant or varying offsets are applied to sensor inputs
- Data Logging using FFLs
  - Data is stored on SD card in PLC
- Trigger Major Faults on PLC
  - Trigger fault such as out-of-bound array access, shutdown PLC

Example: Malicious Add-On



Example: Stack Overflow



trigger1 is a subroutine which recursively calls itself, leading to stack overflow

Stealthiness

- *How can we measure the stealthiness of a LLB?*
- Metric could be Booleans (any change/no change)
- Stealthiness relates to difficulty to detect manually
- We approximate this with RALOC and memory increase
  - *Relative additional lines of code* (RALOC): how much code did we have to add to original program?
  - How big was the memory footprint increase due to the LLB?
- We found that our demo exploits increase RALOC and code by at most 4.09%

Table 1. Comparison of Attacks Performed

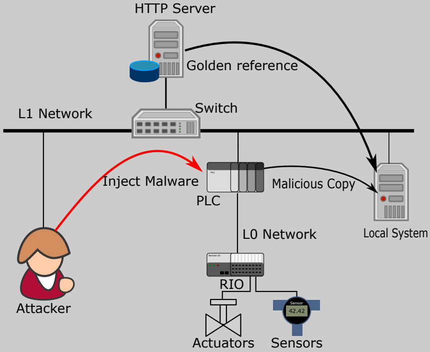
Attack	Increase in Memory(%)
Attack 1: DoS using AOI	2.60
Attack 2: Manipulate Sensor	3.84
Attack 3: Data Logging	3.41
Attack 4: Major Faults	4.09

Countermeasures

- Goal: Detect manipulations of logic of PLCs
  - Possibly automatically reverse the changes
- Legacy-compliant
  - Without changing PLCs
- Cross-vendor?
- If changes to PLCs are allowed
  - Introduce authentication/authorization
  - Trusted execution environments/ TPMs to monitor logic

Proposed system

- Our proposed solution is a *Centralized Logic Store* (CLS)
  - Holds *golden reference* of current code on any PLC
  - Modifying *golden reference* require authentication
  - Engineers can query CLS for *known good* logic
  - Comparison to logic on PLC could be done manually or automatically
- We implemented a proof of concept (Python, HTTP-based)



Conclusion

- We discussed malicious code in logic of PLCs
- We found that in our case, logic was easy to manipulate
  - While firmware was protected by signatures
- Industrial software did not support detection of malicious changes well
  - Manual inspection of logic would be required to determine if changes were made
- We classified, proposed, implemented a number of LLBs
  - Different triggers and payloads
  - Minimal overhead, hard to detect by humans
- We proposed a complementary system of server and client that allows to store and compare reference smaple of logic

Conclusion

- We discussed malicious code in logic of PLCs
- We found that in our case, logic was easy to manipulate
  - While firmware was protected by signatures
- Industrial software did not support detection of malicious changes well
  - Manual inspection of logic would be required to determine if changes were made
- We classified, proposed, implemented a number of LLBs
  - Different triggers and payloads
  - Minimal overhead, hard to detect by humans
- We proposed a complementary system of server and client that allows to store and compare reference smaple of logic

Thank you for your attention! Questions?