HACKING FOR B33R
BRUCON
WWW.BRUCON.ORG

WORKSHOP

Securing Industrial Systems from the core

— *Arnaud SOULLIE (@arnaudsoullie)*

WAVESTONE

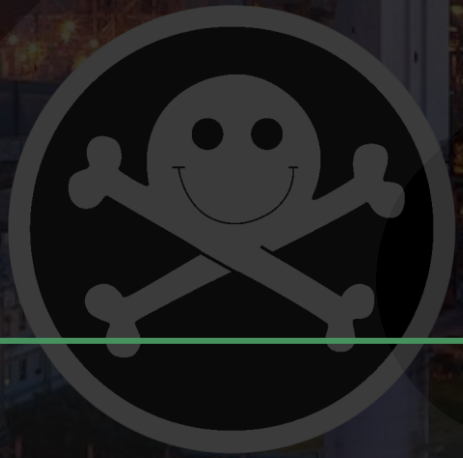
Bio

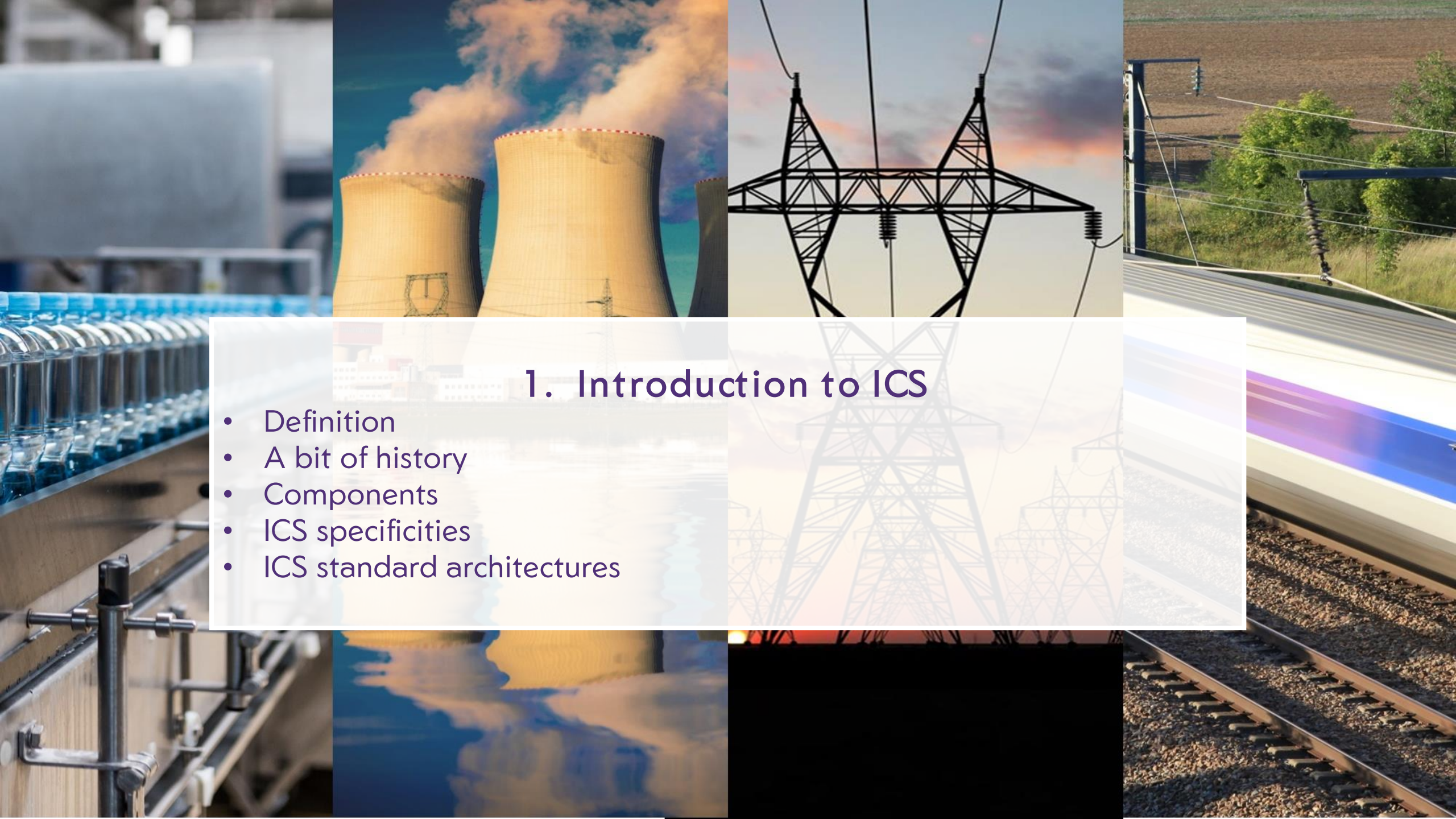
- Arnaud SOULLIÉ
- Senior Manager at WAVESTONE
- Background in pentesting since 2010, ICS in 2012
- ICS cybersecurity assessment for large clients in all verticals (transportation, agrifood, energy (up & downstream)
- Few talks/workshops on ICS pentesting at DEFCON/Brucon/CS3/..
- Creator of DYODE, an open-source data diode for ICS
- Trainer on ICS cybersecurity (Black Hat US & Asia, CS3, ...)

AGENDA FOR TODAY

- INTRODUCTION TO ICS
- PLC PROGRAMMING
- THE TOP 20 SECURE CODING PRACTICES
- CONCLUSION

INTRO TO ICS





1. Introduction to ICS

- Definition
- A bit of history
- Components
- ICS specificities
- ICS standard architectures

WHERE DO WE FIND INDUSTRIAL CONTROL SYSTEMS?

TRANSPORT

AERONAUTICS

INDUSTRY

PHARMA

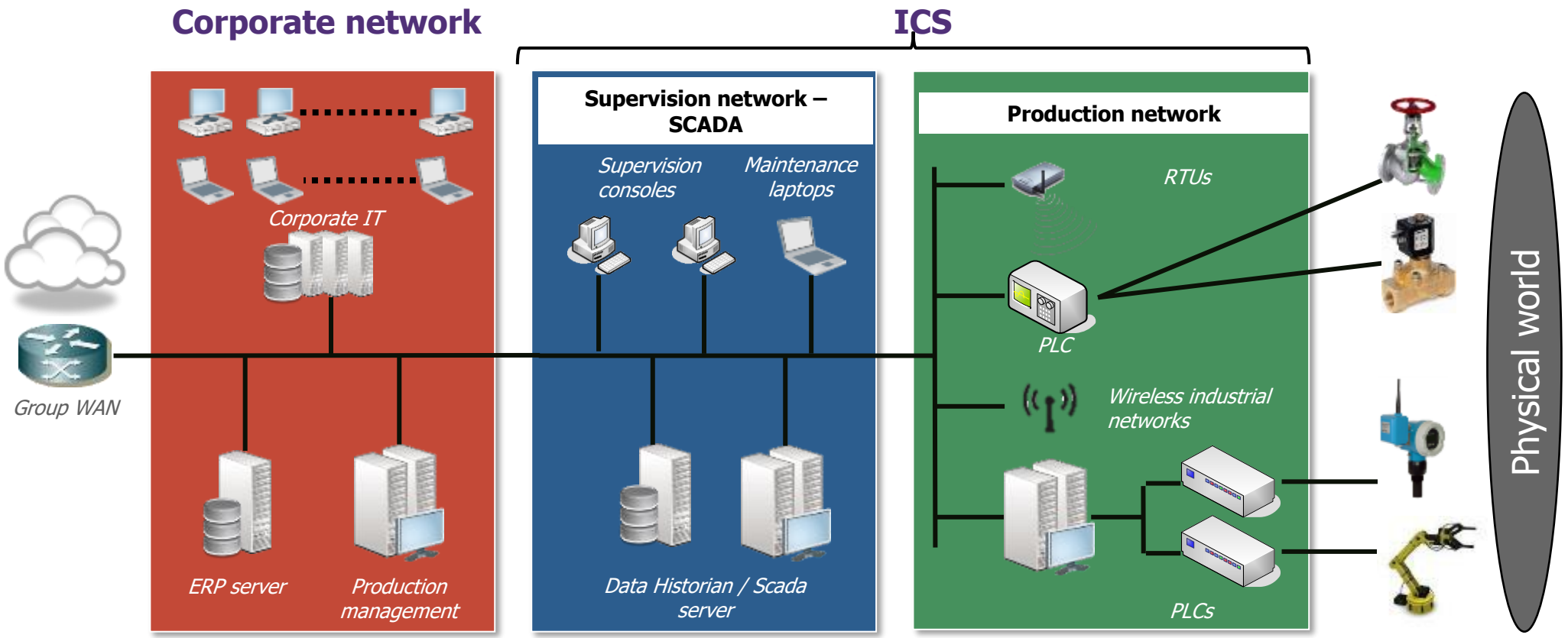
NUCLEAR

MILITARY / DEFENSE

AUTOMOBILE

AGRIFOOD

WHAT IS AN INDUSTRIAL CONTROL SYSTEM (ICS)?



Corporate IS handle data

≠

ICS handle interfaces data with physical world

A BIT OF VOCABULARY

ICS (Industrial Control System)

=

IACS (Industrial Automation and Control Systems)

~=

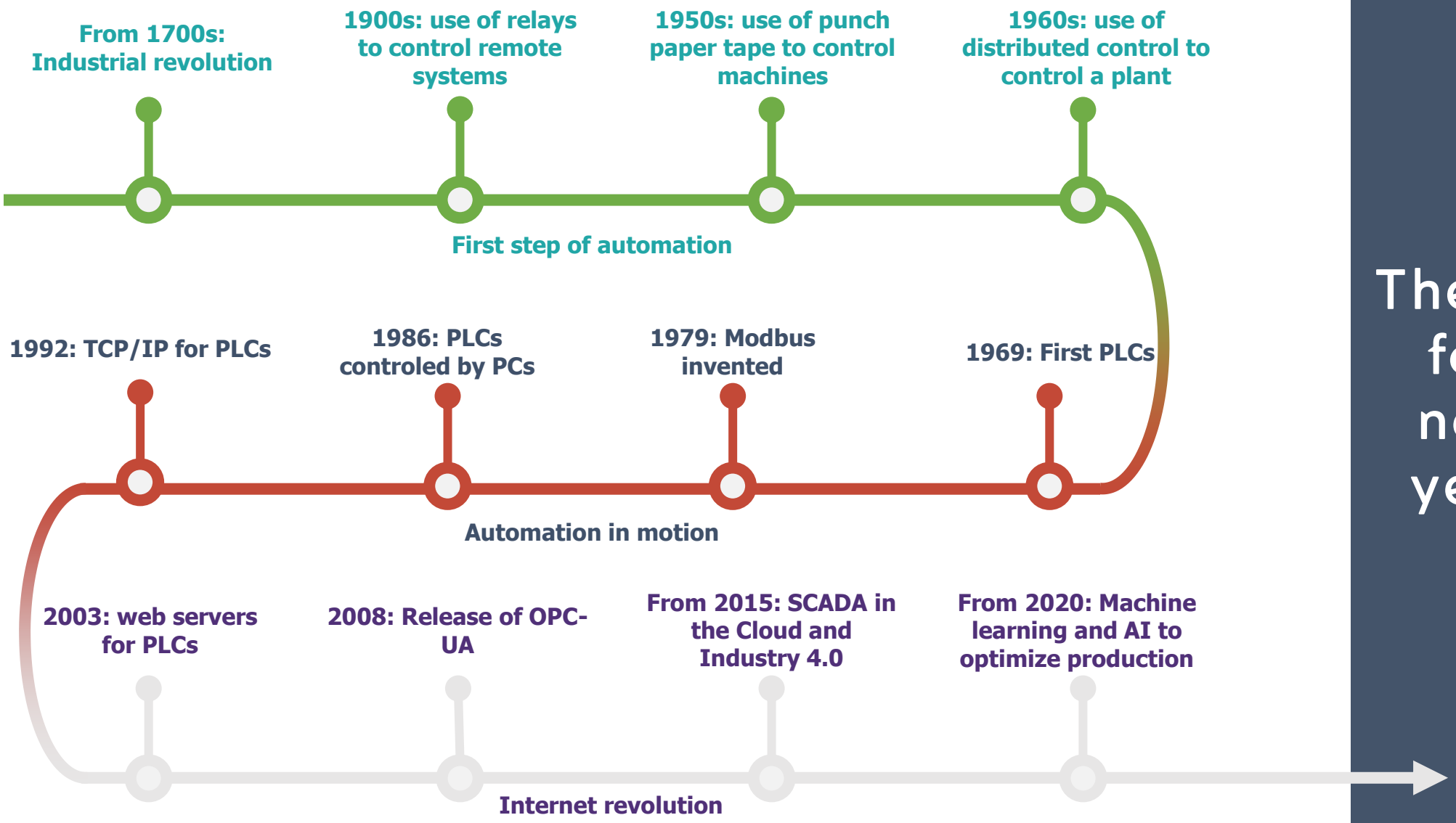
SCADA (Supervisory Control And Data Acquisition)

~=

DCS (Distributed Control System)

Nowadays, people tend to say "SCADA" for anything related to ICS

ICS EVOLUTION TIMELINE



Then what for the next 10 years ??

ICS COMPONENTS

Sensors and actuators: allow interaction with the physical world (pressure sensor, valves, motors, …)

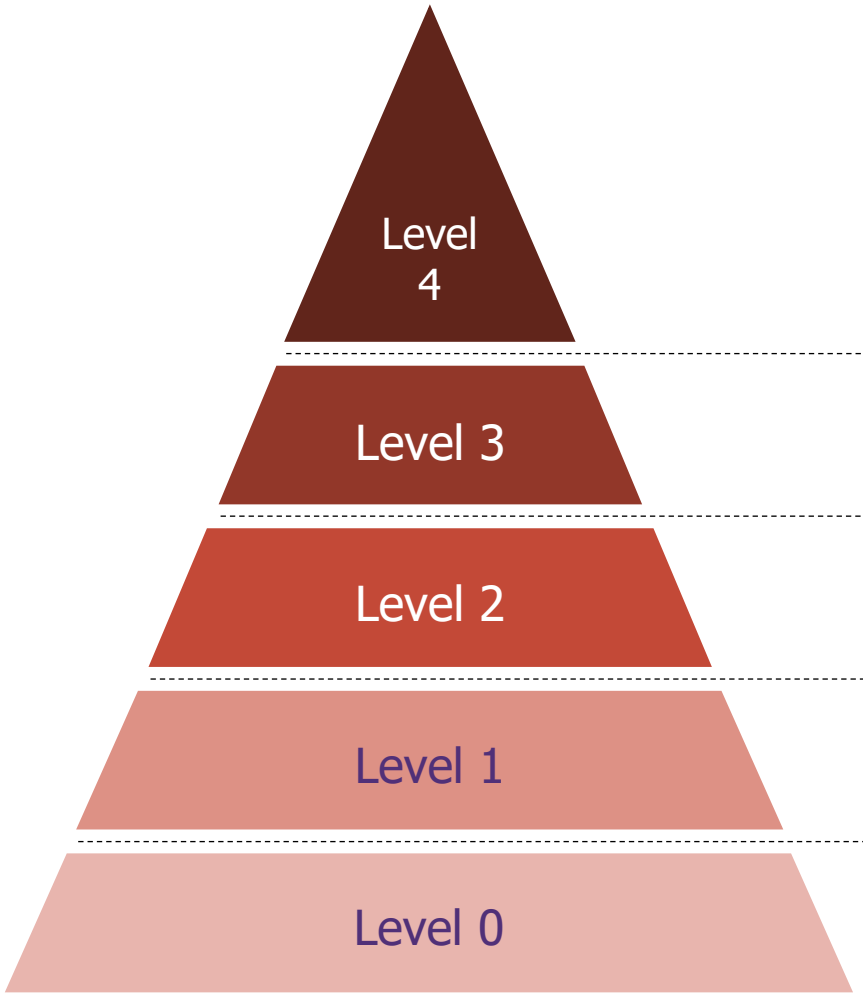
- **Local HMI**: Human-Machine Interface, permits the supervision and control of a subprocess
- **PLC (Programmable Logic Controller)**: manages the sensors and actuators

- **Supervision screen**: remote supervision of the industrial process
- **Data historian**: Records all the data from the production and Scada networks
- **MES**: Manufacturing execution system (production status, scheduling, etc.)
- **RTU**: Remote Terminal Unit (standalone PLC)
- **Other low-level devices**: Intelligent electronic devices, wireless devices, variator frequency drives, remote I/O, etc

ICS VENDORS


EMERSON
Industrial Automation
Schneider Electric
ABB
OMRON
HIRSCHMANN
Rockwell Automation
PHOENIX CONTACT
eWON
MACHINES CAN TALK
SEMAPHORE
TBOX
aspentech
SIEMENS
Honeywell
YOKOGAWA

CIM (COMPUTER INTEGRATED MANUFACTURING) PYRAMID



Global planification (ERP)
Orders and stock management, clients and accounting

Production management (MES)
Execution and control of manufacturing, scheduling

SCADA
Of an industriel process

PLCs

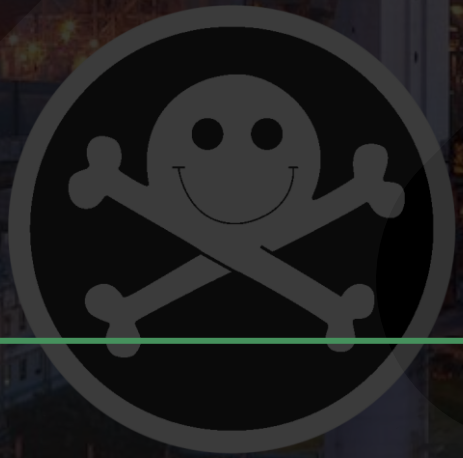
Sensors and actuators

#FOREVERDAYS

#foreverdays is a term coined by @reverseics
Very important concept when talking about ICS
The highest vulnerabilities are not patched.

So it is really worth considering the effort of patch management of ICS equipment when you know

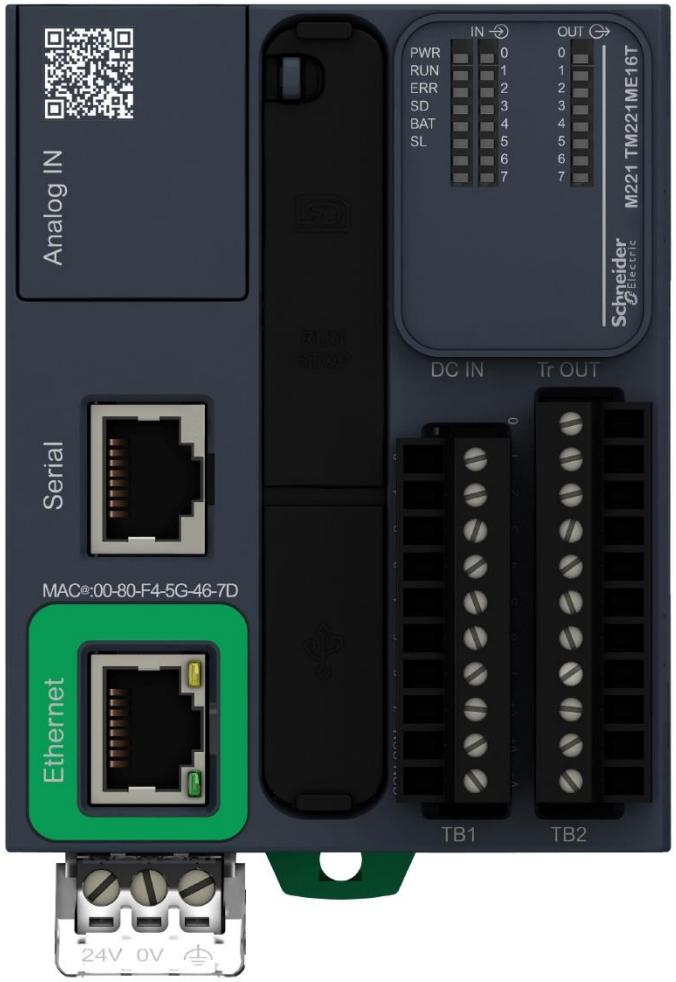
PROGRAMMING PLC



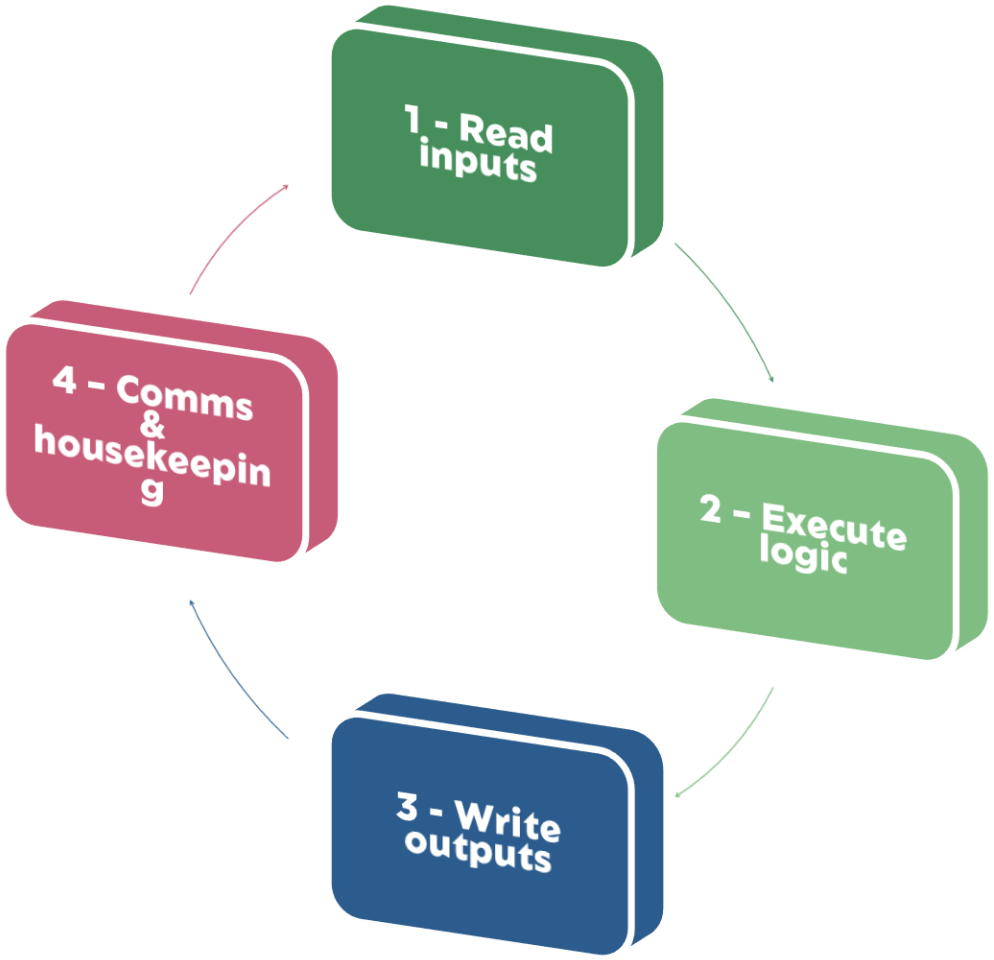
SCHNEIDER ELECTRIC TM221

- Entry-level PLC
- Based on Codesys
- The setup
  - 1 process simulation
  - 1 TM221 simulator (Schneider soMachineBasic)
  - 1 SCADA (Schneider IGSS)
  - 1 attacker machine (Kali Linux)

The setup choice is 100% based on my familiarity with the PLC, programming software and SCADA and their availability as trial versions
It is neither an endorsement nor a disapproval of these specific products



PROGRAMMABLE LOGIC CONTROLLER 101



- Real-time digital computer used for automation
- Lots of analogue or digital inputs & outputs
- Rugged devices (immune to vibration, electrical noise, temperature, dust, …)
- **Limited computational resources**

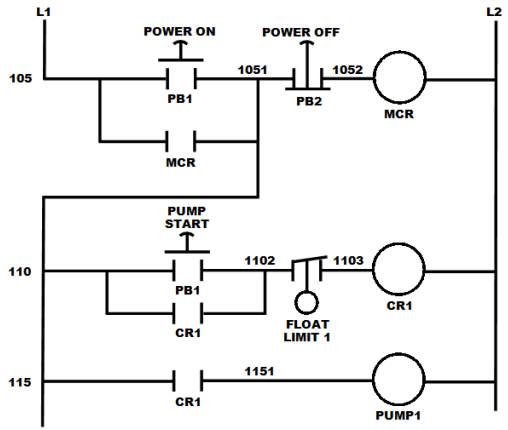
PLC PROGRAMMING

"Ladder Logic" was the first programming language for PLC, as it mimics the real-life circuits

IEC 61131-3 defines 5 programming languages for PLCs

- LD: Ladder Diagram
- FBD: Function Block Diagram
- ST: Structured Text
- IL: Instruction List
- SFC: Sequential Function Chart

Ladder diagram example



Structured text example

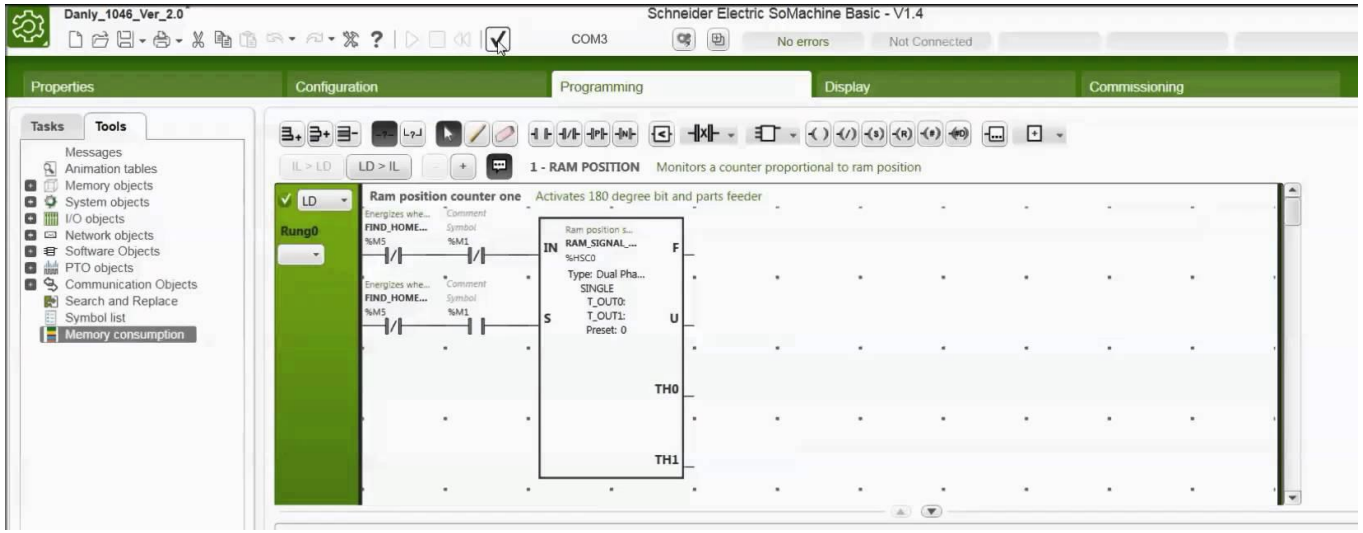
```
(* simple state machine *)
TxtState := STATES[StateMachine];

CASE StateMachine OF
   1: ClosingValve();
ELSE
    ;; BadCase();
END_CASE;
```

Instruction list example

```
LD       Speed
           GT      1000
           JMPCN   VOLTS_OK
           LD      Volts
VOLTS_OK   LD      1
           ST      %Q75
```

PROGRAMMING WITH SOMACHINEBASIC

- SoMachineBasic is the software provided by Schneider Electric to program the entry-level PLCs.
- PLCs used in big plants are usually programmed using Unity Pro, for which there is no free demo version.
- Fortunately, the way this software work is very much the same.

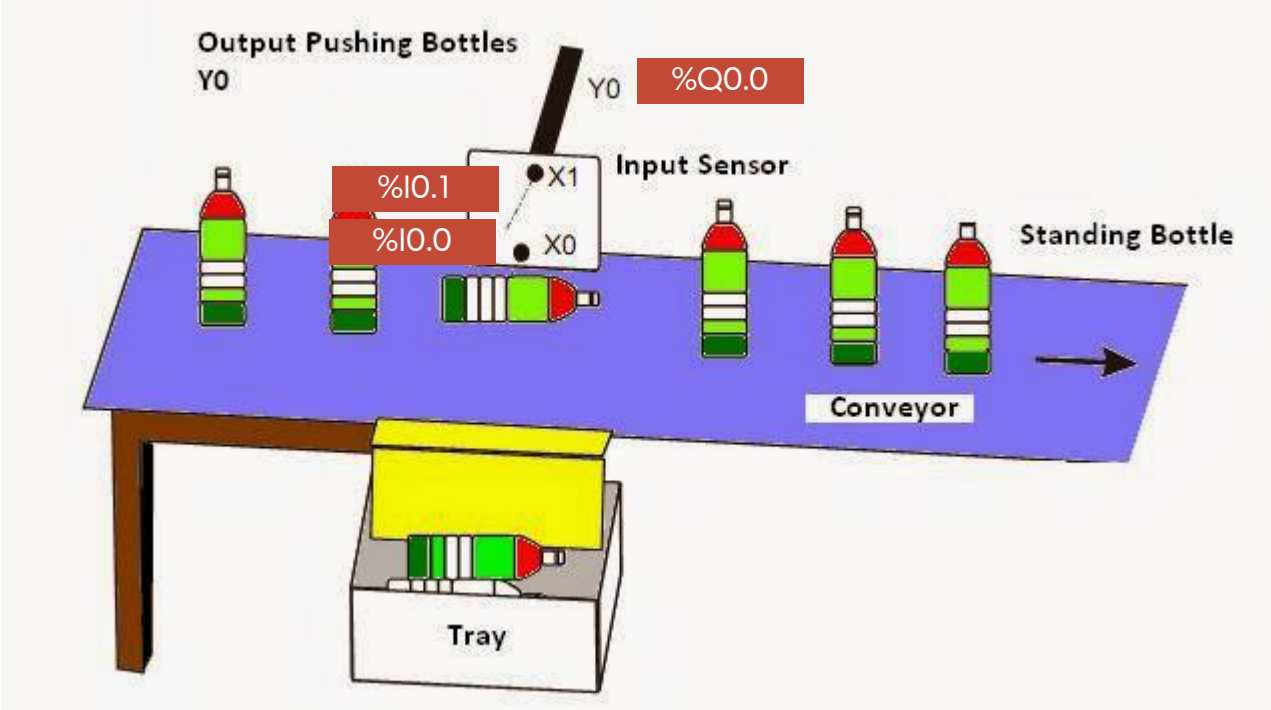


PLC programming

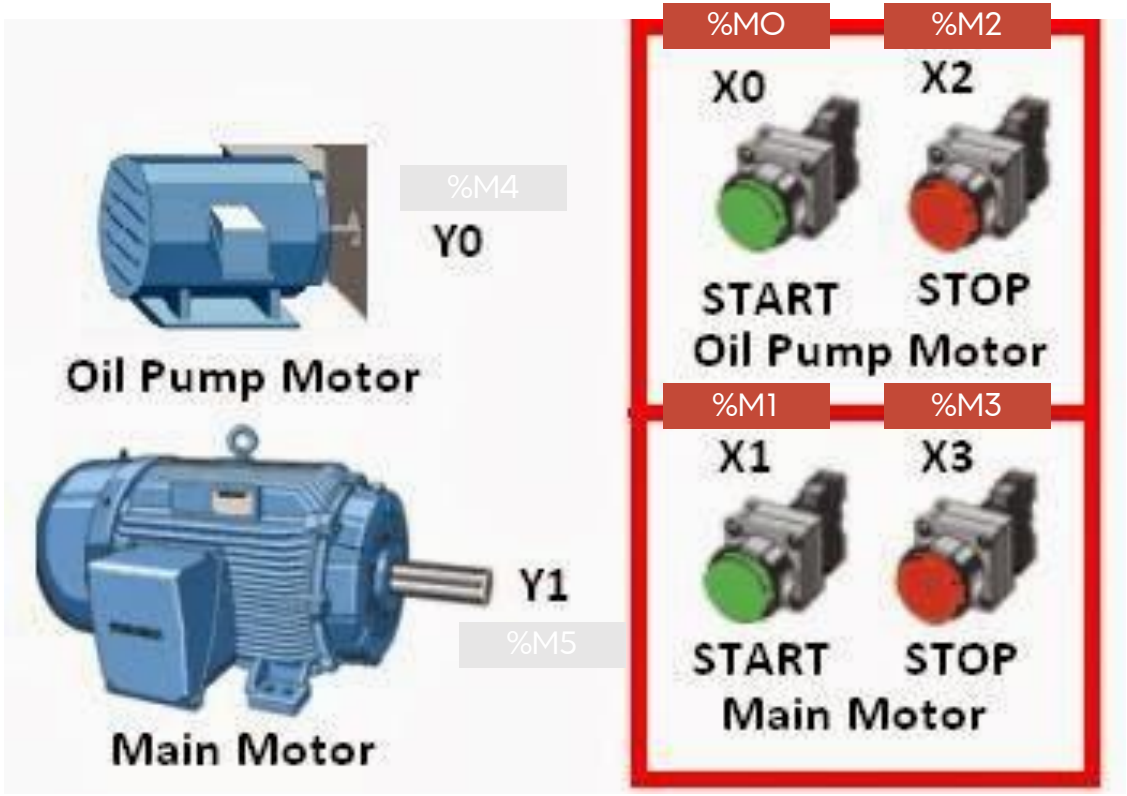
- Create a project
- Define the hardware setup
- Create variables
- Define the program
- Test
- Debug
- Push to PLC
- START

PLC PROGRAMMING EXERCISE

- Production line
- Flipped-over bottles must be put in the tray



PLC PROGRAMMING EXERCISE

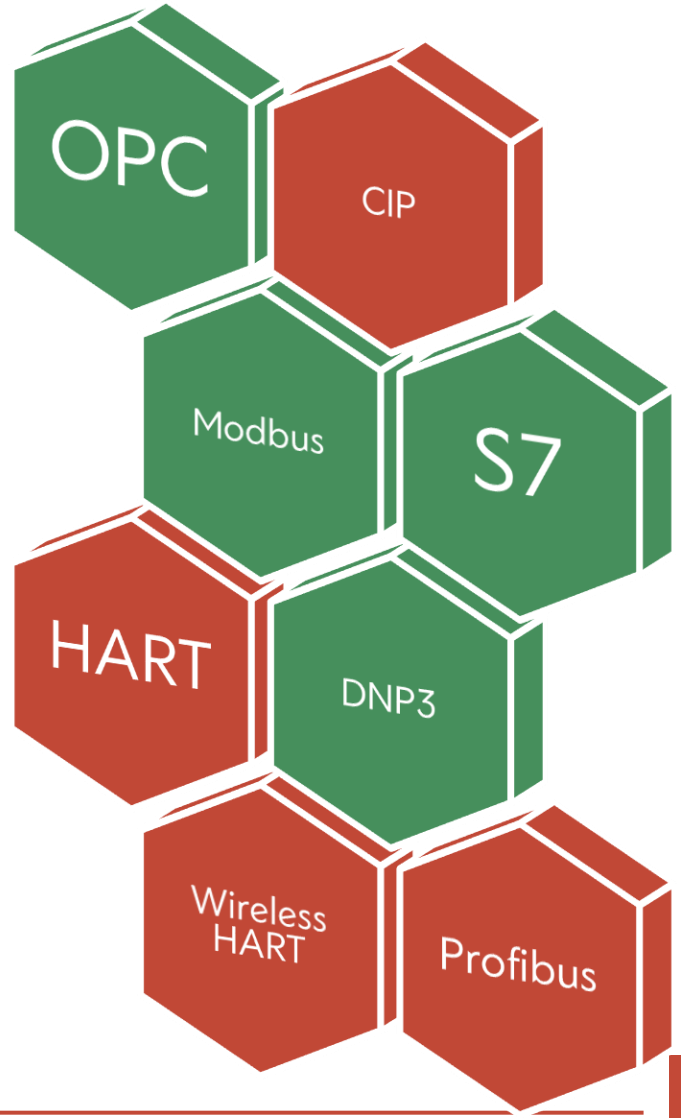


- M0 and M2 are START/STOP switches for the oil pump
- M1 and M3 are START/STOP switches for the main motor
- M4 is the oil pump motor
- M5 is the main motor
- The main motor must only start if the oil pump is running

SECURITY IN PROTOCOLS

At the beginning, specific protocols on specific **physical layer**
(RS232, RS485, 4-20 current loop)
Some protocols were **adapted to TCP/IP**, like Modbus,
and other were developed to allow interoperability.

ICS devices often use **specific protocols**, some of them are **proprietary**, and some of them are **common standards**
We will hereafter cover the most used ones.



MODBUS PROTOCOL

- Serial communication protocol invented in 1979 by Schneider Electric
- Developed for industrial application
- Royalty-free
- Now one of the standards for industrial communications

How it works:

- Master / Slave protocol
- Master must regularly poll the slaves to get information
- Modbus addresses are 8 bits long, so only 247 slaves per master
- There is no object description: a request returns a value, without any context or unit

Security anyone?

- Clear-text
- No authentication

MODBUS PROTOCOL

- Modbus was originally made for serial communications
- However it is now often used over TCP (port 502)

Modbus TCP/IP frame

- Transaction identifier set by the sender
- Protocol identifier set to 0 (default Modbus value)

Transaction identifier	Protocol identifier	Length field	Slave address	Funtion code	Data
					Variable structure depending on the function
2 bytes	2 bytes	2 bytes	1 byte	1 byte	N bytes

MODBUS PROTOCOL

Modbus functions

- The most common Modbus functions allow to read and write data from/to a PLC
- Other functions, such as file read and diagnostics functions also exist
- Undocumented Modbus function codes can also be used to perform specific actions

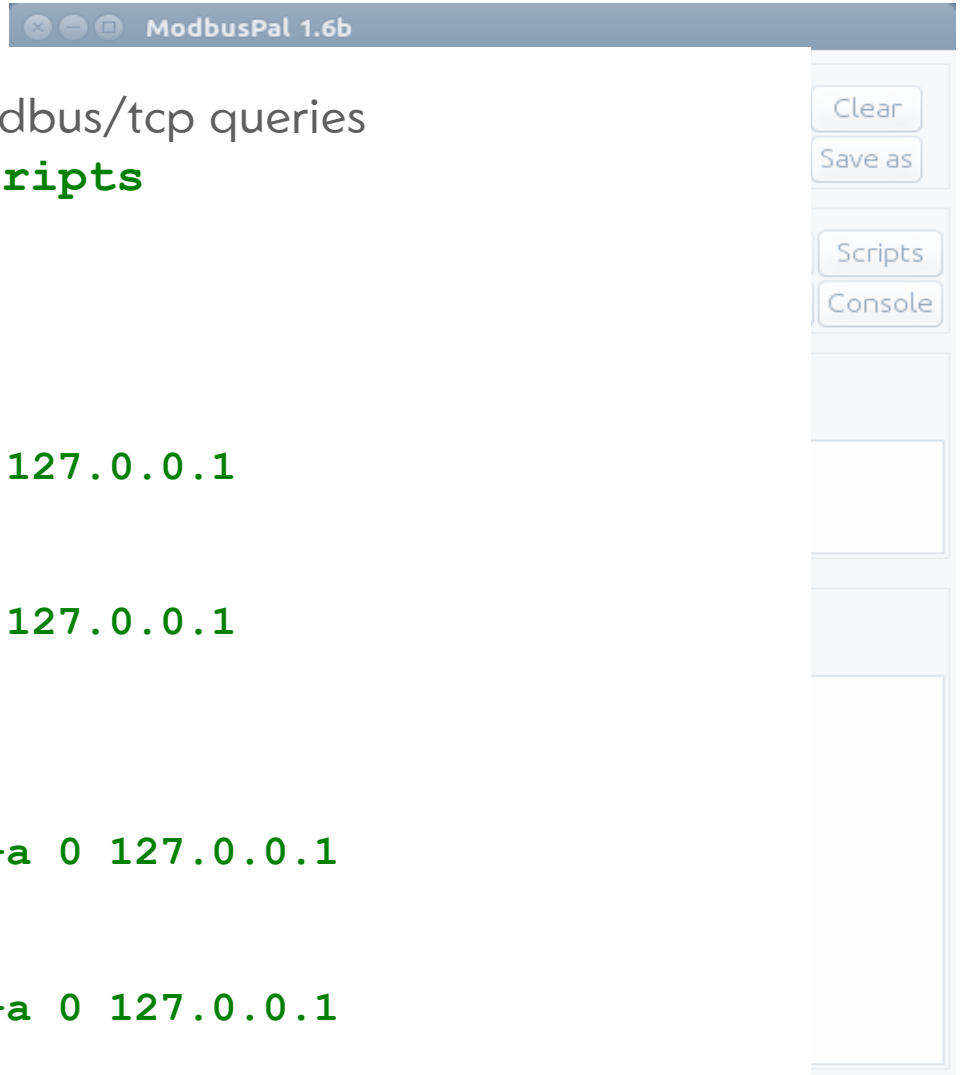
COMMONLY USED MODBUS function codes

Function name	Function code
Read coils	1
Write single coil	5
Read holding registers	3
Write single register	6
Write multiple registers	16
Read/Write multiple registers	23

MODBUS PROTOCOL

Function type			Function name	Function code
Data Access	Bit access	Physical Discrete Inputs	**Read Discrete Inputs**	2
		Internal Bits or Physical Coils	**Read Coils**	1
			Write Single Coil	5
			Write Multiple Coils	15
	16-bit access	Physical Input Registers	**Read Input Registers**	4
		Internal Registers or Physical Output Registers	**Read Holding Registers**	3
			Write Single Register	6
			Write Multiple Registers	16
			Read/Write Multiple Registers	23
			Mask Write Register	22
			Read FIFO Queue	24
	File Record Access		Read File Record	20
			Write File Record	21
Diagnostics			Read Exception Status	7
			Diagnostic	8
			Get Com Event Counter	11
			Get Com Event Log	12
			Report Slave ID	17
			Read Device Identification	43
Other			Encapsulated Interface Transport	43

DEMO: SENDING MODBUS REQUESTS WITH MBTGET



- Mbtget is a perl script to perform Modbus/tcp queries

```
$ > cd ~/toolz/mbtget/scripts
$ > ./mbtget -h
```

- Read requests
  - Coils (1 bit)

```
$ > ./mbtget -r1 -a 0 -n 8 127.0.0.1
```

  - Words (8 bits)

```
$ > ./mbtget -r3 -a 0 -n 8 127.0.0.1
```

- Write requests
  - Coils (1 bit)

```
$ > ./mbtget -w5 #{VALUE} -a 0 127.0.0.1
```

  - Words (8 bits)

```
$ > ./mbtget -w6 #{VALUE} -a 0 127.0.0.1
```

PLC Security

TOP 20 LIST

DISCLAIMER: A COMMUNITY-LED EFFORT

- This presentation is built on top of the hard work of a lot of people:

https://www.youtube.com/watch?v=JtsyyTfSP1I

It all started with Jake Browdsky's presentation at S4

PLC Security

This presentation sparked the idea of the TOP 20 project that was then led by Sarah Fluchs & Vivek Ponnada

Your contact person
Vivek Ponnada
Co-organizer
Email: plc-security-at-admeritia.de
Twitter: @ControlsCyber
LinkedIn: Linkedin

Your contact person
Sarah Fluchs
Co-organizer
Email: plc-security-at-admeritia.de
Twitter: @SarahFluchs
LinkedIn: Linkedin

And 900+ people participated in the discussion to create the first TOP20!

However, this presentation expresses our ideas and does not necessarily reflect the ones of each and every participant to this project!

PLC CODE SECURITY TOP 20 INTRO

1 / 2

Secure PLC Coding Practices: Top 20 List

Version 1.0 (15 June 2021)

PLC Security TOP 20 LIST

1. **Modularize PLC Code**

   Split PLC code into modules, using different function blocks (sub-routines). Test modules independently.

2. **Track operating modes**

   Keep the PLC in RUN mode. If PLCs are not in RUN mode, there should be an alarm to the operators.

3. **Leave operational logic in the PLC wherever feasible**

   Leave as much operational logic e.g., totalizing or integrating, as possible directly in the PLC. The HMI does not get enough updates to do this well.

4. **Use PLC flags as integrity checks**

   Put counters on PLC error flags to capture any math problems.

5. **Use cryptographic and / or checksum integrity checks for PLC code**

   Use cryptographic hashes, or checksums if cryptographic hashes are unavailable, to check PLC code integrity and raise an alarm when they change.

Download the full document on www.plc-security.com

CONTENT OF THE TOP20

Title

Short description

Security objective
- Integrity of
  - PLC logic
  - I/O values
  - PLC variables
- Hardening
- Resilience
- Monitoring

Target group:
- Product supplier
- Integration / Maintenance Service Provider
- Asset Owner

25 / 42

Secure PLC Coding Practices: Details

Version 1.0 (15 June 2021)

PLC Security TOP 20 LIST

12. Validate inputs based on physical plausibility

Ensure operators can only input what's practical or physically feasible in the process. Set a timer for an operation to the duration it should physically take. Consider alerting when there are deviations. Also alert when there is unexpected inactivity.

Security Objective	Target Group
Integrity of I/O values	Integration / Maintenance Service Provider

Guidance

a) Monitor expected physical durations

If the operation takes longer than expected to go from one extreme to the other, that is worthy of an alarm. Alternatively, if it does it too quickly, that is worthy of an alarm too.

Example

a) Monitor expected physical durations

- The gates on a dam takes a certain time to go from fully closed to fully open
- In a wastewater utility, a wet well takes a certain time to fill

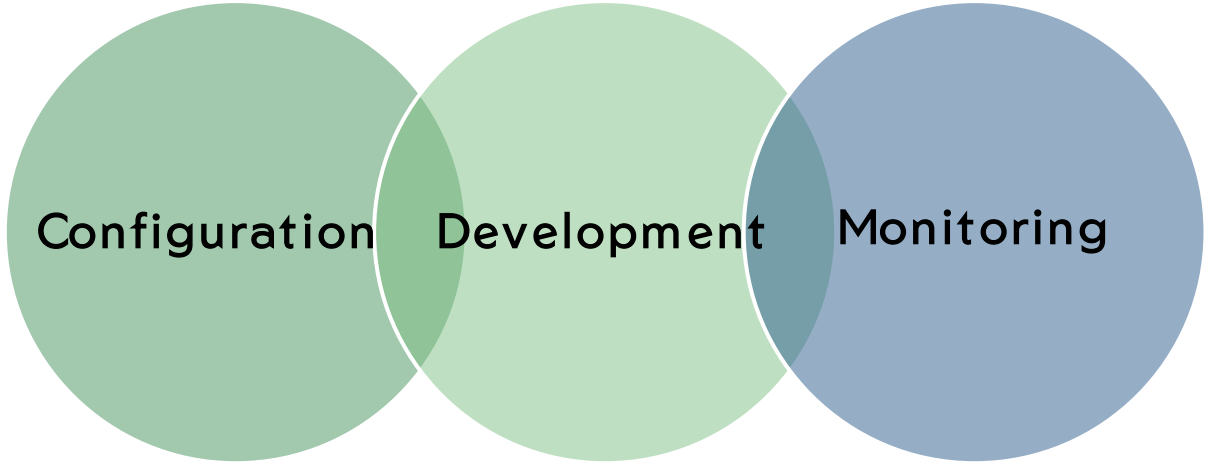
9

Guidance: Everything that helps implementing the practice

Example: Implementation or scenario examples for certain industries, products, ...

My understanding of the TOP 20

My understanding of the TOP 20



Configuration

3. Leave operational logic in the PLC wherever feasible

10. Assign designated register blocks by function (read/write/validate)

13. Disable unneeded / unused communication ports and protocols

14. Restrict third-party data interfaces

15. Define a safe process state in case of a PLC restart

Development

1.Modularize PLC Code

6. Validate timers and counters

7. Validate and alert for paired inputs / outputs

8. Validate HMI input variables at the PLC level, not only at HMI

9. Validate indirections

11. Instrument for plausibility checks

12. Validate inputs based on physical plausibility

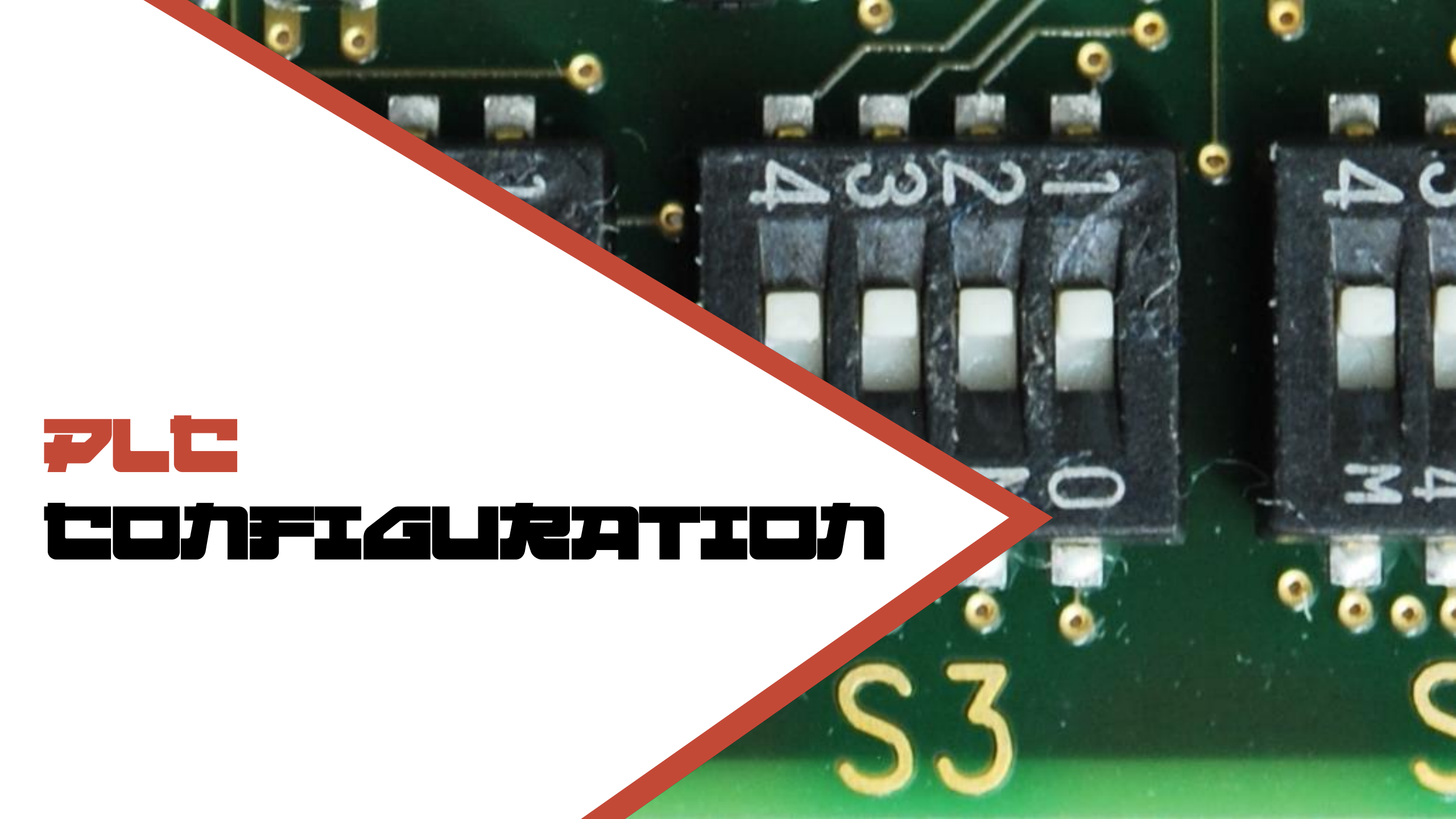
Monitoring

- 2. Track operating modes
- 4. Use PLC flags as integrity checks
- 5. Use cryptographic and / or checksum integrity checks for PLC code
- 16. Summarize PLC cycle times and trend them on the HMI
- 17. Log PLC uptime and trend it on the HMI
- 18. Log PLC hard stops and trend them on the HMI
- 19. Monitor PLC memory usage and trend it on the HMI
- 20. Trap false negatives and false positives for critical alerts

LOTS OF GUI ACTIONS, SO PLEASE TRY TO FOLLOW ALONG, NOT EVERYTHING IS DETAILED IN THE SLIDES!

PLC

CONFIGURATION

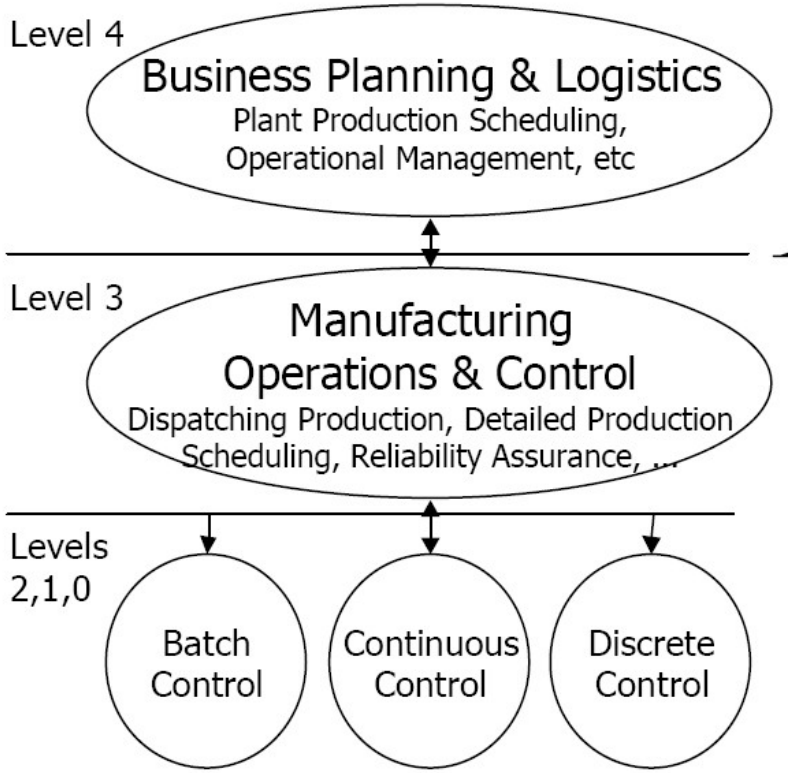


R#3 LEAVE OPERATIONAL LOGIC IN THE PLC WHEREVER FEASIBLE

- While HMIs provide some coding capabilities, these should not be used for operational logic (ie anything that will determine the values of the outputs)

- We can make a parallel with IT: on a webpage, if a user must not be able to perform an action, you would not just hide the button with javascript, you need to modify the application code. In OT, you also need to put the controls in the PLC, and not just hide the button in the HMI

R#3 LEAVE OPERATIONAL LOGIC IN THE PLC WHEREVER FEASIBLE



- Operational logic should be implemented close to the process, in the lower levels of the Purdue model

R#10 ASSIGN DESIGNATED REGISTER BLOCKS BY FUNCTION (READ/WRITE/VALIDATE)

- The Modbus protocol defines several types of data: coils vs registers, input vs holding

4.3 MODBUS Data model

MODBUS bases its data model on a series of tables that have distinguishing characteristics. The four primary tables are:

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

The distinctions between inputs and outputs, and between bit-addressable and word-addressable data items, do not imply any application behavior. It is perfectly acceptable, and very common, to regard all four tables as overlaying one another, if this is the most natural interpretation on the target machine in question.

MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b

R#10 ASSIGN DESIGNATED REGISTER BLOCKS BY FUNCTION (READ/WRITE/VALIDATE)

- Each PLC code variable is mapped to a register (or a coil)
- It is left to the manufacturer to decide how to implement this data type, which results in heterogeneity
- Using dedicated blocks for each type of variable prevents mistakes, like directly using an external value
- Examples of dedicated blocks include:
  - Reading
  - Writing (from HMI / Controller / other external device)
  - Validating writes
  - Calculations

R#10: SETTING VARIABLES

We will define 3 separate blocks for our variables:

- 0-100 for inputs
- 101-200 for validated inputs
- 201-300 for outputs

And we will apply that to both coils & registers

R#13 DISABLE UNNEEDED / UNUSED COMMUNICATION PORTS AND PROTOCOLS

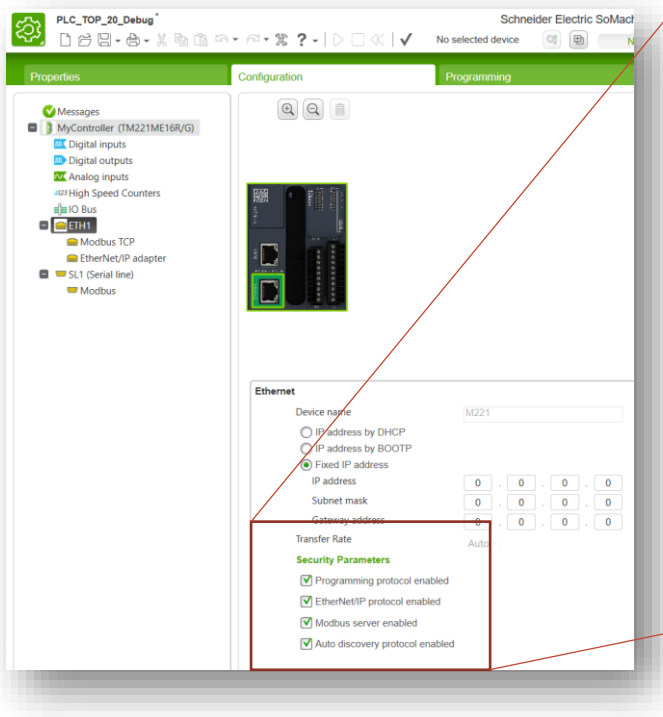
- PLC controllers and network interface modules generally support multiple communication protocols that are enabled by default. Disable ports and protocols that are not required for the application
- We can separate in two categories:
  - ICS protocols: should be limited to the one you use, and favor the most secure ones (if any ;))
  - Additional features: web server, SNMP, FTP,... These features should all be disabled unless you have a specific use case for it (ex: enabling SNMP because it is used by your inventory automated discovery scanner)

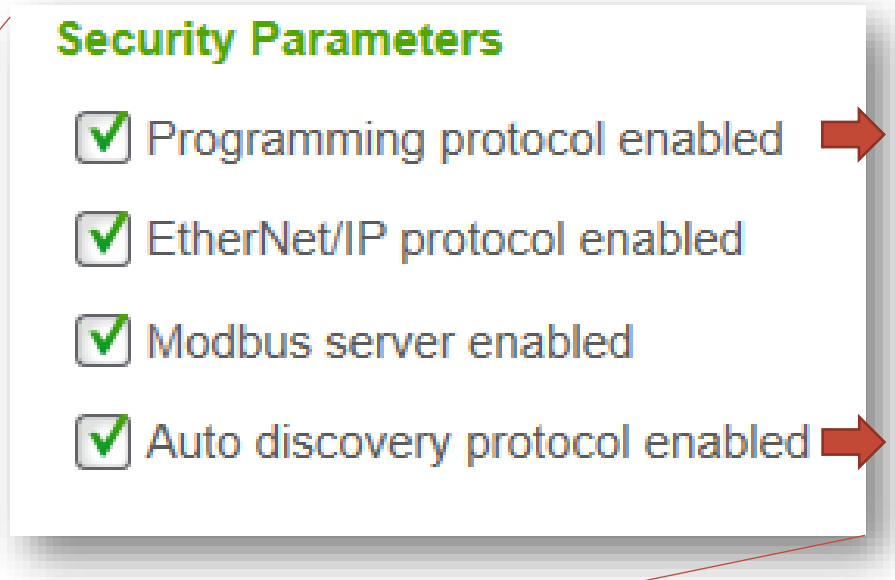
R#14 RESTRICT THIRD-PARTY DATA INTERFACES

- Restrict the type of connections and available data for 3rd party interfaces. The connections and/or data interfaces should be well defined and restricted to only allow read/write capabilities for the required data transfer
- Several things to consider:
  - Architecture: if possible, use a dedicated and independent "data concentrator" to exchange data (ie, no 3rd party system directly connected to your PLC)
  - Protocol: depending on your needs, use a protocol that allows "read only" access, and favor protocols that allow to precisely define what can be done (access control)
  - Monitoring: make sure to check that the data exchange is working and that your PLC gets "fresh" data

R#13/14: HARDENING THE PLC

- The Schneider TM221 is an entry-level PLC, with very limited configuration possibilities





Allows the PLC to be configured and programmed over the network. If disabled, you need to connect using USB

Allows the PLC to respond to specific network scans by soMachineBasic to identify PLCs on the network

R#13/14: HARDENING THE PLC

- Other / Higher-end PLCs will have additional configuration possibilities
  - Disabling services like the web interface, FTP server…
  - Some implement basic firewalling possibilities
  - …

R#15 DEFINE A SAFE PROCESS STATE IN CASE OF A PLC RESTART

- If something commands a PLC to restart in the middle of a working process, we should expect the program to pick up smoothly with minimal disruption to the process. Make sure that the process it controls is restart-safe.
- Behavior of inputs and outputs after a restart of the PLC can be set in the engineering software
- Internal registers can also take default values

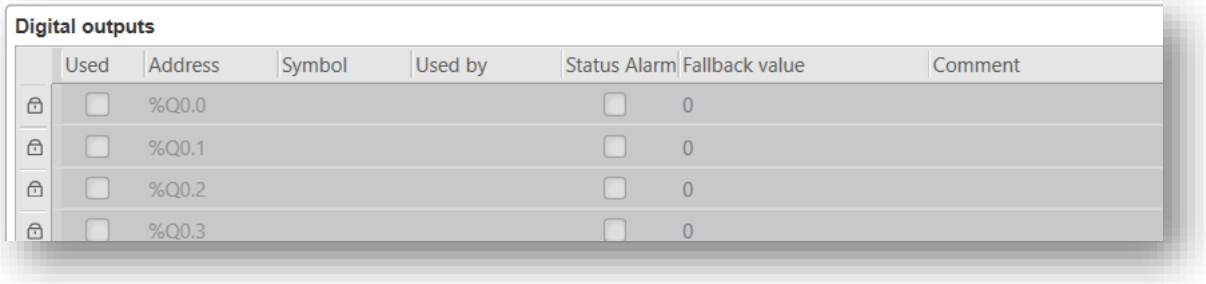
R#15 DEFINE A SAFE PROCESS STATE IN CASE OF A PLC RESTART

- soMachineBasic allows you to choose the default behavior of the outputs if the PLC stops or is in error, and to trigger this beahvior on command

%S9	Fallback outputs	When %S9 is set to 1: • For outputs configured as Status Alarms, PTO, or FREQGEN, the outputs are set to 0. • Fallback values are applied to the physical digital and analog outputs (embedded outputs, TM2/TM3 expansion module outputs and TMC2 cartridge outputs). The data image is unaffected by %S9. The data image reflects the logic applied by the application. Only the physical outputs are affected. • The fallback values are applied regardless of the fallback behavior *(see SoMachine Basic, Operating Guide)* mode configured for specific outputs. When %S9 is set to 0, the data image values are re-applied to the physical outputs. **NOTE:** When the controller is in the STOPPED state and **Maintain values** fallback behavior is configured, a rising edge on %S9 applies fallback values to the physical outputs and to data image values.	0	U

R#15 DEFINE A SAFE PROCESS STATE IN CASE OF A PLC RESTART

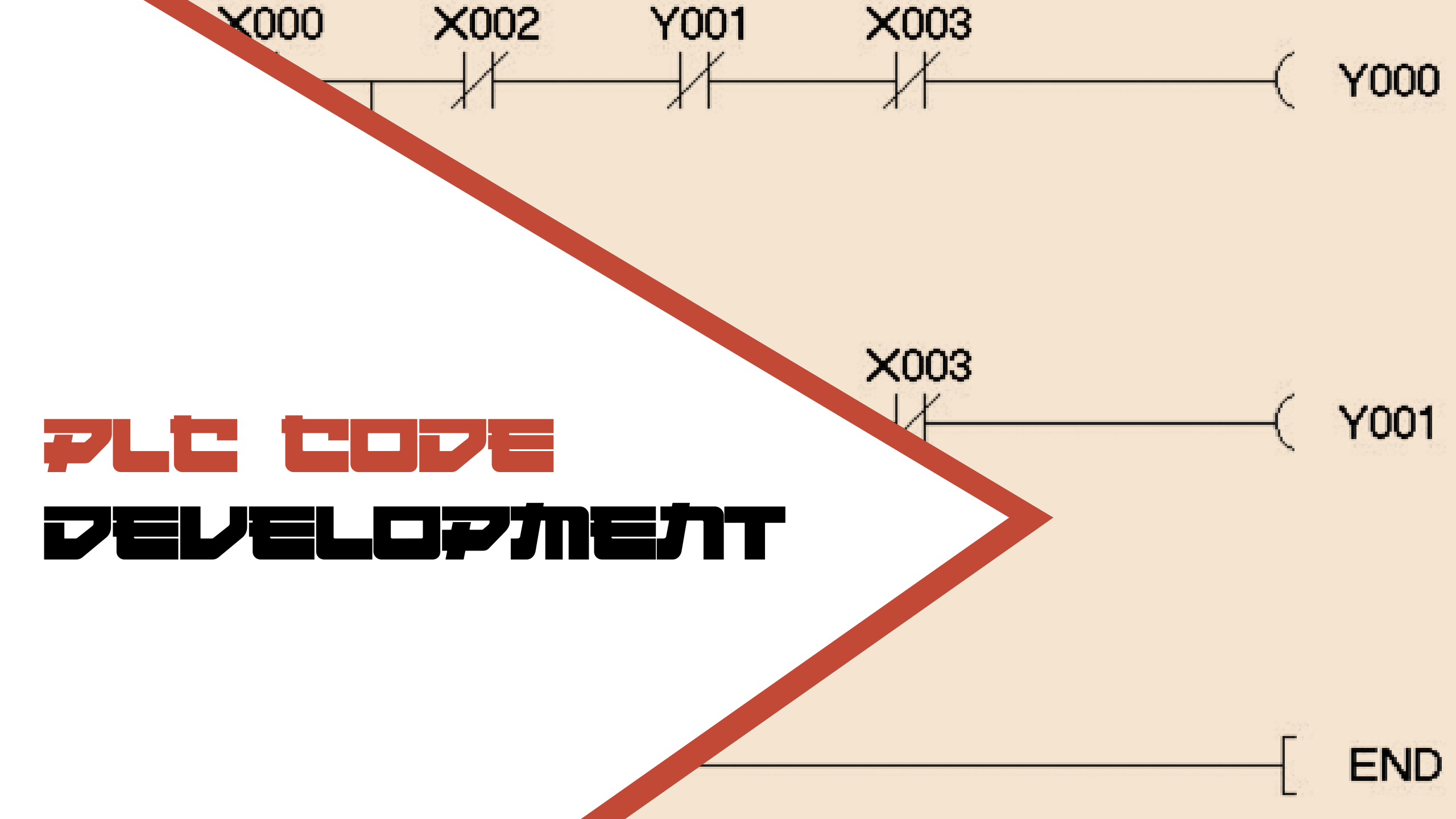
- soMachineBasic allows you to choose the default behavior of the outputs if the PLC stops or is in error.



Digital outputs

	Used	Address	Symbol	Used by	Status Alarm	Fallback value	Comment
	☐	%Q0.0			☐	0	
	☐	%Q0.1			☐	0	
	☐	%Q0.2			☐	0	
	☐	%Q0.3			☐	0	

Fallback value	Yes	1 or 0	0	Specify the value to apply to this output (fallback to 0 or fallback to 1) when the logic controller enters the `STOPPED` or an exception state. The default value is 0. If **Maintain values** fallback mode is configured, the output retains its current value when the logic controller enters the `STOPPED` or an exception state. This field is disabled for the output configured as **Status Alarm**.



PLC CODE DEVELOPMENT

R#1 MODULARIZE PLC CODE

- Do not program the complete PLC logic in one place e.g., in the main Organization Block or main routine. Instead, split it into different function blocks (sub-routines) and monitor their execution time and their size in Kb.
- Create separate segments for logic that functions independently. This helps in input validation, access control management, integrity verification, etc.

R#1 MODULARIZE PLC CODE

- Let's create the following POUs
  - CONFIGURATION
  - PROCESS STEPS
  - MONITORING OF THE PLC

R#6 VALIDATE TIMERS AND COUNTERS

- If timers and counters values are written to the PLC program, they should be validated by the PLC for reasonableness and verify backward counts below zero.
- If remote devices such as an HMI write timer or counter values to a program:
  - do not let the HMI write to the timer or counter directly but go through a validation logic
  - validate presets and timeout values in the PLC

R#7 VALIDATE AND ALERT FOR PAIRED INPUTS / OUTPUTS

- If you have paired signals, ensure that both signals are not asserted together. Alarm the operator when input / output states occur that are physically not feasible. Consider making paired signals independent or adding delay timers when toggling outputs could be damaging to actuators.
- Independent start & stop:
  - Configure start and stop as discrete outputs instead of having a single output that can be toggled on/off. By design, this does not allow simultaneous triggers. For an attacker, it is way more complicated to rapidly toggle on / off if two different outputs have to be set.
  - Timer for restart: Also consider adding a timer for a re-start after a stop is issued to avoid rapid toggling off start/stop signals.

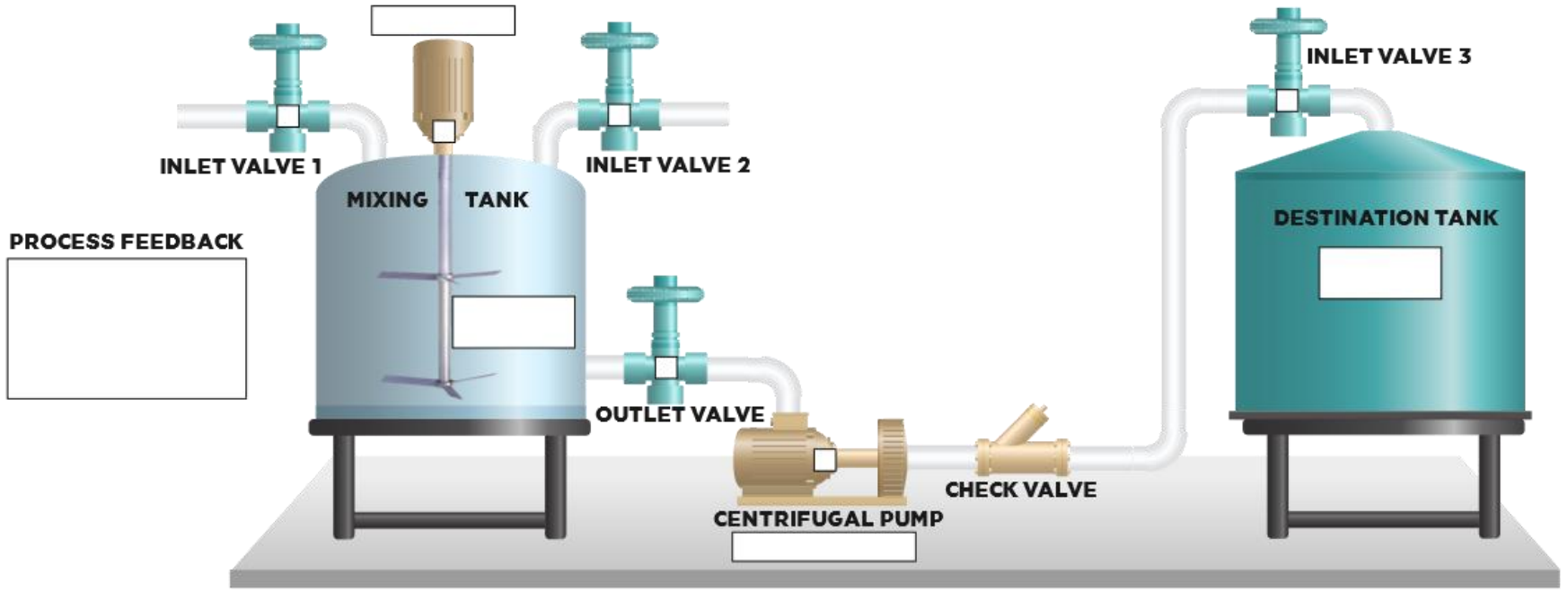
R#8 VALIDATE HMI INPUT VARIABLES AT THE PLC LEVEL

- HMI access to PLC variables can (and should) be restricted to a valid operational value range at the HMI, but further cross-checks in the PLC should be added to prevent, or alert on, values outside of the acceptable ranges which are programmed into the HMI

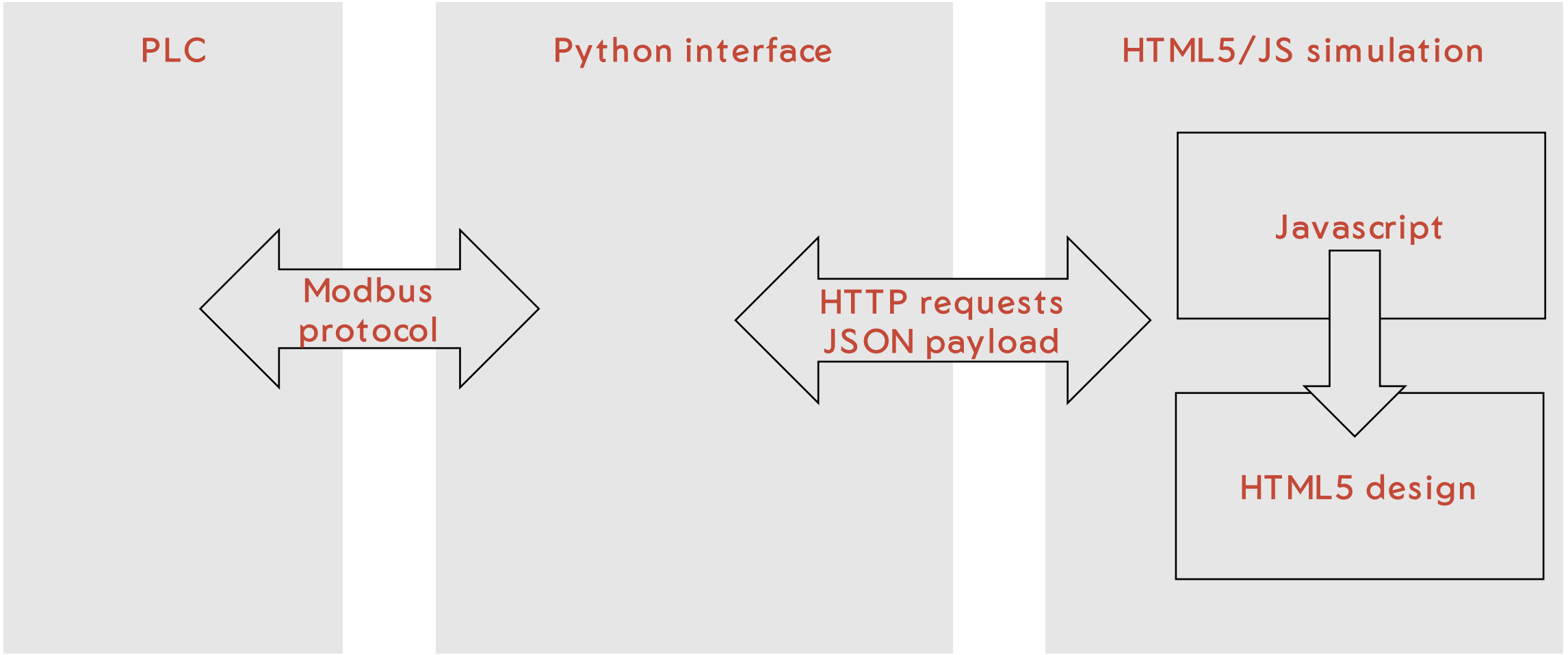
The virtual test rig

DC30 WORKSHOP

Securing ICS from the core: TOP20 PLC secure coding practices



Behind the scenes



R#6 & R#8
VALIDATE INPUTS AT THE PLC LEVEL

- Launch the web application and then access http://localhost:5000
  - cd simulation
  - python –m flask run
- Launch the PLC program with SoMachine Basic
  - Simulation
- Launch IGSS to start the SCADA application

R#6 & R#8
VALIDATE INPUTS AT THE PLC LEVEL

- Attack the process and mess with the timer
  - Use mbtget on the Kali machine to write a negative timer value on %MW5 (ip address 10.0.0.1)
    - ./mbtget -w6 60000 –a 5 10.0.0.1
- Add validation checks to the timer duration provided by the HMI

R#9 VALIDATE INDIRECTIONS

- An indirection is the use of the value of a register in another register. There are many reasons to use indirections.
- Examples for necessary indirections are:
  - Variable frequency drives (VFDs) that trigger different actions for different frequencies using lookup tables.
  - To decide which pump to start running first based on their current run times
  - PLCs do not typically have an "end of an array" flag so it's a good idea to create it in software; the goal is to avoid unusual/unplanned PLC operations.

R#11 INSTRUMENT FOR PLAUSIBILITY CHECKS

- Instrument the process in a way that allows for plausibility checks by cross-checking different measurements.
- Example: compare integrated and time-independent measurements
  - Metered pump and tank level gauge: volumetric change should equal integrated flow.
  - Burner in a boiler: added caloric heat should equal temperature rise.

R#11
INSTRUMENT FOR PLAUSIBILITY CHECKS

- Where can we implement that on our demo setup?

R#12 VALIDATE INPUTS BASED ON PHYSICAL PLAUSIBILITY

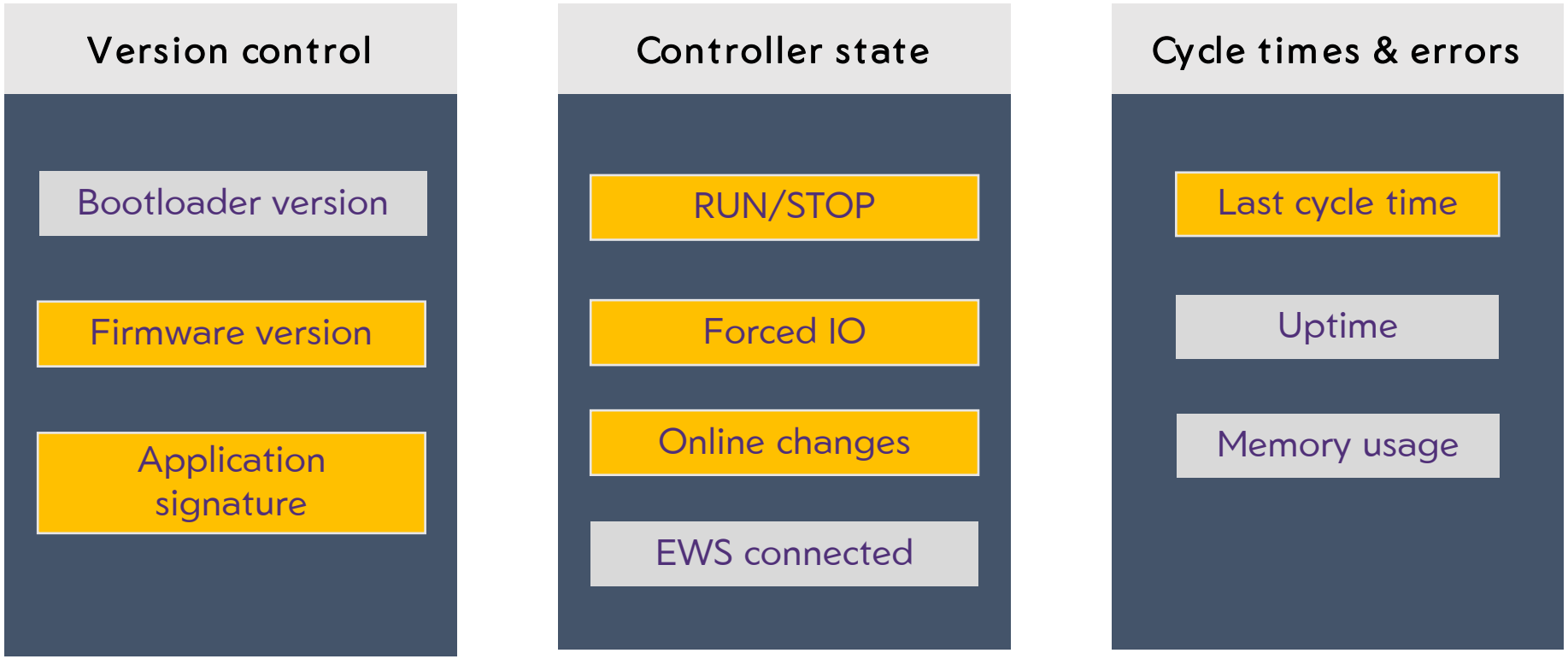
- Ensure operators can only input what's practical or physically feasible in the process.
- Set a timer for an operation to the duration it should physically take. Consider alerting when there are deviations. Also alert when there is unexpected inactivity

PLC MONITORING

PLC MONITORING FOR CYBERSECURITY

XXX Implemented in the demo environment

- Leverage operational data to identify potential cyber-related events



R#2 TRACK OPERATING MODES

2. Track operating modes

Keep the PLC in RUN mode. If PLCs are not in RUN mode, there should be an alarm to the operators.

Security Objective	Target Group
Integrity of PLC logic	Integration / Maintenance Service Provider Asset Owner

Guidance

If PLCs are not in RUN mode (e.g., PROGRAM mode), their code could be changed to track the RUN mode. Some PLCs have a checksum to alert for code changes, but if they do not, there's at least an indirect indicator of a potential issue while tracking operating modes:

- If PLCs are not in RUN mode, there should be an alarm to the operators. If they are aware that someone is supposed to be working on that control system, they can acknowledge the alarm and move on.
- The HMI should be configured to re-alert the operator toward the end of the shift about the presence of the alarm. The goal should be to keep track of any staff or contractors in the plant doing work that might impact the process.

System Words	Function	Description
%SW6	Controller state %MW60012	Controller state: 0 = EMPTY 2 = STOPPED 3 = RUNNING 4 = HALTED 5 = POWERLESS
%SW7	Controller status	• Bit [0]: Backup/restore in progress: ❍ Set to 1 if backup/restore of the program is in progress, ❍ Set to 0 if backup/restore of the program is complete or disabled. • Bit [1]: Configuration of the controller is OK: ❍ Set to 1 if configuration ok. • Bit [2]: SD card status bits: ❍ Set to 1 if SD card is present. • Bit [3]: SD card status bits: ❍ Set to 1 if SD card is being accessed. • Bit [4]: Application memory status: ❍ Set to 1 if application in RAM memory is different from that in non-volatile memory. • Bit [5]: SD card status bits: ❍ Set to 1 if SD card is in error. • Bit [6]: Not used (status 0) • Bit [7]: Controller reserved: ❍ Set to 1 when the controller is connected to SoMachine Basic.
%S14	I/O force activated	Normally set to 0. Set to 1 by the system if at least one input or output is being forced.
%S15	Input forced	Normally set to 0. Set to 1 by the system if at least one input is being forced.

R#2 TRACK OPERATING MODES

- Identify the right bits & word to monitor
- Connect it to the SCADA
- Implement an alarm
- Test!

R#4 USE PLC FLAGS AS INTEGRITY CHECKS

- If the PLC code was working fine but suddenly does a divide by zero, investigate. If something is communicating peer to peer from another PLC and the function/logic does a divide by zero when it wasn't expected, investigate.

R#4
USE PLC FLAGS AS INTEGRITY CHECKS

- Create a buffer overflow situation and try to detect it

System Bit	Function	Description	Init State	Control
%S18	Arithmetic overflow or error	Normally set to 0. It is set to 1 in the case of an overflow when a 16-bits operation is performed, that is: ● A result greater than + 32767 or less than - 32768, in single length, ● A result greater than + 2147483647 or less than - 2147483648, in double length, ● A result greater than + 3.402824E+38 or less than - 3.402824E+38, in floating point, ● Division by 0, ● The square root of a negative number, ● BTI or ITB conversion not significant: BCD value out of limits. It must be tested by the program after each operation where there is a risk of an overflow; then reset to 0 by the program if an overflow occurs.	0	S→U, SIM

R#4
USE PLC FLAGS AS INTEGRITY CHECKS

- Simplest way to create a buffer overflow: add two large numbers and store the result in a standard variable
- Ex:

```
%MW3 := %MW1 + %MW2
%MW1:= 32000
%MW2:= 32000
```

- Use the « animated table » to modify the values
- See if you can cacth the error from the PLC

R#5 USE CRYPTOGRAPHIC AND / OR CHECKSUM INTEGRITY CHECKS FOR PLC CODE

- Use cryptographic hashes, or checksums if cryptographic hashes are unavailable, to check PLC code integrity and raise an alarm when they change

R#5 USE CRYPTOGRAPHIC AND / OR CHECKSUM INTEGRITY CHECKS FOR PLC CODE

Monitor the right system values and check a change in the PLC code changes the checksum

System Words	Function	Description
%SW13	Boot loader version xx.yy	For example, if %SW13=000E hex: • 8 MSB=00 in hexadecimal, then xx=0 in decimal • 8 LSB=0E in hexadecimal, then yy=14 in decimal As a result, boot loader version is 0.14, displayed as 14 decimal.
%SW14	Commercial version, xx.yy	For example, if %SW14=0232 hex: • 8 MSB=02 in hexadecimal, then xx=2 in decimal • 8 LSB=32 in hexadecimal, then yy=50 in decimal As a result, commercial version is 2.50, displayed as 250 decimal.
%SW15-%SW16	Firmware version aa.bb.cc.dd	For example, if: %SW15=0003 hex: • 8 MSB=00 in hexadecimal, then aa=00 in decimal • 8 LSB=03 in hexadecimal, then bb=03 in decimal %SW16=0B16 hex: • 8 MSB=0B in hexadecimal, then cc=11 in decimal • 8 LSB=16 in hexadecimal, then dd=22 in decimal As a result, firmware version is 0.3.11.22 displayed as 00031122 decimal.
%SW94 %SW95	Application signature %MW60028-%MW60034	If the application changes, in terms of configuration or programming data, the signature (sum of all checksums) also changes. If %SW94 = 91F3 in hexadecimal, the application signature is 91F3 in hexadecimal.

➔ How could this be bypassed ?

R#16 SUMMARIZE PLC CYCLE TIMES AND TREND THEM ON THE HMI

16. Summarize PLC cycle times and trend them on the HMI

Summarize PLC cycle time every 2-3 seconds and report to HMI for visualization on a graph.

Security Objective	Target Group
Monitoring	Integration / Maintenance Service Provider

Guidance

Cycle times are usually system variables in a PLC and can be used for summarizing in PLC code. Summarization should be done to calculate average, peak, and minimum cycle times. The HMI should trend these values and alert if there are significant changes.

The cycle time is the time it takes to compute each iteration of logic for the PLC. The iterations are the combination of Ladder Diagrams (LD), Function Block Diagrams (FBD), Instruction List (IL), and Structured Text (ST). These logic components may be joined together with the Sequential Function Charts (SFC).

Cycle times should be constant on a PLC unless there are changes to e.g.

- network environment
- PLC logic
- process

Therefore, unusual cycle time changes can be an indicator that PLC logic changed and thus provide valuable information for integrity checks.

Visualizing values over time using a graph provides an intuitive way to draw attention to anomalies which would be harder to notice by just having absolute values.

System Words	Function	Description	Control
%SW30	Last scan time (master task)	Indicates the execution time of the last controller scan cycle (in ms). **NOTE:** This time corresponds to the time elapsed between the start (acquisition of inputs) and the end (update of outputs) of a master task scan cycle. If the scan time is 2.250 ms, the %SW30 is 2 and the %SW70 is 250.	S

R#16 SUMMARIZE PLC CYCLE TIMES AND TREND THEM ON THE HMI

- Let's monitor the right value
- Try to modify the PLC code (add some code, remove some code)
- Check that the cycle time graph is working

R#17 LOG PLC UPTIME AND TREND IT ON THE HMI

- Keep track of PLC uptime
  - In the PLC itself (if uptime is a system variable in the PLC)
  - In the PLC itself if it has MIB-2 / any SNMP implementation
  - Externally by means of e.g., SNMP

R#18 LOG PLC HARD STOPS AND TREND THEM ON THE HMI

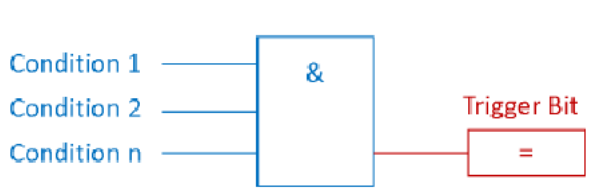
- Store PLC hard stop events from faults or shutdowns for retrieval by HMI alarm systems to consult before PLC restarts. Time sync for more accurate data.

R#19 MONITOR PLC MEMORY USAGE AND TREND IT ON THE HMI

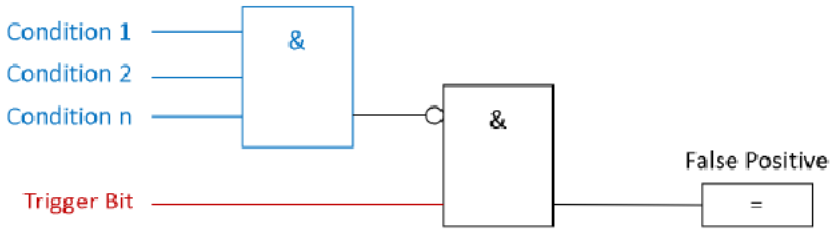
- Measure and provide a baseline for memory usage for every controller deployed in the production environment and trend it on the HMI.

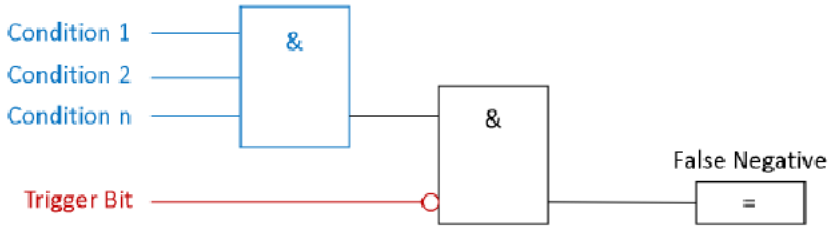
R#20 TRAP FALSE NEGATIVES AND FALSE POSITIVES FOR CRITICAL ALERTS

- Identify critical alerts and program a trap for those alerts. Set the trap to monitor the trigger conditions and the alert state for any deviation.



Raise an alarm if any of the condition is met





Detect false positive & false negative

TOP20

conclusion

TAKEWAYS

Not such good ideas

Operators must supervise in real time another screen with the cybersecurity issues → No !!!

PLC code security is sufficient to prevent attacks on ICS environments → No !!!!

Real takeaways

- PLC code can have an impact on the cybersecurity of a system
- PLC code security best practices can also contribute to the quality & safety of the process
- Great topic to empower automation engineers and show that cybersecurity is not just an « IT » things with firewalls and network probes

What can you do today?

As an automation engineer

- Download & read the TOP20
- See how it could be implemented in your environment
- Participate in the community (translation, implementation guide for specific vendors, additional practices…)

As an asset owner

- Mention the topic to your automation engineers
- Add the TOP20 as a requirement during procurement

As a system integrator

- Train your automation engineers to the TOP20
- Use it as business differentiator for your clients

As an automation vendor

- Make sure the TOP20 is easily applicable to your products
- Publish implementation guidelines
- Get involved in the project

TO LEARN MORE ABOUT ICS CHECK OUT THE ICS VILLAGE!

arnaud.soullie@wavestone.com
alexandrine.torrents@wavestone.com