Top 20 Secure PLC Coding Practices Application Notes

<Name of Vendor / Integrator>
<Name of Use Case>

Content

[Use Case Introduction 1](#_Toc103754288)

[Application Statement 2](#_Toc103754289)

[Application Details 4](#_Toc103754290)

[About <Vendor / Integrator> 5](#_Toc103754291)

[Authors of these application notes 5](#_Toc103754292)

[About Top 20 Application Notes 6](#_Toc103754293)

[About the Top 20 Secure PLC Programming project 6](#_Toc103754294)

# Use Case Introduction

## Type of organization

<add: vendor / integrator / operator / other?>

## PLC make / model

<add: could be multiple>

## Scenario

<add: What is the scenario in which you’re applying the Top 20 Secure PLC Coding Practices?>

# Application Statement

| # | Practice Title | Applied? (yes / no) | Notes |
| --- | --- | --- | --- |
| **1** | Modularize PLC Code | Yes | <add any notes on why / why not / how you apply this practice. More details can be added in “application details” section> |
| **2** | Track operating modes | No |  |
| **3** | Leave operational logic in the PLC |  |  |
| **4** | Use PLC flags as integrity checks |  |  |
| **5** | Use cryptographic and / or checksum integrity checks for 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 |  |  |
| **8** | Validate indirections |  |  |
| **10** | Assign designated register blocks by function (read / write / validate) |  |  |
| **11** | Instrument for plausibility checks |  |  |
| **12** | Validate inputs based on physical plausibility |  |  |
| **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 |  |  |
| **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 |  |  |

# Application Details

<add any content / subsections… or delete>

# About <Vendor / Integrator>

<description>

# Authors of these application notes

|  |  |
| --- | --- |
| Author 1 | Author 2 |
| Author 3 | Author 4 |
|  |  |
|  |  |
|  |  |
|  |  |

# About Top 20 Application Notes

The Top 20 Secure PLC Coding Practices are a community effort with best practices gathered from a large crowd of engineers from all kinds of different organizations. Thus, each single practice has been used by someone in the community.

However, there are many different kinds of PLCs and environments out there, for which the Top 20 as they are may or may not apply. The Top 20 Application Notes are case studies for specific PLCs, specific organizations (vendors, integrators, operators) and their workflows. People who have tried to apply the Top 20 take notes on their experiences – how they applied the practices, what worked, and what did not work. The aim is to gather application examples to help others, one use case at a time, and to eventually improve the Top 20’s real-world applicability. Application notes issued by vendors and integrators are especially important since operators can use them as guidance for the PLCs they have in operation or consider buying.

Sharing your own Top 20 Application Note is easy. Just complete this template (feel free to modify as needed), send to plc-security@admeritia.de so we can publish on the Secure PLC project’s website and social media channels and share widely with your clients, colleagues, prospects, network and across social media.

# About the Top 20 Secure PLC Programming project

For many years, Programmable Logic Controllers (PLCs) have been insecure by design. Several years into customizing and applying best practices from IT gave rise to secure protocols, encrypted communications, network segmentation etc. However, to date, there has not been a focus on using the characteristic features in PLCs (or SCADA/DCS) for security, or how to program PLCs with security in mind. The Secure PLC Programming project – inspired by the existing Secure Coding Practices for IT – fills that gap.

Written for engineers by engineers: The aim of this project is to provide guidelines to engineers that are creating software (ladder logic, function charts etc.) to help improve the security posture of Industrial Control Systems.

These practices leverage natively available functionality in the PLC/DCS. Little to no additional software tools or hardware is needed to implement these practices. They can all be fit into the normal PLC programming and operating workflow. More than security expertise, good knowledge of the PLCs to be protected, their logic, and the underlying process is needed for implementing these practices. To fit the scope of the Top 20 Secure PLC Coding practices list, practices need to involve changes made directly to a PLC.

Fore more information, visit: [plc-security.com](https://plc-security.com/)