Process Controllers Under Attack: Real-Time Performance and Cyber-Physical Risks

[Sinclair Koelemij]

Cyber-Physical Risk Expert | Founder Cyber-Physical Risk Academy | Consultant, Speaker, Trainer, Publisher | Operational Technology | Masterclasses | Training | 45+ years in process automation. OT security focus.

January 17, 2025

Why This Article?

First and foremost, because I often notice a significant lack of understanding about DCS in security discussions. The distinction between DCS and SCADA—two entirely different systems with very different vulnerabilities—is frequently overlooked. When people talk about ICS/SCADA, it's genuinely frustrating to see the widespread misconception that ICS/SCADA inherently includes DCS, ignoring the fundamental differences between these systems.

Hence, this article focuses on the process controller which differs very much from a PLC. And when discussing that topic, we cannot ignore real-time performance and the specific demands it imposes. These demands are fundamentally different from those in a SCADA system, as SCADA and DCS represent completely different approaches to process control.

Introduction.

Process controllers are central to modern industrial operations, managing critical parameters such as temperature, pressure, flow, and level. While both process controllers and PLCs have evolved to

handle overlapping tasks—such as executing logic and control functions—their primary difference lies in their inner workings. PLCs were originally designed for logic control, executing a series of logical algorithms cyclically, making them well-suited for discrete automation tasks. In contrast, process controllers are optimized for continuous process regulation, focusing on precise control strategies to maintain stable operations in complex environments.

Over time, process controllers have incorporated logic execution capabilities, just as PLCs have integrated basic control functions. In this article, I will explain how process controllers work, describe their hardware and software components, and discuss their role and potential weaknesses. By understanding these aspects, you'll gain a clearer view of the challenges that process controllers face. Let's start by defining what a process controller is.

Modern Industrial Control Systems (ICS) depend on both Level 1 and Level 0 components to ensure effective control and safety. These include process controllers, which regulate processes and execute logic functions, alongside devices such as PLCs, Remote Terminal Units (RTUs), safety controllers, process analyzers, and field devices like transmitters and actuators. Together, these components collect data, analyze it in real-time, and execute control actions to maintain stable and secure operations.

By understanding the differences in how process controllers and PLCs operate—particularly their focus and internal mechanisms—organizations can better identify potential vulnerabilities, improve performance, and ensure the safety and reliability of their systems. The following boiler control strategy is typically implemented in a DCS process controller. While modern PLCs may be capable of executing the same strategy, DCS controllers offer significant advantages in terms of algorithms, scalability, and integration compared to implementing the strategy in a PLC.

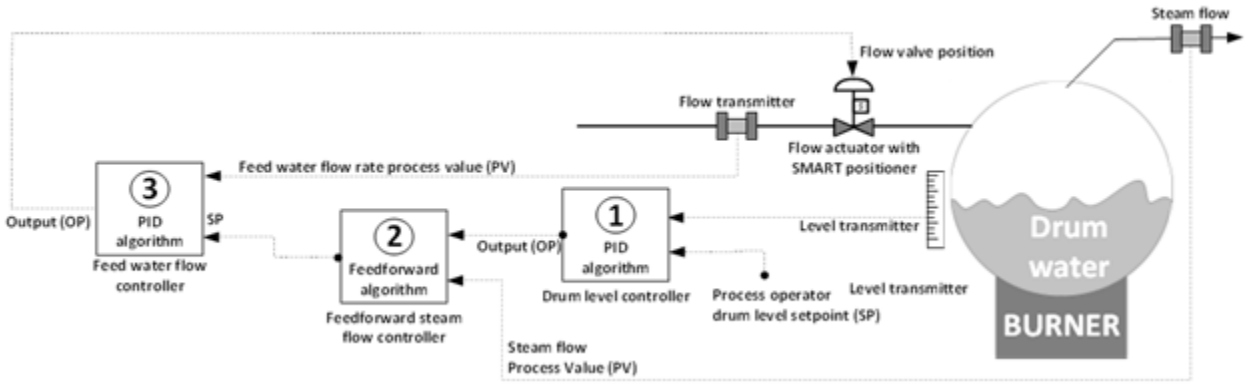

Figure 1 - 3 element boiler control strategy

The three-element drum level control strategy depicted in the drawing ensures precise control of the boiler drum water level by integrating feedback and feedforward control.

· **Drum Level Control (Element 1)**: The drum water level is measured and processed by a PID controller. Its output determines the setpoint for feedwater flow to maintain the desired drum level.

· **Steam Flow Compensation (Element 2)**: The steam flow is measured and used as a feedforward signal. This anticipates changes in steam demand, ensuring proactive adjustments to the feedwater flow.

· **Feedwater Flow Control (Element 3)**: The feedwater flow is measured and processed by a PID controller to adjust the feedwater valve, ensuring the actual flow matches the setpoint.

By combining these three elements, the system quickly reacts to changes in steam demand and feedwater conditions, maintaining drum level stability and efficient boiler operation. A PID algorithm controls a process by adjusting an input to keep a variable, like flow or level, at a setpoint set by the process operator or in some case by higher level advanced control algorithms.

The letters stand for:

- **Proportional (P)**: Corrects output signal based on the current error (difference between setpoint and actual value).
- **Integral (I)**: Fixes past errors by summing them over time.
- **Derivative (D)**: Predicts future errors by looking at how fast the error is changing.

These actions work together to keep the process stable and on target, provided the parameter values are properly tuned. When I discuss various attack scenarios, I will revisit this topic because, as an attacker, modifying these values could destabilize the process, causing it to oscillate and potentially leading to significant dangers.

Another important point to note is the connections between the three elements. These connections can be either soft-wired (configured in software) or hardwired (using physical cables). Typically, they are soft-wired, and in the configuration shown in the diagram, it's likely that all elements reside within the same process controller, as all inputs and outputs seem to come from a single location.

However, in cases where multiple boilers feed steam to a shared steam header, the setup may differ. Boilers in separate locations may be controlled by different process controllers due to distance or to add redundancy. While process controllers are usually redundant, some setups would use separate controllers per boiler or other process equipment to further reduce the risk of simultaneous failure of two units.

In such a setup, where multiple sources feed into the same header, balancing the steam feed to the shared header requires data exchange between the controllers. This affects the configuration of the feedforward strategy in the example and involves sending messages over the Level 1 network if both controllers reside in the same equipment room. In larger systems with multiple equipment rooms distributed over the plant side, these signals may even route through Level 2 networks. This peer communication between controllers is critical, as it must meet strict performance requirements to support the redundancy of the controllers and prevent control mode shedding. I will revisit this topic when discussing attack scenarios that focus on disrupting these control strategies.

Process controllers and their associated hardware.

Let me first discuss the hardware and software of a process controller. Although the exact hardware configuration varies between vendors and releases, a process controller typically consists of

modular hardware components. Key parts include I/O cards that connect to field devices such as transmitters, actuators, and process analyzers; a CPU card with memory to execute control algorithms and store data; and communication cards that interface with process networks and remote I/O chassis. It also includes a power supply (card) to supply and regulate power, and it might have battery back-up unit in case overall power fails. To enhance cybersecurity, some vendors add a micro firewall to protect access to the controller and regulate traffic load. These specialized firewalls can be hardware-based or built-in software firewalls. They may feature configurable filters for flexible security settings or fixed filters that sometimes autoconfigure based on the controller's configuration to enforce predefined protection rules. All modular components are typically housed in a backplane or chassis, often with options for various types of redundancy (power redundancy, controller redundancy, network redundancy, and I/O redundancy) to ensure high availability and performance.

Field devices are traditionally connected to I/O cards, either through analog cabling or network connectivity. Some vendors connect network-based field devices, such as those using Profibus and Foundation Fieldbus, to the level 1 network via gateways. This design choice is, of course, important for the attack surface of the field devices. Traditionally, I/O cards were specific to the type of connection, we had specific digital input cards, digital output cards, analog input cards, analog output cards, and counter input cards. While these are still encountered in legacy systems, newer generations of I/O cards are configurable, with the configuration determining the function rather than the hardware.

In earlier decades—the 1970s, 1980s, and 1990s—auxiliary cards were commonly used between the I/O card and the field device to implement specific functions. For example, auxiliary cards could restrict a valve's travel rate using a manually configured RC delay function. Today, these functions have been fully integrated into software, either within the controller or directly into smart field devices. While this integration reduces costs and increases convenience, it also expands the potential attack surface, creating new opportunities for skilled attackers to exploit.

Today, various devices are commonly used between the I/O card of a controller and a field device to ensure compatibility, protection, and safety. These include signal conditioners, signal limiters, surge protectors, isolators, intrinsic safety barriers, and Zener diodes, each serving a specific function:

· **Signal Conditioners:** Adjust, scale, or isolate signals to match the requirements of connected devices, ensuring accurate and interference-free communication.

· **Signal Limiters**: Restrict the signal to a safe, predefined range, preventing overdriving or damage to field devices like valves or actuators.

· **Surge Protectors**: Protect the I/O card and field device from voltage spikes or transient over voltages caused by lightning or electrical faults.

· **Isolators**: Electrically separate the controller from the field device to prevent ground loops and reduce signal noise or interference.

· **Intrinsic Safety Barriers**: Limit electrical energy to levels that cannot ignite flammable gases or dust, ensuring safe operation in hazardous environments.

· **Zener Diodes**: Clamp voltage to a safe maximum level by shunting excess voltage to ground, providing a simple and cost-effective overvoltage protection solution.

Marshalling is the process of organizing and managing the equipment and wiring between the controller's I/O and field devices. It includes components such as terminal blocks, signal conditioners, isolators, surge protectors, intrinsic safety barriers, and HART multiplexers (MUX) for integrating diagnostics and advanced monitoring. These devices, essential for system reliability and safety standards, were traditionally configured manually using physical potentiometers. However, many now include network functionality, enabling remote configuration and diagnostics, which simplifies maintenance and enhances reliability.

While this improvement offers convenience, it can also introduce cyber-physical security risks if not properly implemented. Modern marshalling centralizes signal management, ensuring proper adaptation, protection, and routing, while also supporting scalability, diagnostics, and streamlined maintenance. However, this comes with the trade-off of an increased attack surface, requiring robust security measures to mitigate potential vulnerabilities.

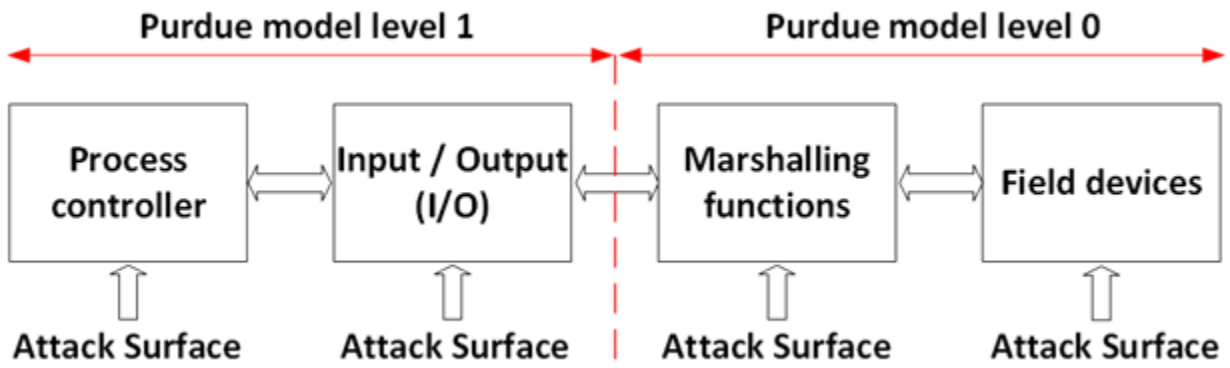

Figure 2 - level 1 and level 0 functions

The above diagram shows how an exposed attack surface provides opportunities for a threat actor to exploit potential vulnerabilities in the system. Protocols used for data exchange might not be secure, and configuration settings may lack proper protection, offering opportunities to cause damage to the process installation. However, it is important to note that such exploitation is only possible if the attack surface is exposed. OT security is not just about removing vulnerabilities; it is often more focused on reducing exposure. With a solid understanding of process automation, many vulnerabilities can be effectively shielded from potential attackers, minimizing the risk of exploitation.

What types of hazards can we identify as relevant to cyber-physical risks? After a successful exploit, an attacker could leverage the attack surface to:

· **Destabilize the process**, for example, by causing control oscillations through modifications to algorithm parameters in the controller. For example, altering tuning parameters (e.g., gain, integral, and derivative settings in PID loops), can cause process oscillations, inefficiencies, product quality issues, or equipment stress.

· **Alter a valve's travel rate**, potentially causing issues such as water hammer or process destabilization with an excessively high travel rate, or delays in critical actions with an excessively low travel rate. The travel rate settings may be exposed, depending on the system configuration, in the controller, marshalling functions, or the actuator itself.

· **Manipulate Sensor Readings**: Manipulating sensor data, such as temperature, pressure, flow rates, or level readings, can cause the controller to make incorrect adjustments, destabilizing the process, creating unsafe conditions, or producing off-spec products. This manipulation can occur at various points in the system. At the controller level, data can be intercepted or altered as it is processed, leading to false process values. At the marshalling level, signals can be tampered with during conversion or scaling, such as modifying a 4-20 mA signal to misrepresent actual values. At the transmitter level, attackers might alter the device configuration, or calibration, to instill false readings directly into the system.

· **Override Interlocks and Permissives:** Modify or disable interlocks or permissives. Interlocks prevent unsafe actions by triggering protective responses during unsafe conditions, while permissives ensure that specific safety conditions are met before allowing an operation to activate. Disabling or modifying these safeguards in the process controller could lead to equipment damage, process instability, or hazardous situations by allowing operations to proceed under unsafe conditions.

· **Change Setpoints**: Adjust critical setpoints in the controller, such as pressure, temperature, level, or flow rates, beyond safe operating limits, potentially causing equipment failure, chemical reactions, or unsafe process conditions.

· **Disable, Corrupt, or Improperly Actuate Actuators**: Prevent actuators (e.g., valves, dampers, or pumps) from responding to commands or improperly actuate them (closing or opening valves), causing disruptions in material flow, cooling, or other critical processes. This could lead to process instability, equipment damage, loss of containment, and other unsafe conditions.

· **Generate Excessive Alarms**: Overload the system with excessive alarms, overwhelming operators, obscuring critical issues, and delaying response times during important events. Unlike SCADA systems, where alarm settings are managed centrally, in DCS environments, alarm limits typically reside within the process controllers.

· **Introduce Latency or Delays**: Inject delays in communication between controllers and field devices, disrupting real-time control and monitoring of the process.

· **Open Unauthorized Pathways**: Exploit a manifold control system to open or close valves in a coordinated manner, creating unintended flow paths. This could lead to contamination, overflow, or mixing of incompatible substances, disrupting batch integrity and process safety.

· **Tamper with Utility Systems**: Interfere with utilities like compressed air, steam, or cooling water supply, affecting overall process stability and potentially leading to dangerous conditions.

· **Compromise Diagnostic and Maintenance Functions**: Corrupt diagnostic data or maintenance tools, delaying identification of faults and leading to prolonged exposure to hazards.

This may seem like a long list, but it's just the tip of the iceberg. Once an attacker gains access to the attack surface, a wealth of possibilities opens up. While causing targeted damage typically requires an understanding of the process, even random changes can, in many cases, result in serious harm.

Process controller software functions.

Now, let's focus on the software. Essentially, anything that can be hacked includes software, whether it's firmware embedded in a device, or a software program downloaded onto it. Additionally, software depends on data—either for operational processing or configuration settings. Both the software and the data can be exploited by an attacker to achieve their goals. In the hardware section, I discussed components such as I/O, marshalling functions, and field devices. Here, I will narrow the discussion to the software functions of the process controller.

I previously mentioned that a process controller utilizes algorithms (e.g., a PID control algorithm), which are part of what is referred to as a process loop. Process loops consist of function blocks that, at a minimum, define the input, the control algorithm, the output, the alarm configuration, and the block scheduling. Process loops are identified by a tag name; for example, in our boiler example in Figure 1, the level control loop (1) could be labeled 10LIC100 (where 10 indicates boiler unit 10, LIC stands for Level Indicator Control, and 100 serves as a sequence number for multiple level sensors). Similarly, a feedforward loop (2) could be labeled 10FF100, and a flow control loop (3) might be identified as 10FIC100. Each of these loops would have their input sources specified, and output destinations defined. An alarm block would specify the various alarm levels and priorities, while another block defines the scan rate. Let's take a closer look at how control loops (often referred to as tags) are scheduled.

In process automation, the sampling rate of a process signal must adhere to the Nyquist-Shannon Sampling Theorem. This theorem states that to accurately reconstruct a signal from its samples, the sampling rate must be at least twice the highest frequency present in the signal. Therefore, the sampling speed of a control loop is chosen based on the frequency of the signal being measured.

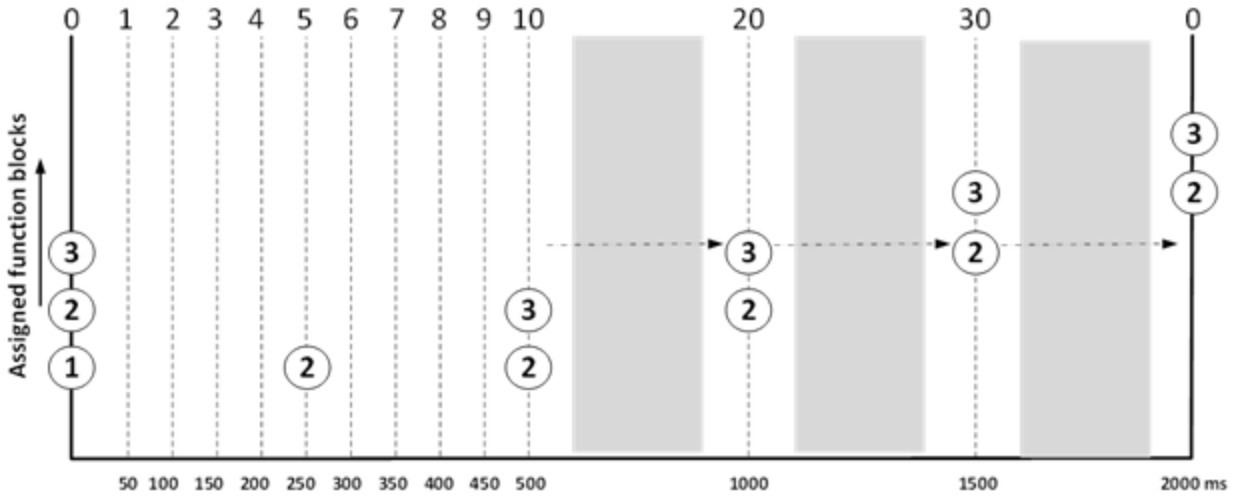

Figure 3 – Controller execution cycles and phases example

For example, flow and pressure loops require much higher sampling rates than temperature or level loops due to their more dynamic and rapidly changing signals. To handle the loop scheduling challenge in a DCS, most vendors implement a system of execution cycles subdivided into execution phases (See Figure 3). This structure ensures that tags are scheduled not only with specific sampling rates but that we can also sample values of different control loops in a specific sequence. Referring back to Figure 1, we see that the 10LIC100 loop (1) must be executed before the 10FF100 loop (2). If executed in reverse order, the 10FF100 loop would calculate its output signal based on "old" data from the previous execution cycle. Similarly, the 10FIC100 loop (3) must be executed after the other two loops to maintain correct process logic. Failing to follow this order could result in less stable control, leading to increased energy costs and possibly off spec product quality.

However, not all loops require the same sampling frequency. For instance, the level loop (1) doesn't require a high sampling rate and is scanned once every 2 seconds, whereas the feedforward loop (10FF100) measures a steam flow rate and the 10FIC100 measures the feed water flow rate—both signals with inherently higher change rates. In this example, the level loop (1) is scanned every 2 seconds, while the flow loops are scheduled to execute every 500 milliseconds. Without this differentiation, the measured signals might not represent the actual process values, leading to process instability.

This example illustrates a simple boiler control strategy, but in reality, a modern process controller can support hundreds of loops. Each execution phase typically has multiple loops assigned to it. However, the performance of any controller is constrained by its CPU processing power, available memory, and I/O communication speed. For example, if loops 1 and 2 are located within the same controller, data transfer occurs in memory, which is very fast. But if the loops reside in different controllers, communication must occur over the network, which is significantly slower.

To manage the performance in a process controller each loop is allocated processing units, memory units, and I/O units, all of which are constrained by the capacity of the process controller. If the combined resource demand of the loops in a single execution phase exceeds the controller's capabilities, an additional process controller must be deployed. Depending on the plant size and the process controller capabilities, this can involve managing anywhere from a handful to over 100 process controllers.

When we talk about real-time control, it is important to understand that each execution phase operates on a strict schedule. For example, if the process controller, of my example, has 50 milliseconds to complete all tasks in phase 3, it must finish its work within that time because the task list for phase 4 will begin immediately after the 50 milliseconds of the previous phase. Execution phases cannot borrow time from one another; if the task list for a phase cannot be completed within its allocated time, the remaining tasks of that phase are dropped.

This is a fundamental difference compared to the time-shared method used by operating systems like Microsoft Windows. Time-shared operating systems, such as Windows and Linux, are designed for fairness and multitasking, ensuring all processes or users receive a reasonable share of CPU time. As a result, response times can fluctuate—sometimes the system is very busy, while at other times it has available capacity. This is very different from real-time systems, which are designed for determinism, ensuring tasks are completed within strict timing constraints.

One potential cyber hazard for process controllers is causing intentional or accidental task execution delays that cause this starvation effect, where tasks at the end of a phase's list are repeatedly skipped. If this occurs multiple times in succession, it leads to process instability and operational issues. Additionally, this can result in functional hazards, such as control mode shedding, which will be discussed further in the context of control hazards.

As long as the control strategy remains fully internal to the process controller, these delays do not occur. However, as soon as the process controller needs to collect values from field devices or exchange data with peer control loops residing in another controller, latency is introduced. For I/O operations, this is partially mitigated by prefetch mechanisms, but depending on the technology used and the system's design, these delays must be carefully considered, particularly if cyberattacks could influence or worsen them. A field transmitter directly wired to a controller's analog input I/O card will not introduce delays. However, a Profibus or Profinet field transmitter can introduce delays, and depending on the system design, an attacker could affect these delays to disrupt operations.

The situation becomes even more critical when dealing with peer-to-peer communication between different controllers. This communication traverses the process control network and is subject to latency introduced by network switches, routers, and, in some cases, firewalls. Although Quality of Service policies are implemented to prioritize commands, process alarms, and process data over general traffic, these measures can only partially mitigate delays. A certain level of network latency remains unavoidable, which impacts a controller's workload and overall system performance.

Control systems that utilize TCP/IP and Ethernet are particularly vulnerable to network performance disturbances. If an attacker gains access to the Level 2 or Level 1 segments of the control network, they can exploit various techniques to degrade network performance, placing the entire system at significant risk. These vulnerabilities underscore the need for robust network segmentation and comprehensive security measures.

Controller traffic encompasses more than just peer-to-peer communication between controllers. Controllers must also provide process information, system data, and alarms to process operators. In addition, an advanced control layer may exist above the primary control tasks executed by the process controllers, requiring continuous data exchange. Controllers often need to periodically (typically every 30 minutes) checkpoint their data to a storage device for backup purposes, and redundant controllers must communicate regularly to stay synchronized. This multifaceted communication further amplifies the criticality of network performance and network segmentation.

Today's operator HMI systems are far more information-intensive than those from the early days of DCS. Modern HMIs provide a comprehensive array of real-time data, advanced visualizations, and analytical tools, far surpassing the basic displays and controls of earlier systems. For example, a process overview graphic offers operators a live view of valve statuses, process values, and control modes, all sourced directly from the process controller. This enhanced access to information supports better decision-making and more effective process monitoring.

However, increased network latency reduces the amount of information that can be provided per second, impacting the responsiveness and effectiveness of the HMI. This is one reason why placing a firewall between Level 1 (where the controllers reside) and Level 2 (where the HMI resides) is not

an ideal solution, as it introduces delays that limit the data rate. A more effective approach is to implement firewall functions at the controller level, which distributes the impact on network performance and minimizes delays.

Mode shedding.

A final topic related to controllers, which I briefly mentioned at the beginning, is mode shedding. A control loop operates in what is referred to as a control mode, which defines how the loop responds to inputs and outputs. There are four commonly defined control modes:

· **Manual:** The algorithm is no longer active, the process operator directly adjusts the control output, bypassing the automatic functionality of the loop.

· **Automatic:** The process controller manages the output based on its internal algorithm, such as a PID controller, to maintain the process variable at the setpoint.

· **Cascade:** The loop operates in coordination with another loop, where the output of one loop serves as the setpoint for another. As is the case in our example where the 10LIC100 loop provides the setpoint of the 10FF100 loop.

· **Computer Cascade:** The loop is managed by a higher-level system, such as an advanced process control or optimization application, which dynamically adjusts the setpoint (Setpoint Control (SPC) or even direct the output (Direct Digital Control (DDC)).

Mode shedding occurs when the connection between the primary and secondary loops in a cascade or computer cascade control configuration is lost, preventing the secondary loop from receiving updated data from the primary loop. For example, if 10LIC100 and 10FF100 reside in two different controllers and communication between these controllers is disrupted, the primary loop (10LIC100) may fail to update the setpoint for the secondary loop (10FF100) multiple times. As a result, the 10FF100 loop, originally operating in cascade mode, transitions—or "sheds"—to either the automatic or manual mode, depending on the configuration.

In automatic mode, the process operator can assume the task of the primary loop by manually adjusting the setpoint. In manual mode, the operator takes full control of the output, bypassing the controller's algorithm entirely. This mechanism ensures continued operation but introduces a greater reliance on operator intervention.

In the case of a computer cascade configuration, a similar phenomenon occurs. Here, the primary loop resides in the advanced control system, and if communication with the controller is disrupted, the same effect is observed: the controller sheds to either automatic or manual mode, ensuring continued functionality while requiring operator involvement to maintain process stability.

Typically, if mode shedding occurs in computer cascade configurations, there is no immediate danger to the process. However, it can lead to less efficient use of materials and energy, along with potential product quality issues. In contrast, cascading failures at the basic control level can have far more serious consequences. Without prompt operator intervention, such failures may result in process instability and, in extreme cases, pose safety risks. If safety risks are detected, the safety instrumented system (SIS) will intervene automatically and initiate an emergency shutdown to protect the system and prevent further hazards.

So again, the network performance is a crucial element, we have cyber attack techniques that can cause mode shedding to all cascade loops in level 1 process controllers simultaneously. Depending on the criticality of the process this can be a serious problem. Of course also in this case the SIS will save the day, but if we consider a situation like we had in the Triton / Trisys attack where the attacker crashed the SIS, he could have very simply caused overall mode shedding, which depending on the process, could have had very serious consequences.

In petrochemical facilities, such as the Petro Rabigh site, ethylene and propylene production processes pose significant cyber-physical risks due to their reliance on precise control under high temperatures and pressures. If an attacker simultaneously crashed the SIS and forced all control modes to shed, the consequences could have been much worse. DCS systems are designed to prevent runaway reactions, overpressures, and overheating, which could lead to explosions or the release of hazardous vapors. Without the SIS to act as a safety net or coordinated control loops to stabilize the process, operators would be forced to intervene manually.

This situation places an immense burden on process operators, who may not be fully familiar with all process details under such conditions. Under normal circumstances, the process is almost fully automated, with advanced control loops steering operations and minimizing the need for frequent manual intervention. The sudden loss of automation forces operators to make critical decisions in a high-pressure environment, potentially without the detailed situational awareness or experience required to stabilize these highly dynamic processes effectively. The complexity of the system, combined with the rapid pace of destabilization, makes preventing escalation extremely challenging.

This highlights not only the critical importance of securing safety systems and control functions against cyber threats but also the need for enhanced emergency preparedness, including robust operator training, process transparency, and decision-support tools. Orchestrated attacks, where multiple targets are simultaneously attacked, create a new challenge. Preparing operators to respond effectively to cyber-induced scenarios is essential to mitigate risks and ensure system stability in high-stakes situations