Self-reconfiguration of industrial control systems as a response to cyberattacks

Jolahn Vaudey, Gwenaël Delaval, Stéphane Mocanu, Éric Rutten

Self-reconfiguration of industrial control systems as a response to cyberattacks

Jolahn Vaudey Stéphane Mocanu
Gwenaël Delaval Eric Rutten
Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38 000 Grenoble, France*
**Institute of Engineering Univ. Grenoble Alpes*
firstname.lastname@inria.fr

***Abstract*—As industrial control systems become increasingly connected, the threat of cyberattacks grows in turn. Classical IT reactions that prioritize confidentiality, like network isolation, cannot be applied as they lead to a loss of availability, hence an issue of safety criticality. This work proposes a reconfiguration-based reaction to attacks, migrating control programs away from compromised components. This reconfiguration is carried out by a controller which solves a constraint programming (CP) problem whenever a compromised device is detected. This controller is automatically generated based on a model of IEC 62443 compliant systems. This approach is tested both on generated models of arbitrary size and to control a set of real Programmable Logic Controller (PLC) overseeing a small-scale training factory.**

I. Introduction

A. Cybersecurity concerns in ICS

Industrial control systems (ICS) consist of an interconnection of hardware and software components controlling a physical industrial process. This is usually orchestrated by a network of Programmable Logic Controller (PLC), rugged computers commonly used in the industry. For historical reasons, these systems communicate internally using mostly insecure protocols, such as Modbus, and for a long time this was not an issue as accessing the system required physical on-site presence. However, ICS nowadays are increasingly connected, and this has lead to the rise of security concerns and cyberattacks [1]. In turn, due to high profile attacks, especially Stuxnet in 2010, awareness of these issues has spread, and research about attack detection, prevention or reaction in ICS is a hot topic. Compared to traditional IT systems, Operational technology (OT) ones have different security priorities. Using the terms of the CIA (Confidentiality, Integrity, Availability) triad, while traditional information systems focus on confidentiality, ICS put availability of the system above everything else. Indeed, as a physical process is involved, a lapse in its control can cause dramatic results, both financial, if the attacker manages to break key components, and potentially life threatening. To avoid such loss of control, whenever an attack is detected, the system should act in order to contain it and restore the affected functions.

This work has been partially supported by the French National Research Agency under the France 2030 label (Superviz ANR-22-PECY-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

B. Reconfiguration

In this paper, we focus on the reaction to attacks on ICS. Specifically, we design systems able to reconfigure themselves while under attack. In the event that a component is compromised, the first reaction would be to isolate it. For ICS, as seen before, this would cause major safety issues, as a part of the process would become uncontrolled. Capitalizing on the increased virtualization of ICS [2], we argue that it is possible to migrate the tasks running on said component to another one. Thus, our goal is to design a reconfiguration controller, started whenever an intrusion is detected, able to find a new configuration by migrating tasks while maintaining as much functionality as possible.

C. Related works

While reconfiguration is not commonly used specifically in ICS security, numerous works apply such techniques to attain specific goals in cyberphysical systems. We will give an overview in increasing order of similarity with our subject. Systems can be repeatedly switched between multiple configurations as a moving target defense ([3]–[5]), aiming to prevent attackers from gaining and exploiting knowledge about the control scheme topology. Reconfiguration is also used to recover from fault in smart grids, for instance by advising new power lines to deploy [6] or by redistributing control functions over the system to deal with perturbations [7]. Another classical answer to compromised components is to isolate them from the network [8], by rerouting the communication flows. Do note that each functionality running on the system must still be maintained. Finally, control reconfiguration in response to attacks has been studied on car systems [5], using predefined backup controllers.

In our case, finding new configuration is modeled as an integer linear programming (ILP) problem, which can also be seen as a constraint programming (CP) problem, with our main decision variables being the task/device

allocations. This is a natural way of expressing optimal resource placement under constraints in general. These methods have been employed recently to distribute control functions to servers in a smart grid [7], or in a completely different context to allocate legends to digital highway signs, regulating traffic in response to an accident [9].

II. Resilience via reconfiguration

A. IEC 62443 Configuration model

The first step in our approach is to develop a model for security of the ICS. We rely on latest cybersecurity IEC standard.

1) IEC 62443 standard: A standard was recently developed to create a security oriented technical specification for industrial automation and control systems, titled IEC 62443. We will only present the first part, which is relevant for this paper [10]. This first document contains the definitions of terms and abbreviations used in the rest of the standard, a description of the current situation in ICS and overall trends, OT cybersecurity concepts, and several models used to represent industrial control systems. We are mostly interested in these models, as we will use them to describe instances of ICS.

In the standard's system model, the plant is divided in *zones*. The components within a zone can be physical devices, informational assets, or applications. Applications need to be run on a compatible device in the same zone. A zone can be physical if the assets share a physical location, or it can be a purely virtual construct. A zone can be considered trusted or untrusted (if it connects to the outside, typically). We will assume that components sharing a zone can communicate freely within the zone's bounds. Each communication between two zones must pass through *conduits*.

The mentioned security requirements are a vector of seven numbers, ranging each from 0 (no protection) to 4 (protection against attackers with extensive resources and knowledge) and corresponding to *foundational requirements*: Access Control, Use Control, Data Integrity, Data Confidentiality, Restricted Data Flow, Timely Response and Resource Availability. This can be understood as a refinement of the CIA triad for ICS.

Conduits are technically zones, set of components sharing a needed security level vector, only wholly dedicated to communications. They are defined by the zones they link. They can be contained within a zone, and cannot contain subzones. A conduit may also include a set of channels, indicating the specific communication links between components. They are compared to cables (channels) within a pipe (conduit) in the standard. Channels are defined by the set of components they link. Conduits can be considered trusted/untrusted by the zones they connect, if their security level is sufficient/insufficient compared to the zone's target level. For now, this has no bearing on the reconfiguration.

For zones and conduits to reach their target security level, the passive capability of their components may not be enough. In this case, countermeasures are added to raise the achieved security level. Those can be technical (devices or software), administrative (usage policies) or physical (locked areas).

2) Reconfiguration problem and hypotheses: Our first working assumption is that the system under control is compliant with IEC 62443, and thus organized in zones that contains applications and devices. We suppose that the attacker will try to compromise host devices or applications, thus components like remote Input/Output units that do not execute arbitrary code will not be considered for attacks. Whenever a device or application is compromised, it will be isolated and the remaining applications will be redistributed on the rest of the system to maintain as much availability as possible. In the default behavior, stopped applications cannot be restarted by this reconfiguration process, as a restarting with default values could lead to hazardous situations. We also assume that, for now, devices can't be moved to a different zone by the controller. In the case of a fully virtual ICS, this hypothesis could be relaxed.

3) System model: We developed a model that describes the parts of IEC-62443 compliant plants that are relevant to the task migration.

The model consists of a set of zones Z (which includes conduits), a set of applications A and a set of devices D. To express meaningful constraints to the reconfiguration, several properties were added to applications and devices. Applications have a size property while devices have a capacity, the maximum total size of applications it can run. This represents a limitation, CPU or storage size needed/available for instance. Applications also have a score formula, that determine how much abstract "gain" the application give when executed, taking into account which device it runs on. They also have set of other components (device sand applications) they need to communicate with in order to be executed, and this communication may only pass through, at most, one conduit. In a formal manner, an instance of our model consists of a tuple of three sets (Z, A, D), along with the following functions:

- $\mathbf{adj} : Z \to 2^Z$, with $\mathbf{adj}(z)$ the set of all zones adjacent to z, meaning z itself and any zone connected through one conduit.
- $\mathbf{dep} : A \to 2^A$, with $\mathbf{dep}(a)$ the set of all applications that a needs to communicate with.
- $\mathbf{dev} : Z \to 2^D$, with $\mathbf{dev}(z)$ the set of all devices in zone z.
- $\mathbf{on} : A \to D$, with $\mathbf{on}(a)$ the device currently running application a.
- $\mathbf{size} : A \to \mathbb{N}$, with $\mathbf{size}(a)$ the size of application a.
- $\mathbf{cap} : D \to \mathbb{N}$, with $\mathbf{cap}(d)$ the capacity of device d.
- $\mathbf{loc} : D \to Z$, with $\mathbf{loc}(d)$ the zone d resides in.

- **score** : $A \times D \rightarrow \mathbb{N}$, with $\mathbf{score}(a, d)$ the score obtained by application a running on device d.
- **comp** : $A \rightarrow 2^D$, with $\mathbf{comp}(a)$ the set of all devices that can run application a, taking every static constraint into account.

As an example, we will give the tuple corresponding to the system illustrated in Figure 1. Scores, size and capacity are not defined in the illustration. We will suppose that each application has a size of 1, each device a capacity of 2, except d_4 which has a capacity of 0. Applications get a score of 3 on their original device, 2 on others.

- $Z = \{z_1, z_2, z_3, z_4, z_5\}$
- $D = \{d_1, d_2, d_3, d_4\}$
- $A = \{a_1, a_2\}$

We will also give the functions' graphs for this instance:

- $\mathbf{adj} = \{z_1 \mapsto \{z_1, z_2, z_3, z_4, z_5\}, z_2 \mapsto \{z_1, z_2, z_4\}, z_3 \mapsto \{z_1, z_3, z_5\}, z_4 \mapsto \{z_1, z_2, z_4\}, z_5 \mapsto \{z_1, z_3, z_5\}\}$
- $\mathbf{dep} = \{a_1 \mapsto d_3, a_2 \mapsto a_1\}$
- $\mathbf{dev} = \{z_1 \mapsto \{d_1\}, \ldots, z_4 \mapsto \{d_4\}, z_5 \mapsto \emptyset\}$
- $\mathbf{on} = \{a_1 \mapsto d_1, a_2 \mapsto d_2\}$
- $\mathbf{size} = \{a_1 \mapsto 1, a_2 \mapsto 1\}$
- $\mathbf{cap} = \{d_1 \mapsto 2, d_2 \mapsto 2, d_3 \mapsto 2, d_4 \mapsto 0\}$
- $\mathbf{loc} = \{d_1 \mapsto z_1, d_2 \mapsto z_2, d_3 \mapsto z_3, d_4 \mapsto z_4\}$
- $\mathbf{score} = \{((a_1, d_1), 3), ((a_1, d \neq d_1), 2), ((a_2, d_2), 3), ((a_2, d \neq d_2), 2)\}$
- $\mathbf{comp} = \{(a_1, \{d_1, d_3\}), (a_2, \{d_1, d_2, d_3\})\}$

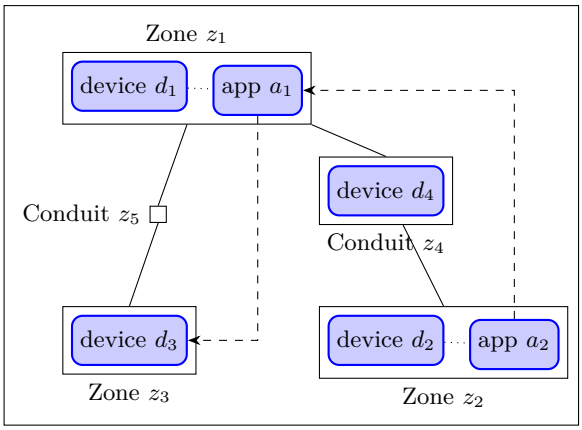

Fig. 1. Simple example of a system divided in zones. Arrows represent dependencies, dotted lines represent current execution location.

Such models are used as input in the reconfiguration process, representing the initial configuration. It is modified whenever a device is detected as changed, by removing from D the compromised component, and from A any application that needed to communicate with it (or with another application removed this way, recursively). Afterwards, the reconfiguration in itself takes place, derived from the current model.

B. Optimization program

The reconfiguration process is modeled as an optimization problem, and managed by an integer linear program (ILP) / constraint program (CP) (The equations can be expressed in both paradigms). The goal is to find a new valid configuration, maximizing the overall application scores.

Three sets of boolean decision variables are created for the program, with the following meanings:

- $run_{a,d}$: The application a run on device d.
- $none_a$: True if application a is not running on any device.
- $in_{a,z}$: The application a is within zone z.

The objective of this optimization problem is to maximize the sum of all applications' scores.

$$\text{Maximize} \left(\sum_{a \in A} \sum_{d \in D} \mathbf{score}(a, d).run_{a,d} \right) \tag{1}$$

This objective is maximized under the following constraints. A given application a may only run on at most one compatible device, and if it is not attributed, $none_a$ is set to true.

$$none_a + \sum_{d \in \mathbf{comp}(a)} run_{a,d} = 1 \qquad \forall a \in A \tag{2}$$

Devices cannot run more applications than it's capacity allows.

$$\sum_{a \in A} (run_{a,d} * \mathbf{size}(a)) \leq \mathbf{cap}(d) \qquad \forall d \in D \tag{3}$$

Equation (4) express that an application is in a zone, if it executes on a device that is within the zone.

$$\left(in_{a,z} - \sum_{d \in \mathbf{dev}(z)} run_{a,d} \right) = 0 \qquad \forall a \in A, \forall z \in Z \tag{4}$$

Two applications that need to communicate cannot be attributed to zones that are not adjacent.

$$in_{a,z} + in_{a_2,z_2} \leq 1 \qquad \begin{array}{l} \forall a \in A, \forall a_2 \in \mathbf{dep}(a), \\ \forall z \in Z, \forall z_2 \in (Z \backslash \mathbf{adj}(z)) \end{array} \tag{5}$$

If an application a has to communicate with another application a_2, and a_2 is not running, a needs to be stopped as well.

$$none_a - none_{a_2} \geq 0 \qquad \forall a \in A, \forall a_2 \in \mathbf{dep}(a) \tag{6}$$

If we note n the number of applications, m the number of devices and p the number of zones in the studied system, the program will contain at most $1+n+m+np+p^2n^2+n^2$ constraints (with each of these 6 numbers the maximum amount of constraints corresponding to the respective equation), so creating this program has a complexity of $O(m + n^2p^2)$.

Once this program is solved, the model is updated again based on the resulting decision variable values. For each

application a, if $none_a$ is true, a is removed from the set of applications A. If $none_a$ is false, $\mathbf{on}(a)$ is set to d, with d the only device for which $run_{a,d}$ is true.

III. Experimental setup

A. Training factory setup

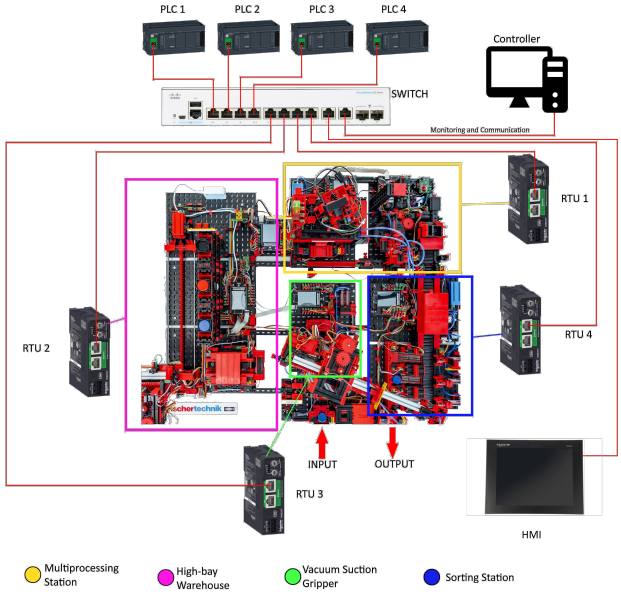

Fig. 2. Training factory setup

In order to have a physical use case, a Fischertechnik training factory[1] was mounted. It imitates the way a plant operates, and is separated in four different modules. One of them manages the input/output of the system, and the gripper that moves resources across modules. Resources are colored items that can be sorted using color sensors. The other modules manage the raw resource storage, the processing unit and the sorter, respectively.

The control program is divided in four parts corresponding to each module, and each PLC is able to reconfigure itself and migrate its program's state to other devices when instructed to by a Modbus command. The four PLC have the full control code loaded, with each part active on at most one PLC at a time. The sensors and actuators are delegated to remote terminal units (RTU). These RTU do not run any control program, and we will assume that they can't be compromised by a distant attacker. The setup is illustrated in Figure 2. Due to the dynamics of the physical process carried out by the factory, the control system's cycle time should be low, and thus was set to 10ms.

B. Model and reconfiguration details

The system is split in four zones, corresponding to the modules. Each contains one RTU, the PLC that initially run the module's control program, and the program itself. The vacuum gripper should be able to communicate with every other programs, and the multiprocessing station should communicate with the sorting station. Additionally, the gripper program must access the sorting station RTU to read specific values. In our model, the conduits are therefore defined based on these initial needs, leading to the layout found in Figure 3. It is assumed that no communication may cross over more than one conduit. We chose to give every zone the same security level, as the number of possible reconfiguration is already small. Due to the strict timing constraints, migrating tasks away from the zone they control is discouraged. This translates to lower application scores on PLCs that are not the original location, and encourage fast reconfiguration. In this example, we state that each PLC may only execute up to two applications due to application size constraints.

[1] https://www.fischertechnik.de/en/products/industry-and-universities/training-models/554868-training-factory-industry-4-0-24v

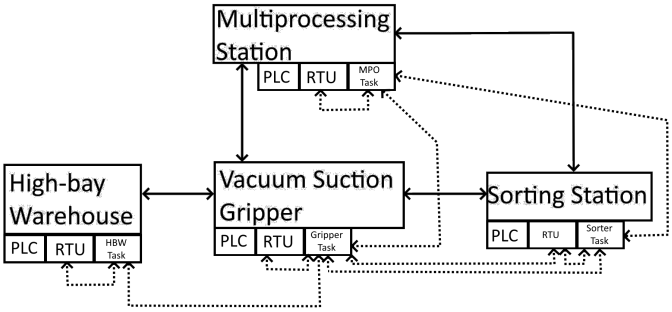

Fig. 3. Zone/conduit layout. Conduits shown with solid lines, applications dependencies with dotted lines.

C. Reconfiguration controller

The reconfiguration controller takes as input the system description to initialize the current configuration. The controller reacts whenever a device is compromised, marking it as such and starting the reconfiguration. A new constraint program is created as defined before, removing compromised devices and applications that needs to communicate with them, and aiming to find a new configuration with maximal score. This new CP is then solved using the OR-Tools CP solver [11]. The resulting values gives, for each application that was not stopped beforehand, whether it is now stopped ($none_a$), and if it still runs, which device executes it (the only device d such that $run_{a,d}$ is true). The new allocation of tasks is computed, and a message is sent to the old and new location to reflect this change. On the other hand, if the application was stopped, a Modbus message is sent to every PLC, halting any remaining parts of the process that interacts with it. Note that halting an application means putting the controlled process in a safe mode.

D. Experimental protocol

We are mainly interested in the measurement of the time elapsed between the attack being signaled and the

reconfiguration controller output. Any other non-temporal criterion should be able to be integrated in the score calculation, and the time elapsed between the attack and its detection is IDS dependent. The reconfiguration can be divided in two phases: solving the CP, and communicating the new solution to the set of PLC.

The controller is run on a DELL precision 7770 laptop, with 128GB of RAM and a i9-12950HX CPU with up to date drivers as of January 2024. Other non-essential applications are stopped. The solver libraries for CP are loaded prior to measurements.

On the other hand, the PLCs and RTU controlling the training factory are powered on, and connected to the controller through a switch, as seen in the setup in Figure 2.

The following is repeated N=200 times:

- Define a random PLC as attacked, and start the timer
- Launch the CP solver to get a configuration without this PLC.
- Get the time elapsed since the timer started to measure the reconfiguration computation time.
- Communicate the new configuration to the PLCs using an implementation of the Modbus TCP protocol.
- Get the time elapsed since the timer started, to get the total reconfiguration time.
- The controller sends a signal to each PLC to reset the programs to their initial locations.

We also wanted to measure the reconfiguration's performance on larger systems. To do so, we created random instances of varying size (from n=4 to 64 zones, growing in increments of 4, with each zone containing 2 devices and 4 applications). The zones are connected by conduits in a tree-like fashion, as illustrated in Figure 4, roughly imitating a SCADA layout. To further imitate such systems, in every zone, initially, one of the applications needs to communicate with the other 3 within its zone and with one of the program within the parent zone. The security levels and applications scores are randomly generated, with the score for a given application maximized if it runs on the device it started in, minimized if it runs on one in a different zone. Device scan only run two applications, so every reconfiguration will lead to stopping at least two applications. For every n tested, 10 different systems were generated.

E. Preliminary results

The results obtained using the training factory are shown in Figure 5. Two main information can be gathered from these. Firstly, solving the constraint program was always faster than applying the results. Each PLC processes its instructions a cycle after they are received, thus a delay of about 10ms is to be expected. As the current implementation is not parallelized, the obtained mean of 34ms is not suprising. As communication is a low priority tasks for the PLC compared to local control, it

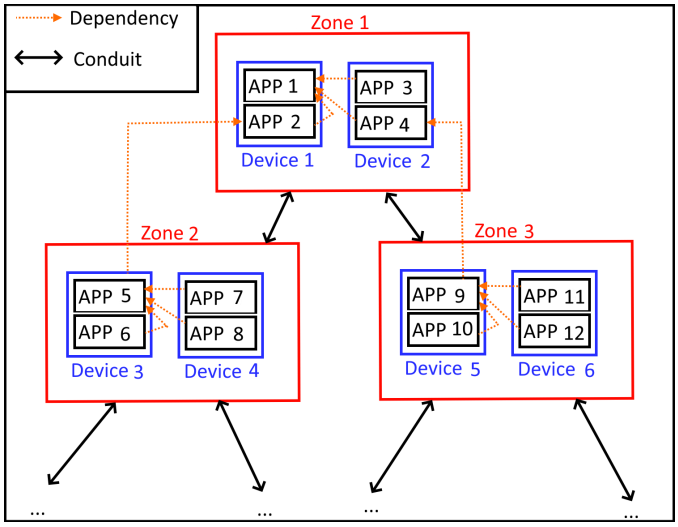

Fig. 4. Organization of the generated systems, with n zones, $2n$ devices and $4n$ applications. The tree is filled from left to right, breadth-first.

is sometimes postponed by a cycle, potentially explaining the worst cases. Enhancing communications will be a priority if we get a larger use case, as treating each of them sequentially will considerably lengthen the control delays. Secondly, the mean reaction time is less than a PLC cycle time, while the worst lasted two cycle times. Almost nothing can happen in such a short time, and if the communication delays were improved, it's almost as fast as the PLC in the system can react anyway. Of course, this level of speed is only attainable because the studied ICS is very small.

	min	mean	max
computation	3	8.1±0.31	20
communication	15	34.3±1.71	69

Fig. 5. Minimum, mean and maximum elapsed time (in ms) for the reconfiguration on the training factory, using a sample size of N=200, and a 95% confidence interval.

For larger generated systems, the results can be found in Figure 6. Keep in mind that the graph follows a logarithmic scale on the y axis. Here, the computation times for small systems seems to be around 10ms, which corroborates the results from the use case. While generally the constraint satisfaction problem is NP-hard, the performances observed here are not quite exponential, thanks to the heuristics used by the solver. However, the increase in computation time is still drastic, reaching a hundred seconds with 256 applications, and for huge systems the variance increases considerably.

This is an extreme case: in most actual ICS systems applications cannot be redistributed to almost any parts of the system and the ratio of zones to applications is much lower. Still, this means that huge control systems cannot be reconfigured quickly by solving such models at runtime,

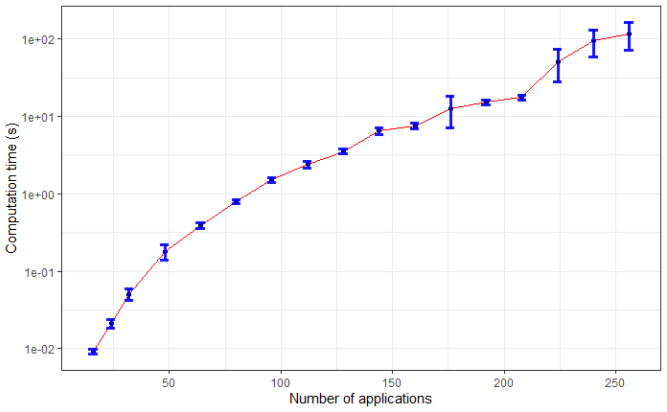

Fig. 6. Graph showing the evolution of the CP solver resolution time as a function of the number of applications in the system. The y-axis follows a logarithmic scale. Each point is computed as a mean of 10 values, and a 95% confidence interval is provided.

and we will need to find alternatives for such cases. In specific ICS managing a slow physical process, taking a few seconds to compute a new configuration could be acceptable, especially if the changes need to be validated by a human operator.

IV. Conclusion and perspectives

In conclusion, we have:

- Devised a method to reconfigure IEC 62443 compliant ICS while under attack, using constraint programming.
- Applied the technique to a physical use case, as well as to generated models of varying size.
- Obtained preliminary results showing a promising reconfiguration speed in small systems, and the problem solved is still tractable even for much larger setups.

As for perspectives, the reconfiguration process could be improved in several manners. First of all, communicating the new layouts using Modbus creates a huge vulnerability in the system, as this protocol was not designed with cybersecurity in mind. The OPC UA standard could be used to design a more secure architecture. For the configuration time to scale, it would also be necessary to parallelize the controller's network communications. To study the effectiveness of our optimization programs for large models representing systems with real time constraints, we could also evaluate the quality of solutions obtained while under a strict time limit. The metric could be a score loss percentage when compared to the optimal. Some other changes are considered to enhance performances. The CP model could be updated instead of recreating it for each step, even if the gain would probably not change the time scale. Indeed, building the model is at worst quadratic in the number of applications, while solving it is exponential. We could try to compute all different configurations offline, and store the computed results, avoiding the solving cost at runtime. If an hypothesis stipulates that the number of devices that could be compromised at the same time is at most n, the solver can be run statically on the model with all combinations of 1 to n different devices removed. Some of our hypothesis could also be relaxed: right now, stopped applications cannot be restarted by the controller, as going back to default values could be dangerous. In case it is not, this restriction could be removed without changing the CP model. On another hand, if the system's network architecture is virtual, we could also allow for devices to be moved from one zone to another. However this would drastically increase the number of constraints in the program, as determining in which zone z an application runs (equation 4) is much more costly when $\mathbf{dev}(z)$ is not statically determined. Lastly, some models could be divided in independent subsystems with their own controllers. This would have a high impact on computation times due to the potentially exponential complexity, and could be used to apply a faster control in specific parts.

References

[1] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A.-R. Sadeghi, M. Maniatakos, and R. Karri, "The cybersecurity landscape in industrial control systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1039–1057, 2016.

[2] T. Cruz, R. Queiroz, P. Simões, and E. Monteiro, "Security implications of scada ics virtualization: Survey and future trends," in *Proc. 15th Eur. Conf. Cyber Warfare Security (ECCWS)*, 2016, pp. 74–83.

[3] A. Kanellopoulos and K. G. Vamvoudakis, "A moving target defense control framework for cyber-physical systems," *IEEE Transactions on Automatic Control*, vol. 65, no. 3, pp. 1029–1043, 2019.

[4] P. Griffioen, S. Weerakkody, and B. Sinopoli, "A moving target defense for securing cyber-physical systems," *IEEE Transactions on Automatic Control*, vol. 66, no. 5, pp. 2016–2031, 2020.

[5] B. Potteiger, Z. Zhang, and X. Koutsoukos, "Integrated instruction set randomization and control reconfiguration for securing cyber-physical systems," in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, 2018, pp. 1–10.

[6] S. B. Meskina, N. Doggaz, M. Khalgui, and Z. Li, "Reconfiguration-based methodology for improving recovery performance of faults in smart grids," *Information Sciences*, vol. 454, pp. 73–95, 2018.

[7] S. Chehida, K. Fellah, E. Rutten, G. Giraud, and S. Mocanu, "Model-based self-adaptive management in a smart grid substation," in *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2023, pp. 1–8.

[8] H. Sándor, B. Genge, Z. Szántó, L. Márton, and P. Haller, "Cyber attack detection and mitigation: Software defined survivable industrial control systems," *International Journal of Critical Infrastructure Protection*, vol. 25, pp. 152–168, 2019.

[9] J. Verbakel, J. Van Meurs, J. Van De Mortel-Fronczak, W. Fokkink, and J. Rooda, "Legend pattern calculation for dynamic traffic management using ILP," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2023, pp. 1–7.

[10] "IEC 62443 part 1-1: Terminology, concepts and models," International electronical commission, Tech. Rep., 2009. [Online]. Available: https://webstore.iec.ch/preview/info_iec62443-1-1%7Bed1.0%7Den.pdf

[11] L. Perron and F. Didier, "CP-SAT," Google. [Online]. Available: https://developers.google.com/optimization/cp/cp_solver/