Chapter 1

ANALYZING CYBER-PHYSICAL ATTACKS ON NETWORKED INDUSTRIAL CONTROL SYSTEMS

Béla Genge, Igor Nai Fovino, Christos Siaterlis and Marcelo Masera

Abstract The fact that modern Critical Infrastructures (CI) depend on Information and Communication Technologies (ICT), is well known. Although many studies have focused on the security of SCADA systems and protocols, today we still lack an efficient way to conduct experiments that measure the implications of attacks against both the physical and the cyber parts of CI. In this paper we present an innovative framework for an experimentation environment that incorporates both physical and cyber systems. We propose the use of an emulation testbed (based on Emulab) to recreate ICT components along with a soft real-time simulator of the physical processes (based on Simulink). The feasibility of our approach has been confirmed by the development of a fully working prototype. Its performance has been evaluated through a series of experiments. Our work aims to advance the experimentation capabilities in the field of Networked Industrial Control Systems (NICS) and allow us to measure and understand the consequences of cyber attacks to physical processes.

Keywords: Critical Infrastructures, SCADA systems, Security, Simulink, Emulab

1. Introduction

Modern Critical Infrastructures (CI), e.g. power plants, water plants and transport systems, rely on Information and Communication Technologies (ICT) for their operation since ICT can lead to cost reduction as well as greater efficiency, flexibility and interoperability between components. In the past CI were isolated environments and used proprietary hardware and protocols, limiting thus the threats that could affect them. Nowadays CI and more specifically Networked Industrial Control Systems (NICS) are exposed to significant cyber-threats; a fact that has

been highlighted by many studies on the security of Supervisory Control And Data Acquisition (SCADA) systems [1, 2]. For example, the recently discovered Stuxnet worm [3] is the first malware that is specifically designed to attack NICS. Its ability to reprogram the logic of control hardware in order to alter physical processes demonstrated how powerful such threats can be; it has served as a wakeup call for the international security community. Stuxnet raised many open questions, but most importantly it reminded us that we still lack an efficient way to conduct experiments that measure the impact of such threats against both the physical and the cyber parts of a CI.

The study of complex systems, either physical or cyber, could be carried out by experimenting with real systems, software simulators or emulators. Experimentation with real production systems suffers from the inability to control the experiment environment in order to reproduce results. Furthermore if the study intends to test the resilience or security of a system, there are obvious concerns about the potential side effects (faults and disruptions) to mission critical services. On the other hand the development of a dedicated experimentation infrastructure with real components is often economically prohibitive and disruptive experiments on top of it could be a risk to safety. Software based simulation has always been considered an efficient approach to study physical systems, mainly because it can offer low-cost, fast and accurate analysis. Nevertheless it has limited applicability in the context of cyber security due to the diversity and complexity of computer networks. Software simulators can effectively model normal operations, but fail to capture the way computer systems fail.

Based on these facts, we have chosen to follow a hybrid approach in between the two extremes of pure simulation and experimentation with only real components. We propose a framework that uses simulation for the physical components and an emulation testbed based on Emulab [4, 5] in order to recreate the cyber part of NICS, e.g., SCADA servers, corporate network etc. The models of the physical systems are developed in Matlab Simulink from which we generate the corresponding 'C' code using Matlab Real Time Workshop. The generated code is then executed in real time and is able to interact with the real components of our emulation testbed.

Our contribution lies in the fact that by following the proposed framework we managed to recreate an experimentation environment for measuring and understanding the consequences of cyber attacks to physical processes while using real cyber components and real malware in a completely safe way, i.e. without the risk of bringing a physical system into an unstable state. Furthermore we show through a series of performance

evaluation experiments that our framework can scale up and accurately recreate large NICS, e.g., having up to 100 Programmable Logic Controllers (PLCs).

Our paper is structured as follows. We begin in Section 1.2 with a short overview of related work and continue in Section 1.3 with a presentation of the proposed framework starting from a high level overview and going in depth down to prototype implementation details. In Section 1.4 we describe the experimental results for the performance evaluation of our prototype. Finally in Section 1.5 we summarize the main conclusions and indicate directions of future work.

2. Related Work

Analyzing the behavior of NICS is challenging as such systems include elements that interact both in the physical and the cyber domains. In this section we briefly present the most relevant papers dealing with this subject.

Chunlei, *et al.* [6] used a real OPC (Object Linking and Embedding (OLE) for Process Control) server, the NS-2 network simulator and real PLCs and field devices to analyze NICS. NS-2 is used to simulate the enterprise network of the SCADA system. Calls from NS-2 are dispatched through software agents to the OPC server that sends Modbus packages to the physical PLCs. In this approach the only simulated element is the enterprise network, while all the other components (servers, PLCs etc.) are real. Although from one point of view such a testbed would provide extremely reliable experimental data, since almost everything is real, it would be hardly able to support tests on large infrastructures such as the process system of a chemical installation, since that would require a complete real industrial system to experiment with. Nai, *et al.* [7] went indeed in this direction, in developing a protected environment for studying the cyber vulnerabilities of power plant control systems. The core of this environment is a real industrial system able to reproduce all the physical dynamics of a real power plant. However the high fidelity of this testing environment is counterbalanced by its poor flexibility (the radical change of scenario would imply the physical deployment of a new industrial system) and the high cost of maintenance of a similar architecture. Hiyama and Ueno [8] used Simulink to model physical systems and Matlab Real Time Workshop to run the model in real time with input and output signals attached to a DSP board. A similar approach to the one proposed by Hiyama and Ueno has been taken by Queiroz, *et al.* [9] to analyze the security of SCADA systems. In this case, only the sensors and actuators are real physical devices, the remaining compo-

nents, e.g., PLCs, and the communication protocols between them are implemented as OMNeT++ modules.

Other researchers have focused on simulating both SCADA and field devices. For example, Chabukswar, *et al.* [10] used the Command and Control WindTunnel (C2WindTunnel) [11] multi-model simulation environment, based on the High-Level Architecture (HLA) IEEE standard 1.3 [12], to enable the interaction between various simulation engines. The authors used OMNeT++ to simulate the network and Matlab Simulink to build and run the physical plant model. C2WindTunnel provides the global clock for both OMNeT++ and Matlab Simulink. With this approach, analyzing the cyber-physical effects of malware is not a trivial task, as it requires a detailed description of all ICT components and more importantly a detailed knowledge on the dynamics of malware, that is not always available.

Davis, *et al.* [13] used PowerWorld [14], "a high-voltage power system simulation and analysis package" [14], to model an entire power grid and run it in real time. The PowerWorld server is connected to a proxy that implements the Modbus protocol and transmits Modbus packages to client applications. Client applications interact with the PowerWorld server using a visual interface that allows them to introduce disturbances into the network and observe the effects. Our approach also uses simulation for the physical layer, however, we also emulate typical components such as PLCs and Master units that are missing from [13].

3. Framework Architecture

In this section we present a framework for an experimentation environment that can support the study and analysis of the physical impact of cyber-threats against Networked Industrial Control Systems (NICS). After providing a brief description of a typical NICS architecture, we present the proposed architecture and our implementation prototype.

3.1 Process Control Architecture Overview

In modern "Industrial Process Control Network Architectures", one can identify two different control layers: (i) the **physical layer** composed of all the actuators, sensors, and generally speaking hardware devices that physically perform the actions on the system (e.g. open a valve, measure the voltage in a cable etc.); (ii) the **cyber layer** composed of all the ICT devices and software which acquire the data, elaborate low level process strategies and deliver the commands to the physical layer. The cyber layer typically uses SCADA protocols to control and manage the industrial installation. The whole architecture can be

thought as a "distributed control system" spread among two networks: the *control network* and the *process network*. The process network usually hosts all the SCADA servers (also known as SCADA Masters) and HMI (Human Machine Interface). The control network hosts all the devices which, on the one side control the actuators and sensors of the physical layer and on the other side provide the "control interface" to the process network. A typical control network is composed of a mesh of *PLC*s (Programmable Logic Controller) as shown in Figure 1. From the

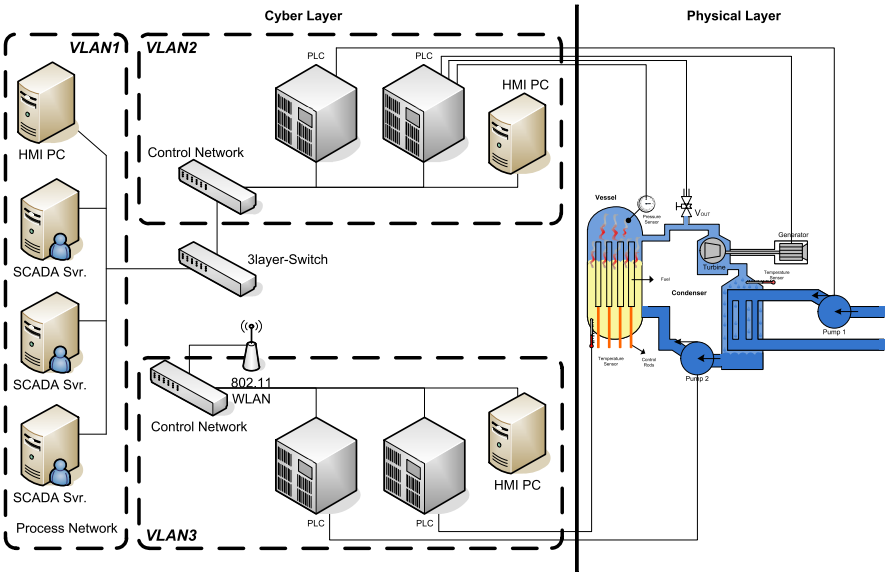


Figure 1: Example of an Industrial Plant Network

operational point of view, the PLCs receive data from the physical layer, elaborate on the basis of that data a "local actuation strategy", and send back commands to the actuators. The same PLCs provide, when requested, the data received from the physical layer to the SCADA servers (Masters) in the process network, and eventually execute the commands received by them. In modern SCADA architectures, the communication between Master and PLCs, is usually implemented in two different ways: (i) through an OPC (Object Linking and Embedding (OLE) for Process Control) layer which help in mapping the PLC devices, (ii) through a direct memory mapping notation making use of SCADA communication protocols like Modbus, DNP3, Profibus etc.

3.2 Approach Overview

The proposed experimentation framework follows a hybrid approach, where the Emulab-based testbed recreates the control and process network of NICS, including PLCs and SCADA servers, and a software simulation reproduces the physical processes. A high level view of the proposed architecture is shown in Figure 2. The argument for using emulation for the cyber components is that the study of the security and resilience of computer networks would require the simulation of all the failure related functions, behaviors and states, most of which are unknown in principle. On the other hand software simulation is a very reasonable approach for the physical layer due to small costs, the existence of accurate models and the ability to conduct experiments in a safe environment. The architecture presented in Figure 2 clearly distin-

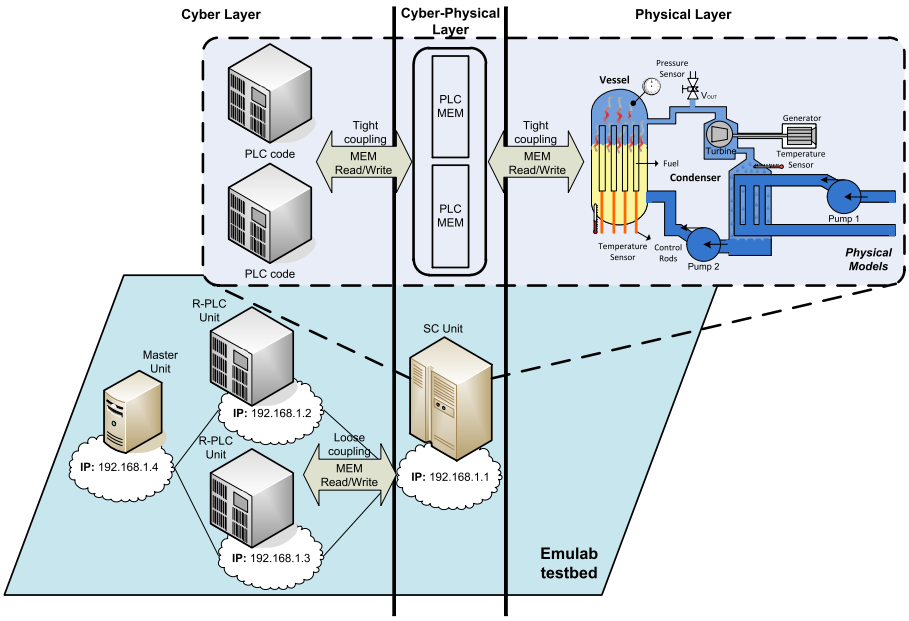


Figure 2: Architecture overview

guishes 3 layers: the cyber layer, the link layer and the physical layer. The cyber layer includes regular ICT technologies used in SCADA systems, while the physical layer provides the simulation of physical devices. The link layer provides the glue between the two layers through the use of a shared memory region.

The cyber layer is recreated by an emulation testbed that uses the Emulab architecture and software [4] to automatically and dynamically map physical components (e.g. servers, switches) to a virtual topology. In other words the Emulab software configures the physical topology in a way that it emulates the virtual topology as accurately as possible. More details can be found in [5].

Besides the process network, the cyber layer also includes the control logic code, that in the real world is implemented by PLCs. In our approach the control code can be run sequentially or in parallel to the physical model. In the sequential case we use a *tightly coupled* code, i.e. code that is running in the same memory space with the model. In the parallel case we use a *loosely coupled* code, i.e. code that is running in another address space, possibly on another host. The main advantage of *tightly coupled* code is that these do not miss values generated by the model between executions. On the other hand, the *loosely coupled* approach enables us to run PLC code remotely, to inject (malicious) code without stopping execution of the model, and to run more complex PLC emulators.

The cyber-physical layer incorporates the PLC memory, seen as a set of registers typical to PLCs, and the communication interfaces that glue together the other two layers. Memory registers provide the link to the inputs (e.g. valve position) and outputs (e.g. sensor values) of the physical model.

The physical layer provides a real-time execution of the physical model. In other words, the execution time is strongly coupled to the timing service of the operating system (OS) on which the model is running. As we use multi-tasking OSs, achieving hard real-time is a difficult task without using kernel drivers. However, we achieve soft real-time by allowing a certain deviation from the OS clock. We further elaborate this subject in the following sections. Choosing a *time step*, i.e. the time between two executions of the model, for a given setting is also not a trivial task. By choosing a small value (in the order of microseconds) we increase the deviation of the execution from the system clock while increasing the number of missed values by the loosely coupled code. On the other hand, by choosing a larger value (in the order of seconds) we might miss certain effects or attacks that could happen under the given time. At the same time, the chosen time step can not be less than the execution time of the physical model. We use the term *resolution* to denote the minimal value of the time step. In case the system also includes tightly coupled code, as such code is run sequentially with the model, the value of the resolution can not be less than the cumulative execution time of PLC code and physical model.

3.3 Detailed Architectural Description

The modular architecture of the framework is given in Figure 3. It includes 3 main units: SC (Simulation Core), R-PLC (Remote PLC) and SCADA Master. Through the remaining of this section we provide a more detailed presentation of each unit.

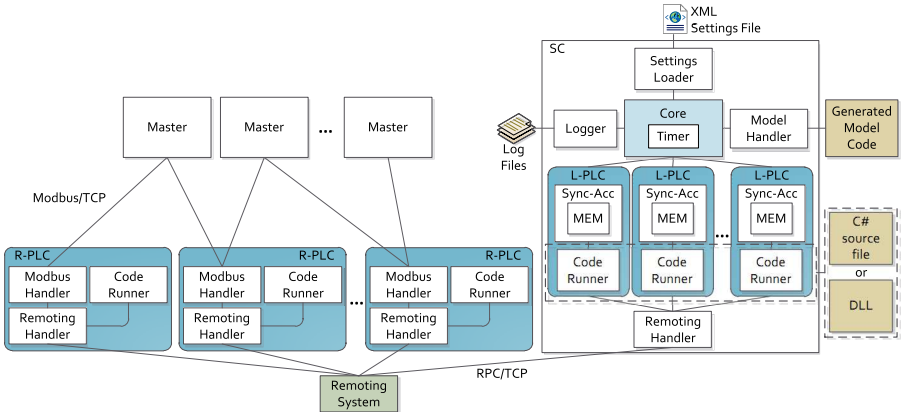


Figure 3: Modular architecture

SC Unit. The main role of the SC unit is to provide a soft real-time execution of tightly coupled code and physical model, synchronized with the clock of the OS, providing at the same time the glue between the cyber and the physical layer. The most important modules of the SC unit are: *Local-PLC* (L-PLC), *Remoting Handler* and *Core*. The L-PLC module incorporates the PLC memory (e.g. coils, digital input registers, input registers, holding registers) used as the glue between the cyber and physical layers and the code runner module. The *Remoting Handler* module handles the communication between the L-PLC modules and the local RPC system. The *Core* module ensures the exchange of data between modules and the execution of the core timer. With this timer the SC unit provides a soft real-time execution of the physical model.

R-PLC Unit. The main role of the R-PLC unit is to run loosely coupled code and to provide an interface for Master units for accessing the model. Its main modules include: the *Remoting Handler* module that implements the communication with the SC unit; the *Code Runner* module that runs the loosely coupled code and the *Modbus Handler* module that implements the Modbus protocol.

Master Unit. The main role of the Master unit is to implement a global decision based on the sensor values received from the R-PLC units. It includes a *Modbus Handler* module used in the communication with the R-PLC units and a *Decision Algorithm* module.

3.4 Implementation Details

The proposed framework has been developed in C#. Using the *Mono* platform we have also been able to port our framework to Unix-based systems. The tightly coupled code can be provided as C# source files or as binary DLLs, both dynamically loaded at run-time. C# source files are dynamically loaded, compiled and executed at run-time using .NET's support for dynamic code execution. Although with a greater execution time, as shown in the next sections, C# source files provide the ability to implement PLC code without the need of a development environment. At this time, the loosely coupled code is written in C# and must be compiled together with the rest of the unit.

For simulating the physical layer we decided to use Matlab Simulink, as we would need to cover a wide variety of plants (e.g. power plants, water purifying plants, gas plants). Matlab Simulink is a general design and simulation environment for dynamic and embedded systems. Its generality requires an accurate knowledge on the system's mathematical equations and the ability to construct the Simulink model based on these equations. Simulink also provides several toolboxes that contain pre-defined components for domains such as Power Systems, Mechanics, Hydraulics, Electronics, and so on. These toolboxes are enriched with every new release, providing a powerful support for designers and effectively reducing design time. For each Simulink model we generate the corresponding 'C' code using Matlab Real Time Workshop. The generated code is then integrated into our framework and its execution time is synchronized with the OS clock.

The communication between SC and R-PLC units is handled by .NET's binary implementation of RPC (called *remoting*) over TCP. By using .NET's remoting we ensure a minimal overhead and the use of a well-established implementation. Currently, for the communication between the R-PLC and the Master units, we implemented the Modbus over TCP protocol. However it is possible to easily add new protocols by substituting the Modbus Handler modules with other ones.

The synchronization of the model execution time with the system clock is implemented within the SC unit using a *synchronization algorithm* (SA). At a first glance, the structure of such an algorithm seems to be trivial, however, the experimental results indicate the existence of

several pitfalls. The main concern proves to be the PLC memory that is a shared resource between the SC unit and R-PLC units. This means that the PLC memory needs to be protected against simultaneous access, which introduces the problem of critical sections known from the field of concurrent programming. Intuitively, a SA would need to run the process model only once for each time step. However, in a multi-tasking environment such an approach introduces accumulated deviations as the OS can stop and resume threads without any intervention possibilities from the user space. Based on this observation, the implemented SA includes a loop to run the process model multiple times in each time step in order to reduce deviations.

4. Performance Evaluation

In this section we evaluate the performance of the framework in terms of scalability, resolution and deviation from the OS clock. We show that the framework is able to support even 100 PLCs with code sizes ranging from a few *if* instructions to 1000.

4.1 Experimental Setup

The following measurements were done in our Emulab testbed running FreeBSD OS. However, the framework has also been tested on Windows 7, Fedora Core 8 and Ubuntu 10.10 OS. In total, we used 8 hosts, 1 for running the SC unit, 1 for running the Master unit and 6 other hosts to run at most 100 R-PLC units (depending on the setup of the experiment). The general architecture of the experimental setup is given in Figure 4. Plant models are constructed in Simulink, from which the

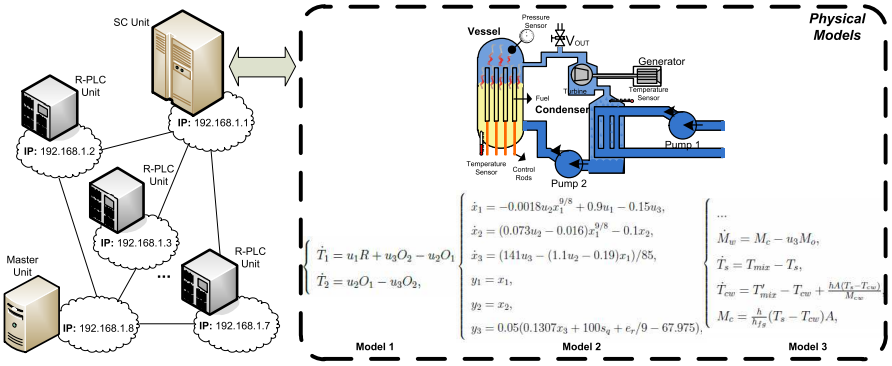


Figure 4: Architecture of the experimental setup and plant models

corresponding 'C' code is generated using Matlab Real Time Workshop. We used 3 plant models. The first one is our own construction, representing a simplified model of a water purifying plant with 2 water tanks. The equations of the model are shown in Figure 4, under the name *Model 1*. For the second model, named *Model 2*, we used the one presented by Bell and Åström [15]. There are several other power plant models available in the open literature [16–19]. We have chosen the Bell and Åström model because it includes estimated parameters from a real power plant and it is a well established model, as other researchers have also been using it to validate their proposals [20, 21]. The model is that of a 160MW oil-fired electric power plant based on the Sydsvenska Kraft AB plant in Malmö, Sweden. It includes the boiler and the turbine part of a power plant. For the third one, named *Model 3*, we extended the model of Bell and Åström with a condenser component. The equations for the condenser are based on the work of Kim, *et al.* [22].

4.2 Plant Model Execution Time

We measured the execution time of the constructed Simulink models for the 3 plants mentioned in the previous section, with the following results. The measured execution time for the first model is 19.2μs. The second model has 4 additional equations (also including equations for s_q and e_r), with the added execution time of 4μs and a total execution time of 23.2μs. The third model has also 4 additional equations, with an added execution time of 4.7μs and a total execution time of 27.9μs. Based on these measurements, the time step chosen for a given system can not be less than the execution time of the model. For instance, in the case of the third model, the time step can not be less than 27.9μs.

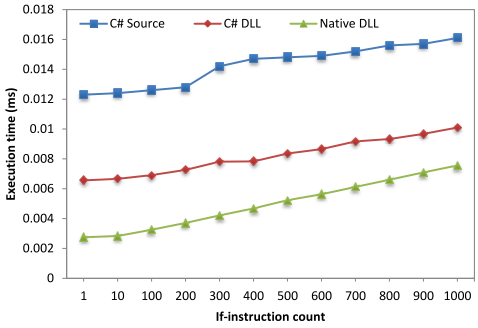


Figure 5: Execution time of tightly coupled PLC code

4.3 PLC Code Execution Time

In the current implementation the tightly coupled PLC code can be provided as a source C# file, as a C# DLL or as a native dynamic library. The loosely coupled PLC code is included as a module in the R-PLC unit, thus it must be written in C# and compiled together with the rest of the unit.

As shown in Figure 5, the execution time of PLC code varies with the type of implementation (i.e. source C# file, as a C# DLL or as a native dynamic library) and number of *if-instructions* (for each *if* we also considered a PLC memory write instruction). The C# source file has the largest execution time as the code is compiled at run-time, while the native dynamic library has the smallest execution time as this is a binary compiled for the target platform. However, the key advantage of using source C# files is that they do not require the presence of development libraries, changes can be made quickly and experiments can be resumed within short periods of time.

4.4 Measuring the Resolution

For tightly coupled code the resolution is a function of the execution time of the model itself, the execution time of tightly coupled code and the added overhead of handling PLC code, i.e. $RES_1 = t_{model} + \sum_i (t^i_{plc} + t_{oh})$. For this setting the missed rate equals zero, as PLC code is running sequentially with the model, i.e. $Miss_1 = 0$. For loosely coupled code the resolution is only a function of the model execution time, i.e. $RES_2 = t_{model}$. However, the number of missed values is not 0 anymore, as it depends on the chosen time step and the number of PLCs, i.e. $Miss_2 = \alpha^N_{st}$, where st is the chosen time step and N is the number of loosely coupled PLCs. The value of α^N_{st} is determined empirically and it can only be approximated, as we are running on multi-tasking OS. For a mixed system that includes both loosely and tightly coupled code, the resolution equals that of the tightly coupled setting, i.e. $RES_3 = RES_1$, and the missed count equals that of the loosely coupled setting, i.e. $Miss_3 = Miss_2$. The minimal value of the time step that can be chosen is thus the resolution of the system.

For tightly coupled code we measured the resolution for up to 100 PLCs and code size ranging from 1 to 1000 *if-instruction* sets. The results shown in Figure 6 correspond to Model 2, with an execution time of 23.3μs and native DLL-based PLC code. Of course, by using other models with a larger execution time or other type of PLC code implementations, the resolution automatically increases. By increasing the number of PLCs we also increase the value of the resolution, as a

larger number of PLCs means a larger number of PLC code that must be executed sequentially with the model. For instance, in case of 100 *if-instruction* PLC code and a single PLC the resolution is 0.029ms and increases up to 0.204ms for 100 PLCs. However, values generated by the model are not missed and PLCs can react to all changes of the model.

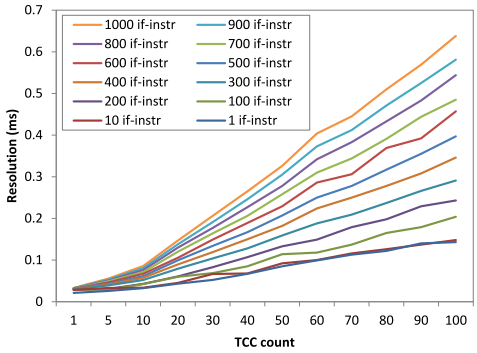


Figure 6: Resolution

For loosely coupled code the resolution equals the model execution time. Naturally, for mixed systems the resolution equals the model execution time plus the tightly coupled code execution time (i.e. Figure 6). As opposed to the tightly coupled setting, we must consider that PLCs will miss values generated by the model as the execution time is not synchronized between units, that we are using multi-tasking OS and that there is a network communication that introduces additional delays. Thus, providing a low miss rate at resolutions of 0.1ms is a difficult task on multi-tasking OS and with the added overhead of network communications. We have chosen 9 time steps ranging from 0.1ms to 1000ms and we have measured the average number of missed reads for each time step. Each PLC reads the remote memory, runs a 100 *if-instruction* set and finally writes the results back to the remote memory. The number of missed reads is influenced by the number of PLCs and the read frequency. We considered up to 100 PLCs and 2 read frequencies: once and three times in each time step.

In Figure 7 (a) and (b) we show the average (from 1 to 100 PLCs) measured percentages of missed reads for, respectively, 1 read/time step and 3 reads/time step. For both settings the average miss rate exceeds 50% for time steps smaller than 1ms, mainly due to network delays, for which we measured a minimum value of 0.25ms, while for time steps larger than 10ms the percentage of missed reads becomes 0 (even for 100 PLCs). By comparing the average values for the two settings, shown in

Figure 7 (c), we notice a slight decrease in the value of the miss rate when using a read frequency of 1/time step. The reason behind this behavior is that by reducing the number of reads we actually decrease the number of simultaneous accesses to the synchronized PLC memory, thus more PLCs are able to access the remote memory in each time step.

To conclude, for the extreme case of 100 tightly coupled code, with each PLC running 1000 *if-instructions*, our framework is able to provide a resolution of 0.638ms with 0 miss rate. On the other hand, achieving a miss rate of zero for the loosely coupled setting is only possible with time steps larger than 100ms.

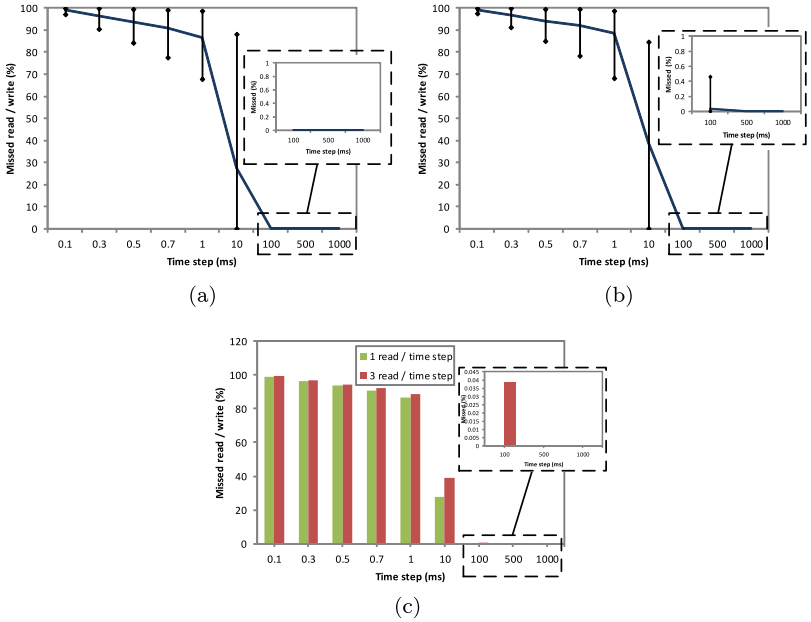


Figure 7: Average missed PLC read percentage: (a) 1 read/time step (b) 3 reads/time step (c) 1 read vs 3 reads/time step

4.5 Measuring the Deviation

The proposed framework maintains the execution time synchronized with the OS clock. Because of the multi-tasking environment used to run the framework, the synchronization is affected by the number of PLCs and the chosen code coupling. The measured deviation for the tightly coupled code setting is shown in Figure 8. For a time step of

0.1ms the evolution of the deviation is shown with a dashed line in order to illustrate the fact that by using more than 20 PLCs it increases gradually with the execution and for 30 PLCs it exceeds 1ms. This is because the cumulated PLC code execution time exceeds the 0.1ms time step. On the other hand, deviation is decreased by increasing the time step. For instance, in the case of time steps 10ms, 100ms, 500ms and 1s the deviation is maintained around 2.2μs even for 100 PLCs.

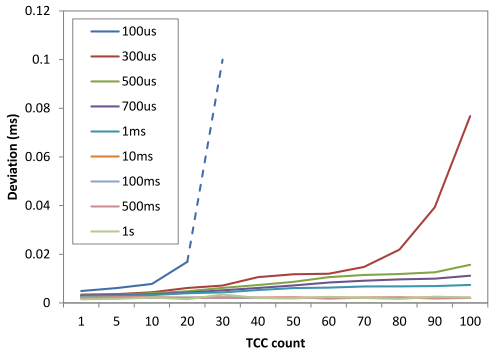
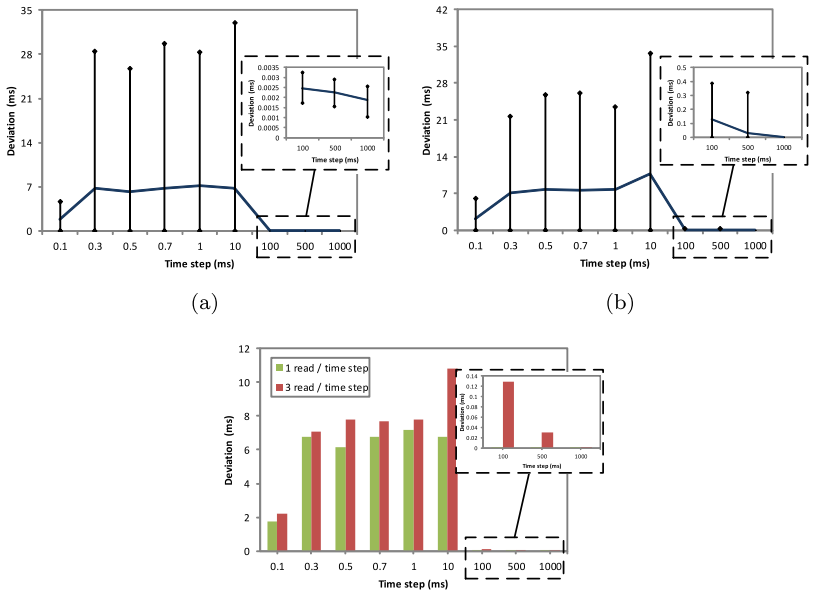


Figure 8: Deviation - tightly coupled PLC code

For the loosely coupled setting the average measured deviation is shown in Figure 9 (a) and (b). For both settings the deviation increases its value starting with a 0.1ms time step, as more PLCs are able to access the remote PLC memory. Starting with a 100ms time step the average deviation is maintained around 0.002ms for 1 read/time step and 0.1ms for 3 reads/time step. By comparing the average values for the two settings, as shown in Figure 9 (c), as expected, a larger read frequency introduces larger deviations. More specifically, the value of the deviation increases up to 50 times for a time step of 100ms.

5. Concluding Remarks

In this paper we have presented a new framework for the security analysis of NICS. Existing approaches either use real physical elements combined with simulated/emulated ones or use a completely simulated system. One of our main contributions is the hybrid architecture of the proposed framework that uses emulation for protocols and components such as PLCs and SCADA Masters and simulation for the physical layer. With this approach we are able to capture the complexity of ICT and to efficiently handle the complexity of the physical layer. Another novelty of our proposal is that it supports both tightly and loosely coupled code.



(a) (b)

(c)

Figure 9: Average deviation - loosely coupled PLC code: (a) 1 read/time step (b) 3 reads/time step (c) 1 read vs 3 reads/time step

Thus, for settings where PLCs can not afford missing any events we can simply use a tightly coupled code, while for injecting new (malicious) code without stopping execution we can simply use a loosely coupled code. By using tightly coupled code we are also able to ensure lower resolutions with zero miss rate. At the same time with the loosely coupled setting we are able to integrate more complex PLC emulators without affecting the architecture of the other units.

The complexity of the models we used ranges from simple water purifying plants to more complex Boiling Water Power Plants. The measurements we made proved that our framework is capable of running complex models within tens of microseconds. We also measured several performance parameters such as resolution, miss rate and deviation. For tightly coupled code we get a higher value for the resolution, but with a lower miss rate and deviation. On the other hand, for loosely coupled code we get a lower value for the resolution, but with a higher miss rate and deviation.

As future work we intend to use the proposed framework for studying the propagation of perturbations in cyber-physical environments, to ana-

lyze the behavior of physical plants and to develop countermeasures. We also intend to analyze the physical impact of attacks using more complex models that also include a description of the physical components.

References

[1] I. Nai Fovino, A. Carcano, M. Masera and A. Trombetta, An experimental investigation of malware attacks on SCADA systems, *International Journal of Critical Infrastructure Protection*, 2(4), pp. 139–145, 2009.

[2] S. East, J. Butts, M. Papa and S. Shenoi, A Taxonomy of Attacks on the DNP3 Protocol, *IFIP Advances in Information and Communication Technology*, Springer Boston, 311/2009, pp. 67–81, 2009.

[3] The Symantec Stuxnet Dossier (http://www.wired.com/images_blogs/threatlevel/2010/11/w32_stuxnet_dossier.pdf), 2010.

[4] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb and A. Joglekar, An integrated experimental environment for distributed systems and networks, *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pp. 255–270, 2002.

[5] M. Guglielmi, I. Nai Fovino, A. P. Garcia and C. Siaterlis, A preliminary study of a wireless process control network using emulation testbed, *Proceedings of the 2nd International Conference on Mobile Lightweight Wireless Systems*, pp. 268–279, 2010.

[6] W. Chunlei, F. Lan and D. Yiqi, A Simulation Environment for SCADA Security Analysis and Assessment, *International Conference on Measuring Technology and Mechatronics Automation*, pp. 342–347, 2010.

[7] I. Nai Fovino, M. Masera, L. Guidi and G. Carpi, An Experimental Platform for Assessing SCADA Vulnerabilities and Countermeasures in Power Plants, *Proceedings of the IEEE 3rd International Conference on Human System Interaction*, pp. 679–686, 2010.

[8] T. Hiyama and A. Ueno, Development of real time power system simulator in Matlab/Simulink environment, *IEEE Power Engineering Society Summer Meeting*, vol. 4, pp. 2096–2100, 2000.

[9] C. Queiroz, A. Mahmood, J. Hu, Z. Tari and X. Yu, Building a SCADA Security Testbed, *Proceedings of the Third International Conference on Network and System Security*, pp. 357–364, 2009.

[10] R. Chabukswar, B. Sinopoli, G. Karsai, A. Giani, H. Neema and A. Davis, Simulation of Network Attacks on SCADA Systems, *First Workshop on Secure Control Systems*, 2010.

[11] S. Neema, T. Bapty, X. Koutsoukos, H. Neema, J. Sztipanovits and G. Karsai, Model Based Integration and Experimentation of Information Fusion and C2 Systems, Simulation (In Press), 2010.

[12] J. O. Calvin and R. Weatherly, An introduction to the high level architecture (HLA) runtime infrastructure (RTI), *Proceedings of the 14th Workshop on Standards for the Interoperability of Defence Simulations*, pp. 705–715, 1996.

[13] C. M. Davis, J. E. Tate, H. Okhravi, C. Grier, T. J. Overbye and D. Nicol, SCADA Cyber Security Testbed Development, *38th North American Power Symposium*, pp. 483–488, 2006.

[14] PowerWorld (http://www.powerworld.com).

[15] R. D. Bell and K. J. Åström, Dynamic models for boiler-turbine alternator units: data logs and parameter estimation for a 160MW unit, Lundt Institute of Technology, Sweden, Report TFRT–3192, 1987.

[16] A. Kumar, K. S. Sandhu, S. P. Jain and P. Sharath Kumar, Modeling and Control of Micro-Turbine Based Distributed Generation System, *International Journal of Circuits, Systems and Signal Processing*, Issue 2, vol. 3, pp. 65–72, 2009.

[17] J. P. McDonald and H. G. Kwatny, Design and Analysis of Boiler-Turbine-Generator Controls Using Optimal Linear Regulator Theory, *IEEE Transactions on Automatic Control*, vol. AC-18, no. 3, pp. 202–209, 1973.

[18] P. K. Chawdhry and B. W. Hogg, Identification of boiler models, *IEE Proceedings*, vol. 136, no. 5, pp. 261–271, 1989.

[19] H. Seifi and A. R. Seifi, An intelligent tutoring system for a power plant simulator, *Electric Power Systems Research*, 62, pp. 161–171, 2002.

[20] W. Tan, H. J. Marquez, T. Chen and J. Liu, Analysis and control of a nonlinear boiler-turbine unit, *Journal of Process Control*, 15, pp. 883–891, 2005.

[21] A. B. Abdennour and K. Y. Lee, An autonomous control system for boiler-turbine units, *IEEE Transactions on Energy Conversion*, vol. 11, no. 2, pp. 401–406, 1996.

[22] Y. S. Kim, M. K. Chung, J. W. Park and M. H. Chun, An Experimental Investigation of Direct Condensation of Steam Jet in Subcooled Water, *Journal of the Corean Nuclear Society*, vol. 29, no. 1, pp. 45–57, 1997.