Chapter 12

ANALYZING CYBER-PHYSICAL ATTACKS ON NETWORKED INDUSTRIAL CONTROL SYSTEMS

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

Abstract Considerable research has focused on securing SCADA systems and protocols, but an efficient approach for conducting experiments that measure the impact of attacks on the cyber and physical components of the critical infrastructure is not yet available. This paper attempts to address the issue by presenting an innovative experimental framework that incorporates cyber and physical systems. An emulation testbed based on Emulab is used to model cyber components while a soft real-time simulator based on Simulink is used to model physical processes. The feasibility and performance of the prototype is evaluated through a series of experiments. The prototype supports experimentation with networked industrial control systems and helps understand and measure the consequences of cyber attacks on physical processes.

Keywords: Industrial control systems, cyber attacks, simulation, testbed

1. Introduction

Modern critical infrastructures such as power plants and water supply systems rely on information and communications technologies, which contribute to reduced costs as well as greater efficiency, flexibility and interoperability. However, these technologies, which underlie networked industrial control systems, are exposed to significant cyber threats [7, 14]. The recent Stuxnet worm [8] is the first malware that was specifically designed to attack networked industrial control systems. Stuxnet's ability to reprogram the logic of control hardware and alter physical processes demonstrates the danger of modern cyber threats. Stuxnet has served as a wakeup call for the international security community, and has raised many questions. Above all, Stuxnet reminds us that an efficient approach for conducting experiments that measure the impact of attacks

J. Butts and S. Shenoi (Eds.): Critical Infrastructure Protection V, IFIP AICT 367, pp. 167–183, 2011.

on the cyber and physical components of the critical infrastructure is not yet available.

The study of complex systems, whether cyber or physical, can be carried out by experimenting with real systems, software simulators or emulators. Experimentation with real production systems is hindered by the inability to control the environments adequately to obtain reproducible results. Furthermore, any study that attempts to analyze security or resilience raises serious concerns about the potential faults and disruptions to mission-critical systems. The alternative, a dedicated experimental infrastructure with real components, is costly and the experiments can pose safety risks. Software-based simulation is generally considered to be an efficient approach to study physical systems, mainly because of its lower cost coupled with fast and accurate analysis. However, it has limited applicability to cyber security because of the complexity and diversity of information and communications technologies. Moreover, while software simulators may effectively model normal operations, they fail to capture the manner in which computer systems fail.

For these reasons, we have chosen to adopt a hybrid approach between the two extremes of experimentation with real components and pure software simulation. Our proposed framework uses simulation for the physical components and an emulation testbed based on Emulab [9, 22] to recreate the cyber components of networked industrial control systems such as SCADA servers and corporate networks. The models of the physical systems are developed using Matlab Simulink, from which the corresponding C code is generated using Matlab Real Time Workshop. The generated code is executed in real time and can interact with the real components in the emulation testbed.

The primary advantage of the framework is that it provides an experimentation environment for understanding and measuring the consequences of cyber attacks to physical processes while using real cyber components and malware in a safe manner. Furthermore, experimental evaluations of the framework demonstrate that it can scale and accurately recreate large networked industrial control systems with up to 100 programmable logic controllers (PLCs).

2. Related Work

Analyzing the behavior of networked industrial control systems is challenging because they incorporate components that interact in the physical and cyber domains. This section briefly describes the most relevant work on the subject.

Wang, *et al.* [21] employ an OPC (OLE for Process Control) server, the ns-2 network simulator, and real PLCs and field devices to analyze networked industrial control systems. ns-2 is used to simulate the enterprise network of a SCADA system. Calls from ns-2 are dispatched via software agents to the OPC server, which sends Modbus messages to the physical PLCs. In this approach, the only simulated component is the enterprise network; all the other components (servers, PLCs, etc.) are real. Because almost every component is real, such a testbed can provide reliable experimental data, but it cannot support tests on large infrastructures such as chemical plants and gas pipelines.

Nai Fovino, *et al.* [15] have also pursued a similar approach by developing a protected environment for studying cyber vulnerabilities in power plant control systems. The core of their environment is a real industrial system that reproduces the physical dynamics of a power plant. However, the high fidelity of this testing environment is counterbalanced by its poor flexibility with respect to handling new systems and its high maintenance costs.

Hiyama and Ueno [10] have used Simulink to model physical systems and the Matlab Real Time Workshop to run the model in real time. A similar approach has been used by Queiroz, *et al.* [18] to analyze the security of SCADA systems. In their case, only the sensors and actuators are real physical devices; the remaining components (e.g., PLCs) and the communication protocols are implemented as OMNeT++ modules.

Other researchers focus on simulating both SCADA and field devices. For example, Chabukswar, *et al.* [4] use the Command and Control WindTunnel (C2WindTunnel) [16] multi-model simulation environment, based on the High-Level Architecture (HLA) IEEE Standard 1.3 [3], to enable interactions between multiple simulation engines. They use OMNeT++ to simulate the network and the Matlab Simulink to build and run the physical plant model. C2WindTunnel provides the global clock for OMNeT++ and Matlab Simulink. Nevertheless, analyzing the cyber-physical effects of malware is a challenging task, because it requires a detailed description of all the cyber components and detailed knowledge of the dynamics of malware, which is rarely available.

Davis, *et al.* [6] use PowerWorld [17], a high-voltage power system simulation and analysis package [17], 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 messages to client applications. Client applications interact with the PowerWorld server via 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, unlike Davis, *et al.* [6], we also emulate typical components such as PLCs and master units.

3. Proposed Framework

This section presents the proposed experimentation framework that supports the analysis of the physical impact of cyber threats against networked industrial control systems. Following a brief description of a typical networked industrial control system architecture, we present the proposed framework and its prototype implementation.

3.1 Control System Architecture

Modern industrial process control network architectures have two control layers: (i) the physical layer, which comprises actuators, sensors and hardware devices that physically perform the actions on the system (e.g., open valves, measure voltages, etc.); and (ii) the cyber layer, which comprises all the information and communications devices and software that acquire data, elaborate

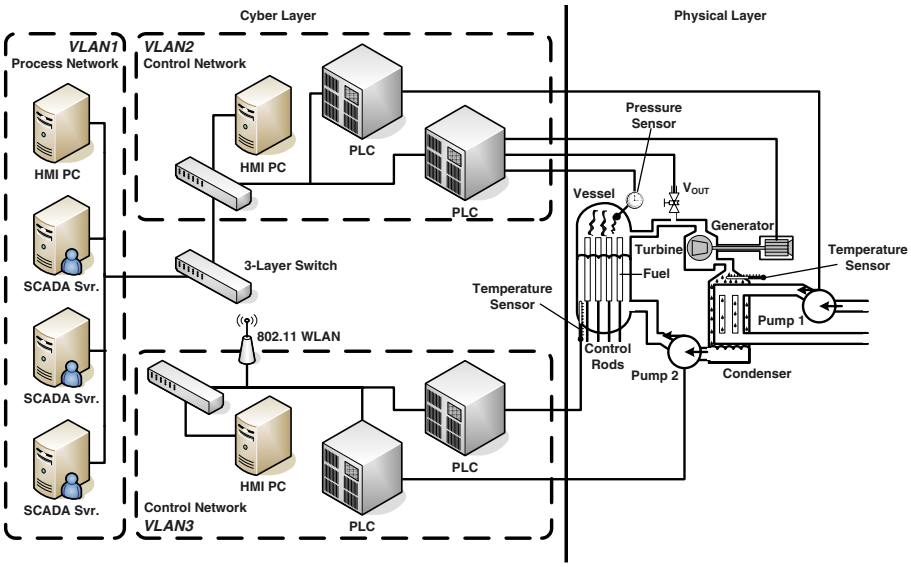


Figure 1. Industrial plant network.

low-level process strategies and deliver commands to the physical layer. The cyber layer typically uses SCADA protocols to control and manage an industrial installation. The entire architecture can be viewed as a "distributed control system" spread among two networks: the control network and the process network. The process network usually hosts the SCADA servers (also known as SCADA masters) and human-machine interfaces (HMIs). The control network hosts all the devices that on 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 PLCs as shown in Figure 1.

From the operational point of view, PLCs receive data from the physical layer, elaborate a local actuation strategy based on the data, and send commands to the actuators. When requested, the PLCs also provide the data received from the physical layer to the SCADA servers (masters) in the process network and eventually execute the commands that they receive. In modern SCADA architectures, communications between a master and PLCs are usually implemented in two ways: (i) through an OPC layer that helps map the PLC devices; and/or (ii) through a direct memory mapping that uses SCADA communication protocols such as Modbus, DNP3 and Profibus.

3.2 Overview of the Approach

The proposed framework engages a hybrid approach, where the Emulab-based testbed recreates the control and process network (including the SCADA

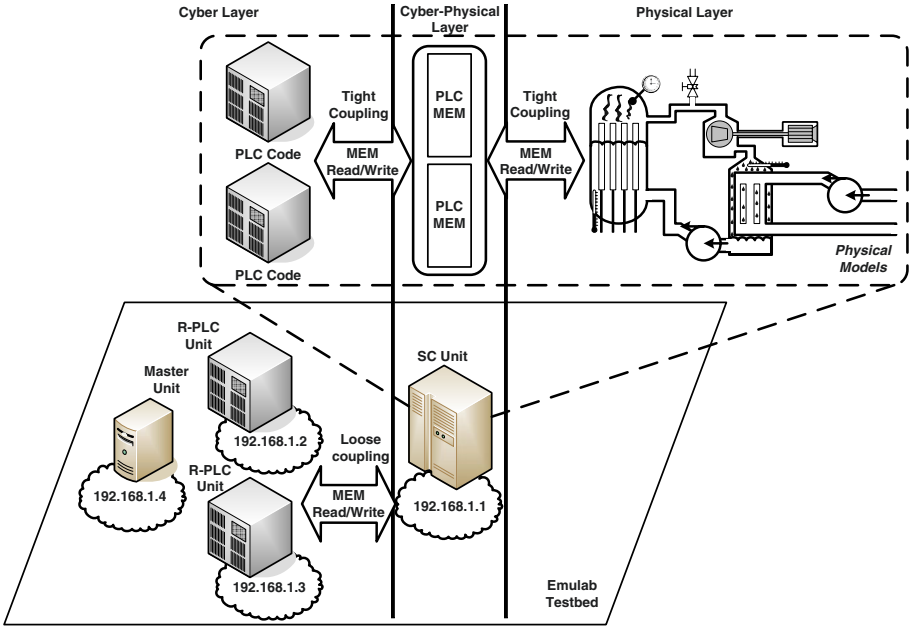


Figure 2. Overview of the framework.

servers and PLCs), and a software simulation reproduces the physical processes. Figure 2 shows a high-level view of the proposed framework. The principal argument for emulating the cyber components is that any study of the security and resilience of a computer network requires the simulation of all the failure-related functions, behaviors and states, most of which are unknown. On the other hand, software simulation is a very reasonable approach for the physical layer because of its low cost, the existence of accurate models and the ability to conduct experiments safely.

The architecture presented in Figure 2 has three layers: the cyber layer, link layer and physical layer. The cyber layer incorporates the information and communications devices used in SCADA systems, while the physical layer provides the simulation of physical devices. The link layer serves as the glue for 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 [22] to automatically and dynamically map physical components (e.g., servers and switches) to a virtual topology. In other words, the Emulab software configures the physical topology so that it emulates the virtual topology as accurately as possible. Interested readers are referred to [9] for additional details.

Aside from the process network, the cyber layer also includes the control logic code, which is implemented by PLCs in the real world. In our approach, the control code can be made to run sequentially or in parallel with the physical

model. In the sequential case, we use tightly-coupled code, i.e., code that runs in the same memory space as the model. In the parallel case, we use loosely-coupled code, i.e., code that runs in another address space, possibly on another host. The main advantage of tightly-coupled code is that it does not miss values generated by the model between executions. On the other hand, loosely-coupled code supports the execution of PLC code remotely, the injection of malicious code without stopping the execution of the model and the operation of more complex PLC emulators.

The cyber-physical layer incorporates the PLC memory (usually a set of registers) and the communications interfaces that glue the other two layers. Memory registers provide the links to the inputs (e.g., valve positions) and outputs (e.g. sensor values) of the physical model.

The physical layer provides the real-time execution of the physical model. The execution time of the model is strongly coupled to the timing service provided by the operating system on which the model executes. Because a multi-tasking operating system is used, achieving hard real-time execution is a difficult task without using kernel drivers. However, soft real-time execution can be achieved by allowing some deviations from the operating system clock. We further elaborate on this topic in the following sections.

The selection of a time step, i.e., the time between two executions of the model, for a given setting is not a trivial task. Choosing a small value (in the order of microseconds), increases the deviation of the execution from the system clock and the number of missed values by loosely-coupled code. On the other hand, choosing a larger value (in the order of seconds), may cause certain effects and attacks to be missed. At the same time, the selected time step cannot be less than the execution time of the physical model. We use the term "resolution" to denote the minimal value of the time step. When a system includes tightly-coupled code, because this code executes sequentially with the model, the resolution cannot be less than the cumulative execution time of the PLC code and physical model.

3.3 Detailed Architectural Description

Figure 3 presents the modular structure of the framework. The framework has three main units: the simulation core (SC), remote PLC (R-PLC) and SCADA master.

- **Simulation Core Unit:** The main role of the simulation core unit is to provide soft real-time execution of tightly-coupled code and the physical model, synchronized with the operating system clock, providing at the same time the glue between the cyber and physical layers. The most important modules of the simulation core unit are: the local PLC (L-PLC), remoting handler and core. Communications between simulation core and remote units are handled by .NET's binary implementation of RPC over TCP (called "remoting"). The local PLC module incorporates the PLC memory (e.g., coils, digital input registers, input registers and

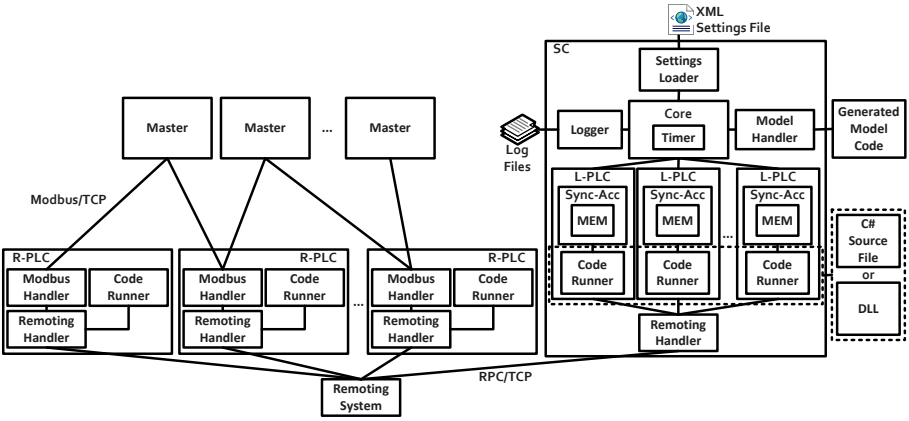


Figure 3. Modular architecture.

holding registers), which is used as the glue between the cyber and physical layers and the code runner module. The remoting handler module handles the communications between the local 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 the help of this timer, the simulation core unit provides soft real-time execution of the physical model.

- **Remote PLC Unit:** The main role of the remote PLC unit is to run loosely-coupled code and to provide an interface for master units to access the model. The main modules include: the remoting handler module, which implements communications with the simulation core; the code runner module, which runs the loosely-coupled code; and the Modbus handler module, which 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 remote PLC units. It includes a Modbus handler module for communicating with the remote PLC units and the decision algorithm module.

3.4 Implementation Details

The framework code was written in C#. The Mono platform was used to port the framework to a Unix system. Tightly-coupled code can be provided as C# source files or as binary DLLs, both of which are dynamically loaded at run time. C# source files are dynamically loaded, compiled and executed at run time using .NET support for dynamic code execution. Although C# source files have longer execution times, they provide the ability to implement PLC code without a development environment. At this time, loosely-coupled code is written in C# and must be compiled with the rest of the unit.

Matlab Simulink was used to simulate the physical layer because a wide variety of plants (e.g., power plants, water purification plants and gas plants) have to be covered. Matlab Simulink is a general design and simulation environment for dynamic and embedded systems. It provides several toolboxes that contain pre-defined components for domains such as power systems, mechanics, hydraulics, electronics, etc. These toolboxes are enriched with every new release, providing powerful support for designers and effectively reducing the design time. C code corresponding to Simulink models are generated using the Matlab Real Time Workshop. The generated code is then integrated into the framework and its execution time is synchronized with the operating system clock.

As mentioned above, communications between the simulation core and remote units are handled by .NET's remoting feature. .NET remoting ensures minimal overhead and the use of a well-established implementation. Currently, we use Modbus over TCP for communications between remote PLCs and master units. However, new protocols are easily added by substituting the Modbus handler module.

The synchronization of the model execution time with the system clock is implemented within the simulation core unit using a synchronization algorithm. At first glance, such an algorithm seems to be trivial; however, an experimental study has indicated the existence of several pitfalls. The main concern is that the PLC memory is a shared resource between the simulation core unit and remote PLC units. This means that the PLC memory has to be protected from simultaneous access, which introduces the problem of critical sections from the field of concurrent programming. Intuitively, a synchronization algorithm would have to run the process model only once for each time step. However, in a multi-tasking environment such an approach introduces accumulated deviations because the operating system can stop and resume threads without any intervention from the user space. Based on this observation, the implemented synchronization algorithm includes a loop to run the process model multiple times in each time step in order to reduce the deviation.

4. Performance Evaluation

This section focuses on the evaluation of the performance of the framework with respect to scalability, resolution and deviation from the operating system clock. The results show that the framework can support as many as 100 PLCs with code sizes ranging from a few if-instructions to 1,000 if-instructions.

4.1 Experimental Setup

The experiments were conducted on an Emulab testbed running the FreeBSD operating system. Note, however, that the framework was also tested with the Windows 7, Fedora Core 8 and Ubuntu 10.10 operating systems. In all, eight hosts were used, one for running the simulation core unit, one for running the

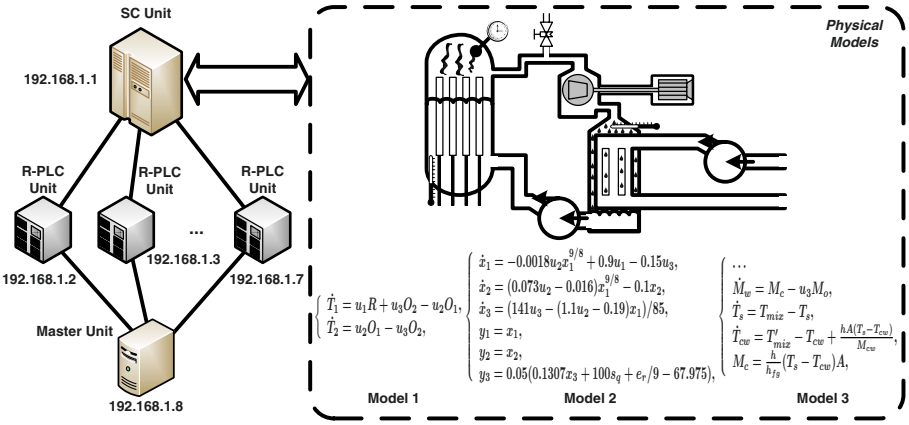


Figure 4. Experimental setup and plant models.

master unit, and six for running up to 100 remote PLC units. Figure 4 presents the experimental setup and the plant models that were used.

Three plant models were constructed in Simulink, from which the corresponding C code was generated using the Matlab Real Time Workshop. The first model (Model 1 in Figure 4) corresponds to a simplified version of a water purification plant with two water tanks. Model 2 corresponds to a 160 MW oil-fired electric power plant based on the Sydsvenska Kraft AB plant in Malmo, Sweden [2]; the model includes a boiler and turbine. Several power plant models are available in the literature [5, 12, 13, 19], however, this particular model was selected because it includes estimated parameters from a real power plant and has been used by other researchers [1, 20] to validate their proposals. Model 3 extends the second model by incorporating a condenser; the equations for the condenser are based on [11].

4.2 Plant Model Execution Time

We measured the execution times of the Simulink models for the three plants mentioned above. The execution time of the first model was 19.2 μs. The second model has four additional equations, including the equations for s_q and e_r, which yielded an added execution time of 4 μs and a total execution time of 23.2 μs. The third model also has four additional equations, which yielded an added execution time of 4.7 μs and a total execution time of 27.9 μs. Based on these measurements, the time step chosen for a system cannot be less than the execution time of the model.

4.3 PLC Code Execution Time

In the current implementation, tightly-coupled PLC code can be provided as a C# source file, a C# DLL or a native dynamic library. Since the loosely-

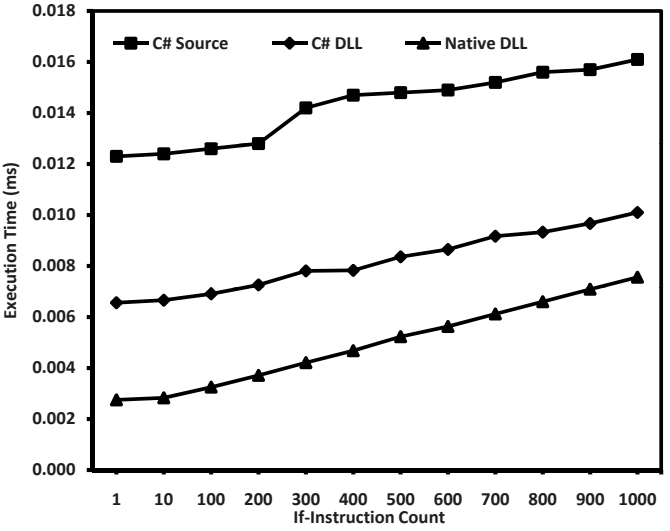


Figure 5. Execution time of tightly-coupled PLC code.

coupled PLC code was included as a module in the remote PLC unit, it had to be written in C# and compiled with the rest of the unit.

As shown in Figure 5, the execution time of PLC code varies with the type of implementation (e.g., C# source file, C# DLL or native dynamic library) and the number of if-instructions (for each if-instruction, we also considered a PLC memory write instruction). The C# source file had the longest execution time because the code was compiled at runtime, while the native dynamic library had the shortest execution time because it was a binary that was compiled for the target platform. However, the key advantage of using C# source files is that they do not require the presence of development libraries, changes can be made rapidly and experiments can be resumed quickly.

4.4 Measuring System Resolution

For tightly-coupled code, the resolution is a function of the execution time of the model, the execution time of the tightly-coupled code and the added overhead of handling PLC code:

$$RES_1 = t_{model} + \sum_i (t^i_{plc} + t_{oh}).$$

For this setting, the missed rate is zero because the PLC code was run sequentially with the model, i.e., $Miss_1 = 0$.

For loosely-coupled code, the resolution is only a function of the model execution time:

$$RES_2 = t_{model}.$$

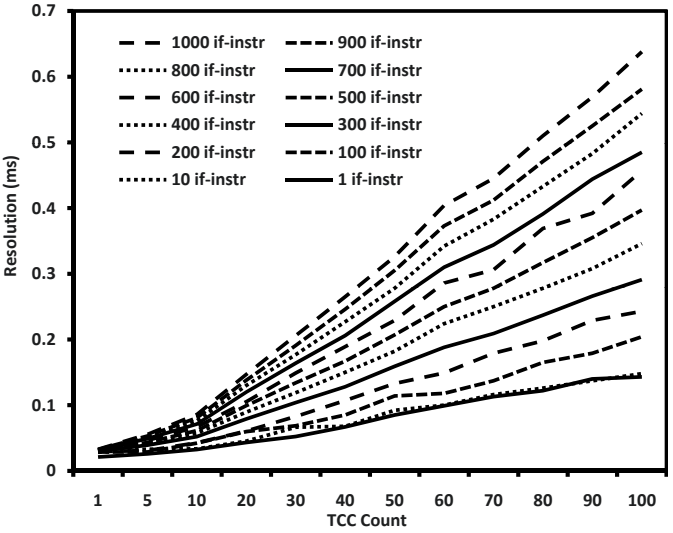


Figure 6. System resolution.

However, the number of missed values is no longer zero because it depends on the chosen time step and the number of PLCs, i.e., $Miss_2 = \alpha_{st}^N$, where st is the chosen time step and N is the number of loosely-coupled PLCs. The value of α_{st}^N is determined empirically and can only be approximated because a multi-tasking operating system is used.

For a mixed system, which includes loosely-coupled and tightly-coupled code, the resolution equals that for the tightly-coupled setting, i.e., $RES_3 = RES_1$, and the missed count equals that for the loosely-coupled setting, i.e., $Miss_3 = Miss_2$. Thus, the minimal value of the time step that can be chosen is equal to the system resolution.

For tightly-coupled code, we measured the resolution for up to 100 PLCs and code sizes ranging from 1 to 1,000 if-instruction sets. The results shown in Figure 6 correspond to Model 2 with an execution time of 23.3 μs and native DLL-based PLC code. Of course, the resolution would automatically increase when using other models with longer execution times or other PLC code implementations. The resolution can also be increased by increasing the number of PLCs because more PLC code must be executed sequentially with the model. For instance, in the case of a single PLC and PLC code with 100 if-instructions, the resolution is 0.029 ms and increases up to 0.204 ms for 100 PLCs. However, the values generated by the model were not missed and the PLCs were able to react to all model changes.

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 (Figure 6). Unlike the tightly-coupled code scenario, it is necessary to consider that PLCs miss values gen-

erated by the model because the execution time is not synchronized between units, a multi-tasking operating system is used, and network communications introduce additional delays. Thus, providing a low miss rate at resolutions of 0.1 ms is difficult because of the multi-tasking operating system and the added overhead of network communications.

We chose nine time steps ranging from 0.1 ms to 1,000 ms and measured the average number of missed reads for each time step. In the experiment, each PLC read the remote memory, ran 100 if-instructions and then wrote 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 two read frequencies (1 read/time step and 3 reads/time step).

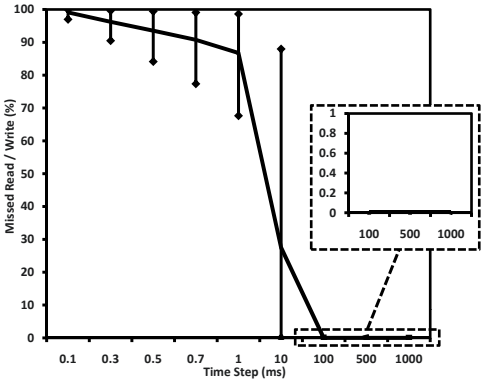
Figures 7(a) and 7(b) show the average (for 1 to 100 PLCs) measured percentages of missed reads for 1 read/time step and 3 reads/time step, respectively. For both settings, the average miss rate exceeds 50% for time steps smaller than 1 ms, mainly due to network delays, for which we measured a minimum value of 0.25 ms. For time steps larger than 10 ms, the percentage of missed reads became zero (even for 100 PLCs). Comparing the average values for the two settings (Figure 7(c)) shows a slight decrease in the value of the miss rate for a frequency of 1 read/time step. The reason is that reducing the number of reads decreases the number of simultaneous accesses to the synchronized PLC memory; thus, more PLCs can access the remote memory during each time step.

In summary, for the extreme case of 100 PLCs executing tightly-coupled code with each PLC running 1,000 if-instructions, the proposed framework provides a resolution of 0.638 ms with a miss rate of zero. On the other hand, achieving a miss rate of zero for the loosely-coupled case is only possible with time steps greater than 100 ms.

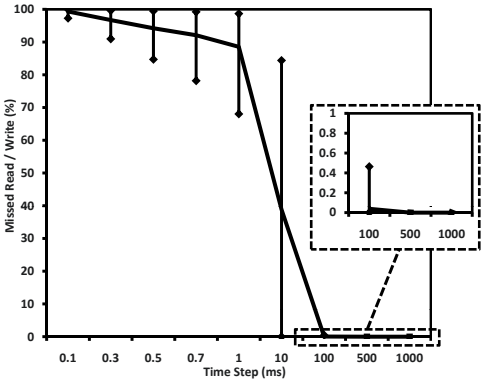
4.5 Measuring System Deviation

The proposed framework maintains the execution time synchronized with the operating system clock. Because of the multi-tasking environment used to run the framework, synchronization is affected by the number of PLCs and the chosen code coupling. The measured deviation for tightly-coupled code is shown in Figure 8. For a time step of 0.1 ms, the evolution of the deviation is shown as a dashed line. Note that when more than 20 PLCs are used, the deviation increases gradually and exceeds 1 ms for 30 PLCs. This is because the cumulative PLC code execution time exceeds the 0.1 ms time step. On the other hand, the deviation is decreased by increasing the time step – for time steps of 10 ms, 100 ms, 500 ms and 1 s, the deviation is maintained around 2.2 μs even for 100 PLCs.

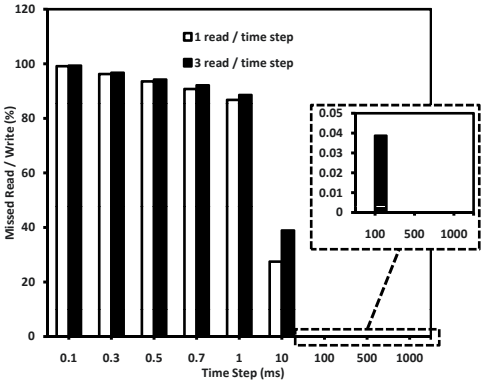
Figures 9(a) and 9(b) show the average measured deviations for loosely-coupled code with 1 read/time step and 3 reads/time step, respectively. For both time steps, the deviation increases starting with a 0.1 ms time step as more PLCs access the remote PLC memory. Starting with a 100 ms time step, the average deviation is maintained around 0.002 ms for 1 read/time step and



(a) 1 read/time step.



(b) 3 reads/time step.



(c) 1 read/time step vs. 3 reads/time step.

Figure 7. Average missed PLC read percentages.

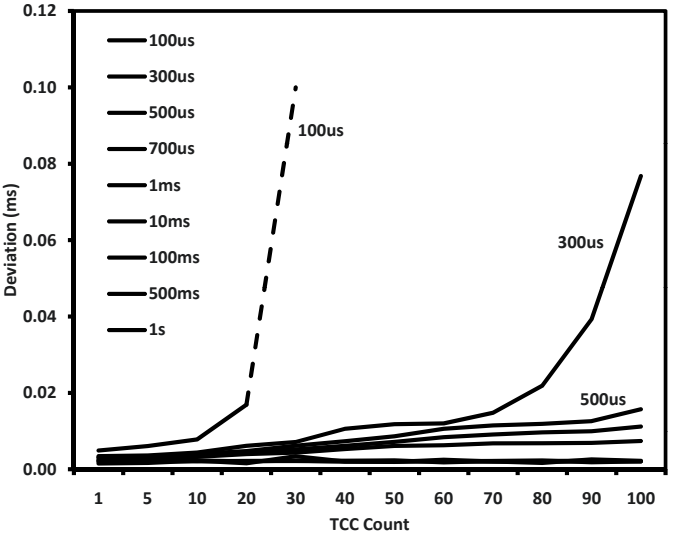


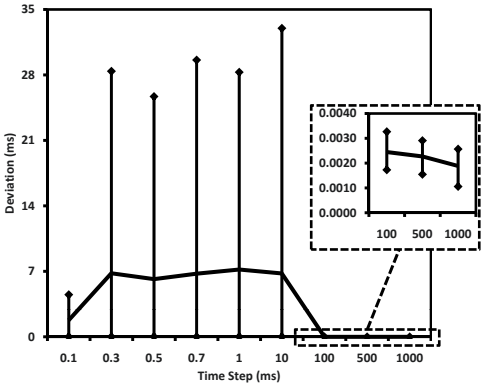
Figure 8. Deviation for tightly-coupled PLC code.

0.1 ms for 3 reads/time step. Comparing the average values for the two settings (Figure 9(c)) shows that a larger read frequency introduces larger deviations, an expected result. More specifically, the value of the deviation increases up to 50 times for a time step of 100 ms.

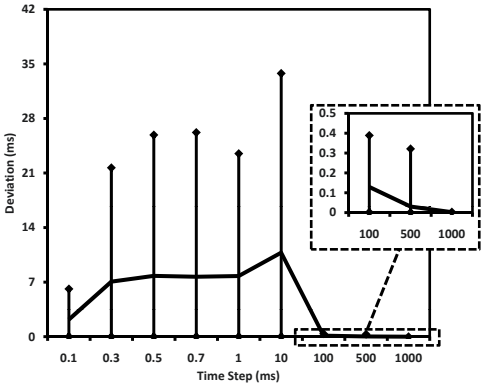
5. Conclusions

The framework for the security analysis of networked industrial control systems represents an advancement over existing approaches that either use real physical components combined with simulated/emulated components or a completely simulated system. The hybrid architecture of the framework uses emulation for protocols and components such as SCADA servers and PLCs, and simulation for the physical layer. This approach captures the complexity of information and communications devices and efficiently handles the complexity of the physical layer. Another novel feature of the framework is that it supports both tightly-coupled and loosely-coupled code. Tightly-coupled code is used when PLCs cannot afford to miss any events; loosely-coupled code permits the injection of new (malicious) code without stopping execution. Tightly-coupled code ensures lower resolution with a zero miss rate; on the other hand, loosely-coupled code permits the integration of complex PLC emulators without affecting the architectures of the other units.

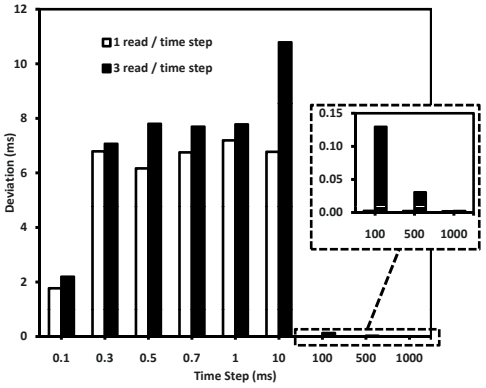
The models considered range from simple water purification plants to complex boiling water power plants. Experimental results demonstrate that the framework is capable of running complex models within tens of microseconds. With regard to parameters such as resolution, miss rate and deviation, tightly-



(a) 1 read/time step.



(b) 3 reads/time step.



(c) 1 read/time step vs. 3 reads/time step.

Figure 9. Average deviation for loosely-coupled PLC code.

coupled code provides higher resolution values, but with lower miss rates and deviations. On the other hand, loosely-coupled code yields lower resolution values with higher miss rates and deviations.

Our future research will use the framework to study the propagation of perturbations in cyber-physical environments, analyze the behavior of physical plants and develop countermeasures. It will also analyze the physical impact of attacks using more complex models that include descriptions of the physical components.

References

[1] A. Abdennour and K. Lee, An autonomous control system for boiler-turbine units, *IEEE Transactions on Energy Conversion*, vol. 11(2), pp. 401–406, 1996.

[2] R. Bell and K. Astrom, Dynamic Models for Boiler-Turbine Alternator Units: Data Logs and Parameter Estimation for a 160 MW Unit, Technical Report TFRT-3192, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1987.

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

[4] R. Chabukswar, B. Sinopoli, G. Karsai, A. Giani, H. Neema and A. Davis, Simulation of network attacks on SCADA systems, presented at the *First Workshop on Secure Control Systems*, 2010.

[5] P. Chawdhry and B. Hogg, Identification of boiler models, *IEE Proceedings on Control Theory and Applications*, vol. 136(5), pp. 261–271, 1989.

[6] C. Davis, J. Tate, H. Okhravi, C. Grier, T. Overbye and D. Nicol, SCADA cyber security testbed development, *Proceedings of the Thirty-Eighth North American Power Symposium*, pp. 483–488, 2006.

[7] S. East, J. Butts, M. Papa and S. Shenoi, A taxonomy of attacks on the DNP3 protocol, in *Critical Infrastructure Protection III*, C. Palmer and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 67–81, 2009.

[8] N. Falliere, L. O'Murchu and E. Chien, W32.Stuxnet Dossier, Symantec, Mountain View, California (www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf), 2011.

[9] M. Guglielmi, I. Nai Fovino, A. Perez-Garcia and C. Siaterlis, A preliminary study of a wireless process control network using emulation testbeds, *Proceedings of the Second International Conference on Mobile Lightweight Wireless Systems*, pp. 268–279, 2010.

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

[11] Y. Kim, M. Chung, J. Park and M. Chun, An experimental investigation of direct condensation of steam jet in subcooled water, *Journal of the Korean Nuclear Society*, vol. 29(1), pp. 45–57, 1997.

[12] A. Kumar, K. Sandhu, S. Jain and P. Kumar, Modeling and control of a micro-turbine-based distributed generation system, *International Journal of Circuits, Systems and Signal Processing*, vol. 3(2), pp. 65–72, 2009.

[13] J. McDonald and H. Kwatny, Design and analysis of boiler-turbine-generator controls using optimal linear regulator theory, *IEEE Transactions on Automatic Control*, vol. 18(3), pp. 202–209, 1973.

[14] 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*, vol. 2(4), pp. 139–145, 2009.

[15] 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 Third Conference on Human System Interaction*, pp. 679–686, 2010.

[16] S. Neema, T. Bapty, X. Koutsoukos, H. Neema, J. Sztipanovits and G. Karsai, Model-based integration and experimentation of information fusion and C2 systems, *Proceedings of the Twelfth International Conference on Information Fusion*, pp. 1958–1965, 2009.

[17] PowerWorld Corporation, Champaign, Illinois (www.powerworld.com).

[18] 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.

[19] H. Seifi and A. Seifi, An intelligent tutoring system for a power plant simulator, *Electric Power Systems Research*, vol. 62(3), pp. 161–171, 2002.

[20] W. Tan, H. Marquez, T. Chen and J. Liu, Analysis and control of a nonlinear boiler-turbine unit, *Journal of Process Control*, vol. 15(8), pp. 883–891, 2005.

[21] C. Wang, L. Fang and Y. Dai, A simulation environment for SCADA security analysis and assessment, *Proceedings of the International Conference on Measuring Technology and Mechatronics Automation*, vol. 1, pp. 342–347, 2010.

[22] 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.