A Cyber-Physical Experimentation Environment for the Security Analysis of Networked Industrial Control Systems

Béla Genge, Christos Siaterlis, Igor Nai Fovino[1], Marcelo Masera[1]

Institute for the Protection and Security of the Citizen, Joint Research Centre, Ispra, 21027, Italy

Abstract

Although many studies address the security of Networked Industrial Control Systems (NICS), today we still lack an efficient way to conduct scientific experiments that measure the impact of attacks against both the physical and the cyber parts of these systems. This paper presents an innovative framework for an experimentation environment that can reproduce concurrently physical and cyber systems. The proposed approach uses an emulation testbed based on Emulab to recreate cyber components and a real-time simulator, based on Simulink, to recreate physical processes. The main novelty of the proposed framework is that it provides a set of experimental capabilities that are missing from other approaches, e.g. safe experimentation with real malware, flexibility to use different physical processes. The feasibility of the approach is confirmed by the development of a fully functional prototype, while its applicability is proven through two case studies of industrial systems from the electrical and chemical domain.

Keywords: Infrastructure protection, network security, testbed, process control systems

1. Introduction

Modern Critical Infrastructures (CIs), e.g. power plants, water plants and smart grids, rely on Information and Communications 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 CIs were isolated environments and used proprietary hardware and protocols, limiting thus the threats that could affect them. Nowadays CIs, or 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, 3]. For example, the recently reported Stuxnet worm [4] is the first malware specifically designed to attack NICS. Its ability to reprogram the logic of control hardware in order to alter the operation of industrial processes demonstrated how powerful such threats can be; it served

Email addresses: bela.genge@jrc.ec.europa.eu (Béla Genge), christos.siaterlis@jrc.ec.europa.eu (Christos Siaterlis), igor.nai@jrc.ec.europa.eu (Igor Nai Fovino), marcelo.masera@jrc.ec.europa.eu (Marcelo Masera)

[1]Currently, the author affiliations have changed as described in the author bios

as a wakeup call for the international security community. Stuxnet raised many open questions, but most importantly it highlighted the lack of an efficient scientific approach to conduct experiments that measure the impact of cyber threats against both the physical and the cyber parts of CIs.

This paper addresses exactly this gap and proposes a novel architecture for experimenting with NICS that is in between the two extremes of pure simulation and experimentation with only real components. The proposed framework uses simulation for the physical components and an emulation testbed based on Emulab [5, 6] in order to recreate the cyber part of NICS, e.g. SCADA servers, corporate network. The models of the physical systems are developed in Matlab Simulink from which the corresponding 'C' code is generated using Matlab Real Time Workshop. The generated code is then executed in real-time and is able to interact with the real components of the emulation testbed.

The main novelty of the proposed framework is that it combines the Emulab testbed with typical NICS components and Simulink models in order to provide a set of key functionalities that are missing from related techniques [7, 8, 9, 10, 11, 12, 13, 14]. As the Emulab testbed was specifically designed to provide capabilities for advanced networking experimentation, it already includes most of the functionalities required by our framework (see Table 2), e.g. event injection, experiment automation, separated experimental and control panes. Although these satisfy the requirements of a cyber security experimentation environment, they do not satisfy all the requirements of a cyber-physical experimentation environment that would enable security experimentation with NICS. Therefore, the proposed framework extends the previous list of functionalities as it supports: experimentation with a wide range of physical processes, typical NICS components, use/testing of real malware/SCADA software, safe resilience/security studies, and high cyber-layer fidelity. Most of the previously mentioned functionalities are missing from related approaches that employ real cyber and simulated physical components [12, 14], as shown later in this paper. On the other hand, approaches that rely on simulated cyber and physical components [11, 10] are less expensive and most of them include several functionalities among the previously mentioned. However, they do not support key functionalities that would enable experimentation with real malware/SCADA software. Finally, simulated cyber components, e.g. in NS-2, might model normal operations, but they fail to capture the complexity of real components, such as complex interactions between heterogeneous software/malware and hardware [15].

The results concerning the performance of the proposed framework show that it can enable experimentation with complex physical processes ranging from simple power plants to complex power grids. Moreover, a series of performance evaluation experiments show that the framework can scale up and accurately recreate large NICS, e.g. having up to 100 Programmable Logic Controllers (PLCs). The applicability of the proposed framework is proven with two case studies. The first one includes an analysis of the effect that a Stuxnet-like [4] malware would have on a power plant [16], while the second study focuses on the effect that network conditions have on the impact of cyber attacks targeting a chemical plant [17].

The paper is structured as follows. After an overview of related work in Section 2, the requirements

Table 1: Terms & Abbreviations

AAA	Authentication, Authorization and Accounting
CI	Critical Infrastructures
ICT	Information and Communications Technologies
L-PLC	Local Programmable Logical Controller
LCC	Loosely Coupled Code
NICS	Networked Industrial Control Systems
OPC	Object linking and embedding for Process Control
OS	Operating System
PLC	Programmable Logical Controller
R-PLC	Remote Programmable Logical Controller
SC	Simulation Core
SCADA	Supervisory Control And Data Acquisition
SDL	Shut Down Limit
SDT	Shut Down Time
TCC	Tightly Coupled Code

for building a new NICS security experimentation framework are discussed in Section 3. This is followed by a detailed presentation of the proposed framework in Section 4, by a comparison to state of the art approaches in Section 5 and by performance evaluation results in Section 6. The two case studies showing the applicability of the proposed framework are presented in Section 7 and the paper concludes in Section 8. A list of terms & abbreviations is given in Table 1.

2. Related Work

The analysis of security & resilience characteristics of NICS is challenging because such systems include elements that interact in both the physical and the cyber domains. The approaches found in the literature for the study of cyber and physical systems vary considerably regarding the use of simulators and real components within experiments. This section provides a brief presentation of the most relevant ones.

An approach that uses real components for the physical parts of NICS and partly simulated ones for the cyber parts was proposed by Chunlei, *et al.* [7]. This approach uses a real OPC (Object linking and embedding for Process Control) server, the NS-2 network simulator, combined with real PLCs and field devices to analyze NICS. In this approach the only simulated element is the enterprise network, while all the other components, e.g. servers, PLCs, are real. Although from one point of view such a testbed would provide 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, because that would require a complete industrial system to experiment with. Nai Fovino, *et al.* [8] went also in this direction, by 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 and the high cost of maintenance of a similar architecture. Another approach that uses real physical devices was proposed by Queiroz, *et al.* [9]. In this case only the sensors and actuators are real physical devices, while the remaining components such as PLCs, and the communications protocols between them are implemented as OMNeT++ modules. Compared to the these approaches, in the proposed framework we use simulation for the physical layer in order to enable disruptive security/resilience studies on NICS. As already stated, such studies are not always possible with the previous approaches.

Other researchers focused on simulating both SCADA and field devices. For example, Chabukswar, *et al.* [10] used the Command and Control WindTunnel [18] multi-model simulation environment 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 process model. With this approach, the analysis of cyber-physical effects caused by real malware is not a trivial task, as it requires a detailed description of all ICT components. Furthermore, it might also need a detailed knowledge on the malware dynamics, that is not always available. In the same category we find the work of Hopkinson, *et al.* [11], in which power system simulators were coupled together with the NS-2 simulator. This approach suffers from the same disadvantages as the solution proposed by Chabukswar, *et al.*, that we overcome in the proposed framework by employing real components in the cyber layer.

There are also approaches that do not focus on entire NICS, but only on specific components. For instance, Davis, *et al.* [12] used PowerWorld [19] to model an entire power grid and to run it in real-time. In this approach the PowerWorld server is connected to a proxy and uses the Modbus protocol to communicate with client applications. Researchers interact with the PowerWorld server through a visual interface that allows them to introduce disturbances into the network and to observe the effects. Although it includes a real network with real PCs and a simulator for physical components, this approach does not provide typical units such as PLCs and SCADA Masters, that are key components within NICS. In the same category we find the work of Hiyama and Ueno [13] that employed Simulink to model physical systems and Matlab Real Time Workshop to run the model in real-time. Although Hiyama and Ueno did not focus on all the components of NICS, their work shows that Matlab can be an effective tool to model and run physical processes in real-time. Finally, we mention the solution proposed by McDonald, *et al.* [14], in which the physical layer was simulated with the PowerWorld server, while the cyber layer included both simulated and real components. This approach is similar to the one proposed in this paper in the sense that it simulates the physical layer and uses real components for the cyber layer. On the other hand, the solution proposed by McDonald, *et al.* does not support a wide variety of physical processes and is constrained on Power Systems. Moreover, the framework proposed in this paper includes real networks and automated techniques for managing experiments, that are missing from McDonald's approach.

The present work builds on the advantages of the previously mentioned methods. For the physical part, instead of using real components as was the case of [7], [8] and [9], it uses simulation. This provides an efficient, safe and low-cost approach with fast and accurate analysis capabilities. For the cyber part of NICS this work adopted an emulation approach based on Emulab [5, 6]. This approach is well-established in the field of cyber security [20] and was chosen in order to overcome the major difficulties that rise while trying to simulate how ICT components behave under attacks or failures.

3. Requirements Analysis

In this section we analyze the requirements behind the design of a new NICS security experimentation framework. We start out with the description of a typical NICS architecture and we continue with the requirements for building a cyber-physical security experimentation framework.

3.1. Architecture of a Typical Networked Industrial Control System

Modern NICS architectures have two different control layers: (i) the physical layer, which comprises actuators, sensors and hardware devices that physically perform the actions on the system, e.g. open a valve; and (ii) the cyber layer, which comprises all the information and communications devices together with their software that acquire data, elaborate low-level process strategies and deliver the commands to the physical layer. The cyber layer typically uses SCADA protocols, e.g. Modbus, 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), human-machine interfaces, and domain controllers. 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 (Programmable Logic Controllers), as shown in Figure 1. From an operational point of view, PLCs receive data from the physical layer, elaborate a local actuation strategy, and send commands to the actuators. 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.

3.2. Analysis of the Required Functionalities

Ideally, an experimental framework for NICS security research would support the execution of complex, large scale and disruptive experiments using rigorous scientific methods. The implemented functionalities

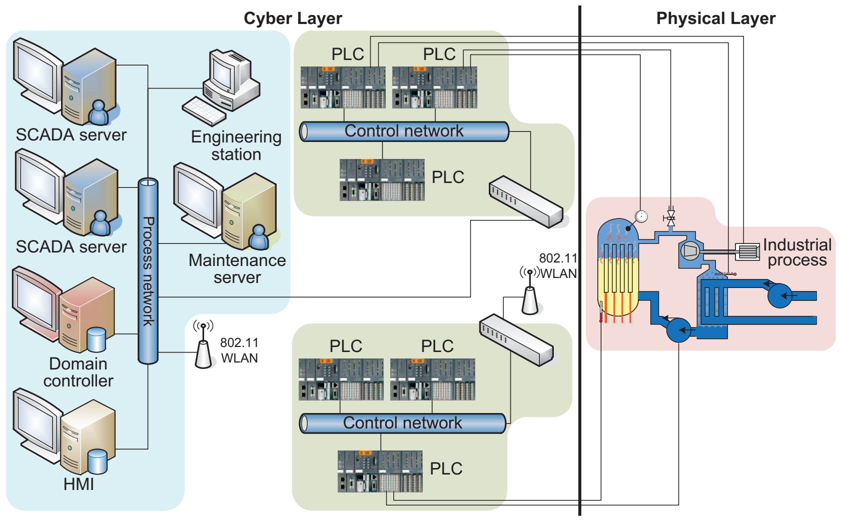


Figure 1: Typical architecture of a Networked Industrial Control System

should include not only typical NICS components, but should also cover capabilities that facilitate the experimentation process. These capabilities are specific to Internet experimentation testbeds and include a wide range of aspects such as control of the experiment's environment, repeatable experiments, experiment automation, and secure remote access, if needed. The complete list of required functionalities is provided in Table 2. For a more detailed presentation of Internet security testbed-related functionalities the reader should consult our previous work [21]. A presentation of the extended functionalities follows:

- Support a wide range of physical processes. A key requirement of the proposed framework is to support more than one type of physical process. That is, the framework must include capabilities to enable experimentation with different CIs ranging from power systems, to chemical systems and even hydraulic systems. In practice it is not unusual to find physical installations with mixed processes.

- Support typical NICS components. As the main goal of the framework is to enable experimentation with NICS, it must reproduce the functionalities of typical NICS. These include components such as PLCs and Master units, but also include industrial protocols such as Modbus, DNP3, or Profibus.

- Support real malware/SCADA software. Security experiments usually involve the presence of an adversary that employs malicious software to reach his/her goals. In this sense the framework should facilitate experimentation with real malware and SCADA software in order to recreate a real environment as faithfully as possible.

- Support high fidelity cyber/physical layers. The proposed NICS security experimentation framework will be applied to a wide range of security studies, that could also include the development and

Table 2: Complete set of required functionalities for cyber-physical experimentation

	ID	Functionality
Specific to NICS	F1	Support a wide range of physical processes
	F2	Support typical NICS components
	F3	Support real malware/SCADA software
	F4	Support high fidelity cyber/physical layers
	F5	Support safe security/resilience experiments
Specific to experiment management	F6	Support control of the experiment's environment
	F7	Support experiment clock and event scheduling
	F8	Support separate control, measurement and experiment planes
	F9	Support experiment description and data storing
	F10	Support repeatable experiments
	F11	Support experiment automation and rapid reconfiguration
	F12	Support extensibility in order to adapt to future needs and requirements
	F13	Support heterogeneity of technologies
	F14	Support clean experiment reconfiguration
	F15	Support monitoring of resources
	F16	Support remote access
	F17	Support Authentication, Authorization and Accounting (AAA)

validation of countermeasures against cyber-physical attacks. Consequently, it must reproduce as accurately as possible real scenarios and architectures.

- Support safe security/resilience experiments. A security experimentation framework must provide the ability to conduct security/resilience experiments in which malware could cause unpredictable effects to physical processes. Therefore, the experimentation framework must provide the ability to conduct disruptive experiments on physical processes in a safe manner.

4. Architecture of the Proposed Framework

This section presents a new framework for conducting security experiments on NICS. Our presentation starts with an architectural overview and continues with a more detailed discussion on the proposed experimentation framework.

4.1. Overview of the Proposed Experimentation Framework

In order to support the functionalities listed in Table 2 the architecture of the proposed framework employs the Emulab-based testbed to recreate the control and process network of NICS and a software simulation to reproduce the physical processes. 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, as shown in Figure 2, clearly distinguishes three layers: the cyber layer, the physical layer and a link layer in between. The cyber layer includes regular ICT components 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.

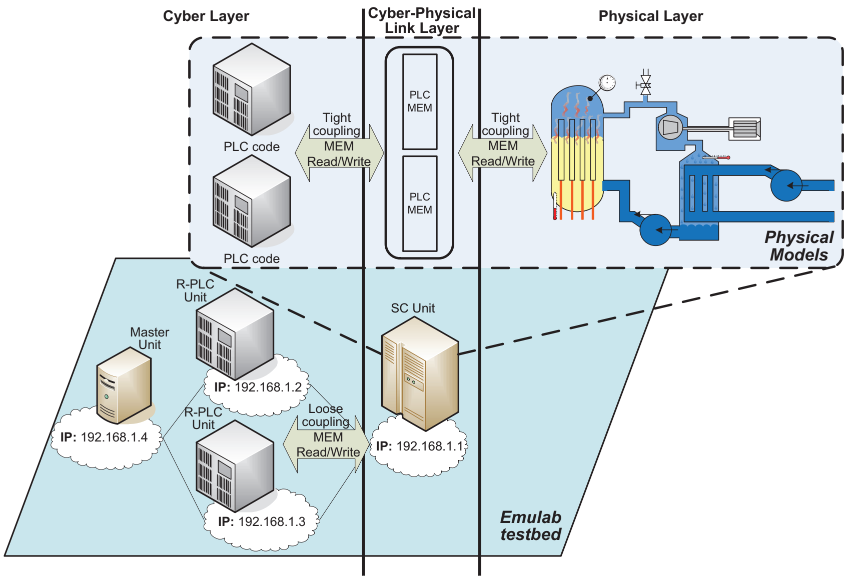


Figure 2: Architectural overview of the proposed framework

The physical layer is recreated through a real-time simulator that runs on a server and executes the model of a physical system. The simulator's execution time is strongly coupled to the timing service of the underlying operating system (OS). As the OS uses multitasking, achieving hard real-time is difficult without the use of kernel drivers. However, soft real-time is achieved by allowing a certain deviation from the OS clock. Throughout the paper the term *time step* is used to denote the time between two successive executions of the physical model in the simulator.

The cyber layer is recreated by an emulation testbed that uses the Emulab architecture and software [5] 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 [6]. Besides the process network, the cyber layer also includes the control logic code, that in the real world is implemented by PLCs. In the proposed approach the control code can be run sequentially or in parallel to the physical model. In the sequential case, a *tightly coupled code* (TCC) is used, i.e. code that is running in the same memory space with the model. In the parallel case a *loosely coupled code* (LCC) is used, i.e. code that is running in another address space, possibly on another host. The main advantage of TCCs is that these do not miss values generated by the model between executions. On the other hand, LCCs allow running PLC code remotely, to inject (malicious) code without stopping execution of the model, and to run more complex PLC emulators. As control code is implemented within TCCs and LCCs, throughout the paper these terms are used to denote PLCs.

The cyber-physical layer incorporates the PLC memory and the communications interfaces that glue together the other two layers. The PLC memory is seen as a set of registers typical to PLCs and provides the link to the inputs and outputs of the physical model, e.g. valves and sensors.

4.2. Detailed Architectural Description

As already mentioned, the proposed architecture uses simulation for the physical layer and an emulation testbed for the cyber layer. The simulated physical process model is run in soft real-time but is open for interaction with other components, e.g. PLCs, by providing access to model inputs, e.g. valves, and outputs, e.g. sensors. As shown in Figure 2, the simulation of the process model is implemented in the *Simulation Core* (SC) unit. This unit links the cyber and physical layers through a synchronized memory region that maps each input and output of the model to a set of memory registers. Another feature of the SC unit is the support for running TCCs sequentially with the process models, thus emulating the behavior of real PLCs that are able to react to events without missing any values generated by the model. As TCCs run sequentially with the model, their reaction time depends on the execution time of the model and on the chosen time step, as defined by human operators.

Although TCCs enable running emulated PLC code they: (i) lack the flexibility required for injecting new (malicious) code; and (ii) lack the extensibility in order to support more complex PLC emulators in the future. For this reason the framework foresees the *Remote PLC* (R-PLC) unit. R-PLCs run emulated PLC code remotely and interact with the SC through a set of communications modules. In addition, R-PLC units implement standard SCADA protocols, e.g. Modbus, that enable Master units to interact with the control network. The main role of the *Master* unit is to take global decisions based on values received from sensors through R-PLC units. Their decisions are translated to commands sent to R-PLC units and thereafter to the actuators of the physical model. In the remaining of this subsection we provide a detailed description of the framework's NICS components (see Figure 3).

SC Unit. The main role of the SC unit is to provide a soft real-time execution of TCCs and physical

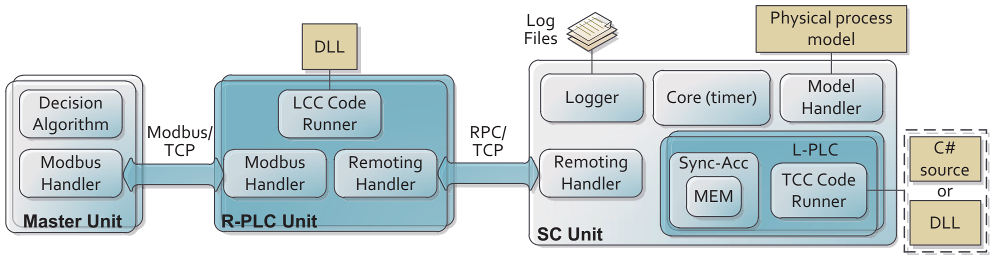


Figure 3: Modular architecture of NICS components within the proposed framework

models, synchronized with the clock of the OS. At the same time the SC unit provides the glue between the cyber and the physical layers. 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. digital input registers, used as the glue between the cyber and physical layers and the TCC runner module. The *Remoting Handler* module handles the communications 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, providing at the same time a soft real-time execution of the physical model.

R-PLC Unit. The main role of the R-PLC unit is to run LCC code and to provide a Modbus interface for accessing the physical process model. Its main modules include: the *Remoting Handler* module that implements the communications with the SC unit; the *LCC Runner* module that runs the LCC; 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 algorithm based on the sensor values received from the R-PLC units. Such algorithms are implemented within the *Decision Algorithm* module, while the *Modbus Handler* module enables communications with the R-PLC unit.

4.3. Implementation Details

Prototypes of SC, R-PLC and Master units have been developed in C# (Windows) and have been ported and tested on Unix-based systems (FreeBSD, Fedora and Ubuntu) with the help of the *Mono* platform. The implementation allows TCCs to be provided either as C# source files or as binary DLLs, both dynamically loaded at run-time. At this time, LCCs are written in C# and must be compiled together with the SC unit. For the physical process simulator we used Matlab Simulink, since it is a general simulation environment for dynamic and embedded systems and covers a wide variety of physical processes, e.g. power plants, gas plants. From Simulink models the corresponding 'C' code is generated using Matlab Real Time Workshop and is integrated into the framework using an XML configuration file. The communications between SC and R-PLC units are handled by .NET's binary implementation of RPC (called *remoting*) over TCP. Currently, the communications between the R-PLC and the Master units are handled by Modbus/TCP. However, other protocols can be added by substituting the Modbus Handler modules.

5. Comparison to State of the Art

Since the focus of the framework is not on a specific model or simulation, the comparison of numerical results is beyond the scope of this paper. We consider that the most appropriate and informative way to confront the different approaches that comprise the state of the art is through the following three perspectives: (i) a qualitative, feature-driven comparison; (ii) a comparison of capital and operational costs; and (iii) an experimental scenario-based comparison. All these aspects are detailed throughout this section.

5.1. Qualitative Comparison to Related Approaches

As already mentioned in the previous sections, the functionalities provided by the proposed framework are only partly supported by related approaches. This subsection provides a detailed comparison of the functionalities found in related approaches to the ones provided by the proposed approach. The comparison is based on a qualitative evaluation, where *High* denotes a strong support for a specific functionality while *Low* denotes a weak functionality support. In case a specific functionality is completely missing we use the "–" symbol. In the following analysis approaches are identified by the first author's surname.

We categorized related approaches based on the use of real or simulated components for the cyber and the physical layer. In this sense, we identified three categories of cyber layer and two categories of physical layer implementations. For cyber layer implementations the following approaches were found: only real components (Nai Fovino [8], Davis [12]); real components combined with simulated ones (Chunlei [7], Queiroz [9], McDonald [14]); and only simulated components (Chabukswar [10], Hopkinson [11]). On the other hand, for physical layer implementations we found the following approaches: only real components (Nai Fovino [8], Chunlei [7], Queiroz [9]); and only simulated components (Davis [12], McDonald [14], Chabukswar [10], Hopkinson [11]). In McDonald's approach we found that the physical layer includes both real and simulated components. However, the presented experiments employed only the PowerWorld simulation server for conducting disruptive studies on physical processes. Therefore, McDonald's approach was included in the category of simulated physical layers. These results are summarized in Table 3.

It can be seen that approaches in which the cyber layer consists exclusively of real components, i.e. Nai Fovino and Davis, provide few experiment management capabilities. Furthermore, these approaches are dedicated to specific physical processes and do not enable experimentation with other types of processes. In contrast, the framework proposed in this paper includes a strong support for a wide range of physical processes, accompanied by advanced experiment management capabilities.

By combining multiple simulators together with real components within the cyber layer, the number of provided functionalities increases in the approaches of Chunlei, Queiroz and McDonald. Nevertheless, compared to the proposed framework, these approaches have several limitations: they are mainly targeted towards a specific domain, i.e. Power Systems, they have a weak support for real malware/SCADA software, and they include only a limited set of experiment management functionalities.

In the context of approaches that use simulators for both cyber and physical layers without any real components, i.e. Chabukswar and Hopkinson, we see an increase in the number of strongly supported functionalities. However, these approaches have a limited applicability in the field of cyber security due to the diversity and complexity of protocols, systems and architectures. Furthermore, both approaches found in this category do not enable experimentation with real malware/SCADA software, and provide a low cyber layer fidelity. In contrast, these functionalities are strongly supported in the proposed framework.

Table 3: Functionality comparison to related work ("C_R" is real cyber; "C_S" is simulated cyber; "P_R" is real physical; "P_S" is simulated physical)

			C_R & P_R	C_R & P_S	C_{RS} & P_R		C_{RS} & P_S	C_S & P_S	
	Functionality (see Table 2)	**Ours**	[8]	[12]	[7]	[9]	[14]	[10]	[11]
Specific to NICS	Power systems	**H**	H	H	L	L	H	H	H
	Chemical systems	**H**	L	–	L	L	–	H	–
	Mechanical systems	**H**	L	–	L	L	–	H	–
	Hydraulic systems	**H**	H	–	L	L	–	H	–
	Typical NICS	**H**	H	L	H	H	H	H	H
	Real malware/SCADA	**H**	H	H	L	–	L	–	–
	Security/resilience studies	**H**	L	H	–	–	H	H	H
	Cyber layer fidelity	**H**	H	H	L	L	L	L	L
	Physical layer fidelity	**L**	H	L	H	H	L	L	L
Specific to experiment management	Environment control	**H**	H	L	L	L	L	H	H
	Event scheduling	**H**	–	–	L	L	L	H	H
	Panes separation	**H**	–	–	–	–	L	H	H
	Storage facilities	**H**	L	–	–	–	–	–	–
	Repeatable experiments	**H**	L	–	–	–	L	H	H
	Automation	**H**	–	–	–	–	L	H	H
	Extensibility	**H**	L	L	L	L	L	H	H
	Heterogeneity	**H**	–	H	–	–	L	–	–
	Clean reconfiguration	**H**	–	L	–	–	L	H	H
	Resource monitoring	**H**	L	–	–	–	–	–	–
	Remote access	**H**	L	–	–	–	–	–	–
	AAA	**H**	–	–	–	–	–	–	–

Based on this analysis we can clearly state that the proposed framework advances the state of the art by providing a wide range of functionalities, that are missing in related approaches. The low fidelity of the physical layer, that is the main disadvantage of the proposed framework, is counterbalanced by its high flexibility, and non-existent maintenance costs. Furthermore, with this approach we enable the safe execution of disruptive security and resilience experiments, that is a key requirement in the analysis of today's NICS.

5.2. Cost Comparison to Related Approaches

Following this subsection we show that by employing the Emulab testbed the framework's capital expenses are counterbalanced by its low operational expenses. For this purpose we compare the cost of our proposal to the cost of an ad-hoc testbed that includes similar NICS-related functionalities to ours, but that does not provide all the experimentation functionalities available within Emulab. Such an ad-hoc testbed could be compared to the ones proposed by Davis, *et al.* [12] and McDonald, *et al.* [14], in the sense that both approaches use real components for the cyber layer and simulated components for the physical layer. Consequently, the results presented in this subsection can be extrapolated to other approaches combining real cyber layers with simulated physical layers and that do not rely on automated experiment management.

The overall cost of running N experiments with the proposed framework can be expressed as $\$(\gamma_{exp}) = \$(\gamma_{proposed}) + N * \$(\gamma_{expsetup})$, where the $\$()$ function is used to estimate the cost of components, $\gamma_{proposed}$ is the complete framework, γ_{exp} denotes N experiments and $\gamma_{expsetup}$ denotes the man-hours needed to set up these experiments. In contrast, the overall cost of experimenting with the ad-hoc testbed includes an additional element $N * \$(\gamma'_{manage})$, in which the γ'_{manage} component denotes the man-hours needed for managing one experiment, e.g. restoring experiment data. In our approach these aspects are automatically handled by the Emulab software and therefore their costs are assumed to be considerably smaller.

Next, we provide a brief presentation of estimated capital and operational expenses for the two testbeds. For more details the reader is asked to consult our previous work [22]. In our estimations we assumed 100 experimental nodes with identical costs in the two testbeds. In terms of capital investments, the proposed framework needs additional, more expensive components, that include the two Emulab servers and control/experimental switches compatible with the Emulab software. Based on these assumptions we estimate the overall experimentation costs (given in EUR) in the two testbeds for $N = 100$ and $N = 300$ yearly conducted experiments. As shown in Figure 4 (a) and (b), the number of yearly conducted experiments clearly influences the overall experimentation costs. For $N = 100$ and less-expensive experiments, e.g. 300 per experiment, the costs of the ad-hoc testbed exceed the costs of the proposed framework only after 5 years. On the other hand, for more complex experiments, e.g. translated to costs of 800 per experiment, the high operational costs of the ad-hoc testbed exceed the costs of the proposed framework in less than one year. The same result is achieved by increasing the number of yearly conducted experiments to $N = 300$.

These estimations show that for a reduced number of yearly conducted experiments, e.g. $N < 100$, and simple experiments, the deployment of the proposed framework might not be feasible. Nevertheless, in practice the exploration of parameters state space, e.g. bandwidth, delays, in only one scenario can increase the number of yearly conducted experiments to more than 300. Consequently, for the study of such scenarios the deployment of the proposed framework is highly recommended. Furthermore, the functionalities provided by Emulab also minimize the number of time-consuming and error-prone configurations done by human operators. This translates to less hours and unavoidably to less money spent debugging and re-running experiments in order to correct human errors. These aspects have not been taken into account in the previous estimations and could further increase the operational costs of the ad-hoc testbed.

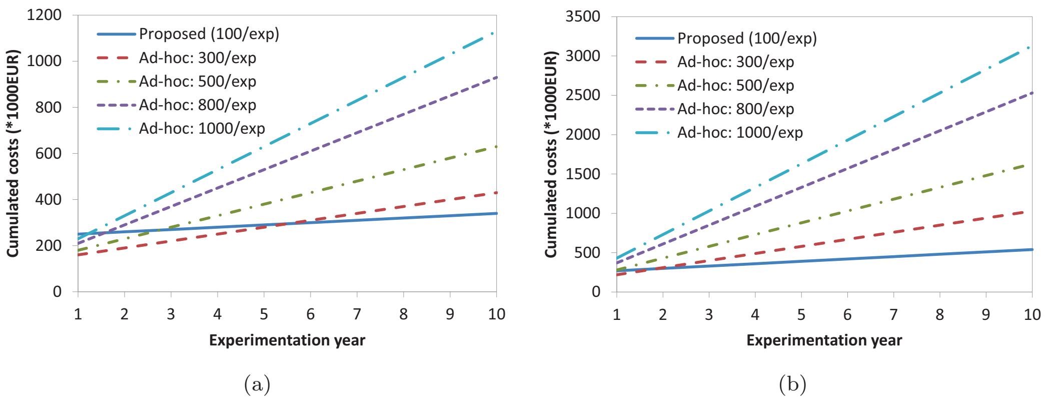
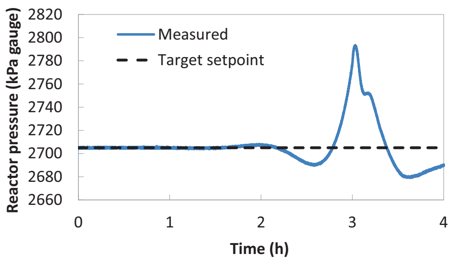


Figure 4: Estimated costs in the proposed framework and an ad-hoc testbed: (a) N=100, and (b) N=300

5.3. Experimental Comparison to Related Approaches

As a final comparison we show that the proposed framework can effectively recreate scenarios similar to the ones reported by state of the art approaches. More specifically, we recreate the Denial of Service attack reported in the work of Chabukswar, *et al.* [10] involving the Tennessee-Eastman chemical process [17] (more details on this process in Section 7.2). The attack targets routers responsible for the data exchange between controllers and sensors/actuators. By employing a TCP SYN attack the controller is completely blinded and is not able to connect to sensors. In our experiment controllers were implemented within the Master Unit, while sensors were implemented as R-PLC Units. As shown in Figure 5, similarly to the results reported in [10], we see an increase in the pressure after the attack is started. As soon as the attack is ended, the controller is able to stabilize the process and brings it back to its normal operating limits. This experimental comparison confirms once again that the proposed framework is able to recreate complex scenarios that are similar to the ones reported in state of the art, with the added value discussed in the previous sections.

Figure 5: Effect of a Denial of Service attack on data routers (similar to the one reported in [10])



6. Performance Evaluation

The results from this section are provided to researchers in order to guide them in setting the framework's parameters in their own experiments. Unfortunately, the description of related approaches [12, 7, 9, 14] does not include a thorough analysis and researchers must identify on their own the capabilities and limitations of such approaches. In contrast, this section presents a clear view on the capabilities and limitations of the proposed framework. For this purpose the implemented prototype was evaluated by measuring three metrics: *resolution*, *miss rate* and *deviation*. *Resolution* is the minimal execution step of the physical model simulator or in other words the minimal value of the *time step*, i.e. the time between two model executions. As the model is run sequentially with TCCs, the value of the resolution equals the sum of the execution time of the model itself and the execution time of TCCs. Beyond the resolution it is also important to measure the amount of values that are generated by the model and are not received by LCCs (because of network delays and multitasking OSs). The term *miss rate* is used to denote the percentage of missed read/write operations by LCCs in each time step. As the intent of the framework is to provide a soft real-time execution of the model, it also provides an approximate value for the *deviation*, i.e. the difference between the model execution time and the OS clock. As shown in this section, the deviation depends highly on the chosen time step and the number of LCCs and TCCs present in the system.

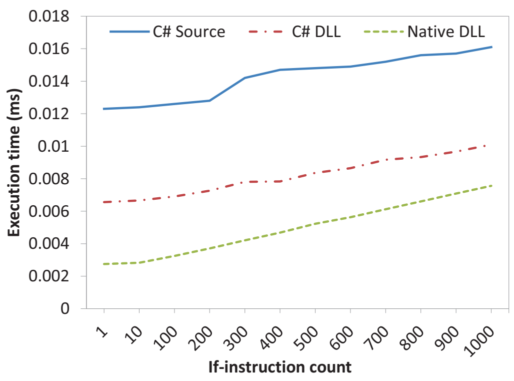
The prototype of the proposed framework was developed and evaluated in the Joint Research Centre's Experimental Platform for Internet Contingencies (EPIC) laboratory. The Emulab testbed includes nodes with the following configuration: FreeBSD OS 8, AMD Athlon Dual Core CPU at 2.3GHz and 4GB of RAM. In total, the experimental setup consisted of 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. The experimental setup also included four Simulink models: (i) a simplified model of a water purifying plant; (ii) Bell and Åström's oil-fired power plant [16]; (iii) the Tennessee-Eastman chemical process [17]; and (iv) the IEEE 9bus power grid test system.

6.1. Physical Process Model Execution Time

The execution time of the physical models depends on their dimensions, e.g. number of equations, mathematical operations. While the execution time of the water plant and Bell and Åström's models is less

Table 4: Execution time of different physical process models

	Water plant	Bell and Åström	Tennessee-Eastman	IEEE 9bus
Time (ms)	0.0192	0.0233	0.277	0.25



	Code execution time – t_{tcc} (ms)		
Instr count	C# Source	C# DLL	Native DLL
1	0.0123	0.0065	0.0027
100	0.0126	0.0069	0.0032
300	0.0142	0.0078	0.0042
700	0.0152	0.0091	0.0061
900	0.0157	0.0096	0.0071
1000	0.0161	0.0109	0.0075

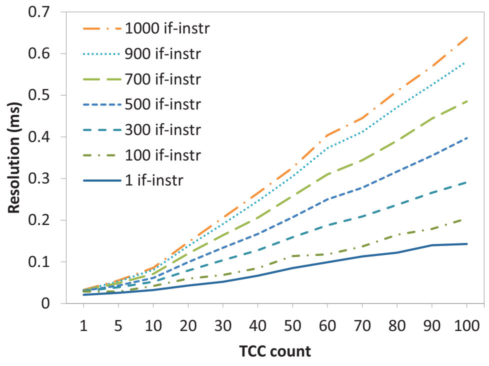
Figure 6: Execution time of TCCs as a function of the implementation type and number of instructions

than 0.1ms, the execution time of the Tennessee-Eastman model is 0.277ms, and is caused by the complexity of the 50 state equations defined within the model. Finally, we measured the execution time of the IEEE 9bus power grid test system that was of 0.25ms, a reasonable value considering that this model includes 3 power generators. These results are depicted in Table 4.

Although the execution time of the physical models included in the performance evaluation was below 0.3ms, it is reasonable to assume that more complex models could run above 1ms or even 1s. In order to enable real-time execution of such models as well, researchers could increase the computing power of the host running the SC unit, or if possible, they could optimize or even parallelize the execution of models.

6.2. Resolution

As mentioned earlier in this section, the minimal value of the time step is called *resolution*, computed as a sum of the model and TCCs execution time, i.e. $RES = t_{model} + \sum_i t^i_{tcc}$. The value of t_{tcc} depends on the type of implementation used for TCCs and the number of *if* instructions (see Figure 6). As PLC code commonly consists of a large number of test sequences, e.g. position of a valve, *if-then-else statements* (*if-instructions*) were used to emulate PLC code. The experimental results showed that the C# source file has the largest execution time as the code is compiled at run-time. In contrast, 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 C# source files is that these do not require the presence of development libraries, changes can be made quickly and experiments can be resumed immediately.



	Resolution (ms)			
TCC count	1 Instr	300 Instr	700 Instr	1000 Instr
1	0.025	0.029	0.031	0.032
10	0.032	0.052	0.071	0.085
30	0.052	0.104	0.164	0.206
70	0.113	0.188	0.344	0.445
90	0.140	0.237	0.444	0.569
100	0.143	0.291	0.485	0.638

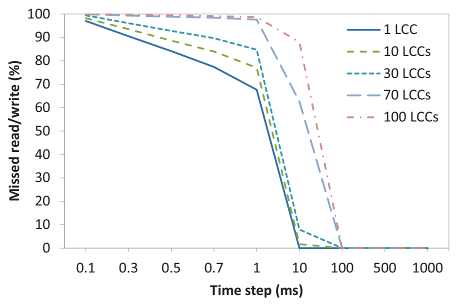
Figure 7: The computed resolution with the Bell and Åström physical process model

Finally, the resolution was measured for several settings involving up to 100 TCCs and code sizes ranging from 1 to 1000 *if-instructions*. The following results correspond to the Bell and Åström physical process model and native DLL-based code. Of course, by using other models or a different implementation type for TCCs the resolution changes automatically. As expected, the value of the resolution has a linear evolution that increases with the number of TCCs and the number of *if-instructions* defined for each TCC. For instance, in the case of a single TCC containing 1 *if-instruction* the resolution is 0.025ms and increases up to 0.143ms for 100 TCCs. On the other hand, in the case of a single TCC containing 1000 *if-instructions* the resolution is 0.032ms and increases up to 0.638ms for 100 TCCs. These results are shown in Figure 7.

The presented measurements show that the developed prototype is able to run complex models together with control logic code provided in the form of TCCs under 0.638ms. The maximal measured value for the resolution, i.e. 0.638ms, corresponds to the extreme case of 100 TCCs with 1000 *if-instructions* each. However, in real scenarios the system does not usually include more than 30 or 50 PLCs, thus resolutions below 0.3ms are feasible. On the other hand, code optimization techniques could be a viable solution to reduce the number of instructions defined within each TCC in order to support more TCCs. In case such techniques are not applicable, the framework provides an alternative to TCCs in the form of LCCs.

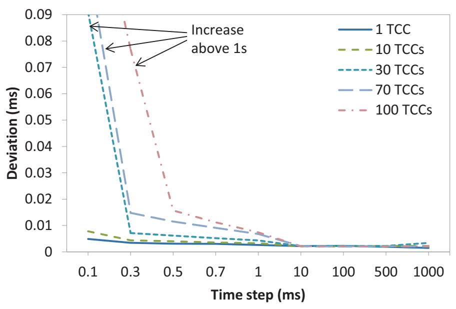
6.3. Miss Rate

Although TCCs will not miss any values generated by the model, as TCCs run sequentially with the model, this is not the case for LCCs. The causes for this are the following: (i) execution time is not synchronized between units; (ii) nodes are running multitasking OSs; and (iii) there is a networking infrastructure that introduces additional delays. The experimental setup for measuring the average miss rate included up to 100 LCCs and 9 time steps ranging from 0.1ms to 1s. Each LCC reads the remote memory, runs 1000 *if-instructions* and finally writes the results back to the remote memory. Figure 8 shows the measured percentages of missed read/write operations. It can be seen that for time steps smaller than 10ms there is a miss rate greater than 50% due to network delays and multitasking OSs. However, the miss rate decreases



	Missed read/write percentage (%)			
Step (ms)	1 LCC	30 LCCs	70 LCCs	100 LCCs
0.1	97	99.41	99.9	99.95
0.5	84.15	92.79	98.83	99.46
1	67.65	84.72	97.6	98.65
100	0	0	0	0
1000	0	0	0	0

Figure 8: Missed read/write percentage



	Deviation without LCCs (ms)			
Step	1 TCC	30 TCCs	70 TCCs	100 TCCs
0.1	0.0049	>1000	>1000	>1000
0.5	0.0031	0.0061	0.0115	0.0157
1	0.0026	0.0043	0.0067	0.00741
100	0.0022	0.0022	0.002	0.0021
1000	0.0015	0.0024	0.002	0.0022

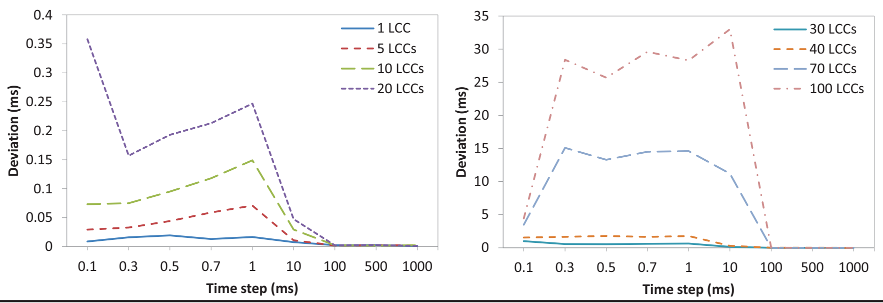
Figure 9: Deviation from the OS clock without LCCs

significantly for time steps above 10ms and reaches zero for 100ms. The reason is that time steps larger than 100ms are enough to cover network delays and task switchings on multitasking OSs.

As shown by these results, the use of LCCs implies studying the physical model in a time granularity above 10ms. However, LCCs provide a dynamic approach for running control logic code by enabling the runtime injection of new (malicious) code without affecting the execution of the simulated physical layer. Furthermore, LCCs are foreseen as an alternative to TCCs that enable the execution of complex control logic code and the integration of other PLC emulators, without affecting the value of the resolution.

6.4. Deviation from the OS Clock

The developed prototype keeps the execution time of the model synchronized with the OS clock. As the underlying OS uses multitasking to support the execution of multiple tasks, the deviation of the execution time from the OS clock can not be avoided. In this section the deviation is measured for up to 100 LCCs and TCCs. First, a scenario with zero LCCs is implemented in order to illustrate the effect of TCCs on the deviation. Second, a scenario with up to 100 LCCs is implemented, thus illustrating the effect of LCCs on the deviation. Figure 9 shows the measured average deviation for the scenario with zero LCCs and up to 100 TCCs. In the same figure we pointed out with arrowed lines that for a 0.1ms time step and starting with 30 TCCs the deviation increases gradually with the execution and exceeds 1s shortly after the experiment is started. The reason for this behavior is that the cumulated code execution time exceeds the 0.1ms time step.



	Deviation without TCCs (ms)							
Step	1 LCC	5 LCCs	10 LCCs	20 LCCs	30 LCCs	40 LCCs	70 LCCs	100 LCCs
0.1	0.0089	0.0294	0.073	0.0358	0.99	1.54	3.48	4.42
0.5	0.0194	0.0442	0.095	0.193	0.543	1.79	13.3	25.7
1	0.0166	0.0708	0.149	0.247	0.636	1.78	14.6	28.3
100	0.0024	0.0019	0.0023	0.0022	0.002	0.0029	0.0017	0.003
1000	0.001	0.0025	0.0023	0.001	0.0019	0.0015	0.0023	0.002

Figure 10: Deviation from the OS clock without TCCs

Consequently, in such cases the framework can not be used for real-time experimentation and researchers must take appropriate measures to reduce the overall execution time of TCCs, e.g. increase the time step, optimize the TCC code, or reduce the number of TCCs. On the other hand, by increasing the time step above 0.1ms the deviation decreases down to 0.0022ms for 100 TCCs. The explanation of this behavior lies in the multitasking OS that is able to provide much better timing precision for time values above 1ms.

If the system also includes LCCs, the value of the deviation changes dramatically as the SC unit is not able to read/write the PLC memory while this is accessed by LCCs (Figure 10). For time steps up to 1ms the deviation increases gradually due to the fact that larger time steps lead to a lower miss rate (Figure 8) and implicitly to a greater number of accesses to the PLC memory. For time steps larger than 1ms the number of simultaneous accesses to the PLC memory decreases, thus the deviation is also decreased.

Based on the presented results we can clearly state that researchers can use several parameters in order to adapt the deviation to their needs. These parameters include the following: time step, TCC code, the number of TCCs, and the number of LCCs. With these parameters the operation of the framework can be tuned in such a manner to enable real-time experimentation with low deviations at the same time.

7. Case Studies

The results from the previous section showed that the proposed framework enables experimentation with complex physical processes, supporting at the same time a large number of PLCs. In this section we continue the evaluation of the framework, with the presentation of two case studies performed in the EPIC laboratory. We show that the proposed framework can be applied to experiment with adversaries employing diverse capabilities and it can easily recreate network architectures through Emulab NS scripts. In the first case study we show the effect of the Stuxnet malware on a Boiling Water Power Plant, while in the second case study we show the effect of network parameters on cyber attacks targeting a chemical process.

7.1. Case Study 1: The Effect of Stuxnet on a Boiling Water Power Plant

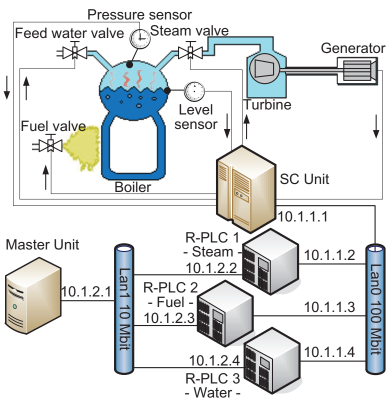
The Stuxnet [4] malware confirmed that industrial systems are vulnerable to cyber threats. According to the Symantec dossier [4], Stuxnet was most likely designed to disrupt the operation of centrifuges used in the process of uranium enrichment. The employed method was a repeated acceleration and deceleration of centrifuges over short periods of time that would eventually lead to their mechanical failure. In this subsection we present an attack scenario inspired from the behavior of the Stuxnet malware on a Boiling Water Power Plant model, that was tested in the EPIC laboratory.

7.1.1. Experiment Description

For this experiment we employed the model of Bell and Åström as it includes estimated parameters from a real power plant. The model is actually that of a 160MW oil-fired electric power plant based on the Sydsvenska Kraft AB plant in Malmö, Sweden and includes 3 units: a boiler unit, a turbine unit and a generator unit. In the experimental setup shown in Figure 11 we use 3 PLCs, one PLC for each control valve. Regular and malicious code is implemented as LCC. The process runs at a typical operating point [23] with the value of the fuel valve set to 0.34, the value of the steam valve set to 0.69 and the value of the feed water valve set to 0.433. The Emulab NS script from Figure 11 shows that nodes and network connections are configured easily with typical NS commands. For instance, the *make-lan* command creates a new network for which we can configure parameters such as capacity and delays, without having to configure additional software. Nodes are also set up from the NS script through commands such as *tb-set-node-os* in order to select the OS and *tb-set-node-startcmd* in order to launch the initialization script of additional software, e.g. SC unit. In the implemented attack scenario only the PLC controlling the steam valve, i.e. R-PLC1, is infected. The infection process involves stopping the regular R-PLC1 unit and starting another unit, thus emulating Stuxnet's ability to rewrite PLC code. The malicious code repeatedly opens and closes the steam control valve, with a frequency of 40s, thus giving enough time to the valve to completely open.

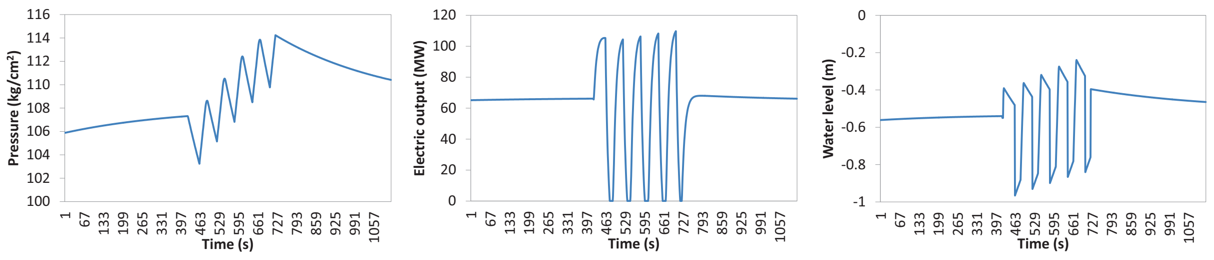
7.1.2. Experimental Results

The attack is conducted over a period of 5 minutes with the effects shown in Figure 12. As soon as the attack is started the pressure increases from 107 kg/cm^2 to 114 kg/cm^2, while electrical output varies from 0 to 108MW. Such variations clearly deviate from the normal electrical output and may disrupt other electrical components, e.g. transformers. The cycling of the steam valve affects not only the pressure and electrical output but also the water level. The water is affected since the evaporation rate is a function of the vapor pressure. According to [24], a change of more than 0.25m in the level of water in the boiler leads



```
set ns [new Simulator]
source tb_compat.tcl
set path2exp ``/users/myuser/StuxnetExp''
set startcmd ``sudo $path2exp/config.sh ''
foreach x {SC RPLC1 RPLC2 RPLC3 Master} {
 # Define node
 set $x [$ns node]
 # Set operating system
 tb-set-node-os $x FreeBSD8
 # Run configuration script on each node
 tb-set-node-startcmd $x $startcmd$x
}
# Create networks
set netw0 ``$SC $RPLC1 $RPLC2 $RPLC3''
set netw1 ``Master $RPLC1 $RPLC2 $RPLC3''
set Lan0 [$ns make-lan $netw0 100000.0kb 0.0ms]
set Lan1 [$ns make-lan $netw1 10000.0kb 0.0ms]
```

Figure 11: Experimental setup and Emulab NS script for creating the experiment in Case Study 1



(a) (b) (c)

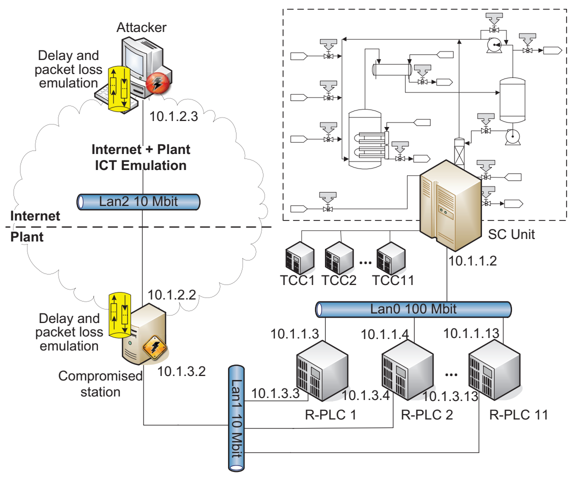
Figure 12: Monitored parameters in Case Study 1: (a) Pressure, (b) Electrical output, and (c) Water level

to alarms and eventually to process failure; consequences that can be expected even in this scenario since the change is 0.3m. These results show that a single infected PLC running malicious code for a short period of 5 minutes can severely affect the normal behavior of the physical process. By further assuming that the attack involves the infection of all three PLCs, the results show even greater variations. In these cases the pressure increases to 160 kg/cm^2 and the water level rises to 0.4m, that is a change of 0.9m. These values clearly suggest the risk of plant failure.

This study showed that with the proposed framework scientists can recreate a scenario with complex malware such as the recently reported Stuxnet worm. Such studies could be further extended with a more thorough analysis on the effects of cyber attacks to physical processes and eventually to the validation of countermeasures against similar attacks.

7.2. Case Study 2: The Effect of Network Parameters on a Cyber Attack Targeting a Chemical Process

The experiment from this subsection includes a powerful adversary and the Tennessee-Eastman [17] chemical process. The main goal of the adversary is to cause the process parameters to reach their shut



```
set ns [new Simulator]
source tb_compat.tcl
set path2exp ``/users/myuser/OutsiderExp''
set startcmd ``sudo $path2exp/config.sh ''
foreach x {SC RPLC1 ... RPLC11 Compr Attack} {
 # Define node
 set $x [$ns node]
 # Set operating system
 tb-set-node-os $x FreeBSD8
 # Run configuration script on each node
 tb-set-node-startcmd $x $startcmd$x
}
# Create networks
set netw0 ``$SC $RPLC1 ... $RPLC11''
set netw1 ``$Compr $RPLC1 ... $RPLC11''
set netw2 ``$Compr $Attack''
set Lan0 [$ns make-lan $netw0 100000.0kb 0.0ms]
set Lan1 [$ns make-lan $netw1 10000.0kb 0.0ms]
set Lan2 [$ns make-lan $netw2 10000.0kb 0.0ms]
```

Figure 13: Experimental setup and Emulab NS script for creating the experiment in Case Study 2

down limits (SDLs). As pointed out by Cárdenas, *et al.* [25] attacks targeting the minimum/maximum value of parameters/control variables are the ones that can damage the process in relatively short time periods. Such attacks cause the accumulation of products, e.g. steam, by completely opening valves that feed products into process units, i.e. feed-valves, and by completely closing valves that free products from the process units, i.e. free-valves. In this scenario the adversary follows the same procedure to force the industrial process to shut down. The main goal of this case study is to show the effect of network parameters, e.g. communications delays and packet losses, on cyber attacks targeting industrial processes. Additionally, this study shows that more complex network architectures and physical processes can be set up easily with an Emulab NS script similar to the one described in the first case study.

7.2.1. Experiment Description

The complexity of the Tennessee-Eastman (TE) chemical process makes it suitable for a wide range of topics, including cyber-physical security-related studies [25, 10]. The TE chemical plant is a process with 41 measured parameters and 12 manipulated variables. The architecture of the TE process includes 5 main units: a two-phase *Reactor*, a product *Condenser*, a recycle *Compressor*, a vapor/liquid *Separator* and a product *Stripper*. More details on the TE process can be found in the original paper by Downs and Vogel [17]. In order to keep the process within its normal operating limits we used the multi-loop control system developed by Sozio [26], implemented in 11 PLCs (as TCC code).

The adversary model we employ in this experiment tries to cause a shut down of the physical process by sending legitimate Modbus packets to PLCs. Identifying the attack vector that could compromise the system to enable such a scenario is not the main focus of this study. However, the Stuxnet worm [4] together with other studies [8] showed that such scenarios are possible in real settings. Network delays and packet losses were emulated with the *Dummynet* software, running on the adversary's and on the compromised stations within the corporate network. The experimental setting for this attack scenario and the associated Emulab NS script are depicted in Figure 13. Because of space considerations the provided NS script includes "..." to denote consequent *RPLC* identifiers starting from *RPLC2* and ending with *RPLC10*. The studied communications delays range from 0s to 3s and the studied packet loss rates range from 0% to 10%. For each parameter we ran one experiment and we recorded the process *shut down time* (SDT), i.e. the time that the process is able to run after the attack is started, before shutting down. This value was also used as a metric of the impact of network parameters on cyber attacks.

7.2.2. Experimental Results

The operation of the TE process for 40h without any disturbances is shown in Figure 14 (a). With the implemented control loops the process is able to run in a steady-state, as shown by the figure depicting the behavior of the selected parameter. Without these control loops, the process parameters would reach their shut down limits (SDLs) after approximately 3.6h [26]. Next, we measured the effect of network delays on the process SDT. For the full network setting, the effect of delays is insignificant, as up to 0.5s the SDT does not show any changes. On the other hand, for network delays of 1s the value of the SDT increases to 4.83min, while for delays of 3s the SDT increases to 5.51min. However, such extreme delays can rarely be measured over the Internet, and even in such cases the impact on the attack is minimal. This behavior is also depicted in Figure 14 (b). In the following step we analyzed the effect of packet losses on the process SDT. Packet loss rates of 5% increased the SDT of the full network setting to 5.16min. However, extreme packet losses of 10% had an insignificant effect on the SDT and increased it to only 5.58min in the same setting. We can also see this effect in Figure 14 (c).

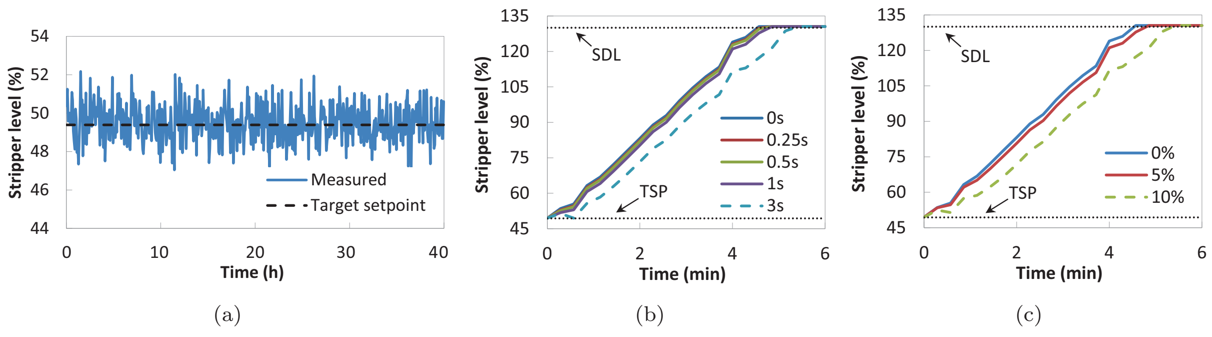


Figure 14: Normal operation (a) and effect off communications delays (b) and packet losses (c) on the Tennessee-Eastman process (SDL – Shut Down Limit, TSP – Target SetPoint)

The main goal of this study was not to be exhaustive in the choice of network parameters, but to prove that communications delays and packet losses have a minor effect on cyber attacks in which the adversary communicates with PLCs. This scenario also confirmed the fact that the proposed framework is suitable for security studies involving complex cyber-physical systems.

8. Conclusions

Recent security incidents [4] indicate the need for testbeds for conducting security experiments with NICS. Current approaches suffer either from high operational costs (ad-hoc testbeds) [8, 12] or lack realism (pure simulators) [10, 11]. This paper overcomes these constraints with a novel framework blending together an Emulab-based testbed which recreates the industrial ICT components and real-time software simulators which recreate the physical systems. The experimental results showed that our framework prototype can efficiently recreate large installations (100 PLCs) and can simulate complex physical systems, e.g. the TE process, in near real-time, with low deviation (< 1ms) and low miss rate (0%), for simulation steps larger than 10ms. The proposed framework was analyzed from several perspectives: (i) a qualitative comparison showed that it provides a set of key functionalities that are missing in related approaches, e.g. supports real software/malware, enables automated and safe experimentation with NICS; (ii) a comparison of capital and operational costs showed that the approach is highly cost efficient; and (iii) experimental results confirmed that the framework can recreate complex scenarios similar to the ones reported in related approaches [10]. Finally, the paper proved through two case studies, one involving attacks in the power sector and another in the chemical sector, that it clearly advances the state of the art since it allows to recreate a wide range of security scenarios in a safe and automated manner. As future work we intend to use the proposed framework to study the newly proposed architectures and protocols for the Smart Grid [27].

References

[1] East S, Butts J, Papa M, Shenoi S. A taxonomy of attacks on the DNP3 protocol. IFIP Advances in Information and Communication Technology 2009;311:67–81.

[2] Cui H, Wang Y. Four-mobile-beacon assisted localization in three-dimensional wireless sensor networks. Computers & Electrical Engineering 2012;38(3):652–61.

[3] Guo H, Xu C, Mu Y, Li Z. A provably secure authenticated key agreement protocol for wireless communications. Computers & Electrical Engineering 2012;38(3):563 –72.

[4] Falliere N, Murchu LO, Chien E. W32.Stuxnet Dossier. `http://www.wired.com/images_blogs/threatlevel/2010/11/w32_stuxnet_dossier.pdf`; 2010. [Online; accessed November 2011].

[5] White B, Lepreau J, Stoller L, Ricci R, Guruprasad S, Newbold M, et al. An integrated experimental environment for distributed systems and networks. In: Proc. of the 5th Symposium on Operating Systems Design and Implementation. 2002, p. 255–70.

[6] Siaterlis C, Perez-Garcia A, Genge B. On the use of Emulab testbeds for scientifically rigorous experiments. IEEE Communications Surveys and Tutorials 2012 (Accepted);.

[7] Chunlei W, Lan F, Yiqi D. A simulation environment for SCADA security analysis and assessment. In: Proc. of the 2010 International Conference on Measuring Technology and Mechatronics Automation. 2010, p. 342–7.

[8] Nai Fovino I, Masera M, Guidi L, Carpi G. An experimental platform for assessing SCADA vulnerabilities and countermeasures in power plants. In: Human System Interactions (HSI), 2010 3rd Conference on. 2010, p. 679–86.

[9] Queiroz C, Mahmood A, Hu J, Tari Z, Yu X. Building a SCADA security testbed. In: Proc. of the 2009 Third International Conference on Network and System Security. 2009, p. 357–64.

[10] Chabukswar R, Sinopoli B, Karsai B, Giani A, Neema H, Davis A. Simulation of network attacks on SCADA systems. In: 1st Workshop on Secure Control Systems, Cyber Physical Systems Week. 2010,.

[11] Hopkinson K, Wang X, Giovanini R, Thorp J, Birman K, Coury D. Epochs: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components. Power Systems, IEEE Transactions on 2006;21(2):548 –58.

[12] Davis C, Tate J, Okhravi H, Grier C, Overbye T, Nicol D. SCADA cyber security testbed development. In: Power Symposium, 2006. NAPS 2006. 38th North American. 2006, p. 483–8.

[13] Hiyama T, Ueno A. Development of real time power system simulator in Matlab/Simulink environment. In: Power Engineering Society Summer Meeting, 2000. IEEE; vol. 4. 2000, p. 2096–2100 vol. 4.

[14] McDonald M, Conrad G, Service T, Cassidy R. Cyber effects analysis using VCSE. Technical Report, SAND2008-5954, Sandia National Laboratories 2008;.

[15] Chertov R, Fahmy S, Shroff NB. Fidelity of network simulation and emulation: A case study of tcp-targeted denial of service attacks. ACM Trans Model Comput Simul 2009;19(1):4:1–4:29.

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

[17] Downs J, Vogel E. A plant-wide industrial process control problem. Computers & Chemical Engineering 1993;17(3):245–55.

[18] Neema S, Bapty T, Koutsoukos X, Neema H, Sztipanovits J, Karsai G. Model based integration and experimentation of information fusion and C2 systems. In: Proc. of the 12th International Conference on Information Fusion. 2009, p. 1958–65.

[19] PowerWorld . PowerWorld server. `http://www.powerworld.com`; 2010. [Online; accessed November 2011].

[20] Mirkovic J, Benzel T, Faber T, Braden R, Wroclawski J, Schwab S. The DETER project: Advancing the science of cyber security experimentation and test. In: Proc. of the IEEE International Conference on Technologies for Homeland Security (HST). 2010, p. 1–7.

[21] Siaterlis C, Masera M. A survey of software tools for the creation of networked testbeds. International Journal On Advances in Security 2010;3(2):1–12.

[22] Genge B, Siaterlis C. Cyber-physical test beds: A cost analysis of capital and operating expenditure. Technical Report, JRC71479, Joint Research Centre 2012;.

[23] Tan W, Fang F, Tian L, Fu C, Liu J. Linear control of a boiler-turbine unit: Analysis and design. ISA Transactions 2008;47(2):189–97.

[24] Ben-Abdennour A, Lee K. An autonomous control system for boiler–turbine units. Energy Conversion, IEEE Transactions on 1996;11(2):401–6.

[25] Cárdenas A, Amin S, Lin Z, Huang Y, Huang CY, Sastry S. Attacks against process control systems: Risk assessment, detection, and response. In: Proc. of the 6th ACM Symposium on Information, Computer and Communications Security. 2011, p. 355–66.

[26] Sozio J. Intelligent parameter adaptation for chemical processes. Master's thesis; Virginia Polytechnic Institute and State University; USA; 1999.

[27] Korres G, Manousakis N. A state estimator including conventional and synchronized phasor measurements. Computers & Electrical Engineering 2012;38(2):294 – 305.

Béla Genge is a Post-Doctoral Researcher at the Institute for the Protection and Security of the Citizen, Joint Research Centre of the European Commission, Ispra, Italy. His research interests include critical infrastructure protection, design methods and composition of security protocols.

Christos Siaterlis is a Scientific Officer at the Institute for the Protection and Security of the Citizen, Joint Research Centre of the European Commission, Ispra, Italy. His research interests include the security, stability and resilience of computer networks.

Igor Nai Fovino is the Head of the Research Division of the Global Cyber Security Center, Rome, Italy. His research interests include critical infrastructure protection, intrusion detection, secure communication protocols and industrial informatics.

Marcelo Masera is the Head of the Energy Security Unit at the Institute for Energy, Joint Research Centre, Petten, The Netherlands. His research interests include securing networked systems and systems of systems, risk governance and control systems security.