MiniCPS: A Toolkit for Security Research on CPS Networks

Daniele Antonioli
daniele_antonioli@sutd.edu.sg

Nils Ole Tippenhauer
nils_tippenhauer@sutd.edu.sg

Singapore University of Technology and Design (SUTD)
8 Somapah Road
487372 Singapore

ABSTRACT

In recent years, tremendous effort has been spent to modernizing communication infrastructure in Cyber-Physical Systems (CPS) such as Industrial Control Systems (ICS) and related Supervisory Control and Data Acquisition (SCADA) systems. While a great amount of research has been conducted on network security of office and home networks, recently the security of CPS and related systems has gained increased attention. Unfortunately, real-world CPS are often not open to security researchers, and as a result very few reference physical-layer processes, control systems and communication topologies are available.

In this work, we present *MiniCPS*, a toolkit intended to alleviate this problem. The goal of MiniCPS is to create an extensible, reproducible research environment for network communications, control systems, and physical-layer interactions in CPS. Instead of focusing on a customized simulation settings for specific subsystems, the main goal is to establish a framework to connect together real CPS soft- and hardware, simulation scripts for such components, and physical-layer simulation engines. MiniCPS builds on Mininet to provide lightweight real-time network emulation, and extends Mininet with tools to simulate typical CPS components such as programmable logic controllers, which use industrial protocols (e. g. EtherNet/IP, Modbus/TCP). To capture physical-layer interactions, MiniCPS defines a simple API to connect to physical-layer simulations. We demonstrate applications of MiniCPS in two example scenarios, and show how MiniCPS can be used to develop attacks and defenses that are directly applicable to real systems.

Keywords

CPS; ICS; SDN; Mininet; Attack Simulation; EtherNet/IP

1. INTRODUCTION

Industrial Control Systems (ICS) and Supervisory Control and Data Acquisition (SCADA) systems traditionally rely on communication technology such as RS-232 and RS-485, and field buses such as PROFIBUS [10]. Due to the long lifetime of industrial components in such settings, transitions to technology such as Ethernet, TCP/IP, and related protocols are often only implemented now. The adoption to the standard Internet protocol suite is expected to enhance interoperability of the equipment, and reduce overall communication costs.

The growing connectivity is also expected to introduce novel security threats, in particular when systems are communicating over public networks such as the Internet. While a great amount of research has been conducted on network security of office and home networks, recently the security of CPS and related systems has gained a lot of attention [4, 16, 26, 30, 31]. In CPS, physical-layer interactions between components have to be considered as potential attack vectors, in addition to the conventional network-based attacks. Examples for real-world CPS are unfortunately often not open to security researchers, and as a result few reference systems (i. e., physical process description, control systems, network topologies) are available. In addition, physical layer interactions between components need to be considered besides network communications. We believe that this will require novel simulation environments, that are specifically adapted to cater for the requirements of CPS and ICS. In particular, such environments could be used to co-simulate (in real-time) using different simulated and emulated components together with real hardware or processes.

In this work, we present *MiniCPS*, a toolkit intended to alleviate this problem. The goal of MiniCPS is to create an extensible, reproducible research environment targeted towards CPS. MiniCPS will allow researchers to emulate the Ethernet-based network of an industrial control system, together with simulations of components such as PLCs. In addition, MiniCPS supports a basic API to capture physical layer interactions between components. Based on MiniCPS, it is possible to replicate ICS in real-time, for example to develop novel intrusion prevention systems, or own software to interact with industrial protocols. While not all CPS systems are using Ethernet-based communication so far, we see a general trend towards wide adoption of Ethernet as physical layer for the communication [10].

MiniCPS can also be used to share different system setup, and can be extended by standard Linux tools or projects. Due to our use of Mininet for the network emulation part, MiniCPS is also well suited to perform research on Software-Defined Networking in the context of Industrial Control Systems.

CPS-SPC'15, October 16, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3827-1/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2808705.2808715 .

We summarize our contributions as following:

- We identify the issue of missing generic simulation environments for applications such as cyber-physical systems. In particular, such simulation environments should support physical interactions, parametric communication links, and specific industrial (ideally all) protocols that are used.
- We present MiniCPS, a framework built on top of Mininet, to provide such a simulation environment.
- We present an example application cases in which we use MiniCPS to develop and refine a specific attack, which we later validated in a real testbed.
- We propose the use of Software-Defined Networking for CPS networks to enable efficient detection and prevention of the attack presented earlier. We design and implement a matching SDN controller, and validate it in MiniCPS.

The structure of this work is as follows: In Section 2, we introduce Mininet and CPS networks in general. We propose our MiniCPS framework in Section 3, and provide an application example in Section 4. In Section 5, we show how MiniCPS can be used to develop a CPS network specific SDN controller. Related work is summarized in Section 6. We conclude the paper in Section 7.

2. CPS NETWORKS AND MININET

In this section, we will introduce some of the salient properties of industrial control system (ICS) networks that we have found so far. In addition, we will briefly introduce Mininet, the network simulation tool we use as part of MiniCPS.

2.1 ICS networks

In the context of this work, we consider ICS that are used to supervise and control system like public infrastructure (water, power), manufacturing lines, or public transportation systems. In particular, we assume the system consists of programmable logic controllers, sensors, actuators, and supervisory components such as human-machine interfaces and servers. We focus on single-site systems with local connections, long distance connections would in addition require components such as remote terminal units (see below). All these components are connected through a common network topology.

Programmable logic controllers. PLCs are directly controlling parts of the system by aggregating sensor readings, and following their control logic to produce commands for connected actuators.

Sensors and actuators. Those components interact with the physical layer, and are directly connected to the Ethernet network (or indirectly via remote input/output units (IOs) or PLCs).

Network Devices. ICS often use *gateway* devices to translate between different industrial protocols (e. g. Modbus/TCP and Modbus/RTU) or communication media. In the case where these gateways connect to a WAN, they are usually called *remote terminal units* (RTUs).

Network Topology. Traditionally, industrial control systems have seen a wide deployment of direct links between components, based on communication standards like RS-232. In addition, bus systems such as RS-485 and PROFIBUS have been used. In particular, focus on reliability led to a deployment of topologies such as rings [5], which could tolerate failure of a single component without loss of communications, with very low reaction time (typically in the order of milliseconds).

Industrial protocols. In recent years, industrial networks are transitioning to mainstream consumer networking technology i. e. Ethernet (IEEE 802.3), IP, TCP. Nevertheless, the need for reliability and interoperability with existing equipment leads to systems that are different from typical home and office networks, such as Ethernet rings, use of IP-layer multi-casting, and custom protocols such as the EtherNet/IP (ENIP) [20]. ENIP is a Real Time Ethernet (RTE) field bus based on TCP/IP with a custom application layer designed to meet typical industrial communications requirements like real-time response and packet determinism. Technically ENIP is an Ethernet based implementation of the *Common Industrial Protocol (CIP)* and it is defined in the IEC 61784-1 standard [10]. CIP messages can be used to query sensor readings from components, set configuration options, or even download new logic on a PLC. In that model, sensor readings or control values are represented by *tags*. CIP uses a request-response model where a client sends a request to a server (for example to read a tag containing a value read from a hardware component) and where the server then sends back a reply (e. g. with the requested value or an error code). Such requests can operate on tags and also on the meta-data associated with the tag, like access control and data type, which are stored in *attributes*. ENIP handles the selected aspects of the communication, for example with connected sessions (with handshake and tear-down messages) and unconnected sessions (without any handshake but with more contextual data in every CIP packet).

Topology layers. Networks for industrial control systems are often grouped in several layers or zones (more detail on such networks in [18, 22]). On the lowest layer (which we call layer 0 or L0), sensors and actuators are connected to controllers such as PLC. The sensors and actuators are either capable of connecting to a network directly (e. g. using ENIP), or they use basic analog or digital signaling, which has to be converted to Ethernet-based communications by *remote input/output* (RIO) devices. Only if actuators and sensors are physically very close to the PLC, the IO modules will be installed as part of the PLC.

The next higher layer (layer 1/ L1) will connect the different controllers (PLCs) with each other, together with local control such as Human-Machine-Interfaces (HMI), local engineering workstations, and Data historians. For simplicity, all these devices are often kept in the same IP-layer subnetwork, although more complex topologies are possible. We also note that industrial Ethernet switches are often focused on electrical reliability, instead of IP-layer functionality (e.g. the Rockwell Automation Stratix 5900 switch). We provide the network topology of a generic ICS network as an example in Figure 1.

2.2 Mininet

Mininet [14] is a network emulator that allows to emulate a collection of end-hosts, switches, routers, middle boxes, and links with high level of fidelity. It enables rapid testing and prototyping of large network setups on constrained resources, such as a laptop. Furthermore, it was build around the Software-Defined Networking paradigm, facilitating SDN research and development [6].

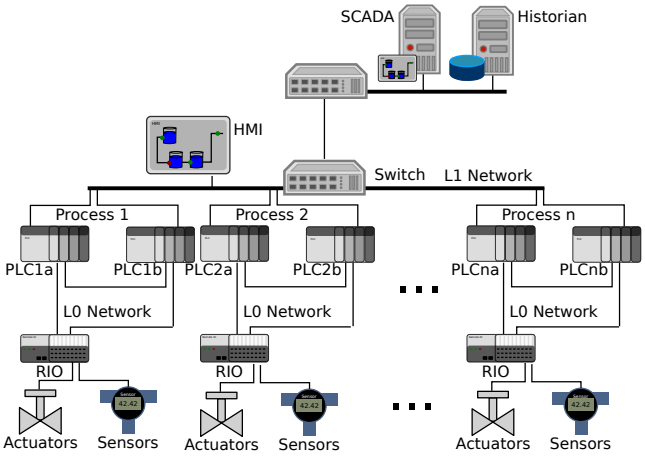

Figure 1: Example local network topology of a plant control network.

Mininet exploits lightweight system virtualization using Linux *containers*. This approach presents various advantages over a full system virtualization: Mininet runs on a single kernel, its computational overhead is lower and the emulator can easily tolerate scalability issues (e. g. one thousand containers vs. one thousand dedicated virtual machines).

Each virtual host is a collection of processes isolated into a container. A *virtual network namespace* is attached to each container and it provides a dedicated virtual interface and private network data. Link are emulated using virtual Ethernet (`veth`) and they can be shaped through Linux Traffic Control (`tc`) to emulate link performance such as delay, loss rate and bandwidth. Each virtual host utilizes its virtual interface to communicate with other network devices.

Mininet can be used in multiple scenarios and can be easily adapted over time to track the evolution of CPS networks. It provides a realistic emulation environment to the user, and one can work with the same addresses, protocol stacks and network tools of a physical network, it is even possible to reuse helper scripts and configuration files from the emulated environment directly in the physical network.

Mininet ships with a set of common topologies, in addition the user can easily extend this collection through the provided public Python APIs. Dynamic interaction within any chosen topology can be achieved through a convenient command line interface. Mininet is free, open-source, well documented and actively maintained by a strong and competent community. Furthermore, Mininet gives the user the opportunity to develop OpenFlow network architectures with direct integration of experimental code into production code.

3. MINICPS

In this section, we present our proposed toolkit *MiniCPS*. MiniCPS combines a set of tools for real-time emulation of network traffic with CPS component simulation scripts, and a simple API to interface with real-time physical-layer simulations. The combined system will allow a) researchers to build, investigate, and exchange ICS networks, b) network engineers to experiment with planned topologies and setups, and c) security experts to test exploits and countermeasures in realistic virtualized environments.

In MiniCPS, components such as PLCs are represented by Python scripts that manage the decoding of industrial protocols and physical layer sensor and actuator signals. All network components (e. g. switches) are emulated using Mininet (as introduced in Section 2). Physical layer interactions are currently connected to suitable simulation software through a simple API (based on shared read/write to files). MiniCPS itself does not focus on providing physical process simulations – it rather connects components to their counterparts, and process simulation engines.

3.1 Goals of MiniCPS

In the following, we provide a discussion of our vision behind MiniCPS, in particular related to its focus on network emulation, overall simulation, and real-time properties. We also discuss co-simulation with real devices, integration of physical-layer simulation tools, and integration of software defined networking.

Network Emulation. MiniCPS focuses on high-fidelity network emulation, which allows simulated components to exchange very similar or the same traffic as seen in the real network, from Ethernet layer up to the application layer (based on Mininet, see Section 2.2). In particular, we aim to not only provide an abstraction of the network to perform simulations on (similar to network simulators such as NS2 [12], OMNet++ [25]), but we target a network emulation that is largely identical to a real network, without the cost or overhead of running a real network or a set of virtual machines. In particular, this would allow us to develop components that are directly using industrial protocols to communicate.

Simulation vs Emulation. We recognize that the difference between emulation and simulation might not be universally defined. In the context of this work, we use the terms *simulation* and *emulation* with the following intentions: components that are emulated will behave (in real-time) exactly at their real-world counterparts. The network emulation in Mininet is an example – there is essentially no difference for the OS between a virtual network adapter in Mininet, and a real physical network adapter. We note that the timing of the emulated components will be similar to the real counterparts, but might not match exactly. The emulated components will run in real-time, and cannot be sped up or slowed down in their operations. Emulated components should be able to interact directly with other emulated components, if their real-world counterparts are able to interact.

In contrast, simulated components operate on a reduced model to analyze selected properties of the target system. The simulation might run in real-time, or faster/slower. The simulated component will typically interact with other components through channels that do not replicate real-world interactions. In this work, we assume that physical processes can only be simulated. We also classify components as "simulated" if they are only approximating their real-world counterparts in behaviour or interactions. We note that in the context of MiniCPS, simulations will be restricted to real-time to allow interoperability with emulated components.

For example, if (in the future) software is available in MiniCPS that processes real-world PLC code (e.g. in ladder logic format), and replicates the behaviour of the PLC accurately, then we would call this a PLC emulation. For now, our tools that represent PLCs in MiniCPS are merely

implementing a subset of PLC functionality, as manually extracted by us from the analysis of the real-world PLC code.

Time. While the traffic data will be mostly identical, the temporal properties of the traffic depend on the involved simulation scripts. In general, Mininet can emulate delays, but does not provide deterministic guarantees for timing. We recognize that this could lead to problems when simulating closed loop control over the network with very tight timing, but we do not face those problems in our current application (a relatively slow water testbed). We estimate that precision in the order of milliseconds could be possible.

Co-Simulation. Based on the high-fidelity network emulation, it is also possible to connect simulated and real networks together, to enable cheap and easy evaluations of real devices in emulated network environments. This integration is possible because the network traffic is not simulated on some abstraction layer, but identical traffic running over an emulated physical network layer.

Physical-Layer Abstraction. The physical-layer integration into MiniCPS is mainly relying on our API to represent the limited number of *interaction points* between the physical process and the sensor/actuator (cyber) components. The API is intended to enable different simulated components to interact on the physical layer directly, without requiring complex specialized interfaces.

Reproducibility. In [11], the authors proposed to use tools such as Mininet to disseminate reproducible research results. In particular, researchers can publish the scripts to generate their network setups, which allows other researchers to reproduce the exact same environments for their experiments. We strongly believe that such dissemination of results would also be helpful in the context of security research, in particular when systems which are less mainstream are considered. While it is relatively easy to replicate office network settings as related software is well-known, specialized application setups such as ICS would be valuable to share.

Integration of SDN. The detailed network emulation will allow us to use novel concepts such as software defined networking in the context of CPS networks (see Section 5). We note that to achieve this compatibility, we will be constrained to real-time simulation instead of being able to simulate with arbitrary speedup.

What MiniCPS does not aim for. MiniCPS does not aim to be a performance simulator, or tool for optimizations. In particular, we consider that full-fledged physical process simulation is out of the scope of MiniCPS. Instead, MiniCPS focuses on connecting component simulation/emulation scripts together in a virtualized environment, and enable simple connections to third party physical layer simulation tools. In addition, we currently put very little emphasis on GUI or visualization. We note that building on top of the physical layer API, and by extending the component logic scripts in general, it should be possible to easily create real-time charts of physical process parameters or controller states.

3.2 Design overview

Components in MiniCPS interact on several layers (see Figure 2). On the top layer, we have the network through which messages are exchanged on top of ENIP, or other protocols. Connected to this network are components, their

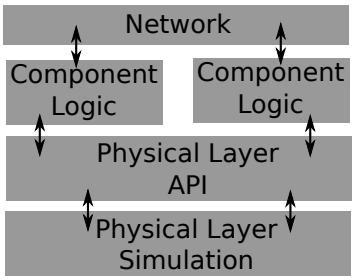

Figure 2: MiniCPS framework layers: CPS components are simulated as component logic, connected through the network emulation, and physical layer simulation.

logic is implemented in simple scripts or more advanced software packages. If the real-world counterpart of these components is interacting with the physical layer, the simulated components can also access specific physical layer properties through a second API, which abstracts the physical layer. To simulate chemical or physical processes, a selection of their properties can be made available through the API (e. g. the tags in the SCADA system as discussed in Section 2), and updated in real-time by simulation scripts.

3.3 Network Communication

For the main network emulation layer of MiniCPS, we are using Mininet (see Section 2). We chose Mininet primarily because it allows lightweight emulation of a large number of hosts and their network connections (Mininet's SDN capabilities are a welcome addition). Mininet allows to directly use IP-based industrial protocols over the virtualized Ethernet connections. In contrast, other network simulation tools usually require an abstract re-implementation of the used protocols (for more discussion, see Section 6). Mininet allows to set basic link properties such as delay, loss rates, and capacity. In MiniCPS, we use this functionality to allow individual links to be configured with individual settings. As a result, we can emulate wide area network connections and local area network connections with different properties.

Based on Mininet, the network communication in MiniCPS uses the default Linux networking stack based on Ethernet. All components have virtualized network interfaces that are connected to each other. In particular, this setup allows us to construct arbitrary topologies such as simple star topologies of switches connected to devices, intermediate routers and firewalls, and topologies such as Ethernet rings. Protocols such as the spanning-tree-protocol or other routing algorithms can be used to automatically avoid looping configurations, and to establish routes. All standard protocols such as ICMP, HTTP, NTP, etc. can be used right away. On top of that, specific industrial protocols can be used. In particular, we use the CPPPO Python library to provide fundamental EtherNet/IP (ENIP) services [13]. In addition to ENIP, CPPPO also supports protocols such as Modbus/TCP. In addition to CPPPO, we also use the pycomm library for ENIP communications [23].

3.4 Physical Layer Interactions

Physical layer interactions between different components in the systems are captured by our PHY-SIM API. This API is essentially a set of resources (currently files), that

provide data in real-time. These resources can be read by components (i. e. a sensor reading some physical property), or written to (typically, by a script that emulates physical processes). The main purpose of the simple API is to allow different tools to interact with it as easily as possible. For example, such tools could be Matlab, Python scripts, or dedicated physics simulators. Representing the physical layer properties as file resources makes this API usable by a wide range of libraries and programming languages. We also envision that it is possible to connect these files to an actual physical process, i. e. to have the process *in the simulation loop* (if suitable interfaces to the physical system are provided). In the long term, the simple API could be extended to a more generic API, for example a RESTful API.

3.5 Implementation

MiniCPS is essentially a set of tools that extends Mininet, in particular by adding scripts to represent components such as PLCs, HMIs, and historians, and by adding the physical layer API and example physical process simulation scripts. Our class hierarchy follows Object Oriented design principles: every reusable, self-contained piece of code is wrapped inside a class (such as a topology, a topology manager or an SDN controller).

Our implementation contains three core modules: constants, topologies, and devices. The *constants* module collects data objects and helper function common to all the codebase. The *topologies* module is a collection of ad-hoc CPS and ICS topologies with realistic addresses and configurable link performance. The *devices* module contains a set of control logic applications (see Section 5). Each core module is mirrored with a testing module counterpart. Our class hierarchy design easily allows Test Driven Development [2] because each topology manager potentially can select a network configuration, a controller, the performance of the virtual links and even the CPU allocation for each virtual host. In other words, a topology manager it is a self-contained topology test. Each test module is a collection of *test_Something* classes with appropriate fixtures. , e.g. to set the Mininet log level at setup.

We used the Python *nosetests* module to automate test design, discovery, execution, profiling and report. The *logging* module enables interactive code debugging/alerting and long time information storage. Each core module and its testing counterpart append information to the same log file, that rotates automatically through five time-sorted backups. SDN controllers log on separated files that are (over)written at runtime. SDN code integration is obtained by means of soft links using an initialization bash script.

We have implemented a first version of MiniCPS, and are currently in the process of testing and extending its functionality. We released the tool to the public under an open source license, with the main project website available at `minicps.net`. All extensions are using Python, and are documented using the Sphinx package.

4. EXAMPLE APPLICATION: MITM TRAFFIC MANIPULATIONS

We mainly use MiniCPS to model the communications and control aspects of a water treatment testbed at our institution (Singapore University of Technology and Design). While the testbed is intended for security research, we find it useful to have the MiniCPS emulation environment to replicate the network settings outside the lab. In addition to simulated interactions with PLCs and sensors, the MiniCPS model also allows us to experiment with different network topologies, and test SDN-related prototypes. In the following, we highlight two such projects based on the MiniCPS model of our testbed. The first application aims to provide on-the-fly manipulation of ENIP/CIP traffic to change commands and sensor values as exchanged between an HMI and a PLC. The second application (in Section 5) concerns SDN controller-based detection and mitigation of ARP spoofing attacks in the testbed.

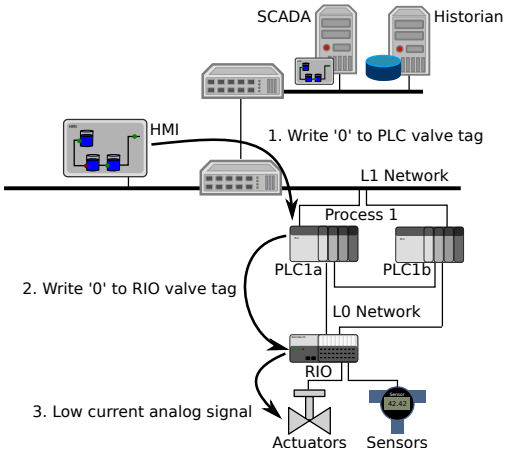

Figure 3: Normal control message flow in the CPS. We omit the acknowledgment reply from the PLC in this visualization.

4.1 System model

Let us assume the following setting (see Figure 3): the HMI is used to manually control the valve of a water feed line towards a water storage tank. The control decision is done on the HMI (e.g. operated by a human), based on the fill-level of the tank as reported by a sensor in the tank. In practice we found that such settings are common.

Based on that scenario, we modeled the network, HMI, PLC, and the physical layer interaction between the valve and the tank in MiniCPS. In particular, we modeled the valve as a Boolean value, and the fill-state of the tank (depending on the current volume of water held, and the inflow and outflow). The valve value is periodically read by a process simulation script. If the valve is open, the current fill-state of the tank is increased by an amount representing the inflow. Both the valve and fill-state are also used by the PLC simulation script, which periodically reads the fill-state and provides it as read-only CIP tag to the emulated network. The simulated PLC also provides a writable CIP tag for the valve control.

4.2 Attack scenario

The attacker has access to the local L1 network, his goal is to arbitrarily change the fill state of the tank. An example attack could aim to fill a water tank it over allowed maximal capacity (and thus damaging the tank or flooding the environment), without being detected.

As first naïve approach, an active attacker could potentially overwrite the valve control tag (as there is no direct

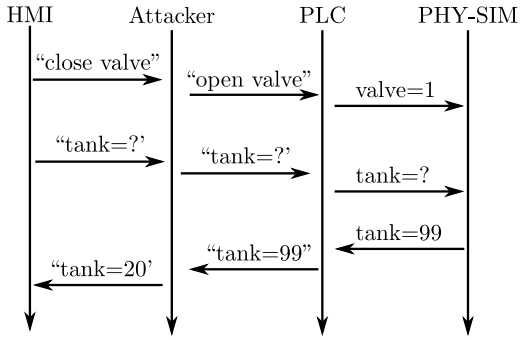

Figure 4: Abstract messages in the extended attack: in addition to the modification of the control messages, the affected measurements from the PLC are also manipulated to hide the attack. In this setting, PHY-SIM could either be a real physical process, or our simulation layer.

access control in ENIP), but the HMI will continuously overwrite the setting to its intended state (in our system, with 10Hz). As a result, to continuously change the valve setting, the attacker has to send a large amount of traffic to compete with the intended control by the HMI, potentially interrupting normal operations.

To mitigate this unintentional countermeasure, we developed a Man-in-the-Middle (MitM)-based attack that does not increase the traffic load on either HMI or PLC, and which does not interferes with other data exchanged between PLC and HMI. The attack is based on ARP spoofing using the ettercap tool [8], and a custom ettercap script to manipulate captured traffic in real-time. We now provide a brief summary of ARP spoofing attacks, and then present the ENIP traffic manipulation attack in more detail.

ARP spoofing is a well-known attack in computer networks [27]. The attacker is connected to the same Link Layer network segment as two victims, that are exchanging messages. The attacker then sends specifically crafted address resolution protocol (ARP) packets to both victims to cause them to send their messages to the attacker, instead of each other. The attacker then forwards the redirected messages to the original recipient, which allows him to perform a stealthy man-in-the-middle attack. We will show a possible countermeasure against this attack in Section 5.

Using ARP-spoofing, an attacker in the Layer 1 network of an ICS system (as described in Section 4.1) can redirect all traffic between two victims, e.g. PLC1 and the HMI.

4.3 MitM Attack

In the MitM attack (see Figure 5), we used the ettercap tool [8] to run an ARP spoofing attack, and as a result install the attacker as MitM between the HMI and the PLC (see Figure 4). We wrote a set of ettercap filter rules to change the value written by the HMI to the valve tag at the PLC. As a result, each time the HMI sent a control message to the PLC to keep the valve closed, the attacker could then change this setting to “open”, without fearing the HMI from overwriting it again. We developed and deployed ettercap scripts to perform this attack in MiniCPS, and were able to successfully change the valve tag to arbitrary values.

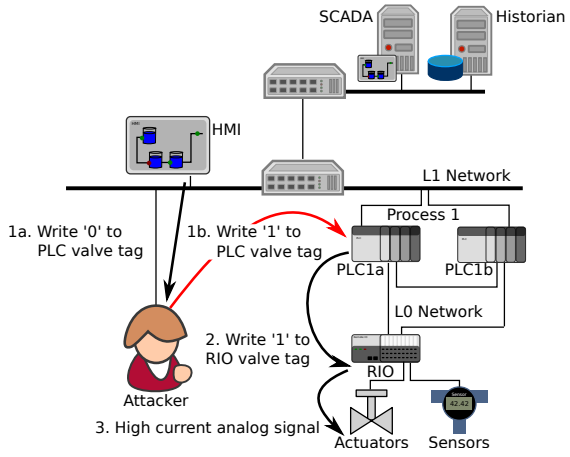

Figure 5: Control message flow during the ARP spoofing attack.

4.4 Including physical layer interactions

In our MiniCPS setup, we then simulated physical layer interactions based on the MiniCPS physical layer API, and several scripts that updated physical layer properties based on a set of simple rules. As result, the valve opened by the attacker led to an increasing fill-state of the tank, which was in turn reported by the PLC when queried by the HMI. We note that this dependency is rather simple, and did not require complex simulations. In practice, this would allow the HMI to at least trigger an alarm condition after the tank is exceeding the maximal fill state.

To prevent such detection, we extended our attack with a second set of filter rules in the attacker. In addition to rewriting the valve control values, the attacker now also rewrote the value of the fill-state tag as reported from the PLC to the HMI. In particular, the attacker could set this value to a constant, or apply some noise to it if wanted. We successfully applied this attack in the MiniCPS environment.

After applying the attack in MiniCPS, we were able to validate the same attack to the real physical testbed, with only minor modifications. The modifications were necessary as the exact CIP messages exchanged between the HMI and PLC in the physical testbed are not yet fully identical to the ones exchanged in our MiniCPS environment. In particular, CIP supports different types of read operations, and our simulated PLCs do not support the variant that is used in the real PLCs (due to limitations in the used CPPPO library). Apart from these modifications, the same attack was successfully run.

5. EXAMPLE APPLICATION: SDN

There are a number of known countermeasures against the ARP spoofing attack from the previous section (e. g. static ARP tables in the hosts, traffic monitoring with an IDS).

As a side-effect of using Mininet in MiniCPS, it is relatively easy to experiment with Software-Defined Network (SDN) technology. Given that, we were interested to see how a customized SDN controller could be used to detect and prevent the attack outlined in the previous section. In particular, SDN promises to allow a controller to manage CPS packet switching rules and to prevent or detect/mitigate ma-

licious hosts packets. SDN-based solutions in the context of smart power grids were also recently proposed in [7]. We compare our work with that work in more detail in Section 6. In a more general context, related work was published recently in [28, 29].

We note that, while in many applications SDN is used to address highly dynamic network conditions, traffic in industrial control systems is usually quite predictable. In particular, topologies and the set of hosts remain static (until the system is updated with new components). In addition, we noticed that components exchange essentially the same traffic, with varying data payload. For example, tag values could be queried every 100ms, and control commands could be sent every second, resulting into regular traffic patterns. As result, we can use the SDN paradigm to extract and enforce these traffic patterns, which allows us to detect and prevent ARP spoofing attacks.

We now introduce SDN in general, the POX controller project in particular, and then show how we used MiniCPS to prototype a simple POX controller design to prevent such ARP spoofing completely in our testbed.

5.1 SDN Background

Software Defined Networking (SDN) is a novel architectural way to think about building networks and OpenFlow is the de-facto standard interface protocol between the SDN controlling logic and the network devices (physical and virtual). Both ideas were proposed by *M. Casado* and they derive from SANE [3], a protection architecture for enterprise networks.

SDN implementation defines a set of abstractions to provide separation of concerns at the *control plane*, in a similar way as the layering model that is used at the data plane. At the bottom of the stack there are network devices that form the physical topology. On top of that there is a Network Operating System (NOS) able to talk to each device and to serve a network view, in the form of an *annotated graph*, to the layer above. A virtualization layer is able to process this graph and provide only relevant details to the level above through an API. At the top of the stack there is the control logic that defines policy over the network assessing the processed graph. Communications between the control logic and the physical devices is bi-directional: network device messages will update the network graph and control plane messages will update the network policy. With this setting, the *end-to-end principle* is also applied to the control plane. The (complex) management of the network is shifted on the edges and central network devices merely act as relays, becoming an homogeneous set of forwarding objects referred as *datapaths*.

A Software-Defined network is managed by a control logic that dictates to the datapaths how to forward packets using a dedicated control plane protocol (e. g. OpenFlow). The control logic may be a centralized program running on a server (known as a SDN controller), a set of programs divided into modules by functionality (known as a Network OS) or even a set of equal or different distributed programs running on different servers. The control logic rules may act on single packets (micro-flow rules) or on a set of packets sharing some properties (aggregate-flow rules). The control logic strategy may be static and loaded at initialization time (proactive), dynamic and updated at runtime (reactive) or hybrid implementing both of them.

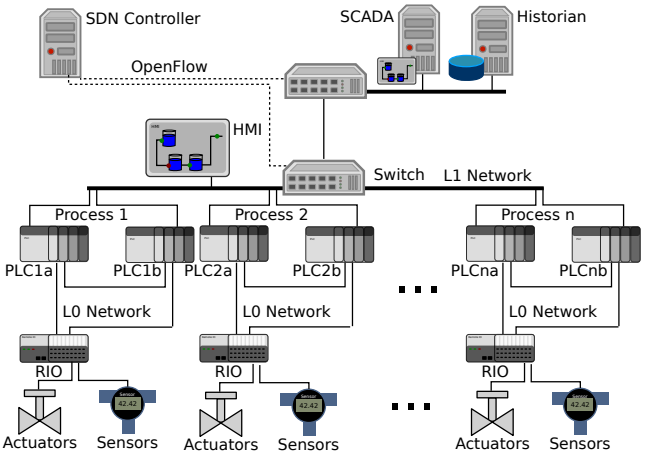

Figure 6: Extension of the generic ICS network with an OpenFlow switch and SDN controller.

There are various interesting projects regarding SDN and OpenFlow, and it is relatively easy to find a platform that implements the core modules, namely the NOS and the virtualization abstractions. In our work we decided to use the POX [19] platform because it is targeted for the research community, it offers out of the box libraries and components, and it is object-oriented, event-driven with synchronous and asynchronous handling capabilities. In addition, POX is completely written in Python and it integrates well with our set of tools (Scapy, CPPPO, Mininet, MiniCPS).

In the POX framework, events represent communication from the network to the control logic (e. g. new datapath connections) and callbacks represent communication from the control logic to the network (e. g. install a new rule). Event handling is priority based: the same event can be handled sequentially by different callbacks, a callback can potentially stop the handling chain and the same callback can handle multiple events.

5.2 Preventing MitM attacks with a custom SDN controller

We now present our SDN controller design, which aims to prevent the ARP spoofing attacks as discussed in the previous Section. In particular, our controller will analyze all ARP traffic, classify it as malicious or benign, and then update the SDN switches with suitable rules to prevent malicious attacks. Our attacker model consists of an attacker able to impersonate a network device (known or unknown to the other testbed hosts) that aims to mount a passive or active MitM attack using ARP poisoning over our real/emulated testbed L1 wired ENIP network.

Our POX controller implements a fully centralized SDN control plane with per-flow forwarding rules. Our control plane program uses both a proactive approach to perform a static pre-mapping and a reactive approach to adapt dynamically to the context. The detection and prevention code runs with higher priority than the management code and it is able to block the event handling chain.

Every time a new switch is connected to the network, our control logic will create a new reference to the *network state* accessible by the switch. According to OpenFlow protocol, when a switch doesn't know how to forward a packet it sends

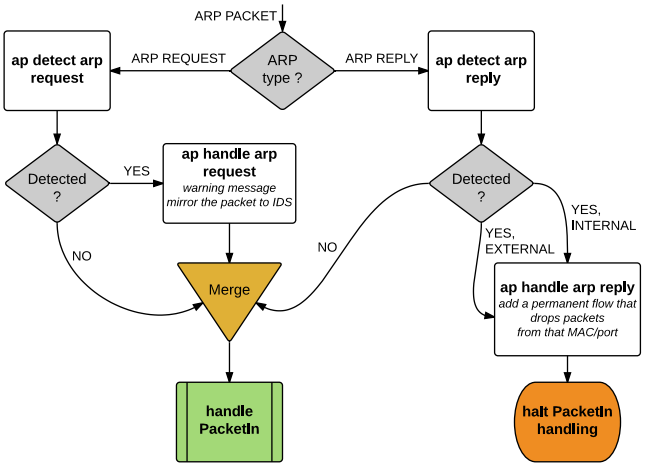

Figure 7: ARP spoofing prevention flow chart. The process either allows the PacketIn event handling (green) or blocks it (orange).

(a part of) it to the controller and it waits for instructions (that's when POX raises a PacketIn event). Our control logic process unknown ARP reply and ARP request packets, sent by the switches, verifying their consistency according to the network state. The controller is able to detect both *internal* and *external* ARP spoofing attempt and to prevent both *passive* and *active* ARP MitM attacks. In normal ARP request/reply circumstances, the controller dynamically updates the network state.

As shown in Figure 7, suspicious ARP request packets are signaled as warnings, and they are mirrored to a well-know port (potentially connected to a dedicated IDS for deep packet inspection), but the PacketIn handling chain is not halted. In contrast, suspicious ARP replies are actively managed: the control logic will install a permanent rule on the relevant switch to permanently block all the traffic coming from the attacker, the blacklist rule is based on the attacker MAC address and the used input port (but can be easily tuned according to the scenario). Finally, the controller will block the PacketIn handling chain.

In addition to this simple attack detection and prevention strategy, we are currently developing more elaborated ARP detection and mitigation techniques, in particular (i) an *ARP cache restoring* handler, and (ii) spoofing detection based on *static mapping* of MAC/IP pairs. The ARP cache restoring feature will periodically or asynchronously sends ARP replies to potentially every host in the network, forcing it to update its ARP cache with fresh and consistent data. The *strong static premap* feature will allow the controller to send to every new datapaths a set of predefined flow rules to speedup initial traffic congestion and policy establishment (e. g. who can talk to who). Eventually, this mechanism can be extended with a dynamic policy checker component, that is able to validate and restore the correct network state requesting and processing general and aggregated flow statistics directly from the datapaths. We think that those two add-ons will protect the network against (pre)mapped switches containing inconsistent rules and DoS attacks. We plan to extend our current centralized design into a more robust distributed scheme by using multiple synchronized controllers able to tolerate single point of failure in the control plane domain.

6. RELATED WORK

Security aspects of CPS have been discussed in [16,26,30, 31], in particular in the context of smart power grid infrastructure and control.

Process and network co-simulation. In [7], Dong *et al* propose a testbed that is similar to our MiniCPS platform in several ways. In particular, they propose to use Mininet as network emulation platform, a power grid simulation server, and a control center simulation server. The envisioned testbed uses Mininet to simulate delays related to dynamic network reconfigurations in the case of failures. In general, the authors just discuss the use case of the smart power grid, with component such as sensors and actuators connected to a central control via RTUs.

We note that MiniCPS differs from the testbed in [7] in several ways. Most importantly, MiniCPS' focus is on sharing reproducible CPS network topologies, in particular related to industrial control systems. MiniCPS focuses on using a set of PLC simulation tools, that directly interact with the network traffic, and the physical layer API. The physical layer API abstraction is not present in [7], as the authors propose the use of a powerful power-grid simulation tool (PowerWorld). In MiniCPS, the (generic) API allows to combine different types of physical layer simulations (e.g., combining water flow, mechanical levers, temperature transfer). Finally, the industrial protocol differs (ENIP vs. DNP3). From [7], it seems that the proposed testbed was not yet fully implemented.

In [4], a framework with similar intent as MiniCPS has been proposed. The framework uses OMnet++ as network simulation tool, and also features simulation of physical layer (e.g. a chemical plants). The authors simulated denial of service attacks on the sensor data, and the resulting control actions. As OMnet++ was used for network simulations, network communication was simulated as abstract messages that were routed through components, instead of simulating the full TCP/IP+industrial protocol stack. As a result, attacks such as our MitM ettercap manipulation could not be simulated in detail (i. e. considering all fields of the CIP/ENIP messages). On the other hand, simulations like [4] allow to use timescales other than real-time.

SDN. On the topic of SDN, SANE [3] represents one the first practical SDN-based solution for secure network design. The proposed implementation already included common SDN core concepts like centralized control logic, high level network policy design and easy network scalability.

SDN and OpenFlow projects involved from the beginning both academia and leading IT industries, that eventually found the Open Networking Foundation (ONF). There are several other recommended papers about SDN [9,21,24] and OpenFlow [17,28].

Physical layer simulation tools. PowerWorld is a commercial (interactive GUI-based) power transmission grid simulation. PowerWorld does not provide capabilities to directly interact with industrial communication protocols and does not simulate communication network aspects. In [1], the authors present a test bed which combines physical layer simulation with PowerWorld, and abstract network simulation based on the RINSE [15] tool.

7. CONCLUSION

In this work, we proposed MiniCPS, which uses Mininet together with a physical layer API and a set of matching component simulation tools to build a versatile and lightweight simulation system for CPS networks. While currently the physical layer simulation is very simplistic, we believe that our general framework will (a) researchers to build, investigate, and exchange ICS networks, (b) network engineers to experiment with planned topologies and setups, and (c) security experts to test exploits and countermeasures in realistic virtualized environments.

MiniCPS builds on Mininet to provide lightweight real-time network emulation, and extends Mininet with tools to simulate typical CPS components such as programmable logic controllers, which use industrial protocols (EtherNet/IP, Modbus/TCP). In addition, MiniCPS defines a simple API to enable physical-layer interaction simulation. We demonstrated applications of MiniCPS in two example scenarios, and showed how MiniCPS can be used to develop attacks and defenses that are directly applicable to real systems.

8. ACKNOWLEDGMENTS

We thank Nicolas Iooss for his support and contributions related to EtherNet/IP support in MiniCPS and the demonstrated attacks, and Pierre Gaulon for his help on the physical layer simulation. We also thank the anonymous reviewers for helpful feedback and suggestions.

9. REFERENCES

[1] D. C. Bergman and D. M. Nicol. Test bed for evaluation of power grid cyber-infrastructure. In P. M. K Popovici, editor, *Real-Time Simulation Technologies Principles, Methodologies, and Applications*. CRC Press, 2012.

[2] T. Bhat and N. Nagappan. Evaluating the efficacy of test-driven development: industrial case studies. In *Proc. of symposium on Empirical Software Engineering (ISESE)*, pages 1–8, 2006.

[3] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. SANE: a protection architecture for enterprise networks. In *Proc. of the USENIX Security Symposium*, pages 137–151, 2006.

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

[5] CISCO. Industrial ethernet: A control engineer's guide. `www.cisco.com/web/strategy/docs/manufacturing/industrial_ethernet.pdf`.

[6] R. de Oliveira, A. Shinoda, C. Schweitzer, and L. Rodrigues Prete. Using Mininet for emulation and prototyping software-defined networks. In *Proc. of Conference on Communications and Computing (COLCOM)*, pages 1–6, June 2014.

[7] X. Dong, H. Lin, R. Tan, R. K. Iyer, and Z. Kalbarczyk. Software-defined networking for smart grid resilience: Opportunities and challenges. In *In Proc. of The Cyber-Physical System Security Workshop (CPSS)*, April 2015.

[8] Ettercap Project. Ettercap. `https://ettercap.github.io/ettercap/`.

[9] N. Feamster, J. Rexford, and E. Zegura. The road to SDN. *ACM Queue*, 11(12):20–40, 2013.

[10] B. Galloway and G. P. Hancke. Introduction to industrial control networks. *IEEE Communications Surveys & Tutorials*, 15(2):860–880, 2013.

[11] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *Proc. of Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, CoNEXT '12, pages 253–264, New York, NY, USA, 2012. ACM.

[12] T. Issariyakul and E. Hossain. *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[13] P. Kundert. Communications protocol python parser and originator. `https://github.com/pjkundert/cpppo`. [Online; accessed 14-June-2015].

[14] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proc. of the SIGCOMM Workshop on Hot Topics in Networks*, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[15] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier. RINSE: The real-time immersive network simulation environment for network security exercises. In *Proc. of Workshop on Principles of Advanced and Distributed Simulation (PADS)*, pages 119–128, 2005.

[16] J. Lin, W. Yu, X. Yang, G. Xu, and W. Zhao. On false data injection attacks against distributed energy routing in smart grid. In *Conference on Cyber-Physical Systems (ICCPS)*, 2012.

[17] N. Mckeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, J. S. Turner, and S. Louis. OpenFlow: enabling innovation in campus networks. *Computer Communication Review*, 38(2):69–74, 2008.

[18] J. R. Moyne and D. Tilbury. The emergence of industrial control networks for manufacturing control, diagnostics, and safety data. *Proceedings of the IEEE*, 95(1):29–47, Jan 2007.

[19] NOXRepo.org. The pox controller. `https://github.com/noxrepo/pox`. [Online; accessed 14-June-2015].

[20] ODVA. Ethernet/IP technology overview. `https://www.odva.org/Home/ODVATECHNOLOGIES/EtherNetIP.aspx`.

[21] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks [white paper]. *ONF White Paper*, pages 1–12, 2012.

[22] T. Phinney. IEC 62443: Industrial network and system security. `https://www.isa.org/pdfs/autowest/phinneydone/`.

[23] A. Ruscito. Pycomm: a collection of modules used to communicate with plcs. `https://github.com/ruscito/pycomm`. [Online; accessed 14-June-2015].

[24] Thenewstack.io. SDN Series, 2015.

[25] A. Varga et al. The OMNeT++ discrete event simulation system. In *Proc. of the European simulation multiconference (ESM)*, page 65. sn, 2001.

[26] E. Wang, Y. Ye, X. Xu, S. Yiu, L. Hui, and K. Chow. Security issues and challenges for cyber physical system. In *Proc. of Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 733 –738, December 2010.

[27] S. Whalen. An introduction to ARP spoofing. `machacking.net/kb/files/arpspoof.pdf`, 2001.

[28] W. You and K. Qian. OpenFlow security threat detection and defense services. *Int. J. Advanced Networking and Applications*, 2351:2347–2351, 2014.

[29] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou. OrchSec: An orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions. In *Proc. of Network Operations and Management Symposium (NOMS)*, pages 1–9, May 2014.

[30] B. Zhu, A. Joseph, and S. Sastry. A taxonomy of cyber attacks on SCADA systems. In *Proc. of Conference on Cyber, Physical and Social Computing*, pages 380–388, 2011.

[31] S. Zonouz, K. Rogers, R. Berthier, R. Bobba, W. Sanders, and T. Overbye. SCPSE: Security-oriented cyber-physical state estimation for power grid critical infrastructures. *Smart Grid, IEEE Transactions on*, 3(4):1790–1799, Dec 2012.