Doctoral Committee:

Ph.D. Sandra Rueda
Associate Professor of Systems and Computing Engineering
Universidad de Los Andes
Advisor

Ph.D. Alvaro Cardenas
Associate Professor of Computer Science and Engineering
The University of California at Santa Cruz
Co-Advisor

Ph.D. Juan Felipe Botero
Assistant Professor of Computer Science and Engineering
Universidad de Antioquia
Jury

Dr. Ing. Otto Carlos Muniz Bandeira Duarte
Full Professor of Electronic Engineering
Universidade Federal do Rio de Janeiro
Jury

Ph.D. Yezid Enrique Donoso
Full Professor of Computer Science and Engineering
Universidad de Los Andes
Jury
A mi familia, amigos y profesores
# Contents

<table>
<thead>
<tr>
<th>List of Figures</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>7</td>
</tr>
</tbody>
</table>

## 1 Introduction 8

## 2 Theoretical Framework 14

### 2.1 Industrial Control Systems 14

### 2.2 Cyber Security Challenges in ICS 15

### 2.3 Software Defined Networking 16

#### 2.3.1 Software Defined Networking Architecture 16

##### 2.3.1.1 OpenFlow 17

##### 2.3.1.2 SDN Controllers 18

### 2.4 Network Function Virtualization 20

#### 2.4.1 OPNFV - A Platform for NFV 21

## 3 Software Defined Networking and Network Function Virtualization to Enhance ICS Security 24

### 3.1 Virtual Infrastructure for ICS Incident-Response 25

#### 3.1.1 Use-Cases of SDN and NFV-enabled Incident Response in Supervisory and Field ICS Networks 27

### 3.2 Implementing SDN and NFV in Control Systems 30

#### 3.2.1 Description of the Process 31

#### 3.2.2 Implementing ICS Networks 32

#### 3.2.3 SDN Controller Application 34

#### 3.2.4 Design of our Physics-Based IDS 35

### 3.3 Experiments and Use-Cases 35

#### 3.3.1 Responding to Compromised Sensors 35

#### 3.3.2 Responding to a Compromised PLC 37

#### 3.3.3 Responding to an Attempted SCADA Compromise 40

#### 3.3.4 Performance 42

#### 3.3.5 Responding to False Alarms 42

### 3.4 Related Work 44

### 3.5 Conclusions and Future Work 45

### 3.6 Acknowledgments 46
4 A Virtual Environment for Industrial Control Systems: A Nonlinear Use-Case in
Attack Detection, Identification, and Response

4.1 Related Work .................................................. 48
4.2 Networked Control Systems - Fault Tolerant Control ................. 51
  4.2.1 Controlled System ....................................... 51
  4.2.2 Networked Controller .................................... 52
  4.2.3 Integrity Attacks ......................................... 53
  4.2.4 Anomaly Detection and Attack Mitigation .................... 53
4.3 Experimental Setup ............................................ 55
4.4 Results .......................................................... 58
  4.4.1 Integrity Attack ........................................... 61
4.5 Conclusions and Future Work .................................. 62
4.6 Acknowledgments ............................................... 63

5 SDN and NFV Security: Challenges for Integrated Solutions .......... 64
5.1 SDN and NFV Integration ....................................... 66
  5.1.1 An Integrated Architecture ............................... 67
  5.1.2 Orchestration and Management in SDN/NFV .................. 68
5.2 A Survey of Proposals to Secure SDN/NFV Platforms ................. 71
  5.2.1 Taxonomy .................................................. 71
  5.2.2 Analysis .................................................... 74
  5.2.3 Limitations ............................................... 78
5.3 New Directions in Mandatory Access Control Systems for SDN/NFV .... 79
  5.3.1 Access Control ............................................ 81
5.4 Conclusions and Future Work .................................. 82
5.5 Acknowledgments ............................................... 83

6 Access Control Policies for Virtual ICS Systems ....................... 84
6.1 Virtualized ICS Systems ........................................ 84
6.2 Access Control Framework for Virtualized ICS Networks ............. 86
  6.2.1 Access Control for Virtualized ICS ......................... 87
    6.2.1.1 Resources .............................................. 87
    6.2.1.2 Operations ............................................ 88
    6.2.1.3 Users and Roles ...................................... 88
  6.2.2 Policy Engine ............................................. 91
    6.2.2.1 Policy Specification .................................. 91
    6.2.2.2 Policy Language ...................................... 91
    6.2.2.3 Policy Language Discussion ......................... 93
    6.2.2.4 Policy Enforcement .................................... 94
6.3 Implementation .................................................. 94
  6.3.1 Oslo HTTP Checks ......................................... 96
  6.3.2 Policy Server ............................................... 97
  6.3.3 Evaluation ................................................. 98
6.4 Related Work ................................................... 102
6.5 Conclusions and Future Work .................................. 103
6.6 Acknowledgments ............................................... 104
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Basic Topology of an Industrial Control System (ICS)</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>General SDN Architecture</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Services of a Floodlight SDN Controller to SDN Applications</td>
<td>19</td>
</tr>
<tr>
<td>2.4</td>
<td>European Telecommunication Standards Institute (ETSI) NFV Architecture</td>
<td>22</td>
</tr>
<tr>
<td>2.5</td>
<td>OPNFV Architecture</td>
<td>23</td>
</tr>
<tr>
<td>3.1</td>
<td>Virtual industrial control facility</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Alternatives for enhancing industrial control systems with software defined networking</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Proposed cloud-based infrastructure for industrial control systems</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>Industrial control system with SDN/NFV enhanced security</td>
<td>31</td>
</tr>
<tr>
<td>3.5</td>
<td>Alternatives to enhance ICS field networks with SDN</td>
<td>34</td>
</tr>
<tr>
<td>3.6</td>
<td>Enhanced industrial-control system with SDN</td>
<td>36</td>
</tr>
<tr>
<td>3.7</td>
<td>SDN controller application</td>
<td>37</td>
</tr>
<tr>
<td>3.8</td>
<td>Attack scenario with compromised sensor</td>
<td>38</td>
</tr>
<tr>
<td>3.9</td>
<td>Experiment with compromised PLC</td>
<td>39</td>
</tr>
<tr>
<td>3.10</td>
<td>Attack scenario with PLC compromised</td>
<td>40</td>
</tr>
<tr>
<td>3.11</td>
<td>Attacker probing the SCADA</td>
<td>41</td>
</tr>
<tr>
<td>3.12</td>
<td>Behavior of the water level in a tank with a random perturbation</td>
<td>43</td>
</tr>
<tr>
<td>3.13</td>
<td>Water tank behavior with false alarms</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>Networked control system with state feedback and reference tracking</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Bias injection attack</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Mechanism to mitigate sensor attacks</td>
<td>54</td>
</tr>
<tr>
<td>4.4</td>
<td>Experimental setup for virtual experimentation non linear plant</td>
<td>56</td>
</tr>
<tr>
<td>4.5</td>
<td>MiniCPS experiment topology</td>
<td>57</td>
</tr>
<tr>
<td>4.6</td>
<td>Water tank level behavior without an attack</td>
<td>60</td>
</tr>
<tr>
<td>4.7</td>
<td>Bias attack experiments</td>
<td>61</td>
</tr>
<tr>
<td>4.8</td>
<td>Mean error of tank1 water level with and without defense</td>
<td>62</td>
</tr>
<tr>
<td>5.1</td>
<td>Extended SDN/NFV architecture</td>
<td>68</td>
</tr>
<tr>
<td>5.2</td>
<td>Technologies that can support an SDN/NFV architecture</td>
<td>70</td>
</tr>
<tr>
<td>5.3</td>
<td>Potential adversaries in SDN/NFV domains</td>
<td>73</td>
</tr>
<tr>
<td>5.4</td>
<td>Architecture of a Secure Network Operating System</td>
<td>80</td>
</tr>
<tr>
<td>5.5</td>
<td>Policy compiler and high level policies</td>
<td>82</td>
</tr>
<tr>
<td>6.1</td>
<td>Industrial control system</td>
<td>85</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>6.2 Virtual industrial control system</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>6.3 Access control framework for virtual ICS</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>6.4 Policy engine implementation</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>6.5 Policy load time</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>6.6 Single request response time</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>6.7 Single request response time with concurrent requests</td>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Differences between the original MiniCPS and our proposed environment</td>
<td>33</td>
</tr>
<tr>
<td>4.1</td>
<td>Summary of related work on testbeds for Industrial Control System Security experimentation</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>Parameter values of the three tank system</td>
<td>58</td>
</tr>
<tr>
<td>5.1</td>
<td>Classification of proposals to orchestrate and manage SDN/NFV environments</td>
<td>69</td>
</tr>
<tr>
<td>5.2</td>
<td>Classification of proposals to secure SDN/NFV environments</td>
<td>75</td>
</tr>
<tr>
<td>6.1</td>
<td>Summary of available operations to handle network resources and virtual instances</td>
<td>89</td>
</tr>
<tr>
<td>6.2</td>
<td>Operations performed on instance resources and their impact in the virtual ICS</td>
<td>90</td>
</tr>
<tr>
<td>6.3</td>
<td>Operations performed on network resources and their impact on the virtual ICS</td>
<td>90</td>
</tr>
<tr>
<td>6.4</td>
<td>Roles examples and their description</td>
<td>98</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Industrial Control Systems (ICS) are networked systems responsible of the functioning of multiple industrial processes such as energy generation and distribution, water distribution, and several industrial processes in the manufacturing sector. Nowadays, ICS play a critical role in our society and their security and continuous functioning is essential to our economies. The stability and security of ICS systems has been recognized as a priority issue for different governments and agencies. For this reason, ICS systems are considered critical infrastructure [18]. The US National Institute of Standards and Technology (NIST) argues that the United states depend on the reliable functioning of critical infrastructure. Cybersecurity threats to these systems place the nation security, economy, and public safety and health at risk [18]. These entities recognize that cyber attacks on ICS systems not only bring financial losses [24], but also could endanger the lives of thousands [102]. Attacks such as the one carried in the Ukraine electrical grid compromised the SCADA systems of some 30 distribution substations in the capital city Kiev and Western Ivano-Frankivsk region, causing more than 200000 consumers to lose power [131]. In addition, this attack was present during the winter season, affecting the access to essential heating services in these cities. This kind of attack reflects the dire need to harden the equipment and protocols present in ICS infrastructures.

Traditionally, these systems were isolated and did not have any communication with other networks, private or public, including internet. Nevertheless, around 2011 the German government created the term "Industry 4.0" to promote the extension of traditional manufacturing environments with information technology (IT) [56] in order to make these environments more flexible and productive; since then, ICS owners have been extending ICS systems with this kind of technology.

This new type of setting can greatly benefit ICS systems in several ways [73][171]: i) a paradigm shift from an static controlled process to an information driven process; manufacturing equipment, with additional devices, monitors the process and can adapt according to specific needs as they are detected; ii) integration of Internet of Things into the manufacturing process; this means interaction with neighboring smart manufacturing components to reach common goals. This integration enables the development of smart factories that are context aware and assist people and machines in the execution of their tasks.

Nevertheless, the integration of telecommunication networks, higher computational capabilities, and Internet of Things components to ICS networks creates security challenges that need to be addressed in order to ensure the proper operation of ICS systems. The challenges
arise for several reasons:

• First, ICS elements have high production costs that usually lead to very long equipment lifetime [82]. As a consequence, some ICS components are too old and do not run modern security measures, they could lack integrity, authenticity, and hardening mechanisms to protect themselves.

• Second, ICS networks did not have security as a main design goal; they were designed under the assumption that they would always be physically isolated [54].

• Third, ICS networks use legacy protocols that are not appropriately hardened for open environments; ICS networks have authenticity and integrity issues that need to be addressed, an attacker could forge and inject malicious packets to alter a component and the ICS expected behavior. Popular industrial protocols such as Profinet [154] and Modbus [91] lack integrity or authenticity protection mechanisms in their default configurations.

The mentioned challenges must be addressed to have ICS systems that are secure in the presence of current connectivity settings and open networks. Nevertheless, this process is just beginning for ICS systems, where particular features, such as real time and high redundancy requirements and interaction with the physical world, must be considered. The associated challenges include a broad set of topics such as integrity of the software used in ICS systems, availability of the network interconnection of ICS elements, integrity and privacy of the exchanged messages and access control mechanisms to allow only authorized users to perform actions over the ICS components.

We require new technologies that can offer more security to ICS systems without the need to replace most of the equipment and protocols already present in ICS infrastructures. We propose exploring new technologies that offer the option of dynamically changing configuration and adding components. These options would enable an ICS system to answer to security incidents and to implement security mechanisms that do not require modifying existing ICS networks. In this thesis we explore the use of Software Defined Networking and Network Function Virtualization in the context of ICS systems to enhance their security.

• **Software Defined Networking (SDN)** is a network paradigm that divides traditional networks in two cooperative planes: data plane and control plane. The former includes switches and elements forwarding data packets between different interfaces. The latter is in charge of making decisions to keep or adjust network behavior. The control plane may be considered as the intelligence of the network; it must decide about routing rules, load balancing, security incidents and requirements of specific applications. Additionally, the control plane may be programmed and extended through generic programming languages and can export its capabilities through APIs. Using these APIs, developers can build custom applications to answer to very specific situations. For example, using SDN it is possible to consider additional network metrics or to change routing configuration depending on current network conditions.

Therefore, SDN brings different benefits to a traditional network such as: i) the control plane has a global vision of the network and can make decisions regarding general
network state, ii) SDN can dynamically adjust network behavior as a reaction to different incidents, iii) ICS administrators may use generic programming languages to develop and deploy programs that implement strategies to dynamically adapt their network to respond to sporadic or new conditions, such as preventive maintenance, faulty equipment, and security incidents.

SDN’s global vision of a network enables operators to identify attacks to different control loops in ICS; administrators of traditional networks cannot easily consolidate data because they do not have one network component with a global vision of the network, they would need to develop and deploy additional distributed agents to collect data and a centralized component to consolidate reports. Additionally, through the use of generic programming languages we can develop specific ICS elements, like state estimators, to detect sensor errors or compromises, and keep ICS expected operation. SDN can also reconfigure a network in the presence of an attack. This could also be achieved with a traditional network, but the global vision and dynamic capabilities of SDN make this process more secure and predictable.

- **Network Function Virtualization (NFV)** is a particular application of Cloud Computing ideas where virtual machines offer network services to different tenants.

In traditional networks, a network service is built by connecting different middleboxes in a chain of services. For example, a load balancer, followed by a firewall, an IDS, and a set of routers. Since this kind of configuration was based on physical equipment, updating such network implied physically reconnecting these devices, which could cause service downtime. In addition, middleboxes are proprietary in most cases and it is difficult to modify or add features to them.

Using NFV, providers may invest in generic hardware and virtualization to offer network services instead of investing in specific middleboxes. This change in configuration also allows network administrators to dynamically reconfigure a network, without causing downtime. Adding new services or features is now a task of software updating.

NFV also adds a particular component: an orchestration service. This is relevant as network services can be divided in phases, each phase running on a different virtual machine, running in a coordinated way to offer a complete network service. One possible scenario is a load balancing service followed by a firewall and an IDS system. Traditionally, each one of these services was offered by a specific hardware element or a set of them; with NFV, network managers can dynamically create and update this service. The orchestrator uses different technologies, such as SDN, to interconnect services and create the required service chain. Finally, the orchestrator monitors percentage of resource use in virtual machines and network links and decides whether to scale virtual resources up or down accordingly.

ICS administrators may use NFV to create Virtual ICS Infrastructures with operational benefits for ICS owners. Also, NFV enables ICS administrators to dynamically create service chains to handle events that may happen in an ICS system.

Previous works have considered the idea of using SDN and NFV to improve network security. Those works, however, do not consider an ICS context. The work presented in [149]...
discusses several security benefits for communication networks such as i) Dynamic flow control that enables administrators to dynamically change the rules that govern network flows and access control rules. These flows and rules can handle different levels of granularity, ii) network wide visibility and centralized flow control can ease network administration without installing additional monitoring equipment, and iii) network programmability that allows developers to build tailored applications that can answer to specific network challenges or to modify existing applications in the face of new threats. In this thesis, we use these SDN features to increase the security of ICS infrastructures.

A similar approach is followed by Hirata et al. in [74] that use SDN and virtual machines to create honeypots to protect web servers. Our thesis uses SDN and NFV to enhance security of industrial control systems through the use of virtual security response functions and dynamic network reconfiguration. SDN and NFV can also be used to develop virtual testing environments for ICS.

Some authors have worked on the development of testbeds for ICS systems [53, 55, 84, 107]. We believe that extending these environments is relevant because this approach avoids the costs generated by the use of testbeds that have physical equipment like [28, 53, 62].

The thesis of this dissertation is therefore:

Because of the addition of IT technology to Industrial Control Systems (ICS), they need to be protected against traditional and new attacks. SDN and NFV have capabilities, already used to enhance network security, that can also be used to implement services that enhance security of Industrial Control Systems (ICS). SDN and NFV capabilities make it possible to automatically start virtual functions, dynamically change network configuration to include or remove components, and redirect traffic through them, thus enabling ICS administrators to mitigate impact of security incidents.

In order to evaluate the thesis of this work, we defined the following research activities:

• Understand SDN and NFV capabilities and use them to launch incident response security functions and automatically reconfigure an ICS network and its components in presence of an identified malicious attack.

• Understand the capabilities of virtual environments to design and deploy virtual testing platforms for ICS, that run various plant models and mitigation strategies; making it possible to evaluate, in advance, the impact of these interactions in a real ICS environment.

• Understand the capabilities of SDN and NFV to deploy virtual industrial control systems that can provide multiple instances of the ICS components to be used by different users, with specific necessities. The use of SDN and NFV can bring economical costs to implement virtual ICS, because the infrastructure owners can invest in generic equipment rather than network or ICS proprietary equipment.

• Understand access control requirements that arise in the context of virtual industrial control systems, to protect these systems from unauthorized access, and design and implement an access control framework that answers to those needs.
As a result of the work, we made the following contributions:

- We developed a proof of concept of an SDN application that dynamically reconfigures a virtual ICS network in response to a detected intrusion. This proof of concept effectively protected an ICS system from a compromised sensor, sending malicious measurements to a PLC. SDN is an effective tool for ICS systems to react to security incidents, without modifying the protocols used in the ICS network.

- We extended a virtual environment to build a virtual ICS environment with virtual incident response functions that react to different attacks performed in the ICS network. These virtual incident response functions can effectively react to a set of security attacks on the ICS network.

- We presented an extensive survey of the main proposals aimed to enhance the security of SDN and NFV technologies. To our knowledge, this is the most comprehensive survey presented in the topic related to SDN and NFV technologies. This survey also shows we are missing access control frameworks for SDN and NFV that enable administrators to write access control policies using a high level language; current proposals use a language that is dependent of the cloud technology implementation.

- We extended a virtual environment to test the security of an ICS network controlling a non linear plant. To our knowledge, this was the first work done in a virtual environment for ICS networks, using non linear plants. This extension is relevant because many real world plants actually are non linear. We realized that this kind of plants are very sensitive to delays or measurement approximations making them harder to protect against security attacks; we need protection systems with traditional technology, but also capable of detecting small variations in sensor readings.

- We presented a proof of concept of an access control framework for virtual ICS systems in OpenStack. We developed a language to define high level policies that mandate access control strategies for virtual ICS. This language allows administrators to define policies that are independent of the cloud technology implementation. In addition, we decided to use a Role Based Access Control (RBAC) approach to make it easier for administrators to assign permissions to users (through roles).

During the development of the present thesis we published the following papers:


- *Virtual Incident Response Functions in Control Systems* in Elsevier Computer Networks (Apr 2018). Andres F. Murillo, Vikram Gaur, Jairo Giraldo, Álvaro A. Cárdenas, Sandra Julieta Rueda [133]. ISSN 1389-1286. This publication and the previous one were extended during the first chapter of this thesis.
• **Leveraging Software-Defined Networking for Incident Response in Industrial Control Systems** in IEEE Software (January 2018). Andres F. Murillo, Vikram Gaur, Jairo Giraldo, Álvaro A. Cárdenas, Sandra Julieta Rueda [119]. ISSN 1937-4194. This chapter was extended in the second chapter of this thesis.


In addition, we presented our work in various conferences:

• "FlowFence, a Denial of Service Defense System Based on Software Defined Networking", in the conference Conference BSides in Bogotá, Colombia (2015)


Finally, we presented our work in two posters in the following presentations:

• "Towards a Secure Operating System for SDN/NFV" in The Universidad de los Andes Meeting of Research in Engineering in Bogotá, Colombia (2017)


This document is organized as follows. Chapter 3 presents the theoretical framework of the thesis and explains the concepts of Software Defined Networking, Network Function Virtualization, and Industrial Control Systems. That chapter also explores the use of SDN and NFV to create virtual incident response functions in ICS environments. Chapter 4 uses a virtual environment to emulate a non linear model plant and discusses the use of virtual environments to create scalable testbeds for ICS systems. Chapter 5 presents a continuation of NFV capabilities and presents an extensive survey of security mechanisms that aim to enhance the security of SDN and NFV. Finally, Chapter 6 explores the access control challenges that arise in a virtual ICS infrastructure and proposed an access control framework for such environments.
Chapter 2

Theoretical Framework

This chapter establishes the theoretical framework of the present thesis. The topics discussed include: Industrial Control Systems (ICS) and their security challenges, Software Defined Networking (SDN), and Network Function Virtualization (NFV). Since one of the objectives of these thesis is to explore the use of SDN/NFV to enhance the security of ICS systems.

2.1 Industrial Control Systems

Industrial Control Systems (ICS) are networked systems that have the objective of controlling an industrial process. To control an industrial process means that local or remote trusted operators use the ICS to change the behavior of an industrial process according to his needs. In this sense, the key aspect of an industrial process is to eliminate the steady error of the industrial process. The steady error is the difference between the behavior expected by the operator and the behavior that the industrial process has in a certain moment.

Figure ?? shows an example of the main components of an ICS system. In ICS systems, two types of networks are clearly delimited: Field Networks and Supervisory Networks. Field Networks connect the sensors, Programmable Logic Controllers (PLCs), and actuators. Supervisory networks interconnect the Field Network with a set of SCADA Servers and other type of servers such as historians. The field network is in charge of maintaining the state of the physical process within certain configured parameters. In order to do this, the field network is composed of a series of control loops. A control loop is a networked system composed of a physical process, a set of sensors, a set of actuators, and one or more Programmable Logic Controllers (PLCs) controlling the process. The Field Network is closest to the physical process and for this reason uses protocols that are specifically tailored for ICS. Most of these protocols have a deterministic behavior with real time guarantees. For this reason, the protocol stack in the field network are different that the traditional TCP/IP stack. Some of the protocols used in this type of networks are Reatl Time Ethernet, Ethernet over IP (ENIP), Controller Area Network (CANBus), EtherCAT, and Modbus.

As explained earlier, the objective of an ICS is to maintain a physical process within a desired behavior. This works as follows: i) In the supervisory network, the ICS operator uses an interface to connect with the SCADA server and configure the parameters of operation of the industrial process. These parameters are called "set points". ii) The SCADA server
forwards this information to the PLCs. iii) In the field network, the sensors measure physical
variables from the physical process and report these measurements to the PLC. iv) The PLC
compares the measurements reported by the sensors and the current state of the system.
The difference between the setpoints and the current state is called the stationary error. v) The PLC applies a control strategy to reduce the steady error to zero; this means calculating
which action should the actuators perform on the physical process to make their behavior
closer to the setpoints. vi) The PLC sends the commands to the actuators. vii) The actuators
apply their actions to the physical process and the process repeats from step ii.

Figure 2.1: Basic Topology of an Industrial Control System (ICS). An ICS is composed of
two networks: Field Network and Supervisory Network. The Field Network has one or
more control loops where a PLC controls a physical process by receiving information from
sensors, calculating control actions and sending commands to the actuators. The supervisory
networks enables the operators to know the process state and configure its operation setpoints

2.2 Cyber Security Challenges in ICS

Industrial Control Systems are considered critical infrastructure [18]. This is because
countries depends on the reliable functioning of ICS providing critical services such as: electrical power, water distribution and treatment, and manufacturing processes. Cybersecurity threats to these systems place the nation security, economy, and public safety and health at risk [18]. In addition, this type of networks were traditionally isolated from public networks and were only accessed in a physical local way. This had the consequence that the protocols and technologies used in such systems did not have security as a design principle. Nevertheless, around 2011 the German government created the term "Industry 4.0" to promote the extension of traditional manufacturing environments with information technology (IT) [56] in order to make these environments more flexible and productive; since then, ICS owners have been extending ICS systems with this kind of technology. Industry 4.0 might benefit
ICS systems in several ways: i) a paradigm shift from an static controlled process to an information driven process; manufacturing equipment, with additional devices, monitors the process and can adapt according to specific needs as they are detected; ii) integration of Internet of Things into the manufacturing process; this means interaction with neighboring smart manufacturing components to reach common goals. This integration enables the development of smart factories that are context aware and assist people and machines in the execution of their tasks. The integration of IT and Internet of Things (IoT) technologies to ICS generates a scenario in which: i) ICS run traditional legacy protocols that were meant to operate in isolation and now should operate in an environment with other types of networks, ii) ICS equipment has long life cycles due to their high replacement cost, meaning that they operate legacy systems difficult to patch and harden. In this scenario we have challenges related to confidentiality, integrity, authenticity, and incident response capabilities.

2.3 Software Defined Networking

Software Defined Networking (SDN) is a telecommunication paradigm that proposes two significant changes in network architecture: layer separation and the control layer programmability. Layer separation enables the division of the functions present in a communication network in two layers: the data layer and the control layer. The data layer is in charge of packet forwarding between router ports. The decisions about to which port route the packet or what set of actions should be applied to the packet are taken in the control layer. The control layer is in charge of taking all the decisions regarding the packets exchanged in the network. The main component of this layer are the controllers, devices that control network behavior; for example, controllers deploy routing algorithms. Regarding the second change, control layer programmability, this programmability allow the controllers to run programs that implement high-level network functions, like deciding about the routing algorithms that routers must implement or applying security policies to the network. Control software is developed in generic programming languages, like Java, C++, and Python. This feature allows quickly development of new network policies to cope with varying network needs.

2.3.1 Software Defined Networking Architecture

Figure 2.2 shows the SDN architecture with its three layers: data, control and application layers. The data layer receives and executes orders from the control layer, for example commands to populate routing tables. In addition, it sends notifications and reports to the control layer about network state. The control layer receives this information and builds a global vision of the network. The control has tools to reconfigure network features based on that vision and on decisions taken by the application layer. The application layer takes all the decisions regarding policy and network behavior and sends commands to the control layer in order to reconfigure the network if needed.

The data layer, also called infrastructure layer, is composed of devices that forward packets, like routers and switches. The network controllers are located in the control layer; these devices run programs that offer routing services, load balancing services and other high-level services in the network. In the early years of SDN, the controllers were monolithic, this means that only one controller was in charge of managing a network segment.
monolithic approach, however, created scalability problems \cite{176}. Nowadays, multiple controllers manage a network, adding redundancy and solving these scalability issues. There are diverse approaches in the deployment of network controllers, some approaches are hierarchic \cite{69} and others distribute control across several controllers \cite{20,42,92,177}. Despite of the distributed nature of the control layer, all the controllers must run the same version of the network applications. For this reason, the control layer is centrally designed (the applications are written and updated once), but implemented in a distributed way (various controllers implement the application logic).

Finally, the applications controlling the network run in the application layer. These applications can be developed on any generic language that has an API to the SDN controller being used in the control layer.

SDN also defines standard interfaces that enable communication between layers, the Southbound and Northbound interfaces, and between controllers, the East and Westbound interfaces. The southbound interface handles communication between the control and data layers. This interface translates the commands delivered by the controllers, in values in the routing tables of the data layer forwarding devices. The most popular southbound interface is OpenFlow, as a matter of fact, it has become a de facto standard. Due to security and performance reasons, the southbound interface uses an out-of-band channel, this allows to isolate the data layer packets, from the administrative packets of the control layer. In addition, the OpenFlow standard mandates that communications in the southbound interface use a secure channel \cite{160}. The Northbound interface enables communication between controllers and applications. The APIs of this interface abstract network and controller capabilities and offer these capabilities to the applications. The East and West bound interfaces allow controllers to synchronize network state control among controllers.

### 2.3.1.1 OpenFlow

OpenFlow is the de facto standard for Southbound interfaces. OpenFlow was developed at Stanford University in 2008 \cite{109}. In Openflow, a set of packets that share the same combination of values at their headers are called "flows". For example, a flow can be defined as "all packets with 10.1.1.1 as source IP". In this way, SDN can apply the same control decisions to flows and not to individual packets. Despite of this, an application can have different levels of granularity in a flow definition. OpenFlow in its version 1.5 has 14 mandatory header fields and 24 optional fields in a flow definition \cite{160}. The patterns used to match values may be specific values or wildcards. The use of wildcards allows to define more general flows, which simplifies network management. Routing and other network commands are represented as entries in flow tables. These tables define duplets of "flow, action", that indicate the operations that should perform the forwarding devices with the packets that are processing. Actions defined by OpenFlow are: forward, drop, enqueue, modify and set value in header. These actions can be sequentially chained on the same flow. In this way, the controller function is to translate the high level requests performed by the applications in entries in the flow tables of the respective forwarding devices. When a packet arrives to a forwarding device and there is no defined flow match for that packet, an event called "miss" is produced. In this event, the first packet of the flow is sent to the controller, via the southbound interface. The controller, using the developed applications, define the set of actions that should be performed on this.
The control layer can build a global vision of the network, and may reconfigure network features based on that vision. The data layer receives and executes orders from the control layer to adjust network behavior. The application layer receives and synthesizes information about network behavior. Using this information, it takes decisions that can change the network configuration. The SDN architecture also defines standard interfaces for communication between layers: the Southbound, the Northbound, and the east and westbound interfaces.

2.3.1.2 SDN Controllers

SDN Controllers are the central element in SDN, since they offer to the applications the capacity to control SDN networks [63]. The controllers are responsible for populating the flow tables of the forwarding devices. This process can be done in a proactive, reactive approach, or a mix of both. In the first approach, the controller pre-installs rules in the flow tables. Thus, the rule installation occurs before the arrival of the first packet to a forwarding device. The main advantage of this approach is the reduction in the forwarding latency and a reduced load in the control plane. The reactive approach uses the *miss* events to populate the forwarding devices flow tables. In addition to the task of populating the devices flow tables, the controllers collect information about the network state. This can also be done through notification events generating from the routers or performing periodic polls on the devices. This network state information is exported to the applications controlling the network.

The controllers offer high level functions to the applications to interact with the network. These functions are offered through APIs defined by the controller. Figure 2.3 presents an example of this scheme. This figure shows the relation between the Floodlight controller [145]...
and some developed applications. The applications can be developed in Java and interact with the Floodlight Java API or can be developed in another languages and interact via the REST API.

To avoid performance problems and offer greater resilience, several controllers can be in charge of controlling a network. The OpenFlow protocol allows a switch to be connected to various controllers, choosing one as the primary and the others as backups. Other controllers, like Onix [92], offer synchronization services between different controller instances. This approach was inherited by modern controllers like Floodlight [145], ONOS [20], OpenDayLight [89], and Neutron [161].

Figure 2.3: Services of a Floodlight SDN controller to SDN applications. A controller offers two type of services: core and internal dependent. Internal dependent services are high level services that the controller offers to the application. These services abstract network capabilities and applications may call them via APIs. The applications have access to two types of APIs, Java and REST. [145]. REST calls can be used by applications that are not developed in Java.

The following paragraphs present an analysis of several available SDN controllers.

- NOX was the first OpenFlow controller available. NOX is a monolithic controller that offers the basic functions of an SDN controller: allows to collect flow statistics from the SDN switches; allows to create, modify and delete entries in the flow tables using reactive or proactive approaches [63]. NOX is developed in C++, but there exists a Python version called POX [41].

- Maestro is a multi-thread controller. Maestro equally distributes the tasks among a set of threads. In addition, processes sets of flows requests in batch, this increases processing efficiency [80]. For this reason, Maestro posses an almost lineal scalability for flow processing requests.
Beacon was developed at Stanford university. It is a multi-thread processor with modular construction [48]. The API to interact with beacon is based on events. Beacon uses a patron observer where the listeners registry for events. In addition, Beacon is the code base of the Floodlight controller.

Floodlight is a controller developed from Beacon. The applications use a REST or Java API to make requests to the controller modules. Because of its performance and application development easiness [116], this controller is supported by fabricants like Cisco, Intel, HP and IBM. Floodlight allows to create multiple controller instances to distribute the network control load.

OpenDayLight is a controller implemented in Java. The controller is designed in layers, where the upper layers are composed of the applications that control the network and business logic [87]. The middle layer is a framework and controller core and the lower layer is composed of the real and virtual devices. OpenDayLight supports OpenFlow and other southbound interfaces.

ONOS is a controller designed to make its core distributed [20]. ONOS is event based, and different modules register to events.

Neutron was not designed as an SDN controller, but as the network control component of OpenStack [161]. OpenStack is a platform for cloud computing. Neutron components are a main service and a set of plugins. The main service listens to requests and events from different network components and sends them to the correct plugin. There are different plugins that allow Neutron to interact with SDN networks and Southbound interfaces like OpenFlow.

2.4 Network Function Virtualization

The Network Function Virtualization (NFV) paradigm inherits several key concepts initially proposed by network virtualization. Network virtualization is a technology that allows to use virtual entities to allow multiple isolated networks to share the same physical resources. A virtual network is composed of virtual machines interconnected in a specific way, establishing a virtual network topology. In virtual networks, specialized software like Open vSwitch [111] is in charge of switching packets among virtual machines. In this sense, Open vSwitch is a platform that allows generic virtual machines to offer services offered by a router. Platforms like Open vSwitch are approximations to the modern concept of NFV. The main objective of NFV is to use generic commodity hardware to create virtual machines and offer networking services with carrier class quality. Traditionally, dedicated hardware is used to offer network capabilities such as routing, firewalls, IDS, proxies and so on. In order to offer a complete network service, these devices had to be physically interconnected and the way they were connected reflected the service topology, also called service chain. NFV changes this paradigm by offering these network capabilities through virtual machines. In addition, to offer a complete network service, these virtual machines can be connected using virtual links, which has the advantage to be dynamic and can be modified on-demand following a change in the network service requirements. This concept is known as Service Function Chaining (SFC). By implementing network functions in generic hardware, the NFV
reduces capital and operational costs in telecommunication networks [113]. Finally, NFV allows to dynamically scale the resources of the entities offering the network services. This scaling, just like in cloud computing, is done by simply creating or deleting virtual machines that join previous ones to offer a given service.

Figure 2.4 presents the architecture proposed by the European Telecommunications Standards Institute (ETSI) for NFV. The ETSI is one of the standardization organizations that currently impulses NFV development. The proposed architecture has three main layers: The first layer is called the Network Function Virtualization Infrastructure (NFVI). This layer has the physical resources of the infrastructure, these resources are virtualized and offered as virtual machines with capacities of: computing, network and storage. This layer is managed by the NFVI manager. The second layer is the Virtual Network Function (VNF) layer. This layer posses the available network functions offered to the network applications. Network functions are simple functions offering a specific network activity such as “packet forwarding”. This layer also posses a VNF Manager, this manager handles the life cycle of the functions. One characteristic of the VNF is that they may be offered by third parties; to guarantee interoperability among the VNF they are described using an standard language. This standard language is known as Topology and Orchestration Specification for Cloud Applications (TOSCA). Finally, in the last layer are the components of Operation Support Systems (OSS) and Business Support Systems (BSS). These components use orchestrators to chain different VNF in complete network applications.

2.4.1 OPNFV - A Platform for NFV

The Open Platform for NFV (OPNFV), a project of the Linux Foundation, currently is the platform with the greatest support for NFV deployment. This project has the objective of creating a reference platform that allows to accelerate the development of services and applications on NFV with carrier class quality. OPNFV is a project that has the support of several well known organizations in telecommunications, like AT&T®, Cisco®, Dell®, EMC®, Ericsson®, HP®, Huawei®, IBM®, Intel®, Juniper Networks®, Nokia®, Redhat®, SUSE®, Vodafone®, ZTE®, VMWare®, Ubuntu®, among others [27].

OPNFV works closely with OpenStack [161], a platform developed for cloud computing. The relation between these two entities is through the Upstream Projects [27], collaborative projects developed between OPNFV and OpenStack to include new features in both platforms. Figure 2.5 shows the architecture of the current version of the OPNFV platform, called Brahmaputra [27]. The physical architecture of OPNFV is configured following the lines of the Pharaoh project. The goal of the Pharaoh project is to offer a standard testbed for OPNFV. As can be seen in the figure, OpenStack is the main component to manage the NFVI. In this way, the OpenStack modules are used to manage the virtual resources of computing, storage and network. In the current OPNFV platform version, it is possible to use different SDN controllers like ONOS, Neutron, and OpenDaylight to control the networking resources. Currently, the OPNFV team is developing tools to perform service orchestration, with service function chaining.

Service orchestration would be handled in two stages. In the first stage, an orchestrator receives the high level requests from the applications that are using the OPNFV platform. After interpreting these requests, the orchestrator looks up the current platform state. Punctually, it looks for the physical resources available, to create the necessary virtual machines
The NFVI infrastructure layer handles resources and their virtualization, the Virtual Network Functions layer chain resources to create VNFs, and offer them to the last layer, the Business and Operation Support Systems (OSS/BSS) \[27\] to orchestrate complete network services.

The present thesis is supported in the three main concepts defined before: Industrial Control Systems, Software Defined Networking, and Network Function Virtualization. The objective of the thesis is to explore the capabilities of SDN/NFV to enhance the security of Industrial Control Systems. In this context, the thesis will explore the dynamic reconfiguration capabilities brought by SDN in order to react to security incidents and the possibilities offered by NFV in order to visualize the ICS system. Finally, having a virtual ICS will also create access control challenges because the virtual nature of the system enables to create multiple instances of some ICS components that could be accessed by different actors with different set of roles.
Figure 2.5: OPNFV Architecture. The infrastructure layer, as specified in the Pharos Project, can be bare metal or virtual. The main OPNFV layer that offers the VNFs is based on OpenStack [49]. The Orchestration and Management capabilities are not part of OPNFV yet.
Chapter 3

Software Defined Networking and Network Function Virtualization to Enhance ICS Security

Many industries and critical infrastructures are monitored and controlled by Industrial Control Systems (ICS). These systems control power grids, water and waste-water management, oil systems, and manufacturing, among others. Because most of these systems are safety-critical, any potential cyber attack may cause significant physical as well as economic damages. Recent attacks to ICS like the power-grid attack in Ukraine \[98\] or the most recent alerts of hackings in U.S. nuclear plants \[112\] show the dire need for improving the security of Internet-connected industrial-control systems. Although several mechanisms to detect attacks have been developed, there is very little work on how to automatically respond to these alerts; most responses are manual, or are hardwired with a fixed response that cannot be configured.

Cybersecurity is a process consisting on (1) protecting, (2) detecting, and (3) responding to attacks \[76\]. Most of the literature on ICS security has focused on preventing and detecting attacks \[59\]; however, responding to attacks has received much less attention \[4, 59\]. In particular, most of the research chapter focusing on intrusion detection for control systems do not discuss what to do after an attack has been detected \[163\].

In this chapter we address this gap in the literature by designing intrusion response mechanisms that keep the control system operating safely while sustaining attacks. To design and implement our intrusion response architecture we use Software-Defined Networking (SDN), and cloud-based virtual infrastructures. While the use of SDN for protecting power systems has been considered before \[45, 137\], previous work has focused on proactive defenses, and not as a response to detected attacks. Similarly while cloud-based intrusion detection systems have been discussed before \[75\], the rules proposed are very basic, like the Quick Draw SCADA signatures by Digital Bond which only detect deviations of the ICS protocols, and do not leverage the semantics of the system \[66\], i.e., the understanding of the physical evolution of the process and how we can use that information to create our response policies. We provide more details to related work in Section \[3.4\].

In short, this chapter has the following highlights:

1. As far as we are aware, we are the first to design and implement an SDN-enabled
intrusion response architecture for ICS that leverages the semantics of the process (the physics of the process).

2. As far as we are aware, we are the first to propose and implement virtual incident response functions that take over critical components of the ICS when parts of it are under attack; e.g., a system simulator takes over sensors when the attacker has compromised the sensors in the system.

3. We extend a water system emulation built in mininet [94] (MiniCPS [10]) in many ways: (a) We implement a software-defined controller and switches to enable intrusion response, (b) we design and implement virtual incident response functions that we can use as a response to sensor and controller attacks, (c) we implement a field-layer network (the network between controllers and field devices) to enable the reconfiguration of compromised low-level devices, (d) we implement a larger control set consisting of three control loops (as opposed to the single control loop included in MiniCPS).

4. Our prototype is open-source and available to the community, so fellow ICS researchers can build their own solutions by extending and modifying our framework.

It is important to note that our intrusion-response architecture requires a good intrusion detection system. Developing and evaluating a good intrusion detection system is a very important area of research but it is also orthogonal to this chapter. In this chapter we focus on the less studied problem of what to do once an alert has been generated by an intrusion detection system, and we do not focus on the design of the intrusion detection algorithm. The particular intrusion detection system we use in this chapter is one we developed and presented at CCS 2016 for the same ICS we consider in this chapter; we refer readers interested in the technical details of the IDS to this previously published chapter [164].

The rest of the chapter is organized as follows: Section 3.1 presents the advantages of using SDN and NFV technology in ICS networks. Section 3.1.1 explains the ways in which SDN and NFV can enhance the security of ICS systems. Section 3.2 shows our proposed extensions of ICS, with SDN and NFV, to provide automatic incident responses to different attacks to ICS networks, and Section 3.3 shows our results. Section 3.4 discusses related work. Finally, Section 3.5 presents conclusions and future work.

3.1 Virtual Infrastructure for ICS Incident-Response

Investing in cybersecurity is one of the most pressing problems facing ICS asset owners. Installing extra-hardware for incident response in each control loop of an ICS might be a difficult security proposition to make, because large-scale industrial control plants have hundreds or even thousands of ICS equipment that exchange information to control physical processes. Procuring new incident-response equipment in each control loop, to respond to attacks that may rarely materialize would be an ineffective use of resources. Therefore to design cost-effective incident response procedures we need to leverage an infrastructure that can be repurposed to the specific attack event taking place.

Figure 3.1: Virtual Industrial Control Facility. Large industrial control facilities have hundreds and in some cases thousands of sensors, PLCs, and actuators in their infrastructure. To effectively respond to attacks, these systems require backup systems (extra controllers, process simulators, honeypots, etc.). Cloud computing can provide virtual network functions to cost-effectively supply these backup functions.

In this chapter we consider how a private-cloud architecture can help create a cost-effective investment for incident response in control systems. Figure 3.1 illustrates a control plant enhanced by the use of Cloud Computing technology: Software Defined Networking (SDN) and Network Function Virtualization (NFV). SDN and NFV make it possible to quickly detect and temporarily replace failing systems with virtual implementations of those systems. In our proposed architecture, IDS systems are scattered across the different field and supervisory networks present in the infrastructure. Whenever an IDS detects a possible intrusion, it notifies an SDN controller with a incident response policy. The SDN controller queries the incident response policy and decides what incident response launch. Then, the SDN controller sends a command to the cloud infrastructure provider to launch the respective Virtual Network Function (VNF) and reconfigures the network to use that VNF to mitigate the attack.

An industrial facility enhanced with cloud technology would reduce operational costs by only deploying and scaling virtual functions that are actually required when an incident occurs. In addition, management tasks would be simplified because of the reduction in the hardware boxes in the facility. Although some functions cannot be provided in a virtual manner, such as the actuators sending specific physical signals to the physical processes, many other functions could be provided by virtual machines, such as system-state estimators to replace backup sensors, virtual Programmable Logic Controllers (PLC), honeypots, routers,
proxies, and switches interconnecting facility equipment. The performance of these virtual machines would be critical to the security of certain operations, and although performance is out of the scope of this article, that problem can be tackled by using robust servers with Real Time Operating Systems at both host and virtual levels.

Software Defined Networking (SDN) and Network Function Virtualization (NFV) are technologies that improve network capabilities by enabling better management of network traffic flows, network visibility, and the deployment and control of network functions using software, instead of hardware-specific middleboxes. This programmability enables the development of new software services designed to meet a growing list of network requirements. In our proposed architecture, NFV supports addition of virtual functions and SDN enables dynamic reconfiguration of the network, in response to failures and incidents.

SDN enables dynamic network reconfiguration and rerouting using a programmatic approach. Several works have already used these SDN features to improve behavior of traditional information technology networks [155]. In addition, some research works are beginning to explore SDN advantages in cyber-physical systems [1, 13, 45, 96, 115]. These advantages originate from four key points of the SDN paradigm: i) SDN enables development and deployment of software applications for Commercial Off-the-Shelf generic servers (as opposed to the traditional use of proprietary hardware and middleboxes with limited software programmability), and ii) SDN separates the network control plane (how routers discover and maintain routing paths) from the data plane (how packets are forwarded between devices) in telecommunications. iii) SDN handles packets through the definition of flows. Flows are a combination of field values in the packets headers and can be protocol agnostic. Meaning that a simple SDN parser can be built to interact with industrial protocols without the need to modify the ICS equipment. Finally, this parser functionality can be implemented via software, which provides SDN controllers with all the flexibility necessary to interact with ICS protocols.

NFV enables the use of virtual machines to offer network functions that were traditionally offered by physical middleboxes. Traditional networks not only forward packets; they also process traffic through network functions like proxies, firewalls, intrusion protection systems, and so on. These functions have usually been implemented in expensive physical middleboxes; dedicated hardware devices inspecting, filtering, or manipulating network traffic. Developing and deploying new network services is difficult with this approach because that would require reprogramming and changes in configuration of these middleboxes. By using virtual machines, NFV enables the implementation of new network functions using software and general computing equipment, rather than dedicated hardware. These functions are called Virtual Network Functions (VNF). Additionally, NFV supports scaling services according to variable demand.

3.1.1 Use-Cases of SDN and NFV-enabled Incident Response in Supervisory and Field ICS Networks

In ICSs, a set of Programmable Logic Controllers (PLCs) control a physical process. PLCs use sensors to gather information about process state, make decisions based on that state, and send commands to actuators in order to change process behavior. A Supervisory Control and Data Acquisition (SCADA) Center, that communicates with the PLCs, collects data about the network and presents gathered information to human operators. The network
that communicates SCADA servers with remote controllers or terminal units is traditionally called a *Supervisory Network*.

Controllers like PLCs interface with sensors and actuators in the field. While traditionally this interface has been analog (4-20mA), the growing numbers of sensors and actuators as well as their increased intelligence and capabilities, has resulted in new *Field Networks* where the PLCs and controllers interface with a remote Input/Output boxes using new Ethernet-based industrial protocols.

Figures 3.2a and 3.2b show two alternatives to leverage SDN for ICS incident response: centralized and distributed respectively. In both cases an SDN controller dynamically reconfigures the network to answer to incidents. In a centralized option each LAN is connected using an SDN switch and one SDN controller makes all forwarding decisions; whenever a new flow arrives, the first packet of the flow is forwarded to the central SDN controller and it installs a forwarding rule in the SDN switch. In addition, an IDS receives a copy of that packet to maintain information about network state and also monitor a portion of the traffic exchanged in each control loop LAN. A centralized SDN controller does not require distributed databases to store network state. Nevertheless, this approach has the disadvantage that all new flows should be forwarded to the central SDN controller increasing its load.

In a distributed approach each LAN network is connected to a different SDN Controller instance. The first packet of a new flow is forwarded to the corresponding controller instance that decides how to handle that flow. The state of the network is kept using a distributed database but consistency should be addressed in order to avoid network loops or black holes. Currently, some SDN Controllers, like ONOS, offer such possibility. Using this approach an IDS instance could be present in each LAN. The distributed database may be shared by the SDN controller and IDS instances to detect wide-area attacks. Finally, SDN controller instances need an SDN Application Manager to enable operators to activate/deactivate specific SDN applications or obtain snapshots of the current network state. Such kind of manager shares many features with SCADA systems present in ICS networks. This approach creates more resilient systems by avoiding central points of failure and having faster response times because all forwarding decisions are locally made. The disadvantage of this approach is a higher complexity in the SDN controller and IDS implementations.

The integration of SDN and NFV to ICSs gives origin to a *ICS-SDN-NFV Architecture*. In this architecture, is possible to develop automated incident responses and the deployment of different defenses without modifying legacy systems, industrial network protocols, or the logic behavior of traditional ICS elements. In particular, SDN and NFV make it possible to dynamically start and stop components that implement different detection techniques, like network monitors and IDSs, and different defense techniques. They can also dynamically scale VNF resources up or down [100, 179], depending on the attack scenarios. Figure 3.3 shows our proposal to improve industrial-control system with SDN and NFV. This architecture extends traditional ICS architectures by adding an IDS at the field level, an SDN Controller at the supervisory level, and a virtualization server to store/run the virtual incident response functions. Taormina et. al. [157] identify different scenarios of potentially compromised components in ICS systems. Using this classification, we have defined some potential incident-response use-cases:

1. When the IDS detects an attack and informs the SDN controller, the controller can
Figure 3.2: Alternatives for enhancing industrial control systems with software defined networking

(a) ICS System with centralized SDN control. In this alternative, a unique SDN controller node is deployed in the SCADA center, this enables to use simple SDN controller implementations, but increases the latency between incident detection and automatic network reconfiguration.

(b) ICS System with distributed SDN control. In this alternative, each control loop is controlled by a distributed SDN controller instance. This enables incident faster response times, but increases the complexity of the SDN controller logic.

reroute the attacker’s traffic to an ICS honeypot. This way we can continue monitoring the attacker to get intel on what the attacker is trying to do to the system (by keeping an attack active we can obtain more information about the attacker’s targets and methods), while at the same time isolating and protecting the real process. The ICS honeypot needs to provide fine-grained emulation of the devices and the physical process under control in order to deceive the attacker.

2. When the IDS detects an attack on the sensor, the SDN controller can change the forwarding rules of the SDN switch so that sensor packets are dropped, instantiate temporarily a plant simulator (a simulator of the physical process under control), and add rules in the SDN switch to communicate that plant simulator with the PLC. The simulator can then estimate the expected state of the process based on the control values as well as potentially other non-compromised sensors. The controller can then use the estimate to compute its next control action. Note that this solution must always be temporary until we find a way to restore the true sensor values: if we continue using the simulation instead of the sensor for extended periods of time then we will be controlling the system using open-loop control (instead of closed-loop control) which is a long-term problem.

3. Another incident response approach is the transfer of services from the compromised device to a redundant device. This approach has been traditionally used in the fault tolerance community, but SDN and NFVs gives us the flexibility of not investing a priori on capital equipment, and using the resources of a virtualized device only on
By using SDN and NFV technology we can create more resilient industrial control systems; systems that proactively react to detected events, avoid central points of failure, and scale resources up and down adjusting to actual network state.

3.2 Implementing SDN and NFV in Control Systems

In this section we leverage an example of a real-world ICS Water Treatment Process (SWaT testbed), and show how to implement incident-response functions with the help of SDN and virtual functions. In particular, we extend MiniCPS [10], a previous effort to implement this example in a co-simulation/emulation environment (modeling the physical system as well as emulating the networks and devices), and we implement an SDN and NFV environment to respond automatically to attacks. We focus our attention on the first three stages of the process: raw water storage, pre-treatment, and ultra-filtration feed water tank. We develop a virtual representation of the physical process that includes the communication network and devices. Our implementation is illustrated in Figure 3.4.

We selected this process for the following reasons: i) Water processes tend to have slow response times which are ideal to test new defense and incident response mechanisms. In the
future, we could extend this architecture to protect other type of industrial control processes with faster response times. ii) This ICS had a virtual implementation through MiniCPS which is Open Source, enabling us to extend their implementation and testing our incident response architecture.

**Attacker Model.** We assume the water treatment plant that is being attacked by someone with a deep knowledge of the process and able to compromise the sensors or the PLCs, and then sending wrong sensor readings to the PLCs or wrong control actions to the actuators. We assume that the attacker does not compromise the SDN controller, switches, actuators, or the IDS present in the topology.

### 3.2.1 Description of the Process

This process is controlled by three PLCs: PLC101, PLC201, and PLC301. These PLCs control the water level in two tanks and also the pH of the water in the system. The two tanks, Tank_101, and Tank_301, store raw and treated water, respectively. Tank_101 is the main water buffer and PLC101 controls its inlet and outlet flows by managing the ON/OFF commands of the valve MV101 and the pump P101. Both tanks are connected through a pipe that possesses sensors to monitor the pH of the water to determine whether PLC 201 needs to activate (ON/OFF) a chemical dosing pump (P201) to maintain the quality of the water.
within desirable limits. Since the tanks are connected, the inlet and outlet flows of Tank 301 are controlled by pumps P101 and P301 respectively. To maintain a desired height of water in both tanks—i.e., between Low level (lowL) and High level (highL)—PLC 101 receives water level readings from sensor LIT101, and PLC 301 from LIT301.

The following equations summarize the control logic (1 = ON, 0 = OFF) at PLC 101:

\[
MV_{101} = \begin{cases} 
1 & \text{if } LIT_{101} < lowL \\
0 & \text{if } LIT_{101} > highL
\end{cases}
\]

\[
P_{101} = \begin{cases} 
1 & \text{if } LIT_{301} < lowL \\
0 & \text{if } LIT_{301} > highL
\end{cases}
\]

and the control logic at PLC 301:

\[
P_{301} = \begin{cases} 
0 & \text{if } LIT_{301} < lowL \\
1 & \text{if } LIT_{301} > highL
\end{cases}
\]

(3.1)

Notice that pump P101 is controlled by PLC 101 but it requires readings from the water level sensor LIT301. Since sensor LIT301 is only available to PLC 301, thus PLC 301 shares water level readings with PLC 101 through the supervisory network.

In the physical process simulation, the dynamic model that was used for describing the water level for each tank, consists on the water volume change at each time instant:

\[
\frac{dV}{dt} = V_{in} - V_{out}
\]

Since the area of the tank \(A\) is constant, the volume is:

\[
V = A \cdot LIT_{i01}
\]

leading to the following discrete time equation:

\[
LIT_{i01}(T(k + 1)) = LIT_{i01}(Tk) + \frac{T \cdot A}{\triangledown} \left( u_{in,i} - u_{out,i} \right).
\]

(3.2)

where \(i = \{1, 3\}\), \(T\) is the sampling period and \(k\) is the sampling instant, \(A\) is the tank base area, \(\triangledown(m^3/h)\) is the constant volume of water that comes from the valve or that is pumped. \(u_{in}\) and \(u_{out}\) are control signals that define the state of the actuators: \(u_{in,1} = MV_{101}, u_{out,1} = u_{in,2} = P_{101}, u_{out,2} = P_{301}\) depend on the control commands sent by the PLCs. For instance, if \(MV_{101} = 1, P_{101} = 1\) and \(P_{301} = 0\), then \(LIT_{101}\) will remain constant while \(LIT_{301}\) will increase.

### 3.2.2 Implementing ICS Networks

In addition to simulating the physical process of the system, we need to have a high-fidelity emulation of the networks in industrial systems. In this section we present our extension of MiniCPS [10] to implement a fine-grained incident response co-simulation for the previously described water treatment system. MiniCPS uses Mininet [94], a light virtualization environment tailored for SDN experiments and emulation. MiniCPS extended Mininet to support EtherNet/IP, an industrial network protocol commonly used in ICS for communications between the supervisory and field networks. A summary of our extensions to MiniCPS can be seen in Table 3.1.

MiniCPS developers created a testing scenario with three PLCs to control water tank levels. A script runs an implementation of the difference equations in Equations (3.2) and (3.1), that simulate the physical process as a mechanism to generate water level readings. The three PLCs directly interact with the physical process to obtain the readings, that is, they read the values from the variables that store them and apply control actions to the valves.
<table>
<thead>
<tr>
<th>MiniCPS</th>
<th>Our System (ICS-SDN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only receives and sends data from one sensor and actuator.</td>
<td>Implements one regular blocking MiniCPS interface and additional sockets to receive or send messages to various entities.</td>
</tr>
<tr>
<td>PLCs directly affect physical process and read directly from it, as such the network between PLCs and field devices is missing in MiniCPS.</td>
<td>Sensors and actuators are included in the topology. The physical world is only changed by actuators. We emulate the network between PLCs, actuators, and sensors.</td>
</tr>
<tr>
<td>Only implements one control loop.</td>
<td>We implement three control loops, thereby expanding three times the complexity of the physical process under consideration.</td>
</tr>
<tr>
<td>Does not use SDN or virtual functions. It also does not consider an incident-response policy.</td>
<td>We implement an intrusion response architecture with SDN and virtual functions that take over the operation of compromised devices.</td>
</tr>
</tbody>
</table>

Table 3.1: Differences between the original MiniCPS and our proposed environment.

and pumps to control the water levels. MiniCPS models the communication between PLCs, it does not implement however, the field network (the communications between PLCs and sensor and actuators).

In this chapter we extend MiniCPS to include an IDS and an SDN/NFV-based incident response mechanism. We also build a full topology of the system with its three control loops in the field network. Figures 3.5a and 3.5b show the topologies we implemented. Figure 3.4 implements an SDN in the field network by having a single switch per loop, combining supervisory and field networks, and Figure 3.5b implements an SDN in the field network by having the PLC act as a router with two interfaces, one to the supervisory network (top) and another interface dedicated to the field network (bottom). While we implemented both solutions, we use the latter implementation in our experiments; we argue that this topology represents more accurately typical ICS topologies where the PLC is the bridge between the field and supervisory network.

In our implementation, each control loop is represented by a LAN, while sensors, actuators, and IDS are represented by hosts. A PLC acts as the router of each of its LAN and the PLCs are connected using Mininet switches. An SDN controller manages the network and a SCADA system periodically receives reports about system state. Finally, we run a script that evaluates one differential equation for each plant in the system. Namely, the two water tanks

---

2 Many industrial control systems have a high availability requirement that puts an upper limit in the maximum accepted tolerance. Usually, this ranges in a few milliseconds and in addition the system requires to have multiple backup equipment in order to ensure system resilience. For this reason, Field Networks use custom topologies and communication protocols such as Rockwell's Ethernet Ring (TRD). TRD is a ring topology that ensures fast convergence when a single link or device fails; for example, a 50 device topology can converge in less than 3ms. Besides of this, TRD enables communications between devices with a period of 400 us. Another advantage of TRD include a reduced wiring, which simplifies overall management and even electrical noise from control equipment. Finally, a token topology is able provides the network with more distance because each daisy chain drop is another Ethernet link with the full stretch of the copper cable allowed between each node as opposed to a traditional center point star configuration using external switches.
(a) One way to extend SDN to field networking is to have an SDN switch that is used by PLCs, sensors and actuators, as well as the SCADA servers.

(b) Another way to use SDN in field networks is to have a PLC with two interfaces, one interface to the supervisory network, and another to the field network.

Figure 3.5: Alternatives to enhance ICS field networks with SDN. Both options are viable and the implementation will depend on the actual physical constraints of the system (e.g., how far are field devices from switches? Do we need complete separation of field and supervisory networks? etc.) In addition, we implemented an intrusion detection system depicted as IDS 101.

and the pH level of the water between the tanks.

Another feature we added to the system is that hosts that can now perform concurrent actions. The original MiniCPS library that implements EtherNet/IP uses blocking sockets causing that hosts can only interact with one host at the same time. Nevertheless, in real control systems, a PLC receives messages in a non-blocking way and can concurrently send control actions to various actuators. To overcome this problem in our implementation, each element in the control system implements at least one blocking EtherNet/IP socket, and also implements regular Python asynchronous sockets to send non-blocking messages. In both traditional and our proposed ICS architecture, all ICS elements have the same sample period and send their information each time a sample period time is executed.

In our scenario, an IDS in the first loop performs Deep Packet Inspection (DPI) on network packets. DPI is the detection process of inspecting not only the protocol headers of a packet, but also its payload content. Details of our IDS implementation can be found in our previous work [164]. Our prototype and source code is available in Github[^3].

### 3.2.3 SDN Controller Application

We used the POX SDN Controller in our experiments. Our SDN Controller application extends the learning switch application originally developed by the POX developers to include a topology map of the ICS elements and to create a socket for each IDS present in the network.

Our application keeps a topology map of each of the SDN switches to relate ports of the ICS elements and the IDS. In addition, our application creates sockets to listen to IDS notifications about compromised elements. Detecting which element has been compromised is out of the scope of this thesis.

### 3.2.4 Design of our Physics-Based IDS

Two IDS are deployed in the topology, $IDS_{101}$ and $IDS_{301}$. $IDS_{101}$ runs code to estimate system state (e.g., water level) and compares it with sensor readings in order to detect malicious measurements from a compromised sensor. The estimation uses a Luenberger State Observer (or a Kalman filter), a tool to predict the system behavior when there is a an approximate model of the physical process [162]. In this case the observer runs the following equation to estimate tank water level:

$$\hat{x}_i(T[k + 1]) = \hat{x}_i(T[k]) + b U_i(T[k]) + Q_i[LITi01(T[k]) - \hat{x}_i(T[k])]$$  \hspace{1cm} (3.3)

where $b = \frac{\tau^2}{A}$, $U_i(T[k]) = u_{in,i} - u_{out,i}$ and $Q_i \in \mathbb{R}$ is the observer gain, which is greater than zero if there are no alarms, and zero if we are responding to an alarm. Let $r_i = |LITi01(T[k]) - \hat{x}_i(T[k])|$ be the residuals that measure the difference between estimations and sensor readings. If $r_i > \tau_i$, the IDS reports a malicious reading from sensor $i$. Although this could also indicate a faulty sensor, in both cases, our IDS would report an attack.

### 3.3 Experiments and Use-Cases

We illustrate the performance of SDN and NFV-enabled incident response functions by considering two experiments, one where a sensor is compromised and the incident response function must replace the sensor readings with a simulation of the process, another experiment where a controller is compromised and the incident response function must replace the controller with a cloud-enabled backup controller. We also design a final use-case where the SCADA system is compromised and the incident response function replaces the real process with a honeypot simulating the physical process.

All the experiments were run on a machine with an Intel(R) Core(TM) i7-4770, (3.40GHz) processor, and 16GB of RAM. This machine was running Ubuntu 16.04 and VirtualBox 6.0.12. We used an Ubuntu virtual machine image with mininet already installed[4] We gave to this virtual machine 4 processors and 2048GB of RAM. In this virtual machine image we installed MiniCPS[6] and applied our extensions.

#### 3.3.1 Responding to Compromised Sensors

For our first experiment, recall that the system must keep the water level of the two tanks between 0.5 and 0.8 of their capacity. Our experiment tests whether the system detects and reacts to an attack that compromises sensor $LIT101$ to send false readings to PLC 101, this

---

[4] Available at: [https://www.virtualbox.org/wiki/Linux_Downloads](https://www.virtualbox.org/wiki/Linux_Downloads)


sensor will always report a water level 0.3 below the real value. This means that PLC 101 will never see a water level above 0.5, causing PLC 101 to keep valve MV101 always open and pump P101 always closed; the first tank will overflow and the second tank will never be filled.

Figure 3.6 illustrates the first incident response system:

1. The IDS instance always receives a copy of the information reported by the sensor $LIT_{101}$.

2. The IDS uses our previous work [164] to identify differences between reported sensor readings and estimated values.

3. If the difference is greater than a given threshold, the IDS notifies the SDN controller that sensor $LIT_{101}$ has been compromised.

4. The SDN applications looks up in its topology map to obtain the input port of the compromised sensor and of the plant estimator.

5. The SDN controller modifies the flow table of the SDN switch to drop packets from the compromised sensor port and forward the messages from the estimator.

Figure 3.6: Enhanced industrial-control system with SDN. The IDS notifies the SDN controller when it detects an attack on the sensor. The controller looks up on the topology to check which in port the compromised sensor uses; then, it creates a flow entry in the switch to drop packets from that port and forward messages from the plant estimator port.

Figure 3.7 illustrates the SDN code and the impact on the flow table at the switch to accomplish this incident response. In the first step, the controller activates a flag to indicate
### POX Application Instructions

```python
if message['Variable'] == 'Switch_flow':
    self.compromised_sensor = True
    self.switch_flow()
```

### Switch Flowtable Result

<table>
<thead>
<tr>
<th>nw_src</th>
<th>in_port</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.10</td>
<td>4</td>
<td>output:2</td>
</tr>
</tbody>
</table>

```python
def switch_flow(self):
    msg = of.ofp_flow_mod(command=of.OFPFC_DELETE)
    nw_src = 192.168.1.10, in_port = 4
```

```python
def _handle_PacketIn(self, event):
    if (in_port == 4) and (nw_src == "192.168.1.10") and (self.compromised_sensor):
        return
    port = self.macToPort(packet.dst)
    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet, event.port)
    action = of.ofp_action_output(port=port)
```

### Switch Flowtable Result

<table>
<thead>
<tr>
<th>nw_src</th>
<th>in_port</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.10</td>
<td>4</td>
<td>output:2</td>
</tr>
</tbody>
</table>

**Figure 3.7:** SDN controller application. When a compromised sensor is detected, the POX Application issues instructions to adjust behavior. The left column shows excerpts of the application code and the left one shows their impact on the switch flow table.

A compromised sensor, after receiving a notification from the IDS. Second, the controller issues an `OFPFC_DELETE` command to delete the flow entry associated to the compromised sensor. By removing the entry, the next packet sent by the malicious sensor will cause a `packet_in` event; this event is used to create the flow rule that will drop all packets from the malicious sensor (including the one that caused the event). Deleting the flow with this method is important, because otherwise, the malicious sensor could keep sending packets until the original flow entry expires. Finally, when the IDS informs the controller about a compromised sensor, it also starts sending estimated values to the PLC.

**Figure 3.8** shows the results of the experiment. When the water tank level reaches 0.5, the malicious sensor starts sending wrong readings. The IDS instance detects such behavior and notifies the SDN controller. Upon reception of this notification, the SDN controller creates flow entries in the SDN switches to start dropping the readings from the malicious LIT101 sensor and start forwarding the estimated values. The figure shows that the behavior of the plant under attack is similar to the scenario with no attack, indicating that the SDN incident response system increases the resiliency of the system.

#### 3.3.2 Responding to a Compromised PLC

For our second use-case, *IDS301* receives all the control commands sent by *PLC301*. When the IDS detects that a command does not correspond to the current system state, the SDN controller is notified. In this case, the SDN application issues a command to the switches to
Figure 3.8: Attack scenario with compromised sensor. Approximately 50 minutes after operation, sensor \textit{LIT101} is compromised and it starts sending a lower level value. When no defense is present, PLC 101 never activates pump \textit{P101} and \textit{Tank} \_\textit{101} overflows. When the defense is present, the IDS running a Luenberger observer identifies the malicious behavior and notifies the SDN controller to drop packets from the sensor. The behavior of the plant follows closely normal operation.

We drop all the packets from \textit{PLC301} and forward the information to a backup virtual PLC. We argue that an attack on this PLC in the current topology is much worse than an attack on the sensor \textit{LIT101} because the PLC not only reports information but sends order to the pump \textit{P301}. Recall that \textit{PLC301} receives the tank 2 water level from sensor \textit{LIT301}; Then, \textit{PLC301} relies this information to the \textit{PLC101} so it can control the pump \textit{P101}.

We consider that our approach could greatly help to increase ICS systems resiliency because the logic of the ICS elements does not need to be modified, and SDN handles the dynamic reconfiguration of the network. Although our application is simple, this model can be extended to include different incident response policies for different types of attacks. For example, \textit{PLC101} not only receives information from the sensor \textit{LIT301} and acts on the pump \textit{P101} and the valve \textit{MV101}, it also receives information from \textit{PLC301}. An attack on this PLC would require a more complex response strategy. If we include a backup function for \textit{PLC101} we would also have to modify the logic of \textit{PLC301} to start sending its information to a different IP. Since our aim is to avoid the modification of the ICS equipment, we would have to modify the packets sent to the \textit{PLC301} in order to reach the backup \textit{PLC101} and not the compromised PLC.

In our experiment, the attacker compromises \textit{PLC301} and when the water level of tank 2
reaches 0.5m it stops sending information about \textit{LIT301} to the \textit{PLC101} and maintains closed the pump \textit{P301}. This would have the consequences of overflowing the water tank 2. This overflow is caused by two reasons: i) Since \textit{PLC101} would not receive information about tank 2 water level, it would never close the pump \textit{P101}, ii) Since the pump \textit{P301} is always closed, there is no way to drain the water out of tank 2. Figure 3.10 shows the results of the experiment. In this figure, the water level of the tank 2 is presented. The blue line, show the normal behavior of the water tank level when no attack is present. The red dotted line, indicates the water tank level when the attack is present. When the water tank level reaches 0.5, the malicious PLC stops sending readings and closes the pump \textit{P301}. This causes the tank 2 to quickly overflow. The blue dotted line presents the behavior of water tank 2 when the IDS is present. The IDS constantly receives the action control calculated by \textit{PLC301} and compares it with the expected action control. When it detects a difference, it notifies the SDN controller that a possible attack on \textit{PLC301} is present. Upon reception of this notification, the SDN controller creates flow entries in the SDN switches to start dropping the readings from the malicious \textit{PLC301} and start forwarding the estimated values by the IDS. The figure shows that the behavior of the plant under attack is similar to the scenario with no attack, indicating that the SDN incident response system increases the resiliency of the system.
Figure 3.10: Attack scenario with PLC301 compromised. Approximately 16 minutes after operation, PLC PLC301 is compromised and stops sending water level information to PLC101 and maintains the pump P301 closed. This causes Tank 301 to overflow. When the SDN defense is present, the IDS installed in the third control loop receives action controls sent by the PLC301 and compares them with the estimated ones. Upon detection of the attack, the IDS instructs the SDN Controller to drop packets from the compromised PLC and forward the estimations of IDS301. The behavior of the plant follows closely normal operation.

3.3.3 Responding to an Attempted SCADA Compromise

In the final case we consider, an external attacker probes the supervisory network in order to try to compromise an equipment in the SCADA system. Such attack could have a huge impact in the system operation because the attacker would be able to change the set point of different control systems. Figure 3.11 illustrates this situation. In the figure, an external attackers sends a probing command to the SCADA equipment in the supervisory network, for example some of the traditional Nmap packets used to check for open ports in a server. Recall that this type of well-known packet passes through the SDN router 0 and could be reported to the SDN Controller and the Field Network IDS. Upon identification of this probing behavior, the IDS notifies the SDN controller to create a virtual honeypot. This virtual honeypot is an exact virtual representation of the control system present in the field network; specially tailored virtual response functions are available at the Cloud Infrastructure, they run the same software version the actual ICS equipment runs and are able to communicate through the same protocols present in the field network. The SDN controller needs to check the actual control network topology in order to select the appropriate virtual functions and build the
honeypot network. This process is very similar to the Service Chaining process described in Network Function Virtualization and basically includes three steps: i) Create virtual machine instances offering the ICS equipment software, ii) Use SDN to create forwarding rules that represent the network topology, iii) Configure the virtual machine state, according to the current control topology state. Steps i and ii are typical of NFV service chaining, while step iii is unique to ICS systems. Afterwards, the attacker probing the network will be probing the virtual honeypot. The use of SDN prevents router 0 from having to rewrite all the packet headers that the attacker sends to the network. In the creation of the honeypot, the virtual machines would be configured with the same IPs as the real ICS equipment. In this way, the SDN controller creates a forwarding rule at router 0 to forward all traffic incoming from the attacker IP to the honeypot network, instead of the real network. If the attacker were able to compromise all the elements in the honeypot, his activities could be interrupted by installing a dropping rule at the router 0 for all the traffic coming from the attacker, the virtual machines should be shut down and their disks put in quarantine for further analysis or simply deleted. The use of a Cloud infrastructure and SDN eases honeypot management and protects real infrastructure because the attacker does not compromise real equipment. It is important to notice that a completely virtualized infrastructure could ease this use case. A complete virtualized ICS system is discussed in chapter 6.2.

Figure 3.11: Attacker probing the SCADA. The SDN controller, after receiving a notification from the IDS, creates a virtual honeypot and reroutes all traffic to it.
3.3.4 Performance

To quantify the response time of our system we measured the time it takes the SDN Controller to modify the flow table at the SDN switch, in order to complete the rerouting procedure. Specifically, we measured the time between the generation of the alert at the IDS and having the switch flow table modified to reroute traffic. We run this experiment 10 times. The result is of $2.3 \pm 0.24$ ms. Although these results were obtained in an emulation environment, we argue that they do not fall too far from a real environment setup. The only delays that our emulation does not consider are those created by network congestion and the medium access control mechanisms, such as those present in Ethernet. Nevertheless, in a real ICS system, the network could be provided with enough resources to maintain the network latency within acceptable limits.

These delays are reasonable for physical systems with relatively slow dynamics, like the water tank in our example, or the pH level changes from PLC201. There are a large amount of physical processes with slow dynamics involving those with water, gas, oil extraction, treatment and distribution, in addition to several chemical processes. While we claim that our solution can be applied to a large set of these slow dynamic ICS, care must be taken in other environments like in the power grid where low latency is required. In future work we will explore the applicability of our success story in ICS to power systems and evaluate the timing and delay constraints and requirements.

3.3.5 Responding to False Alarms

False alarms from the intrusion detection system may cause our defenses to activate when there is no need. In this case we may need to consider how the physical process behaves under our response, to see if there are potential problems. Recall that we have implemented responses for attacks that identify the PLC or the sensors as the compromised devices. The impact on the physical process from a false alarm identifying the controller will be minimal because the virtual function is a redundant control system; however, the impact to the physical process from a false alarm claiming that a sensor has been compromised will be more severe.

Our original implementation did not raise any false alarms because the water filling process is very predictable, and our models are an accurate representation of the system behavior. Therefore in order to create false alarms we had to add perturbations at the intake of the first water tank. We added a random Gaussian perturbation (the random value was truncated to prevent unrealistic values) to the amount of water entering the first tank of the process. The variations in the water level in the tank therefore would be random and could generate false alarms.

Figure 3.12 plot shows the difference between the control under normal conditions, and the control that leverages the incident-response virtual function. While there is a difference (performance degradation), the control with the incident response function manages to follow the real operation of the system, even when ignoring the data from the sensors. The reason for this is that the estimator looks at the control actions going to the valve and can then revise the expected water level in the tank and how the control actions will change them.

Our incident-response function is not applicable to all situations. The more uncertainty we have on the process, or the less accurate our estimates are, then the more damaging the
Figure 3.12: Behavior of the water level in a tank with a random perturbation. The plot shows the difference between the control under normal conditions, and the control that leverages the incident-response virtual function.

Impact of a false alarm will be. Figure 3.13 shows the average deviation over unit of time between the water level in the tank when we operate the tank with the real sensor values, and when we respond to a false alarm when the perturbation noise increases in magnitude. As we can see, the more uncertainty there is in the system, the more damaging the response to a false alarm will be. As such our incident response strategy for compromised sensors needs to be applied with caution, and only when we have confidence in the accuracy of our estimators. Simulations like the one in Figure 3.13 can help identify the breaking point between operating the system safely, even with false alarms, and causing potentially more damage to the system than an attack.

Another word of caution is that our sensor defense is meant to be used for a short period of time while operators have the time to diagnose the system and eliminate the threat or start a safety shutdown. The estimates will (on average) diverge over time from the real-world operation, and therefore an attacker or a false alarm can eventually cause significant problems to the system. We will continue working on defenses that minimize these types of events in the future. In addition, the IDS system we used in this paper is not able to identify which element has been compromised. We need an additional algorithm in order to
determine if an actuator, sensor, or PLC has been compromised. Such algorithm is out of the scope of the present thesis. In addition, a more complex intrusion detection system must be used if more than one component is compromised at the time [163].

### 3.4 Related Work

SDN and NFV are playing a growing role in proposals to protect general IT networks. The Authors in [34, 74, 79] propose using SDN and NFV to build and present a controlled view of a network, thus limiting attack capabilities. In [34], the SDN controller and switches act as a proxy, masking real IP and MAC addresses and other relevant properties of the hosts in a network, and creating for every host a different network view. Hirata et al. in [74] use SDN and virtual machines to create honeypots that protect web servers. Although this work was published before the NFV term became widespread, it combines characteristics of SDN and NFV to create realistic honeypots for web servers. OpenFlow Random Host Mutation [79] proposes a Moving Target Defense technique using SDN and NFV to invalidate information gathered by scanners. Guofei et al. [149] discuss the security benefits that SDN can bring to communication networks. Dynamic network control, wide network visibility, network programmability and a simplified data plane are the main contributions that SDN...
can provide to network security. The use cases presented in this article and our SDN-Cloud based honeypot benefit from such characteristics of SDN.

Previous work has mentioned the benefits in the use of SDN for Smart Grids and Industrial-Control Systems [2, 45]. Hui Lin et al in [9] use a distributed intrusion detection system to detect false data injection in electrical grids. In this proposal SDN is used to gather the information processed by the agents.

Piggin performs analyses some benefits of Cloud Computing for SCADA control systems in [134]. The author presents two models for Cloud computing adoption by SCADA systems, Partially Cloud hosted and Fully Cloud hosted. In the first model, applications pushed data to the cloud to perform data analytics. In the second model, command and control traffic is downloaded to the remotely controlled systems. We argue that in addition to those models, the Cloud can provide incident response functions that benefit from the cloud dynamic scaling properties.

Interest in honeypots for control systems is also increasing. A recent poster [95] presented a high interactive honeypot based on a simulator for the Siemens S7-300 series PLCs. Although the prototype resists OS fingerprinting from Nmap, a knowledgeable adversary can identify the simulator as a honeypot. We argue that instead of using simulations, we need to build specially tailored virtual machines that behave exactly as the ICS equipment. Although the authors in [165] do not use SDN, they propose an approach to automatically configure and run virtual honeypots. The proposal uses Ettercap to scan a particular network and collect data about available services and their configurations. Afterwards, it uses Honeyd (www.honeyd.org) and the collected data to dynamically configure virtual honeypots adjusting them according to network changes.

As stated in the introduction, cybersecurity is a process consisting on (1) protecting, (2) detecting, and (3) responding to attacks [76]. Most of the literature we have discussed here for ICS security focuses on preventing and detecting attacks [59], but a major gap in the literature is how to respond to alerts generated by the IDS [4, 59]. In particular, most of the research chapter focusing on intrusion detection for control systems do not discuss what to do after an attack has been detected [163]. One of our goals in this chapter is to contribute to the literature on attack-response for ICS.

### 3.5 Conclusions and Future Work

This chapter proposes using cloud technology, i.e. SDN and NFV, to develop more resilient industrial control systems. We implemented an open source prototype, that extends MiniCPS, to use SDN to enable programming of automated incident responses to several attacks. Our open-source implementation is available in GitHub (as described in Section 3.2.2).

As a proof-of-concept we tested three use cases where attackers compromised different ICS elements; a sensor, a PLC, and the SCADA; and evaluated the automatic programmed reaction of the system. The behavior of the ICS system in these cases, with the proposed defense, is close to the behavior under normal operation conditions.

SDN enables using programs to dynamically change behavior of cyber-physical networks in response to attacks. This change does not need modifications in the logic of the network elements or their software stack (and therefore it can be used in legacy systems), only needs SDN-enabled network elements. SDN also enables on demand launching of Virtual Network
Functions to offer traditional CPS control and security services. Although we implemented our proof of concept on a water distribution system emulation, our architecture could be applied to different industrial processes if the model of the system is known. In these cases, the response time of the system should be considered in order to ensure that the system responds within the response time of the industrial process. If real time guarantees need to be addressed the works of Herlich et. al could be considered [72].

We plan to extend our work in multiple directions. One key research direction is the use of centralized vs. distributed implementations of incident response. For example, figures 3.2a and 3.2b show two alternatives to leverage SDN for ICS incident response: centralized and distributed respectively. In both cases an SDN controller dynamically reconfigures the network to answer to incidents. In a centralized option each LAN is connected using an SDN switch and one SDN controller makes all forwarding decisions; whenever a new flow arrives, the first packet of the flow is forwarded to the central SDN controller and it installs a forwarding rule in the SDN switch. In addition, an IDS receives a copy of that packet to maintain information about network state and also monitor a portion of the traffic exchanged in each control loop LAN. A centralized SDN controller does not require distributed databases to store network state. Nevertheless, this approach has the disadvantage that all new flows should be forwarded to the central SDN controller increasing its load. POX and OpenDayLight Controllers use a centralized approach.

In a distributed approach each LAN network is connected to a different SDN Controller instance. The first packet of a new flow is forwarded to the corresponding controller instance that decides how to handle that flow. The state of the network is kept using a distributed database but consistency should be addressed in order to avoid network loops or black holes. Currently, some SDN Controllers, like ONOS, offer such possibility. Using this approach an IDS instance could be present in each LAN. The distributed database may be shared by the SDN controller and IDS instances to detect wide-area attacks. Finally, SDN controller instances need an SDN Application Manager to enable operators to activate/deactivate specific SDN applications or obtain snapshots of the current network state. Such kind of manager shares many features with SCADA systems present in ICS networks. This approach creates more resilient systems by avoiding central points of failure and having faster response times because all forwarding decisions are locally made. The disadvantage of this approach is a higher complexity in the SDN controller and IDS implementations. We plan to continue this line of research in future work.

3.6 Acknowledgments

This work is partially supported by the U.S. Air Force Office of Scientific Research under award number FA9550-17-1-0135, the U.S. Department of Commerce by NIST Award 70NANB17H282 and by the Colombian Administrative Department of Science, Technology, and Innovation (Colciencias).
Chapter 4

A Virtual Environment for Industrial Control Systems: A Nonlinear Use-Case in Attack Detection, Identification, and Response

Developing high-fidelity emulation environments for industrial control systems is an essential tool for helping us understand the risks of attacks, to evaluate new defenses, and to deploy honeypots so we can understand better the attackers of these systems. A challenge for any virtual testbed, is to create an environment that behaves as close as possible to a real industrial control network as well as having a high-fidelity evolution of the physical system under control.

There is a lot of interest for designing virtual testbeds for industrial control systems [7, 53, 83, 84]. Some testbeds simulate the physical process and add real hardware to the system (hardware-in-the-loop testbeds), and some are complete simulations (simulating the process and the equipment). Nevertheless, to be able to use these testbeds to design network security products, or to allow attackers to connect to them and interact with them through network penetration tools, we need to provide an emulated network. In this chapter we focus on the use of a virtualization environment to emulate the networks of an Industrial Control system (ICS) controlling a nonlinear physical plant and also in implementing security mechanisms to detect a false-data injection attack and localize where this attack is coming from.

Our virtualization environment represents networks in a realistic way since the virtual nodes have network stacks that represent the corresponding stacks of real devices in an ICS. By considering this stack, our co-simulation environment has higher fidelity of the behavior of a real system than purely simulation-based testbeds.

The rest of the chapter is organized as follows: Section 4.1 presents related work. Section 4.2 presents our model of Network Control Systems, attacks, and defense mechanisms to mitigate such attacks. Section 4.3 explains the environment we deployed for tests. Section 4.4 shows the results of the experiments. Section 4.5 presents conclusions and future work.

1The work this chapter presents was developed in cooperation with Doctoral Student Francisco Cómbita and his advisor Professor Nicanor Quijano from the Department of Electrical and Electronic Engineering at Universidad de los Andes, Colombia.
4.1 Related Work

Virtualization and cloud computing technologies have been used to create virtual testbeds for ICSs [7][33][84]. For example, the CPSTCS testbed [53] uses computer clusters, networking devices, and PLCs at its lower layer, and virtualization at its upper layers to share physical resources among a set of tenants using the testbed. The motivation behind this work is the need to create virtual environments that are flexible, scalable, and enable the testing of security mechanisms in IP-compatible networks. The authors divide their testbed into three layers. The bottom layer groups physical resources like physical switches, routers, hosts and controllers. The next layer, the virtual layer, has virtual switches and virtual hosts, mathematical models of controlled process, software simulators, APPs, and Scripts. Finally, the application layer dynamically assembles resources into a variety of service nodes through cloud-based functions. However, most authors do not show details about the mathematical models on their simulation tools neither they present experiments to evaluate prototype or performance. One difference with our work, is that we aim to build a virtual environment that can emulate with fidelity both, the network, and the physical system under control.

Alves et al. [7] propose a high-fidelity framework to virtualize the main components of a SCADA system, including the physical system, cyber-physical links, distributed control systems, and the network. To validate fidelity, they compare the behavior of their virtual representation and a real SCADA system, a virtualized gas pipeline. The testbed has a PLC that maintains the pressure between low and high setpoints. The virtual system has a Simulink/Matlab model representing the gas pipeline and they use the OpenPLC platform to implement the control logic of a PLC. Results of both systems are analyzed with similar time-responses. In addition, the authors run a DoS attack against both systems finding a great difference between delays (12 sec. in the physical vs. 0.54 ms in the virtual testbed) because the virtual testbed runs on more powerful hardware. Our work differs in the implementation of the network system. We use Mininet to emulate a network with higher fidelity, because each of our nodes implements the complete communication stack that a host or switch would have in a network. In addition, our plant model runs on Python libraries that enables the use of different mechanisms to simulate the physical process and to evaluate the impact of communication in the response of these systems.

The authors in [84] designed a middleware architecture to improve CPS resilience by coordinating and adjusting device interactions. Their prototype uses Sendim, an extension of the SDNSim testbed [83], that includes Cyber-physical (CPS) elements. These extensions however, run mathematical models that estimate the value of the real systems but do not include computing and networking stacks of the devices that are being emulated thus reducing results fidelity.

Authors in [55] present a Network Industrial Control System testbed based on emulab. The testbed uses tightly and loosely-coupled code in the cyber layer (TCC and LCC, respectively) to represent the interaction between PLC and physical models. The physical plant behavior is simulated in Matlab, and communicates with PLC nodes using RPC/TCP, enabling the PLC to read every value generated by the Matlab code.

The testbed discussed in [61] aims to represent in a realistic manner the industrial physical equipment used in ICS. The testbed has real industrial equipment connected through a network. Two examples of man-in-the-middle attack are presented as proof concept of the testbed. The use of real equipment provides a high degree of fidelity of the ICS components,
but result in a costly and hard-to-scale environment.

The Lancaster’s ICS testbed [62] has four zones: safety, manufacturing, demilitarized, and enterprise. Each zone uses industrial and commercial physical equipment to represent a real industrial control system and traditional TCP/IP to communicate between zones. Although the testbed offers a wide variety of ICS equipment, network protocols, and processes to perform experiments expanding and maintaining such testbed might be cost-prohibitive and therefore a virtual environment may be cost-effective.

The work in [107] presents an electrical substation honeynet, based on virtualization technologies to achieve a high fidelity representation of a large substation. The authors use Mininet [94] to emulate the substation network and each Mininet node represents an IED. To minimize the logic complexity of the mininet nodes, each node implements a traffic mirroring service that redirects the messages to a separate machine running SoftGrid [65]. Also, another machine runs PowerWorld simulate the electrical system. Our work follows a similar approach.

NIST developed an ICS testbed to create three different scenarios [28] with physical equipment. The first scenario emulates the Tennessee Eastman Process using Simulink to represent the physical plant. The second scenario, emulates a cooperative robotic assembly for smart manufacturing using EtherCAT, a real-time industrial protocol, to communicate. The last scenario represents a SCADA, and a wide area network running a network emulator, in addition to real equipment. Network emulation tools provide traffic shaping functions to represent different communication links, available bandwidth and delays.

Table 4.1 summarizes related work. There is wide variety of design choices for each testbed as they have different goals. Testbeds focused on security and high fidelity of behaviors tend to rely on physical equipments to represent the involved entities [28,61,62]. Nevertheless, most of these testbeds present a trade-off between fidelity, enabled by using expensive real devices, and network complexity and scalability, enabled by using virtualized environments. The testbeds that focus on network security and the impact of network behavior tend to use virtual or emulation environments to represent the network [53,55,84,107]. The complexity of represented networks varies with each testbed, but those using either Mininet or Emulab have the most complex networks. The majority of these systems use simulation to represent the physical plant behavior, with some exceptions which use physical equipment [28,53,62].

To contribute with the development of high-fidelity virtualized environments for security of CPS systems, we present a virtual environment to emulate a nonlinear plant, considering the behavior of the communication control network. Our goal is to evaluate the impact of message exchange and processing delays between nodes present in the system. The code is available as open source at Github.

In addition to developing a testbed we study in this chapter techniques for attack detection, identification, and response. While several works have studied attack detection [59], attack identification (establishing specifically which device is the source of the malicious data) and attack response have received less attention. This chapter continues our previous work [37] on attack detection, identification, and isolation by running a virtualized environment that emulates the complete cyber-physical system, with detection, identification, isolation and mitigation of bias injection attacks on a three-tank system, validating our pre-
Table 4.1: Summary of related work on testbeds for Industrial Control System Security experimentation

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Industrial Plant Representation</th>
<th>Industrial Equipment Representation</th>
<th>Network Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPSTCS Testbed [53]</td>
<td>Simulation (Matlab, Modelica, Ptolemy)</td>
<td>Physical Equipment</td>
<td>Emulation (Open vSwitch)</td>
</tr>
<tr>
<td>Pipeline Virtual Testbed [7]</td>
<td>Simulation (Simulink)</td>
<td>Emulation (Open PLC)</td>
<td>Physical</td>
</tr>
<tr>
<td>SD-CPS [84]</td>
<td>Simulation (Sendim)</td>
<td>Simulation (Sendim)</td>
<td>Emulation (SDN Network)</td>
</tr>
<tr>
<td>Lancaster ICS Testbed [62]</td>
<td>Physical Equipment</td>
<td>Physical Equipment</td>
<td>Physical Equipment</td>
</tr>
<tr>
<td>Electrical Grid Honeynet [107]</td>
<td>Simulation (PowerWorld)</td>
<td>Emulation (Mininet node SoftGrid)</td>
<td>Emulation (Mininet Network)</td>
</tr>
<tr>
<td>NIST ICS Testbed [28]</td>
<td>Simulation (Simulink) Physical Plants</td>
<td>Physical Equipment</td>
<td>Physical Equipment</td>
</tr>
</tbody>
</table>
4.2 Networked Control Systems - Fault Tolerant Control

This section presents an overview of control systems, describes sensor integrity attacks, and introduces the concept of unknown input observers (UIOs) to identify specific sensors sending incorrect values.

Networked Control Systems (NCS) are feedback control systems with their components linked via a real-time network. The main goal of feedback control systems is to meet requirements regarding process outputs. The controller achieves this by reading information from sensors and using that to compute, and send to actuators, control actions that reduce the error between a desired reference setpoint, and the current measurement of an output variable. These actions control external disturbances and mitigate the effect of the imperfections of the plant model.

4.2.1 Controlled System

Two common control objectives are a zero-steady state error and fast a response. Zero steady-state error means that the defined setpoint and the output value are equal after a transient period. Fast response to changes in the setpoint is usually obtained with a design based on pole placement. Figure 4.1 shows a typical block diagram of this kind of networked control system.

![Networked control system with state feedback and reference tracking.](image)

Figure 4.1: Networked control system with state feedback and reference tracking. The objective of the system is to maintain system output $y_k$ at the desired setpoint $s_k$. [43]

Many industrial plants are nonlinear. Frequently, it is convenient to use linear approximations of plant models to design controllers and fault identification mechanisms. Such
linear approximation can be obtained if the nonlinear model is linearized around the operation point. When the controller is interconnected through a network to the physical process, it is necessary to have a discrete-time model. Thus, we need to choose a sampling time to generate a discrete-time linear time invariant model for a plant:

$$\begin{align*}
    x_{k+1} &= Ax_k + Bu_k \\
    y_k &= C x_k,
\end{align*}$$

(4.1)

where \( k \in \mathbb{Z}^+ \) represents the discrete time instant, \( x_k \in \mathbb{R}^n \) represents the state of the system, \( u_k \in \mathbb{R}^m \) represents the control input vector, and \( y_k \in \mathbb{R}^p \) represents the measurement output vector. The system matrices representing the physical invariants of the system are \( A \in \mathbb{R}^{n \times n} \), \( B \in \mathbb{R}^{n \times m} \), and \( C \in \mathbb{R}^{p \times n} \).

### 4.2.2 Networked Controller

A network controller must satisfy some requirements: tracking of reference inputs, closed loop stability, disturbance rejection, and decoupling from other input interference in the case of multiple input/multiple output systems. A common strategy to meet these specifications is having a feedback state with one integral action. This type of control is usually referred to as PI control.

The discrete-time integrator is given by the following equation,

$$z_{k+1} = z_k + T_s(s_k - \hat{y}_k),$$

(4.2)

where \( z_k \) is the output vector of the integrator, \( s_k \in \mathbb{R}^m \) represents the set-point or reference input vector, \( \hat{y}_k \in \mathbb{R}^q \) represents the controlled output vector, and \( T_s \) represents the sampling time of the discrete-time system.

The nominal control law of a PI control system \( v_k \) is a state feedback given by

$$u_k = -[K_1 \quad K_2] \begin{bmatrix} \hat{x}_k \\ z_k \end{bmatrix},$$

(4.3)

where \( K_1 \) and \( K_2 \) are constant gains and \( \hat{x}_k \) is the estimation of the state variables. The gains \( K_1 \) and \( K_2 \) are computed to stabilize the closed loop control system and achieve the required performance of the whole system.

The dynamics of PI closed-loop control systems are given by

$$\begin{bmatrix} \hat{x}_{k+1} \\ z_{k+1} \end{bmatrix} = \begin{bmatrix} A - BK_1 & -BK_2 \\ -T_s C & I \end{bmatrix} \begin{bmatrix} \hat{x}_k \\ z_k \end{bmatrix} + \begin{bmatrix} 0 \\ T_s I \end{bmatrix} s_k$$

$$y_k = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} \hat{x}_k \\ z_k \end{bmatrix},$$

(4.4)

where \( I \) is an identity matrix with size equals to the number of controlled outputs.

The standard way to perform state estimation to obtain \( \hat{x} \) in non-stochastic linear systems is through the use of a full order Luenberger observer:

$$\begin{align*}
    \hat{x}_{k+1} &= A\hat{x}_k + Bu_k + L(y_v^d - \hat{y}_k) \\
    \hat{y}_k &= C\hat{x}_k,
\end{align*}$$

(4.5)

where \( y_v^d \) is the sensor value received from the network, and where \( L \) is the Luenberger estimator gain.
4.2.3 Integrity Attacks

The objective of traditional security mechanisms is to guarantee integrity and authenticity of messages. Nevertheless, if an attacker breaks this first line of defense, it can send messages with tampered sensor readings. Integrity sensor attacks occur when any of the system sensors cannot maintain data integrity i.e. sensor measurements are not accurate [30].

A system output when a sensor is attacked can be modeled as

\[ y_k^a = \begin{bmatrix} C \quad 0 \end{bmatrix} \begin{bmatrix} x_k \\ z_k \end{bmatrix} + F_s v_k^a, \]  

(4.6)

where \( F_s \) represents the relation between the injected signal and the modified measurement. We only consider bias injection attacks, where the attacker introduces a bias to the real signal.

In a bias injection attack, the attacker’s goal is to deceive the controller to produce an erroneous control action. These attacks may be defined as attacks on the system outputs, but keeping those outputs compatible with the measurement equation of the system [158]. Figure 4.2 shows a way of implementing this kind of attack; there \( F_s \) indicates the output where the bias is added and, \( v_k^a \) is the bias the attacker wants the system to have. For these attacks, the attacker does not require knowledge about the model of the system; the knowledge about current values of the measurements is enough. With current values and the span of the measurements the attacker can compute an attack vector. The block diagram shows the signals that satisfy (4.6).

![Figure 4.2: Bias injection attack. The attacker steals a sample from the measured variable and manipulates (adds a value to) it to deceive the controller [117].](image)

4.2.4 Anomaly Detection and Attack Mitigation

Anomaly detection and isolation techniques detect an anomaly and identify the particular device that is sending misleading data. There are several isolation mechanisms in the fault tolerant control literature, in this chapter we work with structured residuals. Structured residuals have observers that are insensitive to one specific input and are sensitive to the other ones. This mechanism is built using Unknown Input Observers (UIOs). An UIO is a generalization of the Luenberger Observer [33]. A full-order observer can be expressed as

\[
\begin{align*}
\hat{x}_{k+1}^u &= F \hat{x}_k^u + TBu_k + K_U y_k^a \\
\hat{x}_k^d &= \hat{x}_k^u + Hy_k^a,
\end{align*}
\]  

(4.7)
where $\hat{x}_k^{u} \in \mathbb{R}^n$ is the estimated state vector, and $w_k \in \mathbb{R}^n$ is the state vector of this full-order observer, which is computed by the linear transformation $w_k = T x_k$. $F$, $T$, and $K_U$ are matrices that must be designed such that the unknown input $v_s$ is decoupled from the other inputs.

The anomaly isolation mechanism is responsible for identifying the component that is sending misleading information. To achieve this purpose, and based on the concept of a structured residual set, one UIO is associated to each output; each observer is insensitive to anomalies on one sensor and sensitive to the other ones. The $j^{th}$ observer is designed to be insensitive to anomalies on sensor $j^{th}$. Hence, we need to build a bank of observers.

![Diagram](Image)

Figure 4.3: Mechanism to mitigate sensor attacks. It uses values reported by sensors, actuators, and the plant model to determine whether there is an attack or not. If an attack is detected then an additional signal, $a_k$, is injected. [117]

We assume that there are no simultaneous anomalies on sensors, i.e. only one anomaly is active on the whole set of sensors, and we use UIOs to correct the malicious signal. The purpose of an attack mitigation mechanism is to reduce the impact of sensor attacks over networked controlled systems. Figure 4.3 shows a block diagram of this mechanism.

The diagram shows a typical networked controlled system, with an addition, a component to detect anomalies and mitigate attacks. All these blocks are developed as external digital controllers running over one industrial computer, or even over a PLC. This architecture only requires information about control actions, sensor information, and the capability to modify the information that the actuator of the system receives through the network infrastructure. When there is no attack on sensors of the system, the signal $a_k$ is equal to zero and the signal $m_k$ is equal to $u_k$, but when an attack is isolated an additional signal $a_k$ is computed and added to the original signal $u_k$ and this sum $m_k$ is sent to the controller in order to mitigate the effect of the attack.
4.3 Experimental Setup

In chapter 3, we ran integrity attacks and mitigation experiments with a simple model of a physical linear system with uncoupled stages [119, 133]. This chapter extends our previous work to incorporate a new nonlinear plant and a new attack-mitigation strategy based on UIOs. The previous work represented a linear model by running several uncoupled processes. In addition, that model only kept track of one control signal, associated to one variable, and ran a mitigation strategy that simply turned a valve on or off. The new nonlinear model must handle relations among variables. This feature better represents a real plant but also requires a more complex control strategy; a proportional integral (PI) controller. In addition, we extended the model to represent delays because that is a key feature of real plants.

We implemented a prototype of a three-tank system model based on a nonlinear approximation. Chapter 3 describes the original model [8] and Figure 4.4 illustrates it. The controlled plant is a hydraulic system with three cylindrical tanks of the same dimensions, two cylindrical pipes connect the tanks, and two pumps governed by motors DC that supply liquid for two of the three tanks. This system is a testbed that serves as an example of liquid storage in industrial or water treatment plants. In these processes a fundamental requirement is to maintain the predefined operational points because the chemical reactions that are fed with them require it. This hydraulic system prototype is commonly used to verify the effectiveness of controllers and model-based fault diagnosis systems.

Our prototype uses the nonlinear equations shown in (4.8), where:
- \( q_{13}(t) \) represents the water flow-rate from tank 1 to tank 3
- \( q_{32}(t) \) represents the water flow-rate from tank 3 to tank 2
- \( q_{20}(t) \) represents the water flow-rate of tank 2 draining the water out of the system
- \( Q_1(t) \) and \( Q_2(t) \) represent the input water flow-rate to tanks 1 and 2 respectively
- \( L_1(t), L_2(t), \) and \( L_3(t) \) are the levels of the tanks 1, 2, and 3 respectively
- \( S \) represents the cross sectional area of the tanks
- \( S_n \) represents the cross sectional area of the pipes between tanks
- \( \mu_{13} \) represents the outflow coefficient from tank 1 to tank 3
- \( \mu_{32} \) represents the outflow coefficient from tank 3 to tank 2
- \( \mu_{20} \) represents the outflow coefficient of tank 2.

\[
\begin{align*}
S \frac{d}{dt} L_1(t) &= Q_1(t) - q_{13}(t), \\
S \frac{d}{dt} L_2(t) &= Q_2(t) + q_{32}(t) - q_{20}(t), \\
S \frac{d}{dt} L_3(t) &= q_{13}(t) - q_{32}(t),
\end{align*}
\tag{4.8}
\]

\[
\begin{align*}
q_{13}(t) &= \mu_{13} S_n \text{sgn}[L_1(t) - L_3(t)] \sqrt{2g[L_1(t) - L_3(t)]} \\
q_{32}(t) &= \mu_{32} S_n \text{sgn}[L_3(t) - L_2(t)] \sqrt{2g[L_3(t) - L_2(t)]} \\
q_{20}(t) &= \mu_{20} S_n \sqrt{2gL_2(t)}.
\end{align*}
\]

The prototype has three main components. One component emulates the control system
Figure 4.4: Experimental Setup for Virtual Experimentation of Non Linear Plant. A Mininet LAN was created with the control system components. The PLC101 only interacts with the plant through the sensors and actuators.

of the non-linear plant, another component performs attacks on the integrity of the data reported by one of the sensors, and the last component implements countermeasures to mitigate the impact of the performed attacks.

The prototype uses Mininet [94] and MiniCPS [10] to emulate the nonlinear plant and its control system. Mininet is a light virtualization environment tailored for SDN experiments and emulation, it makes it easier to run experiments and provides higher fidelity than we would get by running simulations. For example, when the sensor LIT101 sends a message to the PLC101, the whole Linux communication protocol stack is involved; we argue that this makes our experiments more realistic than simply modeling a network based on link bandwidth and delay parameters.

MiniCPS extends Mininet to support EtherNet/IP, an industrial network protocol commonly used in ICS for communications between the supervisory and field networks. Originally, in MiniCPS architectures, PLCs read plant state and modify it by reading and changing the values of the variables that represent state and actuators. Instead of this, we introduced a field communication layer, where PLCs send messages to the processes that represent sensors and actuators, and these processes are the only elements that can interact directly with the process that represents the plant. This creates a more realistic behavior of a real-world industrial control system.

We used this extended MiniCPS to represent the control topology shown in Figure 4.3 and emulate the attacks shown in Figure 4.2. The topology includes a nonlinear plant, a malicious sensor sending altered values, thus performing an integrity attack, and a PLC that can identify this integrity attack and run mitigation actions.

Figure 4.5 shows our implementation, based on MiniCPS. This implementation has traditional network control system elements: a plant, a pair of actuators (P101 and P201), a pair of sensors (LIT101 and LIT201), and a PLC (PLC101) that runs a state estimator. These elements are represented as follows:

- A Mininet node, running the python script *physical_process.py* represents the physical
Figure 4.5: MiniCPS experiment topology. The MiniCPS topology includes a SQLite database and python scripts representing each element of the industrial control system. Two sensors: lit101 and lit201, two actuators: pumps q101 and 102, one plc, and a physical process script.

- Each of the sensors is represented by a Mininet node; sensor lit101 runs the script `sensor_lit101.py` and sensor lit201 runs the script `sensor_lit201.py`.
- Each actuator is also represented by a Mininet node, running the scripts `pump_q1.py` and `pump_q2.py`.
- The PLC runs the script `plc_101.py`.
- Finally, the state of the physical process is stored in a SQLite Database called "process.sqlite". The same node that runs the physical process stores this database.
- All Mininet nodes communicate through Ethernet/IP sending measurements and action commands between them.

The node that represents the physical process (the three tanks) runs a script that works as follows: first, it reads the values of the actuators, those are values stored in the database. Then, it solves the system of differential equations to calculate the new state of the system and writes the new state to the database. The database has registers for each one of the tanks. After updating the values the script sleeps during the remaining part of the sample period, and then repeats the steps.

As mentioned before, we extended the MiniCPS implementation by including process that represent sensors and actuators. Therefore, our plant node reads its inputs, the state of the actuators, through the MiniCPS database. After that, the plant node uses the previous

\[ \text{odeint is a solver for differential equations systems that is part of the scipy library: } \text{https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html} \]

\[ \text{Available at: } \text{https://www.sqlite.org/index.html} \]
state of the plant and the state of the actuators to compute the next state of the plant; the
next state is the result of solving the nonlinear differential equation system of the plant.
Once the new state is calculated, the plant writes the new level of each tank to the MiniCPS
database. Sensors LIT101 and LIT201 read the corresponding tank level and send it to the
PLC101. Sensor LIT101 acts as a malicious node and runs the integrity attack Figure 4.2
shows. PLC101 reads tank levels and also runs a state estimator of the plant, and compares
these values to detect integrity attacks on the sensor. If the PLC detects an attack, then it
calculates a compensation action to mitigate it. Finally, PLC101 sends the appropriate levels
to the actuators to keep each tank within a desired range level.

All the messages exchanged by the components pass through the switch. As a conse-
quence, topology and flows could be controlled by the SDN Controller. In the current setup,
however, we did not implement any SDN application because the defense mechanism is
installed directly into the PLC101.

Our implementation is available at Github: https://github.com/ComitUniandes/ICS-SDN/
tree/master/francisco-topo

4.4 Results

All the experiments were run on a machine with an Intel(R) Core(TM) i7-4770, (3.40GHz)
processor with 16GB of RAM. This machine runs Ubuntu 16.04, as host operating system,
and VirtualBox 6.0.12⁵. As guest operating system, we used an Ubuntu 14.04 virtual machine
image with mininet already installed⁶. In addition, we installed MiniCPS⁷ and applied our
extensions. We configured this virtual machine with 4 processors and 2048GB of RAM.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0.0154 m²</td>
</tr>
<tr>
<td>Sₙ</td>
<td>5 × 10⁻⁵ m²</td>
</tr>
<tr>
<td>μ₁₃ = μ₃₂</td>
<td>0.5</td>
</tr>
<tr>
<td>μ₂₀</td>
<td>0.6</td>
</tr>
<tr>
<td>Qᵢₘₐₓ i ∈ [1, 2]</td>
<td>1.5 × 10⁻⁴ m³ s⁻¹</td>
</tr>
<tr>
<td>Lⱼₘₐₓ j ∈ [1, 2, 3]</td>
<td>0.62 m</td>
</tr>
</tbody>
</table>

Table 4.2: Parameter values of the three tank system.

Table 4.2 shows the parameter values we used in our Python implementation of the
nonlinear equation (4.8) of the three tanks system.

⁵Available at: https://www.virtualbox.org/wiki/Linux_Downloads
⁶Available at: https://github.com/mininet/mininet/wiki/Mininet-VM-Images
⁷Available at: https://github.com/scy-phy/minicps
The operation point of the level of the tanks is given by \( L_{1_{\text{oper}}} = 0.4 \text{ m}, L_{2_{\text{oper}}} = 0.2 \text{ m}, \) and \( L_{3_{\text{oper}}} = 0.3 \text{ m}, \) where \( L_{j_{\text{oper}}} \quad (j \in \{1, 2, 3\}) \) represents the level of tank \( j \) respectively.

The corresponding operation point for the pumps that govern the inputs of the tanks is given by \( Q_{1_{\text{oper}}} = 3.5018 \times 10^{-5} \text{ m}^3\text{s}^{-1} \) and \( Q_{2_{\text{oper}}} = 3.1838 \times 10^{-5} \text{ m}^3\text{s}^{-1} \) where \( Q_{i_{\text{oper}}} \quad (i \in \{1, 2\}) \) represents the input flow rates needed to reach the operation point levels, when the system does not have any disturbance.

For the three tank system we can find a linear discrete-time model around the operation point (equation (4.1)) so that we can design the control parameters, and the UIO parameters. The sampling time of the system is \( T_s = 1 \text{ s}. \) The matrices \( A, B, \) and \( C \) are given by

\[
A = \begin{bmatrix}
0.9888 & 0.0001 & 0.0112 \\
0.0001 & 0.9781 & 0.0111 \\
0.0112 & 0.0111 & 0.9776 \\
\end{bmatrix}, \quad B = \begin{bmatrix}
64.5687 & 0.0014 \\
0.0014 & 64.2202 \\
0.3650 & 0.3637 \\
\end{bmatrix}, \\
C = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{bmatrix}.
\]

The state variables are chosen as the liquid levels in tanks 101, 301 and 201, respectively.

The remote controller gain for state feedback is \( K_1 \) and the integral gain is \( K_2, \) these gains are obtained using the pole placement technique, and they are given by

\[
K_1 = 10^{-4} \begin{bmatrix}
21.6 & 3.0 & -5.0 \\
2.9 & 19.0 & -4.0 \\
\end{bmatrix}, \quad K_2 = 10^{-4} \begin{bmatrix}
-0.95 & -0.32 \\
-0.30 & -0.91 \\
\end{bmatrix}.
\]

The Luenberger estimator gain is given by

\[
L = \begin{bmatrix}
0.9995 & 0.0005 \\
0.0005 & 0.9995 \\
45.0167 & 42.5017 \\
\end{bmatrix}.
\]

All mentioned components are part of the original networked control system and were developed before any security design.

Figure 4.6 shows the response of the system to changes in the set-points or reference inputs. The figure presents the level of the three tanks and the dotted black line indicates the desired behavior following the setpoints established for each water tank. The control system acts upon Tank1 and Tank3 levels; the level of Tank 2 is a consequence of the level in the other two tanks. The figure shows that the water level follows closely the desired results. There is a small delay between a change of the setpoint and the water tank level, but this delay is always present in Control Systems and is called the transient response of the system.

The Anomaly Isolation and Attack Mitigation block is designed using two UIOs, one for each sensor that may be attacked. UIO1 has three inputs: 1) the control action of pump P101, 2) the control action of pump P201, and 3) information from the sensor LIT201.
Figure 4.6: Water tank level behavior without an attack. The PLC101 can maintain the water tank level in the desired setpoints.

It is worth noting that UIO1 does not have information about sensor LIT101. The recursive equation for the UIO1 is given by equation (4.7) and the following matrices

$$\begin{align*}
F_1 &= \begin{bmatrix}
0.9888 & 1.0000 & 0.0112 \\
0 & 0.0010 & 0 \\
0.0112 & -0.9890 & 0.9776
\end{bmatrix}, & K_{1u} &= \begin{bmatrix}
-1.0000 \\
-0.0010 \\
1.0000
\end{bmatrix}, \\
T_1 &= \begin{bmatrix}
1.0000 & -0.0001 & 0 \\
0 & 0 & 0 \\
0 & -0.0001 & 1.0000
\end{bmatrix}, & H_1 &= \begin{bmatrix}
0.0001 \\
1.0000 \\
0.0001
\end{bmatrix}.
\end{align*}$$

UIO2 also has three inputs: 1) the control action of pump P101, 2) the control action of pump P201, and 3) information from the sensor LIT101. It is worth noting that UIO2 does not have information about sensor LIT201.

The recursive equation for UIO2 is given by (4.7) and the following matrices

$$\begin{align*}
F_2 &= \begin{bmatrix}
0.0010 & 0 & 0 \\
-1.0000 & 0.9781 & 0.0111 \\
-0.9889 & 0.0111 & 0.9776
\end{bmatrix}, & K_{2u} &= \begin{bmatrix}
-0.0010 \\
1.0000 \\
1.0000
\end{bmatrix}, \\
T_2 &= \begin{bmatrix}
0 & 0 & 0 \\
-0.0001 & 1.0000 & 0 \\
-0.0001 & 0 & 1.0000
\end{bmatrix}, & H_2 &= \begin{bmatrix}
1.0000 \\
0.0001 \\
0.0001
\end{bmatrix}.
\end{align*}$$
4.4.1 Integrity Attack

We implemented the bias integrity attacks described in section 4.2.3. In this attack, the sensor LIT101 attacks the integrity of the water tank readings by subtracting a value from the sampled value. In our case, this value was 0.02m. Hence, for each sample reported by LIT101 a value of 0.02m was subtracted. This attack begins around 200s and ends at 350s.

Figure 4.7a shows the results of the bias attack when no defense is present. We can see the impact of the attack; the PLC101 believes that the plant is some centimeters below the desired setpoint level, this causes the PLC101 to compensate for this error causing the water tank level to stay above the desired setpoint level across the entire duration of the attack. We can see that the water tank level 2 is also impacted by these erroneous control actions. Figure 4.7b shows the system response when our defense is present. The defense is able to properly mitigate the impact of the bias attack by keeping the water level above almost all of the attack duration. Shortly after the attack starts, it is identified and the PLC starts correcting the control action to mitigate the impact of the attack. For this reason, after the initial overshoot present around the start of the attack, the water level starts returning to its desired point and stays in this point for the whole duration of the attack.

Figure 4.7: Bias attack experiments.

We performed additional experiments to further test the behavior of the bias attack and our defense. We ran five sets of experiments; in each set we tested the behavior of the system with and without defense. Also, we ran all the experiments for 500 seconds. In all the experiments, the attack starts at 200 seconds and stops at 350 seconds, and the setpoints of water tank 1 and 2 were the same. During the bias attacks, we changed attack values, starting from 0.01 until 0.05, increasing 0.01 each time. To synthesize the results of these experiments, we calculated the mean error of the tank 1 water level. The mean error is defined as the average error between the desired setpoint for all times t of the experiment, and the current
Figure 4.8: Mean error of tank1 water level with and without defense and different values for the bias attack. The maximum mean error with defense is 0.05. The error with the defense is always less than the error without defense.

water tank 1 level. Figure 4.8 shows the results of these experiments. The mean error without defense increases as the value of the bias attack is increased. Until it reaches a value higher than 0.015. In contrast, when our defense is present, the mean error stays at a maximum of 0.05. For comparison, we show in the figure the mean error when no attack is present, we can see that the behavior of the plant is very close to the behavior when no attack is present.

4.5 Conclusions and Future Work

This chapter presents a virtual environment to emulate a network control system controlling a nonlinear plant, and detect and identify attacks. Based on the described work, we think that Mininet and MiniCPS are appropriate environments to model the behavior of nonlinear plants and run security experiments. The use of virtual environments for these experiments enables the creation of realistic testing environments, that can scale up and down, and without the cost of using physical equipment.

This chapter used UIOs as a detection and mitigation mechanism. We ran bias attacks
to evaluate our environment and the mitigation mechanism and found that the mean error of tank1 water level, with defense, was always lower than mean error without defense, and close to mean error without attack. This proves that the use of UIOs to identify the source of a malicious attack proved to be an effective detection and mitigation mechanism.

As future work, we plan to extend our virtual environments to consider Real-Time Operating Systems. This addition would enable us to build more realistic scenarios; we could identify the impact of attacks on the real time requirements of industrial control systems and their plants. This future work would require the development of hypervisors that are able to offer real time guarantees.

4.6 Acknowledgments

This work is partially supported by the U.S. Air Force Office of Scientific Research under award number FA9550-17-1-0135, the U.S. Department of Commerce by NIST Award 70NANB17H282 and by the Colombian Administrative Department of Science, Technology, and Innovation (Colciencias).
Chapter 5

SDN and NFV Security: Challenges for Integrated Solutions

Telecommunication networks do far more than only forwarding packets; they process traffic through different network functions like proxies, firewalls, intrusion protection systems, and so on. These functions have traditionally been implemented through middleboxes, which are dedicated hardware devices inspecting, filtering, or manipulating network traffic. These middleboxes have to be physically connected between each other, and this physical connection creates a service chain. This paradigm has serious disadvantages such as high capital cost due to costly middleboxes, difficulty and long periods of deployment of new services because of the difficulty in reprogramming or reconnecting these middleboxes, and the inability to adapt the capacities of those services to the current demand, which inevitably causes over or under provision of resources.

Network Function Virtualization (NFV) is a new telecommunication paradigm that enables the implementation of these network functions using software and general computing equipment, rather than dedicated hardware. In an NFV platform, a virtualization layer enables the deployment of virtual machines offering these network functions. Virtualization provides various advantages to the deployment of network functions and their management. First, Commercial Off-the Shelf (COTS) generic servers can be used to host these virtual machines, which avoids the use of expensive and dedicated hardware [113], lowering the capital costs of deploying and managing a network. Second, to deploy new services we do not require buying additional equipment, only new software. Finally, virtualization can help scale up and down these network services, depending on the demand, and offer new services. For example, virtualization allows network operators to offer their physical infrastructure to multiple network services, in the same way that cloud computing providers offer their infrastructure to multiple clients. Network services may be offered to different departments in the same company or even to external customers in some cases. Such flexibility in the deployment and management of new services is the main driver behind NFV.

The deployment and management of NFV is facilitated by the use of Software Defined Networking (SDN). Using SDN network traffic is steered between the network functions [122], and adding or modifying the service chain is a matter of simply creating instances of additional virtual machines and using SDN to update the forwarding decisions for such traffic. SDN also facilitates having different forwarding rules for different traffic subsets; in
the traditional approach an administrator would need to include a proxy to split traffic and forward it to different paths or use IP/MPLS labels to identify particular subsets of traffic, while by using SDN/NFV we only need to add a proxy Virtual Network Function (VNF) and update the forwarding rules.

The flexibility of modifying the operation of an SDN/NFV network, including new parties in the management of the network infrastructure, and the issue of sharing the infrastructure with other tenants brings new security challenges, as we need to guarantee that each network service meets its goals even when other (potentially untrusted) parties are also using the same network infrastructure. To maintain separation between different network services, we need to provide fine-grained access control. Research on improving SDN security with access control is a growing area of interest [135, 136, 168, 169]. However, previous efforts have focused solely on SDN, and have not considered the new challenges of an integrated SDN/NFV network. In addition, NFV introduces the concept of Service Orchestration, which enables the creation of network applications through the composition of network functions using a predefined recipe. We consider that Service Orchestration is an important aspect of NFV that brings new security challenges for access control. These security challenges arise because Service Orchestration uses high level recipes to build new network applications. In this sense, the access control policies defined at this point must be also defined at high level. Nevertheless, it is expected that network applications built in SDN/NFV are composed of heterogeneous resources, given the diversity of network functions, possibly running on top of different implementation technologies. For this reason, different enforcement mechanisms must enforce the high level security policies defined during orchestration.

In this chapter, we present a survey in the main security challenges of SDN/NFV integration and discuss the definition of a secure access control system for SDN/NFV. To do so, in Section 5.1 we propose an integrated architecture of SDN/NFV. In the same Section, we discuss the main proposals in Service Orchestration and Management for SDN/NFV. In Section 5.2 we discuss the main proposals aimed to secure the SDN/NFV platform, we also present a taxonomy of those proposals and discuss their limitations in the scope of SDN/NFV. In that analysis, we focus on the proposals offering access control for SDN and NFV. Finally, in Section 5.3 we consider the similarities of SDN/NFV environments with secure operating systems. Inspired by some of the best-practices and lessons learned in the design of reference monitors, mandatory access control, and policy verification, we show how previous work on secure operating principles can facilitate the analysis and design of secure SDN/NFV infrastructures.

Our contributions include, i) presenting an integrated architecture that enables the discussion of a reference monitor and a mandatory access control system for SDN/NFV, ii) discussing the new security challenges in SDN/NFV, iii) presenting the largest (as far as we are aware) survey and taxonomy of SDN/NFV Management, Orchestration, and Security, iv) identifying how secure operating systems can guide our reference monitor design for SDN/NFV, and v) based on this analysis, proposing an extended architecture for an SDN/NFV Secure Network Operating System.
5.1 SDN and NFV Integration

A malicious or compromised network application can exploit the programmability of SDN/NFV networks to interrupt different network services, compromise the confidentiality of information, and affect network behavior in several ways. To prevent abuse of resources available in SDN/NFV environments, it is important to control how each application interacts with the infrastructure. The following use cases highlight the relevance of this control.

**Service Orchestration:** Service orchestration is a process that performs different steps: i) Receives a request for a specific network application, ii) Selects the appropriated VNFs to be included and chained in the application and according the service request chains them in a specific order, iii) Creates virtual machines instances running each of the required VNFs, this step involves looking for an optimal, or near-optimal, placement of VNFs to minimize used resources, power consumption, etc., iv) Interconnects the VNFs, deploying switches and routers to steer the traffic from one VNF to the next, and v) Monitors resource demands to detect whether it is necessary to scale assigned resources up or down. During the service orchestration process, it is very likely that multiple network functions will perform flow operations on the same flow resource, and it is important to enforce the privileges that these network functions have in order to prevent abusive behavior. The flow ownership and priority override proposed by FortNOX [135] for access control in SDN (where each application is the owner of a flow) makes service orchestration difficult because allowing multiple applications to operate on the same flow would require careful planning of the application priority. Access control for SDN/NFV requires more flexible and fine-grained access control mechanisms to enable multiple applications to cooperate in the management of a flow without creating action conflicts. Nevertheless, access control for SDN/NFV also requires to offer a generic policy language that enables the definition of high level access control policies to be enforced in the network applications.

**Virtual Network Function Privileges:** Service orchestration enables the development of new network services by interconnecting virtual network functions. Service orchestration is inspired by best-practices in software development, which decouple each module of software in order to improve life cycle management and enable software reuse. In this way, network services are not expected to be monolithic but rather a collection of multiple VNFs, each of them performing a specific operation towards the service objective. In a future ecosystem where VNFs are created and maintain by a large set of providers, maintaining the integrity of their software will be a challenge. A compromised or malicious VNF can perform additional functions (other than those specified) on the flows that is processing; for example, a firewall VNF that only forwards or drops packets should not be allowed to modify the values of the packet headers. Static mechanisms of VNF validation and authentication are not sufficient to avoid this type of attacks, because even after a VNF is authorized to participate in a service chain, it can be compromised and affect the whole service chain. For this reason, an access control mechanism must dynamically control the operations that applications can perform over available resources.

**Service Chaining using Third Party VNF:** In a service chain, some virtual network functions might be offered by third parties—i.e., parties outside of the network operator domain. Third parties can offer a VNF via a virtual machine image in an off line mode, or by offering virtual machine instances ready to interconnect in the chain. In each case, network operators must ensure that only the authorized network function participates in the service
chain in the way agreed by all parties. This case requires the identification of the source of the resources, and a policy manager capable of specifying and enforcing constraints over the way in which resources may be included as part of a service.

5.1.1 An Integrated Architecture

Industry and academia have proposed reference architectures to integrate SDN and NFV. The Open Networking Foundation (ONF)—the organization standardizing OpenFlow and advocating SDN—proposes a context where an SDN controller sees NFV as a network resource provider [122]. Similarly, the European Telecommunications Standards Institute (ETSI) describes the advantages of SDN/NFV integration; however, their architecture only considers NFV [49]. The Open Platform for NFV (OPNFV) [124] expands the ETSI architecture by adding OpenStack [161] to the virtualization control module at the NFVI layer. Other papers have also discussed the relationship between NFV, SDN and Cloud Computing [113]. While several organizations and academics have discussed various aspects of NFV and SDN integration, they have not defined how an architecture for discussing access control in SDN/NFV. Without the explicit identification of trust boundaries and threat models, we cannot discuss the security issues that SDN/NFV deployments face and the security architectures that can mitigate or prevent these issues. To address this problem in the next subsection we propose an architecture that integrates SDN/NFV elements enabling service orchestration and resource access control.

Figure 5.1 shows our integrated architecture. It extends previous proposals [49, 113, 122, 124] by adding SDN/NFV components missing by ETSI [49], ONF [122], and OPNFV [124], like the SDN controllers and the application interface and by explicitly identifying how these SDN and NFV components interact. In our architecture, a developer interacts with the platform through the application interface shown at the top of Figure 5.1 (in yellow). This API provides the interface to the service orchestration module (in blue), which translates the service creation request sent by the developer and builds a network service recipe. A recipe contains the virtual network functions required to create the service, a topology description and other quality of service parameters that the service must meet. Using this information, the service orchestration module tells the NFVI manager (in red) to create the required number of virtual machine instances running the desired VNFs [57]. After the VNFs are allocated, the service orchestration module uses the SDN controller to interconnect the VNFs according to the topology specified in the service recipe. Finally, after the service is deployed, the service management module collects metrics about the application performance and performs corrective actions (e.g., scaling up or down services, or sending alerts).

Currently available technologies that support the proposed architecture and offer an initial set of capabilities for SDN/NFV include: AT&T ECOMP [11], ONAP [159] (Open Network Automation Platform), and E2 [129] for Service Management and Orchestration modules, SDN controllers ONOS [20] and OpenDayLight [87], and the Open Network Foundation Platform (OPNFV) [27] as an NFVI manager. OPNFV is based on OpenStack [161] and can be used with the KVM hypervisor. Figure 5.2 shows an example of how these technologies can support an SDN/NFV architecture.
5.1.2 Orchestration and Management in SDN/NFV

As we have seen, the orchestration and management modules control most of the resources in SDN/NFV networks, and as such, they require special attention for security purposes. In this section we survey previous work on orchestration and management and organize them according to the following features: architecture, orchestration, configuration, and evaluation. Table 5.1 shows the classification of the proposals based on these aspects.

Architecture. This aspect considers if a service orchestration and management proposal fits within the ETSI Management and Orchestration (MANO) architecture or not. A proposal may extend the ETSI architecture by adding new components within the ETSI boundaries, may extend an ETSI component by adding functionality to it, or it may add an external component. This category also explores the integration of an SDN controller. Most of the proposals we found extend the ETSI architecture [11, 15, 17, 19, 90, 99, 105, 126, 159]. Three of these solutions build on the ETSI MANO module and extend it with more functions [11, 90, 105]. Other solutions [64, 67, 106, 132, 180] propose their own orchestration module outside the NFV architecture; they argue that it is easier to have a global view of the infrastructure from outside of the NFV architecture.

Several of the proposed solutions use SDN because it enables the construction of a global
Table 5.1: Classification of proposals to orchestrate and manage SDN/NFV environments

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Extends ETSI Architecture</th>
<th>Extends ETSI Component</th>
<th>Adds External Component</th>
<th>Integrates SDN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>●</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>●</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>●</td>
<td>●</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>●</td>
<td>●</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orchestration</th>
<th>Resource Management</th>
<th>Traffic Management</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>●</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Physical and Virtual Resources</th>
<th>Virtual Resources Only</th>
<th>Single Tenant</th>
<th>Multi Tenant</th>
<th>Single Domain</th>
<th>Multi Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>●</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>●</td>
<td>-</td>
<td>●</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Validation</th>
<th>Simulation</th>
<th>Testbed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>●</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>●</td>
</tr>
</tbody>
</table>

Legend: ●: feature considered by authors, ○: feature not explicitly stated or exhibits ambiguity
view and management of the network [11, 19, 67, 99, 126, 129, 159], this feature can be used to orchestrate network resources and control traffic flowing through the infrastructure.

**Orchestration.** This aspect classifies a proposal according to the resources it handles. Approximately, 65% of the proposed solutions [11, 15, 17, 64, 67, 90, 99, 105, 129, 159] manage storage, computational, and network resources. Half of the articles address traffic management orchestration [19, 99, 106, 126, 129, 132, 180]. Two projects [99, 129] address management in both categories: resources and traffic, and both solutions use SDN.

**Configuration.** This aspect indicates whether a proposal handles hybrid environments (i.e., environments with physical and virtual resources) or virtual resources only. We also check if the solutions are single-tenant or multi-tenant, and finally, we check if the solutions address single-domain or multi-domain environments. In the table we can see that half of the previous work focuses on a hybrid configuration [15, 19, 64, 90, 106, 126, 180], and the other half focus on virtual configurations [11, 17, 67, 99, 105, 129, 132]. Regarding tenancy and domain, most of the proposals [11, 15, 17, 19, 105, 126] were designed for a single-tenant, single-domain scenario.

**Validation.** This aspect classifies the proposals according to their validation methods and they help us identify the maturity of the technology proposed. In particular we look if the technology was implemented in a emulation such as Mininet, or if they used a testbed. We can see in the table that there is an equal split in the ways the service management and orchestration technologies were validated.
Security. Most of the proposals in Table 5.1 focus on tasks related to Orchestration and Management, but they do not consider security requirements. Nevertheless, in multi-tenant or multi-domain environments several security issues emerge as multiple applications have access to the same resources, applications may affect the behavior of other applications, and various developers offer VNFs. Even in the single-tenant single-domain case, some security issues emerge as they also run VNFs that may come from untrusted developers and compete for shared resources.

The only two proposals in Table 5.1 discussing security are Congress [15] and GBP [126]. In particular, they propose to define and enforce rules to control network services. Congress supports the definition of a security policy to rule data services (conditions to expose data, resource owners, etc.), and also checks compliance of configurations with rules, while GBP allows users to define rules that mediate traffic between participants. Although these works address security issues, their scope is limited. Congress mentions an access control policy, but it does not support this kind of policy yet (it has not identified standard resources, operations, or defined access control rules, all steps which are necessary for access control). Similarly, GBP does not allow users to express access control rules and does not offer enough capabilities in the match/action pair to determine who is allowed to perform certain operations.

5.2 A Survey of Proposals to Secure SDN/NFV Platforms

Having defined the general architecture of SDN/NFV networks and summarized previous work on the Orchestration and Management of this architecture, we focus on security for SDN and NFV platforms. First, we identify SDN/NFV parties that may be malicious or could be compromised. We also analyze previous works and classify them according to type of architecture (SDN, NFV, SDN/NFV) and features that may affect platform security. Finally, we analyze the scope of the proposals and their limitations.

5.2.1 Taxonomy

We grouped previous work based on (1) type of deployment considered, (2) compromised components, (3) security goals, and (4) enforcement points. Table 5.2 summarizes previous work according to our taxonomy. In addition, we separate previous work by columns, depending on whether the authors considered SDN architectures, combined SDN/NFV architectures, or solely NFV architectures.

Type of Network. Our possible types of networks are SDN, NFV, and SDN/NFV (the columns in Table 5.2). As we can see, most of the previous work focusing on security has considered mostly SDN in isolation. Works to secure SDN include policy enforcement and analysis of applications as these are key aspects to secure SDN platforms [3, 14, 16, 22, 29, 52, 86, 88, 97, 103, 104, 123, 135, 136, 139, 144, 148, 151, 153, 168, 169, 172].

SDN is more mature than NFV; SDN was proposed in 2010 [109] and some of its principles were stated in 2007 with Ethane [31], while NFV was proposed in 2012 [36]. As a consequence, the number of proposals to enhance SDN security is larger than NFV.

We did not include in our table SDN works that provide "security as a service" because their goal is not to secure the platform but to enable clients to build their own security services,
like DoS attack detection and reaction \cite{25,116,120,166,174} or enhancement of HoneyNet capabilities \cite{68,130}.

We grouped the NFV works that explicitly use an SDN controller as the component to control network configuration \cite{21,128,140,175} in the second column of Table \ref{tab:deployment}. Finally, although it is expected that most NFV deployments will integrate SDN in the future \cite{11,101,125}, there are some use-cases for NFV security that can be studied isolated from SDN. We grouped them in the last column of Table \ref{tab:deployment}.

**Deployment.** SDN/NFV architectures may be deployed with different configurations changing the trust boundaries of the system and the corresponding access control requirements. There are two parameters that affect trust: (i) tenancy, the number of different parties using physical or virtual resources available in the infrastructure, and (ii) domains, understood as the number of network administrative domains that are involved in the deployment offered to the final customer. Considering these parameters, we have four possible SDN/NFV deployments: Single domain – Single tenant, where the owner of the infrastructure and the user of VNFs are the same; Single domain – Multi tenant, which is analogous to a classic Cloud Computing example where cloud users wanted to ensure that (i) cloud providers are trustworthy and that (ii) other tenants cannot interfere with their security goals; Multi-domain-single tenant, where multiple telecommunication providers have an agreement to offer a service, and where domains should interact only in ways explicitly established by an agreement between operators; and Multi domain – Multi tenant, where a provider participates in services orchestrated among different network users to offer global or national network services.

**Compromised Components.** SDN/NFV domains have different actors, such as software providers, infrastructure owners, and orchestration managers. A malicious or compromised actor can have different effects on the system. For example a compromised software vendor can offer a malicious VNF which could compromise a whole service chain. A compromised hypervisor could affect the behavior of all VNFs running on the physical machine. Finally, a compromised MANO component could harm the whole domain. Figure \ref{fig:adversaries} illustrates the possible adversaries. To identify the adversaries, we analyzed the main articles in the area; sometimes, the attacker model was explicit or it was clearly mentioned which the security objective was. For other cases, these characteristics where not very clear and we had to infer them based on certain phrases or key ideas that the author presented.

In our taxonomy we consider five elements as potentially compromised: VNF applications, SDN applications, SDN controllers, Hypervisors, and Managers. From a logical perspective, SDN switches perform the same tasks that network functions offering a forwarding function. In addition, there exists multiple software implementations of SDN switches. For these reasons, and to simplify our taxonomy analysis we consider SDN switches as VNFs. A malicious manager can be either a malicious ETSI MANO component, or a malicious administrator trying to affect the behavior of its company VNFs or third party VNFs.

**Security Goals.** We define integrity, confidentiality and availability at service level rather than at a packet level. Integrity ensures that commands sent by controlling applications are not adulterated and the corresponding actors implement the intended control action. Examples of integrity violations include a malicious SDN switch that affects the behavior of an SDN application by not applying a command the SDN controller issued, or an unauthorized party altering the flow table of an SDN switch. Confidentiality guarantees that an application cannot observe data or behavior of other applications. An example of a confidentiality
violation in a multi-tenant environment would be that one party gets unauthorized access to the policies being applied by another party to their VNFs. Availability guarantees that offered services keep running with acceptable levels of quality.

Access control guarantees that only authorized parties can perform certain operations on a set of resources. In the SDN scenario, these resources are represented by flow tables on the controllers and switches. In NFV the resources are more diverse, and could be virtual machine instances, software repositories, etc.

Accountability is the capacity of establishing the entities that participated in a particular operation. Including entities that made decisions as well as the ones that performed particular actions.

**Enforcement.** We identified two main mechanisms to improve SDN/NFV security: online enforcement and offline evaluation. The former includes mechanisms that enforce security policies at run time, the latter refers to static analysis or dynamic analysis in a controlled scenario, not in a production environment. Based on the architecture presented in Figure 5.3, we identified the following online enforcement points: SDN Controller, Hypervisor, Platform Manager and Network Orchestrator, the last two are MANO modules. An enforcement point at the controller mitigates faulty or malicious code in controller modules and SDN applications. Enforcement at the hypervisor mitigates the impact of compromised virtual resources, like network functions, SDN switches and routers, and even SDN controllers
if they are virtualized. Enforcement at the platform manager allows having control over network resources and virtual machine instances deployed in the platform. Enforcement at the Network Orchestrator focuses on managing the SDN devices and applications present in the platform.

**Validation.** We also grouped the studied works based on their methods of validation. We classify proposals that build mathematical models or computer simulations as simulations, the cases that use mininet or other emulation technology as emulations, and the cases that use virtual machines and virtual networks as testbeds.

### 5.2.2 Analysis

In most work focusing on SDN, we found that SDN applications are far more popular than considering malicious controllers. This might be representative of real-world threats, as we expect SDN applications to change more frequently, be developed, and supported by more developers than controllers, which increases the risk that in one of these changes, a malicious application might slip in. Although, in some cases researchers assume that platforms build network services developed by only one provider [14,21,29,86,88,97,104,139,144,151–153], we expect this behavior to evolve towards using multiple providers, with the associated advantages and security problems.

Malicious or misconfigured switches have been considered in various works [16,22,52,97,103,128,140,172,175], and their impact is limited to the network service that uses the malicious switch, while malicious or misconfigured SDN applications [3,14,21,29,86,88,97,104,135,139,144,148,151–153,168,172], may have a bigger impact, because SDN applications can affect the behavior of the SDN controller and other resources, including other switches.

**Malicious VNFs.** The most common type of deployment in works that consider malicious VNFs or malicious switches is single domain, including both single-tenant and multi-tenant variations.

Works that propose mechanisms to control malicious VNFs in single-domain multi-tenant deployments use two different points to implement enforcement, the SDN controller [22,128,168] or the Hypervisor [175]. An enforcement point in the SDN controller is appropriate for this type of deployment because the controller has control over VNFs, so it can monitor, handle, and interconnect individual VNFs. Nevertheless, [139] argues that a VNF may craft a malicious packet that could be able to install a rootkit at the SDN controller, thus the need to add enforcement points in the hypervisor. A controller may check different variables at the enforcement point, AuthFlow [52], for instance, checks if an application that requests an operation has been previously authenticated and admitted in the platform. Rosemary also controls operations, but it uses sandboxing to allow only authorized actions generated by platform components [148].

Mechanisms to control malicious VNFs in single-domain single-tenant deployments mainly use offline evaluation and enforcement at the controller and the management module. Offline evaluation can detect misconfigurations, suspicious instructions, and dangerous API calls [97,152]. For multi-tenant scenarios, enforcement at the controller includes actions like mediating all requests from SDN applications to SDN switches [168]. Enforcement at the management module involves monitoring resources and collecting statistics to detect suspi-
Table 5.2: Classification of proposals to secure SDN/NFV environments

<table>
<thead>
<tr>
<th>Applicable in SDN/NFV</th>
<th>SDN</th>
<th>SDN/NFV</th>
<th>NFV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Single domain single tenant | - | ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● }
cious behavior [16]. Malware and misconfiguration detection is important in multi-tenant scenarios because one party could intentionally try to affect other parties.

**Malicious SDN Applications.** The most common type of deployment in this case is single-domain multi-tenant. Adversarial SDN applications are common in this kind of deployment because multiple applications can coexist attached to the same SDN controller and they can be developed by different parties.

Mechanisms to control malicious SDN applications use both offline and online evaluation and enforcement at the SDN controller. Offline mechanisms try to detect malware or misconfiguration that can compromise a network service [14, 29, 86, 104, 151]. The goal of enforcement at the SDN controller [88, 123, 135, 148, 168, 169] is to mitigate the impact of a compromised SDN application on a network service. Similarly, the goal of extensions to the controller or the management module to check new policies generated by SDN applications is to guarantee that new rules do not create configuration errors like loops or blackholes [88].

**Malicious SDN Controllers.** A malicious or compromised SDN controller [21, 22, 29, 97, 123, 128, 136, 139, 140, 169, 172, 175] can have a great impact on network services. The most common type of deployment in this case is single-domain multi-tenant; this may be explained because an SDN controller runs multiple SDN applications that can potentially be developed and deployed by multiple parties. Works that propose mechanisms to control malicious SDN controllers use enforcement at the controller itself. The assumption is that it is not the controller that is malicious, but the modules that have been added to add functionality. For example, [140] argues that not only the SDN applications should be sandboxed, but also some modules of the SDN controller; this prevents an entire network operating system from crashing. Another approach to contain malicious SDN controllers is to use a trust and reputation service [22]. This service requires several SDN controllers that share coordinating tasks and a protocol to select a trusted configuration out of all their different configurations.

**Malicious Hypervisors and Managers.** Finally, some proposals consider that hypervisors and managers may also be compromised [22, 110, 147]. This type of adversary can compromise the whole operation of an SDN/NFV platform.

The most common deployment for this adversary is multi-tenant, both single-domain and multi-domain. This is the only case that considers a multi-tenant multi-domain deployment; we argue that this happens because this kind of deployment provides the only scenario where malicious hypervisors and managers may appear, as there are several infrastructure and NFV providers, as well as various network service clients. In other kind of deployments, it is expected that clients will trust domain management tools and the underlying infrastructure.

The works in [110, 147] consider scenarios where hypervisors try to break the confidentiality of VNFs running in the same machine they control. The work in [110] considers a hypervisor or NFVI that tries to gain access to the policies being used by the VNFs in the infrastructure. They address this problem using cryptography to protect the privacy of outsourced network function policies from the cloud, other tenants, and third parties. Another approach [175] argues that the NFVI Manager and the MANO module should form a Trusted Computing Base (TCB): the NFVI Trust Platform (NFVI-TP). The NFVI-TP would also guarantee trustworthiness of virtualized functions that offer critical security operations like key generation and storage, and ciphering. The work in [147] assumes the same kind of malicious hypervisor and proposes the use of specialized hardware to create protected memory pages that VNFs may use to securely store sensitive information.
We also found that the most considered security goal is availability; it is reasonable as one of the key tasks of SDN/NFV platforms is to ensure connectivity of all flows going throughout the platform, and availability of supported services.

Integrity, confidentiality and availability are partially supported by access control, and this is the second most considered security goal. Access control in SDN is approached by works like FortNOX [135] and SE-Floodlight [136]. Both approaches extend the SDN controller to mediate requests sent by SDN applications and allow or reject those requests according to a previously defined policy. Both proposals use similar characteristics: (1) they both use the controller as a policy enforcer, (2) policies are based on a role hierarchy, and (3) administrators assign roles to SDN applications. SDNShield [169] propose a policy language, compiler, and reference monitor to enforce permissions on SDN applications, inspired by Android manifests with Android applications.

Access control for SDN has also been studied in other proposals; For example Wan et al. [168] identifies operations that should be controlled via permissions to be able to implement the minimum privilege principle, and Ropke et al. [140] mediates critical operations with a function that checks whether a caller is authorized to access a critical operation or not. This enables the creation of access control policies for Network Operating System (NOS) components and for SDN applications, increasing the resilience of the NOS in case of failures or compromised parties. In this type of proposals, the enforcement point is not located at the SDN controller, but on the hypervisor or the element that is in charge of hosting/managing the SDN controller.

More importantly, access control in NFV and SDN/NFV has not been explicitly addressed, except for brief mentions in Congress [15] and Moon [70]. Congress considers access control policies but the subjects of these policies have not been defined yet. Moon aims to build a security management layer for OpenStack and the OPNFV Platform. Moon allows users to create security modules to protect different tenants in OpenStack. These projects include security policies that are enforced by several OpenStack modules (Nova, Swift, and Keystone). One advantage of Moon is that it enables the creation of a centralized security policy and it enforces it across the OpenStack platform. Nevertheless, Moon does not address security properties specific to VNFs yet.

Finally, we analyze related work in security for cloud computing. We consider that NFV and cloud computing share certain elements. First, services are offered by software running on top of virtual machines. Thus, the physical substrate is shared among instances, which arises access control, integrity, and confidentiality issues that need to be addressed. Second, the owner of the application hosts its business logic in an infrastructure that belongs to another entity, creating trust challenges between the parties involved.

Authors in [178] propose Key Policy Attribute-based Encryption to protect user information stored in a cloud. In the proposal, system attributes are associated to a file for encryption and decryption. In a similar way, Excalibur [142] ciphers the data using system attributes and can only be deciphered if the platform trying to read the data has the same set of attributes. A Trusted Platform Module (TPM) seals the information using the platform software stack. Although these proposals enable access control for the files of different users, they only protect the confidentiality of the data. We consider that in NFV access control for the operations needs also to be enforced.

Distributed Information Flow Control (DIFC) is used in [178] to increase the privacy level in multi-tenant environments. Data Flow Control uses labels to represent a privacy or
integrity attribute. Using these labels, policies ensuring certain level of privacy or integrity are enforced. Although DIFC can offer total mediation and a framework to define security policies, its objective is to protect the data integrity or security. We consider that protection is needed in the set of operations a determined NFV can perform.

In [93] different architectures to achieve multi-tenancy in a storage cloud service are presented. The architectures are based on virtual machines at hypervisor level and use mandatory access-control checks in one shared operating system kernel. Nevertheless, we consider that NFV access control should address the operations that VNFs can perform on resources. These operations are not limited to read/write operations; instead, they could affect the network state or create new instances of virtual machines. This is the main difference between Network Function Virtualization and Cloud Computing. Instances in cloud computing mostly offer computing services to tenants, while in NFV these instances offer networking services to create network applications.

5.2.3 Limitations

While some of the discussed proposals address access control, they only consider SDN and do not consider specific characteristics of SDN/NFV. Contrary to SDN, SDN/NFV platforms have heterogeneous resources. While in SDN it is enough to control access to operations over the flows, SDN/NFV platforms also need to control operations on virtual machines and VNFs. Some proposals provide an initial approach to virtual machines operations [70]. Nevertheless, none of the studied works addresses issues related to VNF operations. In addition, the granularity of the proposals may not be enough. For example, for one of the previously defined use-cases, Virtual Network Function Privileges (section 5.1), FortNX cannot handle service chaining because it does not allow multiple VNFs to perform a set of operations on the same resource (a determined flow). SDNShield is a proposal that could handle service chaining if were adapted to the NFV environment. Nevertheless, we consider that their approach is not sufficient to protect SDN/NFV infrastructures. First, we consider that their syntax is complex and does not consider the heterogeneity of resources and operations that may be performed in these architectures. The definition of flow filters, action filters, statistics, and topology filters, does not include other type of resources like virtual machines and instances deployed, these resources also need to be controlled. In addition, due to distributed nature of SDN/NFV environments, multiple enforcement locations are necessary. For these reasons, we consider that a mandatory access control framework for SDN/NFV must provide a two complementary properties: i) a general policy language that allows to describe the diverse resources and operations present in SDN/NFV, and ii) A compiler that translates these policies into security rules enforced across the platform, at different levels, and using different vendors technology.

The rules that integrate the policy that governs behavior are not static; rules change as administrators add new resources, and change or remove old ones. A policy manager must be consistent with this characteristic of the platform: this requires a language to express changes, a module to translate new rules to platform representation and back, and a mechanism to install new rules so they can be enforced. Some works provide policy management for NFV orchestration and management, but they do not provide management of security policies.

Besides, a particular VNF may be used to build different services for two different clients, and while it may be allowed to participate in one, the owner of the other one may decide that
the VNF’s provider is not trustworthy. Thus, SDN/NFV needs a component to consistently handle these type of policies when services are being built. While some works already address trust management both in SDN and NFV, we want to extend their approaches by enabling clients to define service constraints based on trust, i.e., the providers that are allowed to participate in a service.

All the identified requirements should be addressed by a single entity, although its decisions may be executed by other components subordinated to it. Some approaches to secure SDN/NFV platforms like Moon [70] follow this principle. We consider that this is the correct direction: to enable administrators to define policies in a single place and that the platform translates those policies in the appropriate hooks and enforcement mechanisms. Operating systems approach similar problems in a principled and coordinated way, and these principles may be used to guide a security architecture for SDN/NFV platforms.

5.3 New Directions in Mandatory Access Control Systems for SDN/NFV

We envision that access control for SDN/NFV must integrate components at several layers, with a main coordinator, running as part of the Service Orchestration and Management (MANO) Module, making decisions and delegating tasks.

The mechanism should support several tasks: it must enable trusted parties (domain administrators) to define policies to control the set of actions that any software deployed on the platform (SDN and NFV applications) is allowed to perform. These policies must be translated into proper security mechanisms enforced at the required levels in the platform.

This access control mechanism should meet the following characteristics:

- Provide a language that allows administrators to create policies representing the diversity of resources, operations, and users present in the SDN/NFV environment,
- Policies should be created using high level definitions. That is, rules that integrate a policy should express what is allowed rather than how it will be enforced,
- Provide tools to map high level policies to appropriate enforcement points and strategies,
- The access control mechanism should be mandatory and have complete mediation in the SDN/NFV infrastructure over the critical operations and resources.

Operating systems share certain similarities with SDN/NFV platforms, especially considering multi-tenant deployments. In both cases, the infrastructure owner is not the same that deploys software in the platform, also the software is developed and managed by multiple parties, and this software must perform operations that could be considered critical or that have a wide impact on the platform. Considering the similarities between SDN/NFV and operating systems, we argue that secure SDN/NFV environments can obtain valuable insights from access control fundamentals developed by secure modern operating systems. An SDN/NFV architecture enables a variety of network services that must be supported and controlled through an access control mechanism; not all applications running on an instance
of this architecture must have access to all provided services. Considering this, we propose a mandatory access control framework for SDN/NFV. In the following, we extend this definition and explain its components.

In secure operating systems, the general authorization procedure works as follows [167]: A process tries to perform an operation on a specific object. The kernel mediates the request, looks for the labels of the process and the object, and queries a previously defined mandatory access control (MAC) policy to check if it allows the operation for a process and an object with the found labels. If the policy allows the operation then it is performed, otherwise, the operation is rejected. We envision a similar procedure for SDN/NFV environments. To accomplish this, we propose an architecture that implements a reference monitor that should mediate all requests from applications to access resources of the NFV infrastructure (NFVI). Figures 5.4a and 5.4b illustrate similarities between secure operating systems and the proposed secure SDN/NFV architecture. Figure 5.4a shows how secure operating systems deploy mandatory access control and reference monitors. The Reference Monitor and the MAC are part of the operating system kernel. The reference monitor intercepts all operations identified as critical using hooks installed in the interfaces that grant access to critical services. The monitor queries the policy store and the MAC to determine if the operation request should be accepted and answers depending on the access control rule in the policy store. We envision that a similar approach should be followed for SDN/NFV. Nevertheless, this system should have differences with the traditional approach of operating systems due to the diversity and distributed nature of SDN/NFV platforms. First, the tools and language to define access control policies should be offered by the MANO component. Depending on the type of operation, user, and resource being related in the access control rule, appropriate hooks and
enforcement mechanisms should be deployed in the correspondent layer or module handling that entity.

5.3.1 Access Control

In a multi-user environment, it is important to authenticate each user and authorize each request to access resources. The MAC concept introduced in our architecture makes it possible to have authentication and authorization by assigning permissions to specific authenticated users and applications running on their behalf.

As an example, suppose an administrator installs an application to monitor web traffic in a specific network. This application would run on top of several virtual machines across the platform. These virtual machines would have two interfaces, one would be connected to the internal network the application is monitoring, and the other one would be connected to an external domain, for administrative purposes. Although the expected behavior for this application is to only monitor web traffic, an application could also execute other instructions, like creating additional network flows.

To control application behavior, a network administrator would need to define policies and have an enforcement system. An example of policy is: the application only is allowed to receive statistics about web traffic in a particular network. Another policy would constrain management connections to an application by only allowing connections from a specific IP address to a specific port in the server that runs the application.

An administrator would define this kind of high level policies and a policy compiler must translate them into several lower level rules that would be sent to the enforcement points like SDN controllers and NFVI managers. In the example, while the first policy will be translated to a rule to be deployed at the SDN controller in order to restrict access to the flow space, the second policy will be translated to a rule to be deployed at the NFVI manager in order to restrict connections from external domains. After installing the lower level policies in the controllers and managers, new instances of the application will be secure (their behavior will be ruled by the policies). If the same application were deployed on an infrastructure with different SDN controllers or NFVI managers, the high level policies would not change; the compiler translates high level policies to lower level rules according to underlying technology. Figure 5.5 illustrates this behavior. Secure Operating Systems use access control policies that combine Resources, Operations, and Users in order to represent heterogeneous agents, like the presents in Operating Systems or SDN/NFV. In the following, we identify the Resources, Operations, and Users for SDN/NFV environments.

Resources. In SDN/NFV there are three classes of resources: i) computational resources, ii) storage resources, and iii) network resources. NFVI managers handle computational resources (virtual machines with various CPU-RAM-Disk configurations), and storage resources (virtual machine image repositories).

SDN controllers handle network resources including (a) Flowspaces, represented by all the traffic that matches a determined flow descriptor (like network 192.168.254.0/24), (b) Topology, representing current network state (state of links, bandwidth capacity, etc.), and c) Flow statistics, statistics that SDN switches generate and SDN controllers collect.

Operations. Different classes of resources are associated to different operations. Examples of operations on computational resources are create, clone, modify, delete, turn on, and shutdown. Examples of operations on storage resources are copy, delete, and create. Finally,
5.4 Conclusions and Future Work

SDN/NFV environments resemble operating system environments, where multiple applications make use of shared resources to achieve their goals. We discussed these similarities and analyzed the security challenges that SDN/NFV have in this area. We presented an extended SDN/NFV architecture that implements a reference monitor and a mandatory access
control framework. These components enable applications to run on a shared resource platform, ensuring that access the resources follows the policies defined by the administrators.

Further work is required to propose algorithms that can properly resolve conflicts among policies from different applications. The mechanisms used to limit resource distribution among applications and prioritizing certain applications when resources are scarce will also be developed as future work. The current proposal regarding authorization does not consider information-flow based policies, like Biba or MLS. While allow rules enable administrators to assign permissions, on SDN/NFV resources, to applications, they are not enough to forbid flows that are contrary to defined flows, like Biba or MLS policies do because these types of policies require the identification of all possible information flows created by all types of operations allowed in the system, and to detect and deny any possible flow in conflict with policies. A future challenge is how to extended our architecture to offer enforcement for these type of policies.

5.5 Acknowledgments

This work is partially supported by the U.S. Air Force Office of Scientific Research under award number FA9550-17-1-0135, the U.S. Department of Commerce by NIST Award 70NANB17H282 and by the Colombian Administrative Department of Science, Technology, and Innovation (Colciencias).
Chapter 6

Access Control Policies for Virtual ICS Systems

6.1 Virtualized ICS Systems

Industrial Control Systems (ICS) are a set of devices that offer various functions that interact to keep a physical system behaving within a desired set of values. Traditionally, they meet this goal by using dedicated hardware to interact with a physical system at two different levels known as field and supervisory levels. At the field level, sensors, actuators, and Programmable Logic Controllers (PLCs) directly monitor and control a physical process. At the supervisory level, communication equipment, including Supervisory Control and Data Acquisition (SCADA) and Human Machine Interface (HMI) devices, enable human-process interaction. Devices at this level communicate with the physical plant.

Figure 6.1 shows a traditional ICS system and its components. Sensors measure system variables and report their values to the controllers (PLCs), actuators receive commands from the controllers and apply physical signals that adjust the behavior of the system. Controllers run decision algorithms based on data received from sensors to decide what commands should be sent to actuators to keep a physical system within a desired behavior. Controllers can also receive configuration parameters from the supervisory network and report back to this network. In addition, operators at the supervisory network receive information and can adjust the ICS system via SCADA and HMI devices.

We have argued before [133] that this infrastructure may be partially virtualized and deployed in the cloud, bringing capital and operational cost benefits [47, 60, 71, 81]. A virtualized infrastructure can dynamically launch virtual machines that run various control and network functions to ensure that the infrastructure behaves within expected parameters. In addition to control and communication capabilities, this kind of infrastructure can have virtual machines providing security functions to the ICS system [119]. Deploying a virtual ICS infrastructure has the following advantages: i) infrastructure can be dynamically adjusted if control conditions change, without having to modify the physical wiring of the equipment. For example, in the case of SCADA servers, multiple instances could be launched close to the physical process being controlled, thus reducing latency of control and data acquisition processes. ii) Virtualized servers could be dynamically scaled if network conditions change. For example, new instances of an IDS could be launched if malicious activity is detected and
Figure 6.1: An Industrial Control System (ICS) has two types of networks: Field and Supervisory. The Field Network interconnects sensors, actuators, and PLCs. This network has strict real time requirements. The Supervisory Network interconnects operators and Supervisory Control and Data Acquisition (SCADA) servers with the Field Network.

more processing power is required. iii) These virtual infrastructures could run experiments to evaluate ICS conditions and measure the impact of security incident response functions to identified threats.

However, although some ICS components, like SCADA servers and PLCs, can be virtualized and deployed in a cloud-based infrastructure, other components, like sensors and actuators, cannot. Sensors and actuators require special physical interfaces to directly interact with particular physical processes. Based on this difference, we envision an ICS infrastructure with two main zones: a physical zone and a virtualized zone. Figure 6.2 shows both zones; the red zone, at the bottom, groups physical equipment that cannot be virtualized while the blue zone, at the top, groups elements that can be virtualized. We called this ICS infrastructure a virtualized ICS.

A virtualized ICS, following the traditional approach, has a field network and a supervisory network. It could, however, run several instances of the supervisory network; with different configurations, using different data sources to build different views of the field network for different operators. A virtualized ICS environment can also run multiple instances of particular ICS elements for different operators. Also, different actors may consume data from different elements, and may operate components locally or remotely.

To build a virtualized ICS, we used virtualization technology to run a set of virtual functions and interconnect them to build a virtualized ICS layer. In a generic cloud environment, virtual machines provide computational capabilities to users and services, in NFV environments, virtual machines also run Virtual Network Functions (VNFs) to provide networking capabilities. We extended the definition of VNF to include a "Virtual Industrial Control Function" (VICF) in order to name a special set of virtual machines that have specific software to run Programmable Logic Controllers (PLCs), Remote Input/Output Systems
Virtual ICS Layer
Virtual ICS Network
Virtual ICS Network
Virtualization Layer
Sensor and Actuator Layer
Physical System Layer
ICS Operators
OSS/BSS Layer

Figure 6.2: Virtual industrial control system. The Virtualization Layer offers an infrastructure to launch virtual industrial control functions (VICFs) that perform the services provided by PLCs, SCADA servers, and other devices in both the field and supervisory networks.

(RIO), Supervisory Control and Data Acquisition (SCADA), and Human Machine Interface (HMI) equipment. These VICFs are different from traditional VNFs since they have special availability requirements that demand deterministic communication guarantees.

Considering the previously described context, VICFs will be deployed, managed, and used by different sets of users, each of them with a different set of permissions. This feature requires an access control system; the ICS must manage infrastructure resources and allow only authorized users to perform operations over infrastructure resources.

### 6.2 Access Control Framework for Virtualized ICS Networks

Traditional access control systems have three elements that must be organized under specific policies [150]:

- **Users.** Users represent active entities in the system like people or processes. They have credentials that identify them and are assigned permissions to perform operations over system resources.

- **Resources.** The resources represent the objects available in the system like files and programs.

- **Operations.** They represent the operations that users can perform over a given resource, such as: read, write and execute.

To specify system behavior administrators must define rules, that relate users, resources, and operations. For example, a rule may grant a user $U$ permission `execute` over a particular program $P$. An access control policy is the set of rules that govern a particular system.
For a policy to be effective it needs to be enforced. A policy enforcement system is a subsystem that intercepts requests performed by users and queries a system policy database, defined by the system administrator, in order to establish whether the request should be authorized or not.

6.2.1 Access Control for Virtualized ICS

As previously mentioned, we argue that virtualized ICSs require an access control framework to manage virtual ICS resources according to policies defined by administrators. We discuss such framework in this section.

In a virtualized ICS users correspond to operators; people that configure PLC’s setpoints, query state of controlled processes, configure equipment setpoints and measurement parameters, program control strategies on PLC’s, configure network topologies, and react to security or malfunction incidents. Resources correspond to virtual resources present in the system and network configurations; virtual resources are: Virtual Network Functions (VNFs) and Virtual Industrial Control Functions (VICFs) and network configurations are resources or configuration rules embedded in a VNF. Operations represent the actions that can be executed over system resources, such as changing network configuration and launching or deleting VICFs. Additionally, ICS administrators must define access control policies to describe accepted network behavior, and the system must enforce those policies across the whole infrastructure.

6.2.1.1 Resources

In a virtualized ICS, there are three types of resources: VICFs, VNFs, and network configurations.

- Virtual Industrial Control Functions (VICFs) are virtual instances that run specialized software for Industrial Control Systems, for example a PLC. VICFs are different from traditional Virtual Network Functions and Generic Cloud Instances because VICFs demand a real time operating system running in both the instance and the physical host, or a mechanism to provide them with deterministic guarantees regarding message latency. There are already some works offering approaches to meet these requirements in cloud environments [40].

  A VICF can be placed either at the field network or at the supervisory network and is assigned a set of programs that implement an expected behavior. For example, a VICF representing a Remote Input Output (RIO-VICF) would receive commands from a PLC, forward commands to one or several actuator, and report sensor readings to a PLC. A PLC-VICF would implement algorithms to control a physical system and an HMI-VICF would gather information from different sources and report those to the operators.

- VNFs are virtual implementations of traditional network services, they are the main component in a NFV. For example, a VNF could be a virtual machine acting as a router between two networks or an IDS system detecting malicious traffic in the network.

- Network configurations are resources or configuration rules embedded in a VNF, such as network name, pool of IP addresses, and firewall rules. These resources are relevant
because their declaration and configuration establish how the VNFs will route traffic through the network.

### 6.2.1.2 Operations

Operations in a virtualized ICS include environment configuration, management actions over virtual instances, configuration and management of network resources. Examples of configuration actions are create flavor, create zone, and create and update a virtual instance. Examples of management actions over virtual instances are list instance, launch instance, and list flavor. Examples of configuration actions over network resources are create port, update port, create router, and create interface. Examples of management actions over network resources are list network, show network, and list ports.

The set of operations the virtualization layer can execute depend on the type of resource: VICFs, VNFs, or network configurations. The operations performed on VICFs and VNFs are similar, because both represent virtual instances in the infrastructure. These operations are those related to typical virtual machines. Some examples are creation, launching, and cloning of virtual instances. Network resources are related to network configurations, like create a network, update a network, and configure VLAN.

Table 6.2 shows the subset of operations performed on instances in order to deploy, configure, and manage a basic cloud environment; these operations are traditionally offered by virtual infrastructure providers.

### 6.2.1.3 Users and Roles

Users interact with the Virtualized ICS in different ways. Some users are operators, thus they should have access to readings generated by the VICFs in the field network and modify their operation setpoints. Other kinds of users have a limited access and should only be able to read reports generated by the supervisory network, without the option of modifying setpoints. This kind of operations would be show/list operations that do not modify the state of the VICFs or VNFs.

Considering this division of tasks, we decided to use role-based access control. A role is an identity that a user can assume at a certain time, if it is authorized to do so. This identity represents a set of operations that any user with that role will be authorized to execute. [23][78][85]. A given user can assume different roles at different times.

Additionally, we analyzed the identified operations and classified them according to their potential impact on a virtual ICS system. We classified the operations in order to have a metric that enables us to identify higher and lower privileges in a virtual ICS. We are supposing that a role with more privileges would be able to perform more dangerous operations and has a higher level of trust. Operations that only show information without modifying the virtual ICS state, receive a low impact classification, operations that modify, create, or destroy ICS resources receive a high impact classification. Table 6.3 shows the impact level associated to the basic set of operations needed to create, configure, deploy, and manage a basic cloud environment.
| Virtual Instance Operations                                      | List Instances
|                                                                | Show Instance
|                                                                | Create Instance
|                                                                | Delete Instance
|                                                                | Update Instance
|                                                                | Attach Volume to Instance
|                                                                | Detach Volume from Instance
|                                                                | List Flavors
|                                                                | Show Flavors
|                                                                | Delete Flavor
|                                                                | Create Instance Snapshot
|                                                                | Delete Instance Snapshot
|                                                                | List Availability Zones
|                                                                | Show Availability Zones
|                                                                | Create Availability Zones
|                                                                | Delete Availability Zones

| Network Operations                                             | List Network
|                                                                | Show Network
|                                                                | Update Network
|                                                                | Delete Network
|                                                                | List Port
|                                                                | Create Port
|                                                                | Update Port
|                                                                | Delete Port
|                                                                | List Router
|                                                                | Create Router
|                                                                | Update Router
|                                                                | Delete Router
|                                                                | Show Router
|                                                                | List Interface
|                                                                | Create Interface
|                                                                | Delete Interface

Table 6.1: Summary of available operations to handle network resources and virtual instances
<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Instances</td>
<td>Low</td>
</tr>
<tr>
<td>Show Instance</td>
<td>Low</td>
</tr>
<tr>
<td>Create Instance</td>
<td>High</td>
</tr>
<tr>
<td>Delete Instance</td>
<td>High</td>
</tr>
<tr>
<td>Update Instance</td>
<td>High</td>
</tr>
<tr>
<td>Attach Volume to Instance</td>
<td>High</td>
</tr>
<tr>
<td>Detach Volume from Instance</td>
<td>High</td>
</tr>
<tr>
<td>List Flavors</td>
<td>Low</td>
</tr>
<tr>
<td>Show Flavors</td>
<td>Low</td>
</tr>
<tr>
<td>Create Flavor</td>
<td>High</td>
</tr>
<tr>
<td>Delete Flavor</td>
<td>High</td>
</tr>
<tr>
<td>Create Instance Snapshot</td>
<td>High</td>
</tr>
<tr>
<td>Delete Instance Snapshot</td>
<td>High</td>
</tr>
<tr>
<td>List Availability Zones</td>
<td>Low</td>
</tr>
<tr>
<td>Show Availability Zone</td>
<td>Low</td>
</tr>
<tr>
<td>Create Availability Zone</td>
<td>High</td>
</tr>
<tr>
<td>Delete Availability Zone</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 6.2: Operations performed on instance resources and their impact in the virtual ICS

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Network</td>
<td>Low</td>
</tr>
<tr>
<td>Show Network</td>
<td>Low</td>
</tr>
<tr>
<td>Update Network</td>
<td>High</td>
</tr>
<tr>
<td>Delete Network</td>
<td>High</td>
</tr>
<tr>
<td>List Ports</td>
<td>Low</td>
</tr>
<tr>
<td>Show Ports</td>
<td>Low</td>
</tr>
<tr>
<td>Create Port</td>
<td>High</td>
</tr>
<tr>
<td>Update Port</td>
<td>High</td>
</tr>
<tr>
<td>Delete Port</td>
<td>High</td>
</tr>
<tr>
<td>List Routers</td>
<td>Low</td>
</tr>
<tr>
<td>Show Routers</td>
<td>Low</td>
</tr>
<tr>
<td>Create Router</td>
<td>High</td>
</tr>
<tr>
<td>Update Router</td>
<td>High</td>
</tr>
<tr>
<td>Delete Router</td>
<td>High</td>
</tr>
<tr>
<td>List Interfaces</td>
<td>Low</td>
</tr>
<tr>
<td>Show Interfaces</td>
<td>Low</td>
</tr>
<tr>
<td>Create Interface</td>
<td>High</td>
</tr>
<tr>
<td>Update Interface</td>
<td>High</td>
</tr>
<tr>
<td>Delete Interface</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 6.3: Operations performed on network resources and their impact on the virtual ICS
Figure 6.3: Access control framework for virtual ICS. An administrator writes high level policies that specify allowed operations in the virtual ICS; a policy has several rules that specify relations among roles and operations over available resources, and among users and roles. These policies are translated to low level access control rules according to the platform underlying the virtual ICS. When a user tries to perform an operation upon a resource, the enforcement mechanism queries the specified rules to decide whether to authorize or reject the operation.

6.2.2 Policy Engine

The Policy Engine is another component of our access control framework, it includes policy specification and policy enforcement mechanisms.

6.2.2.1 Policy Specification

Using a high level language, virtual ICS administrators may write high level access control policies; a policy defines the roles used in the framework and the resources and operations associated to each role. Additionally, an administrator must associate users to one or more roles; a role grants users with that role particular permissions over particular resources. A policy compiler translates high level policies into low level access control rules. The format of the output rules must be consistent with the underlying technology.

A policy engine can actually translate from a high level language to different underlying platforms; to do so, it would need a way of identifying the different target platforms and use the compiler associated to each specific platform. Figure 6.3 shows the layout of the access control framework.

6.2.2.2 Policy Language

This section describes the language we developed to enable administrators define high-level access control policies. In this language each line represents a statement. The language
has the grammar shown on listing 6.1:

```plaintext
<policy>        := <statement list >
<statement list > := <statement>      
                      | <statement>, <statement list>
<statement>     := <role definition>
                 | <category definition>
                 | <network definition>
                 | <permission definition>

<role definition> := role : <name >(<operation list >)
<category definition> := <name >: <VICF list>
<network definition> := network : <name >(<category list >)
                 | network:(<network list >)
<permission definition> := <name >:<role>(<network list >)

<operation list > := <operation>      
                      | <operation>, <operation list>
operation        := list | show | create | delete | update | attach | detach

<VICF list >     := <VICF>
                  | <VICF>, <VICF list>
<VICF>           := <name >

<category list > := <category>
                  | <category>, <category list>
<category>       := <name >

<network list >  := <network>
                  | <network>, <network list>
<network>        := <name >
```

Listing 6.1: Grammar of our developed high level policy language

A high level policy is a list of statements where each statement either specifies a role, the definition of a category, the definition of a network, or the definition of a permission.

- A role groups a set of operations. A statement that defines a role begins with the reserved word role, followed by the character ",", the name of the role, and finally between parenthesis the operations associated to that role. A role name can be arbitrarily chosen by the policy writer. Table 6.1 lists possible operations.

- A category groups different VICFs that share a logic goal. A statement that defines a category begins with a name (the category name), followed by a character ",", and finally the names of the VICFs to group. The name of the category can be arbitrarily defined by the policy writer. For example, an administrator could want to group all sensors in a field network under the same category.

Categories are useful to simplify the definition of high level policies; they enable administrators to define administration hierarchies and this makes it possible to define rules over categories instead of individual resources.

- A network is composed of a set of categories. A statement that defines a network begins with the reserved word network, followed by the character ",", and then the name of the network. This name can be arbitrarily chosen by the policy writer. Following the name, between parenthesis, are the categories that belong to that network. Note that a
network can be composed either of categories or other networks previously defined in the same policy.

- A permission specifies the role associated to a given user. A permission statement begins with a user’s name, followed by the character ":", the name of a role, and finally between parenthesis, the network over which the role applies.

Listing 6.2 shows a high level policy example. Line 2 defines the role "owner" with the operations "create, delete, modify and show". Line 6 creates the category "sensors" and groups under these category the resources lit101, lit201, ph201, fit201. Lines 7, 8, 11, and 12 also define categories. Lines 9, 13. Line 16 creates the network "virtual_ics" by merging two previously defined networks "supervisory_network and field_network".

Finally, lines 19 and 20 define the operators blue and red with two different roles "owner" and "operator" and assign to these users the virtual_ics network and supervisory_network network respectively.

This means that the blue operator is the owner of the virtual_ics network and hence he or she has permissions to create, delete, modify, or show any resource present in the virtual ICS, while the red operator has only show permissions in the supervisory network. It is relevant to mention that this high-level policy does not depend on the platform used to implement the virtual ICS.

```
# Definition of roles
role:owner(create, delete, modify, show)
role:operator(show)

# Definition of categories for the field network
sensors: lit101, lit201, ph201, fit201
actuators: pl101, mv101, p201
plcs: plc101, plc201

# Definition of the field network
network: field(sensors, actuators, plcs)

# Definition of categories for the supervisory network
scada: scada_1
historian: historian_1

# Definition of the supervisory network
network: supervisory(scada, historian)

# Definition of the virtual_ics network
network: virtual_ics(supervisory, field)

#Definition of users permissions
operator_blue: owner(virtual_ics)
operator_red: operator(field)
```

Listing 6.2: Example of a high-level policy defined using our language

### 6.2.2.3 Policy Language Discussion

This section evaluates the scope of the developed environment (language and mediation extension).
The objective of our policy language is to offer a tool to ease modeling of virtual ICS systems and role management in order to handle permission assignment to operate a given infrastructure. The proposed language enables virtual ICS administrators to simply describe an ICS topology; systems with various components (sensors, actuators and PLCs) and different interconnections. It also enables administrators to combine topology with Role Based Access Control (RBAC) to define high level access control rules. In addition, users will have a clearly defined role to perform a set of operations at a certain time. For example, a user could be given a role to only have modification permissions over a PLC to reconfigure it during maintenance windows. That very same user could have a very limited role that only grants view permissions over the same set of PLCs at a different period of time.

Nevertheless, there are some permissions that would be impossible to define with the current version of our policy language. First, virtual administrators must be very careful giving two different roles to the same user as this may generate policy conflicts; resource permissions may be different for two different roles. An algorithm for the resolution of such conflicts is out of the scope of the present thesis. Second, a resource should only belong to a network or category, if a resource belongs to two network or categories, conflicts may arise because of the permissions assigned to that network or category.

6.2.2.4 Policy Enforcement

The policy enforcement module intercepts the requests generated by the users and queries a previously defined policy in order to accept/deny those requests. In the case of a virtual ICS, this component is composed of modules running at every one of the virtualization platforms available across the entire infrastructure. By placing enforcement modules at the deployment level of the virtual resources, we reduce latency; enforcement is applied at a point close to the place where operation requests are created.

6.3 Implementation

We implemented a proof of concept using OpenStack Stein version (3.18.0)\footnote{Stein installation procedure, release notes, and other documentation available at: \url{https://docs.openstack.org/stein/}}, we used the Devstack\footnote{Devstack is a complete OpenStack environment that can be installed on a single machine for development purposes: \url{https://docs.openstack.org/devstack/latest/}} installation method to deploy it in a single server. OpenStack is an open source cloud service provider that provides the Network Function Virtualization Infrastructure presented in chapter 5. OpenStack has several components that offer particular cloud services. For example, the Keystone service manages identity and credentials of all the users that interact with the cloud, and the Nova service handles virtual machines instances and the Neutron service handles network connectivity in the cloud. In addition, OpenStack uses the Oslo service to mediate the operations performed by the users in their projects. Oslo mediates all operations performed with all the services, but OpenStack requires the definition of an independent set of policies for each service. This is one of the contributions of our proposal, we enable the definition of a high level access control policy in a single file.
Our proof of concept focused on testing the capabilities of our language and policy engine to mediate user actions and allow or reject them according to user’s role and its permissions. In this sense, we did not deployed a full NFV environment that includes a MANO component or an SDN controller to manage the network. These systems could be integrated into our system; a discussion about these components is present in chapters 2 and 5.

OpenStack has two different ways to implement access control policies. The first one is to implement policies directly in the code of the services [127]. Nevertheless, this method makes it difficult to manage, change, and update policies because it implies code modification. The second method uses a YAML or JSON file called "policy". In this case an administrator must define a policy file for each service that is going to be mediated. The access control rules defined in the policy file override those defined in code and this file is checked every time any user performs an API call (i.e. performs an operation with an OpenStack service).

```plaintext
# Rule for cloud admin access
"context_is_admin": "role:admin"

# Rule for resource owner access
"owner": "tenant_id:%(tenant_id)s"

# Rule for admin or owner access
"admin_or_owner": "rule:context_is_admin or rule:owner"

# Rule for admin-only access
"admin_only": "rule:context_is_admin"

# Get `provider:segmentation_id` attribute of a network
# GET /networks
# GET /networks/{id}
"get_network:provider:segmentation_id": "rule:admin_only"

# Update a network
# PUT /networks/{id}
"update_network": "rule:admin_or_owner"

# Update `segments` attribute of a network
# PUT /networks/{id}
"update_network:segments": "rule:admin_only"
```

Listing 6.3: Example of a high-level policy defined using our language

Listing 6.3 shows an example of a policy file. Every line represents a rule that is evaluated and becomes "true" or "false"; if the result is true the operation is allowed, otherwise, it is denied. In addition, everything that is not a reserved word or an API call is considered an alias. For example, the first line creates the rule "context_is_admin" that queries the "role" attribute of the user trying to perform the operation and tests if it is equal to "admin". The second rule shows that the Oslo service receives some user attributes; in this case the attribute "tenant.id" is used to test if the user is the owner of the resource that is the object of the call. Finally, when the term on the left (of ":") is an API call, the service defines a rule for such operation; in the line for the "update_network" API call the operation is approved if the user performing the API call is either and administrator or the owner.

Nevertheless, the traditional way Oslo defines permissions is rather restrictive; Oslo rules either grant or deny permissions based on operations. For example, for the "launch_instance" operation, users can launch any instance that they own. We require, however, more options than only "owner"; for example, an ICS operator that owns all the virtual machines on a field
network may need to allow an operator $A$ to control a subset of instances, while allowing another operator $B$ to control a different subset of instances. In [6.2.2.2], we have two networks: supervisory and field, and the red_operator only has access to the supervisory network. This use case cannot be specified using traditional Oslo access control rules; we need to extend the way access control rules are expressed or enforced in OpenStack.

### 6.3.1 Oslo HTTP Checks

![Policy engine implementation](image)

Figure 6.4: Policy engine implementation. The OpenStack oslo service sends HTTP requests to an HTTP Server that implements our Policy Engine. The Engine uses the high level policies to authorize requests.

To extend the OpenStack authorization mechanism to include our policies, we modified the policy.json files for every one of the OpenStack services adding an entry that defines the query to the HTTP Server.

The oslo policy service has various types of checks that can be used in a given statement. These checks are: against user attributes, against aliases or other rules, or against an HTTP server. The first two checks do not allow us to express the kind of access control rule that we require. The HTTP server, on the other hand, enabled us to express and enforce more complex rules.

Thus, we modified the policy.json files and added terms that redirect the request to our HTTP policy server giving our policy engine the power to decide if the operation should be authorized or not. Note that in our specific case, we added the term with an "and" operator to also allow the default rule to be enforced, but a system administrator could grant all authorization control to the framework. Figure 6.4 shows the architecture of the implementation.

Listing 6.4 shows the configuration for the operation "update_network". The statement has three terms:
The first term, before the ":" character, defines the name of the API call that is being requested by the user.

The second term, before the "and" operator, is the default rule defined for that API call.

The third part, after the "and" operator, is the term that we added to redirect the request to the HTTP server. This term is an URL with the address and port of the HTTP server. The request PATH has been configured as follows: the first term "update" shows the type of operation that the API call refers to. The second term "network" shows the type of resource being used by the operation.

```
"update_network": "rule:admin_or_owner and http://157.253.199.73:8080/update/network"
```

Listing 6.4: Modified Rule entry in OpenStack policy.json

The HTTP request includes user credentials, details about the resource the user is trying to interact with, and the attempted operation. With this information, our HTTP server can check its access control rules (compiled from the high level policy defined above) and authorize/deny the request.

### 6.3.2 Policy Server

We implemented the HTTP policy server using the python HTTPServer, JSON, and urlparse modules. HTTPServer is the basic module for building an HTTP Server in Python. Nevertheless, we implemented this server as a concurrent HTTP server. JSON Python module is a parser for JavaScript Object Notation a lightweight data interchange format. Urlparse Python module allows to parse URLs to ease processing in Python scripts.

We implemented our VICFs using a virtual machine image running Ubuntu 14.04 cloud and a modified version of MiniCPS [10] that does not require Mininet. As we explained in 6.3, MiniCPS enables the emulation of ICS services and implements support for EtherNet/IP, an industrial network protocol commonly used in ICS for communications between supervisory and field networks.

Figure 6.3 shows our policy engine. First, the HTTP server parses a policy file that must follow the established rules, as the example 6.2 shows. Based on the policy file, the server builds a database of roles, resources, and operations.

Later, at run time, the server receives requests from the enforcers, and based on the values associated to that request; path, user credentials, and target information; decides if the request must be authorized or not.

Our implementation is available at github: [https://github.com/ComitUniandes/VICS_Policy_Engine](https://github.com/ComitUniandes/VICS_Policy_Engine)

**Example.** Figure 6.1 shows a use case. This example has two users: blue user and red user. The blue user is the infrastructure owner, and the red user is the field operator. Since the blue user is the owner, he can build and modify the virtual ICS topology. The red user only has read access to resources in the field network. The historian and SCADA servers belong to the supervisory network, while the physical plant, PLCs, sensors and actuators belong the field network.
<table>
<thead>
<tr>
<th>Role Name</th>
<th>Role Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure Owner</td>
<td>Creates and configures the VNF and VICF present in the infrastructure</td>
</tr>
<tr>
<td></td>
<td>Connects this virtual functions using a network topology</td>
</tr>
<tr>
<td>Field Network Operator</td>
<td>Receives information from VICFs present in the field network layer</td>
</tr>
<tr>
<td></td>
<td>Only has permissions to list/show the resources in the field network</td>
</tr>
</tbody>
</table>

Table 6.4: Roles examples and their description

To implement this use case in our framework, we identified two roles; Table 6.4 describes these roles. The blue operator has access to all the resources available in the topology and can perform the following operations: create, delete, modify, and show. The red operator only has permissions to show information about the resources in the field network.

We also defined the high level policy shown in listing 6.2 to specify the corresponding access control rules.

- Lines 1 and 2 of listing 6.2 define these two roles. The owner role has the operations: create, delete, modify, and show and the operator role only has the show operation.

- Lines 6, 7, 8, 14, and 15 create categories for different set of VICFs present in the topology. Using these categories, we created the networks "field", "supervisory", and "virtual_ics" in lines 11, 18, and 21 respectively. Note that the network "virtual_ics" is composed of the networks "field" and "supervisory" and that using a few lines we modeled the whole ICS topology.

- Finally the lines 24 and 25 define the permissions for the red and blue users. Line 24 gives the user "operator_blue" the role "owner" over the virtual_ics. This means he can create, delete, modify, and show all the resources for that network. Line 25 gives the user "operator_red" the role of "operator" over the "field_network", meaning he only has permission to perform show operations on that network resources.

6.3.3 Evaluation

Mediation analysis.

Our implementation uses the OpenStack Oslo service and, as a consequence, it can mediate all API calls (operations) that Oslo mediates. Nevertheless, an ICS administrator could decide to only mediate a set of specific API calls by modifying the original policy.json files. Listing 6.22 shows that our policy is simple, yet descriptive and can be used to describe the components present in a virtual ICS. Finally, our policy language enables an user to have different roles, applied to different resource groups.

High Policy Language compared with OpenStack Oslo

Although the OpenStack Oslo service provides complete mediation over all the operation requests performed by the users, the Oslo language is rather restrictive. For example, an user
with a determined role would have granted permissions over certain operations in all virtual machines available in the OpenStack infrastructure. This means that with the Oslo language is impossible to grant permissions to only a subset of virtual machines. Our language makes possible to simply define this kind of policies. In addition, our language eases the way the elements in an ICS topology are described by grouping different resources in categories.

**Performance.** We run performance tests in our HTTP Server Policy Engine. We run three types of tests; to measure load time, single query time and concurrent query time. All the tests were run on a machine running OpenStack Stein version (3.18.0) with an Intel(R) Core(TM) i7-4770, (3.40GHz) processor, and 16GB of RAM.

- **Policy Load time.** The first test, measured the time it took the server to parse a policy file and build the set of rules. For these measurements, we created various policy files, with sizes of 1 rule, 10 rules, 100 rules, 10000 rules, 10000 rules, and 100000 rules. Figure 6.5 shows the results; we found that time to parse and build the rules grows linearly with the size of the policy file. In addition, this process should only take place each time the policy file is updated and this should not be very frequent.
Figure 6.5: Policy load time. Error bar showing the parsing time of a high level policy file. The parsing time of the policy file increases linearly with the number of rules in the policy file.

Figure 6.6: Single request response time. Error bar showing the response time for a single request.

- **Single Request Time.** The second test measured time to answer a request performed by a user, depending on the size of our policy file. We used the same policy files sizes used in the previous test. Figure 6.6 shows the results. This test shows that request response
time grows linearly with a base time of 1.2 seconds. Nevertheless, the increase in time is of only 200ms using a policy file with 100000 rules. Using data from two popular ICS testbeds: SWat [108] and the Lancaster testbed [62] the number of VICFs on an ICS system would be between 30 and 70. This means that a typical policy file would be below the 1000 rules, assuming 10 users in the virtual ICS.

- **Concurrent Response Time.** This test measures time to answer a set of concurrent requests performed by several users at the same time. For this test, we varied two variables: policy file size, like we did in the previous tests, and for each of those policy sizes we used 25, 50, and 100 concurrent requests. Figure 6.7 shows the results. The impact of policy size is minimum and, as expected, increase in response time is due to the higher number of concurrent requests. Our server crashed when we tried to use concurrent requests for a policy file of 1000000 rules. We left as future work implementing optimizations to deal with high loads.

![Figure 6.7: Error bar showing the response time for concurrent requests with different policy file sizes. The increase in response time is mostly due to the number of concurrent requests.](image)

From the tests we can conclude that the size of the policy file does not greatly impact response time of our policy engine. The engine should only load a policy file when that file is updated, a situation that should not happen very often. In addition, the response time of concurrent requests increases linearly as the number of concurrent requests increase, which is an acceptable and common behavior for these systems.
6.4 Related Work

Although the concept of an ICS with elements hosted in the cloud has been studied before [47, 81, 173], those previous works did not consider an access control model for such environments. Authors in [173] present a Cloud Operating System Design for industrial applications. Following a similar approach, authors in [47] present a virtual supervisory network implemented in OpenStack to remotely monitor a simulation of an Electrical Power Plant; these works shows the possibility of using virtual networks to implement supervisory networks; However, these works did not discuss mechanisms to protect applications that operating system would run.

[81] presents a cloud-based feedback control system that uses multiple instances of a controller on different regions of a cloud, offering some redundancy capabilities. The redundancy resulting of having a virtual infrastructure is one of the benefits of deploying and ICS system in a cloud. The work presented in [58] also uses the redundancy capabilities of a cloud environment to provide multiple links to route ICS information, this can be useful if some of the links were broken. In addition, authors in [71] discuss the challenges and benefits of implementing ICS in cloud environments. They also discuss issues regarding latency variability to design control algorithms and propose a mitigation mechanism to establish a deterministic maximum latency. Finally authors in [156, 181] present some virtual testbeds to perform experiments on ICS.

All the previously mentioned works analyze benefits of virtualization to provide security benefits in ICS contexts. However, they do not consider access control (AC) models in ICS contexts; AC needs to be discussed and implemented because it is a key element to protect the applications that run in this environment.

The subjects of access control and policy management for cloud and NFV environments have also been addressed [6, 12, 39, 105, 138]. Authors in [32] discuss a dynamic access control model based on trust; we did not use trust assumptions in our access control framework because trust assumptions for ICS environments need to be very conservative. An attack could compromise various elements of a given organization and perform harmful operations to the system. Instead of this, we designed a framework based on roles; a given role groups permissions to perform specific tasks in the ICS. Authors in [121] show a dynamic policy manager that translates high level security policies into low-level policies for NFV environments. The framework uses events to update the low level security policies. The low level policies are security functions activated in response to events detected by the framework. We consider our proposal as complementary; we use dynamic network security functions [119], however we also extend such environment with an access control framework. In [51] the authors discuss a mechanism to monitor privacy policies in cloud environments where various functions are called to attain an objective. Our work is not reactive, but proactively enables the deployment of access control policies to manage a virtual ICS.

Banse et. al. [170] discuss the use of the TOSCA policy language to describe security policy functions; we use a more usable language that in addition enables us to represent a virtual ICS topology. In addition, our language does not require to install additional packages or modify the environment configuration files. With TOSCA representing ICS topologies would not be possible; however, representing the topology using an access control language helps
to define policies that are consistent with the ICS topology.

PEaaS [17] defines a mechanism to map security policies into security mechanisms with different properties. High level policies are processed using Natural Language Processing (NLP) to map them into specific security mechanisms. We also designed our policy language with administrators and operations in mind, we want users to be able to express, in a simple way, the components present in a virtual ICS and the allowed operations.

6.5 Conclusions and Future Work

We presented an access control framework for Virtual Industrial Control Systems and implemented a proof of concept using OpenStack. As part of the framework, we developed a high level policy language to enable administrators describe their policies, ICS components and the set of authorized operations.

We used Role-based Access Control (RBAC) because of the simplicity in the definition of high level access control policies; that makes it easier for ICS administrators, probably users without extensive training to specifically teach them to think about access control, to define access control rules.

The architecture of the framework may be extended to include more compilers to translate high level rules that are platform independent to low level rules directly related to the underlying platform. Our policy language and implementation allow administrators to describe a wide set of policies that may be used to specify and enforce access control in a virtual ICS. These policies enable administrators to dynamically extend an ICS infrastructure, according to actual needs, and create a combination of permissions for all the operations and resources available in the network.

We left as future work deployment and tests of VICFs running real time operating systems and time-aware applications to answer to latency requirements. Virtualized cloud-based ICS would need to run specially tailored real time operating systems in their physical and virtual machines. In addition, these virtualized environments could use SDN controlled channels to guarantee a deterministic communication latency for both field and supervisory networks [40].

ICS real time requirements require that virtual ICS include technologies that can offer deterministic guarantees in the virtual communication links. Two technologies are key to achieve this: fog computing [35,44,46,143] and time sensitive communications [26,50,72,141]. Fog computing is a communication paradigm for Internet of Things (IoT) and Industrial Internet of Things (IIoT), This paradigm is similar to cloud computing, but tries to bring the computational resources closer to the process being controlled, potentially reducing the latency in the communication. We consider that for virtual ICS Fog Computing can provide a communication infrastructure to bring the virtualization servers closer to the physical processes of the ICS. Nevertheless, there are some differences between using Fog Computing for Internet of Things and for Industrial Control Systems. First, the data volume generated inside an ICS is not as big as the one created in an IoT scenario; this reduces scalability requirements. Second, incident response strategies are orchestrated in a different manner because used strategies need to be thoroughly tested before deployment; the impact on an ICS system needs to be assessed to guarantee that the system either recovers com-
pletely from the attack or gets to a safe state with safety defined in terms of the protection of human resources, factory resources, and the product being processed. As a consequence, orchestration recipes that implement strategies must be defined beforehand. We left deployment and evaluation of these ideas as future work. Time Sensitive communication is a new IEEE Communication Standard that offers protocols for OSI layer 2 communication with real-time guarantees. This standard extends on ethernet capabilities to provide deterministic communications. We consider that this kind of protocol would be fundamental to virtual ICS.

6.6 Acknowledgments

This work is partially supported by the Colombian Administrative Department of Science, Technology, and Innovation (Colciencias).
Chapter 7

Conclusions and Future Work

ICS administrators and developers can use virtualization technology to develop more resilient industrial control systems. We explored and evaluated how SDN and NFV capabilities can be combined to automatically start virtual security functions and dynamically change ICS network configuration to include and remove ICS components, redirecting traffic through them, to mitigate impact of security incidents. To evaluate SDN and NFV capabilities to improve ICS security, we developed the following research activities:

1. *Understanding the real value of SDN and NFV technology to mitigate malicious attacks in ICS contexts.* We designed and developed a prototype that uses SDN to enable automated execution of incident responses to several attacks. We used MiniCPS to evaluate results in three use cases; in these cases attackers compromised different ICS elements: a sensor, a PLC, and the SCADA respectively. Our defense was an intrusion protection system with several components:

- An IDS that compares a global view of an ICS against a model (including physical and control strategies) to detect anomalies; we used SDN to build a global view of the system.

- The NFV orchestrator launches different virtual incident response functions according to the identified anomaly; NFV technology makes it possible to dynamically launch these functions.

- The SDN controller dynamically reconfigures network to isolate or reroute malicious traffic, or to connect the previously mentioned NFV functions to the ICS network.

Our experiments show that these intrusion protection systems, based on SDN and NFV technologies, are effective tools to increase ICS resilience in presence of integrity cyber-physical attacks. In addition, using these defense mechanisms does not require administrators to modify ICS, network, or communication protocols.

In addition, ICS administrators may use NFV to deploy virtual industrial control systems. In these scenarios, virtual instances can perform functions traditionally executed by network equipment, PLCs, Remote Input/Output Systems (RIOs), SCADA and historian servers. Also, administrators may use SDN to dynamically reconfigure an ICS network or launch virtual
incident response functions. We deployed such virtual environment to represent different plant models with varying degrees of complexity and evaluate different ICS behaviors.

Regarding this topic, we published two papers:


Our code is available at: https://github.com/ComitUniandes/ICS-SDN/tree/master/paper-topo

2. Understanding possibilities of virtual testing environments. Testing environments are useful for ICS administrators because they make it possible to test beforehand ICS behavior in presence of various components or attacks. These environments also allow administrators to test mitigation capabilities of different incident response functions. In addition, virtual environments are more economic than real environments, they can represent ICS behavior without the costs of using ICS physical equipment. We explored the capabilities of our extended MiniCPS environment to emulate a network control system controlling a nonlinear plant; we found that this emulation behaves close to behavior of simulations and physical equipment. Also, we used MiniCPS to run integrity attacks and defenses and evaluate their impact over the system. We found that the behavior of the devices with our defense, while the system in under integrity attacks on the sensors, is close to normal behavior. We found that virtual testing environments, such as MiniCPS, can effectively be used to run experiments and evaluate mitigation capabilities of security response functions.

Regarding this topic we published the following paper:


Our code is available at: https://github.com/ComitUniandes/ICS-SDN/tree/master/francisco-topo

3. Understanding access control requirements. We argue that SDN and NFV environments are akin to operating systems, because multiple users and applications share physical and virtual resources to achieve their goals; this shared environment creates access control challenges. This is specially important if these environments are being used to create a Virtual Industrial Control System as this kind of environment usually involves critical infrastructure. These architectures, virtual ICS based on SDN and NFV technologies, will be used by
multiple users with different privilege levels.

Regarding this topic we published the following book chapter:


To answer to the issue presented by the previous paragraph, we also designed an access control framework for Virtual Industrial Control Systems and implemented a proof of concept using OpenStack. This framework has a high level policy language that enables administrators to write policies that specify access control rules for different users. It uses an RBAC approach to make it easier for administrators to assign permissions to users (through roles). In addition, our language enables administrators to create groups of resources making it possible to assign role permissions over sets of resources. We used OpenStack to implement a proof of concept and ran different tests to evaluate functionality. We also ran tests to evaluate the overhead generated by the framework when an operator tries to change configuration of any component and found that it is not significant. Having a virtual ICS can bring operational and capital cost benefits to ICS systems and enable them to dynamically react to cyber security incidents.

Our code is available at:
https://github.com/ComitUniandes/VICS_Policy_Engine

During the development of the previously presented work, we also found additional paths of research and left several ideas as future work. The following paragraphs present those ideas.

First, ICS security may benefit from running distributed incident response functions. These incident response functions are launched once an Intrusion Detection System (IDS) identifies an anomaly. In a centralized SDN each LAN is connected to another LANs using an SDN switch and one SDN controller makes all forwarding decisions. Whenever a new flow arrives, the SDN switch forwards the first packet of the flow to the central SDN controller and it installs a forwarding rule in the SDN switch. In addition, an IDS receives a copy of that packet to maintain information about network state and also monitor a portion of the traffic exchanged in each control loop LAN. A centralized SDN controller does not require distributed databases to store network state. Nevertheless, this approach has the disadvantage that all new flows should be forwarded to the central SDN controller increasing its load. POX and OpenDayLight Controllers use a centralized approach. In a distributed approach each LAN network is connected to a different instance of the SDN Controller. When an SDN switch receives the first packet of a new flow, the switch forwards this packet to the corresponding controller instance and that instance decides how to handle that flow. The state of the network is kept using a distributed database but consistency should be addressed in order to avoid network loops or black holes. Currently, some SDN Controllers, like ONOS, offer such possibility. Using this approach an IDS instance could be present in each LAN. The distributed database
may be shared by the SDN controller and IDS instances to detect wide-area attacks. Finally, SDN controller instances need an SDN Application Manager to enable operators to activate/deactivate specific SDN applications or obtain snapshots of the current network state. Such kind of manager shares many features with SCADA systems present in ICS networks. This approach creates more resilient systems by avoiding central points of failure and having faster response times because all forwarding decisions are locally made. The disadvantage of this approach is a higher complexity in the SDN controller software and IDS implementations.

Second, Virtual ICS platforms must start considering real time requirements. Consideration of real time requirements would allow developers and researchers to build more realistic scenarios, and have a better assessment of attack impact over industrial control systems and their plants. To follow this line of work, developers would need to run real time operating systems at the virtual machine instances, as well as at the host and hypervisor levels. For an hypervisor to offer real time guarantees, it should support real time answers when controlling operations from the guest and real time communication interfaces.

Third, We may use ideas from Fog Computing to address ICS real time requirements. ICS real time requirements are somewhat similar to the ones addresses by Fog Computing in IoT contexts [35,44,46,143]. Systems with real time requirements must have components with a reduced and deterministic latency, therefore, bringing computational capabilities closer to the physical systems is a good approach. Nevertheless, there are some differences between using Fog Computing for Internet of Things and for Industrial Control Systems. First, data volume generated by an ICS is not as high as the one generated by an IoT system; this feature reduces scalability requirements. In addition, incident response strategies for virtual ICS must be thoroughly tested before deployment; their impact has to be assessed to guarantee that the system either recovers completely from an attack or goes to a safe state with safety defined in terms of the protection of human resources, factory resources, and the product being processed. We left deployment and evaluation of these ideas as future work.

Finally, Policy conflict resolution algorithms for virtual ICS access control frameworks. Further work is required to propose algorithms that can properly resolve conflicts among high level policies defined on the same set of resources. Conflicts could arise if one administrator defines conflicting access control rules related to the same category or network. For example, one access control rule could grant to an user a role that enables the modify operation and a resource and another access control rule could also grant to that user a role with only the show operation on the same resource. These algorithms could use a priority classification of the roles present in the high level policy.
Appendix A

Publications


Abstract: The integration of modern information technologies with industrial control systems has created an enormous interest in the security of industrial control, however, given the cost, variety, and industry practices, it is hard for researchers to test and deploy security solutions in real-world systems. Industrial control testbeds can be used as tools to test security solutions before they are deployed, and in this paper we extend our previous work to develop open-source virtual industrial control testbeds where computing and networking components are emulated and virtualized, and the physical system is simulated through differential equations. In particular, we implement a nonlinear control system emulating a three-water tank with the associated sensors, PLCs, and actuators that communicate through an emulated network. In addition, we design unknown input observers (UIO) to not only detect that an attack is occurring, but also to identify the source of the malicious false data injections and mitigate its impact. Our system is available through Github to the academic community.


Abstract: In the past decade the security of industrial control systems has emerged as a research priority in order to safeguard our critical infrastructures. A large number of research efforts have focused on intrusion detection in industrial networks, however, few of them discuss what to do after an intrusion has been detected. Because the safety of most of these control systems is time-sensitive, we need new research on automatic incident response. In this article we show how software-defined networks, and network-function virtualization can facilitate automatic incident response to a variety of attacks against industrial networks. We also prototype an incident response solution that detects and responds automatically to sensor attacks and controller attacks. Our work shows the promise that cloud-enabled software-defined networks and virtual infrastructures hold as a way to provide novel defense-in-depth solutions for industrial systems.

Leveraging Software-Defined Networking for Incident Response in Industrial Control Systems (Dec
Abstract: In the past decade, the security of industrial control systems has emerged as a research priority in order to safeguard our critical infrastructures. A large number of research efforts have focused on intrusion detection in industrial networks; however, few of them discuss what to do after an intrusion has been detected. Because the safety of most of these control systems is time sensitive, we need new research on automatic incident response. This article shows how software-defined networks and network function virtualization can facilitate automatic incident response to a variety of attacks against industrial networks. It also presents a prototype of an incident-response solution that detects and responds automatically to sensor attacks and controller attacks. This work shows the promise that cloud-enabled software-defined networks and virtual infrastructures hold as a way to provide novel defense-in-depth solutions for industrial systems. This article is part of a special issue on Software Safety and Security Risk Mitigation in Cyber-physical Systems.


Abstract: Network Functions Virtualization (NFV) and Software Defined Networking (SDN) improve network capabilities by enabling the deployment and control of network functions using software instead of hardware-specific middleboxes. This programmability enables the development of new software services designed to meet a growing list of network requirements. Nevertheless, this same flexibility and programmability can facilitate malicious behavior by attackers with partial access to the SDN and NFV management infrastructure. In this chapter, we discuss security challenges that emerge in an SDN/NFV environment and analyze the main proposals aimed to secure SDN and NFV platforms. In addition, we illustrate the similarities with the requirements that secure operating systems address. Inspired by some of the best-practices and lessons learned in the design of these systems, like reference monitors, mandatory access control, and policy verification, we argue that such principles can be used to define a standard SDN/NFV security architecture to facilitate the design and management of SDN/NFV applications.


Abstract: Most Denial of Service (DoS) attacks intend to generate a traffic pattern that is indistinguishable from legitimate traffic, making it hard to detect an attack. Conventional defenses for these attacks are not scalable, are slow to react or introduce an overhead to each routed packet. In this paper, we present FlowFence, a lightweight and fast denial of service detection and mitigation system for Software Defined Networking (SDN). The FlowFence architecture includes routers running daemons to monitor the average occupation of their interfaces to detect congestion conditions, and an SDN controller that coordinates bandwidth assignment of controlled links. The controller limits the flow transmission rate along a path to prevent users’ starvation. The mitigation procedure of starvation state allocates an average bandwidth, while flows exceeding the mean are penalized. The penalization is proportional to the difference between the fair limit and the current bandwidth usage. A system prototype
was implemented and evaluated in the Future Internet Testbed with Security (FITS). The results show that the proposal avoids users’ starvation of network resources without adding much overhead in the network.


*Abstract:* Software Defined Networking (SDN) emerges as an option to implement security features difficult to develop and deploy in traditional network infrastructures. SDN has a programmable component that can build a global view of the actual state of a network and change network configuration to react to actual events: a controller. Additionally, a controller’s functionality may be extended to meet specific requirements. This work studies the features that Floodlight, a Java based SDN controller, offers to extend its behavior. Previous works have studied Floodlight architecture and performance, but not these features. To meet the goal, we selected a known security context for traditional networks: DDoS detection and mitigation. This paper presents design and implementation of the CDM (Collection, Detection, and Mitigation) module, a statistical-based DDoS detection module that extends Floodlight. Statistical algorithms are a good fit for SDN; they have low memory and CPU demands, and can react to changes in network configuration. The module also uses Java features to establish an interface for statistical-based detection algorithms, enabling administrators to use libraries of algorithms and select some of them according to their systems. The results show that Floodlight is easy to extend and flexible. It is also efficient regarding CPU, but requires more memory than other controllers. The collection, detection, and mitigation algorithms run fast, although the time window required to detect statistical change bounds reaction times.

111
Bibliography


115


