Master in Informatics Engineering
Dissertation
Final Report

Security Probes for Industrial Control Networks

Jorge Filipe Barrigas

jorgefb@student.dei.uc.pt

Academic Advisor:

Prof. Tiago José dos Santos Martins da Cruz

Date: July 01, 2014

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Acknowledgements

I would like to express my special appreciation and thanks to my advisor Professor Tiago Cruz for his excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing research.

I would also like to thank the LCT team and Professor Dr. Paulo Simões for all the support in my research.

Finally, I would also like to thank my parents. They were always supporting me and encouraging me with their best wishes.

Summary

During the last years, some of the attacks towards SCADA made headlines, helping raise awareness. Some of the vulnerabilities may even be exploited to harm other systems, eventually, ending compromising specific points of SCADA networks. At the same time, a growing number of threats related with the physical layer made new issues and low-level mechanisms emerged, requiring careful approach and monitoring to secure such devices. Threats of this type are considered to be even more dangerous than SCADA-specific ones, since these operate at a lower level, unnoticed to scanners and anti-viruses. Considering this scenario we can expect a growing number of these threats to be affecting SCADA components, if adequate countermeasures are not taken.

Over the last ten months (from September 16, 2013 until June 27, 2014), a new concept was developed, in the context of the FP7 CockpitCI project, for the current thesis, to reinforce the security of SCADA systems. This concept refers to a device installed in a key juncture of the network, with the purpose of monitoring the behavior of specific components, while reporting detected anomalies. However, even before this thesis (during the second curriculum semester of the year 2012/2013) there was already some research being made in this sense, which reflects in the amount of results included in this document.

Keywords

SCADA Systems

Detection Architecture

Shadow RTU

Probing Architecture

Man-in-the-middle Attack Scenarios

Index

List of Figures

List of Tables

Acronyms

ACL	Access Control List
ADU	Application Data Unit
ANSI	American National Standards Institute
APCI	Application Protocol Control Information
APDU	Application Protocol Data Unit
API	Application Programming Interface
APT	Advanced Persistent Threats
ARP	Address Resolution Protocol
AS	Autonomous System
ASDU	Application Service Data Unit
ASIC	Application-specific integrated circuit
ASLR	Address Space Layout Randomization
AVT	Advanced Volatile Threats
BMS	Backup Master Station
BSD	Berkeley Software Distribution
CI	Critical Infrastructure
CIGRE	International Council on Large Electric Systems (in French: Conseil International des Grands Réseaux Électriques)
CISUC	Centre for Informatics and Systems of the University of Coimbra
COTS	Commercial Off-The-Shelf
COW	Copy-on-Write
CPU	Central Processing Unit
CRAT	Consortium for Research in Automation and Telecommunication
CRPHT	Centre de Recherche Public Henri Tudor
CSRF	Cross-Site Request Forgery
CVE	Common Vulnerabilities and Exposures
DDoS	Distributed Denial-of-Service
DEP	Data Execution Prevention
DLL	Dynamic-link Library
DMA	Direct Memory Access
DMZ	Demilitarized Zone
DNP3	Distributed Network Protocol

DoS	Denial-of-Service
EIA	Electronic Industries Association
ENEA	Energy and Sustainable Economic Development
EPA	Enhanced Performance Architecture
FCTUC	Faculty of Science and Technology of the University of Coimbra (in Portuguese: Faculdade de Ciências e Tecnologia da Universidade de Coimbra)
FP7	Seventh Framework Programme
FSM	Field Security Manager
FUSE	FileSystem in Userspace
GPL	General Public License
GUI	Graphical User Interface
HIDS	Host Intrusion Detection System
HMI	Human-machine Interface
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
ICS	Industrial Control Systems
ICS-CERT	Industrial Control Systems Cyber Emergency Response Team
ICT	Information and Communications Technology
IDMEF	Intrusion Detection Message Exchange Format
IEC	Israel Electric Corporation (or: International Electrotechnical Commission)
IEC101	IEC 60870-5-101 standard
IEC104	IEC 60870-5-104 standard
IED	Intelligent Electronic Device
IFS	Isolated File System
IMT	Inter Message Time
INTC	Interrupt Controller
I/O	Input/Output
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-Process Communication
IRP	Integrated Risk Prediction
IRQ	Interrupt Requests
IRT	Inter Reconnection Time

IT	Information Technology
ITU-T	International Telecommunication Union Telecommunication
LAN	Local Area Network
LCT	Laboratory of Communications and Telematics
LXC	Linux Containers
MAC	Media Access Control
MBAP	Modbus Application Header
MITM	Man-in-the-middle
NIDS	Network Intrusion Detection System
NLM	Number of Lost Messages
NTP	Network Time Protocol
OCSVM	One Class Support Vector Machine
OS	Operating System
OSI	Open System Interconnection
OSVDB	Open Source Vulnerability Database
PDU	Protocol Data Unit
PIDS	Perimeter Intrusion Detection System
PLC	Programmable Logic Controller
PnP	Plug and Play
PoE	Power over Ethernet
PRU-ICSS	Programmable Real-Time Unit and Industrial Communication Subsystem
RAM	Random-access memory
RAT	Remote Access Trojan
REST	Representational State Transfer
RFC	Request for Comments
RMI	Remote Method Invocation
RTU	Remote Terminal Unit
SBC	Single-board Computer
SCADA	Supervisory Control and Acquisition
SMGW	Secure Mediation Gateway
SMP	Security Management Platform
SOA	Service Oriented Architecture
SoC	System on a Chip
SPI	Stateful Packet Inspection

SQL	Structured Query Language
TCP	Transmission Control Protocol
TNLM	Total Number of Loss Messages
TTF	Time To Failure
Tx/Rx	Transmit/Receive
UART	Universal Asynchronous Receiver/Transmitters
UDP	User Datagram Protocol
UK	United Kingdom
UML	User-mode Linux
USB	Universal Serial Bus
VFS	Virtual file System
VM	Virtual Machine
VPN	Virtual Private Tunnel
WAN	Wide Area Network
WP	Work Package
WS	Web Service
XML	Extensible Markup Language
XPath	XML Path Language
XSD	XML Schema Definition

1 INTRODUCTION

1.1 CONTEXT – SECURITY IN SCADA SYSTEMS, COCKPITCI PROJECT

In the context of the European project FP7 CockpitCI, in which the Faculty of Science and Technology of the University of Coimbra (FCTUC) is a partner, the present thesis was developed as a result of the author's involvement in the project as a junior researcher in the Laboratory of Communications and Telematics (LCT) of the Centre for Informatics and Systems (CISUC). Currently, this document is classified as confidential due to ongoing intellectual property protection actions. Initiated in January 2012, and for a period of 36 months, the CockpitCI project aims to reinforce the security of critical infrastructures, commonly referred to as Supervisory Control and Acquisition (SCADA) systems. These are used to control large-scale industrial processes (e.g., keep track of water levels to cool the fuel rods in nuclear plants) through readings and status reports carried by specialized control systems (e.g., a Programmable Logic Controller (PLC) or a Remote Terminal Unit (RTU)), which the SCADA operator supervises with a Human-machine Interface (HMI).

The CockpitCI project is broken down into seven Work Packages (WP), distributed by twelve European partners, being assigned to the University of Coimbra the lead on the WP3000 whose main concern is the design of components to be used on the analysis and detection infrastructure, using local and coordinated detection mechanisms, to isolate and lower the impact of the attacks from the remaining system. The CockpitCI consortium includes other universities (Roma Tre and the University of Surrey), as well as industries partners (Selex ES; Israel Electric Corporation (IEC); Transelectrica), end-users, Small and Medium Enterprises (SME – LYSE Energi), and research centers (itrust Consulting; Multitel; National Agency for New Technologies, Energy and Sustainable Economic Development (ENEA); Consortium for Research in Automation and Telecommunication (CRAT); Centre de Recherche Public Henri Tudor (CRPHT)).

The current thesis addresses a concept proposed by the University of Coimbra, the Shadow RTU, to improve SCADA security at specific areas of the infrastructure, providing an additional layer of safety at a different level from the remaining components. Therefore, the role of the intern in this thesis is to develop a set of security probes (or modules) for the Shadow RTU and, integrate it in the scenario installed in the laboratory (LCT) to make it interoperate with the remaining components developed by the other members of the LCT involved in the project. These probes make up the proposed probing architecture for the Shadow RTU.

1.2 THESIS OBJECTIVES

Due to the increasing relevance given to attacks targeting SCADA in the past few years, there is a need to come up with solutions that are able to react to new kinds of threats. Over time, SCADA systems and architectures have evolved to the point where the number of viruses and possible ways to compromise these keeps growing in diversity and complexity. In particular, low-level threats, i.e., the ones targeting the hardware or initiated by it (e.g., USB pen drives with malicious software intended to compromised other devices), have occupied a special place among these systems and keep getting better in the way the infections are performed, remaining unnoticed from the system and the operators.

In this sense, it is proposed the definition of a set of security probes for Industrial Control Systems[1] (ICS) to sustain the Shadow RTU concept, a device of reduced dimensions, responsible for performing network monitoring, supported on a robust architecture for such. This architecture is composed of a set of modules to handle the monitored network protocol data flowing in and out of PLCs or RTUs, decoding and processing it, while reporting to a central point that decides which measures need to be taken. Following is presented a list of goals to validate the concept:

- Comparison and validation of the devices to take on the role of Shadow RTU (e.g., available resources, behavior under stress conditions);
- Study and implementation of SCADA protocols to be handled by the Shadow RTU;
- Setup of a simple scenario for the initial validation stage of the Shadow RTU (i.e., to evaluate its decoding and reporting capabilities);
- Definition of the functionalities (Application Programming Interface (API)) to be supported by the Shadow RTU;
- Implementation and integration of a communication method (Web Service (WS)) that allows the Shadow RTU to expose its API, for management purposes;
- Implementation of the most basic functionalities of the Shadow RTU (i.e., monitoring, parsing and reporting network events), as well as some additional features (e.g., use of containers to isolate applications);
- Integration of the Shadow RTU in the LCT scenario and, validation with the remaining components (i.e., PLC, Attacker, Management Platform, and Local Correlator).

Throughout the thesis period, there was an event that took place in Mons, Belgium (28 to 30 October, 2013) and another at the International Council on Large Electric Systems (CIGRE – March 12 to 14, 2014), also in Belgium. Both events implied the implementation of a set of demonstration scenarios that were also made part of the work planned for the thesis and therefore, also part of the validation process:

- The first demonstration focused on the Shadow RTU to evaluate its monitoring and decoding capabilities, i.e., the base concept;

[1] SCADA systems are a subset of ICS, historically known for being large-scale processes composed of multiple sites, and large distances.

- For the second demonstration (CIGRE), an Attacker was implemented to complement the validation process of the Shadow RTU.

1.3 STRUCTURE OF THE DOCUMENT

The rest of the document is organized as follows: the second chapter is composed of a State of the Art to address the evolution of SCADA, how their generations have been exposed to different types of threats over time and, an overview of common vulnerabilities found in the past few years. To sustain the idea that designing a good architecture is a halfway to improve system security, a comparative study of many possible solutions is presented. The architecture partitioning approach is also supposed to give an understanding of the options taken in the CockpitCI reference architecture, presented in the next chapter. A set of statistics then present how the industry reacted to one of the most dangerous virus known till today, and how the industry will be prepared in the future for up to come threats. Finally, it is made an approach to various types of threats (mostly, at a low-level) to realize how accurate these are becoming.

The third chapter covers the CockpitCI project, namely, in respect to the probing and detection architectures. The remaining detection agents are also mentioned, along with the description of a series of cyber-threats for which these agents were designed to detect.

The forth chapter describes the Shadow RTU concept, the modules that make up the proposed probing architecture for the Agent's operation and, a series of attacks against which it is effective. A series of functionalities to be supported by the Shadow RTU are also presented, to allow it to be remotely managed.

The fifth chapter describes the entire validation process: on a first stage, a series of Single-board computers (SBC) are compared, along with a set of workload tests to evaluate the behavior of the selected board under stress conditions. At the end of this stage, the initial validation process of the Shadow RTU to evaluate the monitoring and decoding capabilities are presented; on the second stage, a set of applications used for the software validation process are presented, including Modbus TCP and IEC 80670-5-104 protocol implementation, containers, and an approach to allow remote management on the Shadow RTU. Finally, a series of scenarios are presented to validate the Shadow RTU in an attack scenario.

The sixth chapter presents the followed work plan along the first and second semester, including constrains that occurred during that period.

The seventh chapter concludes the document, presenting the contributions to the CockpitCI project and the future work to be done.

2 STATE OF THE ART – SCADA SECURITY

2.1 INTRODUCTION – SCADA GENERATIONS

Even before networks existed (in the 1960s), SCADA systems were already being developed and used in the Utilities industry, in the United States (U.S.), running a “mainframe” with no connectivity to other systems, representing the first generation (or “Monolithic”) of SCADA [SCADA2004]. Even though Wide Area Networks (WANs) existed at the time, communication with RTUs was performed with proprietary solutions, with restrictions in functionality and integration with other types of traffic. So far management was an easy task due to the simplicity inherent to the system itself – the operator only supervised the sensors state, i.e., if either these were connected and running or not. However, the bigger the scenario, the less feasible it becomes to maintain due to the budget overhead it represents, e.g., costs hiring more qualified operators to monitor the process, not to mention that, in some situations, it would be practically impossible due to the extent of the scenario and every component involved in it.

At the time of the first generation, SCADA was composed of two key components used in every recent infrastructure:

- *Remote Terminal Unit (RTU)* – Used to process the information sent by sensors, converting it into a digital format. In the first generation of SCADA, RTUs dictated the communication protocol to be used on the network. PLCs were also used and connected to sensors to perform similar tasks. Both RTUs and PLCs are also referred to as Slaves or Servers;
- *Master Station or Client* – These are used to control the Slaves through a HMI, which the operator uses to monitor the system’s process. At the time of the first and second (“distributed”) generations this device was normally referred to as Master or Mainframe Computer.

The two first versions of SCADA were extremely simple with absolutely no security layer added to it, e.g., the only way to keep track of the system’s operation was through the observation of state indicator leds (1st generation) or with a HMI (2nd generation), becoming an upcoming concern on future generations. The third generation (“networked”) of SCADA brought with it a key advantage: the disaster survivability, i.e., the ability to distribute processes across separate physical areas and avoiding the entire network to be compromised in case an attack is targeted to a specific location. Since its standardization in today’s processes, additional components are now part of SCADA systems (cf. Figure 2-1):

- *Database* – Although it is an integral part of the Master Station (along with the HMI) it plays a very important role to keep the system in operation, avoiding the greatest consequences of system downtime and engineering hours spent restoring it (e.g., an historian is maintained to track system trends and perform diagnostics). In this sense, two types of databases are commonly used: relational and real-time. While the first (relational) is best suited for keeping process state history (e.g., sensor readings), the second (real-time) is used due to performance issues required to maintain system state at specific moments, delivering several distinct advantages such as powerful alarm mechanisms, long term history or (per machine) purpose (or application) specific optimization [RTAP];

- *Communication Server* – As already mentioned, in the first generations of SCADA it was common to see proprietary communication solutions being implemented until the third generation emerged, bringing Ethernet [IEEE802.3] connectivity. That said, it is still possible to see both legacy and networked components working together in today's scenarios (cf. Figure 2-1);
- *Field devices* – Connected to the RTUs, field devices (or sensors) directly interact with the process itself (e.g., water level reading), reporting the measured values to the Master Station. Conversely, depending on the reported values, actuators may change the process (e.g., restoring water levels to normal).

This new generation made possible to distribute SCADA functionalities across the WAN, but also brought with it the existing vulnerabilities and attacks in Internet Protocol (IP) networks [ISA2011] (cf. section 2.2). In fact, even before the integration with conventional IT (Information Technology) networks, SCADA systems had their own vulnerabilities due to the existing architectures, devices, software and special (proprietary) protocols [Choraś2010]. Simply put, the security issues in SCADA are not particularly new. In fact, some of the problems that were commonly reported back then still occur today [ICSA-14-084-01].

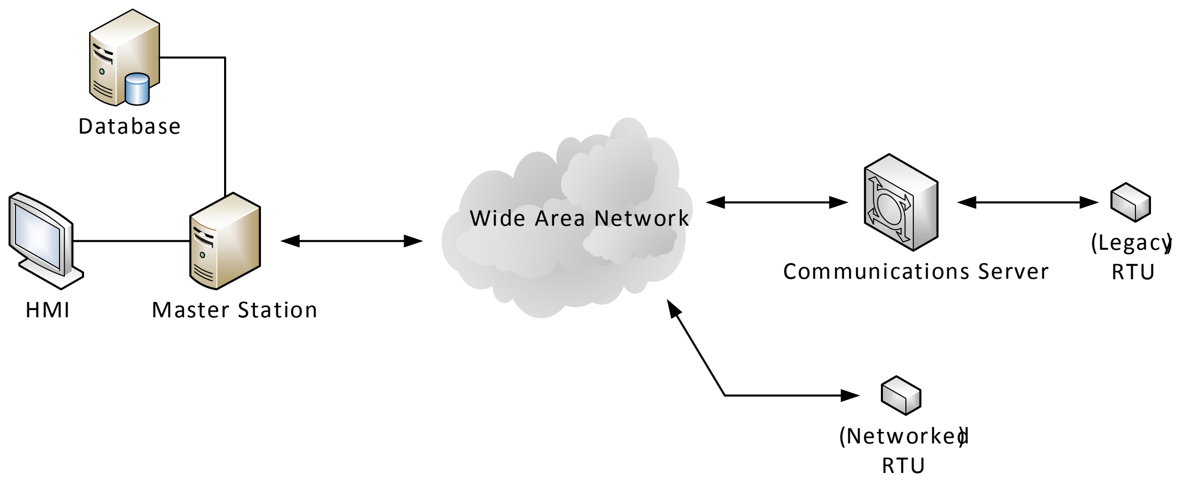


Figure 2-1 – SCADA architecture: Third generation (Adapted from [Edvard2013])

The path followed by most SCADA systems tends to the standardization of both components and communication protocols, allowing the adoption of Commercial Off-The-Shelf (COTS) equipment, leading to a cheaper, simpler and more opened architecture. This was exactly one of the advantages from the adoption of the Internet of Things (IoT) technology [IoT-GSI], alongside with maintenance and integration ease [Harbor2012], leading SCADA to its fourth generation where the capabilities of cloud computing are taken advantage of to reinforce security, availability and responsiveness of the entire network [Combs2011].

2.2 Overview of SCADA Vulnerabilities

Since the first generation of SCADA, security has been an issue for many different reasons. If at first operators trusted on security by obscurity, i.e., trusting on proprietary protocols and interfaces with the idea that no one else would know these; or because the network was physical secured and disconnected from the Internet [Synergist2012], today, with the introduction of standard network protocols (opened and known by everyone), the adoption of commercial Operating Systems (OSs) (e.g., Microsoft Windows), and the implementation of Plug and Play (PnP) devices [ISA2011], just to mention a few, made new issues emerge – E.g., the introduction of Transmission Control Protocol/Internet Protocol (TCP/IP) connectivity increased the number of unauthorized access related issues, forcing vendors to develop specialized Firewalls and Virtual Private Tunnels (VPNs) to secure the communication channels to the SCADA network [Epiphan]; the same types of viruses found in commercial OSs also impacted the ones used inside SCADA networks but, with a greater impact since these can't just be simply stopped to apply the patches and then have the system rebooted. The Stuxnet virus became quite popular for exploiting a vulnerability in Microsoft Windows OSs, by being able to send unauthorized commands to the control equipment and changing these while displaying false information to the operator, making him believe that the entire infrastructure was safe and running properly [Falliere2014]; whenever requested, PnP devices send a detailed description of themselves which the attacker may use to take over control [Clarke2004].

In fact there are many more reasons known to the public to consider these systems unsecure – commercial rivalries, disgruntled ex-employees, hackers, terrorist attacks and malware (e.g., Stuxnet, as mentioned above) –, which isn't properly good for the image of this industry. Anyway, whatever the source of the attacks may be, it is necessary to understand and examine the threats landscape since this is actually a concern related to national security in which the disruption of critical services may result in much serious consequences, such as the failure of the infrastructure of even the loss of lives. Due to the potential disruption of such services, some of the vulnerabilities targeting these are believed to be associated with politically motivated or state-sponsored attacks [Symantec] as was the case of Stuxnet, as confirmed in the summer of 2012 [Shekaraubi2014].

Also known as a Zero-day vulnerability, the Stuxnet virus reduced the lifetime of Iran's nuclear centrifuges by manipulating the way these spin, eventually ending up destroying a fifth of these. In fact, the attack occurred in two separate phases and, its propagation through a USB pen drive inserted by one of the employees working inside the facility. Initially, the goal was to reproduce an electrical blueprint of Natanz plant to understand how the equipment controlling the centrifuges used to enrich uranium worked; Since everything appeared to be normal to the operators, only after a few years of its detection was the second variation of the virus released along with its propagation [Kelley2013].

2.3 Evaluation of Architecture Partitioning Solutions

Since SCADA systems are not as “invisible” as these used to be, designing a secure architecture requires taking into account three key aspects [Yokogawa]:

- *Prevention* – These measures should be implemented before the system design and architecture, and maintained over time – E.g., separation between SCADA and Corporate networks; updated software and OS; firewall and antivirus configuration and; access to technical information and backdoors limitations;
- *Detection* – A logging mechanism is usually associated to these procedures for later examination and detection of deviant behaviors – E.g., history and log data at the application and OS level; audit trails and; alert mechanisms at both software and hardware level;
- *Recovery* – In case a disaster occurs, restoring the system to its default behavior requires a set of previously defined measures, i.e., a recovery plan, to ensure that everything comes back to its proper operation. These systems are not isolated, requiring some measures to be taken – E.g. acquisition of fault-tolerant hardware; fallback mechanisms; an impact assessments (a realistic prediction) and; a backup and procedure plan.

Considering that SCADA specific protocols like Modbus [ModbusTCP] and DNP3 [IEEE1815] were not conceived to be secure, other methods had to be developed to work around the vulnerabilities left by these, such as confidentiality and integrity. One of the most accepted methods to achieve this is through network segmentation, where the entire system is partitioned into distinct security zones to isolate critical parts of the system [Byres2012]. As mentioned by the American National Standards Institute (ANSI) in [ANSI-ISA-99], a security zone is a logical grouping of physical, informational, and application assets sharing common security requirements, which can also be combined with other complementary mechanisms such as Firewalls and the definition of a Demilitarized Zone (DMZ).

Following are presented some of the ways to partition an architecture, as well as an overall assessment for all these [CPNI2005].

2.3.1 Dual-Homed Solutions

One of the most basic ways to achieve isolation is through the use of two network interfaces on the device between the SCADA and Enterprise Network (cf. Figure 2-2 – Consider the blue line connection only). This type of isolation is not very effective since it can be easily bypassed by an attacker that successfully manages to compromise the device “in the middle”, e.g., forwarding packets between the two networks and, possibly, compromising both. The Slammer worm acted on conditions similar to the ones described here, i.e., even though the target was not the SCADA network, it eventually compromised the nuclear plant of Davis-Bess in Ohio, United States [Poulsen2003].

In order to reinforce the security of this solution, a more powerful device should be used instead, e.g., an Historian Server, with basic Firewall policies (cf. Figure 2-2 – Consider the orange line connection only).

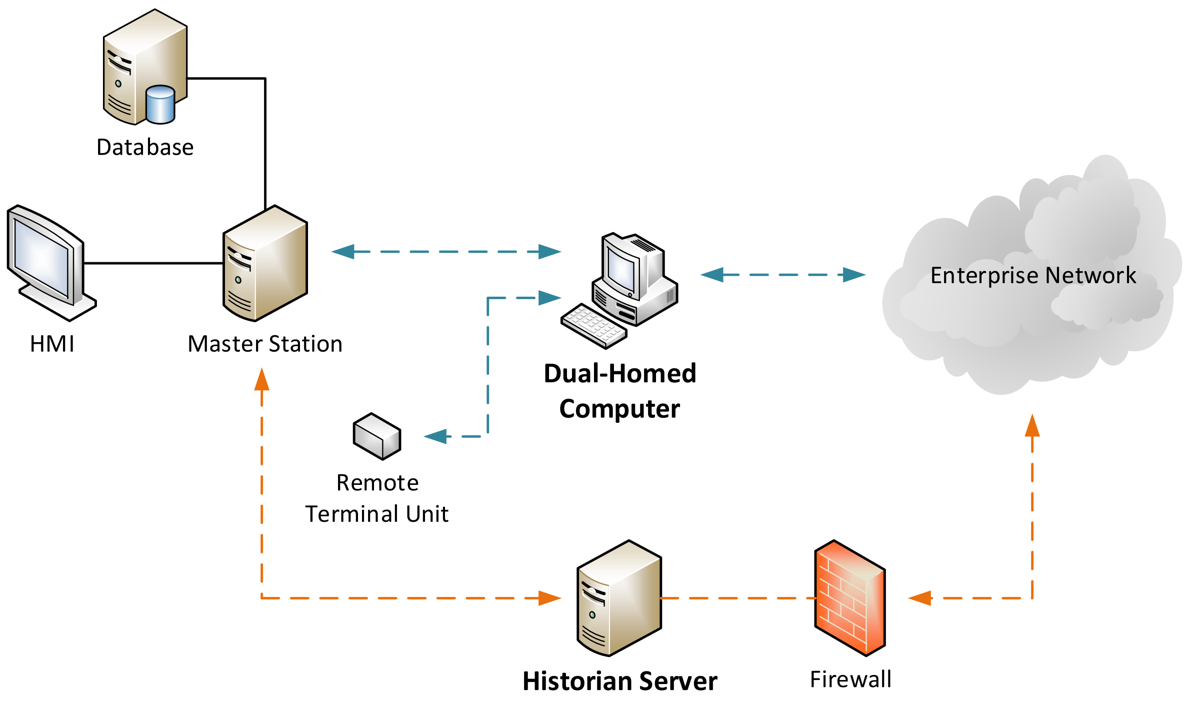


Figure 2-2 – Dual-Homed Computer (blue line communication) and Dual-Homed-Server with Personal Firewall Software (orange line communication) (Adapted from [CPNI2005])

The downside of the latter solution is the possibility of having blocked traffic in the advent of, e.g., remote access to the operation network is needed. Since this approach only provides a server data sharing mechanism, it is also possible for traffic not to be blocked at all.

2.3.2 Router and Firewall-based Solutions

A router or a layer-3 switch configured as a bridge and acting as a packet filter Firewall, can be used to effectively reinforce device-to-device rule sets but can’t prevent more advanced attacks, such as those that take advantage of packet fragmentation techniques, due to the inexistence of Stateful Packet Inspection (SPI) mechanisms (cf. Figure 2-3 – Consider the blue line connection only). This type of solution reminds a bit the concept of security by obscurity, since it is only feasible on networks that are known for not being a target to attackers or, on the other hand, extremely secure. Also, many of these routers can be easily upgraded to support SPI, making this solution equivalent to some of the ones presented in the next section.

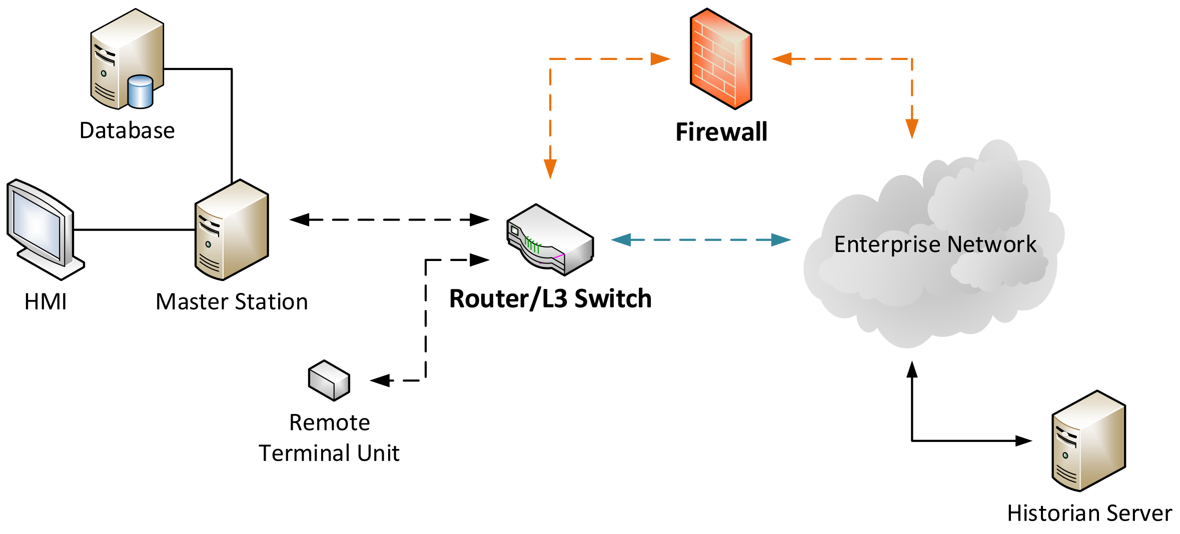


Figure 2-3 – Packet Filter Router/Layer-3 Switch (blue line connection) and Router/Firewall combination (orange line connection) between SCADA and Enterprise Networks (Adapted from [CPNI2005])

The combination of the previous case with a dedicated Firewall makes a better solution, since network traffic will first “hit” the router and, only after handling the bulk data, is then passed to the Firewall (cf. Figure 2-3 – Consider both blue and orange line connections). This solution is mostly used for Internet-facing Firewalls and not so much in SCADA environments.

Most SCADA operators rely on the Firewall as the most important component to secure the infrastructure, applying strict rules to reduce the probability of an external attack to succeed – E.g., using a two-port Firewall makes a better filtering solution than the one discussed above, providing higher degrees of manageability and scalability (cf. Figure 2-4).

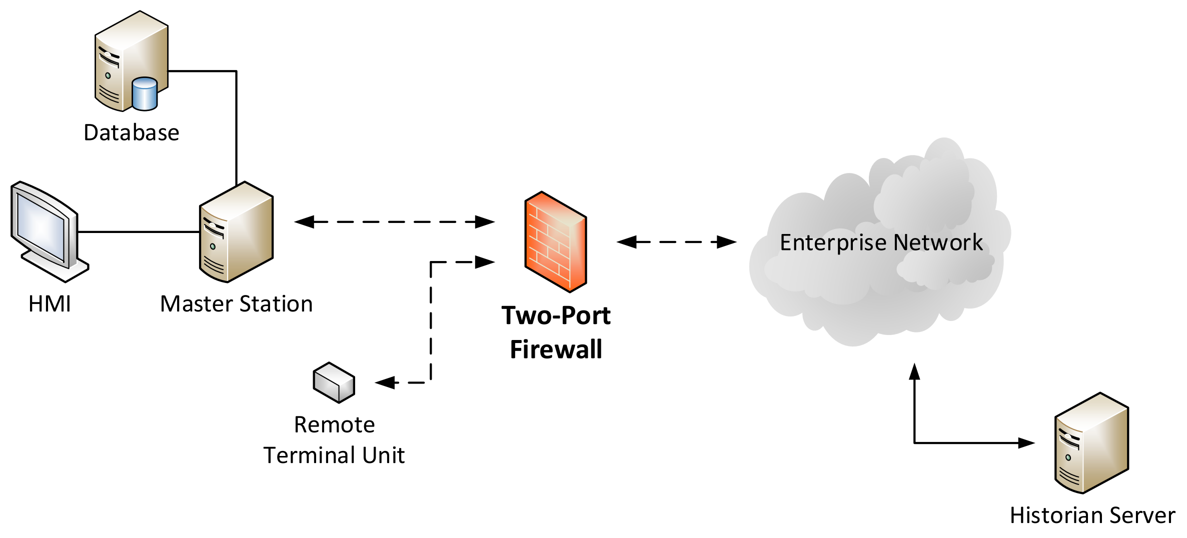


Figure 2-4 – Two-Port Firewall between SCADA and Enterprise Networks (Adapted from [CPNI2005])

Considering the illustration in Figure 2-4, one can deduce that the decisions taken by the operator, when defining where each device will be located, can affect the policies to be applied on the Firewall, e.g., the Historian Server is located on the same side as the Enterprise Network, which requires the configuration of a rule that allows it to communicate with the devices on the SCADA network (e.g., the RTUs) (cf. Figure 2-4). However, a misconfigured Historian could easily compromise the devices inside the production network, since the Firewall "allows" it to do so. On the other hand, if the Historian happened to be installed inside the operation network, there should be a rule on the Firewall to allow some of the devices in the Enterprise Network to consult it which, again, would raise the risk of a compromised computer (e.g., by a virus or worm) to change the stored values in the Historian[2].

2.3.3 DMZ and VPN-based Solutions

Considering the two-port Firewall mentioned in the previous section (cf. section 2.3.3), adding one more interface to it allows the operator to design an architecture with as many DMZs as additional Firewall ports are available. The first two ports are still used to connect to the Enterprise and SCADA networks, and the remaining ones to isolate specific components (e.g., the Historian) or zones of the network (cf. Figure 2-5 – Consider the blue line connection only).

[2] The communication between network hosts and SCADA devices occurs in Structured Query Language (SQL) or Hypertext Transfer Protocol (HTTP).

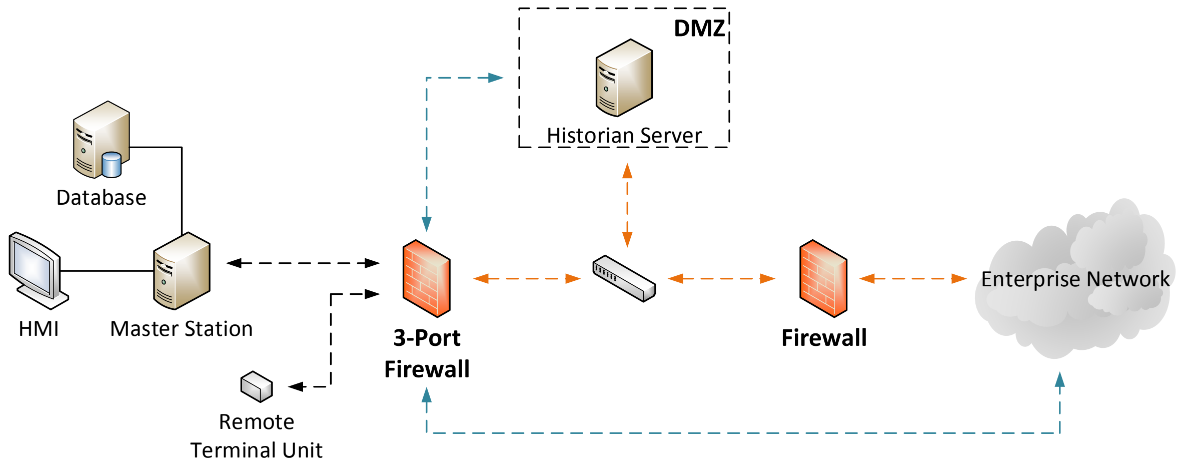


Figure 2-5 – Firewall with DMZ (blue line connection) and Paired Firewalls (orange line connection) between SCADA and Enterprise networks (Adapted from [CPNI2005])

Once again, rules must be applied to ensure the communication between the three zones is performed safely, eliminating the need to establish a direct communication path between the Enterprise and SCADA network. In this case, an Access Control List (ACL) can be "attached" to maintain these rules. Special care should also be taken when configuring the Firewall to only allow communications between the SCADA and DMZ networks to be initiated by the first (e.g., by an RTU), to avoid issues with compromised devices inside the DMZ communicating with the production network. Another concern with this type of configuration is the complexity accumulated in ACLs, making the network more prone to errors.

Considered as one of the most secure solutions presented here, the implementation of two Firewalls between the enterprise and SCADA networks, allows the first one to block arbitrary traffic from entering the production network and the second one to prevent compromised devices from sending data to the SCADA network. The DMZ network is located between these two firewalls (cf. Figure 2-5 – Consider the orange line connection only).

Finally, another solution is presented where instead of segmenting the network with DMZs, VLANs are used instead to allow communication between these and force all traffic to go through a Layer-3 packet filter Switch (cf. Figure 2-6). All traffic inside one VLAN goes through a Layer-2 Switch but, again, if a device in VLAN A needs to communicate with another in VLAN B, the Later-3 Switch is used.

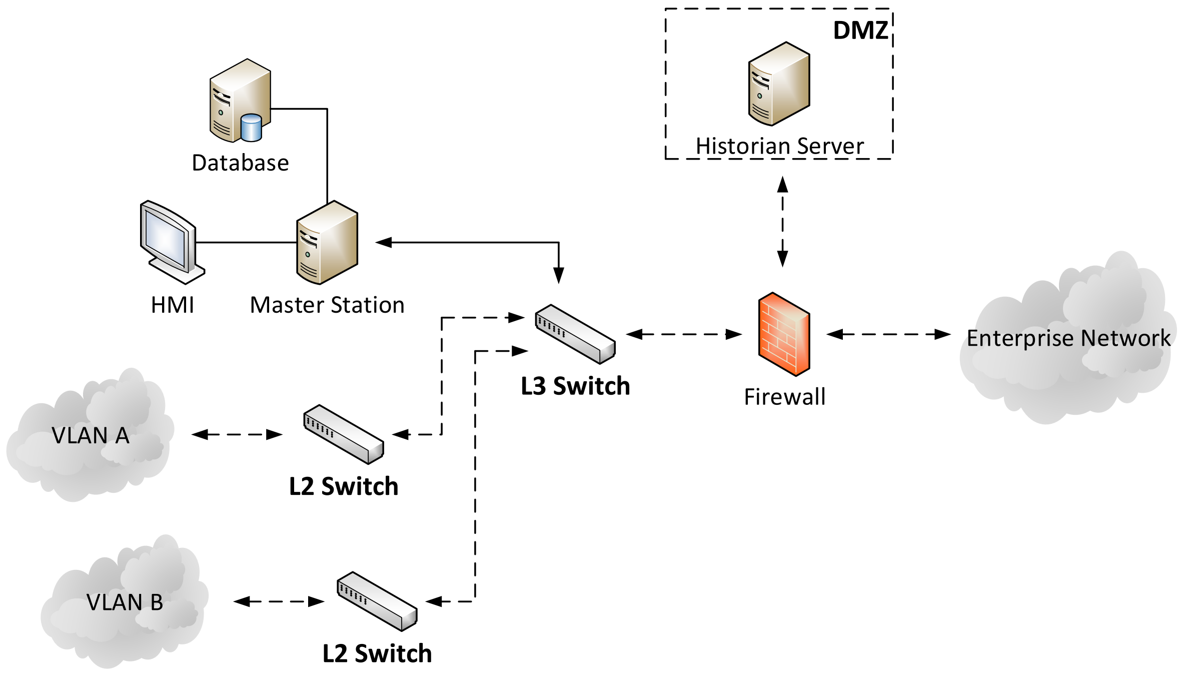


Figure 2-6 – Firewall and VLAN-based Process Network Combinations (Adapted from [CPNI2005])

The combination of some of the previous scenarios makes this one the most scalable and still secure solution – VLAN separation prevents compromised devices from affecting others.

2.3.4 Solution Comparison and Evaluation

It is important to understand how SCADA architecture designs have evolved over time, and how new mechanisms and technologies might have changed the way operators implement more advanced solutions. Following, is presented an evaluation of the architectural designs mentioned above according to three key aspects: Security, to prevent possible attacks; Management, both local and remote and; Scalability, to allow the deployment of both large and small systems.

Table 2-1 – Approximate Security, Manageability and Scalability Ratings for SCADA Segregation Architectures (Adapted from [CPNI2005])

Architecture	Security	Management	Scalability	Score
Dual-Homed Computers	1	2	1	4
Dual-Homed Server with Personal Firewall	2	1	1	4
Packet Filtering Router/Layer-3 Switch	2	2	4	8
Router/Firewall Combination	3.5	3	4	10.5
Two-Port Firewall	3	5	4	12
Firewall and DMZ	4	4.5	4	12.5
Paired Firewalls	5	3	3.5	11.5
Firewall/VLAN-based Combination	4.5	3	5	12.5

Considering the results in Table 2-1, one can deduce that non-Firewall solution are not even an option for today's SCADA scenarios but, on the other hand, the last presented ones, i.e., the ones based on a three zone system, are the most feasible ones.

2.4 POST-STUXNET SECURITY

The discovery of Stuxnet in 2010 created a turning point in SCADA security. The the number of discovered flaws in SCADA software increased by twenty times and, the vendor whose PLC was the ultimate victim of this virus (Siemens [SimaticS7-300]) has patched 92 percent of reported vulnerabilities in their products, over the last seven years [Ptsecurity2012]. The impact was so notorious that by the end of 2011, 64 vulnerabilities in ICS products were found and reported, comparatively to only 9 between 2005 and 2011 and, 98 between the months of January and August of 2011 (cf. Figure 2-7 – blue line). This information was based on vulnerability database information from Industrial Control Systems Cyber Emergency Response Team (ICS-CERT) [ICS-CERT], Common Vulnerabilities and Exposures (CVE) [CVE], Bugtraq [Securityfocus], Open Source Vulnerability Database (OSVDB) [OSVDB] and ProductCERT [CERT].

To give an even clearer idea of the lack of importance given to the ICS industry in past years, it was discovered that by the time the Stuxnet incident was being investigated, one of the exploited vulnerabilities, a Microsoft SQL Server default password issue [CVE-2010-2772], was already known long before the attack. This issue was even mentioned in a forum in May 2005 and the default passwords published 3 years later, in May 2008. It took three additional years for the Stuxnet attack to occur (2010) and have this issue resolved. This case explains pretty much everything that is wrong with SCADA security.

Looking at the following chat (cf. Figure 2-7 – blue line) there is an increasing amount of found vulnerabilities between the years of 2010 and 2011. Most of these were discovered by a security researcher [Auriema], who spotted 93 out of the 129 published flaws.

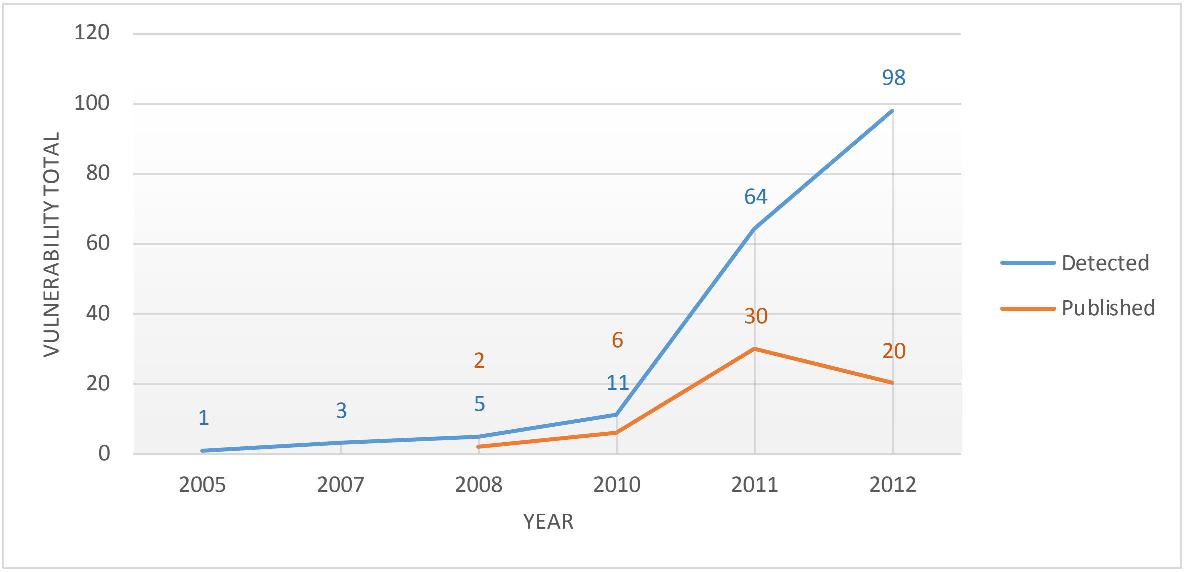


Figure 2-7 – Number of ICS Reported/Detected Vulnerabilities vs. Number of ICS Published Vulnerabilities (Adapted from [Ptsecurity2012])

In an interview [Peterson2011], the researcher explained the methods and techniques he carried to find these vulnerabilities and also why the image of this industry in terms of security is so degraded. The first stage consisted in an understanding the network protocol, while trying to notice existing bugs. For that, the researcher conducted an auditing process of the main operations with a debugger to verify what the program did with the bytes of the incoming packets. Finally, an automated tool was used to modify the original sample packet or build one from scratch until an exception was raised (i.e., until a crash occurred).

To explain why the situation of security in this industry is so critical, the researcher mentioned that there is no interest in these vulnerabilities, something that was confirmed by ICS-CERT in his discussions where, according to him, the bottom line was that SCADA is a field so critical that if someone finds a bug, it should be immediately reported for free and, whoever discovered it should cooperate (also for free) to fix it. By the time he discovered the bugs he did not contact the respective vendors once that, according to him, most companies don't event credit the authors for the patching. To sustain these arguments, the previous chart (cf. Figure 2-7 – blue and orange lines) shows a correlation between the detected vulnerabilities and published exploits from 2008 and 2010. However, from 2011 to the end of September 2012, only 50 exploits were published. Even though these numbers are six times greater than the ones from 2005 to 2010, which is a good thing, it also supports to the arguments reported in [Peterson2011]. Relatively to the low number of published exploits in 2012 when compared to 2011, it may, again, be related to the formalization of the relationships between SCADA vendors and security researchers, as well as the use of responsible disclosure policies between these. Finally, the costs that are incurred in the development of exploitation tools causes a significant delay between the publication of the vulnerability information and the exploits.

2.4.1 Vulnerabilities in Hardware and Software Components

Most ICS vendors started to take special care in securing their products, as it was the case of Siemens, Schneider Electric and Broadwin/Advantech, whose numbers of found vulnerabilities made the top among the remaining companies in this industry. During this period, some vendors changed their security approach from reactive to proactive in a demand to spot and fix flaws, as it happened with Siemens – the company created a specific department for this purpose called ProductCERT, as mentioned above, whose efforts were also included in the results presented in Figure 2-7. To give a more clear idea of the amount of vulnerabilities found in ICS components of various vendors until the year of 2012, following is presented a graph where SCADA systems, HMIs and PLCs make the top (cf. Figure 2-8):

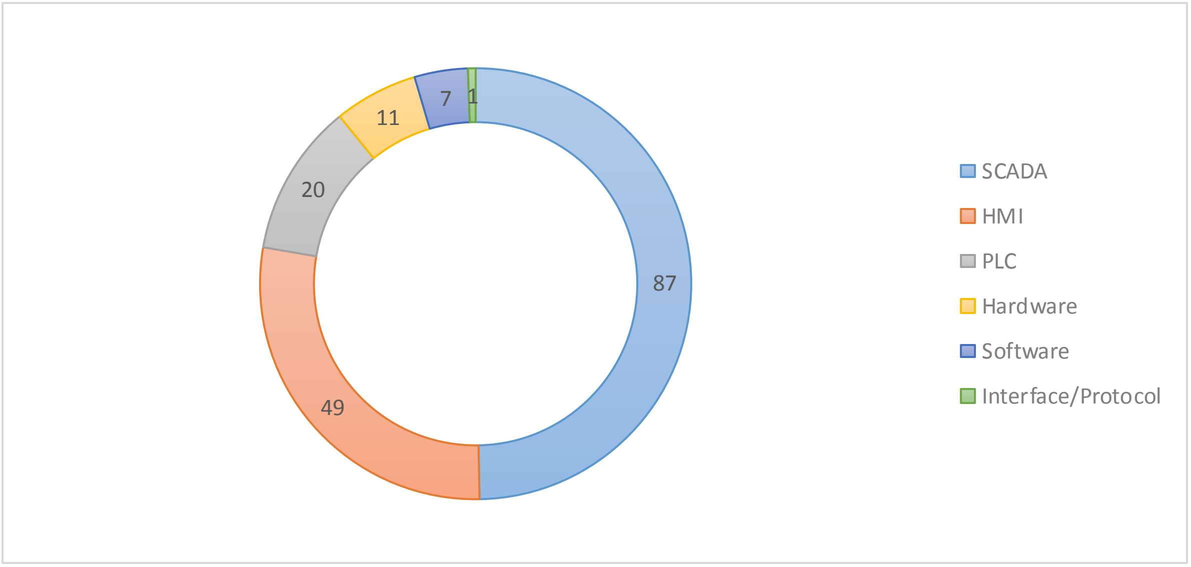


Figure 2-8 – The Number of Vulnerabilities in Different Types of the ICS Components (Adapted from [Ptsecurity2012])

According to [Jackson2012] some vendors (e.g., Siemens) have reportedly made a lot of progress on workstation and server side, implementing better security controls. However, on the PLC side, very little progress has been made. At the same time, in software products, over a third (36%) of reported vulnerabilities had exploits and, half of the vulnerabilities (50%) allowed attackers to remotely execute code (cf. Figure 2-9). More of 40 percent of the bugs found were considered to be “critical”. Once again, these number don’t show a big amount of exploits that are not disclosed.

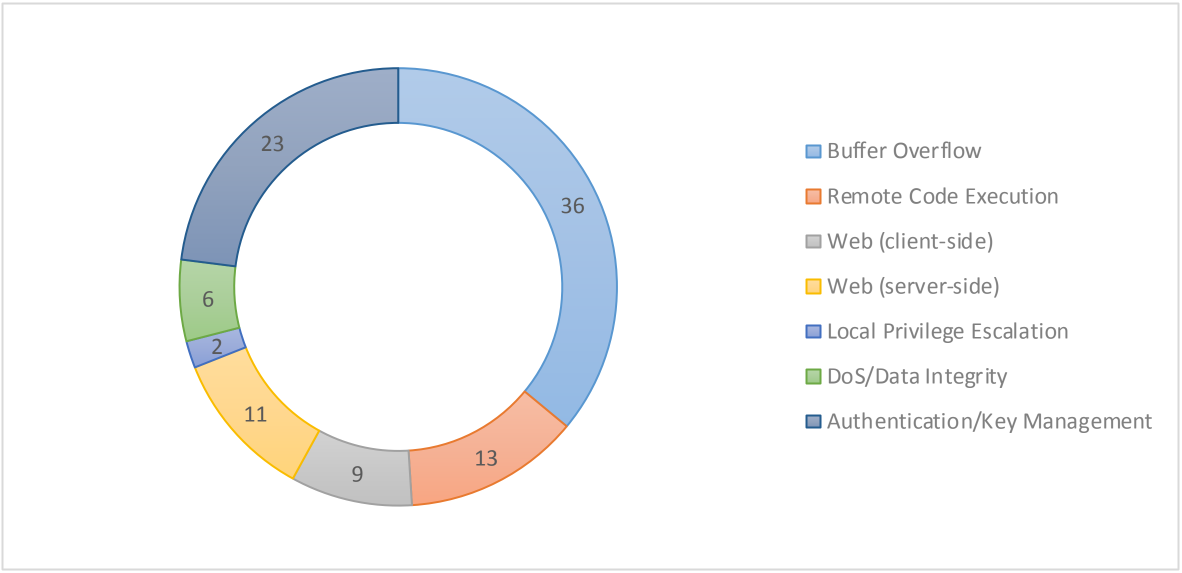


Figure 2-9 – Classification of Vulnerabilities in ICS According to Type (Adapted from [Ptsecurity2012])

Considering the graphic in Figure 2-9, vulnerabilities associated with Buffer Overflow [owasp2009] allow the attacker to have control over the program, ending or freezing it, leading to a Denial-of-Service (DoS) [ST04-015] and the execution of arbitrary code on the victim's system. These two vulnerabilities (DoS and Remote Code Execution) make up 50 percent of the picture (cf. Figure 2-9). Also relevant are the 23 percent allocated to Authentication and Key Management issues.

To give an idea of how serious vendors take the issue of fixing vulnerabilities, according to [Ptsecurity2012], Siemens patched 88 percent of its vulnerabilities; Advantech/Broadwin, 91 percent; and Schneider Electric, 93 percent. Other vendors fixed a high percentage of vulnerabilities as well (above 65 percent), except for Lantronix and Schweitzer Engineering Laboratories who have made no fixes at all. Considering the number of vulnerabilities fixed promptly, 81 percent of these were fixed within 30 days of public disclosure or before the flaws became widely known. However, every fifth bug was not even fixed at all or it was fixed after a significant delay.

Fixing vulnerabilities in traditional SCADA products is in fact a relatively thankless process if one considers that these products were never created under a security development life cycle program. In fact, it is going to be a never-ending process because there are systematic problems in the product and so, it is just a matter of time before one decides to stop patching bugs and actually start redesigning the code.

2.4.2 Future Threats in ICS/SCADA

Even though the Stuxnet worm has received a lot of attention in the latest years, the greatest threat to ICS are copycats that could use it as a blueprint for future attacks, creating mutations with the same basic techniques. Unlike Stuxnet which only affected the Siemens SIMATIC family components and STEP 7 PLC projects with specific proprieties [Falliere2010], these mutations may even extend the range of vendors, becoming less selective. The fact that it took 12 months for the Stuxnet to be detected proves that the use of conventional protection mechanisms (e.g., Antivirus) are never sufficient and that, a fast and reliable discovery of such threats are a key aspect for defense against vulnerabilities that are yet to come.

That being said, following are described the activities of the Stuxnet virus across all the affected layers in four stages, to provide a way of planning proactive measures against future mutations of it and minimize its negative impact [Rössel2011]:

1. *Operating System Infection* – As already mentioned, the worm spreads across networked and non-networked devices running Microsoft Windows through a USB pen drive, exploiting four previously known vulnerabilities (zero-day exploits). At this point, the virus installs two device drivers signed with private keys stolen to two reliable companies, Realtek and JMicron. These vulnerabilities have in fact been in several generations of the OS and so, it wasn't something particularly new.
2. *Software Manipulation* – The worn now manipulates the OS databases as well as any STEP 7 projects it finds, while making itself persistent on the system to locate the controllers referenced by those projects to be used as future targets (cf. step 3). Afterwards, the worm renames the Dynamic-link Library (DLL) files inside the directory responsible for the communication between the SIMATIC Manager and the S7 controllers and replaces it with a wrapper DLL of its own.
3. *Controllers Manipulation* – The wrapper DLL mentioned in step 2, allows the Stuxnet to inject malicious code into compromised PLCs with very specific properties (S7-417 series of controllers), selectively. This code combines denial-of-control and denial-of-view techniques into a Man-in-the-middle (MITM) attack, making legitimate PLCs lose control of the process without anyone (e.g., operators) or anything (e.g., HMIs) noticing it. At this point, the code permanently manipulates the frequency converters and turbine controls to disrupt the process, as also mentioned in chapter 2.2.
4. *Communication with Control Servers* – The infected computers now try to communicate with command and control servers over the Internet to upload collected information from the target and its environment. Once the connection is established to the exterior, the Stuxnet can even receive updates and execute the new data.

Simply put, Stuxnet became the first known rootkit for ICS capable of hiding itself, while taking advantage of the programming software to upload code into the PLC. When the operator used the infected machine to check on the PLC, it would also hide the injected malicious code. At this point it is important to understand that rootkits are still evolving and becoming serious threats, not only for SCADA but in every domain. In general, rootkits are known for being extremely difficult to remove from the infected machine due to their ability to trick the detection software, a characteristic known as polymorphism, i.e., the ability to rewrite the code, requiring the entire system to be reinstalled in some cases, e.g., when the rootkit is installed in the kernel. However, if the threat goes beyond the kernel level (e.g., firmware rootkits) it may be necessary to replace the infected hardware [Kassner2008].

2.5 Low-Level Threats

Low-level threats are today a serious concern, not only for SCADA operators, becoming more complex and hard to understand each day – basically anything that has the ability to process or be processed is going to be used to expand the attack surface on a given system, e.g., from physical and virtual ports to browsers and the OS. The trend of focusing in security at the application level may have contributed to put aside these kinds of threats, making also path for new high-speed technologies like the USB 2.0 and FireWire to join the team [Wilson2013].

The complexity and accuracy inherent to these types of threats may possibly call the attention for any attacker with intentions of making a step beyond Stuxnet, raising the awareness in ICS security even more. What really makes such vulnerabilities really interesting is that these operate at the lowest level, i.e., the hardware, enabling full control of devices at a root level, making it even possible to perform the infection before these completely boot.

Following are described some of the most relevant types of low-level threats to better understand how these attacks are performed [Causey2013].

2.5.1 Advanced Volatile Threats (AVTs)

These threats are targeted to the computer's volatile memory. According to Triumfant, whose activity is mainly focused in the detection, analysis and remediation of malicious attacks that evade traditional endpoint protection solutions such as the ones listed here, AVTs are becoming increasingly popular and may event one day replace Advanced Persistent Threats (APTs – described below) as the most common type of attack. By writing the malicious code in memory rather than in disk, it makes it harder to be detected or even from knowing it was even there (e.g., after a system reboot or an overwrite action is performed). On the other hand, and since the attacker doesn't get the persistence either, he would have to "re-attack" the system/device to gain access once again.

2.5.2 Memory Attacks

In this case, a vulnerable application or module in the OS is typically used as a starting point, requiring either physical access or a way to load code into Random-access Memory (RAM) in a persistent way. Even though these types of attacks are not necessarily new, most of these are still unsolved or still unsecure (e.g., Stack and heap-based buffer overflows that operate in memory). Memory injection attacks using DLL files are a typical example (cf. Figure 2-10): once stored in disk, the installed anti-virus will scan the DLLs so these can be loaded in memory at a later stage. However, it is possible to replace the loaded DLL, tricking the processor into executing the new code. At this point, the real damage begins – now, it all depends on the difficulty to compromise the chosen platform or OS. This previous example is only applicable on Microsoft Windows OSs, however, even Linux or BSD-based (Berkeley Software Distribution) kernels can be exploited [Iozzo2009] for the same type of attack.

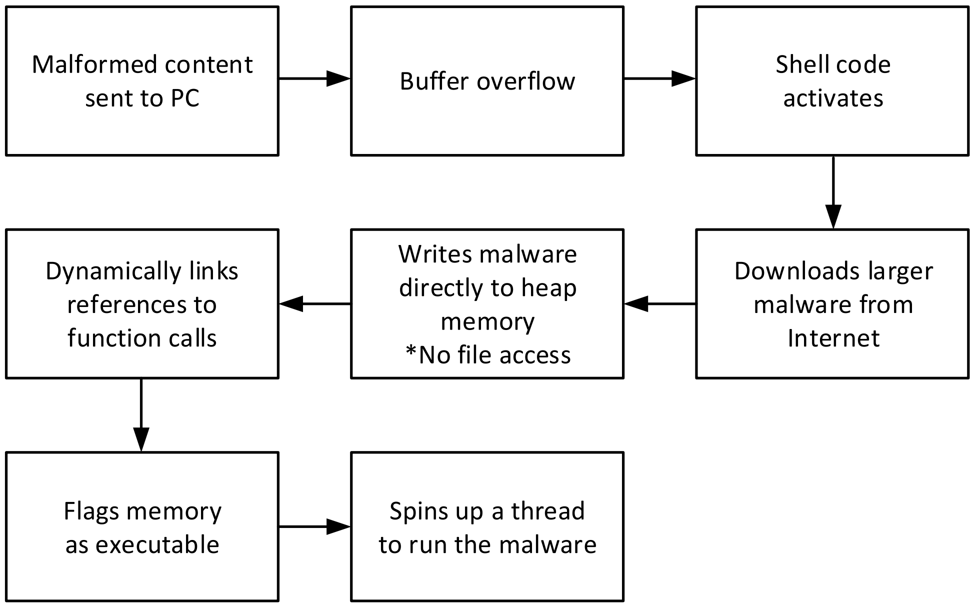

Malformed content sent to PC
Buffer overflow
Shell code activates
Downloads larger malware from Internet
Writes malware directly to heap memory *No file access
Dynamically links references to function calls
Flags memory as executable
Spins up a thread to run the malware

Figure 2-10 – Reflexive Memory Injection (Adapted from [Causey2013])

One thing to be noted is that these types of attacks, once operating in volatile memory, can spread to disk and become a persistent threat. Nevertheless, there are still two more things that makes these types of attacks scarier: The first one, relates to on how hard it is for anti-viruses to detect such threats, since these reside on RAM; the second one, relates to the fact that Remote Method Invocation (RMI) can use either authorized or protected processes (by Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR), respectively), to pull libraries into these.

The issue to be noted here is that memory attacks don't even required user interaction, e.g., the case reported in [Pauli2013] describes an example where external physical interfaces that use Direct Memory Access (DMA) are vulnerable to a sophisticated malware dubbed DAGGER. Once again, this attack was not detected by the antimalware system because it was out of its detection scope.

2.5.3 Advanced Persistent Threats (APTs)

This type of threat involves writing data to disk to maintain access and expand capabilities, e.g., by hosting itself inside the victim's network for long periods of time and retrieving sensitive information. The extent of damage is still unknown but can infect a system as low as the firmware and, since these are only detected when active its detection is almost impossible.

2.5.4 Firmware Attacks

Firmware is a very particular type of code, also impossible to scan and access by anti-viruses and HIDSs making these the most difficult type of persistent threat to handle. What makes it even more dangerous and hard to detect is the fact that devices like printers [Cui2013] and Wireless access points, which contain firmware, don't have any kind of security software in it making its information (e.g., log files) even more unreliable.

2.5.5 Rootkits

Starting on RAM, Rootkits then exploit a non-critical module or application until reaching the user and kernel space, where it will stay undetected until it finds a way to strike its final goal, the OS (cf. Figure 2-11). The Stuxnet virus, as already mention here, is included in this category. Rootkits can be User or Kernel Mode: While the first (User Mode) runs without any kind of privileges, allowing itself to reside inside an authorized process in memory, the second (Kernel Mode) is way more difficult to detect and remove since these run with system level privileges, allowing themselves to make damages at a deeper level in the system.

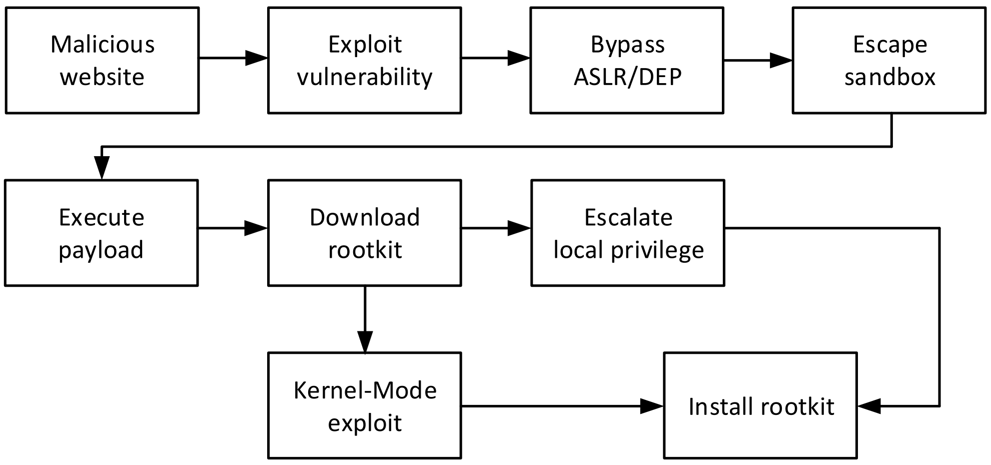


Figure 2-11 – Modern Rootkit Flow (Adapted from [Causey2013])

2.6 Methods of Exploitation

A recent article published in [Mello2013] warned energy industry enterprises using SCADA systems to be on alert for targeted spear phishing attacks conducted through emails sent to the employees containing malware that, could possibly spread into the network and open a back door for future attacks. However, the biggest issue is when an employee has to access the Internet to update something on the SCADA network, possibly starting the contamination. Another example of a recent compromise are watering hole attacks which take the process a step further by infecting the victim with malware embedded in a website likely to be visited by it.

The fact is that no enterprise is capable of protecting its network against all the malicious attacks, no matter how secure the perimeter may seem or be. Considering this, organizations must invest in solutions at a host and network level that focus on detection and prevention of infections. Also, no one should ever consider that if the Anti-virus does not show any alerts it is equivalent to good news, even less if one is dealing with lower-level threats that operate below the application level (cf. section 2.5).

2.7 Secure Execution Environments

The differences between ICS and general purpose IT systems in terms of security present opposite priorities relatively to availability, integrity and confidentiality [ISA-99.00.01-2007]: while IT security is more concerned with securing "back office" or business systems, placing confidentiality at the top of its priorities, in ICS systems the concern of having the entire infrastructure operating at all times, gives availability the highest priority and, confidentiality the lowest. In both systems integrity ranks the second place (cf. Figure 2-12).

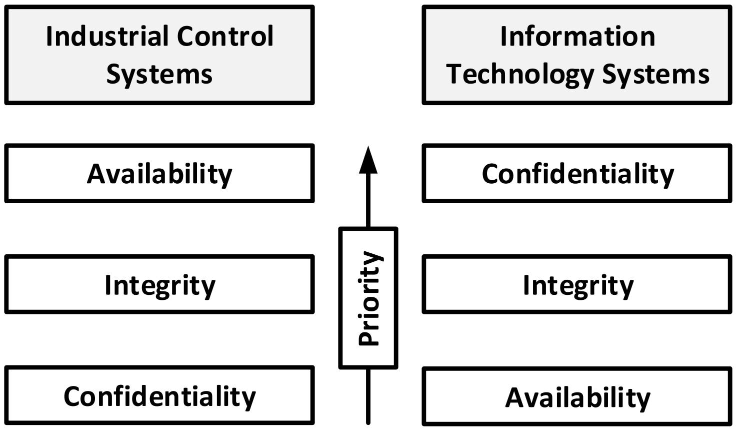


Figure 2-12 – Security priorities between ICS and IT networks

The fact that ICS places confidentiality at the bottom of priorities does not mean that operators care less with security. It means that, unlike IT applications which in general are more tolerant to delays, in ICS these delays could represent a possible threat to the operation of the infrastructure itself and the purpose it serves. Since security in IT has always been taken with great care, the adoption of some of its preventive measure could actually be of a great use in ICS as well. One of these measures relate to the deployment of lightweight virtualization and/or sandboxing solutions in an attempt to create an additional and parallel level of safety for ICS components, allowing the execution of unsafe or untrusted code inside a protected environment.

An overview of two approaches to achieve confinement for software testing purposes are following presented for a better understanding of the idea presented above.

2.7.1 PEE within a Computer System

The idea presented here refers to an agent that installs itself within the device's file system to control the modifications performed to it, by intercepting the Inputs and Outputs (I/Os), and classifying the application as secure or unsecure according to a configuration utility [Jooste2008]. If the latter attempts to modify the protected execution environment, the agent terminates the original I/O request, creates a new one and redirects it to an alternate environment (cf. Figure 2-13).

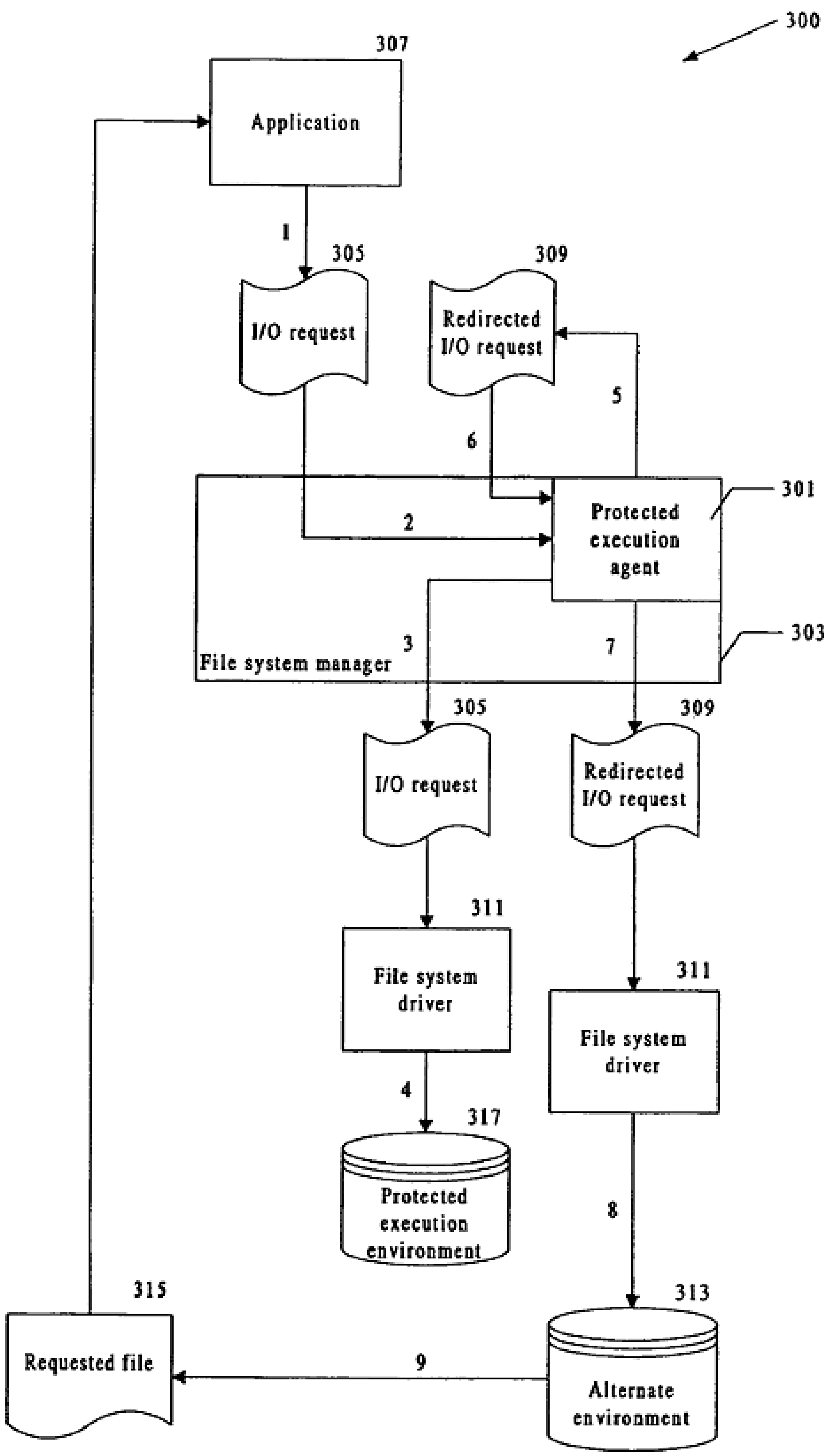


Figure 2-13 – Diagram illustrating a system-level overview of an embodiment of the invention [Jooste2008]

The configuration of the protected environment involves classifying the installed applications as authorized or unauthorized, by executing the utility method as system administrator (regular users are not allowed to perform such configuration). After the execution, a list of both types of applications is produced. Also, to aid the classification procedure, the load path of the process ID is used and an active process data structure is created and maintained to associate a process ID for an application with its load path and a parent-child data structure to track the relationship between an unauthorized application that launches another unauthorized application (parent-child relation).

2.7.2 One-way Isolation

This approach refers to the protected environment as Safe Execution Environment to describe a way to experiment new software without damaging the system. This is achieved by reproducing the application's behavior inside the safe environment via one-way isolation – once inside, processes are given read-only permission and, write operations are forbidden from escaping the environment [Sun2005]. The safe environment also includes commit and rollback functionalities, and a consistency criteria for the previous actions; environment reproduction, i.e., once found to be stable the application can run on the "original" file system; and confinement, as mentioned before.

The features presented above are implemented on an Isolated File System (IFS), created by interposing file system operations within the kernel at the Virtual file System (VFS) layer (cf. Figure 2-14). Also, the implementation of IFS requires the support of Copy-on-Write[3] (COW) optimization strategies.

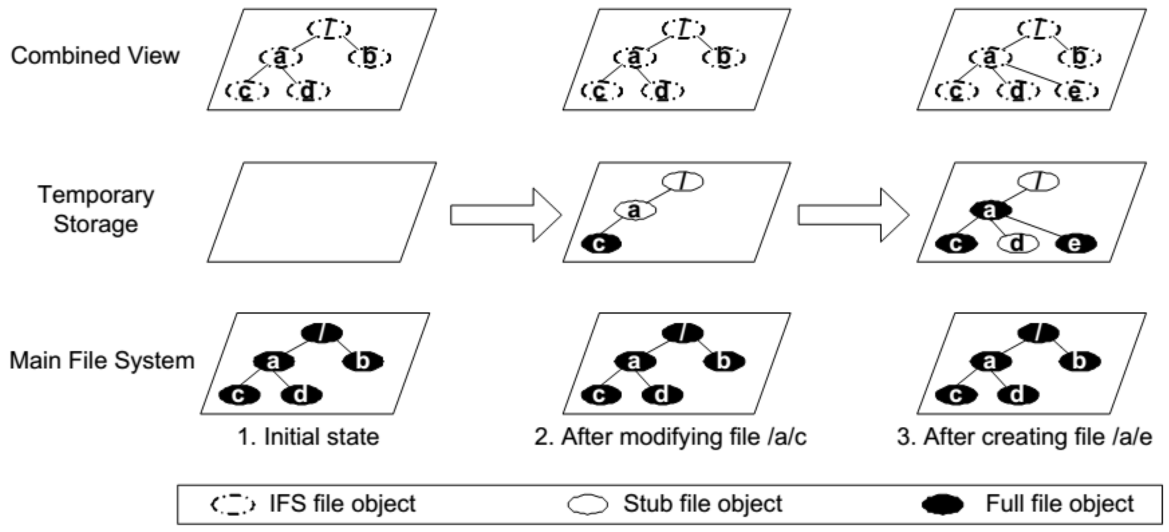


Figure 2-14 – IFS Layout on Modification Operations [Sun2005]

[3] Copy-on-Write is a strategy used when many separate tasks use similar copies of the same data stored in memory or disk. Instead of creating a different copies for each process, a pointer is given to the same resource.

The illustration in Figure 2-14 demonstrates the operation of the IFS: The Main File System corresponds to the host file system; the Temporary Storage holds modified copies of files and directories, overriding the view of the previous (i.e., the Main File System); and the Combined View corresponds to the combination of the previous two cases. The IFS operations are maintained in a table called “inode table”, indexed by the inode number of file system objects, with a field indicating whether the inode corresponds to an object in the Temporary Storage or in the Main File System. Other optimization techniques are used to make this approach as efficient as possible, while keeping low performance overheads.

2.8 Secure Execution Approaches

To keep the desired ICS components continuously running, without the risk of getting compromising by the execution of unsafe code, these algorithms should run inside a protected environment, giving the ability to experiment as the output gets confined. Many solutions have already been developed, mainly, in the context of desktop software (e.g., checking if new updates do not compromise the remaining applications running in the system, or to evaluate how vulnerable is the system to a new virus), keeping the host system safe. Most of these solutions are used for later reproduction of the actions carried inside the secure environment into the system itself (cf. section 2.7.2). However, most of these solution require high processing capabilities and/or provide unnecessary features that do not make sense for some ICS components, requiring a careful approach to balance both functionality and overhead.

There are popular solutions built-in most UNIX-like systems, such as *chroot* [Chroot] and more sophisticated sandboxing ones like *systrace* [Systrace], to run applications a somewhat confined environment. However, more advanced sandboxing techniques exist such as the ones described in the following section.

2.8.1 Virtualization

This is the most basic and maybe logic way to isolate the actions performed by an application in IT systems, on an OS (e.g., using Virtual Machines (VM)) to keep the code away from the real hardware. Some functionalities, namely, the firewall must be carefully configured to disallow any attempt to access the host environment or, if necessary, the virtual network interfaces could simply be disabled or removed. This gives the operator the ability to either isolate both systems completely or give a two-way isolation, between both host and VM environments.

A VM instance requires a kernel, a bootable OS, shared memory and processing with the host system. Lighter[4] implementations of a virtual system like *User-mode Linux* (UML) may not be feasible due to

[4] In the sense that UML lacks most of the features present in most virtualization solution (e.g., *Oracle VirtualBox* or *VMware*).

low performance issues when compared with, e.g., Linux Containers (cf. section 2.8.3), and the fact that some system architectures are yet not supported.

2.8.2 File System and Process Confinement

Using *chroot* to confine a process into a specific zone of the file system is not safe – using *ptrace* debugging tool [Ptrace] (used for intercepting system calls) on another process outside of *chroot*, one could easily get out of that same zone. To work around this issue, each process would have to be run as a different user to prevent it from attaching to another one using *ptrace*. On the other hand, this might cause a race condition when setting the new user. Nevertheless, since there is no way to prevent malicious code from opening network sockets such solution might be useless for most components requiring safe network access.

It is also possible to have hybrid solutions where *chroot* is used along with container calls (cf. section 2.8.3) for file system isolation, *namespaces* [Namespaces] for process isolation and *prctl* [Prctl] to disable process trace (*ptrace*). *Seccomp* [Seccomp] is another mechanism making use of *prctl*, that only allows processes to call read, write, exit and *sigreturn* functions (any other call terminates the process – *SIGKILL*). This approach does not even allow memory allocation, since it does not virtualizes any system resources (only process isolation is guaranteed).

2.8.3 Containers

Linux containers (LXC) [LXC] use a set of namespacing tools to achieve lightweight virtualization of a whole system or just a single process, by calling *unshare*[5] [Unshare] and *clone* [Clone] tools. By doing so, newly created network sockets are in a different *namespace* and, network isolation is achieved. The *cgroups* functionality is also used to limit and isolate the usage of resources (e.g., Central Processing Unit (CPU), memory and disk I/O).

Isolating process IDs is done properly since it can't see or *ptrace* anything outside of the container; File system isolation is also done properly by (un)sharing mount points and combining it with *FileSystem in Userspace* (FUSE) [FUSE], useful for writing VFSs without editing the system kernel – VFS provides an abstraction within the kernel space with an interface to *userspace* applications; for networking isolation, a properly firewalled virtual or physical network adaptor can be used.

From a container lockdown perspective, LXC is still not completely secure since root users still have access to *dmesg* [Dmesg] and the *proc* directory [Proc], allowing these to access */proc/sysrq-trigger* and restart the host machine[6] [Ramesh]. For such reason, some administrator choose LXC-based solutions (cf. section 2.8.4) to overcome this issue.

[5] *unshare* is a command line interface to unshare Linux *syscall* and allows a program to run with some parts of the process execution context unshared from parent.
[6] Running *echo b > /proc/sysrq-trigger* will reboot the machine.

2.8.4 Container-based Solutions

Other solution take advantage of the capabilities provided by LXC (cf. section 2.8.3) such as the ones described here:

- Linux-VServer – The first container-based implementation to appear for the Linux system that performs process and resource isolation through the capabilities inherent to the kernel (e.g., file system isolation using *chroot* or memory limits using *rlimit* [Rlimit]), as well as inter-container security via a global PID space that prevents other containers from seeing each other's processes [VServer]. Its biggest advantage is scalability for a growing number of containers. As for the disadvantages is the impossibility to implement live migration and partition checkpoint and, network subsystems are not virtualized (i.e., routing and IP table are shared with other containers).
- OpenVZ – Unlike the previous, isolation between containers is guaranteed using kernel *namespaces* and, features like migration and partition checkpoint are possible [OpenVZ]. Each container is allowed to have its own network stack (i.e., its own routing and IP table) and the host system can even assign a real network device, providing better network performance. Using the inter-process communication (IPC) [Ramankutty2004] kernel namespace capabilities, each container is also allowed to have its own shared memory segments, semaphores and messages. Resource management is much better than the previous solution (VServer), allowing for more specific configurations, such as limiting memory usage and various in-kernel objects (e.g., IPC shared memory segments or network buffers), and fair processing resources distribution per container.

2.8.5 Access Control Policies

To achieve high levels of security and confinement the *Security Enhanced Linux* (*SELinux*) [SELinux] allows the system administrator to take control over user and application access to specific resources (e.g., files), using policies that can't be modified neither by users nor (malformed) applications. An example, is the *targeted* policy used in most Linux distributions that puts a great number of confined daemons in a controlled state [Targeted].

Instead of specifying which users are able to perform read, write and execute operations over files, *SELinux* allows the specification of who can unlink, append or move the same resource (e.g., files, network resources and IPC). This fine-grained control over resources makes this solution great for confinement purposes and, for very modular architectures.

2.9 Containers Comparison and Evaluation

A complete study conducted in [Xavier2013] comparing a set of performance metrics between native LXC, LXC-based solutions (*OpenVZ* and *VServer*) and a truly virtualized system (*Xen*) is presented (cf. Table 2-2). These benchmarks were conducted to evaluate how much a single container could influence/interfere with another one results, with 6 different stress tests (CPU, memory, disk and send/receive network intensive tests and, a fork bomb test) for High Performance Computing (HPC) applications, using a Dell PowerEdge R610 [R610] with a 2.27 GHz processor (8 cores each), 16GB of RAM and a Gigabit Ethernet adapter.

Table 2-2 – Performance isolation for LU applications[7] [Xavier2013]

	LXC	OpenVZ	VServer	Xen
CPU Stress	0	0	0	0
Memory	88.2%	89.3%	20.6%	0.9%
Disk Stress	9%	39%	48.8%	0
Fork Bomb	DNR	0	0	0
Network Receiver	2.2%	4.5%	13.6%	0.9%
Network Sender	10.3%	35.4%	8.2%	0.3%

The conclusions discussed in [Xavier2013] demonstrate that the CPU stress tests do not impact any of the solutions. Only all the other resources have impacts when stressed, impacting well-behaved guests (containers), leading the researches to believe that while the kernel is handling the stressed guest calls, it won't be able to handle calls from well-behaved guests, influencing the performance results. The fork bomb test allowed the researches to validate the security issues on the LXC solution, due to the impossibility to limit the number of processes by *cgroups* – something that OpenVZ and VServer deal with perfectly.

It should be noted that carrying these performance tests to a device with reduced capabilities, would have a different impact and meaning over the analysis made to the results. Normally, these kind of evaluations are performed on devices with a lot of processing capabilities (e.g., in the context of cloud computing with emerging approaches to migrate from current virtualized solution to container-based ones) and so, adapting these to a different context would require a different approach.

2.10 Isolation Contexts

The various types of implementations described in the previous sections can be categorized depending on one out of two situations: if the isolation is applied at an existing user-space (App-level – restricting what a process or user can do) or if the user-space is entirely sandboxed (OS-level). In the case of VMs, these isolate both the user-space and the kernel-space creating a "full" sandboxed solution.

[7] LU – Lower-Upper Gauss-Seidel solver (a pseudo application).

Table 2-3 – Sandboxing implementations by context[8]

Context	Solution	Features						
		File-System isolation	Network isolation	Root isolation	Disk quotas	CPU quotas	I/O rate	Memory limits
Os & App.	Virtual Machine[9]	N/A						
OS-level	Chroot	Partial[10]	No	No	No	No	No	No
	LXC	Partial[11]	Yes	No	Partial	Yes	Yes	Yes
	LXC-based	Yes[12]	Yes	Yes	Yes	Yes	Yes	Yes
App-level	Seccomp	No						
	SELinux							

[8] Only open-source implementations, covered by the GNU General Public License (GPL) license [GPL] were considered, leaving aside FreeBSD jails [Jails].
[9] Virtualization is not an OS-level approach since it reproduces a completely new system, i.e., new kernel and OS.
[10] Using a root user account, one can easily work around the *chroot* confinement. However, this tool was never designed for this goal.
[11] The Linux kernel does not support user namespace separation. LXC has to be combined with FUSE.
[12] Native LXC solution have been recently subject to an attack as mentioned in [lxc2011].

3 CockpitCI Reference Architecture

3.1 Requirements for the Proposed Architecture

The proposed CockpitCI detection architecture takes special consideration with the integrated layers, providing an autonomous solution with self-healing and preventive capabilities at the Field layer, where the RTUs operate, making it difficult for attackers willing to take advantage of the deployed components to actually compromise these. To overcome this contradiction, i.e., Autonomy vs. Security, a hybrid schema was developed covering two levels: at first, at the Control center (or Control Room – cf. Figure 3-2), accurate assessments are performed to provide the operator with qualitative and quantitative measures with information from the field level, other infrastructures and smart detection agents; secondly, at the field level, a detection system is introduced to continuously analyze the inputs and outputs of the RTUs and prevent attacks from occurring [D3.1_2013] – in these situations, i.e., in case of attack, RTUs are supposed to behave accordingly, either ignoring commands or isolating themselves, until the operator comes up with a solution.

The integrated analysis and detection layers should deploy the mentioned smart agents and operate as close as possible to a real-time Distributed Monitoring and Perimeter Intrusion Detection System (PIDS) that receives the filtered and analyzed information of possible attacks. In this sense, both probing [D3.1_2013a] and detection [D3.1_2013b] architectures were conceived, in order to provide a set of security mechanisms to process the information sent by probes (or sensors) and other monitoring devices. While the first architecture (Probing architecture – cf. Figure 3-1) tries to address the problem of probe placement though zone separation, the second (Detection architecture – cf. Figure 3-2) builds on the previous to give relevance to the correlation structure, while introducing two innovative concepts, namely, the Shadow RTU and Backup Master Station (BMS), used for capturing and analyzing data (e.g., search for abnormal activity), respectively.

3.2 Probing Architecture

The Probing architecture is composed of three security zones to allow the traffic to be captured and the intrusions detected (cf. Figure 3-1): the IT Network is basically composed of HMIs, allowing the operator to check the system state. This network is not part of the SCADA system and still it may represent an entry point for attackers willing to compromise components inside it; the Operations Network is comprised of Master Stations and its complementary components used for data acquisition and visualization, i.e., the Database server and HMI; The Field Network contains the RTUs, Honeypot and Shadow RTU.

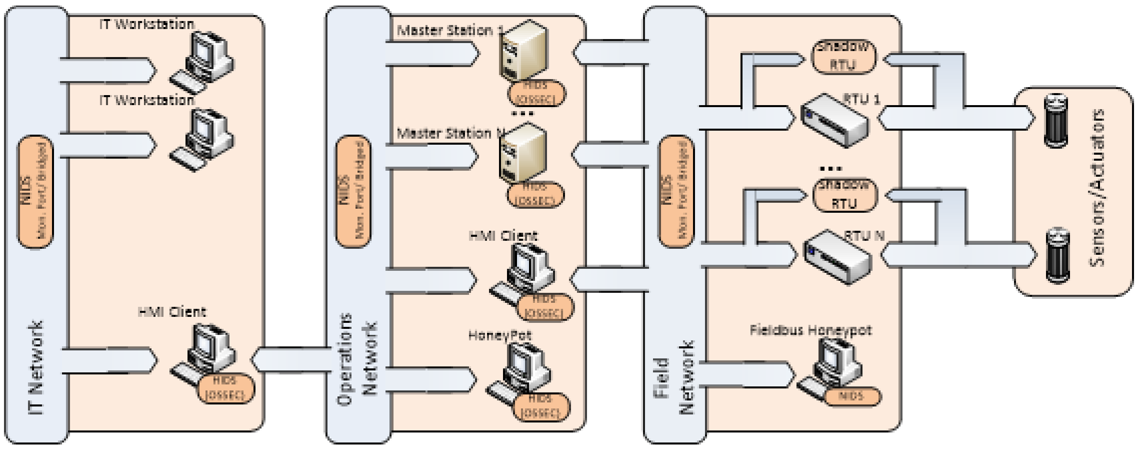


Figure 3-1 – Generic probing architecture [D3.1_2013a]

This separations was done to differentiate infrastructure contexts for handling with different detection, correlation and reaction strategies, and defining security perimeters to control the exchange of information between each zone. These perimeters were strategically positioned to address detection but also reaction and countermeasure mechanisms.

3.3 Detection Agents

Following is a brief description of each proposed component (or agent) operating at the Field Network, responsible for the detection of anomalies, and their respective roles on the infrastructure to give an understanding of the best suited mechanisms to deal with the detection of specific threats.

3.3.1 Network and Host Intrusion Detection Systems

The Network Intrusion Detection System (NIDS) is a sensor located at the boundary of each delimited zone to monitor the traffic flowing between these, using patter-based mechanisms (e.g., signature and anomaly-based) to search for unexpected behavior and activity, such as probe scans, DoS or MITM attacks. The Host Intrusion Detection System (HIDS) is used to search for the same type of anomalies as the NIDS does, but at the host level, e.g., on the Master Station, Database or HMI consoles, processing the device logs, performing signature checking, and monitoring key system structures, e.g., monitoring system calls between applications and the OS, to detect abnormal behavior (e.g., using machine learning methods), intrusions and the integrity of the system itself. This is the way to get basic control of data flows at both generic and specific levels.

3.3.2 Honeypot

The honeypots are conceived to be a dummy target for attackers trying to compromise real SCADA devices. There are three different kinds of honeypots, for each zone of the detection architecture, to simulate the operation of real devices, e.g., the Operations Network honeypot simulates a Master Station. However, Field Network honeypots are a bit different from the ones operating in the other two zones due to the nature of the network itself – at this level, honeypots simulate SCADA control devices (PLCs) unlike to what turns out in the remaining networks where it predominates an ICT (Information and Communications Technology) environment. Since the honeypots are just fake SCADA devices, these are not connected to real system components and do not perform any kind of production task and so, any received communication will have a high chance of representing an illegitimate operation. As in the honeypots operating at the IT and Operation Networks, at the Field level the range of operations must be contained and protected with a Layer-2 Firewall between the honeypot and the rest of the network, to limit the interaction with the attacker, preventing him to gain access to the system and remain undetected.

3.3.3 Shadow RTU

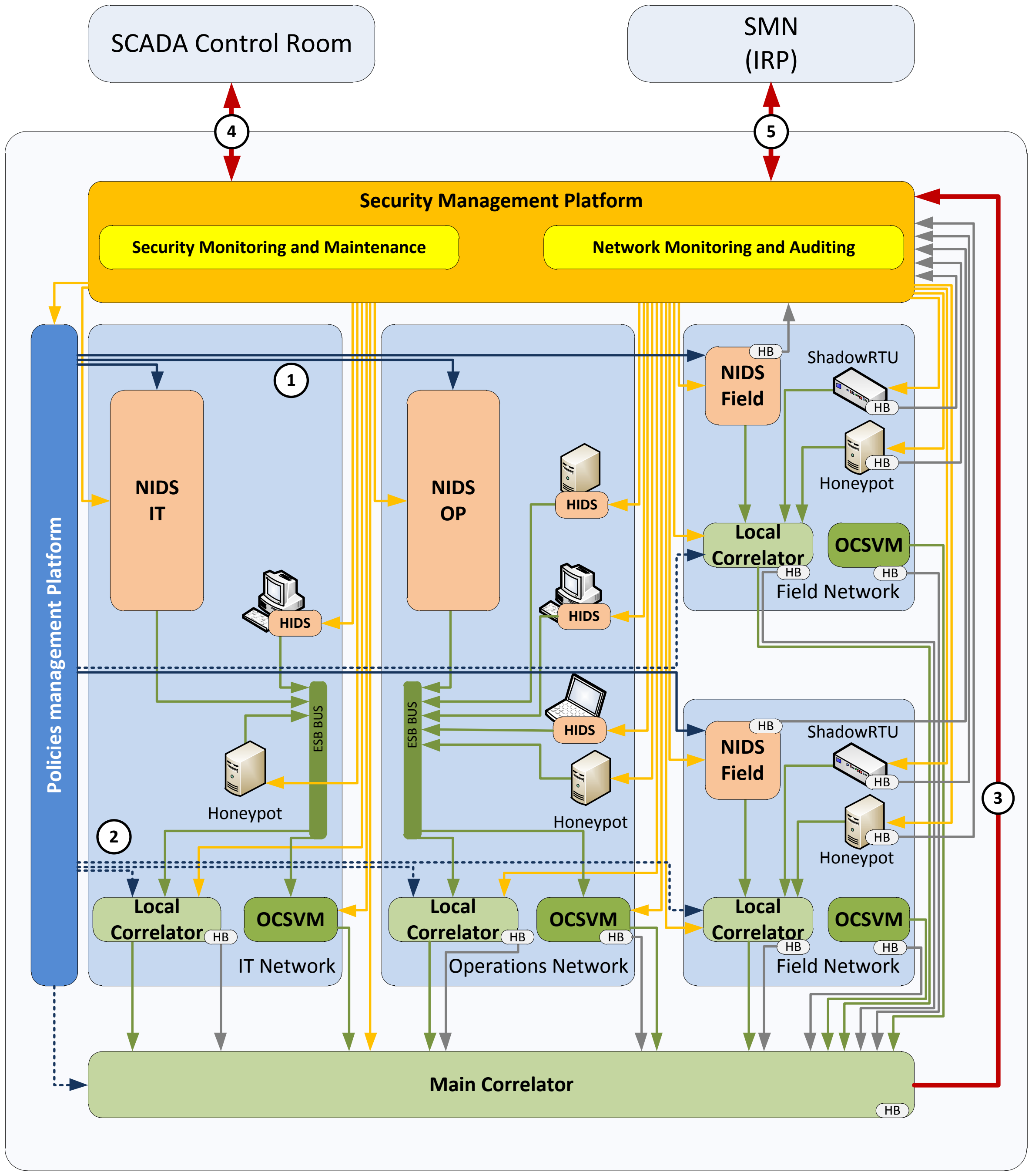
The Shadow RTU operates transparently to the entire system while monitoring the activity of the RTUs without any physical intrusion, receiving both inputs and outputs with a passive Ethernet Tap so these can be later processed and the security status reports sent to the security management infrastructure. This concept goes in line with another one, the Smart RTU, allowing the Shadow RTU to extend its capabilities with resilience and self-healing mechanisms. This is also the scope of this thesis and so, more details on this component are mentioned in the next chapter (cf. chapter 4).

3.4 Detection Architecture

The Detection architecture is composed of two levels of correlation to process and analyze the captured information provided by the security probes (cf. Figure 3-2): Local correlation is performed at each zone with the acquired network data, which is then processed for synthesis purposes, e.g., removal of duplicated events. At this level, it is possible that the Local Correlators also have decision and reaction capabilities, according to their zone scope; The Main Correlator receives the events from the Local Correlators, detecting network transversal attacks, i.e., attacks initiated at one Network zone throughout the others. In some cases, correlation may also be carried at the Agent or service level when attacks are specifically targeted to these and, where very specific rules are needed to conduct the correlation of events.

Considering the detection architecture the CockpitCI team had several attacks in mind, namely:

- Sending unauthorized commands to control devices, e.g., changing alarm thresholds, while displaying wrong information to the operator;
- Introducing perturbations on the network, delaying data flows and;
- Infecting devices with malicious software, e.g., viruses and worms.



Legend:

HB – Heart Beat Mechanism
NIDS – Network Intrusion Detection System
HIDS – Host Intrusion Detection System
OCSVM – One Class Support Vector Machines
ESB – Enterprise Service Bus

1	Detection Policies
2	Correlation Policies
3	Anomaly and Security Event Detection
4	Management Information and alarms
5	Processed events and IRP results

Figure 3-2 – Proposed CockpitCI detection architecture (Security management flows in red, network security information flows in green) (Updated v10.4 from the [D3.1_2013b])

Unlike the agents presented in the previous section (cf. section 3.3), there are other components with a domain specific behavior as it is the case of the Field Security Manager (FSM) that incorporates the Local Correlator, BMS and Heartbeat logic at the Field Network, inside a specific Autonomous System (AS). For simplicity and clarity purposes, the FSM is illustrated as a Local Correlator and, possesses a key role in the detection architecture (cf. Figure 3-2): the Local Correlator processes the events sent by both NIDS and Shadow RTU, of the same AS it is running at; the BMS provides some level of autonomy, guaranteeing that the system keeps running in the advent of an attack, e.g., if an attacker attempts to reprogram the RTUs, the control center isolates the AS and the BMS forces every device to reboot and restore to its normal operation; the only way for the BMS to know that the AS has been isolated from the rest of the network is through the Heartbeat mechanism, a periodic request-response signal. If for some reason the BMS fails to receive a response to a performed request, it assumes the AS is isolated and a set of predefined actions are followed (e.g., shutting down or rebooting every RTU).

In order to communicate the security events generated in the SCADA network, a Secure Mediation Gateway (SMGW) is used between the SCADA and the Secure Mediation Network (SMN), which receives the events in a safe way, allowing it to assess threats in a global scale. Also, the SMN is the only way by which the CockpitCI internal components have to communicate with their respective modules. The Integrated Risk Prediction (IRP) is a module designed to support the process of decision making, providing current situation awareness and risk assessments through the prediction of short-term situations.

The Security Management Platform (SMP) is used to manage the infrastructure components, making use of security audit and maintenance mechanisms (cf. Figure 3-2 – yellow boxes inside the SMP) which the operator supervises to check how effective the deployed measures (including the detection agents) truly are. The operator may also define a set of rules to detect and confine anomalies, using a policy management console.

The One Class Support Vector Machine (OCSVM) is a machine learning tool engine used for analysis purposes to adjust the risk levels of the architecture components according to a specific criteria, namely, the effectiveness and range of a specific attack.

One thing to be noted in this architecture is that there are no specific reaction mechanisms. However, it provides mechanisms for execution of reaction countermeasures, e.g., in case a security event is triggered and a fast reaction is required for a limited period of time. Due to these timing requirements and availability needs, the architecture presented above needs to be implemented on a separate network, e.g., using a VLAN, to avoid interfering with the control network, guaranteeing its normal operation.

3.5 CockpitCI Security Ontology

As mentioned in the previous section (cf. section 3.4) the CockpitCI team had a few attacks in mind for each detection agent of the presented architecture (cf. Figure 3-2). To defend the infrastructure against these and similar anomalies, the following security ontology (cf. Table 3-1) briefly describes the effectiveness of each mechanism for dealing with such threats [D3.1_2013].

Table 3-1 – Security ontology for the components of the CockpitCI cyber-analysis and detection reference architecture (Adapted from [D3.1_2013c])

Mechanisms	Cyber-threat/symptom	Reason
Shadow RTU	Sending of unauthorized commands; Master impersonation	The Shadow RTU is able to detect abnormal command activity from unauthorized origin.
	RTU reprogramming	The Shadow RTU is able to detect abnormal behavior from the monitored device.
	Abnormal delay	In some situations, the Shadow RTU may be able to monitor traffic and detect excessive delay.
	MITM attacks	The Shadow RTU is able to detect abnormal behavior from the monitored device, by monitoring commands and actions.
	Probe attacks	The Shadow RTU is able to detect abnormal command activity from unauthorized origin.
Honeypot	Sending of unauthorized commands	The presence of traffic on the honeypot is a sign of unauthorized activity.
	Master impersonation	
	RTU reprogramming	
	Probe attacks	
Network IDS	Sending of unauthorized commands	Domain-specific NIDS can monitor the command flow and detect these issues. However, conventional NIDS can also be useful in some cases, when the commands come from an unknown Master Station.
	Master impersonation	When properly configured, NIDS can detect abnormal command flows from unknown origins.

	MITM attacks	Depending on the nature of the attack, Domain-specific NIDS can track state changes on the command flow.
	Probe attacks	NIDS are able to detect traffic traces corresponding to such situations.
	IP protocol level attacks (Smurf, Address Resolution Protocol (ARP) spoofing, flooding, etc.)	When combined with firewalls, NIDS can be very effective in detecting and stopping such attacks
	DoS attacks	When combined with firewalls, NIDS can be very effective in detecting and stopping such attacks
Host IDS	Rootkits	HIDS are able to detect signature changes on critical system files.
	Tampering	HIDS are able to detect signature changes on critical system files or unexpected configuration changes.
	Remote Access Trojan (RAT) attacks	HIDS are able to detect signature changes on critical system files, but can also check for open TCP ports or unexpected configuration changes.
	Worms and virus	HIDS are able to detect signature changes on critical system files or unexpected configuration changes
	Unauthorized access	HIDS are able to analyze logs to check for unauthorized access.
FSM/BMS	Abnormal delays and interruptions	FMS/BMS has the means to correlate information about abnormal behavior on the AS, being able to (accordingly with established policies) to proceed with autonomous remediation.
	Unexpected systems isolation	FMS/BMS has the means to ensure safe operation levels are maintained (accordingly with established policies).

	Abnormal PLC behavior	FMS/BMS has the means to initiate remediation actions in such cases (accordingly with established policies), which can go to the extent of reprogramming the field device.
MultiZone Correlation	Zone-specific abnormal activity	System is able to detect and pinpoint specific problems, affecting a particular zone of the Critical Infrastructure (CI).
	System-wide abnormal activity	Enables detection and tracking not only of ongoing threats, but also of ab initio symptoms, related to probing and attack preparation. Global correlation provides a broad perspective on the security status of the CI, enabling detection of sophisticated behavior patterns, involving several zones of the CI.

4 Proposed Probing Architecture

4.1 Shadow RTU Concept

In order to increase the protection levels of the SCADA system, the CockpitCI detection layer uses the Shadow RTU, a device responsible for monitoring, collecting and processing the network data handled by a single RTU or PLC. To do so, the Shadow RTU uses a special device, an Ethernet Tap (cf. section 4.2.1), that allows it to listen to the exchanged information between the PLC and the Master Station while remaining unnoticed from both communicating ends. Once installed in the detection layer architecture, the Shadow RTU also connects with a Local Correlator and the SMP to report the security events and allow remote management by the system's operators (cf. Figure 4-1).

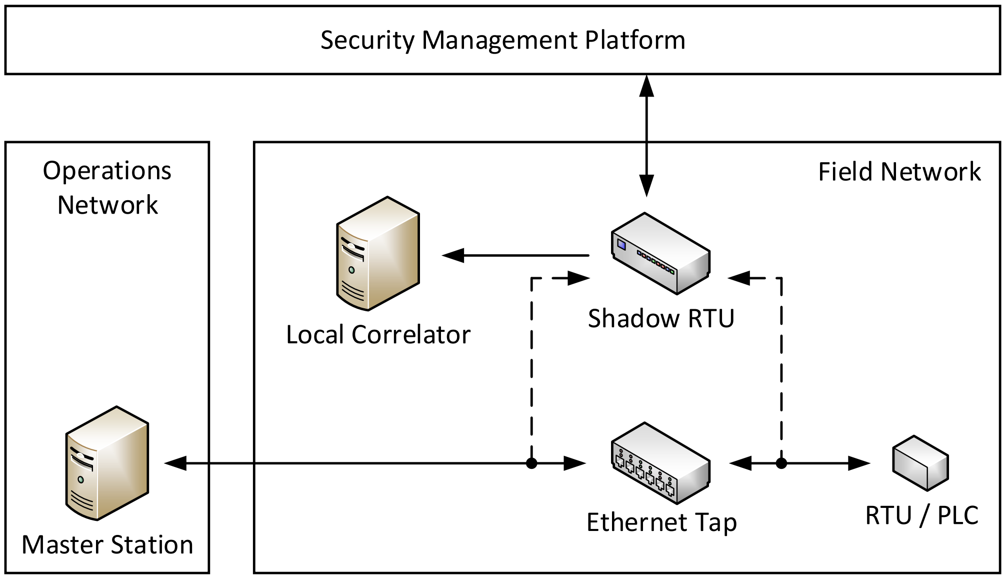


Figure 4-1 – Proposed Probing Architecture (Shadow RTU Concept)

Whenever requested by the Master Station, the RTU reads the values from the sensors, reporting these back. If these values need to the changed, the Master requests the RTU once again to regulate the values on the actuator, so these can be read once more from the sensor and reported back, again, to the Master. This process involves SCADA specific protocols, such as the Modbus TCP (cf. Annex A) or the IEC 60870-5-104 (IEC104) (cf. Annex B), which the Shadow RTU inspects to make sure the request are being performed by a legitimate Master Station and the responses from an equally reliable source.

Once the network information is collected, the Shadow RTU may either process it, sending the resulting events to the Local Correlator in a special message format, or it may perform a lower-level correlation itself and automatically inform the Control Room (SMP) of the anomalies that are taking place at the moment. The bidirectional flow between the Shadow RTU and the SMP is also justified by the Heartbeat mechanism, to inform the later of its operational status and, also, to allow the operator to manage the agent (cf. section 4.2.6).

4.2 PROBING MODULES

The definitions for the term “probe” may vary in the context of computer networking: it may be an action taken with the purpose of knowing the current state of the network (e.g., sending an Internet Control Message Protocol (ICMP) packet to a specific host to check if it is alive); or a mean to gain access to a device by taking advantage of a known weak point [Probes2000]. Relatively to the proposed architecture, it refers to a device (the Shadow RTU) installed on a key juncture of the network with the purpose of monitoring and collecting information about a specific process, i.e., communication between a particular host and a PLC or RTU, or a component’s activity, i.e., again, a PLC or RTU.

Generally speaking, probing methodologies provide the operator with a series of statistics concerning the network operation, such as traffic analysis and most active users, or applications and protocol usage, just to mention a few. If a specific device starts behaving differently or in a way that makes it suspicious, the operator is notified and the monitored information for the latest activity analyzed. This makes it possible to effectively manage the network, ensuring a proper operation and performance of all components involved in the process.

In this sense, a set of security modules were developed for the Shadow RTU to collect and store detailed data on exchanged protocol messages that can be analyzed to validate the true overall performance of the process being monitored. These probes, when combined, provide an accurate and secure solution for handling network events with reporting capabilities, even for high utilization links where some probing mechanisms would fail and leave the network vulnerable to attacks.

The following six modules were conceived to work together and give support to the purposes of the Shadow RTU, namely, to the monitoring and reporting capabilities, to keep the central point aware of anomalies that the PLC or RTU may be subject.

4.2.1 Network Event Monitoring

In its most basic mode of operation, the Shadow RTU simply listens to every network event targeting the PLC or the RTU that is being monitored, in a passive manner. For that, it uses a specific method that makes it completely transparent to the network and the process itself [Einwechter2002]. Considering this, a simple setup is proposed in which an Ethernet tap is installed between the monitored device and the one making the requests (usually a Master Station, through an HMI) to provide a copy of all network traffic to the Shadow RTU (cf. Figure 4-2).

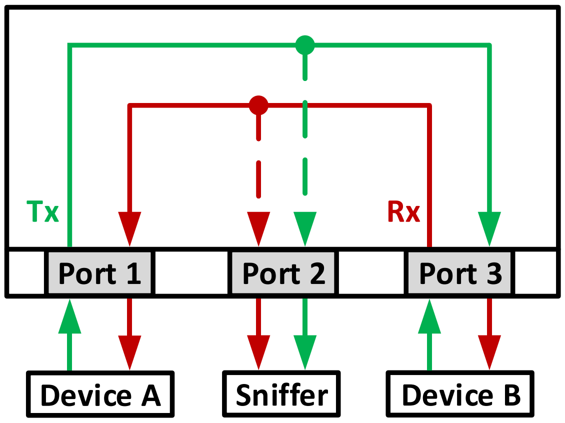


Figure 4-2 – Ethernet TAP device concept

The most basic setup for an Ethernet tap (or network tap) is illustrated in Figure 4-2: two ports for the communicating devices and a third one (monitor/mirror port) for the Shadow RTU (in this case, also referred to as a Packet Sniffer) to listen for the flowing traffic. This method does not only allow the Shadow RTU access all the information flowing between device A and B but also makes it physically impossible to introduce data on the wire, complying with the above mentioned requirements.

In turn, the Shadow RTU works in promiscuous mode which causes all the traffic to be received and processed by it even if the frames have a different destination Media Access Control (MAC) address, i.e., not intended to be received by the Packet Sniffer. This method makes it great for legacy systems since no modifications are required to the existing network.

4.2.2 Network Data Collection

Providing a redundancy mechanism for the Shadow RTU is essential due to the amount of information it has to deal with while monitoring the network events, and so prevent data loss in the advent of an unexpected shutdown of the device or, the zone in the file system to store the events gets compromised. To cope with this, a copy of the events is maintained in buffer until a specified threshold is reached and the information dumped into disk (cf. Figure 4-3 – read and write functions refer to the operations performed by the application process; pcap_set_buffer_size function sets the buffer size to use [pcap_set_buffer_size]) [pcap]. Measuring the network activity is also necessary to manipulate the amount of buffer size to use in the process of traffic monitoring.

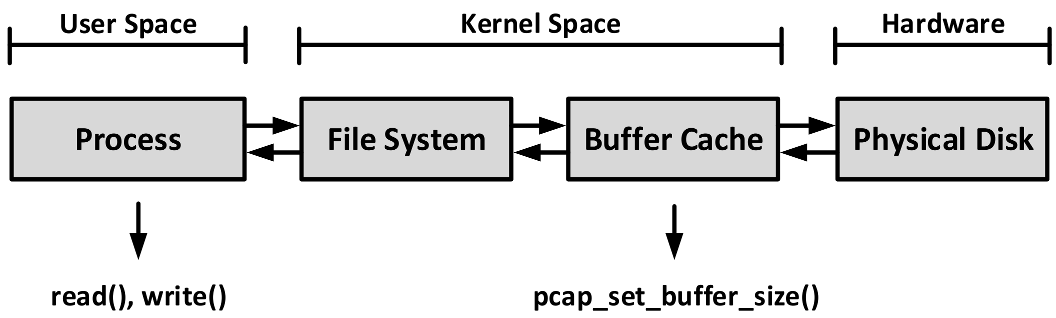


Figure 4-3 – Generalization of the Linux file system implementation

Since the amount of information collected by the Shadow RTU can be quiet large, there is also a filtering syntax mechanism [pcap-filter] to select only the type of network events of interest to be stored and processed afterwards.

After the event filtering, a second stage is responsible for backing up the network events into an external server and synchronized using sophisticated update mechanisms [rsync]. With this method it is also possible for the Shadow RTU to pull specific blocks of data, make the desired changes and then upload the performed modifications.

4.2.3 Network Data Processing

Depending on the resulting overhead, tasks can be carried by dedicated devices with specialized processing capabilities. This concept is fundamental to understand which tasks are meant to be performed by the Shadow RTU and which should be processed by more capable devices – E.g., event correlation is a heavier process and so should be carried by the operating Correlators; other tasks such as network event monitoring (cf. section 4.2.1), filtering (cf. section 4.2.2) and network message integrity check are carried by the Shadow RTU since the required processing levels are lower, allowing the device to operate properly for other tasks.

To look for inconsistencies in the exchanged Modbus TCP messages, a protocol decoder is used to allow the operator to manipulate and see the information inside each frame [pymodbus]. So, for each flowing network packet, the decoder is applied and a simple verifications are done.

4.2.4 Network Event Configuration

As a result of the monitoring operation conducted by the Shadow RTU (cf. section 4.2.1), it is also necessary to provide the operator with means to (pre)configure the device (e.g., by uploading code into it) to carry simple tasks in the domain of system and/or network status and reporting – E.g., collect physical magnitudes every five seconds and report these for later analysis.

Making these events work on the Shadow RTU may be done by simply enabling basic modules to extend the kernel functionalities [lkm].

4.2.5 Network Event Programming

In case more sophisticated thresholds are needed to handle the network events (unlike the process of event configuration (cf. section 4.2.4)) a safe execution environment, which is illustrated in Figure 4-4, is added to prevent malicious or malformed code from compromising the Shadow RTU [lxc], making it inoperable. Since the "eventing" code is designed to control a specific process (i.e., a PLC or RTU), it should be possible for the operator to remotely execute it so that it is activated using a specific logic (in this case, with asynchronous calls – cf. section 4.2.6) when the event occurs – E.g., if a third message to write on a specific register (or coil) of a PLC is detected, the appropriate handler (i.e., an alert) will be raised to inform the SMP.

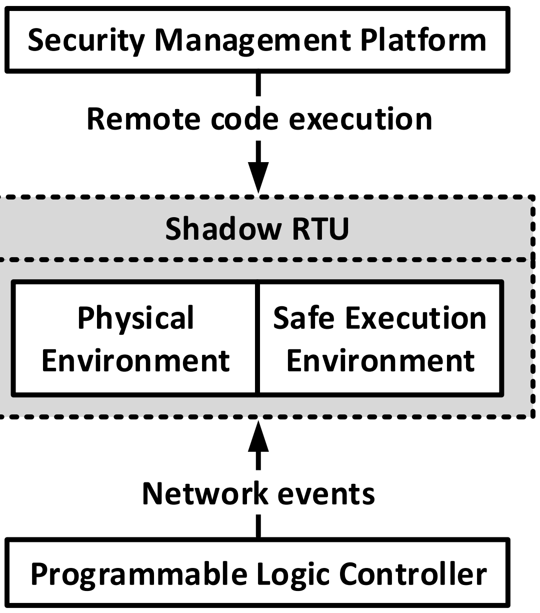

Security Management Platform
Remote code execution
Shadow RTU
Physical Environment
Safe Execution Environment
Network events
Programmable Logic Controller

Figure 4-4 – Representation of the safe execution environment for handling complex or unsafe code using asynchronous calls

Allowing code to be remotely executed poses an additional attack vector that, in order to be succeeded must compromise both the RTU (or PLC) and the Shadow RTU, which makes this very unlikely to occur since each component is based on a different technology or platform.

The idea behind the protected environment is for it to be connected to the inputs and outputs (I/O) and block any attempt to change the physical environment by the execution of unsafe code. With this approach the operator is left with the responsibility of defining a modular representation for the behavior of the complex algorithm and, to determine which network events are considered as normal (or abnormal).

4.2.6 Event Execution

The set of features to manipulate the Shadow RTU are described on a data model (cf. section 4.4) provided by the agent itself and a component management adaptor, also responsible for maintaining its proprieties (i.e., state and semantics). This adaptor performs the attribute mapping for the specific interfaces of each component of the architecture (in this case, the Shadow RTU via an Extensible Markup Language (XML) file) into a uniform data model structure (an XML Schema Definition (XSD) file), creating a uniform management interface [D3.4_2013]. The execution of the functionalities referred in the data model is performed through a WS installed in the Shadow RTU and the calling of Hypertext Transfer Protocol (HTTP) methods from the SMP (cf. Figure 4-5).

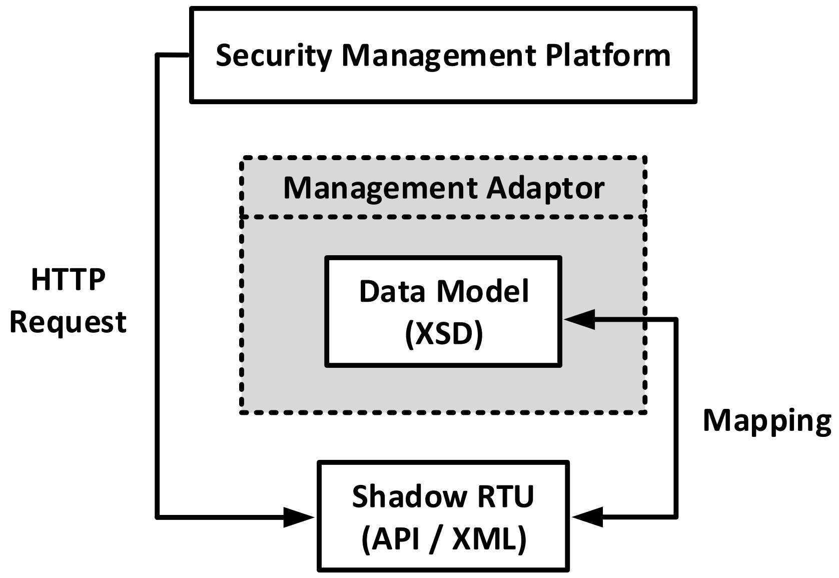


Figure 4-5 – Uniform management interface through the Management Adaptor

For each implemented HTTP method (Get, Post, Put and Delete) the XML file is manipulated and an operation is carried inside the Shadow RTU, e.g., to start a network monitoring routine, the operator must call the HTTP PUT method against the XML Path Language (XPath) expression that leads to the "State" tag, relative to the monitoring operation, and change its default value from "down" to "up". From this point on, the operator may check the service state using the GET method and, read directly from the XML file; call the PUT method to stop the execution; or even the DELETE method if he finds no need to keep such service in the data model. To add a new service into the data model, the POST method is used.

4.2.7 Event Reporting

In response to the events handled (or processed) by the Shadow RTU, reporting mechanisms are used to either alert the SMP or to simply forward information to be processed by the Local and Main Correlators and, the generated events by these reported to the SMP. The exchange of event information between the Shadow RTU and the Local Correlator requires the implementation of a special library on the Agent (Event Bus Publisher Library[13]) to produce events using a general XML

[13] Previously called Enterprise Service Bus (ESB), a software architecture model, has stated in "*D3.4 – Design of the Dynamic Perimeter Intrusion Detection System*" (Submitted: July 19, 2013). The illustration in Figure 4-

data format, the Intrusion Detection Message Exchange Format (IDMEF) [RFC4765], in a Service Oriented Architecture (SOA) (cf. Figure 4-6).

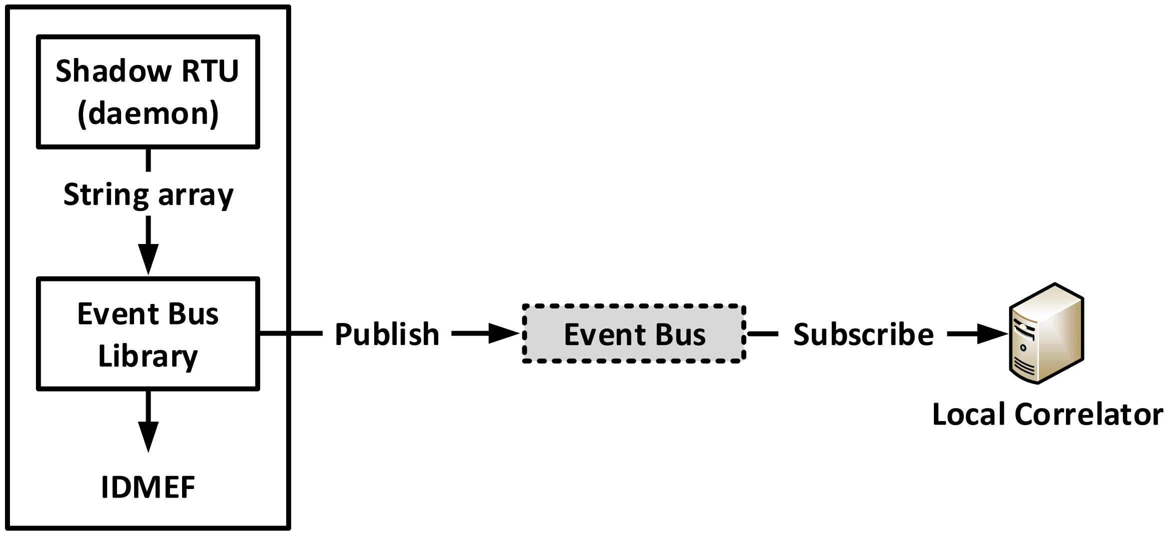


Figure 4-6 – Event transmission using a Publish-and-Subscribe messaging pattern

Since the Correlators do not support IDMEF natively, the library is used on the Agent to parse and filter the network events, which are then sent to the Event Bus using a Publish-and-Subscribe messaging pattern, e.g., once the daemon running on the Shadow RTU executes, it passes an array of strings related to event to the Event Bus Library to be read and converted into the new format (IDMEF) and, finally sent to the Correlator through the Event Bus.

4.3 Motivations and Requirements

Designing a new module to enhance the security at specific junctures of the network, without any need to change the regular operation of the existing one, e.g., modifying the architecture or the existing physical connections, requires coming up with an approach that on the one hand proves to be easy to set up and that, on the other hand, provides an understanding on how effective and reliable the new concept truly is – one must consider that this component is supposed to be uninterruptedly running for many years in a production environment.

Considering that the proposed Detection architecture (cf. Figure 3-2) already had other powerful security agents operating in both Operation and Field Networks (i.e., NIDS, HIDS and honeypot), the introduction of the Shadow RTU as a Field device must present peculiar and distinctive advantages over these. Some of the key features to be supported by this agent are described below. These will

6 was edited according to the latest developments, has discussed with the developer responsible for that module.

be in line with the presented Security Ontology for the components of the CockpitCI detection reference architecture (cf. Table 3-1):

- *Packet Sniffer* – Transparently sniff network packets at the lowest level (including physical errors) and still remain unnoticed from the entire system (i.e., none of the communicating devices are aware of this device);
- *Packet Decoder* – While the packets are being captured, the Shadow RTU performs the decoding using protocol implementations for Modbus TCP or IEC104, depending on the protocol used in the SCADA system where the Shadow RTU is deployed. This may be used to perform tampering checks, i.e., verify if the Modbus TCP packets have not been changed in transit. If a Modbus TCP instruction is sent with some of the parameters modified, these are immediately checked and an alarm is triggered;
- *Continuous Monitoring* – Monitoring the predictable bandwidth usage and traffic patterns, would allow the Shadow RTU to trigger an alarm if some of these parameters changed dramatically. In such cases, it could be an indicator of a malware infection, Distributed Denial-of-service attack (DDoS), brute-force, or a random equipment failure, e.g., a device that usually responds in intervals of 10 seconds and, suddenly, changes to a frequency of 20 seconds (also an indicator of excessive delay) [Higgins2013]. This can also be used to detect zero-day threats, such as the Stuxnet worm (cf. chapter 2).

  Other temporal patterns could be included, such as: Inter Message Time (IMT), Inter Reconnection Time (IRT), Time To Failure (TTF), Number of Lost Messages (NLM) and Total Number of Loss Messages (TNLM);
- *Message Integrity Check* – In result of an IP Spoofing attack, where an attacker sends unauthorized commands to a RTU, the Shadow RTU allows the detection of inconsistencies between the messages sent by the Master and the ones received by the RTU/PLC;
- *Abnormal Behavior Detection* – A usual way to intercept a connection between two end devices (e.g., a Master and a Slave device, such as a PLC), is commonly done by adding an extra agent between the conversation, as it occurs in MITM attacks. To change the way one of these components acts, one could try to inject new code into the RTU or PLC to be loaded on boot. RTU or PLC reprogramming attacks could be done by accessing the polling/communication circuit or by replacing the valid programming files for the Slave with malicious ones so that in the next reboot, these are immediately downloaded [Shaw2004]);
- *HMI Exploitation/Modification Detection* – In some cases the HMI may be perceived as a gateway into the ICS environment, using dictionary attacks (e.g., brute force attacks) against it to log in using default credentials [Wilhoit2013]. In this case, attackers are looking to exploit two types of vulnerabilities: SQL injection – E.g., the Master is a victim of a successful brute force attack, through the HMI (Master Impersonation). In most cases, this type of attack is often targeted to the backend database; Cross-Site Request Forgery (CSRF) – malicious

exploitation of a website by transmitting unauthorized commands from a user that the site trusts.

Some of the mentioned attacks could also be detected by other components of the CockpitCI detection layer, such as the honeypot: sending unauthorized commands (e.g., using the Master IP address – Master impersonation) or a Probe attack to collect information about the network activity (monitoring), in an attempt to gain access to a specific device (e.g., the Master Station). What follows from a "successful" series of probes could be a (D)DoS attack. The main difference relatively to the honeypot in terms of detection, is the ability to perform inspection at a lower level, providing an additional layer of security (at a deeper level).

4.4 Management Interfaces

The Shadow RTU provides a series of management functionalities, organized in three categories, included in a data model and, manipulated using WSs. The first set of features are related to the management of the device itself, while the second and third set relate to the applications developed by the operator to manage the events and the monitored process.

4.4.1 System Management

The operator is provided with a set of functionalities to perform out-of-band management for the most basic parameters of the system, using a dedicated channel. The functionalities for this interface are described below:

- *Set Agent State* – Allows the operator to reboot the agent and keep track of the last time the signal was sent;
- *Set Remote Access* – Set up a private VPN with all the necessary certificates and keys for both server (Shadow RTU) and client. The parameters provide the path for the generated certificates and keys;
- *Set Network Interfaces* – Set a series of network parameters for both interfaces (i.e., management and monitoring interfaces). These parameters include: interface name (e.g., eth0); type (e.g., static or dynamic); IP Address; Mask; Gateway; State (up or down); and Promiscuous mode (yes or no);
- *Set Time* – Synchronize the time and date with a preconfigured set of Network Time Protocol (NTP) servers list, to coordinate the events and logs with the correct timestamps. Other parameters include the state of the service and the last time and date the Shadow RTU synchronized with the time server. Changing the time servers must be done carefully, since it may cause time-based events (e.g., *cron*) to be run more than once, or not at all [Ntpdate2005];
- *Set Task Schedule* – Manage the tasks (jobs) the system is supposed to execute on a regular basis using the *crontab* format, e.g., backup the event logs to an external backup server every

day of the week, at 8 PM. The parameters include the *crontab* fields (minute, hour, day, month and week); the name of the application to run; and the last time and date it executed;

- *Set Startup Scripts* – Enable/disable applications at system startup depending on whether these are required or not by the system. The parameters vary depending on the application – E.g., one of the applications that must be running at the startup for the Shadow RTU, is the WS itself to allow the operator to call the HTTP methods on it. For this specific application the parameters are: the listening IP address; listening port; number of concurrent processes it can handle; and the state;
- *Set Kernel Module* – Create and compile new modules into the kernel to extend its functionalities. The parameters include: the name and path of the module; an indication if it is used by any other module; the log file; and the state (i.e., if it is already loaded into the kernel).

4.4.2 Event Management

The following parameters are used to identify the properties of every application uploaded into the Shadow RTU by the operator. Each one of these applications are responsible for handling specific events.

The following parameters can be used to manage an event:

- *Set Event Name* – The name of the application must be self-explanatory for easy understanding of what it is supposed to do, e.g., the script “mbus_monitoring.py” monitors Modbus TCP packets and dumps the information into a file, which is then parsed by the “mbus_parser.py” script;
- *Set Event Description* – A textual description of what the application is supposed to do;
- *Set Event Log* – Specify the name and location to store the log file of the application;
- *Set Container ID* – Identifies the container in which the application is running. It is not mandatory to run every application inside a container, since it is the operator’s responsibility to decide what should be confined to prevent malformed code from compromising the Shadow RTU;
- *Set Event State* – Set the application state to “up” or “down” to easily identify if it is running;
- *Set Time and Date* – Identify the last time and date the application was executed.

4.4.3 Container Management

To execute code in a safe environment, a set of parameters are required to create the containers. Each container is identified by an integer associated with the event application (Set Container ID – cf. section 4.4.2), along with other parameters:

- *Set Container Name* – Identifies the container in which the applications will run. By default, the containers use the root user account so, when logged in the console will present "root@containerName";

- *Set Container Description* – A textual description of what the container is for;
- *Set Network Interface* – Set the IP, Mask and MAC address for the container. This interface will be automatically bridged to the Shadow RTU;
- *Set Container State* – Start or stop the container execution;
- *Set Container on Boot* – Run the container automatically when the Shadow RTU boots.

5 Concept Validation

5.1 Hardware Validation

The operations performed by the Shadow RTU do not require high performance hardware (e.g., CPU, memory and storage capacity), leaving room for additional functionalities without creating too much overhead. In this sense, it was conducted an analysis over a series of components and methodologies, to create the most reliable solution, so that it can be safely deployed in production environments.

5.1.1 System Requirements

Considering availability and reliability requirements necessary for the proper operation of ICS/SCADA systems, such as a component's fast response time and behavior under stress conditions, deciding which should be the most appropriate device to take the role of the Shadow RTU requires a careful analysis over the available hardware solutions. This choice should at first take in consideration the cost, size and processing capabilities, to comply with some of the initial requirements. That said, a research was conducted to evaluate some of the existing Single-board Computers (SBC) (also referred to as System on a Chip (SoC)), leading to an initial specifications comparison between two Arduino devices (Arduino Uno [Arduino] and Intel Galileo [329676-002US]), a BeagleBone Black [Coley2014] and a Raspberry Pi [RaspberryPi] (cf. Table 5-1).

Table 5-1 – Hardware specifications for a series of SBC

	Arduino Uno	Intel Galileo	BeagleBone Black (Rev. C)	Raspberry Pi (Model B Rev. 2)
Price	~ $30	~ $70	~ $45	~ $35
Size	2.95” x 2.10”	4.2” x 2.8”	3.4" x 2.1"	3.37” x 2.125”
CPU	16 MHz ATMega 328	400 MHz Intel Quark X1000	1 GHz ARM Cortex-A8	700 MHz ARM11
RAM	2 KB	256 MB DRAM	512 MB DDR3L	512 MB SDRAM
Flash	32 KB	8 MB (also uses a microSD card)	4 GB (also uses eMMc and SD cards)	(only uses an SD card)
Input Voltage	7-12 v	5 v	5 v	5 v
Minimum Power	42 mA (0.3 W)	3000 mA (15 W)	2000 mA (10 W)	700 mA (3.5 W)
Digital GPIO	14	14	65	8

Since the Arduino project started in 2005 as project for students, other companies have also started building their own SBC microcontrollers, keeping the board size, price and electric consumption low, while investing in better (more capable) hardware [Orsini2014]. However, there are peculiar differences between some of the initially released boards (e.g., Arduino Uno) and more recent ones (e.g., Raspberry Pi). While the first ones were not conceived to be a computer, the latest boards already support a complete OS, allowing these to be programmed in a completely different way.

It should be noted that having the most capable device in terms of processing power, does not always mean having the best suited solution for every situation. There is another set of variables that must be equally considered when it comes to meet a different kind of requirements:

- Does it require some sort of background knowledge to take advantage of the device capabilities or, is it simple to start working with?
- Is it an embedded system designed specifically for developers or, is it a fully-fledged computer running a well-known and recent kernel?
- Is there any kind of support for the device (e.g., communities of enthusiasts devoted to give support and help other users)?

- Does it possess adequate hardware capabilities, balanced with reasonable power consumption levels?

Most of the applications running on the Shadow RTU will require reasonable processing capabilities and, two network interfaces (one for remote access and the other for network monitoring purposes) to operate – both requirements put aside the Arduino Uno, mostly because of its slow processor.

Even though all the remaining boards have an Ethernet network interface, the one on the Raspberry Pi is actually a built-in USB-Ethernet adaptor which means that, the greater the network activity, the higher the CPU consumption levels – this is where both the Galileo and BeagleBone have an advantage. As for the Intel Galileo, even though it has a slower CPU frequency and half the memory, relatively to the other two, it doesn't make it particularly slower. In fact, the x86 architecture brings a great advantage to its side but neither the price nor the required power to operate, make it a feasible option.

The final decision lies solely over the Raspberry Pi and the BeagleBone, being the advantage over the last. The decisive factor here relies on the processing capabilities (CPU) and the fact that it possesses on-board flash storage, enough to host a complete Linux distribution (e.g., GNU/Linux Debian). Both price and power consumption do not make it an unfeasible option as the Intel Galileo did, since those are not that different from the Raspberry Pi. However, by the time the validation process started, the initial versions of its latest revision (Rev. C) were being released (April, 2013), making it easier and faster to acquire the Raspberry Pi instead. The latest revisions were only commercialized in May, 2014. Nevertheless, all developments were made towards the possibility of porting these to the BeagleBone – in fact, this SBC will be the basic platform for a new generation of the ShadowRTU prototype, which exceeds the original specifications for the first one. This new generation is under development and makes use of three particular BeagleBone Black characteristics that make it stand out among other SBCs:

- *Programmable Real-Time Unit and Industrial Communication Subsystem (PRU-ICSS)* – The PRU-ICSS consists of dual 32-bit RISC cores (Programmable Real-Time Units, or PRUs), shared, data, and instruction memories, internal peripheral modules, and an interrupt controller (INTC). The programmable nature of the PRUs, along with their access to pins and events, provides flexibility in implementing custom peripheral interfaces, fast real-time responses (without need for using real-time OS for the purpose), power saving techniques, specialized data handling and DMA operations, and in offloading tasks from the other processor cores of the SoC. The BeagleBone Black's TI chip (TI AM335x ARM® Cortex-A8 processor from the Sitara Family) has two PRUs (PRU0 and PRU1) that can be programmed using a small, deterministic instruction set architecture. Each PRU can operate independently or in coordination with each other and can also work in coordination with the device-level host CPU. This interaction between processors is determined by the nature of the firmware loaded into the PRU's instruction memory.

- *Ethernet implementation* – Unlike the Raspberry Pi (which implements its Ethernet interface using an USB-Ethernet Application-specific integrated circuit (ASIC) (prone to performance and overhead issues), the BeagleBone implements its Ethernet interface in the form of a dedicated component of the main processor. This brings a significant benefit, as it allows to reduce the overall capture overhead on the Shadow RTU by a significant margin). This is one of the most serious shortcoming of the Raspberry Pi, as there is a significant CPU overhead caused by the fact that the embedded USB-Ethernet ASIC is not able to perform DMA transfers (albeit the host USB chip is able to do it), having to encapsulate Ethernet packets within USB protocol frames.
- *Embedded 12-bit Analog-to-digital converter (ADC)* – Unlike the Raspberry Pi (which requires an external ADC such as the MCP3008), the BeagleBone incorporates a 7 channel, 12-bit ADC, which is very useful for physical monitoring of the control outputs of a PLC/RTU, while reducing the part count. Together with the PRU-ICSS, the embedded ADC makes it becomes possible to capture information from ADC within strict, real-time constrains without depending on a real-time OS. Nevertheless, the Xenomai RT patch is entirely compatible with the Beaglebone Linux Kernel, and might be used to further improve real-time capabilities of the device.

As such, these capabilities have turned the BeagleBone Black into to best candidate SBC to host the next interactions of the Shadow RTU.

5.1.2 Network Monitoring Options

Using port mirroring capabilities on a switch connecting two devices to monitor the communication, might not provide the expected results since, in some cases, it is expected to get a copy of both links, from device A and B, on a single one, connected to the switch. With such configuration an operator risks downgrading the network's performance and losing packets on the monitoring device. Also, one has to take special attention to the fact that the links being monitored cannot exceed the performance of the port mirroring device. On the other hand, using a passive network tap it is possible to avoid bottlenecks and packet loss, allowing the communication to go on even in the absence of power (in case a robust tap is used).

A comparison of both tap and port mirroring capabilities (pros vs. cons) are summarized and presented bellow (cf. Table 5-2).

Table 5-2 – Tap vs. Port Mirroring

	Tap	Port Mirroring
Pros	Less risk of having dropped packets.	Low cost solution. Embedded in more robust switches.
	All packets including physical errors are received by the sniffer.	Configurable from any device connected to the switch.
	Full visibility into full-duplex networks.	Intra-switch traffic copy.
Cons	If not using an aggregator tap, it may be necessary a dual-receive interface when using a full-duplex tap.	Packets are dropped when handling heavy utilized full-duplex links.
		Physical errors are filtered making it difficult to perform some types of analysis.
		When copying packets the load is placed over the switch's CPU.
	Requires additional hardware.	Lower priority is set in port mirror ports than on regular ones.
	Cannot monitor intra-switch traffic.	Frame timing interaction can be changed, modifying response times.

Depending on the network characteristics each option should be deployed accordingly: a network tap is a good solution when the analysis requires inspecting all traffic (including physical-layer errors) on a network whose utilization may range between moderate and heavy; for an effective compromise between a tap and port mirroring, an aggregator may be used (described below – cf. section 5.1.3), providing some of the advantages of a tap and none of the disadvantages of port mirroring; in case the analysis is not affected by packet loss, port mirroring can be considered as the right solution for low utilization networks.

5.1.3 Passive Ethernet Taps

An easy way to build a passive Ethernet Tap involves cutting a RJ45 wire [Hamilton2007] and attaching two Ethernet connectors to the transmission (Tx – white with orange stripe) and reception (Rx – white with green stripe) pairs (cf. Figure 5-1). This method doesn't allow high speeds, since only two pairs are being used, leaving the remaining ones untouched. Also, it would be unthinkable to put such solution in a production scenario for many reasons, e.g., the wires connected to the Ethernet connectors are not shielded and so, are subject to external interferences. Nevertheless, this was actually the way the initial tests were conducted to evaluate how useful an Ethernet Tap would be when deployed in a scenario as the one presented in the next chapter (cf. Figure 5-2). There are many other types of Ethernet Taps that provide higher speeds and zero packet loss, according to some vendors [Securicore], which might not be exactly true since there is always a 0-1 probability of losing packets, due to a 5-10 milliseconds switchover time. In high availability networks this can be a bigger problem, causing routers and switches to renegotiate their links (e.g., VLANs and Spanning Tree) [Gómez2005].

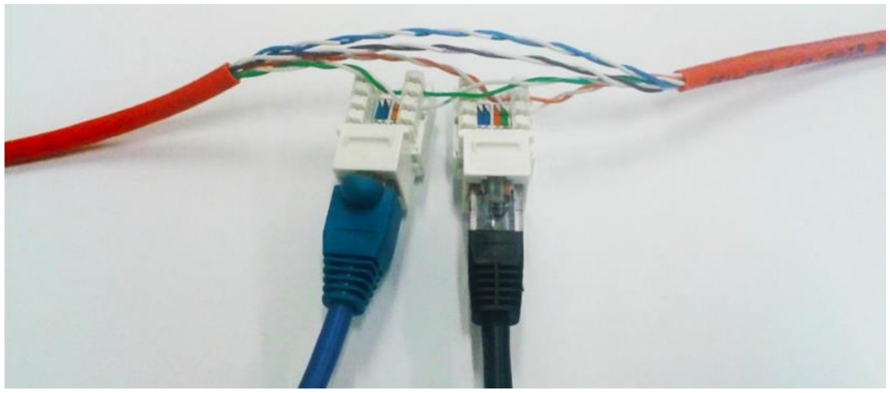


Figure 5-1 – Ethernet Tap using connectors attached to the Tx and Rx pairs

Ethernet taps may be setup using various approaches, apart from the one mentioned above. These may even use a throwing star shaped circuit board, specifically designed to be soldered with Ethernet connectors and capacitors, and still produce the same effect [Ossmann2009]. In a production network, more sophisticated solutions should be used, with higher levels of precision to avoid losing packets in the presence of high network activity. Following are described different types of network taps [Gómez2004]:

- *Adaptive Tap* – Used to convert signals (e.g., from Gigabit-Tx to Gibagit-Sx, or from Gibagit-Lx to Gibagit-Sx) and capturing network traffic.
- *Regenerator Tap* – The idea is to generate multiple streams of the monitored network traffic from a single point, using a single Tap or, two or more Taps [Netoptics]. It can be used when it is necessary to analyze traffic in both directions with different machines (e.g., for both intrusion detection and protocol analysis).
- *Aggregation Tap* – Used to monitor both directions on a single port, unlike any of the previous. Some solutions make it possible to inject *RESET TCP* packets to kill unwanted communications, making it particularly useful for environments with a NIDS with active response capabilities, or with a NIPS (Network Intrusion Prevention System) [NetworkCritical].

As for the tests performed at the LCT, a USB powered Ethernet (Aggregation) Switch Tap [Dualcomm] and a USB Ethernet adapter [Apple] were acquired, allowing the Shadow RTU to still have one Ethernet port with an IP address assigned to it (used for remote access), leaving the other one connected to the Ethernet Tap. Both Ethernet Tap and adaptor were connected to the available USB ports on the Raspberry Pi, and still operated correctly without disruption.

5.2 Base Scenario Setup

The connections between each device involved in the validation process of the Shadow RTU are following described (cf. Figure 5-2). This scenario only includes the necessary components for the Shadow RTU to carry its monitoring functionalities, focusing solely on the interconnection process between these.

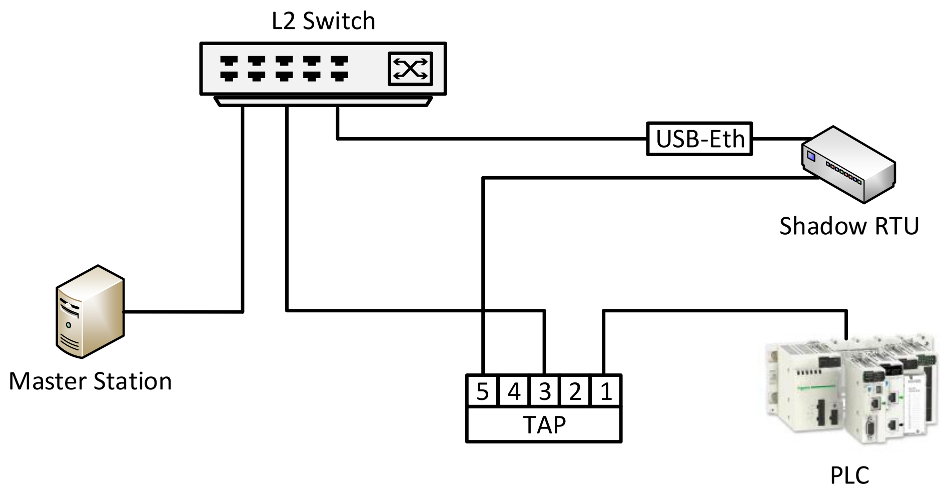


Figure 5-2 – Base Scenario for the Validation Process

The Ethernet Tap is composed of five ports, some of those with specific purposes: Ports 1 and 5 are used for port mirroring (outgoing and incoming network packets on port 1 are forwarded to port 5); and Ports 1 and 2 as pairs for Power over Ethernet (PoE) inline power pass through. The remaining ports (3 and 4) are used for regular switching. That said, and since the idea is to monitor the network packets flowing in and out of the PLC [M340], port 1 is connected to it and, port 5 to the monitoring Ethernet port on the Shadow RTU (on-board adaptor), working in promiscuous mode. The second port on the Shadow RTU (USB-Ethernet adaptor) is connected to a Layer-2 Switch [SRW2008] for remote access. To enable connectivity from the Master Station (or any other device) to the PLC, the third port on the Ethernet Tap is used to connect to the Switch.

5.3 Scenario No. 1: Workload Tests

By the time the workload tests were initiated, Raspberry Pi's were manufactured mainly in China and, only after (and solely) in the United Kingdom (UK), at Sony Corporation. The decision to move the production implied the modification and improvement of some of the components used to build the device, namely the Ethernet interface. The initial tests conducted with the models manufactured in

China led to the conclusion that the installed Ethernet interfaces ware causing oscillations and packet losses [RaspberryPiIssueNo29], something that with the ones made in the UK didn't happen. However, the reason the workload tests were carried out had to do with the fact that Ethernet port on the Raspberry Pi is actually a built-in USB Ethernet, as mentioned in previous chapters (cf. section 5.1).

To perform the test, a laptop computer (client machine) took the place of the Master Station (cf. Figure 5-2) to generate a predefined set of UDP (User Datagram Protocol) data streams, so as to subject the monitoring network interface of the Shadow RTU with different congestion levels and so, evaluate the CPU behavior. The idea is to determine which are the acceptable congestion levels the device might undergo without crashing.

5.3.1 Workload Stack

Following are described the set of congestions levels that were carried against the Shadow RTU (cf. Table 5-3), using a network throughput test tool [Iperf]. The following tests were conducted for a growing amount of UDP bandwidth congestion levels and a fixed packet length of 1470 bytes[14]. Before setting the workload stack, previous testing was performed to verify at which levels the CPU, packet loss and delay parameters started varying the most. During the tests both Master Station and PLC were connected, simply exchanging read register messages at each second. The workload tests were actually conducted in a different TCP port from the one used by Modbus TCP.

[14] The default UDP datagram length used in Iperf [Iperf], which is about 6 time more the maximum size of the Modbus TCP Protocol Data Unit (PDU).

Table 5-3 – Workload Stack (UDP Data Streams)

Workload	Type	Description
10 Mbytes/sec	Minimum	Minimum workload. Very low CPU usage is expected. At this point, no problems should be reported.
25 Mbytes/sec	Minimum/Average	Minimum/Average workload. Medium CPU usage is expected. The Raspberry Pi should work properly.
50 and 60 Mbytes/sec	Average/High	Average/High workload. High CPU usage is expected. The Raspberry Pi is expected to present some lag but still working fine.
75, 85 and 95 Mbytes/sec	High	High workload. Very high CPU usage is expected. Some errors and/or packet losses are expected.
100 Mbytes/sec	Maximum	Maximum workload. At this point the Raspberry Pi is supposed to crash or become inoperable.

5.3.2 Experimental Results and Analysis

The results obtained from the workload tests were retrieved with a simple script that monitored the system tasks [Top], to get the average CPU usage of the network throughput test tool [Iperf]. The same tool provides the results for Jitter and Packet Loss (cf. Table 5-4). For each congestion level, ten tests were performed for a period of 120 seconds, the resulting values averaged and the respective standard deviations calculated.

Table 5-4 – Raspberry Pi congestion results[15]

UDP Bandwidth (Mbytes/sec)	Interval (sec)	Jitter (ms)	Lost Packets (%)	Avg. CPU Usage[16] (%)
10	120	0.139	0.15	7.05
25		0.037	0.09	14.9
50		0.117	0.09	36.7
65		0.161	1.4	60.2
75		0.215	20	81.7
85		15.987	64	63.1
95		20.244	97	42.1
100		6.954	98	39.6

Considering the results presented in Table 5-4, the biggest turning point occurred between the 75 and 85 Mbytes per second congestion levels. Even though at 65 Mbytes/sec the CPU usage was already more than 50 percent, only when the bandwidth raised to 75 Mbytes/sec, the number of lost packets began to increase. On the other hand, Jitter only started to increase at 85 Mbytes/sec. This can be justified by the huge leap in lost packets from the previous stage (15 percent) to the current (64 percent). Following is a chat to better illustrate the relation between the number of lost packets and the average CPU usage of the network throughput test tool (cf. Figure 5-3).

[15] Jitter results were rounded to two three decimal places. Lost Packets and CPU usage were rounded to two and one decimal places.
[16] These values are referent to the iperf [Iperf] process.

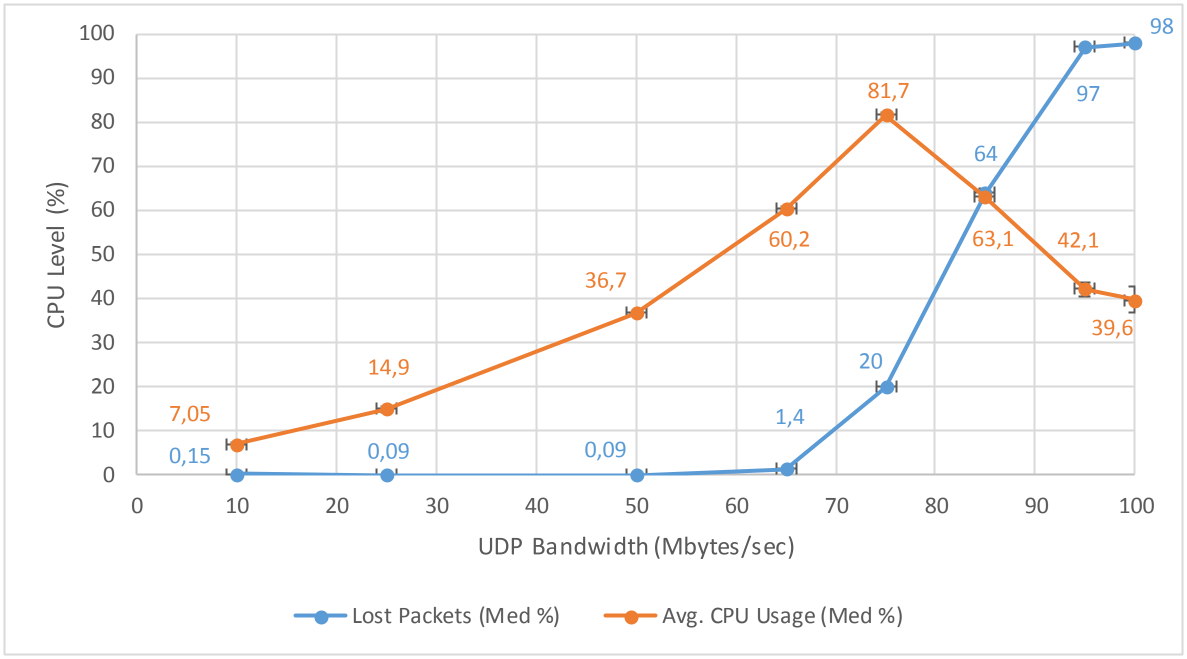


Figure 5-3 – CPU Usage: Avg. CPU usage of the network throughput test tool (y-axis) x UDP bandwidth (x-axis)

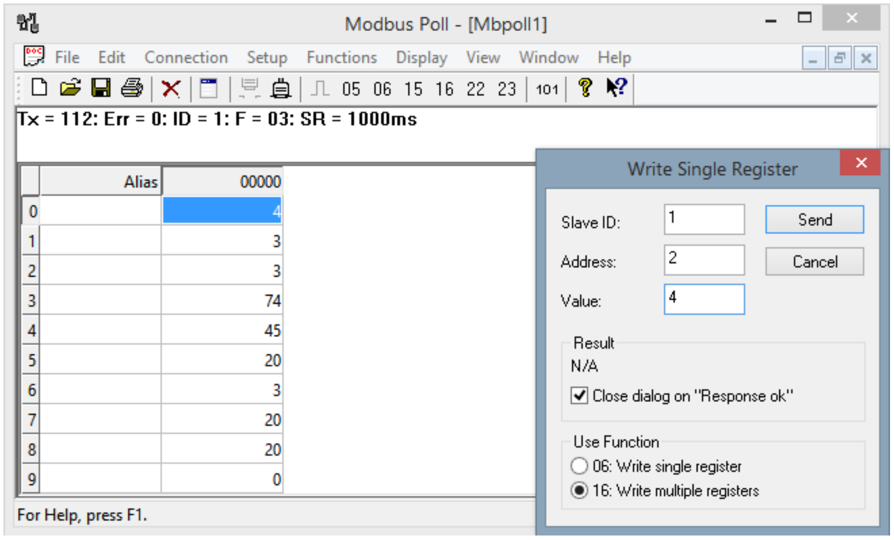
While performing the test with a bandwidth of 85 Mbytes/sec, the CPU levels started to drop, as illustrated in Figure 5-3. However, when analyzing the system processes in real-time [Top], a specific one named *ksoftirqd/0* [ksoftirqd] started consuming more than a half of CPU percentage than it did for every other previous test (about 25%, when previously it only used about 15% (75 and 65 Mbytes/sec) and, 10% (50 Mbytes/sec) of CPU resources). Actually, this process is used to handle interrupt requests (IRQs) that the system queues for later processing. If this process takes more than a tiny percentage of CPU time, means that the device is under heavy interrupt load, causing the CPU levels of the most demanding process (i.e., the network troubleshooter process) to drop – this means that, even though the CPU levels of the network application process lowered, the global system CPU levels continued rising. However, since Modbus TCP traffic is not too heavy, if the Shadow RTU ever finds itself on a similar situation an event should be immediately reported.

5.4 Software Validation

The most basic set of features to be supported by the Shadow RTU, such as the deployment of a Modbus TCP library and a WS application for remote management are described in the next chapters, along with a set of complementary functionalities for the Agent itself (e.g., safe code execution) and other components to support the final validation process.

5.4.1 Modbus Simulation Tool

The following tool was used to simulate the operation of the Modbus TCP protocol [ModbusTools] and, verify that the Shadow RTU was properly monitoring the requested operations carried by the Mbus Pool client application (e.g., change of a specific register) against the PLC (cf. Figure 5-4). The tool also includes a server side application (Mbus Slave) but, since the LCT had already acquired a real Modbus PLC there was no point in using a simulator for it.



Tx = 112: Err = 0: ID = 1: F = 03: SR = 1000ms

	Alias	00000
0		4
1		3
2		3
3		74
4		45
5		20
6		3
7		20
8		20
9		0

Slave ID: 1
Address: 2
Value: 4

Result
N/A
✓ Close dialog on "Response ok"

Use Function
○ 06: Write single register
● 16: Write multiple registers

Figure 5-4 – Modbus Poll (back) and Write Single Register (front) GUI [ModbusTools]

For the write register operation, the only parameters to specify are the Slave ID, Register Address, Value and Function Code (to write on a single or multiple registers), as illustrated in Figure 5-4. Once the operation is submitted, a packet monitoring tool may be used on the Shadow RTU to see the corresponding messages (cf. Annex D) [Tcpdump].

5.4.2 Modbus TCP and IEC 60870-5-104 Libraries

When performing the monitoring operation, the Shadow RTU may also implement a protocol decoder over the exchanged messages to find out what are the operations being requested by the Master Station and, to verify if the returned responses match (i.e., have not been modified in transit). For such, a complete implementation of the Modbus standard specification was adopted, with both client and server modes [Pymodbus]: as for the client features, it allows read/write operations on coils and registers; most of the extended protocol (diagnostic/file/pipe/setting/information); payload builder/decoder utilities and, synchronous and asynchronous operation modes; the server mode, provides a full control context (e.g., device information and counters) and, a few number of backing contexts (e.g., database, redis [Redis] or a slave device). Other two implementations of the protocol exist but were immediately discarded, since one of these only supports Modbus RTU [MinimalModbus] and, the other presented higher CPU loads [Modbus-tk] while operating in client/server mode, when compared to Pymodbus. Also, none of those implementations provide a complete decoding library necessary to perform a deep analysis over the Modbus TCP packets.

Relatively to the IEC104 standard, there is an open-source project written in C [OpenMRTS], that emerged as an attempt to create a complete solution of the *iecsock* library for Linux systems. To get it working, it is necessary to register both master and slave sessions, set specific session hooks on the events of interest and, receive the Application Service Data Units (ASDUs) (cf. Annex E) – while the link layer parameters are under the user control, the connection and session management is handled by the *iecsock* library. There is also a Perl implementation [Net::IEC104] which, like the previous, contains both client and server modes. However, it only supports ASDU NN 30, 35, 36 and 37 for direction control information and ASDU 100 and 103 for controlled direction of system information. None of these implementations are as complete as the ones mentioned for the Modbus TCP protocol and so, one should only expect a minimal set of functionalities from these.

5.5 Containers: Required Isolation Features

As mentioned in the previous chapters (cf. section 4.4), due to the complexity inherent to some applications these should be executed within a confined environment, to avoid compromising the host system. Only when the code is found to be safe, should it migrate to the host file system and remain there until new changes are made to the algorithm. In general, container applications have a somewhat large set of features from which only a few are required to be implemented on the Shadow RTU, such as the ones mentioned bellow:

- *File system isolation and quotas* – Unlike application-level approaches that apply isolation to the process itself, at the OS-level confinement is applied at the file system, meaning that malformed applications will not compromise the host file system. This could be achieved by the utilization of a virtual file system.
- *Network isolation* – The host system has to operate as an intermediary between the physical interface and the isolated environment, “replicating” the network data into the container. This means that the host system connections shouldn’t be used.
- *Root isolation* – This feature is supposed to allow root access between containers without affecting their operation.
- *CPU, memory and I/O allocations* – Resources should be allocated considering the application requirements to avoid overhead issues and allow multiple containers to be running simultaneously.

The information presented in [Xavier2013] (cf. section 2.9), shows that all containers are in some way more or less equivalent and so, the choice must be carefully balanced between overhead and functionality. That said, LXC were found to be the best solution since process isolation is allowed and, CPU and memory levels can still be kept low if only the required resources are put to use.

However, installation of LXC on the Shadow RTU was not particularly trivial, requiring a "LXC-friendly" kernel to be built specifically for the Raspberry Pi on another machine running Linux (a technique known as Cross-compiling [Lekhonkhobe2014]) with the necessary modules.

5.6 Remote Management

As mentioned in previous sections (cf. section 5.2) one of the network interfaces on the Shadow RTU was left with an IP address to allow the operator at the Control Room to manage its data model, an XML file with all the supported functionalities (cf. section 4.4). From the Control Room, the operator is provided with a graphical user interface (GUI) application where he types the XPath expression corresponding to the desired operation, i.e., the full path of the resource and the action to be applied to it, to communicate via Representational State Transfer (REST) messages using simple HTTP calls (e.g., HTTP POST method to create a new resource; HTTP PUT and DELETE to update and delete an existing resource, respectively; and HTTP GET to obtain the current value of a specific resource). In turn, the Shadow RTU has a WS application listening on the same port as the one used by the operator to perform the requests, to answer to HTTP requests and parse the XML file using a specific library. If the request happens to be successful, a second application responsible for managing the modules state on the Shadow RTU is invoked (e.g., modules to change the network interface state; to monitor and decode the Modbus TCP messages; to detect impersonation attacks towards the Master Station; and to report events to the Event Bus, just to mention a few). The following diagram explains the process described above (cf. Figure 5-5).

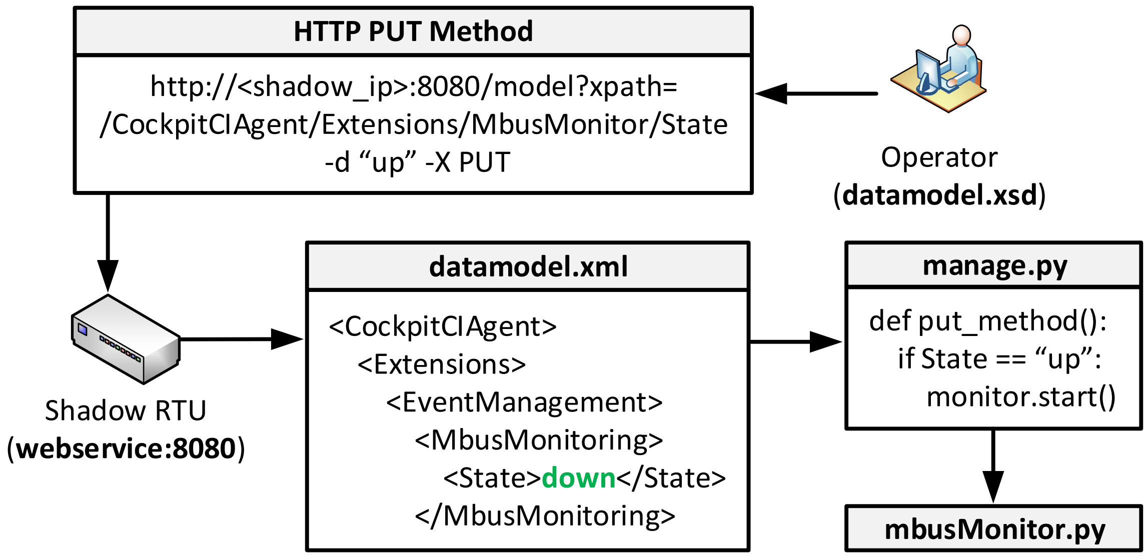


Figure 5-5 – Operator managing the Shadow RTU through its data model using Web Services

The example in Figure 5-5 illustrates the operator performing an HTTP Put request over a specified resource (*/CockpitCIAgent/Extensions/MbusMonitor/State*), the Modbus TCP monitoring application state. The WS application running in the Shadow RTU uses the specified XPath expression to modify the resource value in the data model and, execute the corresponding operations (manage.py and mbusMonitoring.py). The operator is provided with the data model schema (datamodel.xsd) of the Shadow RTU to known in advance what is the path of the resource to manipulate. Consistency between the XML and XSD files is also maintained to avoid having a scenario where the operato r tries to manipulate an inexistent resource.

The remote management approach, though the implementation of a WS on the Agent makes it very simple for the operator to control the services provided by it, without having to deal with the applications code. For every new service added to the Shadow RTU, a set of parameters are defined in the data model (cf. section 4.5.2), making service management a much easier and simpler task for the operator.

5.7 Scenario No. 2: Mirroring Mode

The mirroring mode is the most basic mode of operation of the Shadow RTU, used to dump the monitored network packets with an Ethernet Tap. In the following scenario the Shadow RTU is connected to an external machine (Message Checker), where the Modbus TCP messages are decoded [Pymodbus]; a client device simulating the Master Station; and a PLC to process the requests (cf. Figure 5-6). The Message Checker was introduced as an auxiliary mechanism to present the Shadow RTU running in its most basic mode of operation and, other complementary features to help in the detection of attacks (cf. section 5.8).

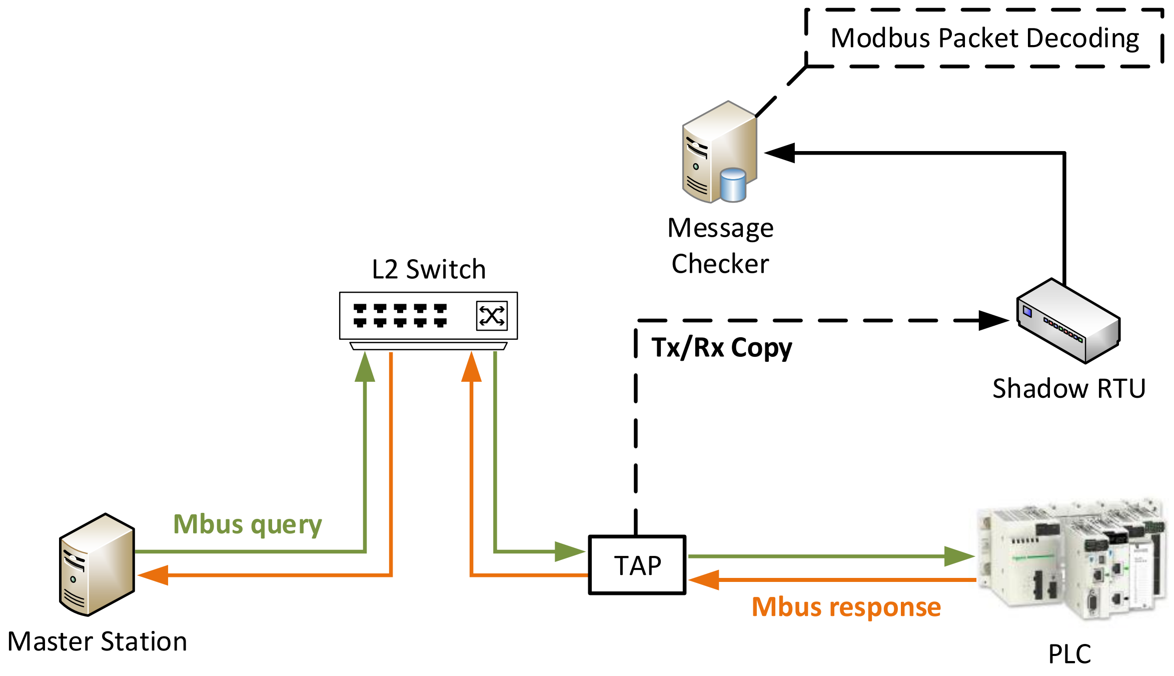


Figure 5-6 – Mirroring mode enabled

Once the client machine exchanges Modbus TCP messages with the PLC, these are captured by the Shadow RTU and saved into a file, using a specific tool [Tcpdump], which is then sent and analyzed at the Message Checker [Wireshark] (cf. Annex D). With this scenario it was proved that the Shadow RTU could efficiently capture every packet flowing between the two communicating ends, without losing messages or disruption. The network packets were only analyzed and decoded at the Message Checker (cf. Annex F), because a graphical tool was needed [Wireshark] for an easier visualization when presenting the Shadow RTU concept in Moons, Belgium (September 30, 2013).

5.8 Scenario No. 3: ARP Cache Spoofing

The following scenarios introduce a new agent, an Attacker, connected to the Local Area Network (LAN) to conduct an ARP cache spoofing attack, a technique whereby fake ARP reply messages are sent to the LAN. The messages sent to both communicating ends, have their respective destination IP associated to the Attacker's MAC address, causing all traffic to be sent to the Attacker instead (cf. Figure 5-7). Before the ARP reply messages reach their destinations, both ARP tables on the HMI and PLC match the respective destination IP to the correct MAC address. Only when the Attacker stops sending these fake messages, will both HMI and PLC receive legitimate replies from each other and, have the original communication path reestablished. This happens because ARP is a stateless protocol, meaning that received replies are automatically cached and/or overwritten, even if the ARP entries have not expired [RFC826]. Also, every device in the LAN receives the ARP replies even if not requested.

In Figure 5-7 the message flows are indicated by their corresponding number and color, to distinguish the ARP replies sent to the HMI and the PLC and, in Figure 5-8, to illustrate the path of both Mobdus query and response messages.

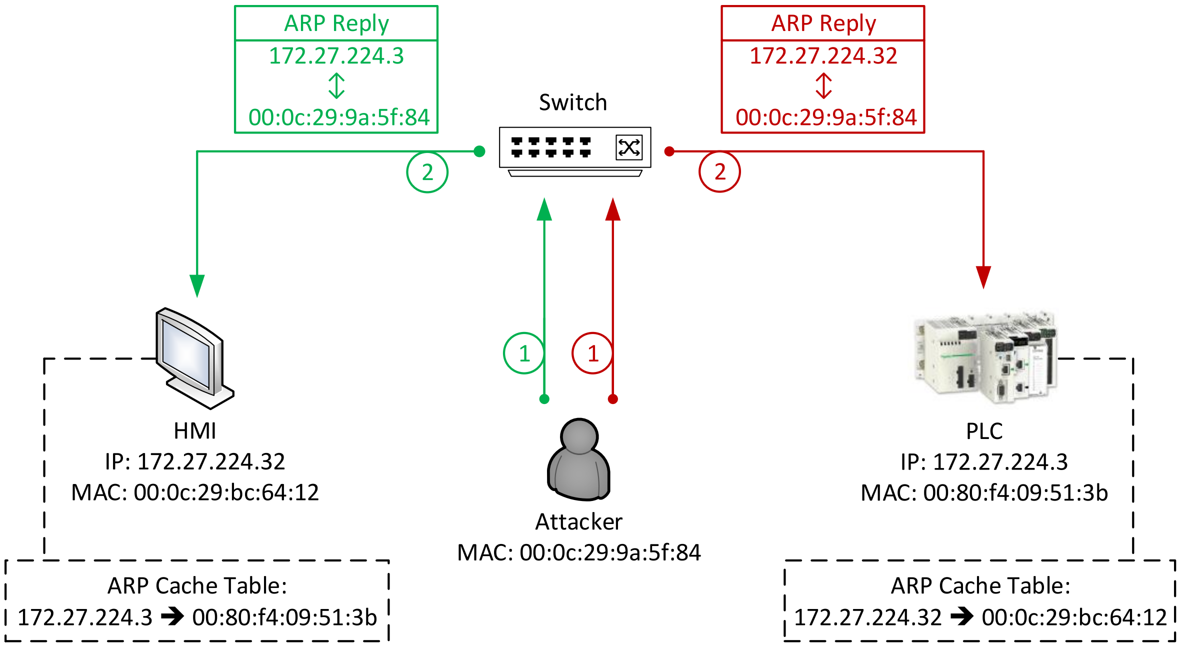


Figure 5-7 – ARP Cache Spoofing (Attacker sends gracious ARP messages)

To perform the ARP Cache Spoofing, the Attacker doesn't need an IP address since ARP reply messages only need to associate an IP to a destination MAC address. To conduct the attack a network security tool for MITM attacks [Ettercap] was used on the Attacker to scan the network and, select the target hosts (HMI and PLC) to send the malformed ARP replies. After that, the Attacker sniffs the remote connection and sees all traffic flowing between the HMI and PLC (e.g., using a network monitoring tool [Tcpdump]). Once the Attacker receives the exchanged Modbus messages, he forwards these to their respective ends.

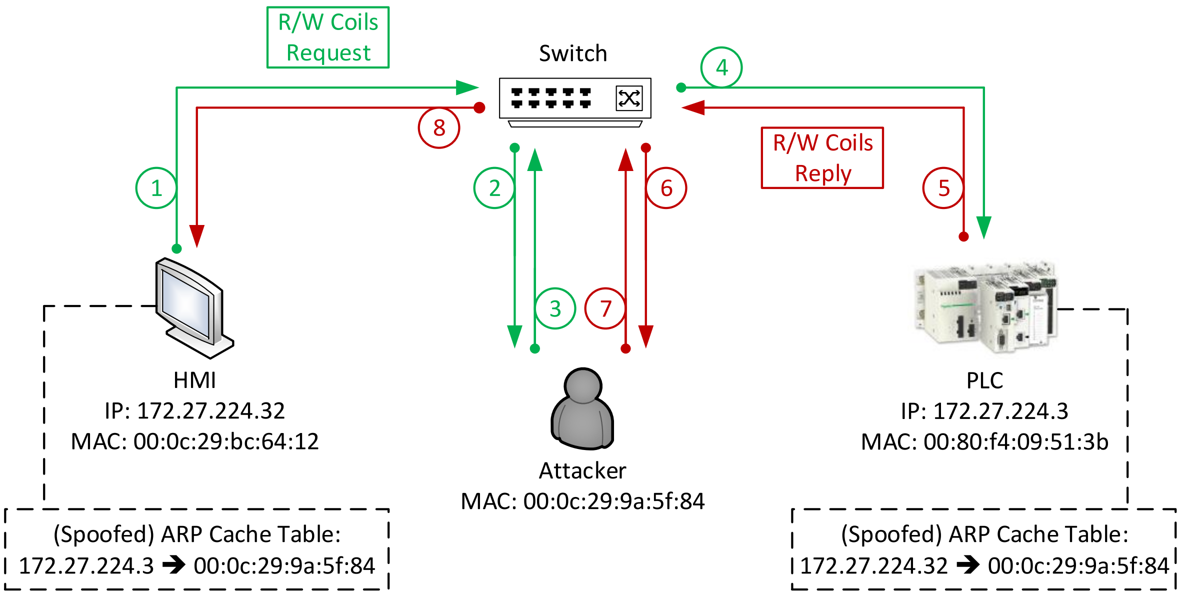
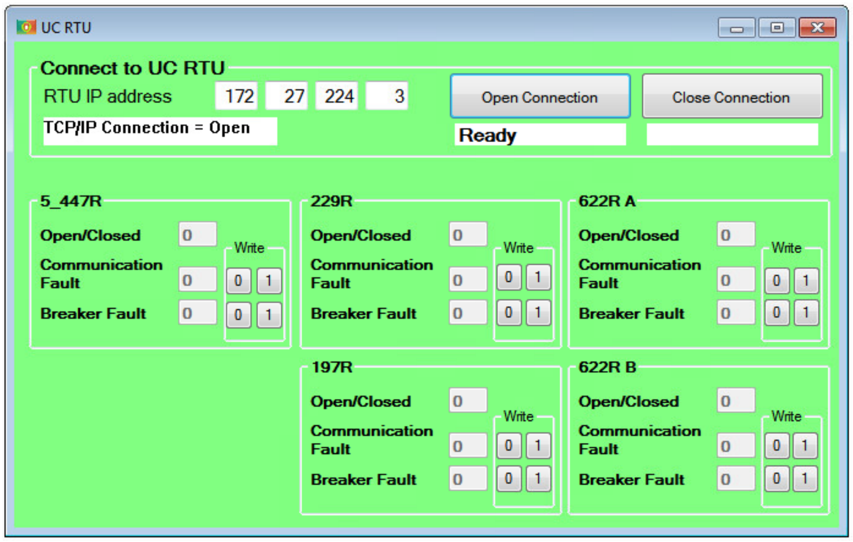


Figure 5-8 – ARP Cache Spoofing: MITM Attack (Attacker receives and forwards the messages)

Actually, the ARP cache spoofing attack if often used as an opening for other attacks. In the scenario illustrated above (cf. Figure 5-8), the ARP entries have been spoofed and the Modbus traffic flows between the HMI and the PLC unchanged, i.e., the Attacker only sees the traffic. At this point, the Attacker may choose to conduct a more robust MITM attack (e.g., receive the Modbus queries form the HMI, modify their respective parameters and only then, forward these to the PLC). A different type on MITM attack may also be performed, in which the Attacker forwards the traffic to a different device or application (cf. section 5.9).

5.9 Scenario No. 4: PLC Simulator

Once the Attacker has the Modbus messages being forwarded to him, the range of attacks he can now perform may vary depending on the level of damage he intends. For the following scenario, the Attacker cloned every register from the PLC to create an exact copy of it (i.e., a simulator), diverting the connection and, making the HMI believe it is communicating with the real one (cf. Figure 5-10). To complement the scenario and make sure that the HMI would no longer be communicating with the real PLC, a relay was connected to it. This relay would turn on and off when the operator changed a specific register value from 1 to 0 using a graphical interface developed by Roma Tre, one of the CockpitCI project partners (cf. Figure 5-9). The Modbus protocol implementation [Pymodbus] was also used to make it easier to change the relay state, by writing on the respective coil address.



Connect to UC RTU

RTU IP address: 172 . 27 . 224 . 3

TCP/IP Connection = Open

Ready

	5_447R	229R	622R A	197R	622R B
Open/Closed	0	0	0	0	0
Communication Fault	0	0	0	0	0
Breaker Fault	0	0	0	0	0

Figure 5-9 – HMI interface developed by Rome Tre

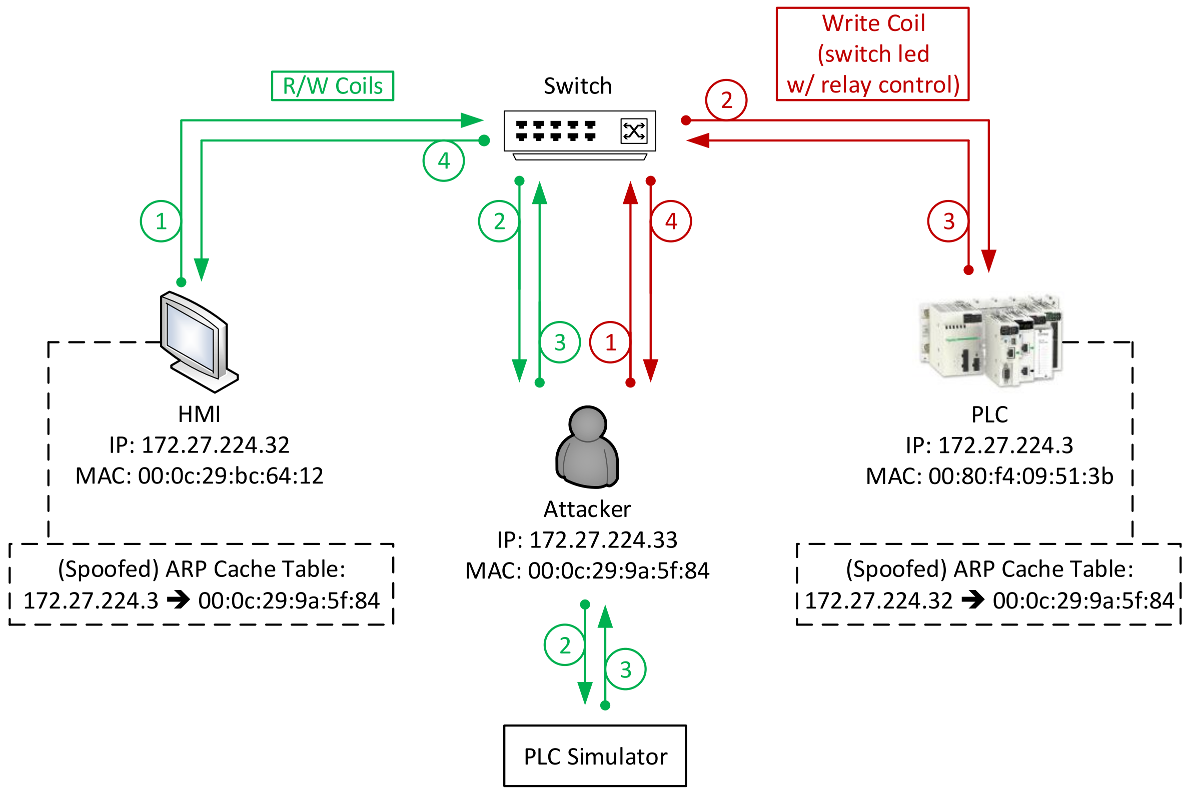


Figure 5-10 – ARP Cache Spoofing: MITM Attack (attacker runs a PLC simulator and controls the relay)

For the scenario illustrated in Figure 5-10, each color represents a separate process. With the HMI communicating with the fake PLC, the Attacker is now free to send Modbus requests to real PLC or, change the requests from the HMI in transit. In that sense, only the first situation was tested. Also, trying to manipulate the register that controls the relay state is no use because the connection is being diverted to the simulator, giving to the operator the idea that he is interacting with the real PLC.

5.10 Scenario No. 5: Message Inconsistency Check

The following scenario was designed to alert the operator for situations where it might be communicating with a fake PLC without noticing it (cf. section 5.9). In this case, for every Modbus query message sent by the HMI, a copy is also sent to the Message Checker. The same happens when the Attacker sends a query to the PLC, i.e., the Ethernet Tap sends a copy to the Shadow RTU, which it then forwards to the Message Checker (cf. Figure 5-11).

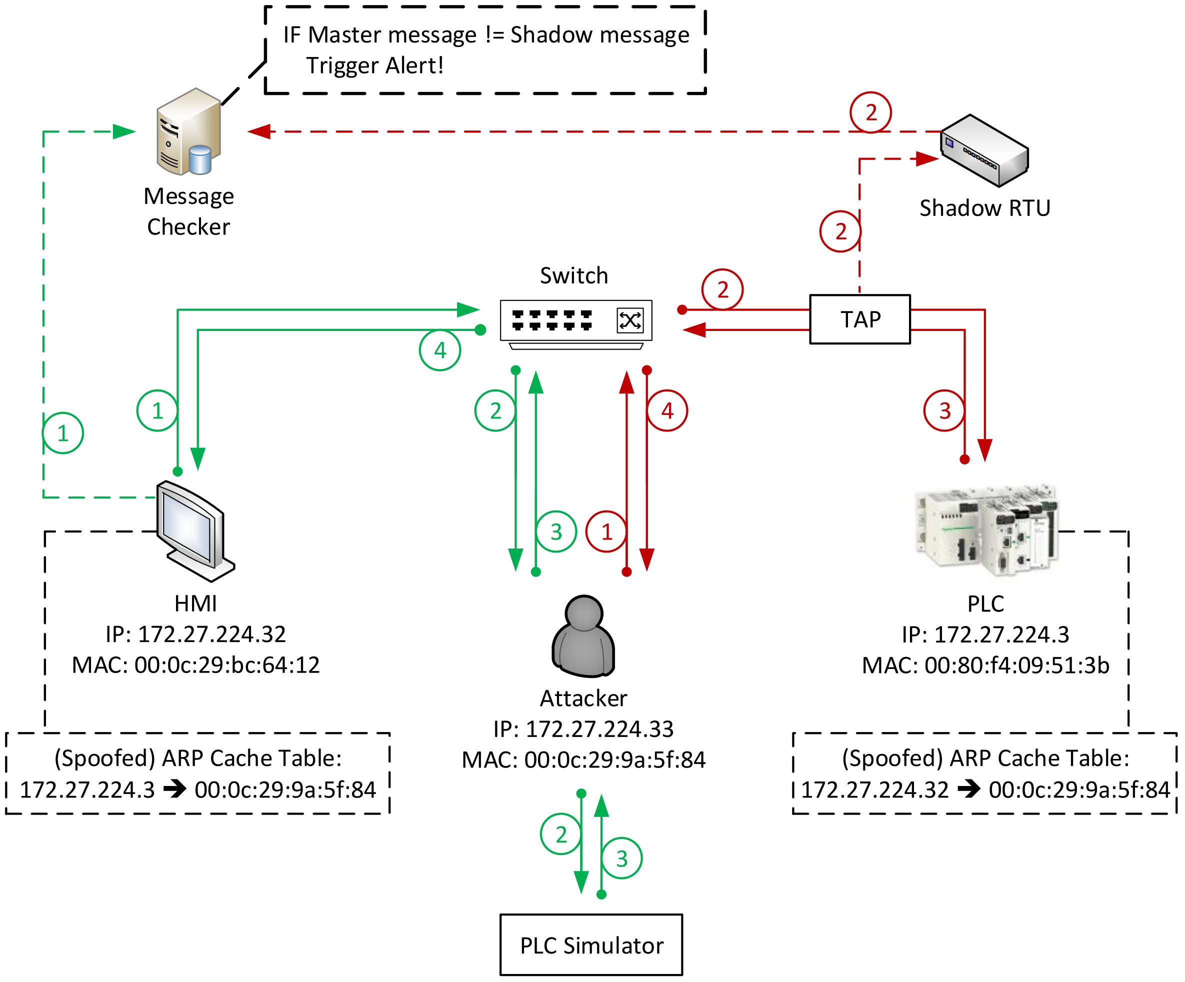


Figure 5-11 – MITM attack (HMI and Shadow RTU send a copy to the Message Checker)

Since both messages don’t match, an alert is triggered and sent directly to the Control Room where the operator is informed about the ongoing attack. The Message Checker, just like the Shadow RTU, could also be connected to the Event Bus to report such anomalies (cf. section 4.4). This scenario made possible to present a case where the Shadow RTU is actually useful and peculiar relatively to the remaining detection components. Another test was performed without the aid of the Message Checker, which consisted in detecting a Master impersonation attack (cf. section 3.5) – once the Shadow RTU detects that the PLC MAC address between two consecutive messages differs, the respective event information (IDMEF) is sent to the Local Correlator, through the Event Bus.

6 Work Plan

6.1 Introduction

The following chapters describe the followed work plan and its evolution throughout the current academic year, in both first and second semester. Some of the constraints found along the path are also mentioned.

6.2 1st Semester Schedule

By the time the thesis had officially started, there was already some work done in the context of the State of the Art and validation process of the Shadow RTU. Also, both CockpitCI project and SCADA systems in general were known. That said, some modification to the scheduled work plan had to be performed to reflect this. Nonetheless, some of the work that was already done had to be performed again to rectify some inconsistencies that were found at the time or, simply to improve the documentation and adapt it to new developments. The performed tasks are described below with a corresponding identifier (ID) to match the diagram presented at the end (cf. Table 6-1):

- ID 1 – Shadow RTU concept validation.
  - Preparation of a document comparing single-board devices to take the role of the Shadow RTU. At this point, workload tests were carried with the Raspberry Pi to analyze the behavior of the device under stress conditions.
  - Preparation of a document with a complete analysis of the Modbus TCP and IEC104 protocol standards. Many implementations of both protocols were also tested to find out which ones possessed the necessary features to perform packet decoding.
  - Study and evaluation of different network monitoring approaches to be used by the Shadow RTU. At this stage, occurred a validation process using a passive Ethernet tap to monitor and capture network packets.
- ID 2 – Preparation of the Shadow RTU demonstration.
  - Preparation of a document to discuss the purpose of the Shadow RTU as a Field device running inside the Field Network. A series of attacks detected by the Shadow RTU were also described, to present where it is really effective.
  - Definition of the first use case: Shadow RTU operating in mirroring mode, using a passive Ethernet tap to capture the exchanged Modbus TCP messages between a Master Station and a PLC.
  - Definition a second use case: Detection of a MITM Attack using a second machine to compare both messages sent by the Master Station and Shadow RTU.
- ID 3 – Management Interface Functionalities: Preparation of a document to define the most basic functionalities (API) to be supported by the Shadow RTU. This document underwent

many revisions until the final version.

- ID 4 – Definition of the Probing architecture modules.
    - Preparation of a document to define which should be the modules to include in the Shadow RTU. This stage was particularly important and time consuming, since it defines the key aspects that make the Shadow RTU effective.
    - Evaluation of Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) implementations to be used for the communicating with the Even Bus (i.e., Local Correlators). This task was preceded by a short period to study a new programming language (Python), applied in this task and all the following ones.
    - Study on the many available approaches to achieve confinement for applications. This stage was time consuming due to the range of available solutions and the constraints to implement on the Raspberry Pi.
- ID 5 – Writing of the intermediate version of the thesis report.

The biggest constraint during this semester was related to the implementation of containers. Most of these were developed for the x86 architecture and, the one that was actually compatible with ARM processors (and suited for the board's hardware), was in its early stage. Trying to run it on the Raspberry Pi required some time, as also mentioned in previous chapters (cf. section 5.5).

The following table presents the tasks mentioned above, with the respective time each one took to be completed (cf. Table 6-1).

Table 6-1 – 1st semester scheduling

			2013				2014
ID	Task	Weeks	Sep.	Oct.	Nov.	Dec.	Jan.
1	Concept Validation	2	16-30				
2	Demo	5	23-30	01-28			
3	API	4		31	01-24		
4	Probing Arch.	5			25-30	01-30	
5	Report	½					13-28

6.3 2ND SEMESTER SCHEDULE

During the second semester, the focus was on the integration and the developed of the probing modules of the proposed architecture, integrated in the CockpitCI architecture. Following is described the schedule for the 2nd semester:

- ID 6 – Preparation of the CIGRE demonstration.

    - Development of an Attacker machine to conduct an ARP cache spoofing attack targeted to the HMI and PLC.
    - Implementation of a PLC simulator to be used by the Attacker to conduct a MITM attack.
- ID 7 – Development of the Web Service to be used by the Shadow RTU and remaining detection agents (e.g., honeypot). This was the most time consuming task to the semester, since it involved not only the Web Service development, but also all the applications to be used by the Agent.
    - Definition of the data model (XML and XSD)
    - Implementation of the base Web Service application to manipulate the data model
- ID 8 – Implementation of the most basic functionalities (API)
    - Modbus TCP monitoring, Modbus TCP parser/decoder and, ARP cache spoofing detection. Other basic functionalities such as, changing the IP address and start/stop the application execution, were implemented.
    - Generation of a simple IDMEF message for an HMI impersonation attack. Once the Shadow RTU detects the attack, the message is generated and sent to the Even Bus to be processed.
- ID 9 – Integration of the Shadow RTU and respective probing architecture in the scenario setup at the LCT.
- ID 10 – Writing of the final version of the thesis report

As for the second semester the biggest constrains were found in the development of the Attacker for the CIGRE demonstration. The initial idea was to change the Modbus TCP packets in transit, once the ARP cache spoofing had been performed. However, taking into account the difficulties encountered, the attack was instead conducted using a PLC simulator to perform the MITM attack.

The following table presents the tasks performed in second semester (cf. Table 6-2):

Table 6-2 – 2nd semester scheduling

			2014				
ID	Task	Weeks	Feb.	Mar.	Apr.	May	June
6	CIGRE Demo	6	01-28	01-12			
7	Web Service	4		15-31	01-13		
8	Functionalities (API)	6			14-30	01-31	
9	Integration[17]	8		15-31	14-30	01-31	
10	Final Report	4					01-30

[17] The integration process occurred from the beginning of the Web Service development until the implementation of the Shadow RTU functionalities.

7 Conclusion

7.1 Developed Work Overview

The present document covered the proposed probing architecture relative to the Shadow RTU, one of the detection agents operating in the Field Network of the CockpitCI detection architecture. Since the main purpose of this component is to raise security awareness over the monitored devices, the presented State-of-the-Art focused on vulnerabilities targeted to SCADA and on an overview of the industry after the impact caused by of one of the most well-known rootkits so far. Also in this section, the problem of architecture partitioning was addressed to sustain the decisions made by the CockpitCI team and its three layer architecture. Finally, a series of low-level threats and preventive measures implemented in IT systems were presented. The idea with these last subjects is to provide an overview of what might be the future security concerns of SCADA operators, and how making use of containers could raise the safety levels in these networks.

Relatively to the proposed probing architecture, a series of modules were developed, namely, for monitoring the PLC activity, process the captured network information and report anomalies. After the implementation of some use cases to validate the concept, an integration phase was followed to allow the Security Management Platform to remotely manage the agent and, to make it report security events to the Local Correlators, through the Event Bus.

7.2 Contributions

During the involvement of the student in the CockpitCI project, some contributions were made: the information contained in Annexes A, B and C relative to the Modbus TCP, IEC 60870-5-104 standard and, a comparison between IEC 870-5-101, Modbus and DNP3 were included in deliverable D3.1; some of the the information contained in chapter 5, relative to the Ethernet tap, monitoring options and single-boards comparison were included in D3.3; a brief chapter on the Shadow RTU concept was added in D3.4 (cf. Annex G); and both chapters 4 and 5 were included in D3.5.

7.3 Future Work

The work presented in this thesis will be continued, namely, as for the API functionalities and integration with the architecture. Even though the most basic features of the Shadow RTU have already been implemented, not all of them have. The idea is provide a larger set of available functionalities and implement a more robust use case for reporting security events. So far, the Shadow RTU only reports security events for an HMI impersonation attack, which is a simple case. More sophisticated cases will be implemented in the next months.

8 REFERENCES

[SCADA2004] National Communications System, "*Supervisory Control and Data Acquisition (SCADA) Systems*", Chantilly, Virginia, October 2004

[RTAP] Industrial Defender, "*Real-time Object Database / Process / Event / Alarm Management / History*". Available at: http://www.rtapscada.com/rtap/real_time.php

[IEEE802.3] IEEE 802.3 Ethernet Working Group. Available at: http://www.ieee802.org/3/ (Last Access: June 11, 2014)

[ISA2011] Tai-Hoon Kim, Hojjat Adeli, Rosslin John Robles, Maricel Balitanas, "*Information Security and Assurance: International Conference, ISA 2011*", Springer, Brno, Czech Republic, 2011

[Choraś2010] Choraś, M.; Flizikowski, A.; Kozik, R.; Hołubowicz, W., "*Critical Information Infrastructures Security: 4th International Workshop, CRITIS 2009 – Decision Aid Tool and Ontology-Based Reasoning for Critical Infrastructure Vulnerabilities and Threats Analysis*", Springer, 2010

[ICSA-14-084-01] ICS-CERT, Advisory (ICSA-14-084-01), Festo CECX-X-(C1/M1) Controller Vulnerabilities. Available at: http://ics-cert.us-cert.gov/advisories/ICSA-14-084-01 (Original Release Date: April 24, 2014)

[Edvard2013] Edvard, "*Three generations of SCADA system architectures*" – Third generation SCADA System. Available at: http://electrical-engineering-portal.com/three-generations-of-scada-system-architectures (Published: April 22, 2013)

[IoT-GSI] International Telecommunication Union, "*Internet of Things Global Standards Initiative*". Available at: http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx

[Harbor2012] Harbor Research, "*Smarter SCADA & The Internet of Things*", Smarter SCADA Workshop, October 2012

[Combs2011] Larry Combs, InduSoft, "*Cloud Computing for SCADA*", Control Engineering (Vol. 58 Issue 12, p22), December 2011

[Synergist2012] Austin Scott, Synergist SCADA Inc, "*The Top 5 SCADA Security Threats for 2012*". Available at: http://www.synergistscada.com/the-top-5-scada-security-threats-for-2012/ (Published: March 3, 2012)

[Epiphan] Epiphan Systems Inc., "*External SCADA Monitoring*". Available at: http://www.epiphan.com/solutions_new/?arid=84

[Falliere2014] Nicolas Falliere, Symantec, "*Stuxnet Introduces the First Known Rootkit for Industrial Control Systems*". Available at: http://www.symantec.com/connect/blogs/stuxnet-introduces-first-known-rootkit-scada-devices (Updated: January 23, 2014)

[Clarke2004] Clarke, G.; Reynders, D.; Wright, E., "*Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems*", IDC Technologies, Great Britain, 2004

[Yokogawa] Yokogawa Electric Corporation, "*SCADA Security*", Yokogawa System Center Europe B.V. Available at: https://www.yokogawa.com/scd/pdf/SCADAsecurity.pdf

[Byres2012] Byres, E.; Cusimano, J., Tofino Security, "*7 Steps to ICS and SCADA Security*", February 16, 2012

[ANSI-ISA-99] American National Standard, "*Security for Industrial Automation and Control Systems – Part 1: Terminology, Concepts, and Models*", October 2007

[CPNI2005] Center for the Protection of National Infrastructure (CPNI), "*Firewall Deployment For SCADA and Process Control Networks – Good Pactice Guide*", February 15, 2005

[Poulsen2003] Kevin Poulsen, Security Focus "*Slammer worm crashed Ohio nuke plant network*". Available at: http://www.securityfocus.com/news/6767 (Updated: August 19, 2003)

[Symantec] Vulnerability Trends, SCADA Vulnerabilities, Symantec. Available at: http://www.symantec.com/threatreport/topic.jsp?aid=scada_vulnerabilities&id=vulnerability_ trends

[Shekaraubi2014] Shekaraubi, S., International Policy Digest, "*Iran's Case against Stuxnet*". Available at: http://www.internationalpolicydigest.org/2014/03/18/irans-case-stuxnet/ (Published: Arch 18, 2014)

[Kelley2013] Kelley, M., Business Insider, "*The Stuxnet Attack On Iran's Nuclear Plant Was 'Far More Dangerous' Than Previously Thought*". Available at: http://www.businessinsider.com/stuxnet-was-far-more-dangerous-than-previous-thought-2013-11 (Published: November 20, 2013)

[SimaticS7-300] SIMATIC S7-300: the modular universal controller for the manufacturing industry, Siemens. Available at: http://w3.siemens.com/mcms/programmable-logic-controller/en/simatic-s7-controller/s7-300/pages/default.aspx

[Ptsecurity2012] Positive Technologies, "*SCADA Safety in Numbers*", 2012

[ICS-CERT] The Industrial Control Systems Cyber Emergency Response Team (ICS-CERT). Available at: https://ics-cert.us-cert.gov/

[CVE] Common Vulnerabilities and Exposures (CVE®). Available at: http://cve.mitre.org/about/index.html

[Securityfocus] Security Focus. Available at: http://www.securityfocus.com/

[OSVDB] Open Sourced Vulnerability Database. Available at: http://osvdb.org/

[CERT] ProductCERT Security Advisories, Siemens. Available at: http://www.siemens.com/innovation/en/technology-focus/siemens-cert/cert-security-advisories.htm

[CVE-2010-2772] CVE-ID: CVE-2010-2772. Available at: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2772

[Auriema] Luigi Auriemma (http://aluigi.altervista.org/)

[Peterson2011] Peterson, D., "*Interview with Luigi Auriemma of 34 0day ICS Vulnerabilities*". Available at: http://www.digitalbond.com/blog/2011/03/22/interview-with-luigi-auriemma-of-34-0days-ics-vulnerabilities/ (Published: March 22, 2011)

[Jackson2012] Kelly, J., Dark Reading, "*SCADA Security In A Post-Stuxnet World*", 2012. Available at: http://www.darkreading.com/vulnerabilities---threats/scada-security-in-a-post-stuxnet-world/d/d-id/1138634? (Published: November 06, 2012)

[owasp2009] Open Web Application Security Project, Buffer Overflow. Available at: https://www.owasp.org/index.php/Buffer_Overflow (Last Revision: February 20, 2009)

[ST04-015] Security Tip (ST04-015), Understanding Denial-of-Service Attacks. Available at: http://www.us-cert.gov/ncas/tips/ST04-015 (Last Access: February 06, 2013)

[Falliere2010] Nicolas, F., Symantec, "*Stuxnet Infection of Step 7 Projects*". Available at: http://www.symantec.com/connect/blogs/stuxnet-infection-step-7-projects (Updated: January 23, 2014)

[Rössel2011] Rössel, T., "*Post-Stuxnet Industrial Security: Zero-Day Discovery and Risk Containment of Industrial Malware*", 2011

[Kassner2008] Kassner, M., Tech Republic, "*10+ things you should know about rootkits*". Available at: http://www.techrepublic.com/blog/10-things/10-plus-things-you-should-know-about-rootkits/ (Published: September 17, 2008)

[Wilson2013] Wilson, T., Dark Reading, "*Attacks On Volatile Memory Can Be Detected, Researchers Say*". Available at: http://www.darkreading.com/attacks-breaches/attacks-on-volatile-memory-can-be-detect/240162195 (Published: October 03, 2013)

[Causey2013] Causey, B., Dark Reading, "*Below The Application: The High Risk Of Low-Level Threats*". Available at: http://www.darkreading.com/vulnerability/below-the-application-the-high-risk-of-l/240157884 (Published: July 09, 2013)

[Iozzo2009] Iozzo. V, "*Let your Mach-O fly*", January 2009

[Cui2013] Cui, A.; Costello, M.; Stolfo, S., "*When Firmware Modifications Attack: A Case Study of Embedded Exploitation*", Columbia University, June 2013

[Mello2013] Mello, J., CSO, "*Spear phishing poses threat to industrial control systems*". Available at: http://www.csoonline.com/article/2134000/access-control/spear-phishing-poses-threat-to-industrial-control-systems.html (Published: September 26, 2013)

[ISA-99.00.01-2007] American National Standard, ANSI/ISA-99.00.01-2007 - Security for Industrial Automation and Control Systems - Part 1: Terminology, Concepts, and Models, 29 Oct. 2007

[Jooste2008] Jooste, S., "*PROTECTED EXECUTION ENVIRONMENTS WITHIN A COMPUTER SYSTEM*", Patent No.: US 7,444,671 B2, October 2008.

[Sun2005] Sun, W.; Liang, Z.; Sekar, R.; Venkatakrishnan, V., "*One-way isolation: An effective approach for realizing safe execution environments*", 2005

[Chroot] chroot(1) – Linux manual page, "*Run command or interactive shell with special root directory*". Available at: http://linux.die.net/man/1/chroot

[Systrace] Systrace(4) – BSD manual page, "*Enforce and generate policies for system calls*" Available at: http://www.manualpages.de/OpenBSD/OpenBSD-5.0/man4/systrace.4.html (Updated: August 12, 2012)

[Ptrace] Ptrace(2) – Linux manual page, "*Process trace*". Available at: http://linux.die.net/man/2/ptrace

[Namespaces] Kerrisk, M., "*Namespaces in operation, part 1: namespaces overview*". Available at: http://lwn.net/Articles/531114/ (Published: January 4, 2013)

[Prctl] Prctl(2) – Linux manual page, "*Operations on a process*". Available at: http://linux.die.net/man/2/prctl

[Seccomp] Corbet, J., "*Seccomp and sandboxing*". Available at: https://lwn.net/Articles/332974/ (Published: May 13, 2009)

[Unshare] Unshare(2) – Linux manual page, "*Disassociate parts of the process execution context Synopsis*". Available at: http://linux.die.net/man/2/unshare

[Clone] Clone(2) – Linux manual page, "*Create a child process*". Available at: http://linux.die.net/man/2/clone

[FUSE] Filesystem in Userspace. Available at: http://fuse.sourceforge.net/ (Updated: January 01, 2014)

[Dmesg] Dmesg(8) – Linux manual page, “*Print or control the kernel ring buffer*”. Available at: http://linux.die.net/man/8/dmesg

[Proc] The Linux Kernel Module Programming Guide, “*Chapter 5. The /proc File System*”. Available at: http://linux.die.net/lkmpg/x710.html

[Ramesh] Ramesh, “*Hack 75. Safe Reboot Of Linux Using Magic SysRq Key*”. Available at: http://linux.101hacks.com/sysadmin-tasks/magic-sysrq-key/

[VServer] Linux-VServer. Available at: http://linux-vserver.org/Welcome_to_Linux-VServer.org

[OpenVZ] OpenVZ. Available at: http://openvz.org/Main_Page

[Rlimit] getrlimit(2) – Linux manual page, “*Get/set resource limits*”. Available at: http://linux.die.net/man/2/getrlimit

[Ramankutty2004] Ramankutty, H., “*Inter-Process Communication - Part 1*”. Available at: http://linuxgazette.net/104/ramankutty.html (Published: July, 2014)

[SELinux] SELinux Project. Available at: http://selinuxproject.org/page/Main_Page

[Targeted] https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security-Enhanced_Linux/chap-Security-Enhanced_Linux-Targeted_Policy.html

[Xavier2013] Xavier, M.; Neves, M.; Rossi, F.; Ferreto, T.; Lange, T., Rose, C., “*Performance Evaluation of Container-based Virtualization for High Performance Computing Environments*”, Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil, 2013

[R610] Dell PowerEdge R610. Available at: http://www.dell.com/downloads/global/products/pedge/en/server-poweredge-r610-specs-en.pdf

[GPL] GNU General Public License. Available at: http://www.gnu.org/licenses/gpl.html

[Jails] Riondato, M., Chapter 15. Jails, Part III. System Administration – Jails. Available at: http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html

[lxc2011] Evading from linux containers. Available at: http://blog.bofh.it/debian/id_413 (Published: July 30, 2011)

[D3.1_2013] WP3000 - Cyber analysis and detection, “*CockpitCI Deliverable D3.1.2: Requirements and Reference Architecture of the Analysis and Detection Layer_V1.0*”, Department of Informatics Engineering, University of Coimbra, Portugal, 2013.

[D3.1_2013a] WP3000 - Cyber analysis and detection, “*CockpitCI Deliverable D3.1.2: Requirements and Reference Architecture of the Analysis and Detection Layer_V1.0 – Figure 5-4 Generic probing architecture*”, Department of Informatics Engineering, University of Coimbra, Portugal, 2013.

[D3.1_2013b] WP3000 - Cyber analysis and detection, “*CockpitCI Deliverable D3.1.2: Requirements and Reference Architecture of the Analysis and Detection Layer_V1.0 – Figure 5-11 Proposed CockpitCI detection architecture*”, Department of Informatics Engineering, University of Coimbra, Portugal, 2013.

[D3.1_2013c] WP3000 - Cyber analysis and detection, “*CockpitCI Deliverable: D3.1.2: Requirements and Reference Architecture of the Analysis and Detection Layer_V1.0 – Table 5-2*

Security ontology for the components of the CockpitCI cyber-analysis and detection reference architecture", Department of Informatics Engineering, University of Coimbra, Portugal, 2013.

[Probes2000] Teo, L., Linux Journal, "*Network Probes Explained: Understanding Port Scans and Ping Sweeps*". Available at: http://www.linuxjournal.com/article/4234 (Published: December 01, 2000)

[Higgins2013] Higgins, K., Dark Reading, "*Experiment Simulated Attacks On Natural Gas Plant*". Available at: http://www.darkreading.com/perimeter/experiment-simulated-attacks-on-natural/240157897?itc=edit_in_body_cross (Published: July 08, 2013)

[Shaw2004] Shaw, W., Cyber SECurity Consulting , "*SCADA System Vulnerabilities to Cyber Attack*", Article published part of the September/October 2004 Issue. Available at: http://www.electricenergyonline.com/show_article.php?mag=&article=181 (Published: September, 2004)

[Wilhoit2013] Wilhoit, K., "*The SCADA That Didn't Cry Wolf – Who's Really Attacking Your ICS Equipment? (Part 2)*", August 04, 2013

[Einwechter2002] Einwechter, N., Symantec, "*Implementing Networks Taps with Network Intrusion Detection Systems*". Available at: http://www.symantec.com/connect/articles/implementing-networks-taps-network-intrusion-detection-systems (Updated: November 02, 2002)

[pcap_set_buffer_size] pcap_set_buffer_size(3pcap), Linux manual page, "*pcap_set_buffer_size – set the buffer size for a not-yet-activated capture handle*". Available at: http://www.tcpdump.org/manpages/pcap_set_buffer_size.3pcap.txt (Updated: 5 April 2008).

[pcap] pcap(3), Linux manual page, "*pcap – Packet Capture library*". Available at: http://www.tcpdump.org/manpages/pcap.3pcap.html (Updated: 16 April 2014)

[pcap-filter] pcap-filter(7), Linux manual page, "*pcap-filter – packet filter syntax*". Available at: http://www.tcpdump.org/manpages/pcap-filter.7.html (Updated: 6 January 2008)

[rsync] rsync(1), Linux manual page, "*rsync – a fast, versatile, remote (and local) file-copying tool*". Available at: http://rsync.samba.org/ftp/rsync/rsync.html (Updated: 26 January 2014)

[pymodbus] Pymodbus – A fully featured Modbus protocol stack in python. Available at: https://github.com/bashwork/pymodbus (Updated: 04 April 2013)

[lkm] Linux Loadable Kernel Module HOWTO: Introduction to Linux Loadable Kernel Modules. Available at: http://www.tldp.org/HOWTO/Module-HOWTO/x73.html (Updated: 24 September 2006)

[lxc] LXC – Linux Containers: Userspace tools for the Linux kernel containers. Available at: https://linuxcontainers.org/ (Updated: 6 March 2014)

[D3.4_2013] FP7-SEC-2011-1 Project 285647, Cyber-security on SCADA: risk prediction, analysis and reaction tools for Critical Infrastructures, "*D3.4 - Design of the Dynamic Perimeter Intrusion Detection System – Preliminary*" (Updated: 30 June 2013)

[RFC4765] RFC4765, The Intrusion Detection Message Exchange Format (IDMEF). Available at: https://www.ietf.org/rfc/rfc4765.txt (Updated: March 2007)

[Ntpdate2005] Ntpdate. Available at: http://www.hants.lug.org.uk/wiki/Ntpdate (Updated: 29 June 2005)

[Arduino] Arduino Uno Revision 3. Available at: http://arduino.cc/en/Main/arduinoBoardUno

[329676-002US] Intel® Quark™ SoC X1000 Datasheet – Revision 02, May, 2014

[Coley2014] Coley, G., BeagleBone Black System Reference Manual – Revision C.1, May 22, 2014

[RaspberryPi] Raspberry Pi Model B Revision 2, Upcoming board revision. Available at: http://elinux.org/RPi_Documentation

[Orsini2014] Orsini, L., ReadWrite, "*Easy Arduino: Two Projects To Help You Get Started*". Available at: http://readwrite.com/2014/04/21/easy-arduino-projects-basics-tutorials-diy-hardware#awesm=~oHQXs831wCzZFw (Published: April 21, 2014)

[Hamilton2007] Hamilton, T., "*UTP Cable Termination Standards 568A Vs 568B*", February 12, 2007

[Securicore] Securicore Inc., Security and network solutions. Available at: http://www.securicore.ca/products/

[Gómez2005] Gómez, D., Network Taps. Available at: http://www.dgonzalez.net/papers/roc/node4.html (Updated: May 22, 2005)

[Ossmann2009] Ossmann, M., Throwing Star LAN Tap. Available at: http://greatscottgadgets.com/throwingstar/ (Posted: 2009)

[Gómez2004] Gómez, D., Network Taps. Available at: http://www.infosecwriters.com/hhworld/hh9/roc/node4.html (Updated: July 08, 2004)

[Netoptics] Net Optics, Regeneration Taps. Available at: http://www.netoptics.com/products/regeneration-taps

[NetworkCritical] Network Critical, Breakout/ Aggregation/ Regeneration TAPs v.2. Available at: http://www.networkcritical.com/Products/Smart-Network-Access-Modular-System/Breakout-Aggregation-Regeneration-TAPs

[Dualcomm] USB Powered 5-Port 10/100Base-T Ethernet Switch TAP. Available at: http://www.dual-comm.com/port-mirroring-LAN_switch.htm

[Apple] Apple USB Ethernet Adapter. Available at: http://store.apple.com/us_smb_78313/product/MC704ZM/A/apple-usb-ethernet-adapter

[M340] Schneider Electric, Modicon® M340™ automation platform – Catalog 2010.

[SRW2008] Linksys, Managed Gigabit Switch, Model: SWR2008. Available at: http://setuprouter.com/router/linksys/srw2008/manual-243.pdf

[RaspberryPiIssueNo29] Raspberry Pi issue #29: Running X/LXDE causes packet loss/dmesg errors on networking. Available at: https://github.com/raspberrypi/linux/issues/29 (Published: May 31, 2012)

[Iperf] iperf(1) – Linux manual page, "*Perform network throughput tests*". Available at: http://iperf.sourceforge.net/ (Updated: December 06, 2013)

[Top] top(1) – Linux manual page, "*Display Linux tasks*". Available at: http://linux.die.net/man/1/top

[ksoftirqd] ksoftirqd(9) – Linux manual page,"*Softirq daemon*". Available at: http://www.tin.org/bin/man.cgi?section=9&topic=ksoftirqd

[ModbusTools] Modbus Tools. Available at: http://www.modbustools.com/

[Tcpdump] tcpdump(8) – Linux manual page, "*Dump traffic on a network*". Available at: http://www.tcpdump.org/ (Updated: January 14, 2014)

[Wireshark] Wireshark, A free and open-source packet analyzer. Available at: http://www.wireshark.org/

[Redis] Redis, Sn open source, BSD licensed, advanced key-value store Available at: http://redis.io/

[MinimalModbus] MinimalModbus, Easy-to-use Modbus RTU implementation for Python. Available at: http://minimalmodbus.sourceforge.net/apiminimalmodbus.html (Updated: June 22, 2014)

[Modbus-tk] Modbus-tk, Implementation of the Modbus protocol in the Python programming language. Available at: https://github.com/glastopf/modbus-tk (Updated: November 10, 2013)

[OpenMRTS] OpenMRTS, An open source IEC 870-5-101/104 components implementation for telecontrol and supervisory systems. Available at: http://sourceforge.net/projects/mrts/ (Updated: March 13, 2013)

[Net::IEC104] Net::IEC104, Perl implementation of IEC 60870-5-104 standard (server and client). Available at: https://github.com/vlet/iec104 (Updated: May 17, 2011)

[Lekhonkhobe2014] Lekhonkhobe, T., Building Cross Compilers. Available at: https://wiki.debian.org/BuildingCrossCompilers (Updated: October 02, 2014)

[RFC826] Plummer, D., An Ethernet Address Resolution Protocol. Available at: http://tools.ietf.org/html/rfc826 (Published: November, 1982)

[Ettercap] Ettercap, A comprehensive suite for man in the middle attacks. Available at: http://ettercap.github.io/ettercap/ (Updated: September 21, 2013)

[Modbus2006] Modbus-IDA, "*Modbus Application Protocol Specification V1.1b*", December, 2006

[IEC60870-5-1] International Electrotechnical Commission, "*Telecontrol equipment and systems. Part 5: Transmission protocols - Section One: Transmission frame formats*", 1990

[IEC60870-5-2] International Electrotechnical Commission, "*Telecontrol equipment and systems - Part 5: Transmission protocols - Section 2: Link transmission procedures*", 1992

[IEC60870-5-3] International Electrotechnical Commission, "*Telecontrol equipment and systems - Part 5: Transmission protocols - Section 3: General structure of application data*", 1992

[IEC60870-5-4] International Electrotechnical Commission, "*Telecontrol equipment and systems - Part 5: Transmission protocols - Section 4: Definition and coding of application information elements*", 1993

[IEC60870-5-5] International Electrotechnical Commission, "*Telecontrol equipment and systems - Part 5: Transmission protocols - Section 5: Basic application functions*", 1995

[IEC60870-5-101] International Electrotechnical Commission, "*Telecontrol equipment and systems - Part 5-101: Transmission protocols - Companion standard for basic telecontrol tasks*", 2003

[IEC60870-5-102] International Electrotechnical Commission, "*Telecontrol equipment and systems - Part 5: Transmission protocols - Section 102: Companion standard for the transmission of integrated totals in electric power systems*", 1996

[IEC60870-5-103] International Electrotechnical Commission, "*Telecontrol equipment and systems - Part 5-103: Transmission protocols - Companion standard for the informative interface of protection equipment*", 1997

[IEC60870-5-104] International Electrotechnical Commission, "*Telecontrol equipment and systems - Part 5-104: Transmission protocols - Network access for IEC 60870-5-101 using standard transport profiles*", 2006

[IEEE1815-2010] IEEE, IEEE Standard for Electric Power Systems Communications – Distributed Network Protocol (DNP3) - IEEE 1815-2010, 2010

[TriangleMicroworks1999] West, A., Triangle MicroWorks Inc., "*Communication Standards in Power Control*", 1999

[Weiqing2010a] T. Weiqing, "*Realization of IEC 60870-5-104 Protocol in DTU - Fig.2 network reference model*", 2010

[Weiqing2010b] T. Weiqing, "*Realization of IEC 60870-5-104 Protocol in DTU - Fig. 3 APDU of the defined telecontrol companion standard - Figure 5.25 Application function codes*", 2010

[Gordon2004] C. Gordon, "*Modern SCADA Protocols: DNP3, 60870.5 and Related Systems*", 2004

[Lian2011a] MAYR Software, "*Types of control field formats - I-Format*", 2011

[Lian2011b] MAYR Software, "*Types of control field formats - S-Format*", 2011

[Lian2011c] MAYR Software, "*Types of control field formats - U-Format*", 2011

[Jay2003a] M. Jay, "*Comparison of protocols used in remote monitoring: DNP 3.0, IEC 870-5-101 & Modbus - Figure 6 - ASDU frame details for IEC 870-5-101*", 2003

[ABB2010a] ABB, "*RER620 - IEC 60870-5-101/104 Communication Protocol Manual - Table 2: Supported transmission services initiated by the controlling station*", 2011

[ABB2010b] ABB, "*RER620 - IEC 60870-5-101/104 Communication Protocol Manual – 2.2.2 Balanced transmission*", 2011

[Profibus] PROFIBUS and PROFINET International (PI). Available at: http://www.profibus.com/

ANNEXES

ANNEX A – MODBUS TCP PROTOCOL

The TCP version of the Modbus protocol uses the same philosophy of the Modbus RTU variant, with some changes in terms of message structure. It was introduced to add support to TCP/IP networks, using TCP port 502 and non-privileged ports (above 1024) for Slave and Master devices, respectively. Its framing is based on the RTU variant, with an identical Protocol Data Unit (PDU) but, without the remaining Application Data Unit (ADU) fields (the Additional Address and Error Check fields are not used) and, an additional header (MBAP – Modbus Application Header) containing the following fields (cf. Figure 8-1):

- A Transaction Identifier (2 bytes), identifying the request to ensure coherence in case responses arrive out of sequence (in Modbus TCP, a slave can handle several requests simultaneously).
- A Protocol Identifier (2 bytes), filled by the Master station.
- The Length field containing the number of used bytes.
- The Unit Identifier is used when RTU devices are used in a TCP environment, through protocol gateways.

The CRC16 error check field used in Modbus RTU is discarded in Modbus TCP, since it is assumed that the TCP/IP protocol stack already offers integrity control for payloads. It is also possible to convert Modbus RTU equipment for TCP operation, using special-purpose gateways for translation purposes [IDC2009].

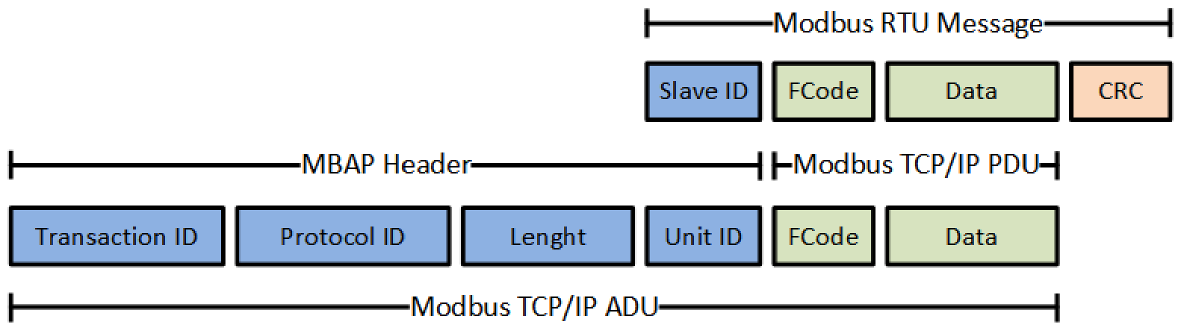


Figure A-1 – Modbus TCP Framing [Modbus2006]

The fields referent to the Modbus TCP/IP PDU, relate to requested operations (Function Code), and additional information (Data). The Function Code field may contain any integer value from 0 to 255, being the range of [0, 127] referent to requests sent by the Master and, the range of [128, 255] relative to responses sent by the Slave. The meaning of these values is explained with a simple pseudo-code (cf. Table 8-1):

Table A-1 – Function Code Operation

Function Code Operation (pseudo-code)

```
START
IF (values between [128, 255])
    There was an error executing the command
ELSE IF (response value = request value)
    Slave executed the command
ELSE
    Slave alerts Master of an error
END
```

Note that the value 0 (zero) is never used, even though it was referred that the range of these values is between 0 and 255. Reading the code above (cf. Table 8-1), if the values range between 128 and 255 it means that an error occurred. On the other hand, if both request and response values match, means that the Slave executed the command successfully. In case none of the above conditions are met, the Slave notifies the Master that an error occurred.

The functions used for reading, writing and all the other remaining operations are categorized by class, the respective Function Name and Code (cf. Table 8-2):

Table A-2 – Transaction types

Class	Function Name (FN)	Function (FC)
Class 0	Read Multiple Registers	3
	Write Multiple Registers	16
Class 1	Read Coils	1
	Read Input Discretes	2
	Read Input Registers	4
	Write Coil	5
	Write Single Register	6
	Read Exception Status	7
Class 2	Force Multiple Coils	15
	Read General Reference	20
	Write General Reference	21
	Mask Write Register	22
	Read/Write Registers	23
	Read FIFO Queue	24

As for the Data field, in case the requested command has successfully executed, the message contains the exact information requested by the Master. Otherwise, it only includes information concerning the error message (cf. Table 8-3).

Table A-3 – Data Bytes Operation

Data Bytes Operation (pseudo-code)

START
IF (execution = SUCCESS)
    The message contains the data requested by the Master
ELSE
    The message contains information about the error
END

ANNEX B – IEC 60870-5-101/104

The IEC 60870-5 (IEC 60870 part 5) standard defines five transmission protocol documents for sending basic telecontrol messages between two systems, using permanent and directly connected data circuits and, standard profiles necessary to uniform applications, such as the IEC 60870-5-101 (IEC101), where the way a device acts is defined. Each one of these documents are then briefly described:

- *IEC 60870-5-1* [IEC60870-5-1] – Specification of standards for coding, formatting and synchronizing data frames to be transmitted, of fixed and variable length, meeting specified data integrity requirements. These services are provided by the data link and physical layers for telecontrol applications.
- *IEC 60870-5-2* [IEC60870-5-2] – Services for data link transmission using a control field and an optional address field (some point-to-point topologies do not require either the source or the destination address).
- *IEC 60870-5-3* [IEC60870-5-3] – Rules for general structuring of application data in frame transmission, without specifying details about information fields and their contents. These rules are also intended to be used by a great variety of telecontrol application in the future.
- *IEC 60870-5-4* [IEC60870-5-4] – Rules for the definition and coding of information elements, particularly digital and analogue process variables used in telecontrol applications.
- *IEC 60870-5-5* [IEC60870-5-5] – Definition of basic application functions that perform standard procedures for telecontrol systems, situated between the Open System Interconnection (OSI) application layer and the application program (cf. Table 8-4, green shaded section). These functions are used for specific telecontrol tasks, which result in the following application profiles, generated by the IEC Technical Committee 57 (Working Group 03):
    - *IEC 60870-5-101* [IEC60870-5-101] – Transmission protocols (basic telecontrol tasks).
    - *IEC 60870-5-102* [IEC60870-5-102] – Transmission of integrated totals in electric power systems (not widely used).
    - *IEC 60870-5-103* [IEC60870-5-103] – Transmission protocols (informative interface of protection equipment).
    - *IEC 60870-5-104* [IEC60870-5-104] – Transmission protocols (network access for IEC101).

Any functions that are not defined in the documents listed above, must be specified within the profile. Examples of these functions are: Station Initialization, Cyclic Data Transmission, Data Acquisition by Polling and Station Configuration.

Unlike Modbus protocol, the IEC 870-5 standard (IEC 60870-5), as well as the DNP3 [IEEE1815-2010], is based on a three-layer reference model (the Enhanced Performance Architecture (EPA) – cf. Figure 8-2), used for efficient implementation within RTU devices, also defining basic application functionality for a user layer, which adds interoperability for functions like Clock Synchronization and File Transfer.

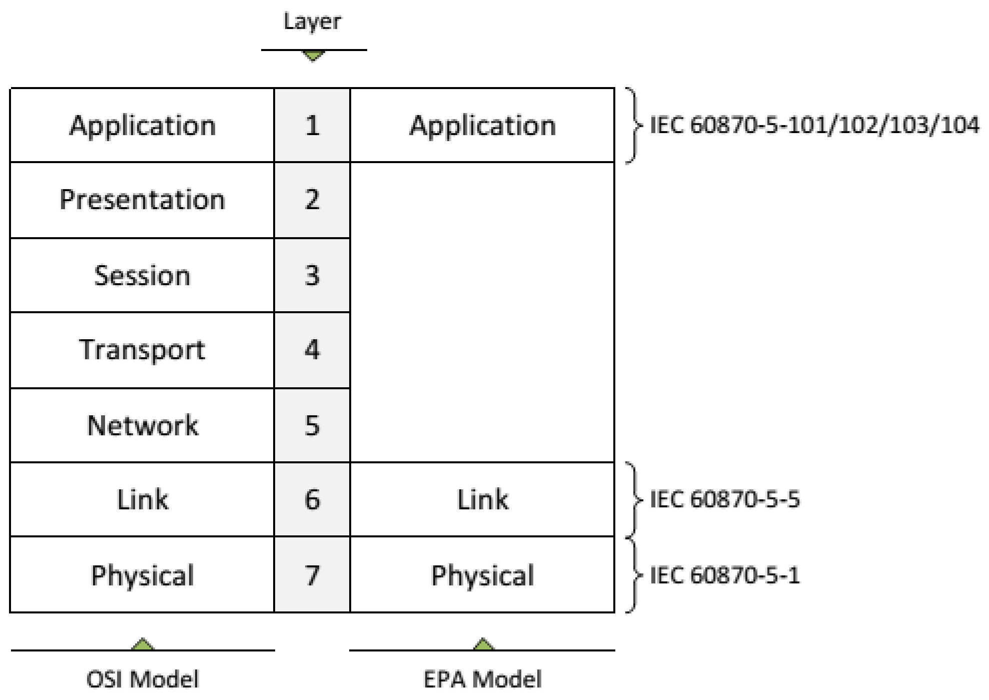


Figure B-1 – Reference models [TriangleMicroworks1999]

The main reason the EPA model only has 3 layers is to reduce the overhead of the 7-layer model (OSI), so it can be optimized for SCADA environments. As shown in the previous illustration, a correspondence is made between the OSI and EPA models so it is better understood where the layers match. Also, next to the EPA model, at each one of its 3 layers, is represented the location of the base documents and profiles, as mentioned above.

As stated before, the IEC104 standard is in fact an extension of the IEC101 to support TCP/IP connection, transporting IEC101 ASDUs (Application Service Data Units), based on the ISO-OSI reference model but, only using 5 of these layers. So, while the IEC101 standard is intended to work on serial RS232 lines, the IEC104 standard, an extension of the previous, communicates over TCP/IP with changes implemented on the transport, network, link and physical layers to enable such communications. The application layer is maintained in both standards. The IEC 60870-5-104 reference model is located in the application layer (cf. Table 8-4).

Table B-1 – IEC 60870-5-101/104 network reference model [Weiqing2010a]

Layer	Description
User layer[18]	Selected application functions of IEC 60870-5-5:
	a) Station Initialization
	b) Cyclic Data Transmission
	c) General Interrogation
	d) Command Transmission
	e) Parameter Loading
	f) File Transfer
	g) Data Acquisition by Polling
	h) Acquisition of Events
	i) Clock Synchronization
	j) Transmission of Integrated Totals
	k) Test Procedure
Application layer (7)	Selection of ASDU from IEC 60870-5-101 and 104.
	Application Protocol Control Information (APCI). Transport Interface (User to TCP interface).
Transport Layer (4)	Selection of TCP/IP Protocol Suite (RFC 2200)
Network Layer (3)	
Link Layer (2)	
Physical Layer (1)	

In respect to the 3 layers referent to the IEC101 protocol, following is a description of what each one of these represents:

- *Application Layer* – Selected application information elements of IEC 60870-5-4 for definition and coding of information elements and, the ASDUs of IEC 60870-5-3 for general structure of application data. The contents and sizes of individual information fields of the ASDUs (cf. Figure 8-4) are specified according to the declaration rules for information elements defined

[18] User layer does not correspond to a real layer of the OSI model. It is just a representation to better understand the application functionality defined in IEC 60870-5.

in IEC 60870-5-4.

Also, Type Information defines the structure, type and format for information objects. These 2 predefined parameters (Elements and Type Information) do not allow the addition of new information elements or types by any vendor. In fact, the information elements have been defined for equipment protection, voltage regulators and for meter values to interface Intelligent Electronic Devices (IEDs) with the RTUs.

- *Link Layer* – Selected link transmission procedures of IEC 60870-5-2 for data link transmission services and the transmission frame formats of IEC 60870-5-1. The transmission mode (balanced or unbalanced) is also defined in this layer as well as the provided addresses for each link.
- *Physical Layer* – Selected International Telecommunication Union Telecommunication Standardization Sector (ITU-T) recommendations, defining the hardware-dependent specifications of the IEC 60870-5-101 and 104 communication interfaces, compatible with Electronic Industries Association (EIA) standards RS-232 and RS-485 , also supporting fiber optic interfaces.

  The IEC 60870-5-1 standard offers the asynchronous FT 1.2 frame format, specified in IEC101, to provide data integrity with the maximum efficiency for acceptable convenience of implementation, using standard Universal Asynchronous Receiver/Transmitters (UARTs).

The IEC104 protocol provides 255 bytes APDU packets (including start character and length identification), meaning that the ASDU maximum length is 253. Also, the APDU length includes 4 octets of control field and ASDU, meaning that the maximum ASDU length is 249. So, this type of provision limits an APDU packet to send up to 121 normalized measured values without the quality descriptor or a 243 single-point information data. Otherwise, if the amount of collected data by an RTU exceeds the above limit, the APDU packet has to be divided before being sent. The APDU packet structure is illustrated bellow (cf. Figure 8-3), as previously mentioned.

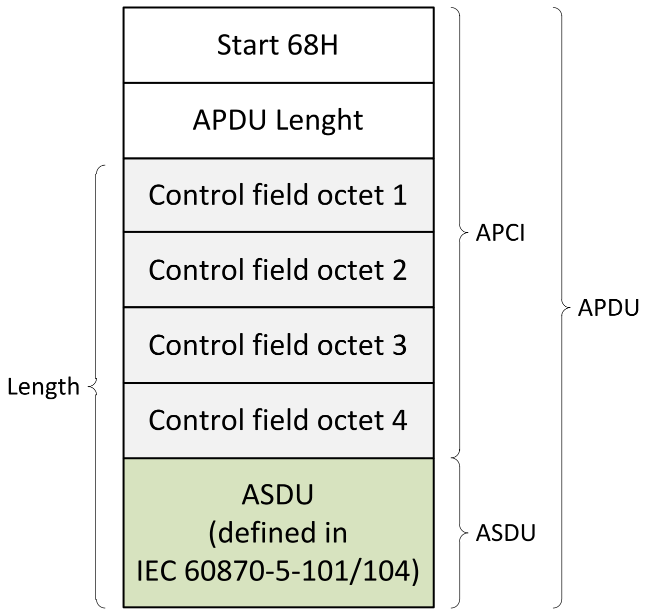
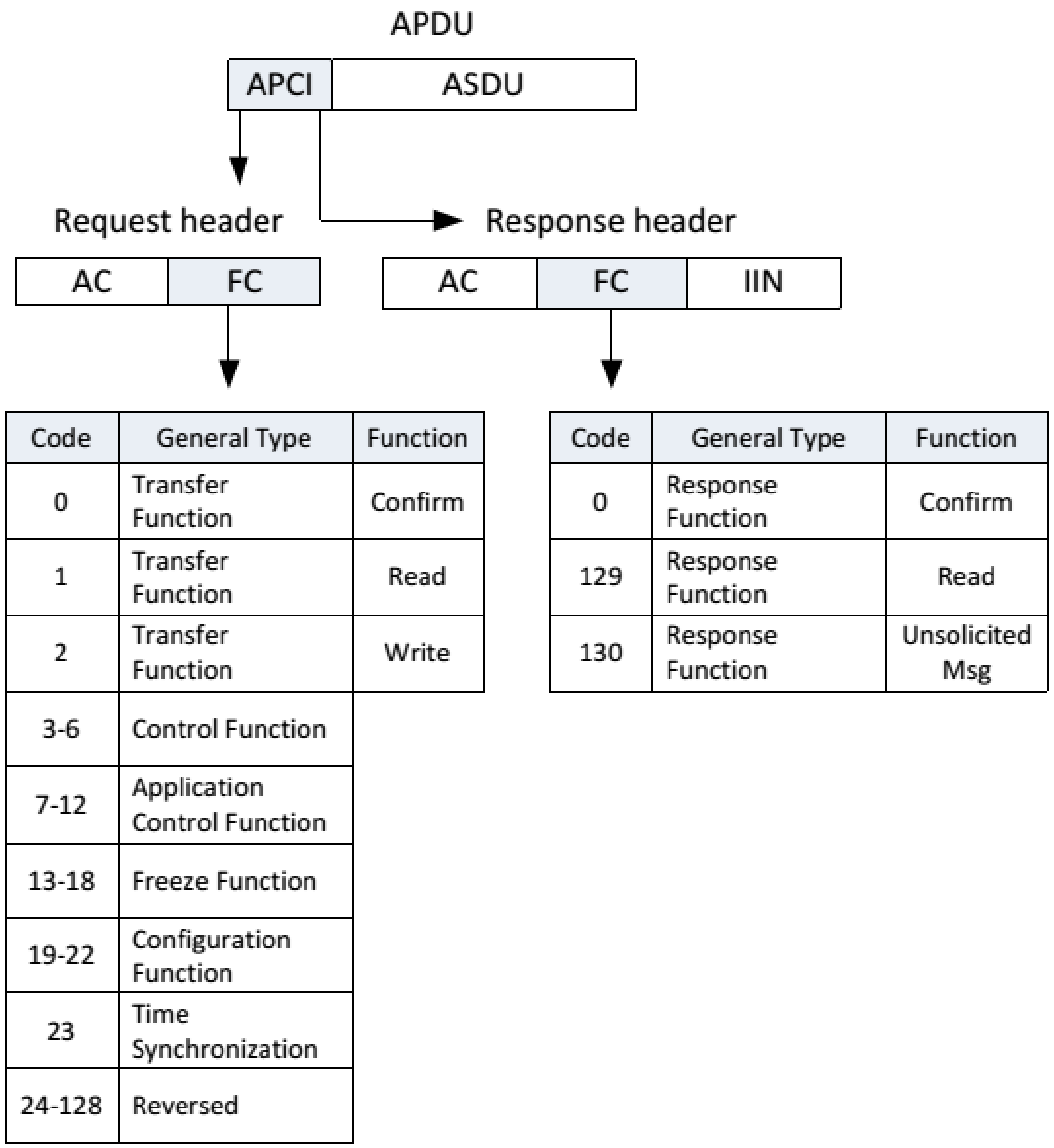


Figure B-2 – APDU Packet [Weiqing2010b]

The application header (cf. Table 8-4), is referred to as the Application Protocol Control Information (APCI), which may either be 2 or 4 bytes, depending whether it is a request or a response (cf. Figure 8-4).



Code	General Type	Function
0	Transfer Function	Confirm
1	Transfer Function	Read
2	Transfer Function	Write
3-6	Control Function	
7-12	Application Control Function	
13-18	Freeze Function	
19-22	Configuration Function	
23	Time Synchronization	
24-128	Reversed	

Code	General Type	Function
0	Response Function	Confirm
129	Response Function	Read
130	Response Function	Unsolicited Msg

Figure B-3 – Application function codes [Gordon2004]

The keyword AC stands for Application Control, which has a corresponding Function Code (FC) and type. The control fields of the Modbus APDU illustrated above (cf. Figure 8-3) define the control information for protection against message loss or duplication, start and stop of message transfers, and for supervision of transport connections. These octets can be classified into three kinds of message formats by its definition.

- *I format (Numbered information transfer)* – This filed is used for APDUs containing an ASDU, i.e., information being indicated by a 0 (zero) in the first bit position. The frame is then represented bellow (cf. Figure 8-5).

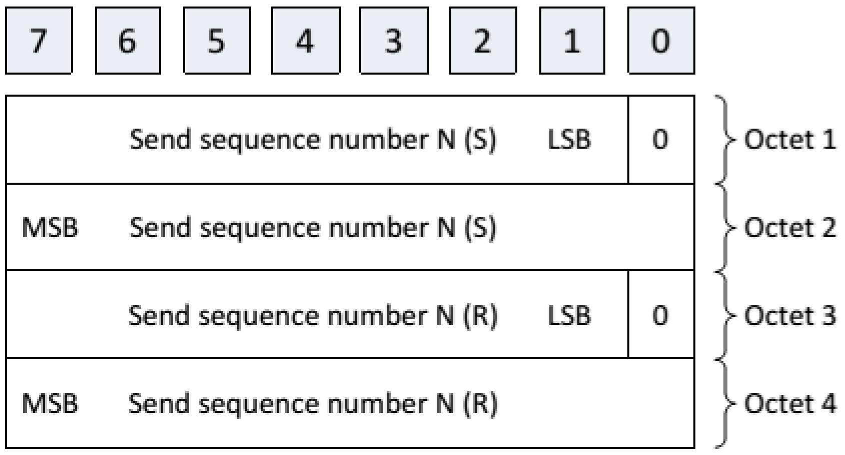

7
6
5
4
3
2
1
0
Send sequence number N (S) LSB
0
Octet 1
MSB Send sequence number N (S)
Octet 2
Send sequence number N (R) LSB
0
Octet 3
MSB Send sequence number N (R)
Octet 4

Figure B-4 – Information (I) format control field (Variable length frame) [Lian2011a]

- *S format (Numbered supervisory functions)* – This filed is used for APDUs containing only the APCI header. Unlike the previous, these frames do not have any information attached and so, are only used for controlling the transport of the APDUs. It is indicated by 1 in the first bit position, followed by a 0 (zero) in the second bit position (cf. Figure 8-6)

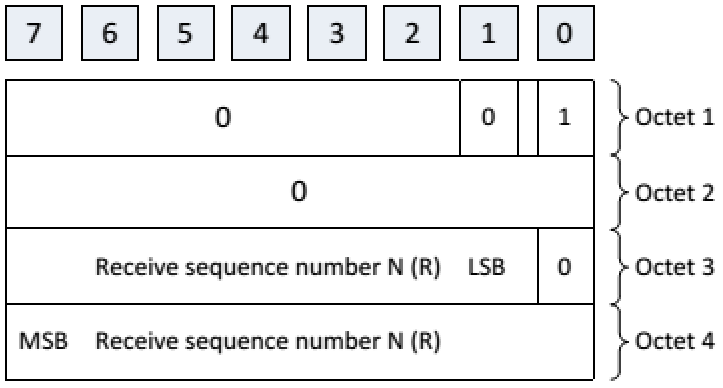

7
6
5
4
3
2
1
0
0
0
1
Octet 1
0
Octet 2
Receive sequence number N (R) LSB
0
Octet 3
MSB Receive sequence number N (R)
Octet 4

Figure B-5 – Supervisory (S) format control field [Lian2011b]

- *U format (Unnumbered control functions)* – Just like the previous, this field is also used in APDUs that only contain the APCI. It is used as a start-stop mechanism for information flow or when more than one connection is available between stations, also allowing a changeover between these connections without losing data (cf. Figure 8-9). Also, it should be noted that there are no sequence numbers.

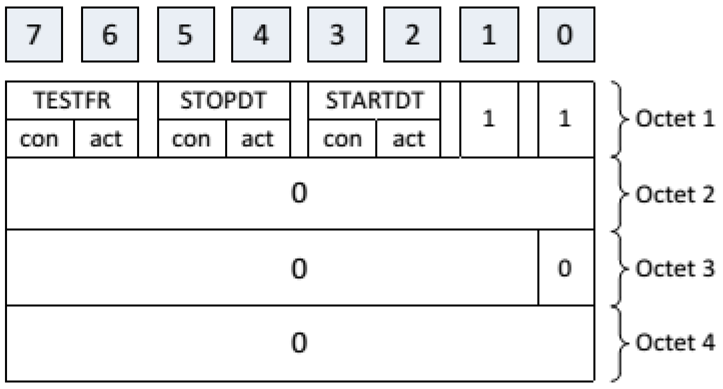


Figure B-6 – Unnumbered control (U) format control field [Lian2011c]

The sequence numbers in these control fields are used to control the APSUs flows in both directions. Once the receiver gets an APDU, it advertises the sender of the highest sequence number, using an I or S format message, so the sender can re-send ASDUs that might have been lost. This also depends on whether the receiver is sending information in the opposite direction.

A detailed view of the ASDU frame (green shaded section in Figure 8-3) is illustrated bellow with the respective fixed and variable fields, as in the IEC 60870-5-1 document (cf. Figure 8-8).

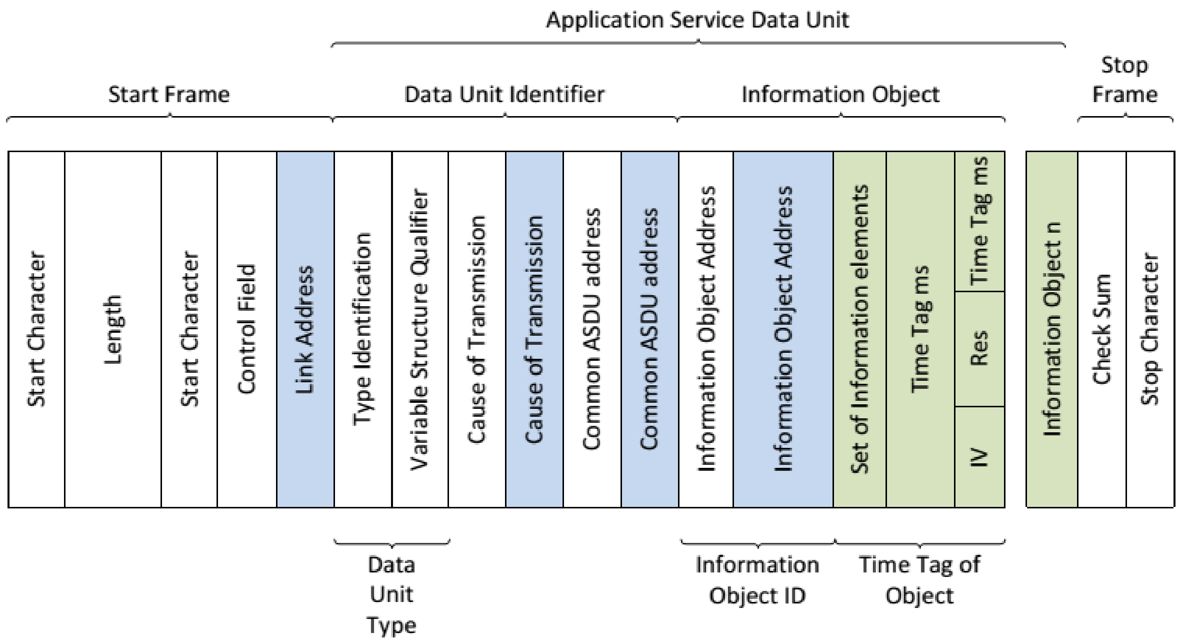


Figure B-7 – ASDU Frame [Jay2003a]

The blue shaded sections are the optional fields which will be determined by a system level parameter shared by all devices in the system. The green shaded sections are the variable fields per ASDU, whose common address size is determined by a fixed system parameter, in this case 1 or 2 octets (bytes). The remaining fields are all fixed per ASDU.

The IEC101 standard profile has 2 definitions that are not present in any of the documents previously referred:

- *Control direction* – Transmission from the controlling station to the controlled station.
- *Monitor direction* – Direction of the transmission from the controlling station to the controlled station.

In order to aid administrators in the configuration of their SCADA systems, the IEC101 profile defines a check list in which these can ensure interoperability between the used devices and the ones from other vendors. This list does not only contains information from the ASDU for both control and monitor direction (as previously referred), but also includes parameters such as baud rate, ASDU field length common address, link transmission procedure and, basic application functions defined in IEC 60870-5-102 and 105 documents. This check list allows vendors to define their devices or system in a protocol perspective.

When communicating, both devices in the SCADA system using the IEC 60870-5-101 protocol can perform its transmissions in 2 different modes: balanced and unbalanced. At the data link layer, the IEC101 standard profile species whether an unbalanced (includes multidrop) or balanced (includes point-to-point) transmission mode is used, together with the link procedures (and corresponding link function codes) to be used and, an unambiguous number (address) for each link. Following are described each one of the transmission modes:

- *Unbalanced mode* – In this case, only the Master station can initiate a transmission, polling the controlled outstations, which can only respond when the requests are sent by it, the Master. The supported transmission service types selected from IEC 60870-5-2 are described below (cf. Table 8-5):

Table B-2 Service types initiated by the Master station [ABB2010a]

Service	Description
SEND / NO REPLY	Global messages and cyclic set-point commands for the Master station
SEND / CONFIRM	Control and set-point commands from the Master station
REQUEST / RESPOND	Data polling from the controlled outstations

- *Balanced mode* – Unlike the previous case, here any station involved in the communication can initiate the transmission of messages, acting as the controlling (Master) stations or controlled outstations, simultaneously. Therefore, these devices are called combined stations, being restricted to point-to-point and to multiple point-to-point configurations. Following are described the supported transmission services (cf. Table 8-6).

Table B-3 – Service types initiated by controlling and controlled stations [ABB2010b]

Service	Description
SEND / NO REPLY	Global messages and cyclic set-point commands. This can only be initiated by a controlling station with a broadcast address in a multiple point-to-point configuration
SEND / CONFIRM	Control and set-point commands

Following is a list of basic application functions implemented by the current standard, as it has been referred:

- *Data acquisition* – Since the data may appear faster than the communication link is able to transfer, the controlled station buffers all data such as, command replies or process values collected cyclically, upon change or request from the Master Station. The actions performed on the buffered data varies whether balanced or unbalanced transmission is used: For unbalanced transmission, on the link layer the controlled stations wait for a request coming from the Master Station, which polls the buffered data. On the other hand, for balanced transmission, the controlled station transmits the data to the Master Station without delay.
- *Event acquisition* – The events occur at the controlled station's application level, being also buffered for the same reasons mentioned for Data acquisition.
- *Interrogation* – This function is used to update the controlling station after an internal station initialization or when information loss is detected, being performed either by an interrogation group (1-16) at a time, or all groups at once. When requested, the controlled stations transmit the actual values of their process variables.
- *Clock synchronization* – After the clock of the controlled station is synchronized with the one on the controlling station, it keeps synchronizing periodically with the *C_CS ACT* command. This provides a correct chronological sequence of time-tagged events or information objects. When an ASDU is received, the time information must be corrected by one of the end devices. Also, the transmission delays are calculated by a delay acquisition command so the time is corrected at the controlled station when sending.
- *Command transmission* – In order to change the state of the operation equipment, a command may be sent by the controlling station, which can be one of the following:
    - *Direct command* – Used to immediately control operations in the controlled stations. For safety purposes, the permissibility and validity of the received messages are checked.
    - *Select and execute command* – Used to prepare a specified control operation in a controlled station, check if the correct control operation is prepared and, finally, execute the command. In this case, the preparation is checked by an operator or by an application procedure and if the controlled station does not receive the correct

execute indication, the control operation does not start. The controlled station receives a command transmission confirmation through an activation confirmation response and after the command is executed, an activation termination response is sent to the controlling station.

- *Integrated totals transmission* – An integrated total is a value that is integrated over a specified period of time. In the other hand, a system parameter corresponds to the specific clock times and the periodic time interval of successive acquisitions of the integrated totals. Two methods for acquiring counter information are: Acquisition of integrated totals (Freeze-and-read); and acquisition of incremental information (Clear-and-Read).
- *Protocol and Link parameters changes* – When changed, the new values of the protocol and link parameters take effect after they have been committed.
- *Transmission delay acquisition* – Time correction is determined by the sum between the transmission delay and the internal equipment delay. To obtain the value of the transmission delay, either parameterization or using a dynamic procedure (initiated by the controlling station), are both valid alternatives.
- *Analog Value Deadband* – The use of the deadband feature allows a user to reduce the number of unnecessary events using analogue measurements for each point, which might be configured using proper tools, by setting 2 parameters:
    - *Range* – Considering a range of 0.05 (5%), if the data point value changes beyond this value from the previously sent one, the data will be sent as deadband data.
    - *Interval* – Limits the deadband value to be sent once per configured time window, in seconds. To disable this feature, the interval is set to 0.

Annex C – Comparison of IEC101, DNP3 and Modbus

With the Modus, DNP 3 and IEC104 protocols already described, it is interesting to check a comparative table in which their features are briefly described. Table 8-7 describes a comparison based on the features of each protocol. Some of the most relevant information in this table includes, features by layer (Physical, Data link and Application layers), addressing, required parameters, application specific information, etc.

Table C-1 – Comparison of IEC101, DNP3 and Modbus

Feature	IEC 870-5-101	DNP 3.0	Modbus
Standardization	IEC Standard (1995) Amendments 2000,2001	Open industry specification (1993)	Not Applicable
Standardization Organization	IEC TC 57 WG 03	DNP users group	Modicon Inc.
Architecture	3-layer EPA architecture	4-layer architecture Also supports 7 layer TCP/IP or UDP/IP	Application layer messaging protocol
Physical layer	Balanced Mode - Point to Point Multipoint to point Implementation by X.24 / X.27 standard Unbalanced Mode - Point to Point Point to Multipoint Implementation by V.24 / V.28 standard	Balanced mode transmission It supports multiple masters, multiple slave and peer-to-peer communication RS 232 or RS 485 implementation TCP/IP over Ethernet, 802.3 or X.21	Balanced mode of transmission RS 232 serial interface implementation Peer to peer communication TCP/IP over Ethernet
Data link layer	Frame format FT 1.2 Hamming distance - 4	Frame format FT3 Hamming distance-6	Two types of message frames are used: ASCII mode and RTU mode
Application layer	Both IEC 870-5-101 and DNP 3.0 provides: > Time synchronization	Remote starting / stopping of software applications	Does not give time stamped events. We have sequence of events (without time

	> Time stamped events > Select before operate > Polled report by exception > Unsolicited responses > Data group/classes Limited to single data type per message Can control one point per message only No internal indication bits No application layer confirms for events	Polling by data priority level Broadcast addressing Multiple data types per message are allowed Internal Indication field IID present in response header Application layer confirms events; use of CON bit is made	but not event list with time. Does not provide polled report by exception Checksum ensures proper end-to-end communication
Device Addressing	Link address could be 0, 1, 2 bytes Unbalanced link contains slave address Balanced link is point to point so link address is optional (may be included for security)	Link contains both source and destination address (both always 16 bits) Application layer does not contains address 32 b point addresses of each data type per device	Addresses field contains two characters (ASCII mode) or 8 bits (RTU mode) Valid address in range 1-247 Address 0 used for broadcast
Configuration Parameters required	Baud rate Device addresses Balanced / unbalanced Frame length	Baud rate Device addresses Fragment size	Baud rate Mode ASCII or RTU Parity mode

	Size of link address Size of ASDU address Size/structure of point number Size of cause of transmission		
Application Specific information model	A few application specific data types available Data objects and messages are not independent to each other	Permits vendors to create application specific extensions Data objects and messages independent to each other	Allows user to create application specific model
Cyclic transmission	Eliminates static data poll message from master Interrupted by event triggered communication request	Available but interval cannot be remotely adjusted	Not Applicable
Dominant market	Europe (South America, Australia and china)	North America (Australia and china)	Used worldwide for application with low volume data
Online configurations	Enable/ disable communication control objects Loading configuration Change report / logging behavior	Define group of data Selecting data for responding Enable/ disable communication control objects Loading configuration Change report / logging behavior	Efficient online configuration could be made by Modbus TCP/IP

Open for other encoding solutions	Not Available	Yes open for other encoding solutions like XML	Yes. One could write source code in programming languages like C, VC++ & JAVA etc.

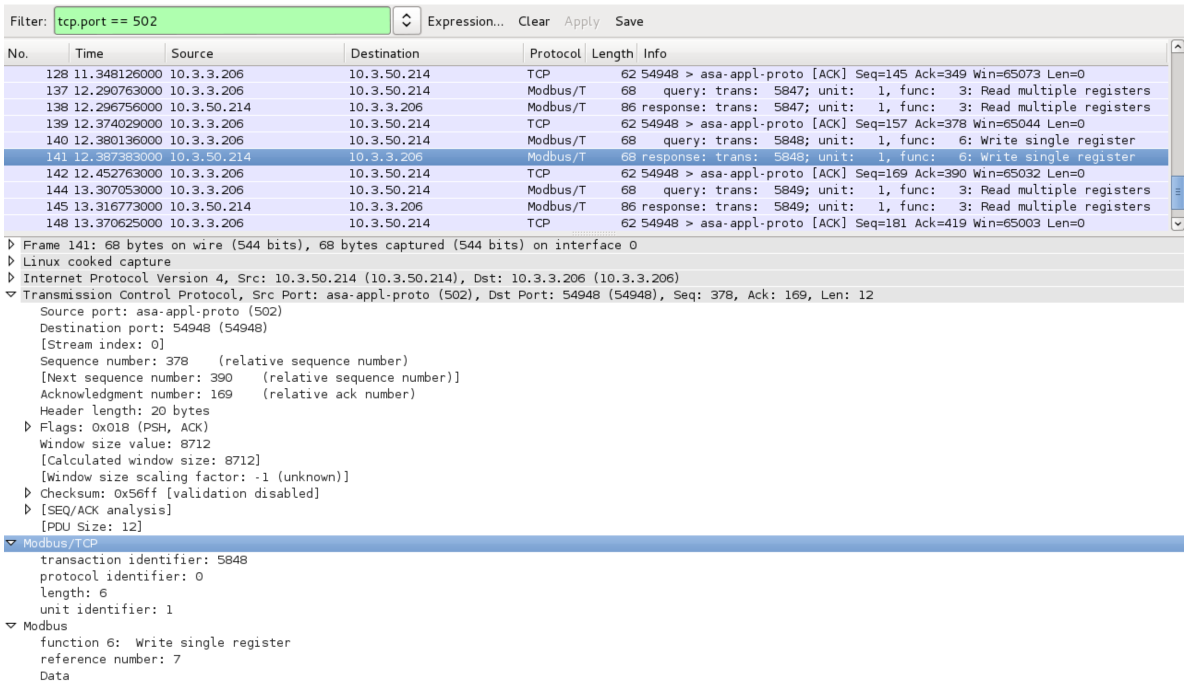
The protocol selection depends on the scenario and the operator expectancies. There isn't a best protocol for the every situation. With this in mind, some issues must be cleared in order to get the proper solution, as those presented below:

- Application domain – When dealing with utilities or oil and gas industries, the operators should go either with IEC101/104 or DNP 3. Mainly, if the SCADA system has requirements such as time-stamping. Since the Modbus protocol is more a general purpose solution, it is more suited for industrial applications with direct register mapping with small volumes of data.
- Communication devices – Depending on which are the communicating devices, one might have one of the following situations: If communication with substations there are protocols meant for protection control and metering, such as Modbus, IEC103 or Profibus [Profibus]; if the communication is established outside the substations, protocols used for the exchange of data between substations and master control centers are IEC101/104 or DNP 3; For communications between applications, there is the standard IEC 61968, still under development.
- Specific requirements (e.g., amount of data, bandwidth, response time and distance between devices) – When sending large volumes of data, both DNP 3 and IEC101 present good solutions, since the first sends small number of large sized data, and the second sends large number of small sized data. However, if there is the need to transmit huge volumes of data across long distances, working with high baud rates, the DNP 3 becomes the favored one over the IEC101. But if a much more simple setup is to be used, the Modbus is the perfect solution since it requires less memory, has fewer data types, has smaller frame sizes, is fast (packs a lot of information per message) and it is safer maintaining the data integrity since it is always required to poll the process.
- Devices to equip (e.g., Embedded devices, PLCs, Personal Computers (PC)) – If using embedded controllers with little memory requirements, the Modbus protocols is the best solution.
- Interface functions (e.g., Parameterize relays remotely, download disturbance data and events, retrieve measurements) – Since Modbus is much more simple than the other two protocols, a master and slave have to be implemented, in order for the protocol to operate fully (if different vendor devices are used it may present an obstacle). Although, if much more simple operations are required (e.g., read/write register) the Modbus protocol can be used without any kind of problems.

- Domain players – Depending on the brand and model of the device, the proper protocol should be selected from a list of the supported ones.
- Geographical location – If the scenario is going to be deployed in Europe, the most obvious choice would be IEC101/104. If in North America, DNP3 would be used instead.

ANNEX D – MODBUS TCP PACKET ANALYSIS

Once performed the network packet capture, resulting from the exchange of Modbus TCP messages between the Master and Slave devices, a deeper analysis was conducted using the Wireshark tool. The following capture presents a Read Multiple Registers operations intercalated with a Write Single Register (cf. Figure 8-9).



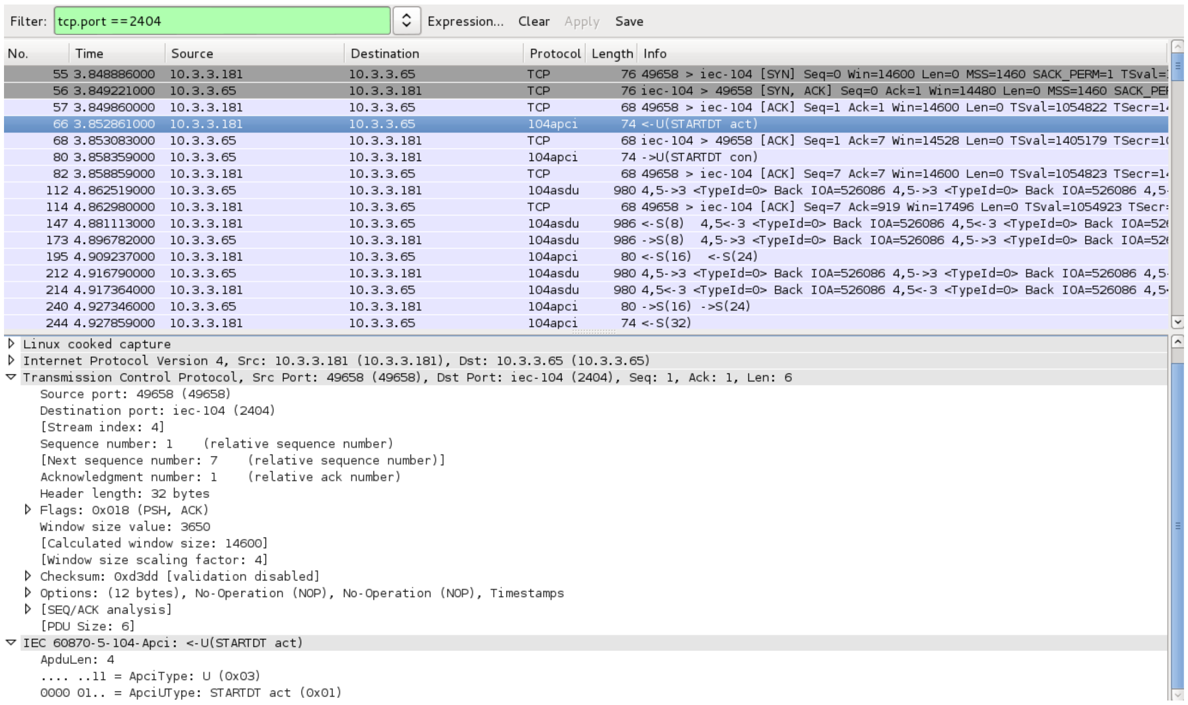
```
Filter: tcp.port == 502     Expression...  Clear  Apply  Save

No.  Time          Source        Destination   Protocol  Length Info
 128 11.348126000 10.3.3.206    10.3.50.214   TCP       62 54948 > asa-appl-proto [ACK] Seq=145 Ack=349 Win=65073 Len=0
 137 12.290763000 10.3.3.206    10.3.50.214   Modbus/T  68    query: trans:  5847; unit:   1, func:   3: Read multiple registers
 138 12.296756000 10.3.50.214   10.3.3.206    Modbus/T  86 response: trans:  5847; unit:   1, func:   3: Read multiple registers
 139 12.374029000 10.3.3.206    10.3.50.214   TCP       62 54948 > asa-appl-proto [ACK] Seq=157 Ack=378 Win=65044 Len=0
 140 12.380136000 10.3.3.206    10.3.50.214   Modbus/T  68    query: trans:  5848; unit:   1, func:   6: Write single register
 141 12.387383000 10.3.50.214   10.3.3.206    Modbus/T  68 response: trans:  5848; unit:   1, func:   6: Write single register
 142 12.452763000 10.3.3.206    10.3.50.214   TCP       62 54948 > asa-appl-proto [ACK] Seq=169 Ack=390 Win=65032 Len=0
 144 13.307053000 10.3.3.206    10.3.50.214   Modbus/T  68    query: trans:  5849; unit:   1, func:   3: Read multiple registers
 145 13.316773000 10.3.50.214   10.3.3.206    Modbus/T  86 response: trans:  5849; unit:   1, func:   3: Read multiple registers
 148 13.370625000 10.3.3.206    10.3.50.214   TCP       62 54948 > asa-appl-proto [ACK] Seq=181 Ack=419 Win=65003 Len=0

▷ Frame 141: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▷ Linux cooked capture
▷ Internet Protocol Version 4, Src: 10.3.50.214 (10.3.50.214), Dst: 10.3.3.206 (10.3.3.206)
▽ Transmission Control Protocol, Src Port: asa-appl-proto (502), Dst Port: 54948 (54948), Seq: 378, Ack: 169, Len: 12
    Source port: asa-appl-proto (502)
    Destination port: 54948 (54948)
    [Stream index: 0]
    Sequence number: 378    (relative sequence number)
    [Next sequence number: 390    (relative sequence number)]
    Acknowledgment number: 169    (relative ack number)
    Header length: 20 bytes
  ▷ Flags: 0x018 (PSH, ACK)
    Window size value: 8712
    [Calculated window size: 8712]
    [Window size scaling factor: -1 (unknown)]
  ▷ Checksum: 0x56ff [validation disabled]
  ▷ [SEQ/ACK analysis]
    [PDU Size: 12]
▽ Modbus/TCP
    transaction identifier: 5848
    protocol identifier: 0
    length: 6
    unit identifier: 1
▽ Modbus
    function 6:  Write single register
    reference number: 7
    Data
```

Figure D-1 – Modbus packet analysis

While communicating, both end devices exchange Modbus TCP (Modbus/T) request and response messages, along with a TCP packet sent from the Master Station, used for control. The selected response message, presents two Modbus sections: the Modbus/TCP section keeps track of every field contained in the MBAP header; while the following section (Modbus) presents the Modbus TCP/IP PDU.

ANNEX E – IEC 60870-5-104 PACKET ANALYSIS

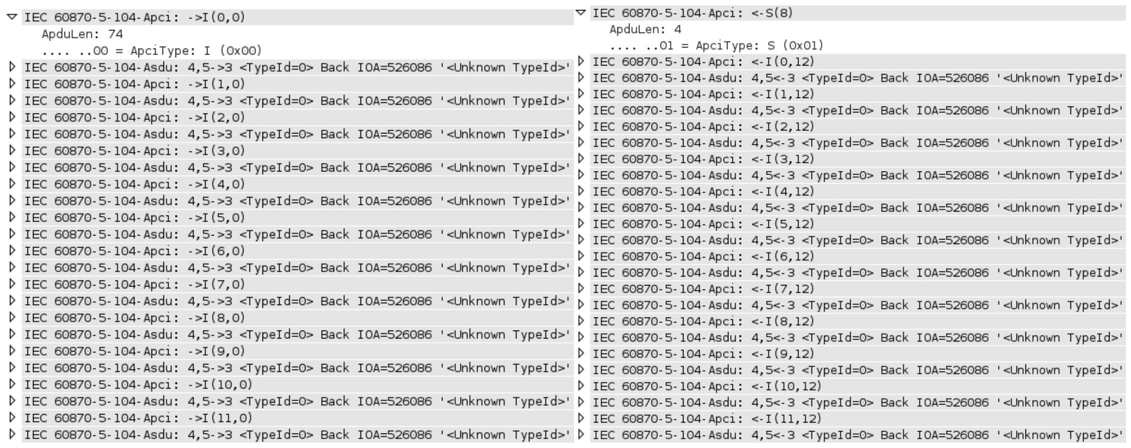
Following are presented the results from the experiments using OpenMRTS and the network monitoring and capture tool Wireshark (cf. Figure 8-10).



```
Filter: tcp.port ==2404    Expression...  Clear  Apply  Save

No.    Time         Source       Destination  Protocol Length Info
    55 3.848886000  10.3.3.181   10.3.3.65    TCP        76 49658 > iec-104 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=
    56 3.849221000  10.3.3.65    10.3.3.181   TCP        76 iec-104 > 49658 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PE
    57 3.849860000  10.3.3.181   10.3.3.65    TCP        68 49658 > iec-104 [ACK] Seq=1 Ack=1 Win=14600 Len=0 TSval=1054822 TSecr=1
    66 3.852861000  10.3.3.181   10.3.3.65    104apci    74 <-U(STARTDT act)
    68 3.853083000  10.3.3.65    10.3.3.181   TCP        68 iec-104 > 49658 [ACK] Seq=1 Ack=7 Win=14528 Len=0 TSval=1405179 TSecr=1
    80 3.858359000  10.3.3.65    10.3.3.181   104apci    74 ->U(STARTDT con)
    82 3.858859000  10.3.3.181   10.3.3.65    TCP        68 49658 > iec-104 [ACK] Seq=7 Ack=7 Win=14600 Len=0 TSval=1054823 TSecr=1
   112 4.862519000  10.3.3.65    10.3.3.181   104asdu   980 4,5->3 <TypeId=0> Back IOA=526086 4,5->3 <TypeId=0> Back IOA=526086 4,5
   114 4.862980000  10.3.3.181   10.3.3.65    TCP        68 49658 > iec-104 [ACK] Seq=7 Ack=919 Win=17496 Len=0 TSval=1054923 TSecr
   147 4.881113000  10.3.3.181   10.3.3.65    104asdu   986 <-S(8)  4,5<-3 <TypeId=0> Back IOA=526086 4,5<-3 <TypeId=0> Back IOA=52
   173 4.896782000  10.3.3.65    10.3.3.181   104asdu   986 ->S(8)  4,5->3 <TypeId=0> Back IOA=526086 4,5->3 <TypeId=0> Back IOA=52
   195 4.909237000  10.3.3.181   10.3.3.65    104apci    80 <-S(16)  <-S(24)
   212 4.916790000  10.3.3.65    10.3.3.181   104asdu   980 4,5->3 <TypeId=0> Back IOA=526086 4,5->3 <TypeId=0> Back IOA=526086 4,5
   214 4.917364000  10.3.3.181   10.3.3.65    104asdu   980 4,5<-3 <TypeId=0> Back IOA=526086 4,5<-3 <TypeId=0> Back IOA=526086 4,5
   240 4.927346000  10.3.3.65    10.3.3.181   104apci    80 ->S(16) ->S(24)
   244 4.927859000  10.3.3.181   10.3.3.65    104apci    74 <-S(32)

▷ Linux cooked capture
▷ Internet Protocol Version 4, Src: 10.3.3.181 (10.3.3.181), Dst: 10.3.3.65 (10.3.3.65)
▽ Transmission Control Protocol, Src Port: 49658 (49658), Dst Port: iec-104 (2404), Seq: 1, Ack: 1, Len: 6
    Source port: 49658 (49658)
    Destination port: iec-104 (2404)
    [Stream index: 4]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 7    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header length: 32 bytes
  ▷ Flags: 0x018 (PSH, ACK)
    Window size value: 3650
    [Calculated window size: 14600]
    [Window size scaling factor: 4]
  ▷ Checksum: 0xd3dd [validation disabled]
  ▷ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▷ [SEQ/ACK analysis]
    [PDU Size: 6]
▽ IEC 60870-5-104-Apci: <-U(STARTDT act)
    ApduLen: 4
    .... ..11 = ApciType: U (0x03)
    0000 01.. = ApciUType: STARTDT act (0x01)
```

Figure E-1 – IEC104 packet analysis (Master IP: 10.3.3.181; Slave IP: 10.3.3.65)

The selected frame (frame nº 66) corresponds to a 104apci protocol packet, containing a U frame used as a start-stop mechanism for information flow (cf. Annex B). The frame nº 80, corresponds to the response sent to the Master for this very same packet. Following, a 104asdu packet (frame nº 112) is sent to the Master, in which an I frame has its first field incremented by 1, maintaining the second field with the value 0. The following response packet (frame nº 147) has its second field of the I frame beginning with 12, incrementing the first field from 0 to 11 (cf. Figure 8-11).



```
▽ IEC 60870-5-104-Apci: ->I(0,0)
    ApduLen: 74
    .... ..00 = ApciType: I (0x00)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(1,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(2,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(3,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(4,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(5,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(6,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(7,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(8,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(9,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(10,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: ->I(11,0)
▷ IEC 60870-5-104-Asdu: 4,5->3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
```

```
▽ IEC 60870-5-104-Apci: <-S(8)
    ApduLen: 4
    .... ..01 = ApciType: S (0x01)
▷ IEC 60870-5-104-Apci: <-I(0,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(1,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(2,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(3,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(4,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(5,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(6,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(7,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(8,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(9,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(10,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
▷ IEC 60870-5-104-Apci: <-I(11,12)
▷ IEC 60870-5-104-Asdu: 4,5<-3 <TypeId=0> Back IOA=526086 '<Unknown TypeId>'
```

Figure E-2 – Frames nº 112 (left) and nº 147 (right)

The Type identification (TypeId=0) is not used since the range goes from 1 to 127 – it is actually used for standard definitions of the IEC101 protocol; the range [128, 135] is used for message routing; and [136, 255] for special use (private – not defined in the standard).

ANNEX F – PYMODBUS: PACKET DECODING

With regard to the execution of the parsing script (message-parser.py) from Pymodbus, following is presented the output when decoding the message referent to the Write Single Register operation. The output also presents information for both server and client decoder. The information to decode (000100000006010600120001) relates to the encoded format of the message. After that, a set of hexadecimal values are assigned to each field, for both client and server decoders.

Table F-1 – Pymodbus decoding script output

Message-parser output
```
================================================================================

Decoding Message 0001000000060106001200 01

================================================================================

ServerDecoder

--------------------------------------------------------------------------------

name = WriteSingleRegisterRequest

protocol_id = 0x0

unit_id = 0x1

value = 0x1

skip_encode = 0x0

address = 0x12

check = 0x0

transaction_id = 0x1

documentation = This function code is used to write a single holding register in a remote device. The Request PDU specifies the address of the register to be written. Registers are addressed starting at zero. Therefore register numbered 1 is addressed as 0.


ClientDecoder

--------------------------------------------------------------------------------

name = WriteSingleRegisterResponse

protocol_id = 0x0

unit_id = 0x1

value = 0x1

skip_encode = 0x0

address = 0x12

check = 0x0
```

transaction_id = 0x1

documentation = The normal response is an echo of the request, returned after the register contents have been written.

Annex G – Contribution for D3.4

The contribution relates to section 3.1.1.3 of deliverable D3.4 where the Shadow RTU concept is described. The deliverable D3.4 is on the DVD delivered with the thesis document.