



FP7-SEC-2011-1 Project 285647

Cybersecurity on SCADA: risk prediction, analysis and reaction tools for Critical Infrastructures

D5.4 – CockpitCI System Factory Trials Report

General information	
Submission date	22/12/2014
Dissemination Level	Public
State	Final Version
Work package	WP5000
Task	Task 5006
Delivery Date	31/08/2014

Editors

Name	Organization
Pietro Ricci, Antonio Graziano, Federico De Padova	Selex-ES

Authors

Name	Organisation
Pietro Ricci, Antonio Graziano, Federico De Padova	Selex-ES
Stefano Panzieri	Roma 3
Tjago Cruz	FCTUC

Reviewers

Name	Organisation	Approval Date
Leonid Lev	IEC	
Serguei Iassinowski	Multitel	

Disclaimer: All entities with access to this document in its present form, will not at any time or in any way, either directly or indirectly, use for personal benefit or divulge, disclose, or communicate any proprietary information hereby included, without prior consent of its intellectual property owners. This document must be protected and be treated as strictly confidential until further notice.

Table of contents

Executive summary	7
1 Introduction	8
1.1 Document Structure	9
1.2 Glossary	9
1.3 Acronym and symbols	10
2 CockpitCI system test overview	12
2.1 CockpitCI scope and objectives of system tests	12
2.2 Entities, protocols and interfaces involved in the tests	17
2.3 Features of SMGW deployed in factory unit tests	21
2.4 Setup of the SMGW	23
2.4.1 Set up of the SMGW applications	25
2.4.2 Deployment of applications over SMGW	28
2.5 Security set up on SMGW	30
2.5.1 Secure Socket Layer set up	31
2.5.2 Firewall set up	34
2.6 Security validation for SMGW	35
2.6.1 Scan Report	39
2.6.1.1 Scan Report for vulnerability assessment	39
2.6.1.2 Scan report for penetration tests	41
3 SMGW Factory unit tests	44
3.1 SMGW Factory tests tools	44
3.2 Communication test with DL, SCADA Adaptor and IRP	44
3.2.1 Tests on Detection Layer application running in SMGW	45
3.2.2 Tests on SCADA Adaptor application running in SMGW	51
3.2.3 Tests on IRP application running in SMGW	56
3.3 Test SMGW-SMGW	65
4 System integration test	71
4.1 Configuration of main components	72
4.1.1 DL configuration	72
4.1.2 SMGW configuration	74
4.1.3 IRP configuration	75
4.2 Cyber attack #1	78
4.2.1 Network FIN SCAN Source: 0 Destination: 172.27.21.43	78
4.2.2 DL operation	79
4.2.3 SMGW operation	80
4.2.4 IRP operation	82
4.3 Cyber attack #2	85
4.3.1 Network SYN Flood Source: 0 Destination: 172.27.21.38/ 172.27.21.43	85
4.3.2 DL operation	86
4.3.3 SMGW operation	90
4.3.4 IRP operation	92
4.4 Cyber-attack #3 spp_arp spoof:	95
4.4.1 ARP Cache Overwrite Attack; Source: 0 Destination: 172.27.21.43	95
4.4.2 DL operation	96
4.4.3 SMGW operation	99
4.4.4 IRP operation	100
4.5 Physical Fault #1	103
4.5.1 Door Open Source: 0 Destination: 172.27.21.37	103
4.6 Cyber-Physical Fault	106
4.6.1 Mechanical fault on ML element of Optical ring node	106

4.6.2	spp_arp spoof: ARP Cache Overwrite Attack Source: 0 Destination: 172.27.21.43.....	109
4.6.3	Network SYN Flood Source: 0 Destination: 172.27.21.44.....	111
4.7	Cyber-Physical Fault #1	114
4.7.1	Mechanical fault on ML element of Optical ring node.....	114
4.7.2	spp_arp spoof: ARP Cache Overwrite Attack Source: 0 Destination: 172.27.21.43.....	117
4.7.3	Network SYN Flood Source: 0 Destination: 172.27.21.44.....	119
5	Conclusions	122
6	References.....	126
7	Appendix A – SMGW management panels.....	127
8	Appendix B – SMGW Hardware specs and delivery details	129
9	Appendix C - Installation of OpenVAS using Kali Linux.....	131

List of figures

Figure 1 – Test levels performed for CockpitCI	12
Figure 2 – Test approach overview	13
Figure 3 - Entities and interfaces in the tests.....	18
Figure 4 - SMGW architecture.....	19
Figure 5 - Metadata Publisher Service Dashboard	23
Figure 6 - Metadata publisher test panel	23
Figure 7 - Subscribers register management page.....	23
Figure 8 - SMGW virtual machine start options	24
Figure 9 - SMGW login page.....	24
Figure 10 – SMGW webapps directory.....	26
Figure 11 - Tomcat home page	27
Figure 12 - Tomcat authentication page.....	27
Figure 13 - Tomcat application manager.....	28
Figure 14 – Deployment panel	28
Figure 15 - Tomcat Web Application Manager: WAR file deployment	29
Figure 16 – Management of an application	29
Figure 17 - OpenVAS Architecture	37
Figure 18 – Comparison between two runs	39
Figure 19 - Scan Report.....	40
Figure 20 - Report on available services	41
Figure 21 - Report on retrieving of data of SMGW	41
Figure 22 - Report on activities	41
Figure 23 - Log of exploiting.....	42
Figure 24 – Brute-force by passing the SMGW credentials	42
Figure 25 - Bruteforce attack by passing dummy credentials	43
Figure 26 - Report on the SMGW security check after the test.....	43
Figure 27 - Firefox Poster Add-on	44
Figure 28 – DL application test scheme	46
Figure 29 - Poster Application	47
Figure 30 - Set URL on Poster application	47
Figure 31 – Select the IDMEF file.....	47
Figure 32 – Set the content type	48
Figure 33 – Apply the POST method.....	48
Figure 34 – Response from the server to POST.....	48
Figure 35 – Error in POST	49
Figure 36 – Manual settings of the parameters Poster	49
Figure 37 – POST of message using manual input of parameters	50
Figure 38 - Contents of localldata table.....	50

Figure 39 – SCADA Adaptor application test scheme	52
Figure 40 - Set URL on Poster application	53
Figure 41 – Select the IDMEF file.....	53
Figure 42 – Set the content type	53
Figure 43 – Apply the POST method.....	53
Figure 44 – Response from the server to POST.....	54
Figure 45 – Error in POST	54
Figure 46 – Manual settings of the parameters Poster	55
Figure 47 – POST of message using manual input of parameters	55
Figure 48 - Select in cockpitadaptor table	56
Figure 49 – Web service information page.....	57
Figure 50 - Cockpit WSDL endpoint.....	57
Figure 51 - Request to SMGW	62
Figure 52 - Response to the IRP	63
Figure 53 – Test configuration to test the communication between different SMGWs	67
Figure 54 - Certificate acquisition	68
Figure 55 - Certificate details	68
Figure 56 – X.509 Certificate export.....	69
Figure 57 - Sending messages without certificates	69
Figure 58 - Correct message exchange between two SMGWs	70
Figure 59 - Incorrect use of Http protocol in sending messages	70
Figure 60: Detection Layer architecture	72
Figure 61 - University of Coimbra Testbed.....	73
Figure 62 - Dump of local database of the SMGW	74
Figure 63 The power grid represented in the GUI	75
Figure 64 The SCADA network as panel in the GUI.....	76
Figure 65 The telecommunication network in the IRP	77
Figure 66 The Risk Prediction for FISR 1 and FISR 2	77
Figure 67 FIN Scan.....	78
Figure 68 Launching a fast FIN scan via Nmap.....	79
Figure 69 FIN scan network trace	80
Figure 70 IDMEF message for a FIN scan	80
Figure 71 - Dump of the SMGW database in case of Network FIN SCAN	81
Figure 72 Attack is collected from the SMGW and elaborated by Cyber Simulator.....	82
Figure 73 XML SCAN cyber fault is activated on RTU 10 and on R&F2 that is on the path.....	83
Figure 74 Propagation on the SCADA TLC network.....	84
Figure 75 Propagation on the Electric Grid.....	84
Figure 76 Calculated RISK for FISR 1 and FISR 2	85
Figure 77 SYN Flood attack	86
Figure 78 Shadow RTU detecting a SYN flood.....	86
Figure 79 Generating SYN floods using <i>hping</i>	87
Figure 80 IDMEF message for flooding attack against a PLC	89
Figure 81: IDMEF message for flooding attack against the HMI	90
Figure 82 – SMGW database dump in case of attack against the PLC	91
Figure 83 – SMGW database dump in case of attack against the HMI.....	92
Figure 84 SYN FLOOD attack is collected from SMGW and elaborated from Cyber Simulator.....	93
Figure 85 CISIA propagation of the attack.....	94
Figure 86 Propagation on the Electric Grid.....	94
Figure 87 RISK computed for FISR 1 and FISR 2	95
Figure 88 MITM with ARP Poisoning.....	96
Figure 89 Detecting ARP Poisoning	97

Figure 90: Launching the first stage of the MiTM attack – ARP poisoning	97
Figure 91: Network trace captures for the ARP poisoning attempts.....	98
Figure 92: IDMEF message for MiTM attack	98
Figure 93 – Dump of SMGW database in case of ARP poisoning MITM	99
Figure 94 Attack is collected from the SMGW and elaborated from Cyber Simulator	100
Figure 95 Propagation on the SCADA TLC network.....	101
Figure 96 Propagation on the Electric Grid of ARP Poisoning	102
Figure 97 Calculated RISK for FISR 1 and FISR 2	103
Figure 98 A Door alarm is generated in RTU 4.....	104
Figure 99 CISIA propagation on the SCADA TLC network is absent.....	104
Figure 100 The Operative Level of RTU 4 is decreased.....	105
Figure 101 A fault on the element ML is detected	106
Figure 102 The TLC network will have a decreased Operative Level compromising the efficiency of some RTUs	107
Figure 103 Distribution of Operative Levels among RTUs.....	108
Figure 104 The risk of both FISRs is slightly increased	108
Figure 105 ARP poisoning attack on RTU 10.....	109
Figure 106 The Operative Level of RTU 10 is decreasing after a propagation	110
Figure 107 Distribution of OLs and Cyber Faults on the Electric Grid.....	110
Figure 108 The FISRs risks are increasing in a different way	111
Figure 109 Cyber Propagation of both attacks on SCADA TLC network	112
Figure 110 The Operative Levels of all RTUs and the Cyber Faults	112
Figure 111 Only one FISR can be considered possibly safe at this point	113
Figure 112 A fault on the element ML is detected	114
Figure 113 The TLC network will have a decreased Operative Level compromising the efficiency of some RTUs	115
Figure 114 Distribution of Operative Levels among RTUs.....	116
Figure 115 The RISK of BOTH FISRs is slightly increased	116
Figure 116 ARP poisoning attack on RTU 10.....	117
Figure 117 The Operative Level of RTU 10 is decreasing after a propagation	118
Figure 118 Distribution of OLs and Cyber Faults on the Electric Grid.....	118
Figure 119 The FISRs risks are increasing in a different way	119
Figure 120 Cyber Propagation of both attacks on SCADA TLC network	120
Figure 121 The Operative Levels of all RTUs and the Cyber Faults	120
Figure 122 Only one FISR can be considered possibly safe at this point	121
Figure 123 - WSO2 Application Server Management Console	127
Figure 124 – WSO2 Application Server Key store management	128
Figure 125 – WSO2 Application Server security management panel	128
Figure 126 – Folder on FTP site.....	129
Figure 127 - Virtual machine directory.....	130
Figure 128 - SMGW repository using OVF standard	130
Figure 129 – Kali Linux start page.....	131
Figure 130 - OpenVAS installation on Kali Linux	132
Figure 131 - Greenbone start page	133

List of tables

Table 1 – Requirements list.....	17
Table 2 - Test summary	125

Executive summary

This document summarizes the factory system tests [1] performed on CockpitCI and is divided in two sections, the first one describes the factory unit tests on each component of the CockpitCI tool and the second one addresses the system integration test. The CockpitCI tool is composed of several components such as DL, SCADA Adaptor, IRP and SMGW that have been developed by different teams. The factory unit tests on the first three components have been described in deliverables D3.5 and D4.3.

The deliverable is organized as follows. The first part is mainly devoted to describe the factory trials on the SMGW and its interaction with the other components. In fact, in the service scenario foreseen for the CockpitCI tool, the SMGW is the central element that receives and dispatches messages towards all the other components. The factory tests have been focused on the verification of the correctness of the operations performed in the message exchange and on the security aspects.

The second part of the deliverable describes the test activities on the CockpitCI tool before the delivery on the Hybrid Test Bed for final validation. It includes the description of the end-to-end integration tests across all entities involved in the message exchange and demonstrates the behavior of the system in case of cyber attacks.

The factory integration tests have been performed over the virtual LAN comprising DL, SCADA Adaptor, SMGW and IRP across the labs located in Coimbra and University of Roma TRE.

In all cases the tests on the CockpitCI system have demonstrated the compliance with the requirements.

1 Introduction

The main objective of this deliverable is to describe the system trials performed on the CockpitCI system for the validation of the solution before the final deployment on the HTB. Tests over all components of the system have been carried out in order to check the match between the requirements identified during the design phase with the features available on the final deployment.

The tests have been carried out in several stages, starting from the test on the single subcomponent up to the validation of the integrated system. The description of the factory tests on DL, SCADA Adaptor and IRP is available in the related documents reporting these activities (Deliverable 3.5 [2] and Deliverable 4.3 [3]).

This report focuses, in the first section, on the test on SMGW that have been performed taking into account all requirements foreseen in the design phase and all the interaction with the other components of the CockpitCI tool. The SMGW enables the communication between the different components in CockpitCI and the interfaces of the applications must be able to receive and forward messages with DL, SCADA Adaptor and IRP. These applications have been deployed on a virtual machine running as SMGW and have been tested for the validation.

This target was achieved by verifying the correspondence between the requirements and the behavior of entities involved in communication. During the factory tests DL, SCADA Adaptor and IRP have been emulated by means of a software tool. In the integration tests DL, SCADA Adaptor and IRP applications have exchanged messages using the SMGW as central node. As described in Deliverable D5.3, the service scenario is divided in interdomain and intradomain, the SMGW is the element at the edge of these two domains and the communication among the entities involved in message exchange is based on web service approach. The interSMGW domain is based on a federated model and the message exchange can be performed only using SOAP. In intraSMGW case, the messages can be exchanged both using SOAP and REST web services.

The SMGW interfaces are based on REST and SOAP web services and the POST/GET methods are available on HTTPS using an X.509 certificate.

The factory integration tests have been carried out taking in account the following cyber attacks:

- Scouting: Network FIN SCAN;
- DOS (Denial Of Service): Network SYN Flood;
- MITM (Man In The Middle Attack): spp_arp spoof: ARP Cache Overwrite;

In addition two situation have been described:

- a fault;
- a fault at the same time of a cyber attack.

For each attack the input and output of each component have been provided together with the risk values of different reconfigurations of the network in response of a fault.

1.1 Document Structure

- Section 2 provides the CockpitCI system test overview.
- Section 3 illustrates the SMGW Factory tests according the following scheme:
 - Detection Layer test
 - SCADA Adaptor test
 - SMN-IRP interface test
 - SMGW-SMGW
- Section 4 provides the report on Integration test.
- Section 5 provides the conclusions.

1.2 Glossary

Terminology	Description
Adverse event	Any event which may cause a degradation of the capability of the CI to provide its services.
Critical Infrastructure	A national or transnational asset which is deemed essential for the maintenance of vital societal functions. It could be in the field of health, safety, security, economic or social well-being of people.
Cyber attack	A global intrusion plan that enables the intruder to achieve his malicious objective.
Industrial control system	Industrial control system is a general term that encompasses several types of control systems used in the industrial sector, including supervisory control and data acquisition (SCADA) systems used to control Critical Infrastructures.
Potential cyber attack	Simple and/or composite security event which represent <i>symptoms</i> of possible attacks
Risk	A combination of the probability/likelihood for an accident to occur and the resulting negative consequences if the accident occurs.
SCADA operator	Personnel in charge of managing a CI in order to deliver the requested services.
SCADA system	The set of elements which perform supervision and control of an industrial process or a Critical Infrastructure, including the proprietary communication network which links the field devices to the control centre.
Security alarm	Alarm released in presence of a potential cyber attack with variable degree of confidence
Security event	Event that might be potentially relevant, from a cyber security point of view
Security operator/staff	Personnel in charge of the security of the CI.

System of Systems	An interdependent network of Critical Infrastructures
Service	It is what an infrastructure produces and makes available to its customers or other infrastructures.

1.3 Acronym and symbols

Acronym or symbols	Explanation
ARP	Address Resolution Protocol
CERT	Computer Emergency Response Team
CI	Critical Infrastructure
CRUTIAL	Critical Utility InfrastructurAL Resilience
DB	Data Base
DL	Detection Layer
DMZ	Demilitarized Zone
DNP	Distributed Network Protocol
DoS	Denial of Service
DNS	Domain Name System
EMS	Energy Management System
ENTSO-E	European Network of Transmission System Operators for Electricity
EU	European Union
FCTUC	Faculdade de Ciências e Tecnologia University of Coimbra
FP	Framework Programme
FR	Functional Requirement
HMI	Human-Machine Interface
HTB	Hybrid Test Bed
I/O	Input/Output
ICS	Industrial Control System
ICT	Information and Communication Technology
IDS	Intrusion Detection System
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IPS	Intrusion Prevention System
IRP	Integrated Risk Predictor
ISO	International Organization for Standardization
LAN	Local Area Network
MiTM	Man in The Middle

MPLS	Multi Protocol Label Switching
N.A.	Not Applicable
NFR	Not Functional Requirement
NIDS	Network Intrusion Detection System
OCSVM	One Class Support Vector Machine
PIDS	Perimeter Intrusion Detection System
PLC	Programmable Logic Controller
QoS	Quality of Service
QA	Quality Assessment
RTU	Remote Terminal unit
SCADA	Supervisory Control and Data Acquisition
SDH	Synchronous Digital Hierarchy
SI	System Integration
SLA	Service Level Agreement
SLS	Service Level Specification
SMGW	Secure Mediation Gateway
SMN	Secure Mediation Network
SONET	Synchronous Optical NETworking
SoS	System of Systems
SW	Software
STM	Synchronous Transport Module
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
UDP	User Datagram Protocol
UR	User Requirement
VPN	Virtual Private Network
WEP	Wired Equivalence Privacy
Wi-Fi	Wireless-Fidelity
WLAN	Wireless Local Area Network
WP	Work Package
WSDL	Web Service Description Language

2 CockpitCI system test overview

This section describes the specific test types that have been performed, the scope and objectives of each test, the assessment policy, the test environments and the testing tools that have been utilized.

2.1 CockpitCI scope and objectives of system tests

The test activities have been addressed to check the correct implementation and operation of the whole system. The CockpitCI system is composed of several components, each of which must operate correctly in order to assure the proper communication process end-to-end.

The testing activities have been carried out at different levels. The first level has been focused on the single component and the second level on integration in order to prove that the system works as an integrated unit. The objective is to ensure that each element of CockpitCI meets the functional requirements as indicated in the design specification and that all components of the system interface each other correctly avoiding gaps in the data flow.

The multi-level approach is useful basically to identify locally missing areas and prevent problems during the final delivery of CockpitCI on HTB. As shown in Figure 1, in CockpitCI the following levels have been foreseen:

- Factory Unit Testing is the level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
- Factory Integration Testing is a level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

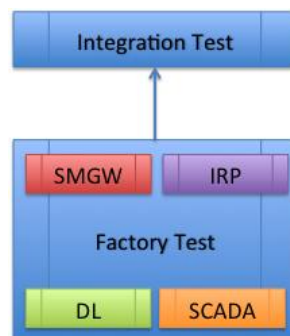


Figure 1 – Test levels performed for CockpitCI

Figure 2 shows the overview of entities involved during the factory unit and integration tests. The factory unit tests have been performed in stand alone configuration and later the system integration tests have been carried out before the final delivery on the HTB.

The factory unit tests have been carried out separately on SMGW, DL, SCADA Adaptor and IRP. On the last three entities, dedicated reports describing the results of the test activities are available (Deliverable 3.5 [2] and Deliverable 4.3 [3]).

On the factory unit tests, this document describes specifically the activities that have been carried out on SMGW in order to check if the deployed applications run as expected also assuming that the tests on DL, SCADA Adaptor and IRP have been already performed and reported. The left side of Figure 2 shows the entities involved in the factory unit tests (in blue) and the related interfaces (in grey). The right part instead shows the entities involved in the second phase when DL, SCADA Adaptor, IRP and SMGW run together to prove that the integrated system works as designed. Also in this case the right side of Figure 2 shows the entities involved in the factory integration tests (in blue) and the related interfaces (in grey).

In the factory unit tests described in this document, the SMGW is involved in 2 service scenarios:

- a single SMGW exchanging messages with DL, SCADA Adaptor and IRP;
- two SMGWs exchanging messages between them.

In the first situation the entities exchanging messages with SMGW are emulated by means of a software tool. In the second one, two SMGWs running over virtual machines have been deployed on a local network.

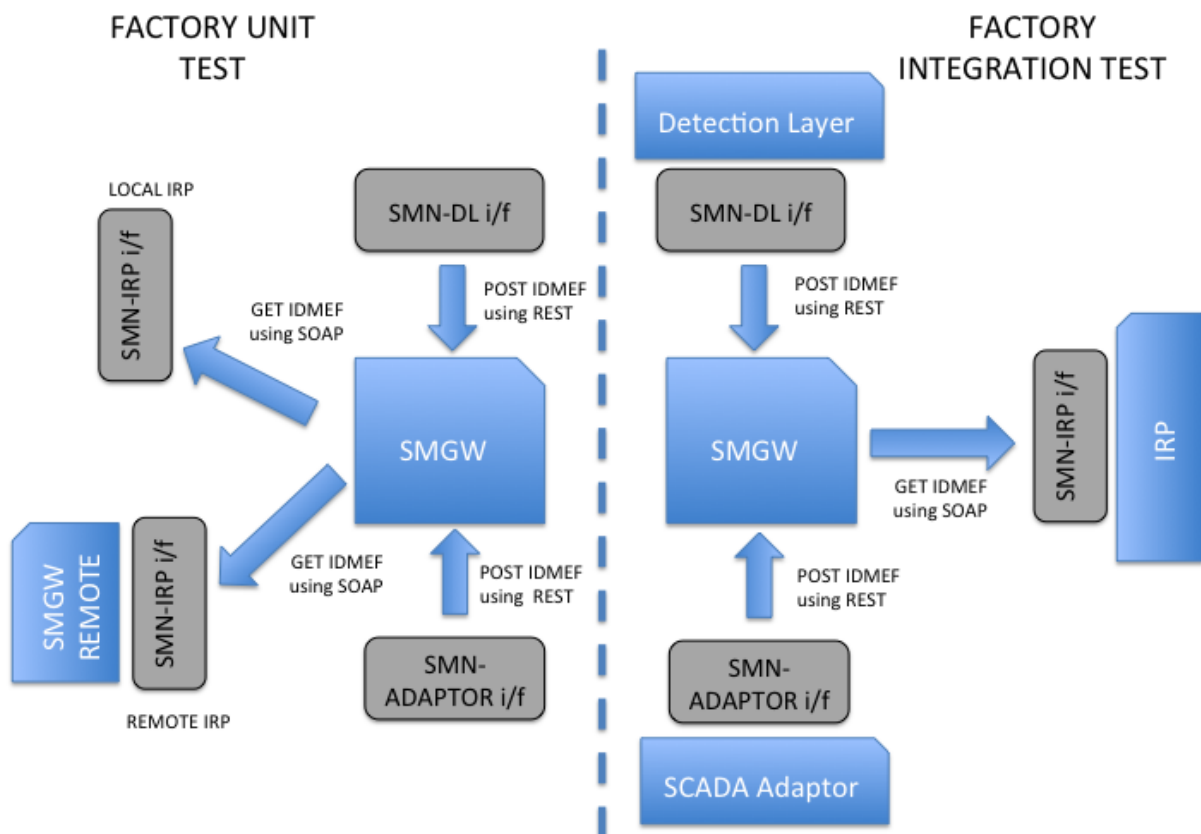


Figure 2 – Test approach overview

Testing the functional requirements of an application aims at verifying that an application's features and operational behaviour correspond to their specifications. There are two basic strategies that are used to carry out tests: black box and white box. The white box approach is based on the full knowledge of the software. This approach, in general, is pursued by developers because they know the code. Examples of this approach are the techniques

based on the coverage of the path, decisions, conditions, etc. The black box approach is what is generally used by the tester and is based on the assumption of not knowing how the functionalities are implemented. The only interest is in what it does, so the view is (almost) the same as an end-user. In CockpitCI the black box approach has been selected in order to identify failures of different types such as:

- failures in the executions of servlets (programs running within a Web server);
- incorrect storage of data being stored into a database;
- failures due to the existence of incorrect URI references between web services;
- defects in generated client messages.

In addition to applications testing, the security validation has been performed in order to verify the effectiveness of the overall application's defences against undesired access of unauthorised users, its capability to preserve system resources from improper use and granting the access to services and resources for authorised users. In CockpitCI the environment where the applications are running is deployed to provide protection mechanisms and to avoid or reduce damage due to intrusions. A vulnerability assessment is described in order to check the match also with these requirements.

The factory integration tests have been focused to check and validate the correct efficiency of the implementation of the whole system. The tests cover some operative scenarios where all the components of the CockpitCI tool are used end-to-end. End-to-end testing refers to user-level testing of component-based systems and it verifies that the integrated components work correctly as part of the overall system by adopting a black-box approach. In the integration test environment, the components are deployed as follow:

- the DL and SCADA Adaptor deployed in the laboratory of UC, Coimbra (Portugal);
- the SMGW deployed in the laboratory of Roma TRE, Rome (Italy);
- the IRP is deployed in the laboratory of Roma TRE, Rome (Italy).

The two laboratories are connected via Internet connections; therefore the three main components of the CockpitCI are on the same virtual LAN.

The factory integration tests have been carried out by simulating some security events that have been detected by the DL, which generates an IDMEF message which is propagated through the SMGW towards the IRP; the IRP receives the IDMEF message, performs a risk prediction cycle and produces outputs as result of its elaboration. With the same mechanism, the IRP is able to receive messages from SCADA Adaptor. Several attacks have been simulated in order to validate and release the test environment. For each different attack, input and output of each of the main components of the CockpitCI tool have been described.

According to Deliverable D5.3, where the Secure Mediation Network requirements have been specified, a test list has been arranged. Table 1 shows the defined functional system requirements described in the previous deliverables. The third column provides details on the implementation of the CockpitCI tool. The fourth column describes the required feature that must be fulfilled for each specific requirement.

Requirement id	Short description	Implementation	Feature that must be fulfilled by the test
SMN_1.	The Secure Mediation Network <i>shall</i> provide near real-time secure, reliable and available data exchange between the Detection Layer and the IRP, as well as among different CIs	A SOA-based architecture for data exchange has been adopted to fulfil the requirement.	Delivery of IDMEF file end-to-end from DL to IRP
SMN_2	The Secure Mediation Network <i>shall</i> interface with the detection layer	A proper DL interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired in SMGW database from DL entity by means of a POST method.
SMN_3	The Secure Mediation Network <i>shall</i> interface with the prediction tool	A proper IRP interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired from IRP by means of a GET method
SMN_4	The Secure Mediation Network <i>shall</i> interface with external peer Secure Mediation Networks through an Internet connection	A proper interface has been foreseen in the architecture of the SMN.	A SOAP message should be sent between SMGWs using a publish / subscribe approach.
SMN_5	The Secure Mediation Network <i>shall</i> support non real-time exchange of other kinds of information among CIs	A message broker with separate queue for each topic, each with its own priority, has been introduced in the architecture for InterSMGW communications.	The messages can be consumed by remote SMGW by using a publish subscribe mechanism. The remote invocation can consume messages either by selecting all messages in the database or requesting all not already accessed.
SMN_6	The Secure Mediation Network <i>shall</i> acquire CI independent metadata from SCADA adaptors	A proper SCADA Adaptor interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired in SMGW database from SCADA Adaptor by means of a POST method.
SMN_7	The Secure Mediation Network <i>shall</i> store information obtained by all interfaced components in a dedicated database	Proper DB modules of the SMGW will manage exchange and storage of metadata.	Data arriving from DL and SCADA adaptor should be collected in dedicated databases.
SMN_8	A specific framework shall be included in the Secure Mediation Network in order to allow local CockpitCI components and external SMGWs to retrieve metadata useful for their purposes	A message broker implementing a topic oriented, publish-subscribe mechanism has been adopted. Moreover, Web services implementing the request-response scheme have been introduced for metadata retrieval.	A metadata publisher should assure the capability to retrieve the metadata.
SMN_9	The Secure Mediation Network <i>shall</i> perform information discovery at peer SMGWs to retrieve state information of interdependent CIs	SMGW-SMGW communication will be based on publish/subscribe paradigm, hence supporting the standard procedures such as	In message exchange the SMGW should check the list of subscribers and of subscribed services

		service publishing, discovery, subscription, etc.	
SMN_10	All ingoing and outgoing connections of the Secure Mediation Network <i>shall</i> be secure connections	Confidentiality, authenticity and integrity of ingoing and outgoing flows are provided by various security mechanisms, at both transportation and application level (IPSec, VPN, HTTPS, WS-Security, etc...) as detailed in the whole report.	At application level the security mechanisms should be adopted. The web services should be exposed at least using https.
SMN_11	The Secure Mediation Network <i>shall</i> disclose stored global awareness metadata of the local CI, to authorized subscribers, i.e. other SMGWs	SMN will disclose metadata provided by the IRP (e.g. current and predicted CI status) only to authorized subscribers (e.g. other SMGWs).	The SMGW should filter the notifications only to subscribers.
SMN_12	The Secure Mediation Network <i>shall</i> accept subscriptions from peer SMGWs to be notified when updated metadata is available	SMGW-SMGW communication will be based on publish/subscribe paradigm, hence supporting the standard procedures such as service publishing, discovery, subscription, etc.	The SMGW-SMGW communication should be assured by means of Web Services.
SMN_13	The Secure Mediation Network <i>shall</i> perform client authentication on the basis of client profiles and certificates	The Web Server(s) providing REST and SOAP Web services will be configured in order to accept only requests from clients authenticated by means of X.509 certificates. Moreover, for each service request, requestor authorization is verified at application level.	Usage of X.509 certificates over SOAP and REST exposed methods.
SMN_14	The Secure Mediation Network <i>shall</i> perform security auditing	The SMGW provides a log capability, providing selective recording of all service requests and operations performed by each SMGW.	A security vulnerability scan should be performed.
SMN_15	In order to guarantee a reliable and effective risk prediction, the Secure Mediation Network <i>shall</i> keep synchronized with remote CIs' metadata	According to the publish/subscribe paradigm, whenever new relevant data are available at a remote CI side, all the subscribers to the particular data receive an update.	The notification messages should be sent to all subscribers.
SMN_16	The Secure Mediation Network	A control interface is	Implemented at level of

	<i>shall</i> provide a management interface allowing a certain degree of security and policies configuration	foreseen.	interchange SMGW-SMGW
SMN_17	The Secure Mediation Network <i>should</i> provide the possibility to define who and in which way can access a certain piece of information	The definition and management of proper policies in the SMN will allow to achieve such a result.	The SMGW should filter the remote nodes on the basis of an access list.
SMN_18	The Secure Mediation Network <i>should</i> provide the possibility to define trust relations between different CIs	The definition and management of proper policies in the SMN will allow to achieve such a result.	The SMGW should establish the trust with remote nodes on basis of certificates.
SMN_19	The Secure Mediation Network <i>should</i> enforce different communications protocols/technologies in each particular context	For each different communication, the SMN management functionalities allow the administrator to select the protocol /technology from a list of suitable choices .	Different interfaces should be implemented.
SMN_20	The Secure Mediation Network <i>should</i> enforce Service Level Agreements (SLA) or Service Level Specifications (SLS) between CIs	The definition and management of proper policies in the SMN will allow to achieve such a result.	The management panel of the SMGW should assure the SLA on the basis of predefined proprieties.

Table 1 – Requirements list

2.2 Entities, protocols and interfaces involved in the tests

The system factory tests, unit and integration, involve the following entities each using specific interfaces as shown in Figure 3:

- the DL and SCADA Adaptor using REST;
- the SMGW using REST and SOAP;
- the IRP using SOAP.

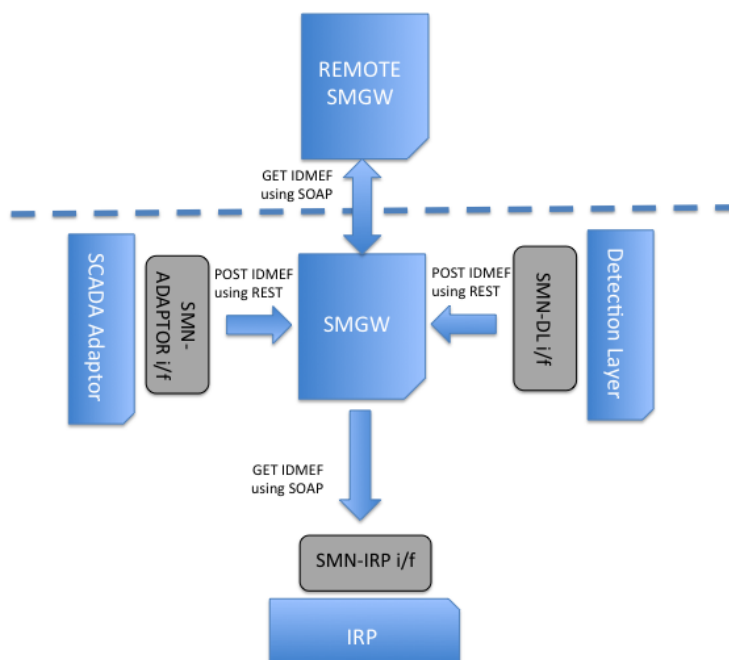


Figure 3 - Entities and interfaces in the tests

During the unit factory tests, only the SMGW has been employed and the following operating conditions have been covered:

- dispatch of a message containing an IDMEF file as payload from DL (emulated) towards the SMGW;
- dispatch of a message containing an IDMEF file as payload from SCADA Adaptor (emulated) towards the SMGW;
- dispatch of a message containing an IDMEF file as payload from SMGW towards a local IRP using a GET request (emulated);
- dispatch of a message containing an IDMEF file as payload from SMGW towards a remote IRP using a GET request (emulated).

During the integration factory tests the following operating conditions were covered:

- dispatch of a message containing an IDMEF file as payload from DL towards the SMGW;
- dispatch of a message containing an IDMEF file as payload from SCADA Adaptor towards the SMGW;
- dispatch of a message containing an IDMEF file as payload from SMGW towards a local IRP.

The SMGW is the central entity in the CockpitCI tool and, according to the architecture scheme shown in Figure 4, the test activities have been focused to verify the correct implementation of all entities, protocols and interfaces foreseen in the design phase.

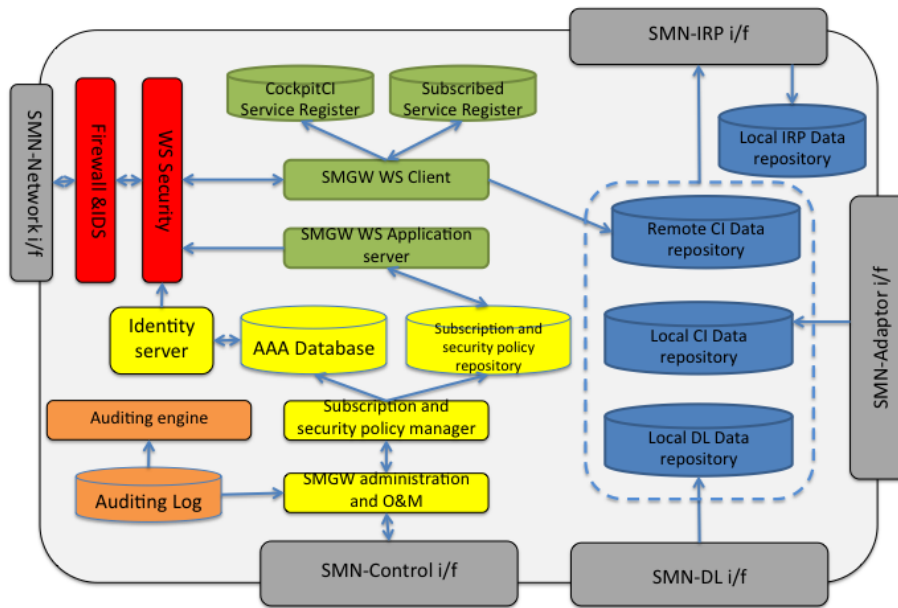


Figure 4 - SMGW architecture

The design of CockpitCI SMGW [1] is based on the simultaneous use of two Web Services types: SOAP and REST. In the development of SMGW, the adopted scheme is addressed to manage the interactions between the components included in CockpitCI and the tests have been designed to verify the correct support both of synchronous request/response and asynchronous publish/subscribe mechanisms. The main functionalities in charge of the SMGW include:

- receiving POST messages within the local CI from DL and SCADA Adaptor;
- processing GET and POST requests within the local CI with the IRP;
- processing GET requests from remote IRPs;
- publishing to authorized subscribers the availability of updated data provided by the IRP;
- managing identities of users in terms of profiles, keys and certificates;
- providing authentication and authorization to external clients through a federated model managing identities and authentication information maintained by another SMGW;
- applying security mechanisms (e.g. encryption/decryption and signature/authentication).

The main protocols, data formats and architectural models supported by the SMGW node in CockpitCI are:

- HTTP/HTTPS protocols running over TCP/IP as basic standard transport protocols for SOAP and REST;
- management of the messages queues in order to support messaging applications and communication patterns with message-delivery guarantees;
- XML format as standard language;
- SOAP support in order to exchange structured XML-based messages among service provider and service consumer;

- REST support in order to include in the CockpitCI the components using RESTful web API.

As a general approach the SMGW must enable the deployment of different security configurations where an element, that provides and consumes information within a CI, can belong both to the same domain identified with a SMGW or can be located under a different administrative domain identified with another SMGW. This introduces basically the identification of two different macro-domains: intraSMGW and interSMGW. The intraSMGW refers to the capabilities enabling the publication and consumption of data and services of a set of components, such as IRP or DL, connected to the same SMGW. Instead the interSMGW refers to the capabilities to exchange data and services across different SMGWs.

The security services provided by the SMGW include the authentication, authorization, confidentiality, and integrity for:

- publication and subscription of services over interSMGW domains;
- communications among entities within a local Domain (Risk Predictor, SCADA, etc.) where the front-end towards federated entities is the SMGW.

Currently SOAP is considered as the main standard protocol for web services with WS-* extensions. WS-Security applies security to Web services assuring integrity, non-repudiation and confidentiality. Nevertheless the performance degradations due to heavy XML processing and the complexities associated with the usage of various WS-* specifications are two of the most common disadvantages of the SOAP messaging model. Because of these concerns, in CockpitCI the REST model has been employed as alternative approach to SOAP. RESTful web services can be considered as a lightweight alternative to the bulky and complex SOAP based web service standards. In RESTful web services, the emphasis is on point-to-point communication over HTTP/HTTPS.

In the SMGW architecture, REST and SOAP web services have been used over different interfaces for intraSMGW communication in the following situations:

- SCADA Adaptor and DL interfaces by means of REST;
- IRP interfaces by means of SOAP.

The REST services provide a lighter weight alternative than SOAP but, at the same time, they do not provide security services, so the SMGW has been enriched with additional security features in order to be equipped with the protection mechanisms. In the final test environment the use of REST services is restricted by the security policies at gateway side (using SSL with certificates and firewall rules) so the REST can be exchanged only within the intraSMGW domain. By means of the POST method the data can be pushed from DL and SCADA Adaptor towards the SMGW. The sending of messages towards the IRPs have been developed using GET methods in which the IRP can perform a request of data to the SMGW. Locally also the IRP can push data towards the SMGW by using a POST method as DL and SCADA Adaptor.

Functional test comprises the conformance of the service behaviour against the design. The service is isolated considering it as a black box from a functional test perspective. The testing has been conducted for each service in isolation by manipulating the XML messages and by stressing both positive and negative conditions. The evaluation of return values have been important factor for these tests in order to check the accordance of the outputs with the expected values. Different tests have been carried out to verify the behaviour of the system

both in case of correct input and in case of malformed data entry. The factory unit and integration tests have included:

- testing of data fields;
- testing to verify the format and value of data fields such as date, files, alpha-numeric, numeric and special characters;
- testing to verify the correct transfer of the messages end-to-end.

The final factory integration testing has focused on the interactions between the services and interfaces to form an end-to-end transaction comprising all the entities involved in the data exchange in connection with the SMGW.

2.3 Features of SMGW deployed in factory unit tests

The SMGW is a system deployed in each CI in order to allow reliable and secure communications with other CIs. Each SMGW contains the following software modules:

- interfaces towards DL, SCADA Adaptor and IRP within the intraSMGW domain;
- interfaces towards remote SMGWs belonging to federated CIs through the SMGW Web Services Application Server;
- administration applications (e.g., Administration, Operation & Maintenance, Subscription & Security Policy Management, Test, Simulation) through browser based Human Machine Interfaces (HMI).

The general approach in the development of the SMGW and in the testing phase has been the use of open source software platforms. The open source model has been selected in development of SMGW because it exploits the following items:

- the open source platforms are peer reviewed software which leads to increased reliability as demonstrated by several infrastructures over Internet that is largely composed of open-source programs such as DNS, sendmail, Apache and languages such as HTML or Perl.
- the open source approach enables anyone to inspect the software to check security weaknesses. The continuous and broad peer-review improves security through the identification and elimination of defects that might otherwise be missed. For example the open source Apache Web server is largely adopted as secure alternative to closed source Internet Information servers. The availability of source code also facilitates in-depth security reviews and audits by government customers.
- with open source software the developers needn't deploy a solution from scratch and the software is suitable for inter-body collaboration, rapid prototyping and experimentation.

Taking in account this approach, Linux OS and Java servlet have been selected as basic elements of the SMGW. A servlet container is a software module that extends the capabilities of a server so it can respond to the types of requests foreseen during the requirements specification phase. All the applications have been developed in Java and Apache Tomcat has been selected as servlet container. The Apache framework allows the use of a security model for each web service with a large range of options. According to the architecture of the SMGW described in deliverable D5.3 [2], the deployed server contains a

set of Java applications that run over Tomcat. Also the software for validating the security of the SMGW has been selected among the open source tools.

In order to fulfil the intraSMGW communication requirements, the SMGW implements SOAP and REST interfaces using SSL protocol stack deployed by means of the Tomcat server. The SSL stack has been deployed using a self generated X.509 certificate. The SOAP and REST web services have been developed both using GET and POST methods and each interface exposes these methods according to the design of the SMGW.

For the interSMGW functionalities, the SMGW adopts the open source WSO2 [10] oxygen middleware that provides a browser based interface for the setup of the configuration. The WSO2 component of SMGW is designed to allow the exchange of data among the federation of SMGWs and implements the Metadata Publisher application. The MetadataPublisher is a Web Service that implements the serverside *Information sharing framework* through an Apache Axis 2 web service and it provides the following operations as shown in Figure 5 and Figure 6:

- *publishMetadata*, that publishes the local datasets towards remote SMGWs.
- *getPrediction*, that retrieves available datasets from remote SMGWs in local repository.

The frontend of SMGW towards interdomain uses Apache Rampart of Axis2 as security module. Rampart secures SOAP messages according to specifications in the WS-Security stack then providing authentication, integrity, confidentiality and non - repudiation for Axis2 web services.

Rampart is deployed as a module in the security stack of Axis2 and introduces two handlers:

- "*org.apache.rampart.handler.RampartReceiver*": it intercepts the incoming message, and hands it over to the RampartEngine. RampartEngine is responsible for management of security in the incoming SOAP message by means of WSSecurityEngine module. All security actions such as decryption of the message, validation of the digital signature, validation of the timestamp and authentication of the user are performed inside the Rampart module.
- "*org.apache.rampart.handler.RampartSender*": it intercepts the outgoing message and handles it over to *org.apache.rampart.RampartMessageBuilder* that is responsible for enforcing security of the outgoing SOAP messages.

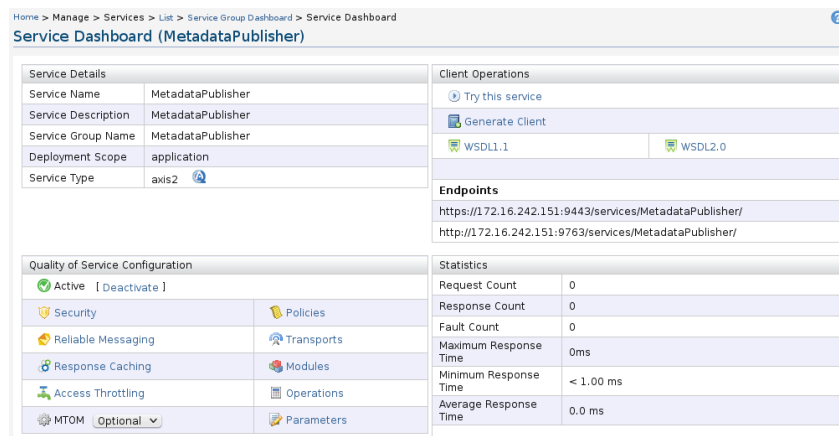


Figure 5 - Metadata Publisher Service Dashboard

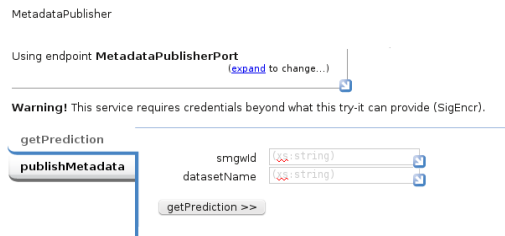


Figure 6 - Metadata publisher test panel

The subscribers to the services have been managed by means of a table in a database. Figure 7 shows an example of the subscribers register list.

Subscribers Register Management									
Id	Name	Cert. Name	Status	Main URL	Main Conn. Status	Alt URL	Alt Conn. Status	Operations	
1	SMGW1	SMGW1	1	https://localhost:9443/services/MetadataPublisher	1	https://localhost:9443/services/MetadataPublisher	1	Edit, Delete	
2	SMGW2	SMGW2	1	https://172.16.242.146:9443/services/MetadataPublisher	1	https://172.16.242.146:9443/services/MetadataPublisher	1	Edit, Delete	
3	SMGW3	SMGW3	1	https://172.16.242.160:9443/services/MetadataPublisher	1	https://172.16.242.160:9443/services/MetadataPublisher	1	Edit, Delete	
4	SMGW4	SMGW4	0	https://172.16.242.130:9443/services/MetadataPublisher	0	https://172.16.242.130:9443/services/MetadataPublisher	1	Edit, Delete	

Figure 7 - Subscribers register management page

2.4 Setup of the SMGW

As explained in detail in the appendix, the SMGW has been distributed over OpenSUSE Linux OS. When the virtual machine starts the user must select the “Cockpit SMGW ver 0 1” option as shown in Figure 8.

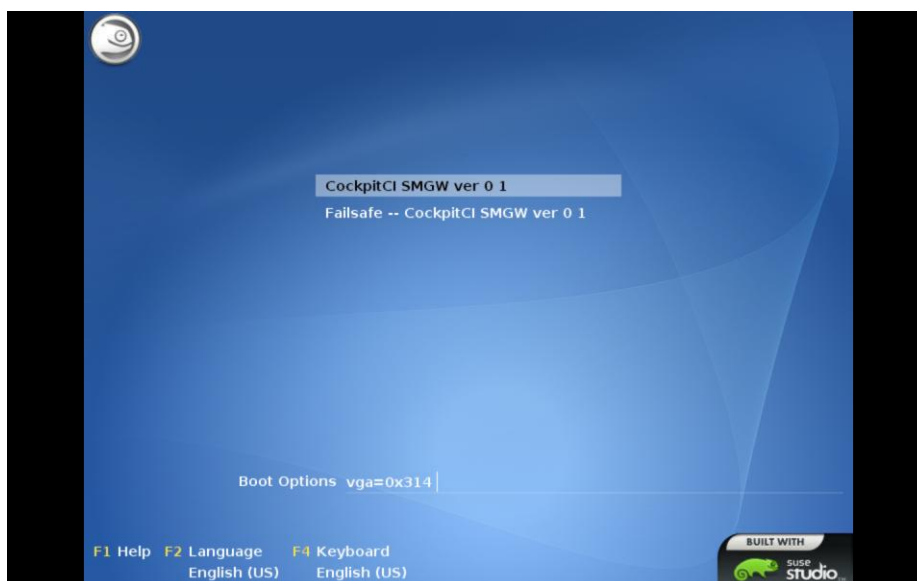


Figure 8 - SMGW virtual machine start options

Afterwards the user has to put the credentials on the login page as in Figure 9 using the following user account/password:

- User: smgwroot
- Pw: cockpitcomm

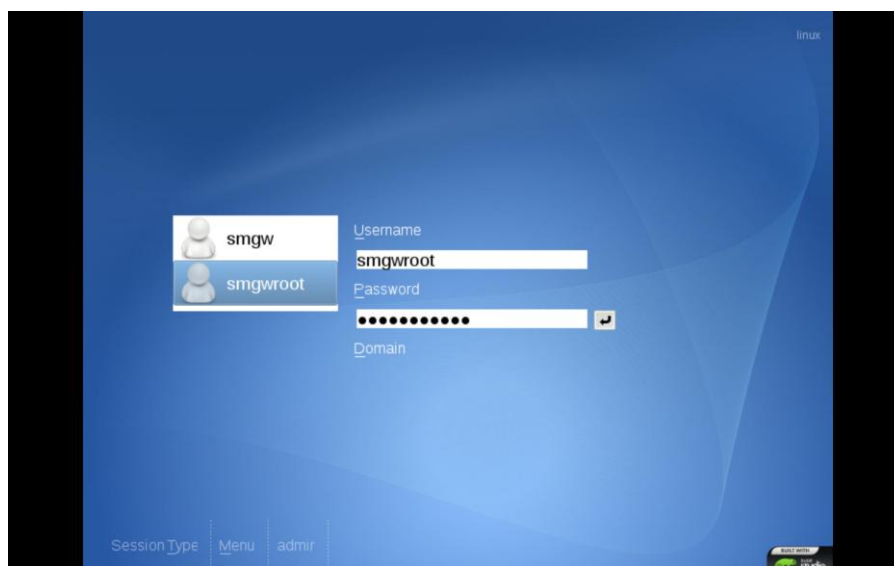


Figure 9 - SMGW login page

Almost all CockpitCI applications are preconfigured in the available distribution, some or new releases of WEB services can be manually deployed on the Tomcat server. All the accounts for TOMCAT, MySql and WSO2 are preconfigured in this version.

The following user accounts/passwords are preconfigured.

- Tomcat: admin (pwd: cockpitcomm)

- Wso2: admin (pwd: admin)

As first stage the installer has to verify that the mysql server is running (both mysql and tomcat should automatically start at the end of the bootstrap) by entering the command:

```
>rcmysql status
```

If the server is not running, the installer must start it by entering the command:

```
>rcmysql start
```

In order to check if the Tomcat server is running the command is:

```
>rctomcat6 status
```

If the server is not running, start tomcat, by entering the command:

```
>rctomcat6 start
```

2.4.1 Set up of the SMGW applications

This section describes the basic setup of the applications of SMGW in the CI network.

Tomcat is the web server/servlet container that can host Java applications developed in the CockpitCI project in order to enable the web services of SMGW towards DL, Scada Adaptor and IRP.

The deployment is the process of installing a web application by means of a WAR file or by a custom web application into the Tomcat server and it may be accomplished in two modes such as:

- static mode when the web application is installed before Tomcat is started;
- dynamic mode by direct manipulation of already deployed web applications or remotely by using the Tomcat Manager web application.

The Tomcat Manager is a web application that can be used interactively to deploy and manage web applications. In Tomcat the context element represents a web application, which runs within a particular virtual host. Each web application is based on a Web Application Archive (WAR) file or on a directory containing the corresponding unpacked contents. The web application used to process each HTTP request is selected by Catalina that is the Tomcat's servlet container implementing the specifications for servlet itself. Catalina is based on matching the longest possible prefix of the Request URI against the context path of each installed application. The servlet mappings is defined in the web application deployment descriptor file that is located at /WEB-INF/web.xml.

In order to provide the deployment of the applications in SMGW the Tomcat's basic elements are the following:

- \$CATALINA_HOME: it is an environment variable that points to the directory where Tomcat is installed. In SMGW it is in [/usr/share/tomcat6/conf](#).
- \$CATALINA_BASE: it is an environment variable that points to the directory of a particular instance of Tomcat if multiple instances of Tomcat are configured. If this

- variable is not set explicitly, then it will be assigned the same value as \$CATALINA_HOME. In SMGW it is in [/usr/share/tomcat6/conf](#).
- Web applications are available under \$CATALINA_HOME/webapps directory. In SMGW it is in [/usr/share/tomcat6/conf/webapps](#) (Figure 10).
 - Document root: it is the top-level directory of a web application, where all the resources (JSP pages, HTML pages, Java classes, images...) that constitute that application are placed.
 - Context path: it is the name which is relative to the server's address (i.e http://localhost) and represents the name of the web application. For example, if the web application is put under \$CATALINA_HOME/webapps/CockpitCIApp directory, it will be accessible by means of the URL http://localhost/MyWeb, and its context path is [/CockpitCIApp](#).
 - WAR: it is the extension of a file that packs a web application directory hierarchy in zip format. WAR stands for Web ARchive. Java web applications are usually packed in WAR files for deployment. WAR files can be created by command line or an IDE like Eclipse.
 - JAR libraries which are shared among web applications are put under \$CATALINA_HOME/lib directory. Application-specific JAR libraries are put under web application's [/WEB-INF/lib directory](#).

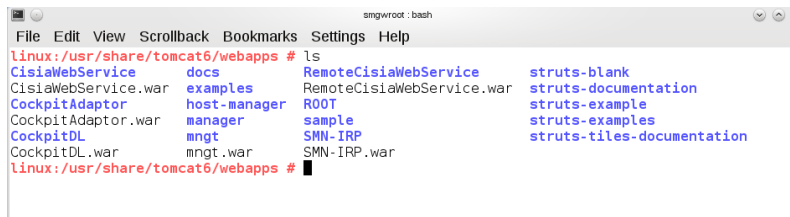


Figure 10 – SMGW webapps directory

As in Figure 11, in CockpitCI the Tomcat Manager panel is accessible by the URL <http://localhost:8080> or in SSL <https://localhost:8443>.

The accounts of the users can be managed by editing the file /usr/share/tomcat6/conf/tomcat-users.xml. In SMGW this file has been edited by replacing the default configuration with the following lines.

```
<tomcat-users>
<user name="admin" password="cockpitcomm" roles="admin"/>
<user name="admin" password="cockpitcomm" roles="manager"/>
</tomcat-users>
```

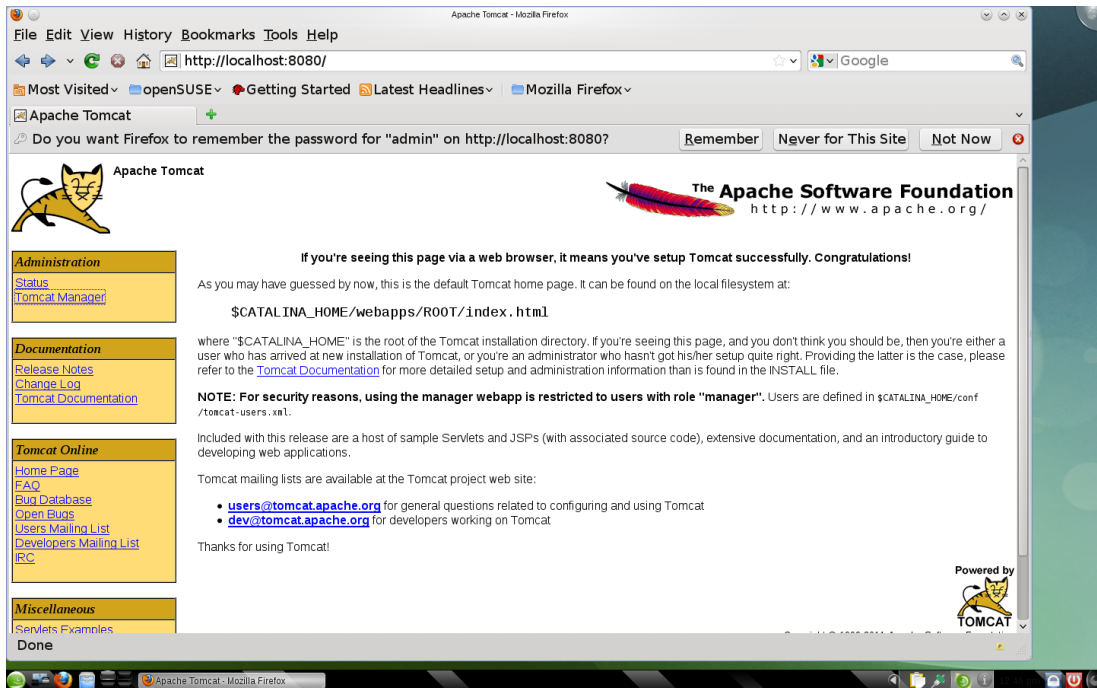


Figure 11 - Tomcat home page

The user must select the Tomcat Manager link from the “Administration menu”. If the user uses the 8443 port the browser notifies that the connection is untrusted and, in this case, the server certificate has to be manually accepted.

The user must provide the User Name and password as specified in Figure 12.

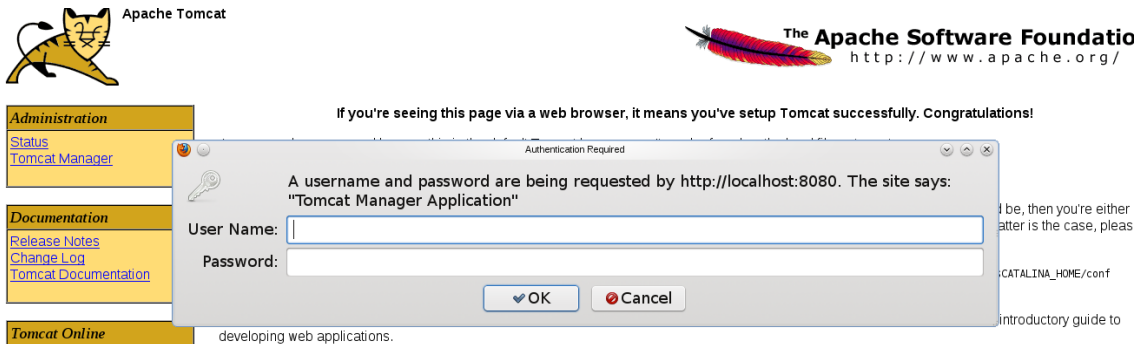


Figure 12 - Tomcat authentication page

User Name: admin

Password: cockpitcomm

Figure 13 shows the Tomcat application manager.

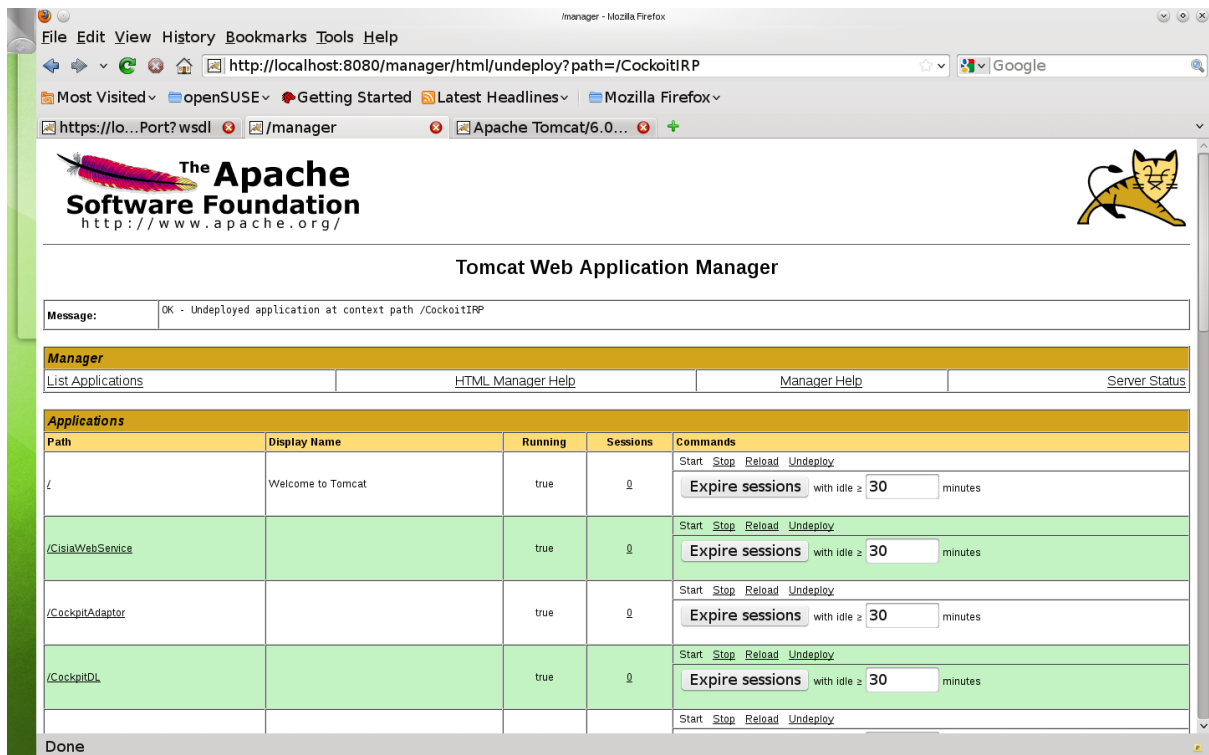


Figure 13 - Tomcat application manager

2.4.2 Deployment of applications over SMGW

Using the deployment by means of the “manager application” the web applications can be installed and removed remotely via the web interface provided by Tomcat. By using the manager application the administrator can:

- view the list of applications deployed on the server and their status;
- start, stop and restart an individual application;
- deploy a new web application either by uploading a WAR file or supplying a directory on the server;
- undeploy an individual application.

After supplying the correct user name and password of the administrator, in order to make the deployment Tomcat makes available the panel shown in Figure 14.

Deploy

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file URL:

WAR or Directory URL:

WAR file to deploy

Select WAR file to upload

Figure 14 – Deployment panel

In the panel there are two options for deploying a web application using the manager:

- Deploy directory or WAR file located on server.
- WAR file to deploy.

The former way is the suitable one if the application’s WAR file or directory resides on the server and if the URL is known. The latter way is easier when a WAR file is picked up and uploaded to the server.

By selecting the “Browse” button the administrator can pick up a WAR file and then select the “Deploy” button. For example by selecting CockpitDL.war file (Figure 15).

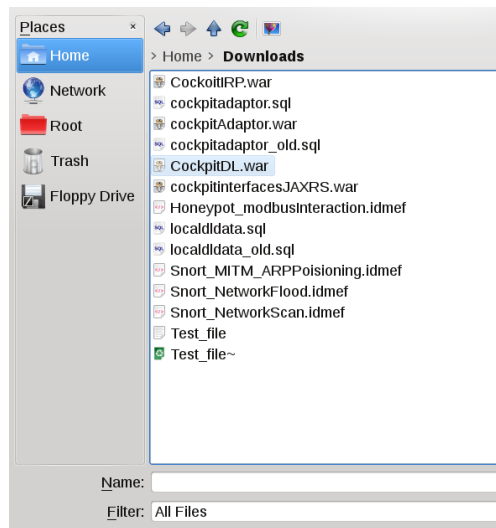


Figure 15 - Tomcat Web Application Manager: WAR file deployment

As soon as the WAR file is uploaded to the server, it is unpacked into \$CATALINA_HOME\webapps directory. The manager adds the newly deployed application to the list of applications (Figure 16):

/CockpitDL		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
------------	--	------	---	--

Figure 16 – Management of an application

The deployed application, CockpitDL in this case, is up and is running without the need of restarting the server. Typically, a web application can be accessed by typing its context path following the server’s IP/domain (including port number if any). For example, the CockpitDL application above can be accessed in web browser by typing:

<http://localhost:8080/CockpitDL>

Alternatively an application is accessible by means of the manager application by clicking on the context path (first column in the list of applications).

2.5 Security set up on SMGW

In case of intraSMGW data exchange, the security in SMGW is performed both by means of SSL protocols and through the firewall settings.

The SSL, or Secure Socket Layer, is a technology which allows web browsers and web servers to communicate over a secured connection. This means that the data sent are encrypted by one side, transmitted, then decrypted by the other side before processing. This is a two-way process, meaning that both the server and the browser encrypt all traffic before sending out data. Another important aspect of the SSL protocol is the authentication so, during the first attempt to communicate with a web server over a secure connection, the server presents the web browser with a set of credentials in the form of a "Certificate" as proof of who the site is and what it claims to be. Tomcat takes advantage of secure sockets when it is running as a stand-alone web server as in case of SMGW. In fact when Tomcat is running primarily as a Servlet/JSP container behind another web server, such as Apache for example, it is usually necessary to configure the primary web server to handle the SSL connections from users.

In order to implement SSL, a web server must have an associated Certificate for each external interface (IP address) that accepts secure connections. The Certificate is a "document" cryptographically signed by its owner and that has to be difficult for anyone else to forge. For example, in sites involved in e-commerce or in business transaction where the authentication of identity is important, a Certificate is typically purchased from a well-known Certificate Authority (CA) such as for example VeriSign. Such certificates can be electronically verified and the Certificate Authority grants for the authenticity of the certificates.

In other cases an administrator may simply want to ensure that the data being transmitted and received by the server are private and cannot be spied by anyone who may be eavesdropping on the connection. Java provides a simple command-line tool, called keytool, allowing an easily creation of a "self-signed" Certificate. Self-signed Certificates are simply user generated Certificates which have not been officially registered with any well-known CA.

Among the digital certificate formats, an X.509 [5] certificate is a digital certificate that uses the widely accepted international X.509 public key infrastructure (PKI) standard to verify that a public key belongs to the user, computer or service identity contained within the certificate. It contains information about the identity to which a certificate is issued and the identity that issued it. Standard information in an X.509 certificate includes:

- Version – which X.509 version applies to the certificate (which indicates what data the certificate must include);
- Serial number – the identity creating the certificate must assign it a serial number that distinguishes it from other certificates;
- Algorithm information – the algorithm used by the issuer to sign the certificate;
- Issuer distinguished name – the name of the entity issuing the certificate (usually a certificate authority);
- Validity period of the certificate – start/end date and time;
- Subject distinguished name – the name of the identity the certificate is issued to;
- Subject public key information – the public key associated with the identity;
- Extensions (optional).

The SSL protocol is designed to be as efficient as securely possible and encryption/decryption is a computationally expensive process from a performance point of view. Any page within an application can be requested over a secure socket by simply prefixing the address with https: instead of http:. Any pages which absolutely require a secure connection should check the protocol type associated with the page request and take the appropriate action if https is not specified. The tests on SMGW have been carried out using both http and https.

2.5.1 Secure Socket Layer set up

Java Keytool is a key and certificate management utility. It allows users to manage their own public/private key pairs and certificates. Java Keytool stores the keys and certificates in what is called a keystore. By default the Java keystore is implemented as a file. It protects private keys with a password. A Keytool keystore contains the private key and any certificates necessary to complete a chain of trust and establish the trustworthiness of the primary certificate.

The easiest way to create X.509 certificates on Linux is the “openssl” command and the auxiliary tools. When the OpenSSL package has been installed usually an auxiliary command CA and/or CA.pl, has been installed, too. In SMGW the openssl command has been used to create the certificates.

For the SMGW a Certification body (ca.crt) with key (ca.key) has been created:

```
# openssl genrsa -des3 -out ca.key 1024
# openssl req -new -x509 -key ca.key -out ca.crt
```

In the creation of the certificate questions are asked, here is a sample of responses:

```
Country Name (2 letter code) [AU]: IT
State or Province Name (full name) [Some-State]: Italy
Locality Name (eg, city) []: Rome
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Selex
Organizational Unit Name (eg, section) []: Cyber
Common Name (eg, YOUR name) []: tester
Email Address []: Antonio.graziano@selex-es.com
With this CA the SMGW signs the certificates.
```

With Keytool the keystore (.keystore) has been generated in Tomcat. The password is: cockpitcomm.

```
# keytool -genkey -alias tomcat -keyalg RSA \ -dname
"CN=localhost,OU=Cyber,O=Selex,L=Rome,ST=Rome,C=IT" \ -keystore .keystore
```

The certificate certreq.pem has been extracted by the keystore

```
# keytool -certreq -keyalg RSA -alias tomcat -file certreq.pem -keystore
.keystore
```

Then the certificate certreq.pem has been signed with the CA

```
# openssl x509 -req -in certreq.pem -out cockpitci.crt -sha1 -CA ca.crt -
```

```
CAkey          ca.key          -CAcreateserial  -days          3650
```

The CA has been imported in the keystore as root:

```
# keytool -import -keystore .keystore -alias root -file ca.crt
```

The signed certificate by the CA has been imported in the keystore:

```
# keytool -import -keystore .keystore -alias tomcat -file cockpitci.crt
```

Tomcat has been configured for secure connections by editing sever.xml file in /usr/share/tomcat6/conf and adding the statement:

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
    maxThreads="150" scheme="https" secure="true"
    keystoreFile="conf/.keystore" keystorePass="cockpitcomm"
    clientAuth="false" sslProtocol="TLS" />
```

Additionally the file web.xml must be edited in /usr/share/tomcat6/conf in order to add the following statement:

```
<security-constraint>
  <!-- Specifies the part of the application to be authenticated -->
  <web-resource-collection>
    <web-resource-name>Secure Task Services</web-resource-name>
    <!-- Denotes which URI patterns needs to be protected. -->
    <url-pattern>/localhost:8080/*</url-pattern>
  <!-- Only POST, PUT & DELETE calls are authenticated. Omitting http-method tag
  altogether will secure all access to the url-pattern above -->
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
    <http-method>DELETE</http-method>
  </web-resource-collection>
  <!-- Specifies which roles defined in tomcat-users.xml have access to the resources. -->
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
```


<!-- Requests are processed without encryption -->

<user-data-constraint>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>

<!-- Denotes the authentication method, which in our case is BASIC -->

<login-config>

<auth-method>BASIC</auth-method>

</login-config>

These implemented features of SMGW meet the following requirements:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_10	All ingoing and outgoing connections of the Secure Mediation Network <i>shall</i> be secure connections	Confidentiality, authenticity and integrity of ingoing and outgoing flows are provided by various security mechanisms, at both transportation and application level (IPSec, VPN, HTTPS, WS-Security, etc...) as detailed in the whole report.	At application level the security mechanisms should be adopted. The web services should be exposed at least using https.
SMN_13	The Secure Mediation Network <i>shall</i> perform client authentication on the basis of client profiles and certificates	The Web Server(s) providing REST and SOAP Web service will be configured in order to accept only requests from clients authenticated by means of X.509 certificates. Moreover, for each service request, requestor authorization is verified at application level.	Usage of X.509 certificates over SOAP and REST exposed methods.
SMN_18	The Secure Mediation Network <i>should</i> provide the possibility to define trust relations between different CIs	The definition and management of proper policies in the SMN will allow to achieve such a result.	The SMGW should establish the trust with remote nodes on basis of certificates.

Not using the certificates, the Web Services exposed by SMGW are not accessible unless the X.509 is installed on the client side.

2.5.2 Firewall set up

Iptables is a command-line firewall utility that uses policy chains to allow or block traffic. When a connection tries to establish itself on your system, iptables looks for a rule in its list to check it. If it doesn't find one, it resorts to the default action.

iptables uses three different chains: input, forward, and output.

- Input – This chain is used to control the behavior for incoming connections. For example, if a user attempts to SSH into your PC/server, iptables will attempt to match the IP address and port to a rule in the input chain.
- Forward – This chain is used for incoming connections that aren't being delivered locally. This mode is used when the router wants to forward the received data to some other destination for example doing NATing or something else that requires forwarding.
- Output – This chain is used for outgoing connections. For example, if an host tries to ping `www.selex-es.com`, iptables checks its output chain to see what the rules are regarding ping and `www.selex-es.com` before making a decision to allow or deny the connection attempt.

The most common used commands are:

- Accept – Allow the connection.
- Drop – Drop the connection and act like it never happened. This is best if the server doesn't want the source to realize that the system exists.
- Reject – Don't allow the connection, but send back an error. This is best if the server doesn't want a particular source to connect and if it wants them to know that the firewall is blocking the remote sources.

When a packet coming from the Internet arrives at the firewall's interface it is then inspected by the rules. If the packet is destined for a protected network, then it is filtered by the rules in the FORWARD chain of the filter table. If the packet is destined for the firewall itself, then it passes through the mangle table of the INPUT chain before being filtered by the rules in the INPUT chain of the filter table. If it successfully passes these tests then it is processed by the target application.

iptables enables protection for common attacks against:

- Syn-flood: In this attack the system is flooded with a series of SYN packets. Each packet causes the system to issue a SYN-ACK responses. Then the system waits for an ACK that follows the SYN+ACK (3 way handshake). Since attacker never sends ACK back, entire system resources get filled by ACK queue. Once the queue is full the system will ignore incoming requests from legitimate users for services (http/mail etc). iptables forces SYN packets check making sure that new incoming tcp connections are SYN packets otherwise dropping them with the rule:
 - `iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP`
- Block Spoofing and bad addresses: Using iptables the server can filter suspicious source address dropping packets coming from the attacker. Spoofing can be classified as:
 - IP spoofing – Disable the source address of authentication, for example rhosts based authentication. Filter RPC based services such as portmap and NFS;

- DNS spoofing.
- Filter incoming ICMP, PING traffic: the server can block all ICMP and PING traffic for outside except for the internal network.

The firewall setup on the SMGW has been carried out by closing all service ports except for the following:

- 22 – for SSH and SFTP remote access;
- 5900 – for UltraVNC;
- 8443 – for TomCat application server over SSL.

2.6 Security validation for SMGW

The definition of vulnerability has evolved over time through the problems of hardware and software configuration towards more general models. The vulnerability management is entrusted to the administrators who must detect and clean up the system from possible threats. On the security side, the most widely used approaches for the security can be grouped into three categories:

- Examination. These are used to evaluate systems, applications, networks, security policies and to discover new procedures regarding the vulnerability (usually are carried out manually).
- Analysis and identification of objectives. These techniques are able to identify systems, doors, services and potential weaknesses; are usually performed using automated tools. They include network discovery, network port and identification services.
- Validation for the vulnerability of the target. These techniques confirm the existence of vulnerabilities, can be performed either manually or in an automated manner. These include password cracking, penetration testing and social engineering.

Since no technique, taken individually, is able to provide a complete picture of the security of a system or network, it is more suitable to adopt a combination of these techniques to ensure higher safety ratings.

A general safety test includes both Vulnerability Assessment and the Penetration Test that can be carried out depending on the needs of the security perimeter of an organization. Tests of external security are held outside the company firewall. This approach offers the possibility to check the security conditions as observed from the outside world. Often these tests begin with the recognition of the registration data and other public information which helps to identify vulnerabilities. These operations follow the identification of external hosts and services listening. These safety tests also focus on the discovery of methods of access, such as wireless access points, modems, and portals to internal servers. This approach is addressed to detect vulnerabilities and to demonstrate the potential damage that this kind of attacker could cause.

The Vulnerability Assessment can be carried out as the first step of each penetration test and is not simply a software tool but an approach to evaluate the degree of system security. This vulnerability assessment consists of a process to evaluate the effectiveness of the security mechanisms and to identify, quantify and assign different levels of priority about any flaws within the system. The main mechanisms, with which it can be conducted, are:

- automatic;
- semi-automatic;
- totally manual.

In automatic case, a dedicated software analyses the target of the evaluation on the basis of an updated database. In the second case the above steps are performed manually, going to track down in the technical literature relevant any known vulnerabilities or trying to discover new ones. The main benefit in the use of automatic tools is repeatability of test routines nevertheless the target system can be exposed to the risk of a large number of false positives, i.e. the reports of vulnerabilities do not correspond to real state. The manual approach allows to avoid the false positives and the problem of updating the data of the vulnerabilities, but does not allow to have consistency and repeatability. Furthermore, if the target of the evaluation is large (a considerable number of systems, a large network) manual evaluation can become very long. One possible solution to this problem is to perform an automatic assessment, followed by manual verification of vulnerabilities discovered by the tool automatically. A useful approach is to use the automatic assessment in order to determine a small area of subsystems potentially at risk, on which the potential risks identified by the automated tool can be evaluated manually. A further division can be identified:

- intrusive;
- nonintrusive.

These two methods differ from each other only for the potential results on the system under analysis. The intrusive instruments, trying to detect the presence of a possible vulnerability, operates directly on the test system causing, usually, a reaction of the system itself. In opposition, the non-intrusive instruments look for a collection of information, covering:

- the target;
- the model of the operating system;
- the services owned;
- the version of the services in analysis;
- the type of port to which the different services are associated.

Intrusive scanners also differ in a second element, i.e. that they can be detected and blocked by a well configured security system, or even worse, may raise alarms, unless the target can be vulnerable and contains applications with serious security flaws. Usually a vulnerability assessment does not include an evaluation of the risks arising from the discovered vulnerabilities (risk assessment) but can include an analysis of the operations useful for the elimination of vulnerabilities.

A vulnerability scan looks for known vulnerabilities in a system, reports potential exposures and it is the first level in the security analysis. A more deeper evaluation can be carried out by means of penetration tests and can be designed to exploit weaknesses. The vulnerability assessment has been carried out on the release of the SMGW to provide basically a vulnerability assessment in order to produce a report of the potential exposures. Then a light penetration test has been carried out using a software tool in order to perform only a first check on the exposed services and to test if the services can be easily discovered and attacked. Deeper penetration tests require the design of specific activities and their plan is still under evaluation.

The mistakes in designing and writing software produce implications for security and it can turn into "vulnerabilities" that hackers can exploit to directly access to sensitive data. The

errors can give the rise to "exposure" able to provide information or means to obtain a pass to the direct access of the resource target. Currently there are several tools for analysing vulnerability, provided both by commercial and open source organizations. These suppliers offer detection systems that monitor networks and systems from attacks. Those tools perform the activities of analysing any threat within the systems and of execution of tests that include different versions of software, in order to obtain accurate data analysis. Among the available tools, OpenVAS has been selected as analyser of vulnerability and it has been applied in CockpitCI.

OpenVAS [7] is a framework, which includes services and tools for the scanning and the management of vulnerability. A vulnerability scanner is an application that allows to scan a target system (IP / HOSTNAME), based on a range of ports and a series of policy. This application is supported, during the scan, by a database of vulnerabilities called Network Vulnerability Tests (NVTs) that is frequently updated and used by the scanner to analyse the critical services. The original name of this application was GNessus, a fork of Nessus, born as a natural development from the open source community. OpenVAS in CockpitCI has been deployed by using Kali Linux release. Figure 17 shows the basic OpenVAS architecture.

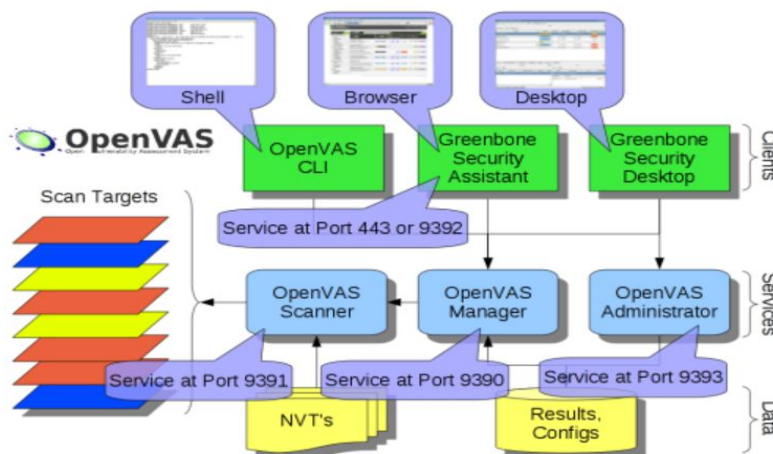


Figure 17 - OpenVAS Architecture

The following points illustrate the architecture of OpenVAS:

- OpenVAS CLI: it is the set of tools that allow, through the shell, the administration of OpenVAS. This layer allows to manage and create reports of various Vulnerability Assessment.
- Greenbone Security Assistant is a web-based tool that can be operated via a graphic interface, through which reports can be created. In addition profiles can be managed for scanning and monitoring the various Vulnerability Assessments.
- Greenbone Security Desktop: as OpenVAS CLI and Greenbone Security Assistant is an application that allows to manage everything through a graphical interface via desktop.
- OpenVAS Manager: it is the most important block of OpenVAS, the manager receives task/information from OpenVAS Administrator and the various administration tools CLI/Web/GUI, then it uses the OpenVAS Scanner that will perform the Vulnerability Assessment. It also includes the component that processes the results of the scans, so it also generates the final report.

- OpenVAS Administrator: it is the component through which you can manage users and upgrades.
- NVTs: is the content of the tests that detect vulnerabilities, which are currently more than 20,000.
- OpenVAS Scanner: it is the component that allows you to perform scan on hostname/IP, port range "from-to" or entire networks (eg 192.168.1.0/28). The scanning can be started at various levels. OpenVAS has four scanning options:
 1. Full and fast: exploits most of NVTs, is optimized by the use of the information previously collected.
 2. Full and fast completed: exploits most of NVTs, including those that can cause a shutdown of the service / remote system. This profile is optimized by the use of the information previously collected.
 3. Full and very deep: exploits most of NVTs, but is slower because it does not exploit the information previously collected.
 4. Full and very deep completed: exploits most of NVTs, including those that can cause a shutdown of the service / remote system. This profile is slower because it uses the information previously collected.
- Results database: is the database in PostgreSQL where OpenVAS collects the report and the configuration.

OpenVAS NVT Feed are signed by the certificate called "OpenVAS: Transfer Integrity" that is used to help the user to verify proper integrity of NVTs after a sync. Besides the OpenVAS NVT Feed, there is also a commercial feed, offered by Greenbone Networks, whose name is Greenbone Security Feed.

For the light penetration test Metasploit Framework has been employed [8]. Metasploit is a tool whose purpose is to allow a tester to quickly write attacks and automate the execution. The tool comprises a library of attacks for the most common (and not) vulnerabilities and an archive of payloads and utility tools ready to use. The Metasploit project was created to provide information on intrusion techniques and to create a functional knowledge base for developers and security professionals. The tools and information are provided for legal security research and testing purposes only.

Metasploit uses different libraries which hold the key to the proper functioning of the framework. These libraries are a collection of pre-defined tasks, operations, and functions that can be utilized by different modules of the framework. The most fundamental components of the framework are:

- Ruby Extension (Rex) library. Some of the components provided by Rex include the wrapper socket subsystem, the implementations of protocol clients and servers, a logging subsystem, utilities for the exploitation and a number of other useful classes.
- MSF Core library which extends Rex. Core is responsible for implementing all of the required interfaces that allow for interacting with exploit modules, sessions, and plugins. This core library is extended by the framework base library which is designed to provide wrapper routines for dealing with the framework core, as well as providing utility classes in order to manage, for example, different output formats.
- Base library is extended by the framework's User Interface that implements support for the different types of user interfaces such as the command console and the web interface.

The implemented tests on SMGW meet the following requirement:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_14	The Secure Mediation Network <i>shall</i> perform security auditing	The SMN provides a log capability, providing selective recording of all service requests and operations performed by each SMGW.	A security vulnerability scan should be performed.

Available Kali Linux distribution with OpenVAS and Metasploit Community Edition for scan and Pro Edition for the force-brute attack was used in testing.

2.6.1 Scan Report

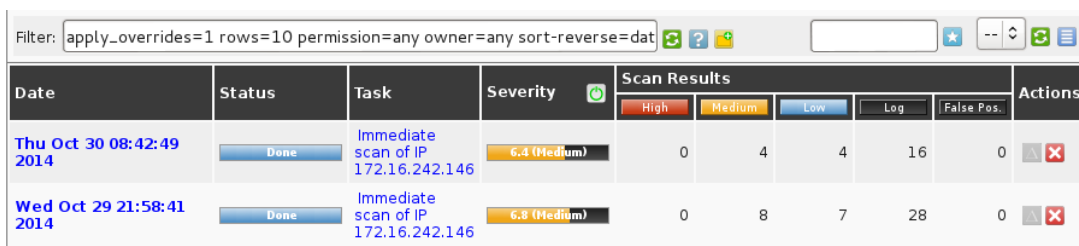
The report summarises the results found and describes for SMGW every issue found using OpenVAS and Metasploit.

2.6.1.1 Scan Report for vulnerability assessment

The report has been produced using OpenVAS that has been configured by taking in account the following options:

- vendor security updates are not trusted.
- overrides are on. When a result has an override, this report uses the threat of the override.
- notes are included in the report.
- the report might not show details of all issues that were found.
- the report only lists hosts that produced issues.
- issues with the threat level "Debug" are not shown.
- issues with the threat level "False Positive" are not shown.

The vulnerability check was performed using, in a first phase, both port 8080 and port 8443 with SSL. Then port 8080 was disabled and only port 8443 was allowed.



Date	Status	Task	Severity	Scan Results					Actions
				High	Medium	Low	Log	False Pos.	
Thu Oct 30 08:42:49 2014	Done	Immediate scan of IP 172.16.242.146	6.4 (Medium)	0	4	4	16	0	
Wed Oct 29 21:58:41 2014	Done	Immediate scan of IP 172.16.242.146	6.8 (Medium)	0	8	7	28	0	

Figure 18 – Comparison between two runs

The results of the main vulnerabilities using only the port 8443 are the following:

- Medium vulnerability on 8443/tcp port:
 - Apache Tomcat 'Transfer-Encoding' Information Disclosure and Denial Of Service Vulnerabilities
 - Solution: Update of the Tomcat Version

- Apache Tomcat 'sort' and 'orderBy' Parameters Cross Site Scripting Vulnerabilities
 - Solution: Software Updates
- Check for SSL Weak Ciphers
 - Solution: Change of configuration in order to disable the weakness
- POODLE SSLv3 Protocol CBC ciphers Information Disclosure Vulnerability
 - Solution: Update of OpenSSL
- Low vulnerabilities on 8443/tcp port:
 - Apache Tomcat Authentication Header Realm Name Information Disclosure Vulnerability
 - Apache Tomcat Security bypass vulnerability
- Low vulnerabilities on 22/tcp port:
 - openssl-server Forced Command Handling Information Disclosure Vulnerability
- General vulnerabilities:
 - SSL Certificate - Self-Signed Certificate Detection

The vulnerabilities are mostly due to the need to upgrade the software, in particular the version of Tomcat. This version was chosen during a project phase where the newest releases were still not available or still out of availability. The selection of the current version is a conservative choice in order to facilitate the stability of the software. The remaining vulnerabilities are related to configuration such as, for example, the set of possible cipher options.

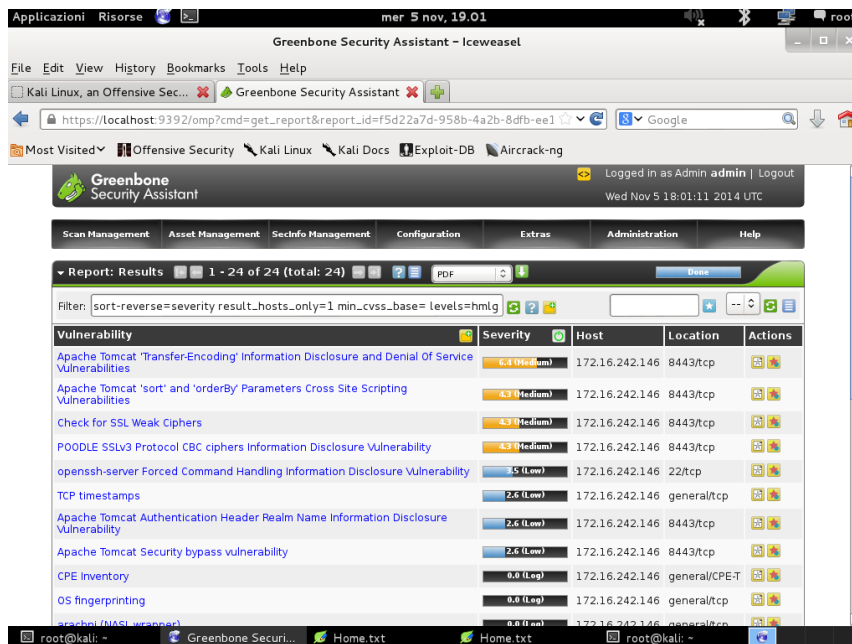


Figure 19 - Scan Report

2.6.1.2 Scan report for penetration tests

Metasploit has provided a scan of the SMGW. It has detected the two available services over the SMGW. The service SSH on port 22 is protected by the default authentication key, instead the service over the port 8443 is protected by means of X.509 certificate.

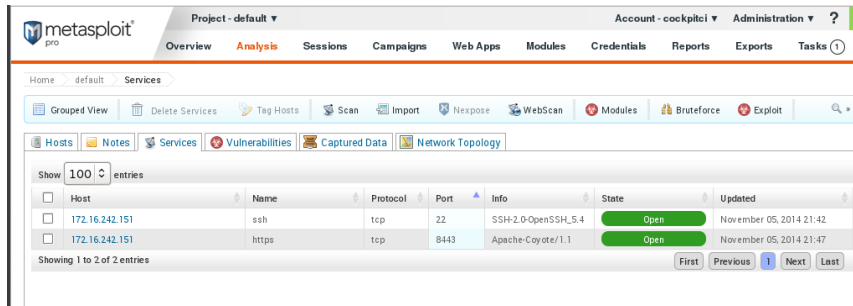


Figure 20 - Report on available services

After the scan, by accessing to the note panel the information on the machine can be listed.

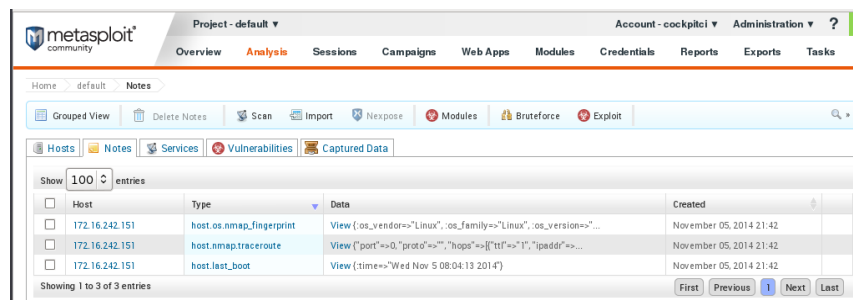


Figure 21 - Report on retrieving of data of SMGW

Over the web interface the tasks that have been performed can be checked as shown in Figure 22.

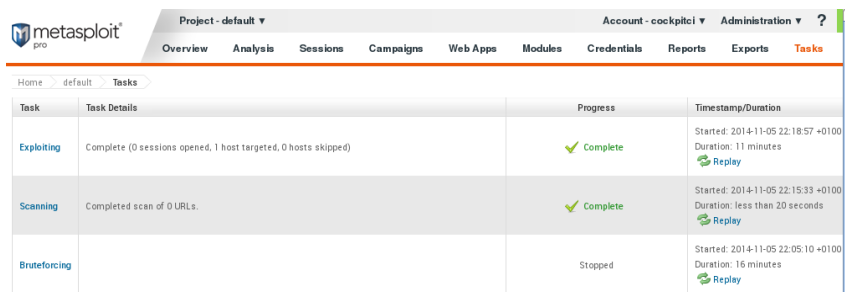


Figure 22 - Report on activities

The exploitation functions have scanned the SMGW in order to look for all available services.

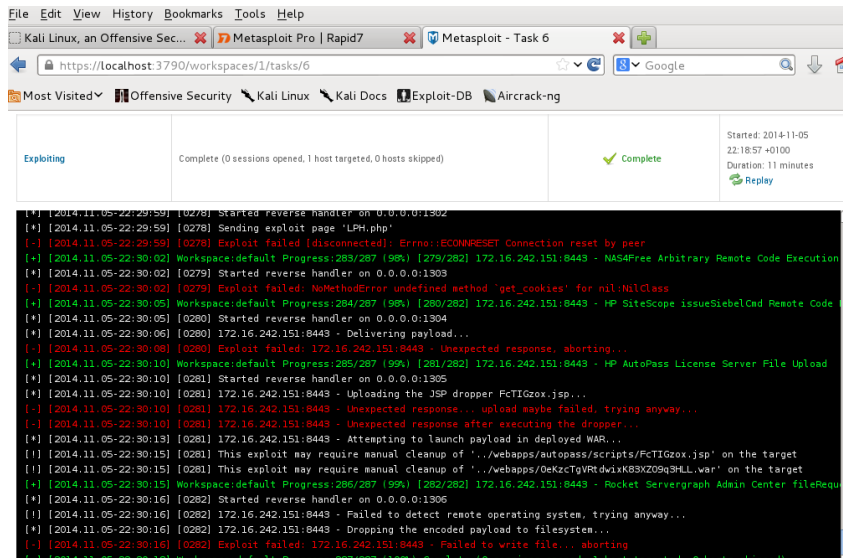


Figure 23 - Log of exploiting

A brute-force attack has been carried out passing the SMGW machine (user and password for SMGW, as shown in Figure 24 and using dummy credentials as shown in Figure 25). This attack consists of systematically checking all possible keys or passwords until the correct one is found.

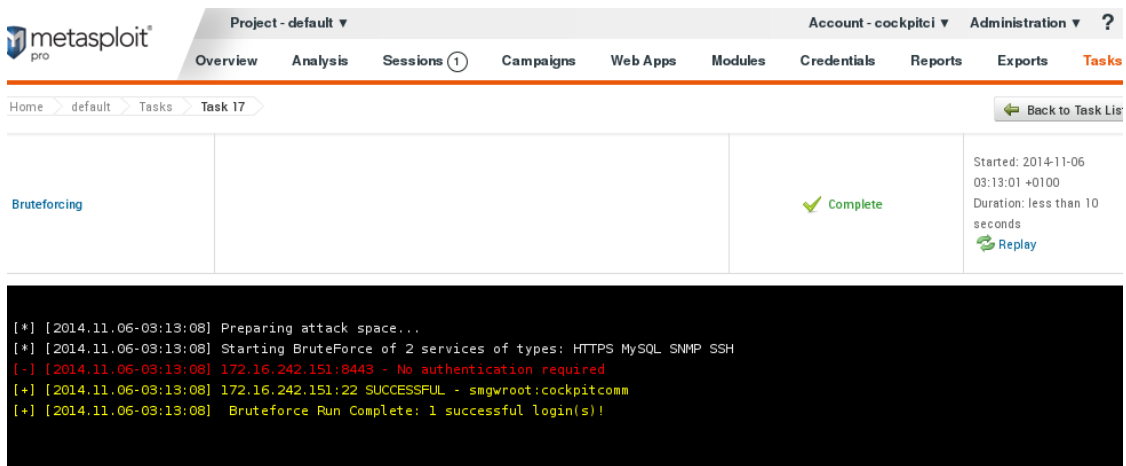


Figure 24 – Bruteforce by passing the SMGW credentials

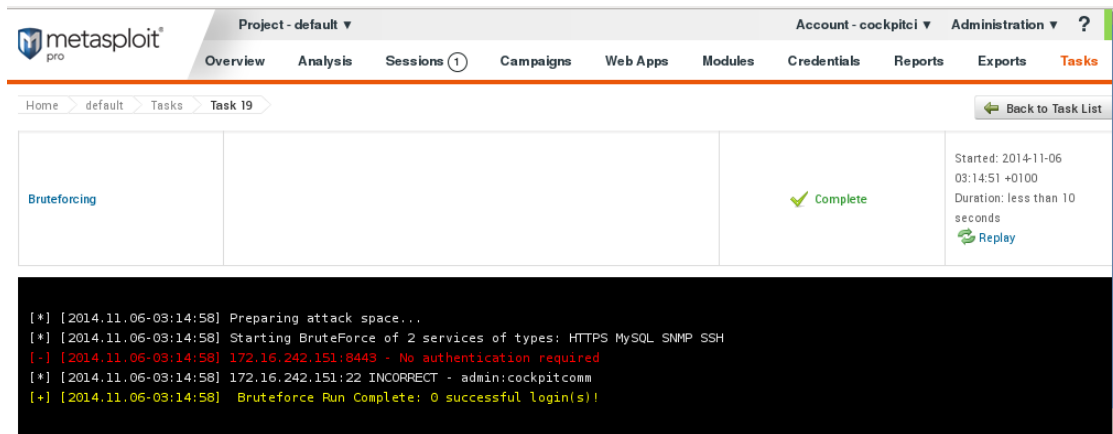


Figure 25 - Bruteforce attack by passing dummy credentials

The two experiments show that the attack was successful on port 22 (Figure 24) and not on port 8443 where the services of SMGW are exposed. The port 8443 uses X.509 certificates among the nodes instead of other forms of authentication.

Figure 26 shows the summary report provided by Metasploit.

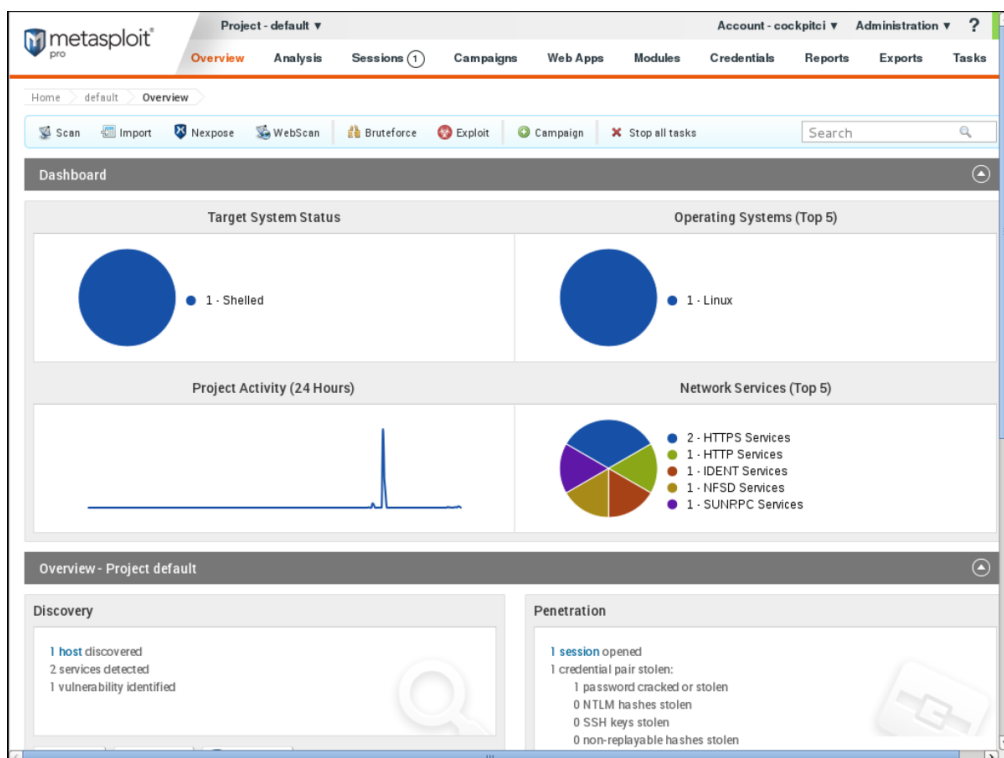


Figure 26 - Report on the SMGW security check after the test

3 SMGW Factory unit tests

3.1 SMGW Factory tests tools

In order to carry out the Unit test an add-on of a web browser has been employed in order to emulate the entities interfacing with the SMGW. For this scope the Poster Firefox addon (<https://addons.mozilla.org/en-US/firefox/addon/poster/>) has been employed [9].

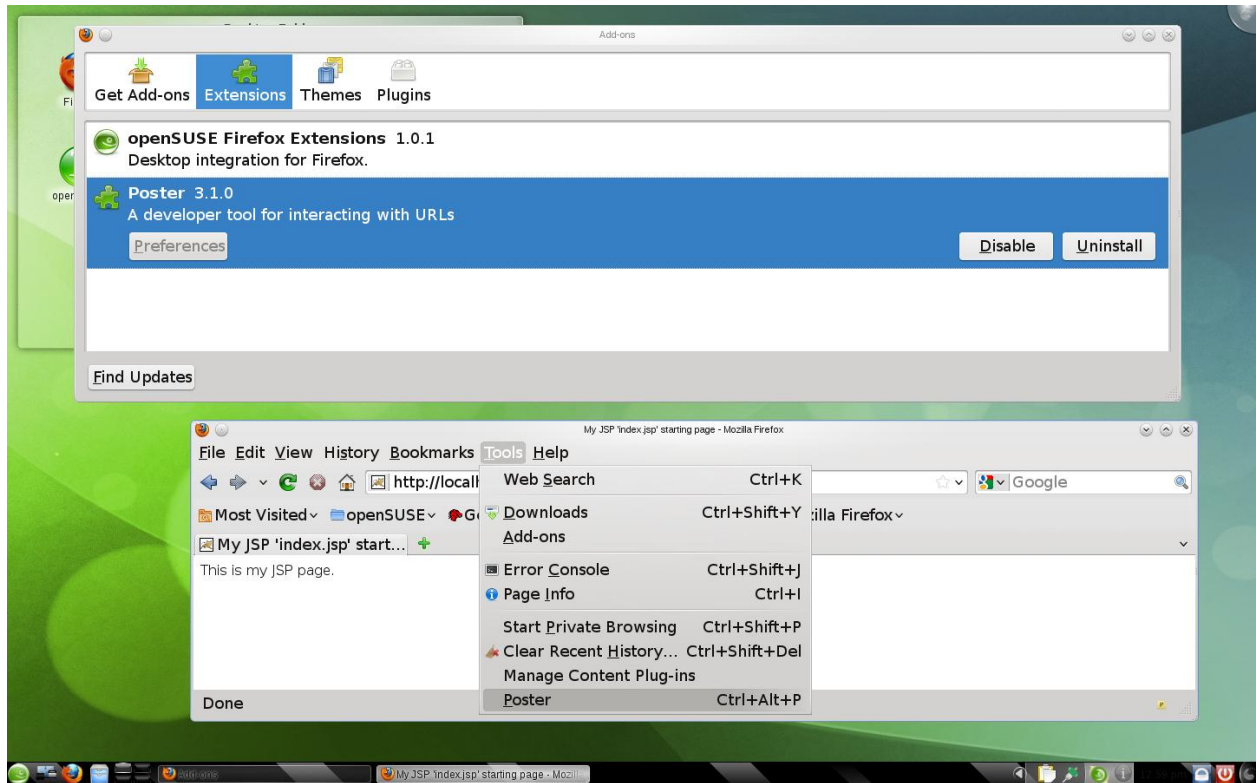


Figure 27 - Firefox Poster Add-on

Poster is a developer's tool for interacting with web services and other web resources that lets you make HTTP requests, set the entity body, and content type. This allows you to interact with web services and inspect the results.

The use of Firefox Mozilla and Poster 3.1.0 Add-on (<https://addons.mozilla.org/en-US/firefox/addon/poster/>) is recommended.

3.2 Communication test with DL, SCADA Adaptor and IRP

This section is addressed to describe the tests that were performed on the single software modules. On each part of the SMGW some tests were performed in order to verify the correct implementation of planned features. These tests are designed to demonstrate the capability of the SMGW to exchange messages using different protocols (REST and SOAP). This capability satisfies the requirement SMN_19.

Requirement id	Short description	Implementation	Required features of SMGW
SMN_19	The Secure Mediation Network <i>should</i> enforce different communications protocols/technologies in each particular context	For each different communication, the SMN management functionalities allow the administrator to select the protocol /technology from a list of suitable choices .	Different interfaces should be implemented.

3.2.1 Tests on Detection Layer application running in SMGW

The Detection Layer application in SMGW has been developed in order to collect messages coming from DL entity. It receives messages from intraSMGW DL entity by means of a REST interface and stores them in a database. The web service resource is accessible by means of an interface based on the HTTP/HTTPS standard methods and is addressable by means of an URI. The factory unit tests have been carried out by using the loopback address.

As preliminary operation, in order to deploy the detection layer application, the CockpitDL.war software module must be uploaded on Tomcat server.

The first test aims to simulate the SMGW receiving data from the Detection Layer in IDMEF format as shown in Figure 28. This figure shows the message flow from the REST client up to the storage in the local repository. The REST client in the DL application is simulated by means of a Poster message using the POST method. The message contains an IDMEF file that is stored in the local database. If the storage procedure is correctly executed the server returns back a 200 OK message. If the message is not delivered to the server the following messages can be sent back:

- 400 Bad Request,
- 401 Unauthorized
- 415 Unsupported media type
- 500 Internal Server Error

SMGW receives DATA from DL

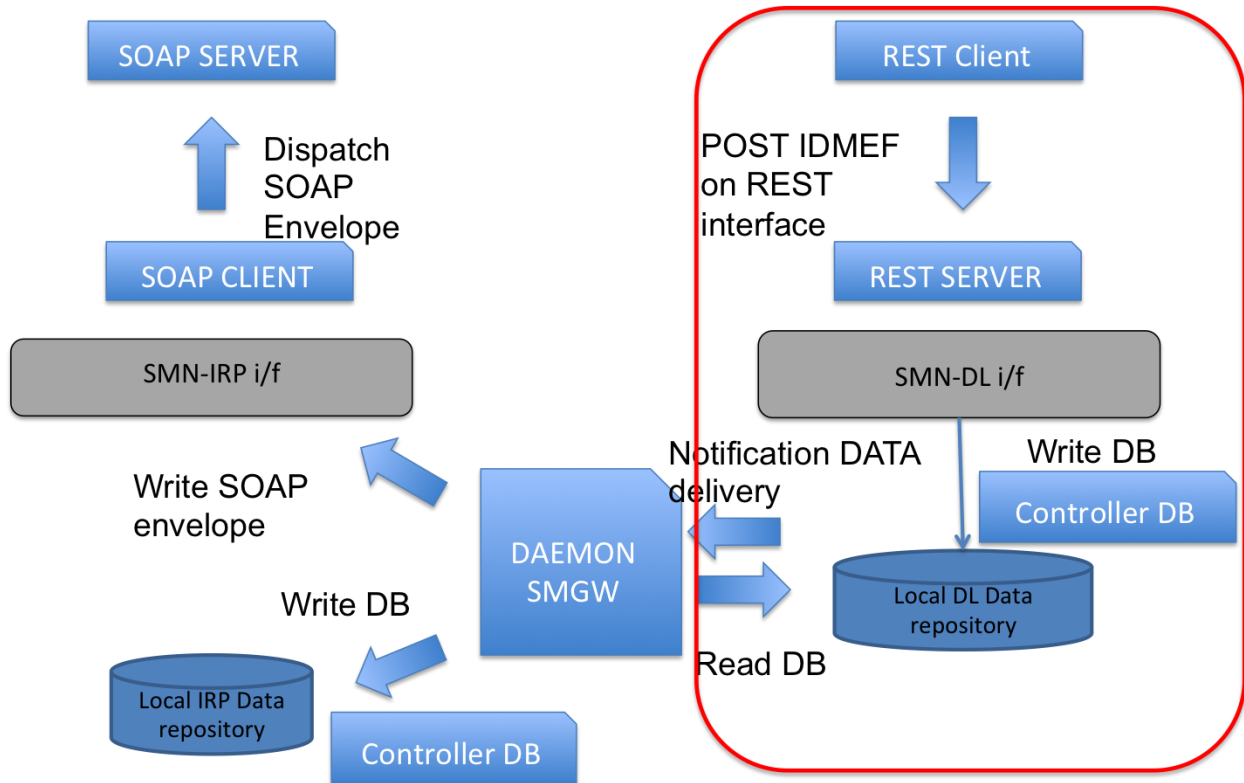


Figure 28 – DL application test scheme

This test shows how the SMGW receives data from DL by means of a POST method in “idmef” format and how it saves them into a DB. This test can be performed on localhost by using the Firefox Poster add-on (Figure 29). The GET method is disabled for REST services in the final release of software.

This test meets the following requirement:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_2	The Secure Mediation Network <i>shall</i> interface with the detection layer	A proper DL interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired in SMGW database from DL entity by means of a POST method.

In order to perform this test the item Tools->Poster in the menu must be selected to open the poster window. After that POST methods can be tested.

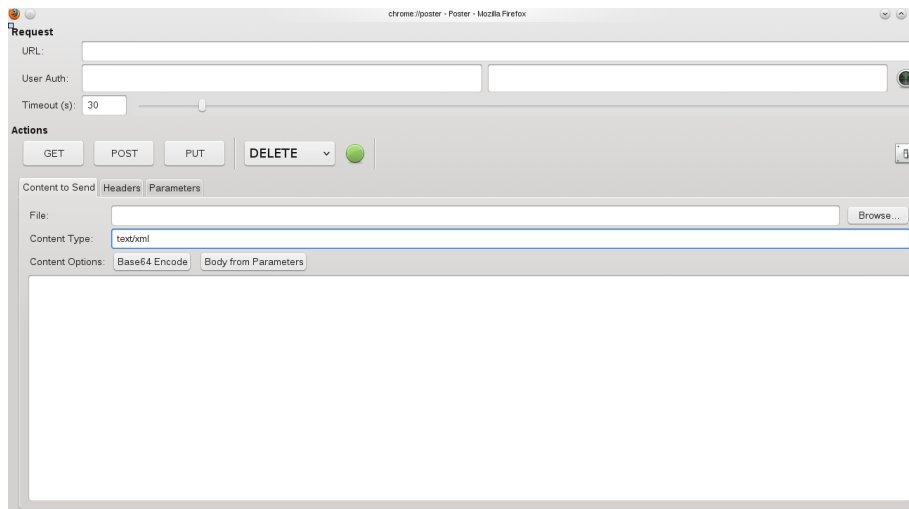


Figure 29 - Poster Application

Open a Poster Request URL using the following link as shown in Figure 30: <http://localhost:8080/CockpitDL/services/idmefentities/addFile>

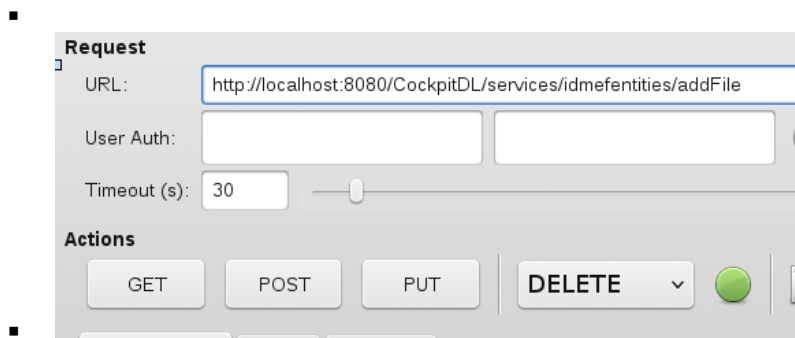


Figure 30 - Set URL on Poster application

Using this method an IDMEF file can be sent towards a listening web server. The URL identifies the resource on the Web Server and the IDMEF message is charged by means of a file. The file of the IDMEF message can be selected by browsing the file system as shown in Figure 31.



Figure 31 – Select the IDMEF file

The IDMEF files are contained in /home/smgwroot/Downloads/ directory. The following IDMEF files have been used to perform the tests:

- Snort_NetworkScan.idmef
- Snort_NetworkFlood.idmef
- Snort_MITM_ARPPoisoning.idmef
- Honeypot_modbusInteraction.idmef

Set the content type with **application/idmef** as shown in Figure 32.

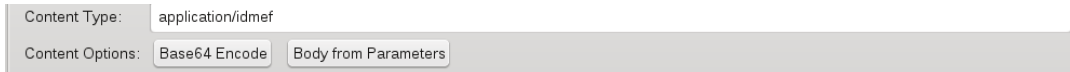


Figure 32 – Set the content type

Select the POST button as shown in Figure 33.



Figure 33 – Apply the POST method

You will get the response from the server as shown in Figure 34. The status indicator “200 OK” means that the POST method has been successful.

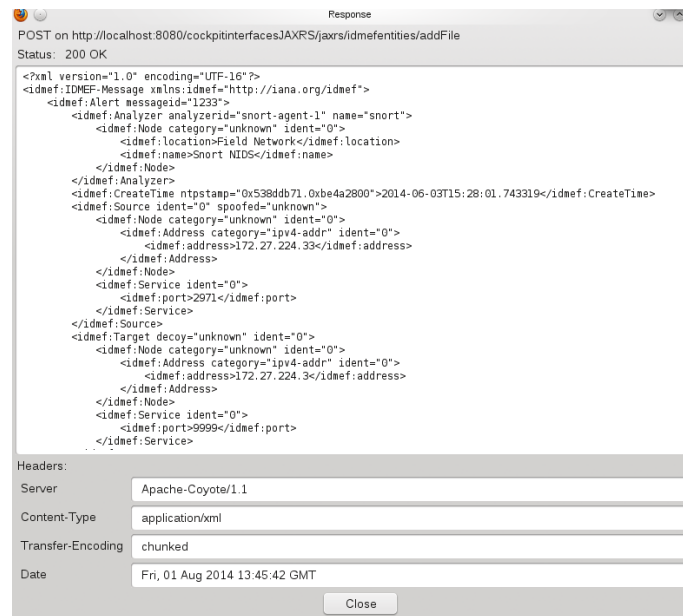


Figure 34 – Response from the server to POST

For example the application returns an error 415 if the format of the file declared in the “Content Type” is different from the IDMEF format as shown in Figure 35.

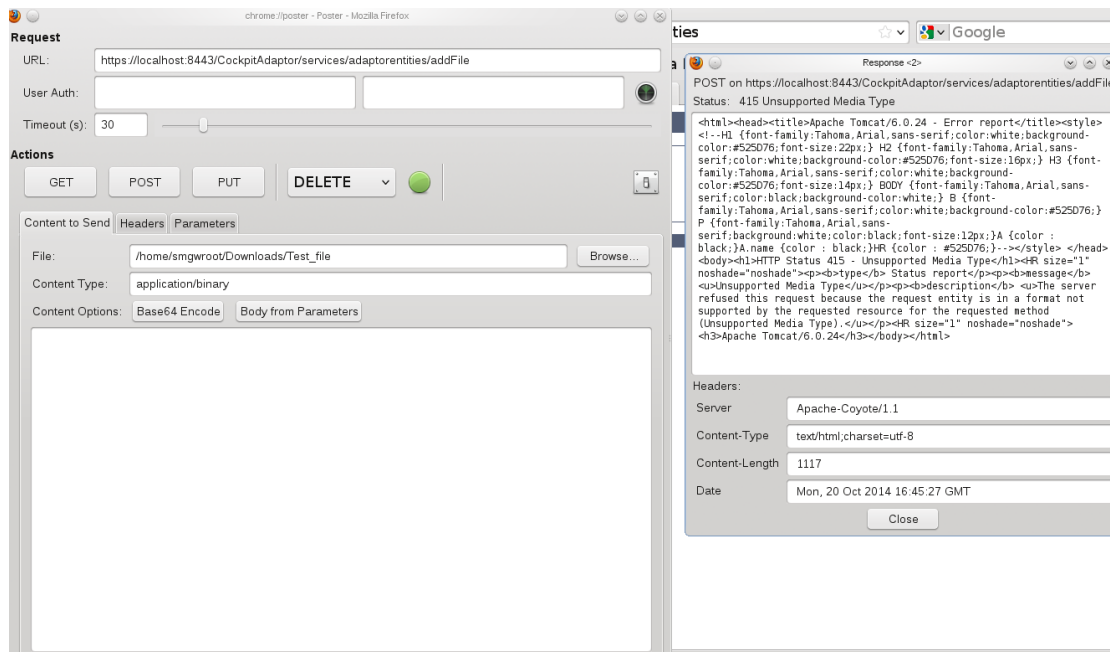


Figure 35 – Error in POST

The content's file has been saved into DB that can be checked with a select on table "localddata".

Using <http://localhost:8080/CockpitDL/services/idmefentities/addContent> the single field can be filled on "Panel" tab.

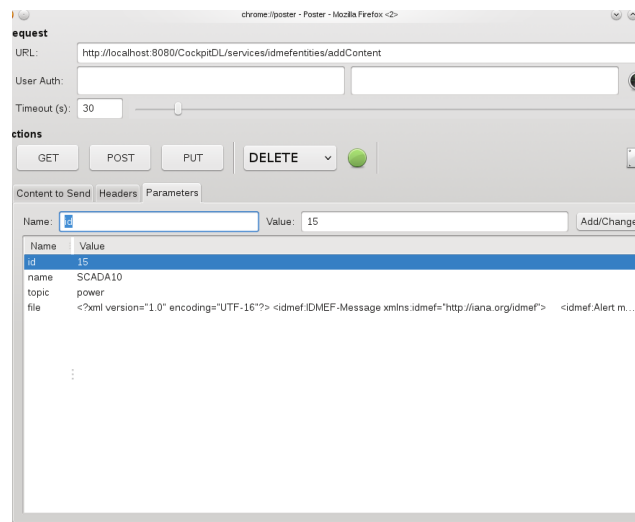


Figure 36 – Manual settings of the parameters Poster

The fields in the panel are:

- id: numeric identifier;
- name: name of the resource;
- topic: identifier of the topic in text format;
- file: content of the file.

By selecting the command POST the message is sent towards the SMGW as in Figure 37.

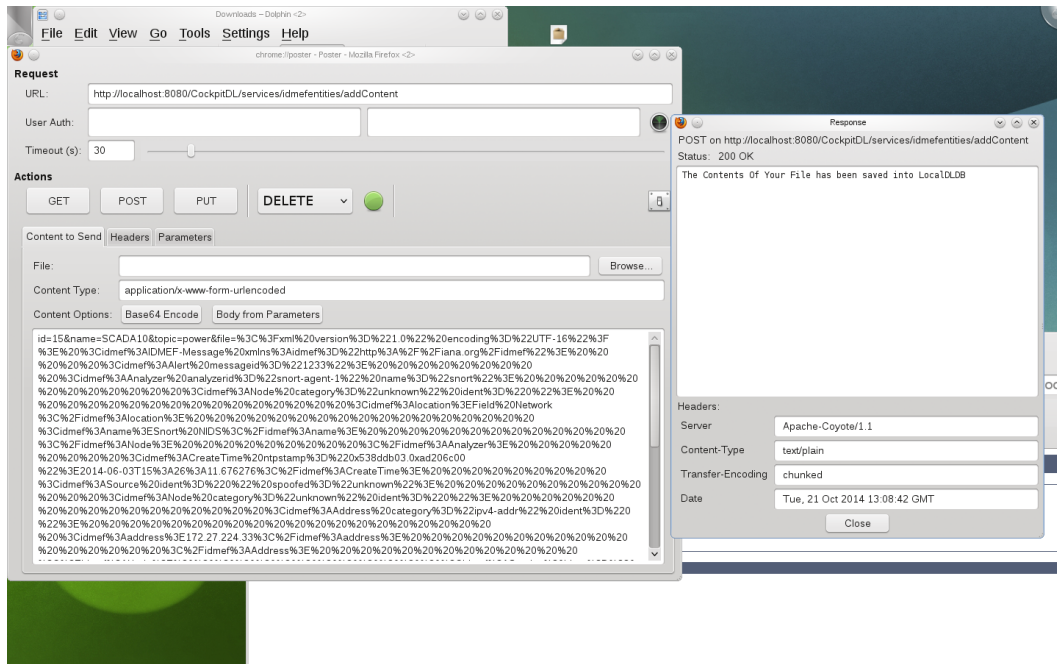


Figure 37 – POST of message using manual input of parameters

By command line the database can be inspected with the following commands:

- Type: su root – pw: cockpitcomm
- Type: mysql
- Type: connect cockpit;
- Type: show tables;
- Type: mysql> select * from localdlldata;

A user can check the field by means of the command “describe localdlldata;” as shown in Figure 38.

```
mysql> describe localdlldata;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id | smallint(6) | NO | PRI | NULL | auto_increment |
| id_dl | smallint(6) | YES | | NULL | |
| name | text | YES | | NULL | |
| topic | text | YES | | NULL | |
| IDMEFFile | longtext | YES | | NULL | |
| timestamp | text | YES | | NULL | |
| sent_irp | tinyint(4) | NO | | NULL | |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

Figure 38 - Contents of localdlldata table

This test meets the following requirement:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_7	The Secure Mediation Network <i>shall</i> store information obtained by all interfaced components in a dedicated database	Proper DB modules of the SMGW will manage exchange and storage of metadata.	Data arriving from DL and SCADA adaptor should be collected in dedicated databases.

3.2.2 Tests on SCADA Adaptor application running in SMGW

SCADA Adaptor application has been developed using REST based interface. A resource is accessible by means of an interface based on the HTTP/HTTPS standard methods. A Restful web service typically is addressable by means of an URI. This document explains how to perform the test by using the localhost address. These tests meet the following requirement:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_6	The Secure Mediation Network <i>shall</i> acquire CI independent metadata from SCADA adaptors	A proper SCADA Adaptor interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired in SMGW database form SCADA Adaptor by means of a POST method.

As a preliminary operation, in order to deploy the detection layer application, the CockpitAdaptor.war must be uploaded on Tomcat server. In this case the application is pre-charged.

As in the case of the Detection Layer the CockpitAdaptor.war application is able to receive data and files in IDMEF format. The following POST methods are available:

- <http://localhost:8080/CockpitAdaptor/services/adaptorentities/addContent>
- <http://localhost:8080/CockpitAdaptor/services/adaptorentities/addFile>

The test aims to demonstrate that the SMGW is able to receive data using the REST interface as shown in the diagram in Figure 39. This figure shows the message flow coming from the SCADA Adaptor REST client up to the storage database in the local repository. The REST client in the SCADA Adaptor application is simulated by means of a Poster message using the POST method. The message contains an IDMEF file that is stored in the local database. If the storage procedure is correctly executed the server returns back a 200 OK message. If the message is not delivered to the server the following messages can be sent back:

- 400 Bad Request,
- 401 Unauthorized
- 415 Unsupported media type
- 500 Internal Server Error

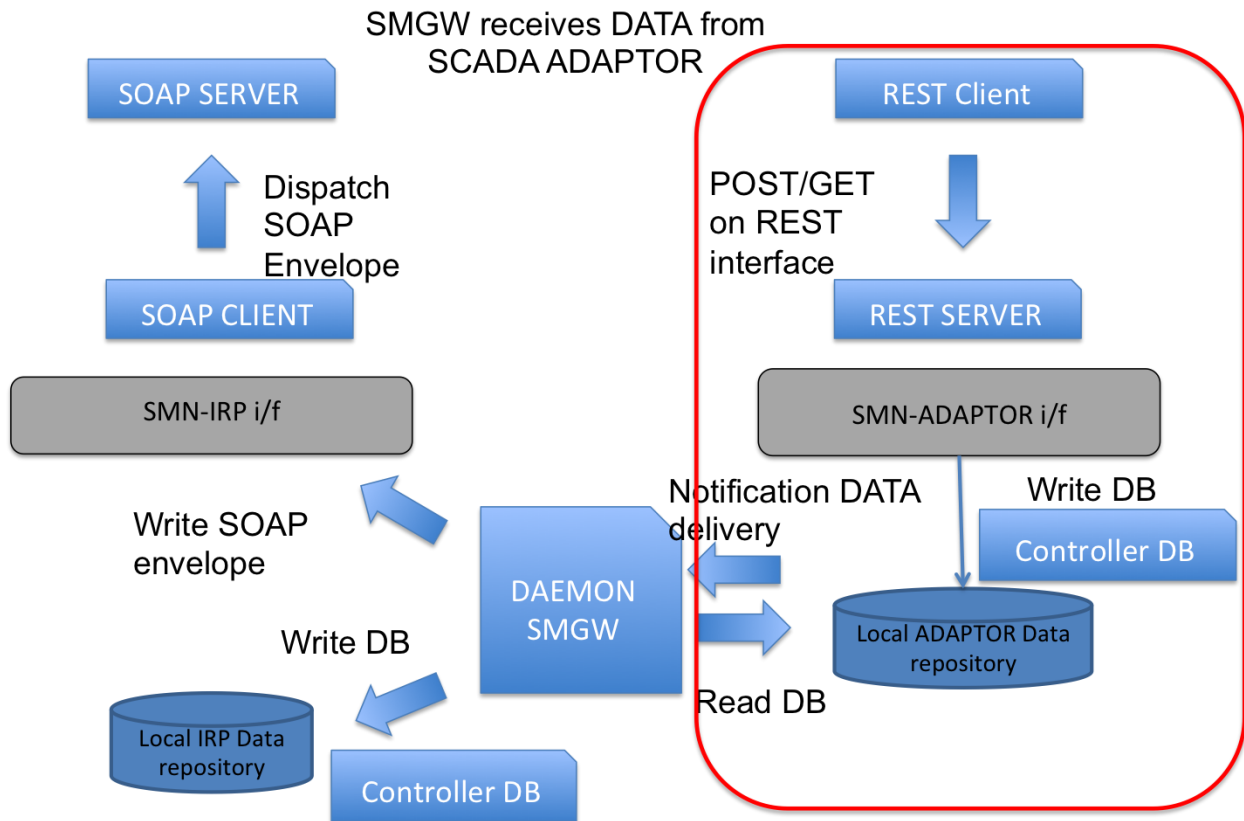


Figure 39 – SCADA Adaptor application test scheme

As in the previous case, the local test have been performed by using the Poster tool in order to perform the HTTP/HTTPS requests and to set the message contents. This allows the use of the web service interface and the inspection of the results. The use of Firefox Mozilla and Poster 3.1.0 Add-on (<https://addons.mozilla.org/en-US/firefox/addon/poster/>) is recommended also in this case.

The tests have been carried out by using the addFile or addContent methods. In order to perform them, the tester has to open a Poster Request URL using the following reference:

<https://localhost:8443/CockpitAdaptor/services/adaptorentities/addFile>

<https://localhost:8443/CockpitAdaptor/services/adaptorentities/addContent>

Open a Poster Request URL using the following link as shown in Figure 40:
<https://localhost:8443/CockpitAdaptor/services/adaptorentities/addFile>

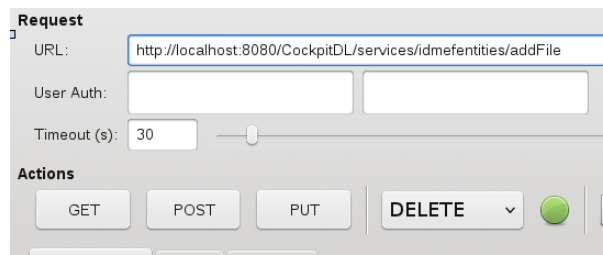


Figure 40 - Set URL on Poster application

Using this method an IDMEF file can be sent towards a listening web server. The URL identifies the resource on the Web Server and the IDMEF message is charged by means of a file. The file of the IDMEF message can be selected by browsing the file system as shown in Figure 41.

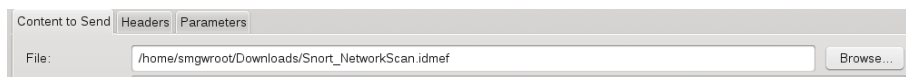


Figure 41 – Select the IDMEF file

The IDMEF files are contained in the /home/smgwroot/Downloads/ directory. The user can select one among the following files (they are the same as in the previous case because the IDMEF format is the same):

- Snort_NetworkScan.idmef
- Snort_NetworkFlood.idmef
- Snort_MITM_ARPPoisoning.idmef
- Honeypot_modbusInteraction.idmef

Set the content type with **application/idmef** as shown in Figure 42.

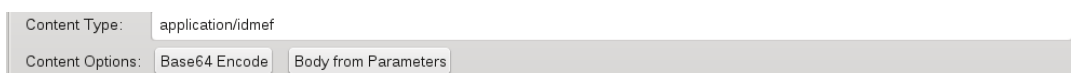


Figure 42 – Set the content type

Select the POST button as shown in Figure 43.



Figure 43 – Apply the POST method

You will get the response from the server as shown in Figure 44. The status indicator “200 OK” means that the POST method has been successful.

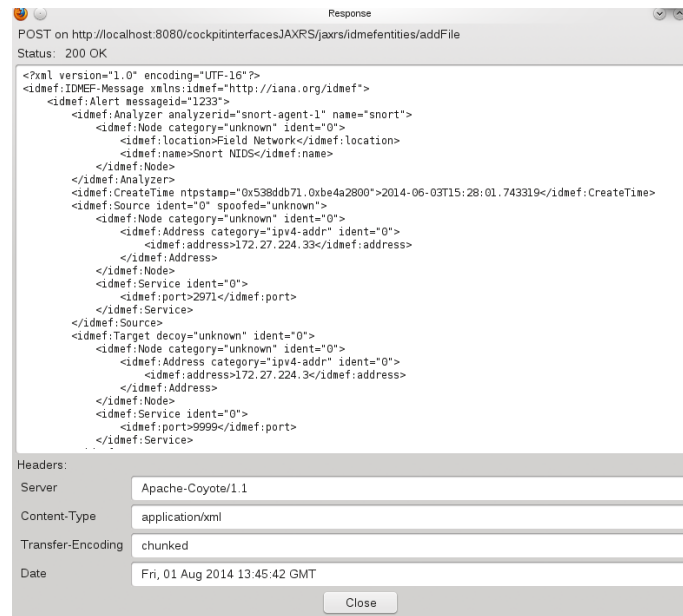


Figure 44 – Response from the server to POST

For example the application returns an error 415 if the format of the file declared in the “Content Type” is different from the IDMEF format as shown in Figure 45.

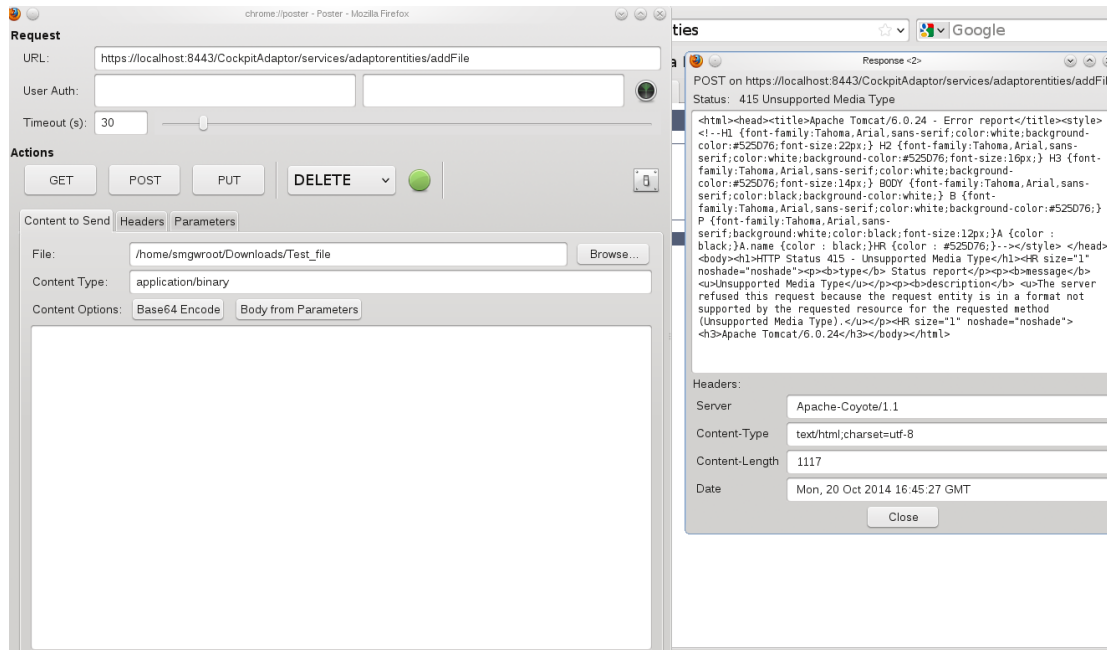


Figure 45 – Error in POST

The content’s file has been saved into DB that can be checked with a select on table “cockpitadaptor”. The database can be checked either by an interface as in Figure 46 or by command line.

Using <https://localhost:8443/CockpitAdaptor/services/adaptorentities/addContent> the single field can be filled on “Panel” tab.

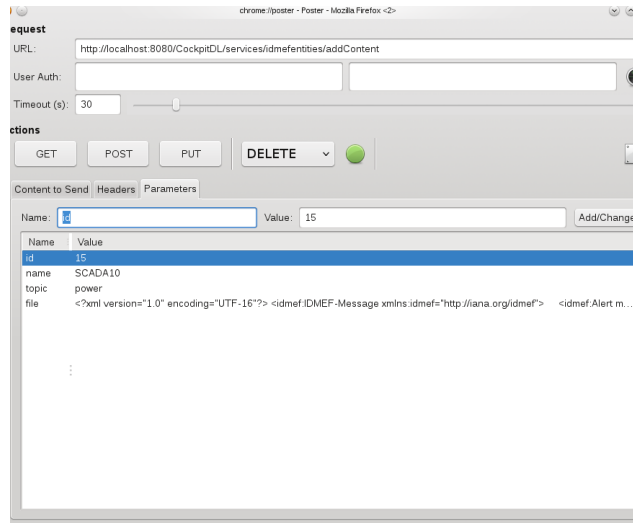


Figure 46 – Manual settings of the parameters Poster

The fields in the panel are:

- id: numeric identifier;
- name: name of the resource;
- topic: identifier of the topic in text format;
- file: content of the file.

By selecting the command POST the entities are sent towards the SMGW as in Figure 47.

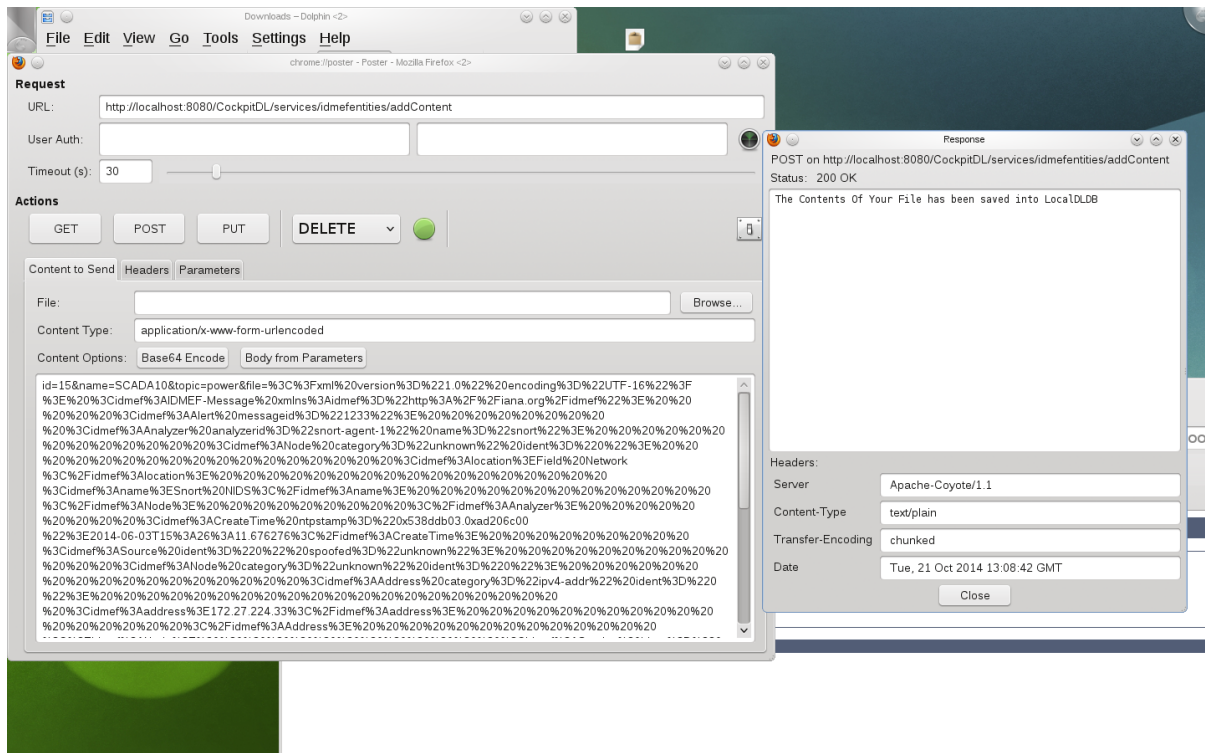


Figure 47 – POST of message using manual input of parameters

By command line the database can be inspected following the commands:

- Type: su root – pw: cockpitcomm
- Type: mysql
- Type: connect cockpit;
- Type: show tables;
- Type: mysql> select * from cockpitadaptor;

A user can check the field by means of the command “describe cockpitadaptor” as shown Figure 48.

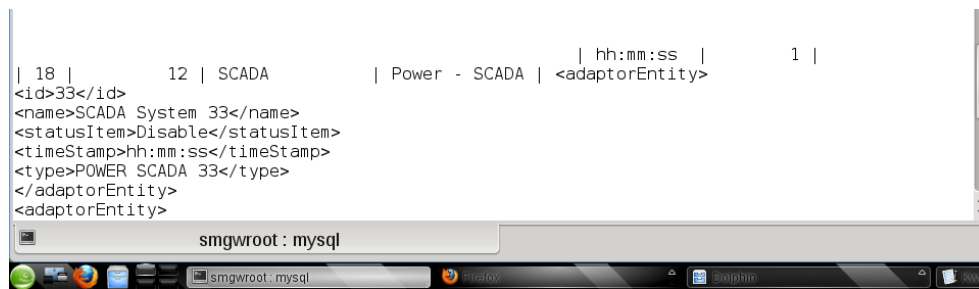


Figure 48 - Select in cockpitadaptor table

3.2.3 Tests on IRP application running in SMGW

The IRP application on SMGW is based on the SOAP architecture. A resource is accessible by means of an interface based on HTTP method and a SOAP service is addressable by means of an URI. This document explains how to perform the test by using the localhost address.

In order to know how the client SOAP communicates to service provider (Server Soap) it must have the below information:

- Location of web services server;
- Functions available, signature and return types of function;
- Communication protocol;
- Input output formats.

Using SOAP, the information provider has to publish an XML file which must have all the methods required for the services so a client can access them. This XML file is called WSDL and this test meets the following requirement:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_3	The Secure Mediation Network <i>shall</i> interface with the prediction tool	A proper IRP interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired from IRP by means a of GET method

For consuming the IRP web service the tester must deploy the CockpitIRP.war on Tomcat server and run it.

After the user can access the information page, as shown in Figure 49, by browsing the URL [ssl https://localhost:8443/SMN-IRP/IRPPort](https://localhost:8443/SMN-IRP/IRPPort) (or without SSL <http://localhost:8080/SMN-IRP/IRPPort>)



Figure 49 – Web service information page

The tester can click the WSDL URL to see the all information provided by the server IRP SOAP as shown in Figure 50: <http://localhost:8443/SMN-IRP/IRPPort?wsdl> (or without SSL <http://localhost:8080/SMN-IRP/IRPPort?wsdl>). In the figure the WSDL is published by means of the end point in the SMGW towards the intradomain entities.

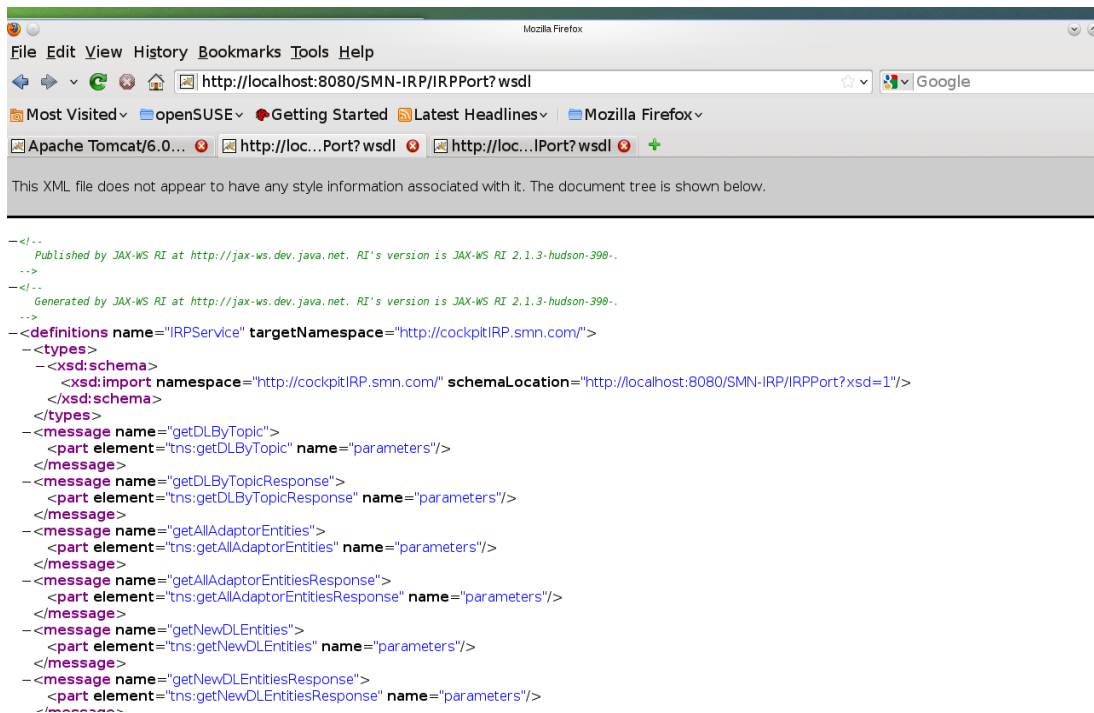


Figure 50 - Cockpit WSDL endpoint

The information coming from DL and SCADA Adaptor are stored In the local database. The IRP can perform selective request of unread messages. Each time the IRP makes a query to get new information stored in SMGW, the request is flagged in the database setting true the "read field".

The WSDL provided towards the IRP is:

```
<?xml version="1.0" encoding="UTF-8"?><!-- Generated by JAX-WS RI at
http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3-hudson-390-. --
><definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://cockpitIRP.smn.com/" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="IRPService"
targetNamespace="http://cockpitIRP.smn.com/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://cockpitIRP.smn.com/"
schemaLocation="IRPService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="getDLByTopic">
    <part element="tns:getDLByTopic" name="parameters"/>
  </message>
  <message name="getDLByTopicResponse">
    <part element="tns:getDLByTopicResponse" name="parameters"/>
  </message>
  <message name="getAllAdaptorEntities">
    <part element="tns:getAllAdaptorEntities" name="parameters"/>
  </message>
  <message name="getAllAdaptorEntitiesResponse">
    <part element="tns:getAllAdaptorEntitiesResponse" name="parameters"/>
  </message>
  <message name="getNewDLEntities">
    <part element="tns:getNewDLEntities" name="parameters"/>
  </message>
  <message name="getNewDLEntitiesResponse">
    <part element="tns:getNewDLEntitiesResponse" name="parameters"/>
  </message>
  <message name="getAllDLEntities">
    <part element="tns:getAllDLEntities" name="parameters"/>
  </message>
  <message name="getAllDLEntitiesResponse">
    <part element="tns:getAllDLEntitiesResponse" name="parameters"/>
  </message>
  <message name="getNewAdaptorEntities">
    <part element="tns:getNewAdaptorEntities" name="parameters"/>
  </message>
  <message name="getNewAdaptorEntitiesResponse">
    <part element="tns:getNewAdaptorEntitiesResponse" name="parameters"/>
  </message>
  <message name="getAdaptorEntityByTopic">
```

```
<part element="tns:getAdaptorEntityByTopic" name="parameters"/>
</message>
<message name="getAdaptorEntityByTopicResponse">
  <part element="tns:getAdaptorEntityByTopicResponse" name="parameters"/>
</message>
<portType name="IRPInterface">
  <operation name="getDLByTopic">
    <input message="tns:getDLByTopic"/>
    <output message="tns:getDLByTopicResponse"/>
  </operation>
  <operation name="getAllAdaptorEntities">
    <input message="tns:getAllAdaptorEntities"/>
    <output message="tns:getAllAdaptorEntitiesResponse"/>
  </operation>
  <operation name="getNewDLEntities">
    <input message="tns:getNewDLEntities"/>
    <output message="tns:getNewDLEntitiesResponse"/>
  </operation>
  <operation name="getAllDLEntities">
    <input message="tns:getAllDLEntities"/>
    <output message="tns:getAllDLEntitiesResponse"/>
  </operation>
  <operation name="getNewAdaptorEntities">
    <input message="tns:getNewAdaptorEntities"/>
    <output message="tns:getNewAdaptorEntitiesResponse"/>
  </operation>
  <operation name="getAdaptorEntityByTopic">
    <input message="tns:getAdaptorEntityByTopic"/>
    <output message="tns:getAdaptorEntityByTopicResponse"/>
  </operation>
</portType>
<binding name="IRPPortBinding" type="tns:IRPInterface">
  <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getDLByTopic">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="getAllAdaptorEntities">
    <soap:operation soapAction=""/>
```

```
<input>
  <soap:body use="literal"/>
</input>
<output>
  <soap:body use="literal"/>
</output>
</operation>
<operation name="getNewDLEntities">
  <soap:operation soapAction=""/>
  <input>
    <soap:body use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
  </output>
</operation>
<operation name="getAllDLEntities">
  <soap:operation soapAction=""/>
  <input>
    <soap:body use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
  </output>
</operation>
<operation name="getNewAdaptorEntities">
  <soap:operation soapAction=""/>
  <input>
    <soap:body use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
  </output>
</operation>
<operation name="getAdaptorEntityByTopic">
  <soap:operation soapAction=""/>
  <input>
    <soap:body use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
  </output>
</operation>
</binding>
<service name="IRPService">
```

```

    <port binding="tns:IRPPortBinding" name="IRPPort">
<soap:address location="http://localhost:8080/SMN-IRP/IRPPort"/>
    </port>
</service>
</definitions>

```

This test provides the fulfilling of the following requirement:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_3	The Secure Mediation Network <i>shall</i> interface with the prediction tool	A proper IRP interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired from IRP by means a of GET method

The WSDL provides three GET methods for queries on DL data:

- **getAlIDLEntities**: by invoking this operation the tester can retrieve all data rows from detection layer database;
- **getNewDLEntities**:by invoking this method the tester get those data rows from detection layer database, in which they are not flagged (those data that has not been sent to IRP);
- **getDLByTopic**: by invoking this operation the tester can retrieve those data from detection layer database filtered by Topic.

In the below figures there is a client soap (java) that calls the **getAlIDLEntities()** method, in which it can retrieve all data from database.

The URL can be set both with SSL using 8443 port and HTTPS or without SSL as in Figure 51. The content/type field is set to: text/xml.

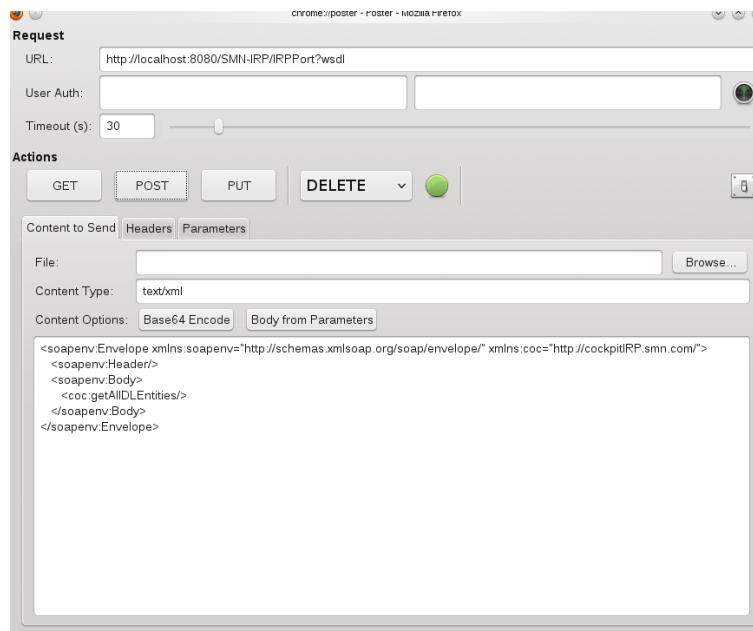


Figure 51 - Request to SMGW

The user has to do the request using the following XML code:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:coc="http://cockpitIRP.smn.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <coc:getAllDLEntities/>
  </soapenv:Body>
</soapenv:Envelope>

```

Figure 52 shows the response to the IRP.

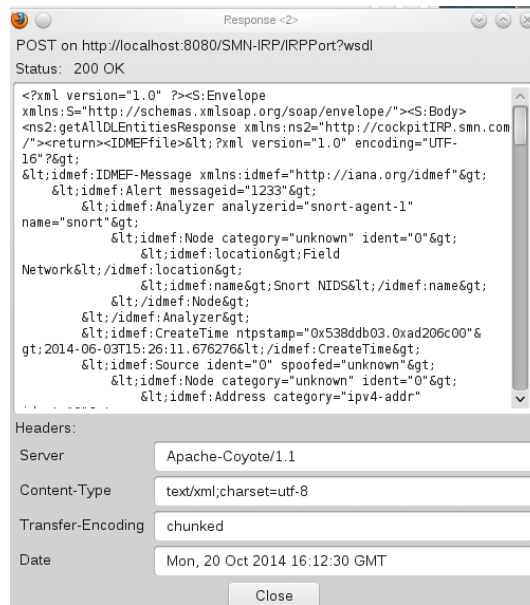


Figure 52 - Response to the IRP

Also the **getNewDLEntities** method can be applied without parameters such as in the following example:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:coc="http://cockpitIRP.smn.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <coc:getNewDLEntities/>
  </soapenv:Body>
</soapenv:Envelope>
  
```

For **getDLByTopic ()** an argument as parameter can be set in order to allow the selection by means of a topic.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:coc="http://cockpitIRP.smn.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <coc:getDLByTopic>
      <!--Optional:-->
      <arg0>?</arg0>
    </coc:getDLByTopic>
  
```

```
</soapenv:Body>
```

```
</soapenv:Envelope>
```

The WSDL provides three GET methods for queries on SCADA Adaptor data:

- **getAllAdaptorEntities**: by invoking this operation the tester can retrieve all data rows from Scada Adaptor database;
- **getNewAdaptorEntities**: by invoking this method the tester can get those data rows from Scada Adaptor database, in which they are not flagged (data that has not been sent to IRP);
- **getAdaptorEntityByTopic**: by invoking this operation the tester can retrieve those data from Scada Adaptor database filtered by Topic.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:coc="http://cockpitIRP.smn.com/">
```

```
<soapenv:Header/>
```

```
<soapenv:Body>
```

```
<coc:getAllAdaptorEntities/>
```

```
</soapenv:Body>
```

```
</soapenv:Envelope>
```

Also the **getNewAdaptorEntities** method can be applied without parameters such as in the following example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:coc="http://cockpitIRP.smn.com/">
```

```
<soapenv:Header/>
```

```
<soapenv:Body>
```

```
<coc:getNewAdaptorEntities/>
```

```
</soapenv:Body>
```

```
</soapenv:Envelope>
```

For **getAdaptorEntityByTopic ()** an argument as parameter can be set in order to allow the selection by means of a topic.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:coc="http://cockpitIRP.smn.com/">
```

```
<soapenv:Header/>
```

```
<soapenv:Body>
```

```
<coc:getAdaptorEntityByTopic>
```



```

<!--Optional:-->

<arg0>SCADA3</arg0>

</coc:getAdaptorEntityByTopic>

</soapenv:Body>

</soapenv:Envelope>
  
```

In order to consume the web services a test routine can implement two options:

- In IDE(eclipse,netbeans,) by selection on java file (soap web service consumer) it is required a new web service in top down (from WSDL to JAVA) including the URL of WSDL (<http://localhost:8080/SMN-IRP/IRPPort?wsdl> or <https://localhost:8443/SMN-IRP/IRPPort?wsdl>) and selecting finish.
- Add to java code the following lines:
 - URL wsdlLocation=new URL(<http://localhost:8080/SMN-IRP/IRPPort?wsdl> ");
 - CisialmplService service= new CisialmplService(wsdlLocation,new QName("<http://cockpitIRP.smn.com/>", "CisialmplService"));

All methods in the WSDL have been performed by using the procedure shown in Figure 51 and in all cases the server replied correctly by using the format shown in Figure 52 with a response 200 OK. With all the other test by using malformed XML statement or incorrect URL the server replied with the following HTTP error codes:

- 400 Bad Request,
- 401 Unauthorized
- 415 Unsupported media type
- 500 Internal Server Error

3.3 Test SMGW-SMGW

The SMGW-SGMW communication model has been developed using the SOAP. A resource is accessible by means of a web service end point where a subscriber can consume information. A SOAP web service typically is addressable by means of an URI containing a WSDL that describes the provided services.

As explained in section 2.2 the communication interSMGW has been developed by setting the security parameters of the Application Server and deploying within the server the publisher application. The requirements fulfilled by the deployed architectural model are the following:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_16	The Secure Mediation Network <i>shall</i> provide a management interface allowing a certain degree of security and policies configuration	A control interface is foreseen (see Section 7).	Implemented at level of interchange SMGW-SMGW
SMN_17	The Secure Mediation Network <i>should</i> provide the possibility to define who and in which way	The definition and management of proper policies in the SMN will	The SMGW should filter the remote nodes on the basis of an access list.

	can access a certain piece of information	allow to achieve such a result.	
SMN_20	The Secure Mediation Network <i>should</i> enforce Service Level Agreements (SLA) or Service Level Specifications (SLS) between CIs	The definition and management of proper policies in the SMN will allow to achieve such a result.	The management panel of the SMGW should assure the SLA on basis of predefined proprieties.

The tests have been carried out by using a couple of SMGWs in mirror mode on VMware hypervisor in order to send a message from one of them and consuming it on the other one. The published messages are stored in a local database before the publishing. The incoming messages consumed on the remote SMGW are stored in a local database containing all the collected messages. The messages are sent in XML format in order to allow the settings and filtering of parameters.

The tests have been carried out in order to check the fulfilment of the following requirements:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_4	The Secure Mediation Network <i>shall</i> interface with external peer Secure Mediation Networks through an Internet connection	A proper interface has been foreseen in the architecture of the SMN.	A SOAP message should be sent between SMGWs using a publish / subscribe approach.
SMN_5	The Secure Mediation Network <i>shall</i> support non real-time exchange of other kinds of information among CIs	A message broker with separate queues for each topic, each with its own priority, has been introduced in the architecture for InterSMGW communications.	The messages can be consumed by remote SMGW by using a publish subscribe mechanism. The remote invocation can consume messages either by selecting all messages in the database or requesting all not already accessed.
SMN_8	A specific framework shall be included in the Secure Mediation Network in order to allow local CockpitCI components and external SMGWs to retrieve metadata useful for their purposes	A message broker implementing a topic oriented, publish-subscribe mechanism has been adopted. Moreover, Web services implementing the request-response scheme have been introduced for metadata retrieval.	A metadata publisher should assure the capability to retrieve the metadata.
SMN_9	The Secure Mediation Network <i>shall</i> perform information discovery at peer SMGWs to retrieve state information of interdependent CIs	SMGW-SMGW communication will be based on publish/subscribe paradigm, hence supporting the standard procedures such as service publishing, discovery, subscription, etc.	In message exchange the SMGW should check the list of subscribers and of subscribed services
SMN_11	The Secure Mediation Network <i>shall</i> disclose stored global	SMN will disclose metadata provided by the	The SMGW should filter the notifications only to

	awareness metadata of the local CI, to authorized subscribers, i.e. other SMGWs	IRP (e.g. current and predicted CI status) only to authorized subscribers (e.g. other SMGWs).	subscribers.
SMN_12	The Secure Mediation Network <i>shall</i> accept subscriptions from peer SMGWs to be notified when updated metadata is available	SMGW-SMGW communication will be based on publish/subscribe paradigm, hence supporting the standard procedures such as service publishing, discovery, subscription, etc.	The SMGW-SMGW communication should be assured by means Web Services.
SMN_15	In order to guarantee a reliable and effective risk prediction, the Secure Mediation Network <i>shall</i> keep synchronized with remote CIs' metadata	According to the publish/subscribe paradigm, whenever new relevant data are available at a remote CI side, all the subscribers to the particular data receive an update.	The notification messages should be sent to all subscribers.

Figure 53 shows the test model adopted to test the communication between different SMGWs.

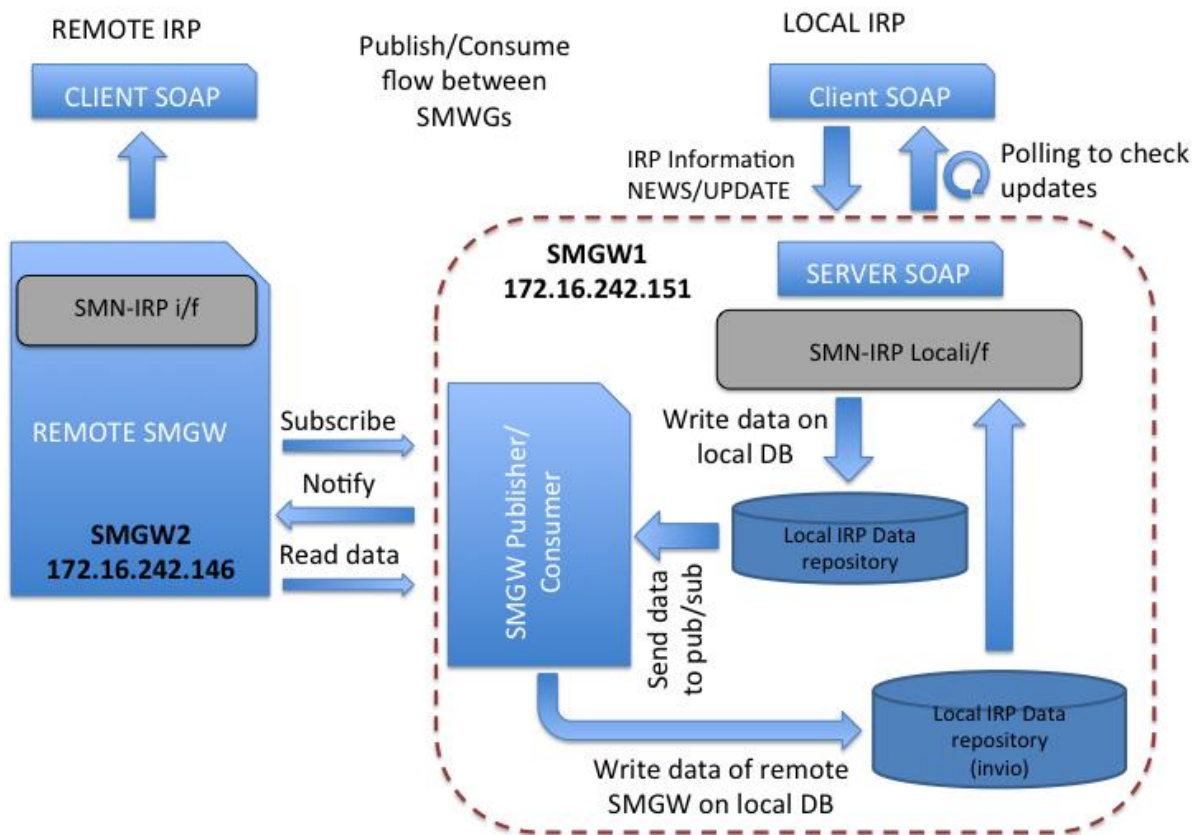


Figure 53 – Test configuration to test the communication between different SMGWs

The URI for the delivery of messages among the two SMGWs is the following.

<https://172.16.242.151:8443/MessageBroker/services/MessageBroker/publish>

The first step has been the mutual exchange of certificates between the SMGWs as shown in Figure 54 by using the browser capabilities.

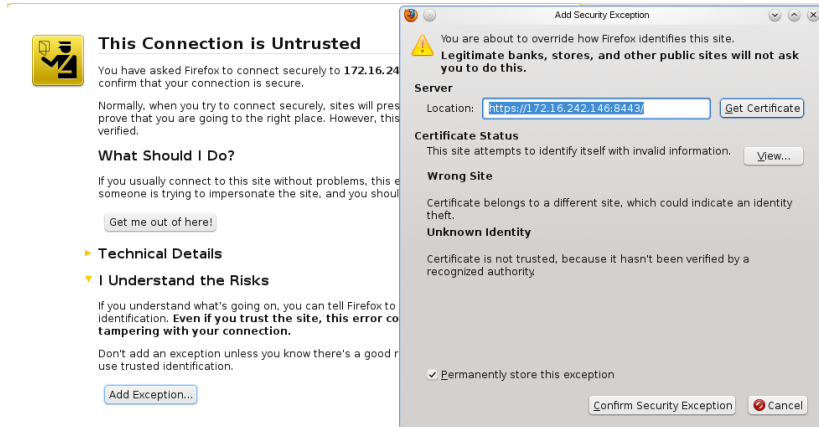


Figure 54 - Certificate acquisition

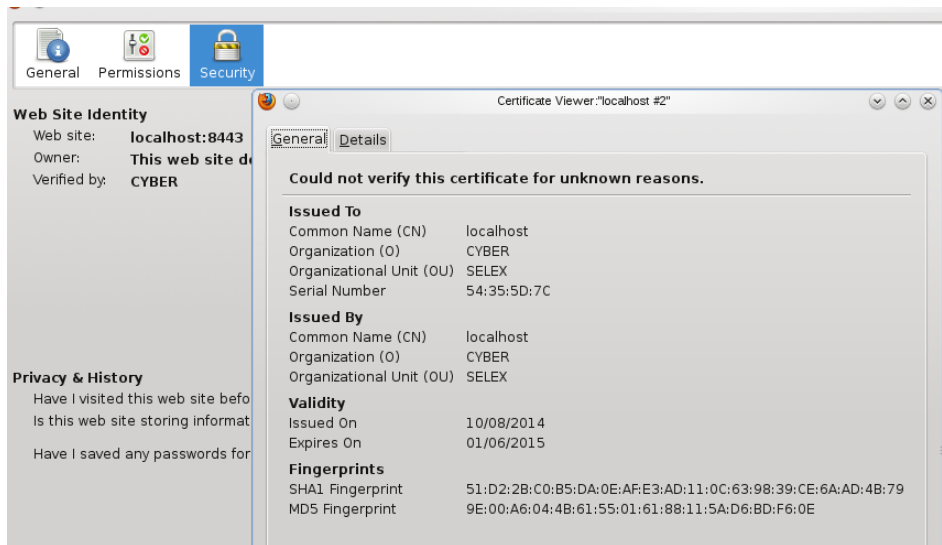


Figure 55 - Certificate details

In order to avoid automatic exchange, the certificate of the SMGW can be manually exported, as shown in Figure 56, by using the procedure in the browser.

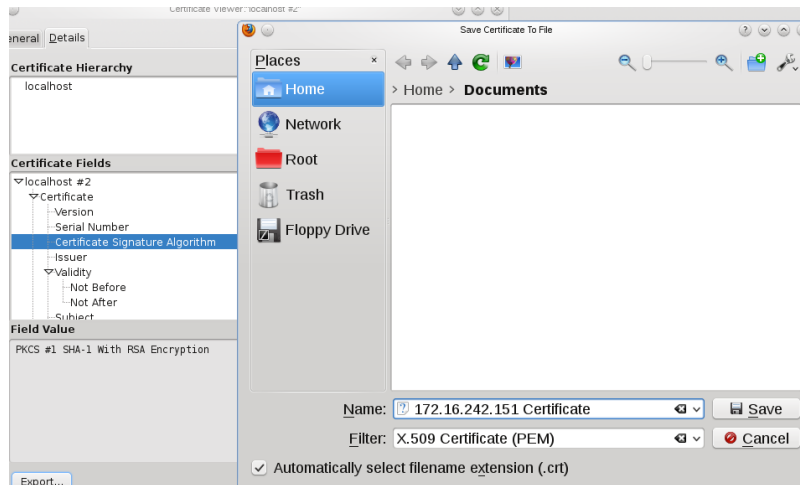


Figure 56 – X.509 Certificate export

Without certificates the message exchange between SMGWs doesn't work. Using the Poster application, the message published from 172.16.242.151 machine towards 172.16.242.146 is not consumable because the application remains in response wait status as shown in Figure 57.

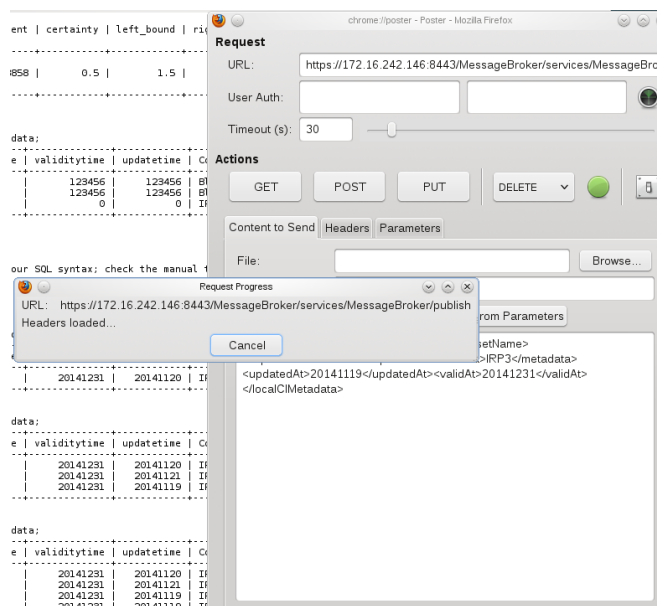


Figure 57 - Sending messages without certificates

The Figure 58 shows the situation where the machine 172.16.242.151 sends a message towards 172.16.242.146. As shown in the left side of the image, the message is stored in the local machine, in this case 172.16.242.151, and is forwarded towards the remote machine using the Poster application. The dump of the database on the remote machine (the right side of the picture) shows all the messages stored.

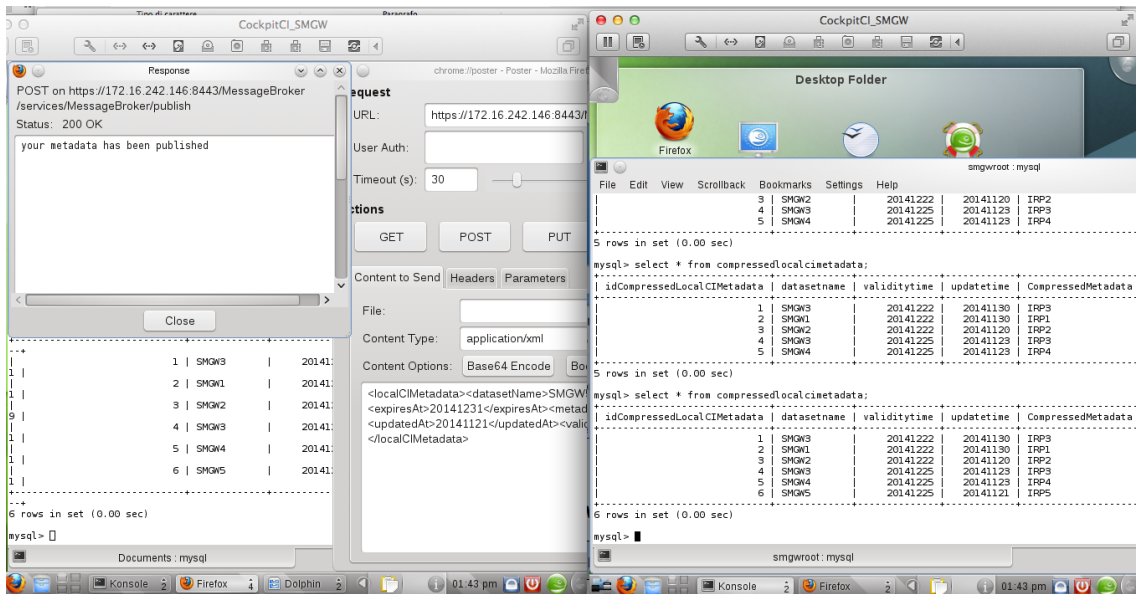


Figure 58 - Correct message exchange between two SMGWs

When an incorrect protocol is used, as shown in Figure 59, the poster application reports an encrypted message coming from the remote machine and the database is not populated with the new message.

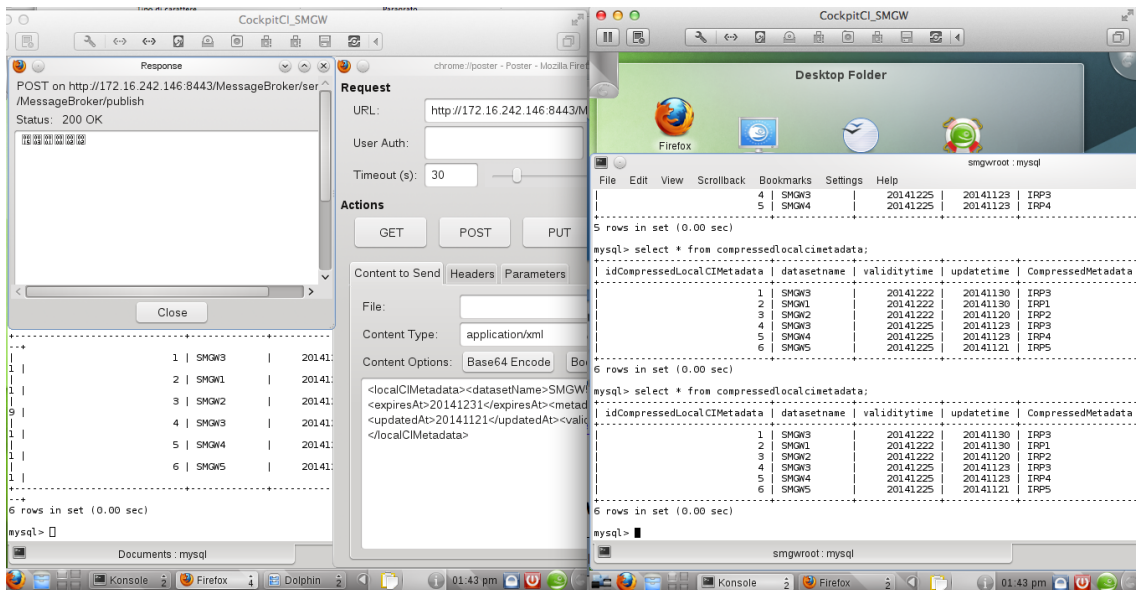


Figure 59 - Incorrect use of Http protocol in sending messages

4 System integration test

This section describes the tests performed to validate the integration among the main components of the CockpitCI tool: the Detection Layer (DL), the Secure Mediation Gateway (SMGW) and the Integrated Risk Predictor (IRP).

The components are deployed as follows:

- The DL is deployed in the laboratory of UC, Coimbra (Portugal);
- The SMGW is deployed in the laboratory of Roma TRE, Rome (Italy);
- The IRP is deployed in the laboratory of Roma TRE, Rome (Italy).

The two laboratories are connected via Internet on a virtual private network (VPN) that extends a private network across a public network; therefore, the three main components of the CockpitCI are on a virtual LAN.

The report describes how a security event is detected by the DL, which generates an IDMEF message; the IDMEF message is propagated through the SMGW to the IRP; the IRP receives the IDMEF message, performs a risk prediction cycle and outputs the results of its elaboration. This test meets the validation of the following requirement:

Requirement id	Short description	Implementation	Required features of SMGW
SMN_1.	The Secure Mediation Network <i>shall</i> provide near real-time secure, reliable and available data exchange between the Detection Layer and the IRP, as well as among different CIs	A SOA-based architecture for data exchange has been adopted to fulfil the requirement.	An IDMEF file should be transferred from DL to IRP

The following cyber attacks will be considered:

- Scouting: Network FIN SCAN;
- DOS (Denial Of Service): Network SYN Flood;
- MITM (Man In The Middle Attack): spp_arpspoof: ARP Cache Overwrite;

In addition, we will describe:

- A fault;
- Hybrid situation where a fault occurs at the same time of a cyber attack.

For each different attack, we will describe the processing, input and output of each of the main components of the CockpitCI tool.

We will give also the risk values of two different reconfigurations (FISRs, Fault Isolation Service Restoration) on the network in response of a fault.

4.1 Configuration of main components

In this section, we report the configuration of the main components of the CockpitCI tool during the tests.

4.1.1 DL configuration

The architecture of testbed scenario of the University of Coimbra was deployed to match the PIDS architecture and illustrated by Figure 60. It contains three main networks/scopes (IT, Operation and Field) and an additional Management network to connect all of the CockpitCI detection layer components to the SMP.

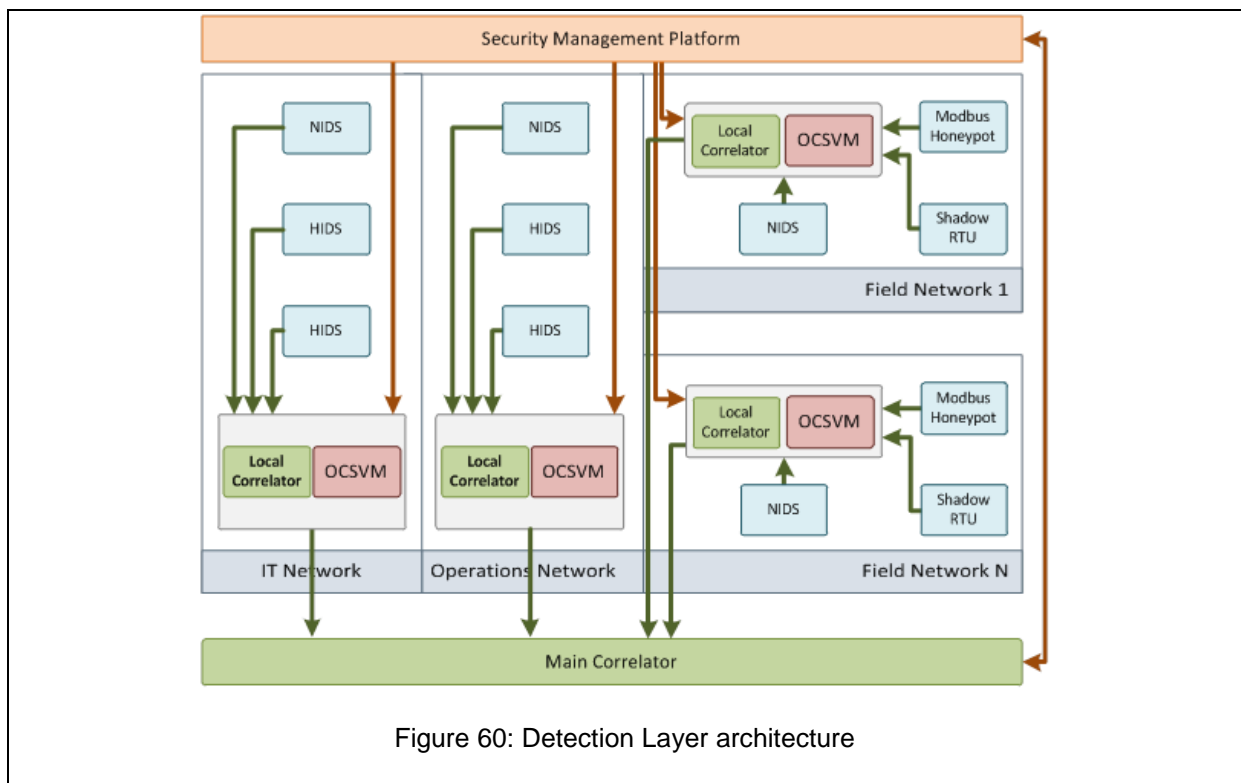
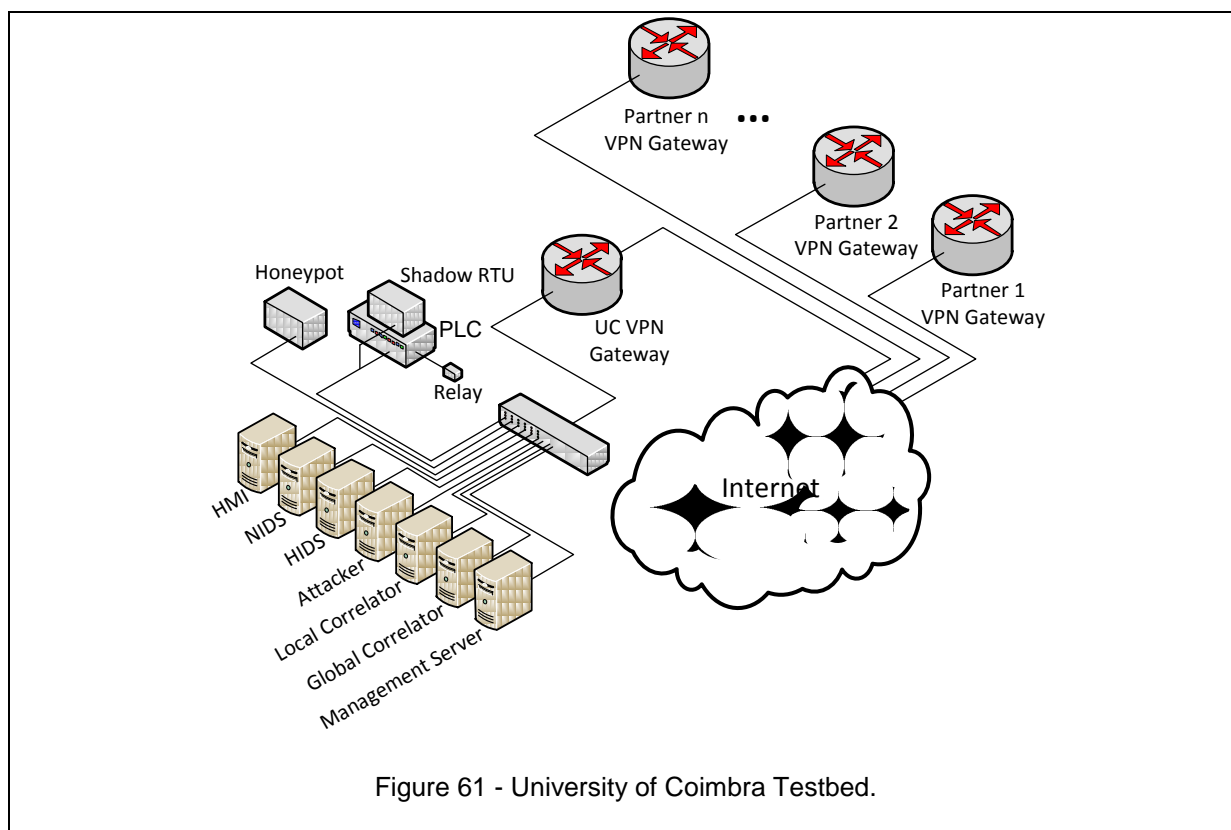


Figure 60: Detection Layer architecture

The testbed deployed at University of Coimbra, illustrated in Figure 61, comprises components from the Field and Operational scopes as well as the management network to connect the former components to the management server, which is part of the SMP. The remaining scopes of the CockpitCI detection layer should behave and integrate in a similar way. The field network contains several detection agents (some of them are SCADA specific), and the correlators (both local and global), and also an HMI and a PLC representing a small SCADA system.



The complete list of the currently deployed components is comprised by the following elements:

- **Honeypot** – SCADA honeypot, acting as a PLC simulator to detect unwanted communications.
- **Shadow RTU** - prototype deployed behind an Ethernet TAP is used to collect and analyze all the traffic to the PLC without being noticed.
- **PLC/RTU** - A Schneider PLC, Modicon M340 connected to a relay simulating an instance of a power grid station.
- **HMI** - Virtual Machine running Windows 7 to interface the PLC, reading and writing the PLC registers.
- **Local Correlator** - Virtual Machine running CentOS 6.5 hosting a correlator instance able to correlate local events.
- **Global Correlator** - Virtual Machine running CentOS 6.5 hosting a correlator instance able to correlate global events and forward them to the upper layers.
- **OCSVM System** – Virtual Machine hosting the OCSVM component for the field network.
- **Management Server** - Virtual Machine running CentOS 6.5 used to track and manage all the other agents.
- **NIDS** - Virtual Machine running CentOS 6.5 capable to detect some network based attacks.
- **HIDS** - Virtual Machine running CentOS 6.5 capable to detect some host based attacks.
- **Attacker** - Virtual Machine running CentOS 6.5 with access to all networks, meaning a compromised environment.
- **VPN Gateway** - Virtual Machine running Fedora 16 connecting the testbed of University of Coimbra and the testbeds of the remaining partners.

All the components have access to both Field and Management network. In general, the Field network is used for SCADA specific communications such as communications between the HMI and the PLC. This network is also used to launch attacks on the SCADA system, e.g., a MITM attack between the HMI and the PLC, or an interaction between an intruder and the honeypot. However, the Management network is used for management functions such events reporting between a detection agent and the correlators, and agent configuration.

4.1.2 SMGW configuration

In integrated test configuration, the SMGW is installed on a server in Roma TRE and is running on a virtual machine using 172.16.226.224 IP address. The services are available by means of the Tomcat server and are addressable using the following URI for REST messages coming from DL and SCADA Adaptor:

[https:// 172.16.226.224:8443/CockpitAdaptor/services/adaptorentities/addFile](https://172.16.226.224:8443/CockpitAdaptor/services/adaptorentities/addFile)

[https:// 172.16.226.224:8443/CockpitDL/services/idmefentities/addFile](https://172.16.226.224:8443/CockpitDL/services/idmefentities/addFile)

And the following URI for SOAP messages to IRP:

<https://localhost:8443/SMN-IRP/IRPPort>

The SMGW stores the incoming messages from DL and SCADA Adaptor in an internal database and sends selectively the contents to IRP according to the invoked method published by means a WSDL. The database for DL and SCADA Adaptor are populated with the IDMEF received by means of the POST methods and the IRP gets the information by means POST choosing if to receive all messages stored or only the new ones (flagged as not already read). Figure 62 shows an example of dump of database on the SMGW.



Figure 62 - Dump of local database of the SMGW

4.1.3 IRP configuration

In order to show the configuration of the IRP we will use the panels produced by the IRP itself.

The IRP internally employs a model of the power grid, as depicted in Figure 63. TF and CB are the two substations that feed two lines, represented respectively by pink and violet colours. The green small rectangles are open remote-telecontrolled breakers, and instead the red ones are closed remote-telecontrolled breakers.

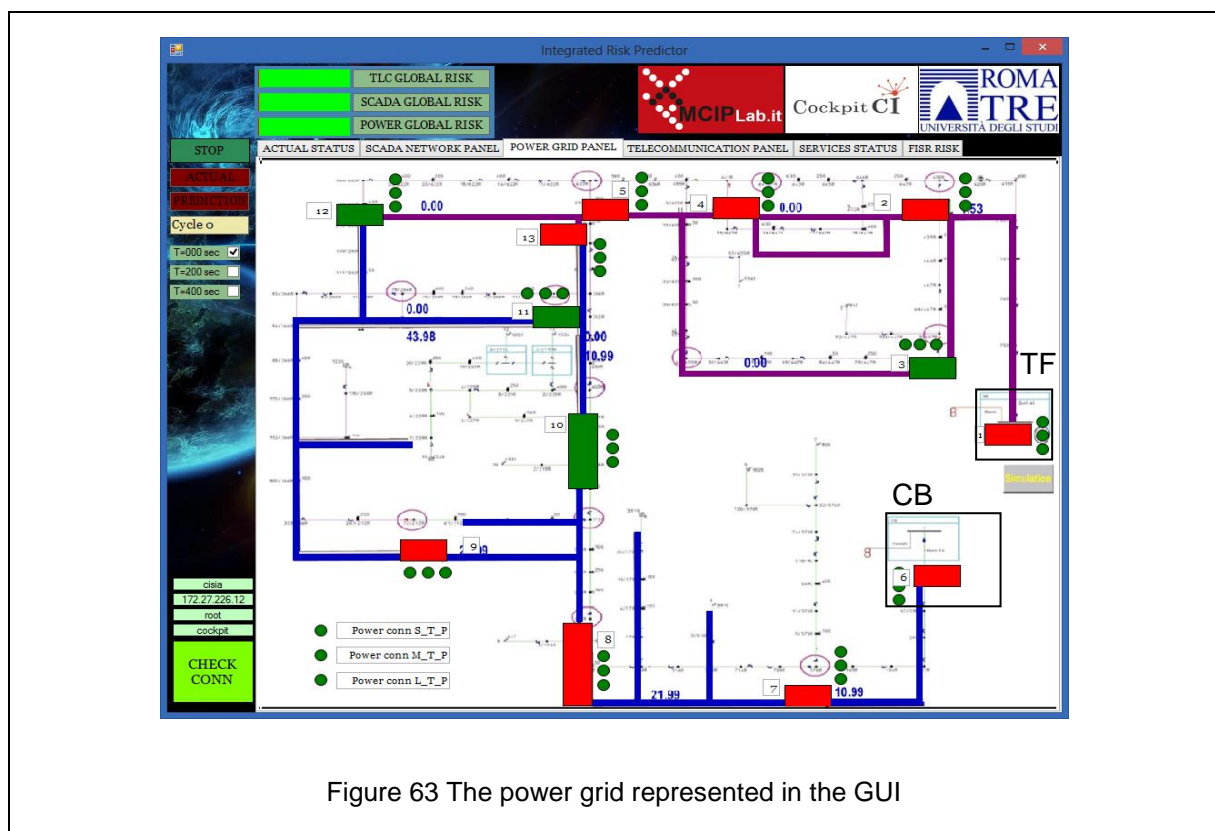


Figure 63 The power grid represented in the GUI

The IRP also internally employs a model of the SCADA network, which is represented in Figure 64. This network is the telecommunication network devoted to transmit data from the Energy Management System (as primary and backup control centre) to the RTUs (Remote Terminal Units) and vice versa. The RTUs are physically connected to the breakers in the power grid.

The other elements of the SCADA network are router and gateways, necessary to transmit packets and to change the transmission protocol.

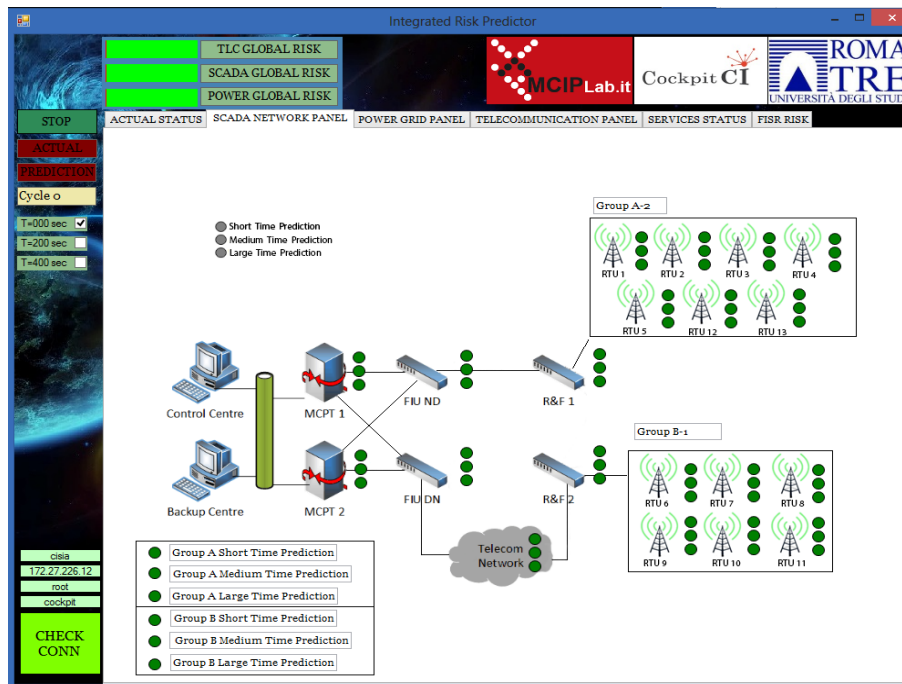


Figure 64 The SCADA network as panel in the GUI

The third panel shows the telecommunication network in Figure 65 and, in the last panel we see the global risk values of two different FISRs procedures that can be performed in response of a fault on the grid located in correspondence of the red cross (see Figure 66).

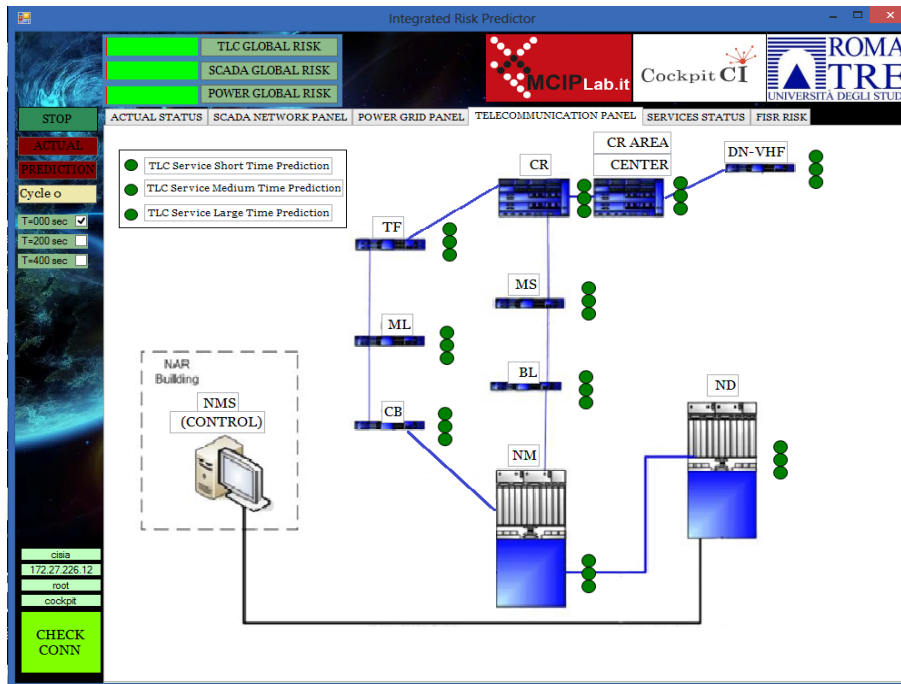


Figure 65 The telecommunication network in the IRP

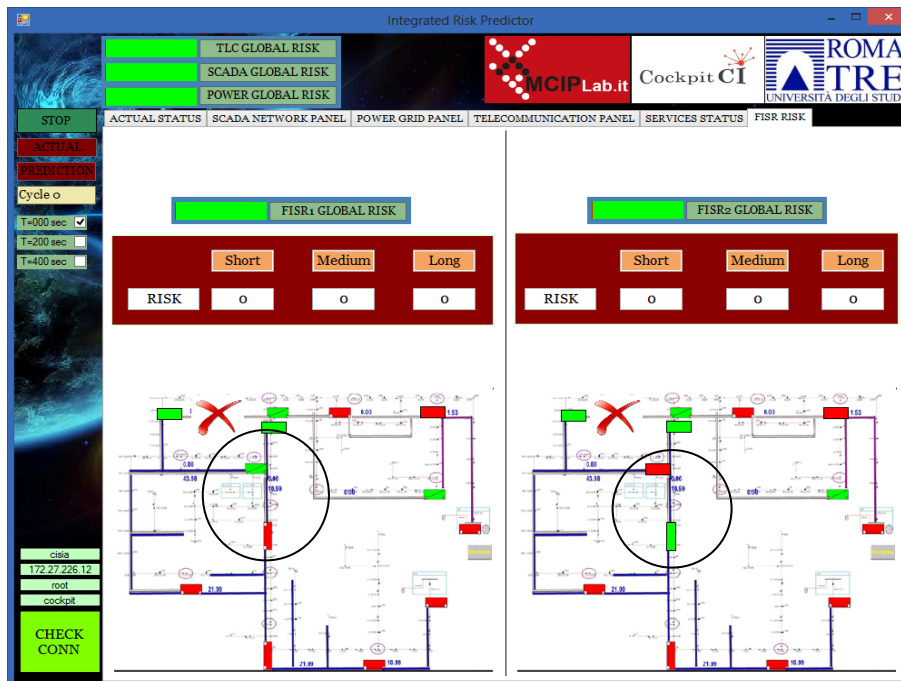


Figure 66 The Risk Prediction for FISR 1 and FISR 2

4.2 Cyber attack #1

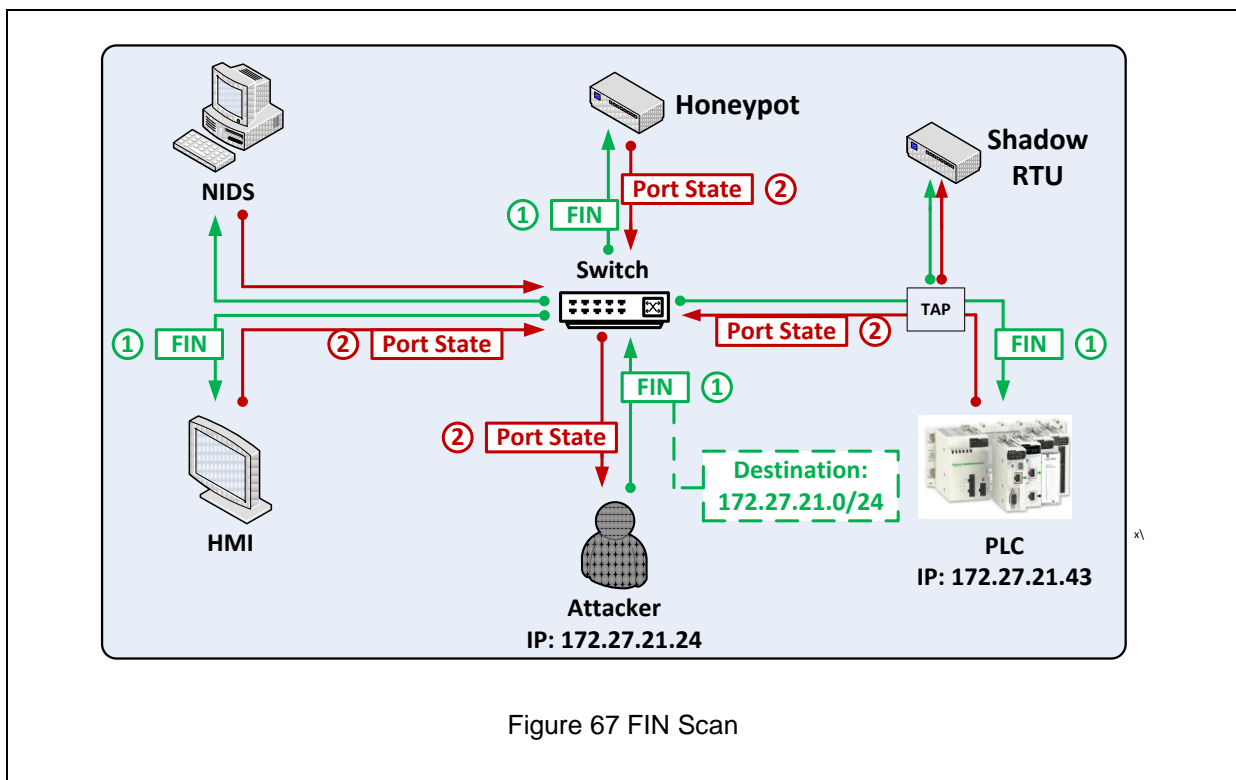
4.2.1 Network FIN SCAN Source: 0 Destination: 172.27.21.43

In this section, we report the operation of the main components of the CockpitCI tool in presence of cyber attack #1.

A typical first step for an attacker is to perform a scouting survey of the infrastructure. In this perspective, network scan procedures are a valuable tool to get an overview of the network elements and its topology.

There are several techniques to perform this on a TCP network, such as SYN or FIN scans, the latter also being known as “stealth scans”, as some firewalls may log SYN (“half-open”) attempts to restricted ports. On the other hand, a FIN packet sent to a closed port on certain hosts (mostly UNIX-based platforms and most PLC/RTU devices) will pass undetected, generating a *RST* response which resets the connection or being ignored for an open port. See Figure 67 for graphic description.

This attack must showcase the malware detection capabilities of the CockpitCI tool, through the NIDS, OCSVM and Shadow RTU.



4.2.2 DL operation

The *Nmap* tool was used to perform a fast *FIN* scan to a TCP network and port address range, faking the initial stages of a TCP connection. From an attacker standpoint, this procedure would allow to get information about the hosts and services running in the network.

At the detection layer, this scenario can be detected by a Network IDS (NIDS), the OCSVM module and the SCADA Honeypot. In addition, the Shadow RTU can be effective in these situations, as it is able to passively intercept the traffic going to the PLC (see Figure 5).

The attack was implemented as shown in Figure 68, using the Nmap tool.

```

[root@attacker]# /root/attack1.sh

Starting Nmap 5.51 ( http://nmap.org ) at 2014-11-26 17:38 IST
Nmap scan report for 172.27.21.41
Host is up (0.00024s latency).
PORT      STATE      SERVICE
22/tcp    open|filtered ssh
80/tcp    open|filtered http
100/tcp   open|filtered newacct
101/tcp   open|filtered hostname
502/tcp   open|filtered asa-appl-PROTO
MAC Address: 00:0D:22:04:09:DD (Unitronics)

Nmap done: 1 IP address (1 host up) scanned in 1.21 seconds
  
```

Figure 68 Launching a fast FIN scan via Nmap

This command generated the network trace shown in Figure 69, which is fed into the OCSVM and NIDS components.

```

[root@nids]#tcpdump -nnNeti eth0 'tcp[tcpflags] & tcp-fin !=0'
17:39:40.754402 IP 172.27.21.24.35310 > 172.27.21.41.http: Flags [F], seq
3038984269, win 4096, length 0
17:39:40.754410 IP 172.27.21.24.35310 > 172.27.21.41.ssh: Flags [F], seq
3038984269, win 2048, length 0
17:39:40.754415 IP 172.27.21.24.35310 > 172.27.21.41.newacct: Flags [F], seq
3038984269, win 4096, length 0
17:39:40.754422 IP 172.27.21.24.35310 > 172.27.21.41.asa-appl-PROTO: Flags [F],
seq 3038984269, win 4096, length 0
17:39:40.754429 IP 172.27.21.24.35310 > 172.27.21.41.hostname: Flags [F], seq
3038984269, win 4096, length 0
17:39:41.805577 IP 172.27.21.24.35311 > 172.27.21.41.hostname: Flags [F], seq
3038918732, win 3072, length 0
17:39:41.805589 IP 172.27.21.24.35311 > 172.27.21.41.asa-appl-PROTO: Flags [F],
seq 3038918732, win 3072, length 0
17:39:41.805593 IP 172.27.21.24.35311 > 172.27.21.41.newacct: Flags [F], seq
3038918732, win 1024, length 0
17:39:41.805598 IP 172.27.21.24.35311 > 172.27.21.41.ssh: Flags [F], seq
3038918732, win 1024, length 0
  
```

```
17:39:41.805605 IP 172.27.21.24.35311 > 172.27.21.41.http: Flags [F], seq 3038918732, win 2048, length 0
```

Figure 69 FIN scan network trace

Similarly, the SCADA Honeypot and the Shadow RTU also detect a similar pattern. Finally, the correlator analyses the received events and produces the IDMEF message pictured in Figure 70.

```
<?xml version="1.0" ?>
<idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef">
  <idmef:Alert messageid="1233">
    <idmef:Analyzer name="snort" analyzerid="snort-agent-1">
      <idmef:Node category="unknown" ident="0">
        <idmef:location>Field Network</idmef:location>
        <idmef:name>Snort NIDS</idmef:name>
      </idmef:Node>
    </idmef:Analyzer>
    <idmef:CreateTime ntpstamp="0x5473403c.0xb9360c00">2014-11-24T16:27:08.723481</idmef:CreateTime>
    <idmef:Source spoofed="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv4-addr" ident="0">
          <idmef:address>172.27.21.24</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>48770</idmef:port>
      </idmef:Service>
    </idmef:Source>
    <idmef:Target decoy="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv4-addr" ident="0">
          <idmef:address>172.27.21.41</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>502</idmef:port>
      </idmef:Service>
    </idmef:Target>
    <idmef:Classification text="Network FIN SCAN" ident="0"/>
  </idmef:Alert>
</idmef:IDMEF-Message>
```

Figure 70 IDMEF message for a FIN scan

4.2.3 SMGW operation

The SMGW receives the IDMEF file from the DL containing information regarding the flooding attack against the HMI, stores it in the local database and forwards it as soon as the IRP performs a GET request. Figure 71 shows the dump of the local database of the SMGW. It contains the IDMEF and the related information such as date, time and a field indicating if the message has been already forwarded towards the IRP.


```
| 2014-11-24 17:41:08.35 | 1 |
| 37 | 8 | DL1 | Power | <?xml version="1.0" encoding="UTF-16"?>
<idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef">
  <idmef:Alert messageid="1233">
    <idmef:Analyzer name="snort" analyzerid="snort-agent-1" >
      <idmef:Node category="unknown" ident="0">
        <idmef:location>Field Network</idmef:location>
        <idmef:name>Snort NIDS</idmef:name>
      </idmef:Node>
    </idmef:Analyzer>
    <idmef:CreateTime ntpstamp="0x5473403c.0xb9360c00">2014-11-
24T16:27:08.723481</idmef:CreateTime>
    <idmef:Source ident="0" spoofed="unknown">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv4-addr" ident="0">
          <idmef:address>172.27.21.24</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>48770</idmef:port>
      </idmef:Service>
    </idmef:Source>
    <idmef:Target decoy="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv4-addr" ident="0">
          <idmef:address>172.27.21.41</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>502</idmef:port>
      </idmef:Service>
    </idmef:Target>
    <idmef:Classification ident="0" text="Network FIN SCAN" ident="0"/>
  </idmef:Alert>
</idmef:IDMEF-Message>
```

Figure 71 - Dump of the SMGW database in case of Network FIN SCAN

4.2.4 IRP operation

The FIN SCAN alarm is collected by the IRP and written into the cyber_attacks DB. The Cyber Simulator is called to assess the potential damages. With this kind of attack, we know that a more dangerous attack could be run very soon but not much damage has been done till now. In Figure 72, the interface to the SMGW of the IRP is shown. The attack is received by the IRP and, after the execution of the Cyber Simulator, CISIA is run. In Figure 73 we see the SCAN cyber-attack fault that is activated as initial state of CISIA.

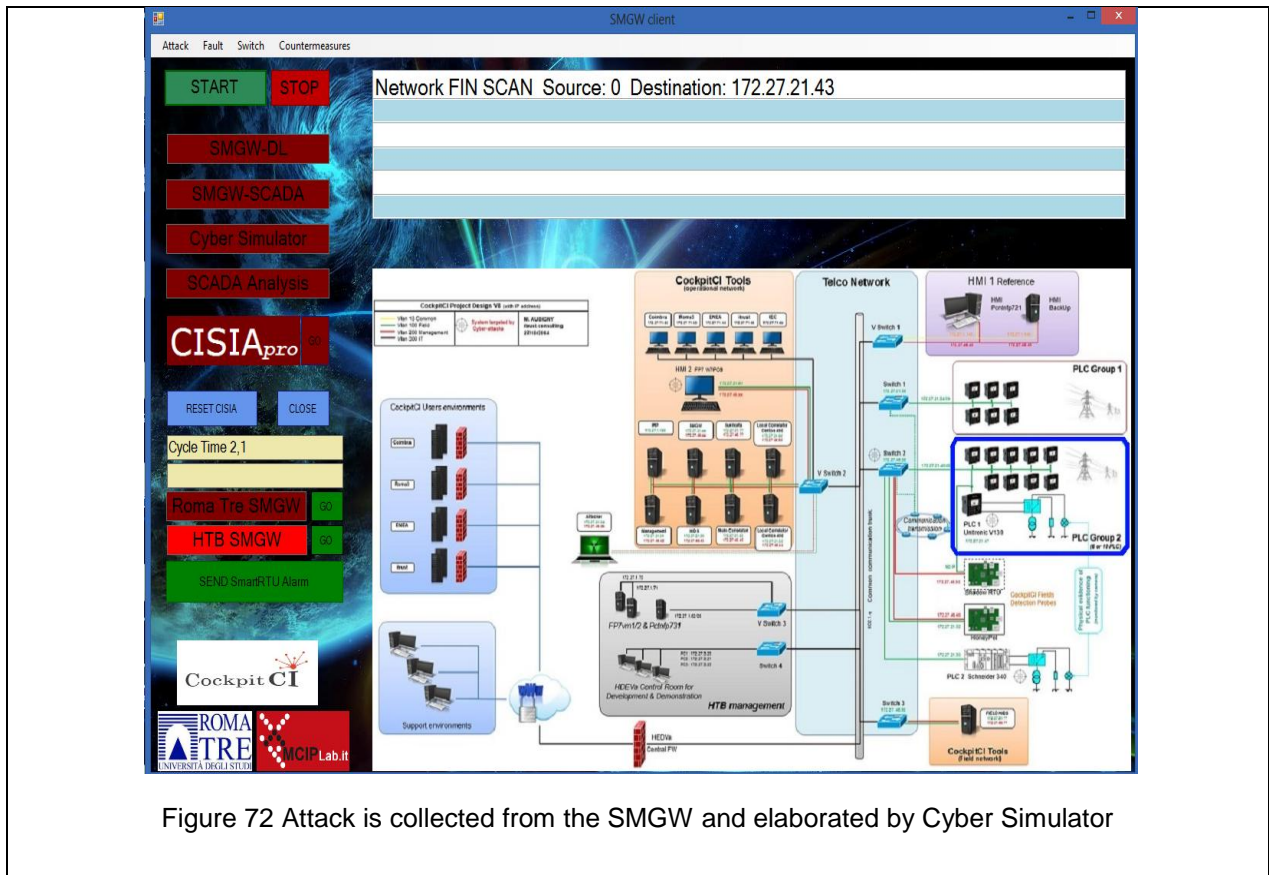


Figure 72 Attack is collected from the SMGW and elaborated by Cyber Simulator

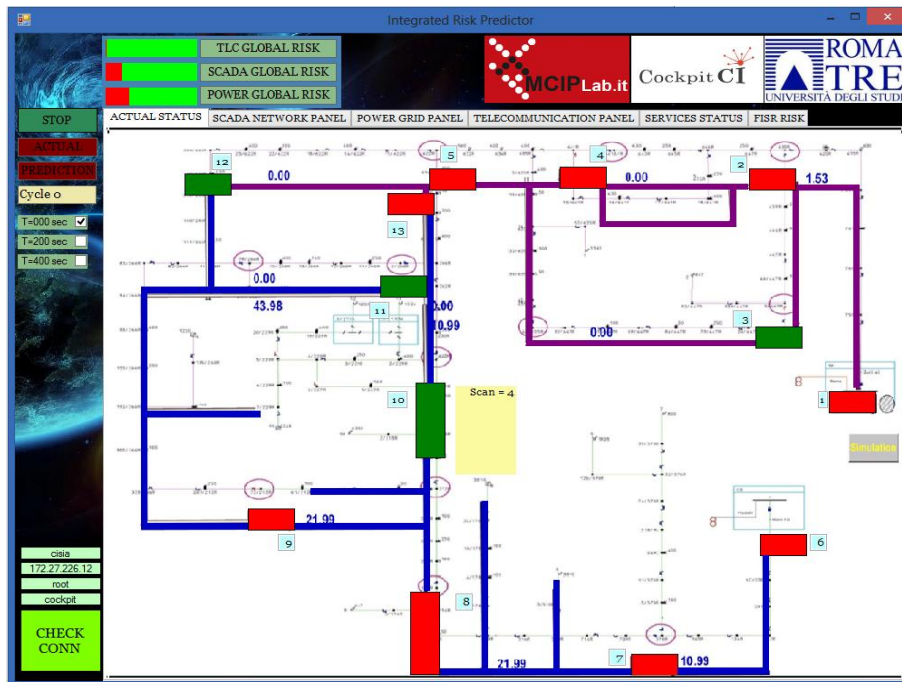


Figure 73 XML SCAN cyber fault is activated on RTU 10 and on R&F2 that is on the path

In Figure 74, the attack is propagated to the other RTUs from the R&F2 node whose Operative Level has been set to 0.5 by the Cyber Simulator. The Cyber Simulator assumes that the SCAN has been transmitted through this path, and so this device could be used to scan other RTUs.

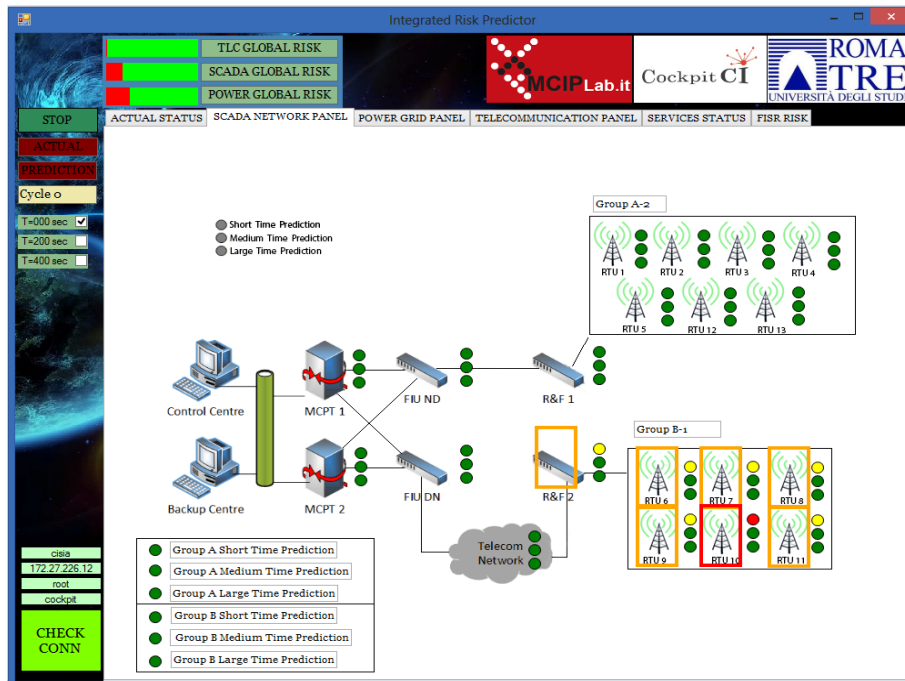


Figure 74 Propagation on the SCADA TLC network

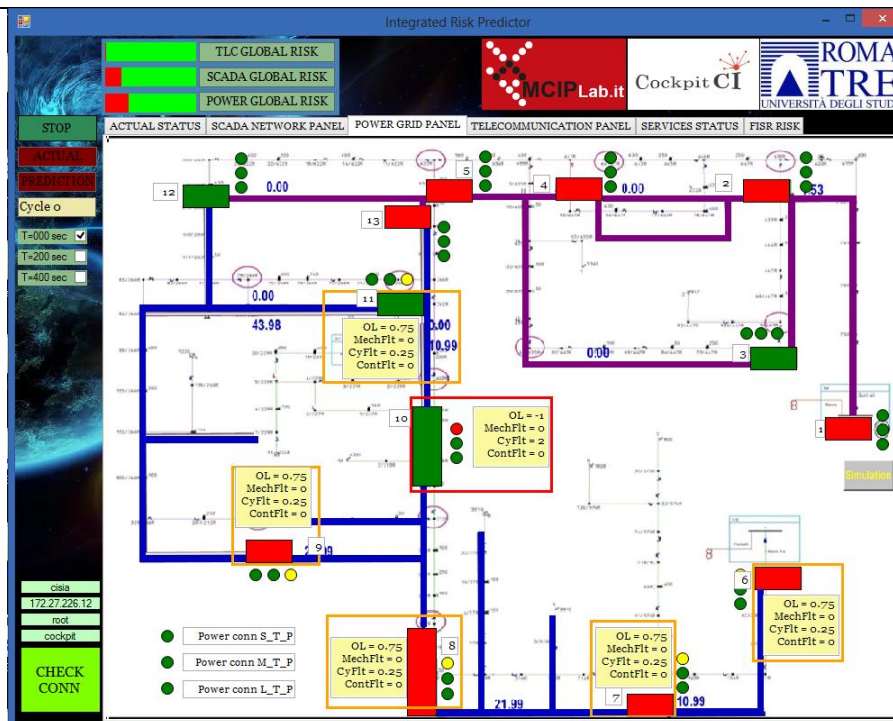


Figure 75 Propagation on the Electric Grid

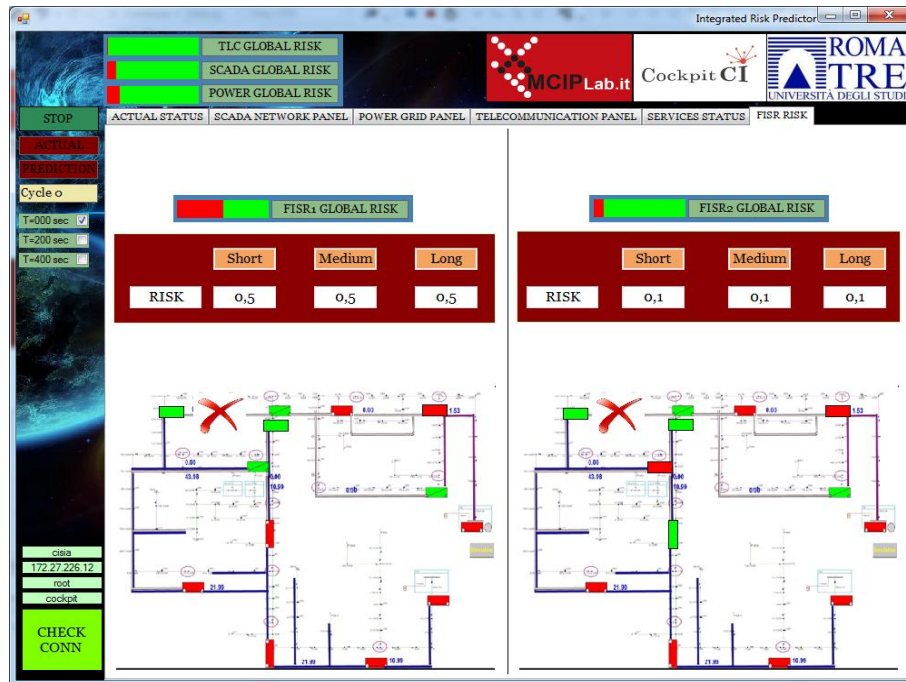


Figure 76 Calculated RISK for FISR 1 and FISR 2

In Figure 75 we see the effect on the elements of the Electric Grid. Orange and red rectangles highlight the compromised RTUs. The operator can immediately realize which is the best FISR procedure to perform, i.e., the one with lower global risk, in Figure 76.

4.3 Cyber attack #2

4.3.1 Network SYN Flood Source: 0 Destination: 172.27.21.38/172.27.21.43

In this section, we report the operation of the main components of the CockpitCI tool in presence of cyber-attack #2.

Denial-of-Service attacks can be implemented using flooding techniques: Smurf or SYN flooding attacks are examples of such techniques. The SYN attack illustrated exploits the nature of TCP connections, by sending a series of SYN packets (signaling the intention to initiate a TCP connection) to the PLC/RTU, to which it responds with a SYN-ACK that is never acknowledged (ACK) by the sender. A SYN flood creates a large numbers of half-open connections on the attacked device (waiting for an ACK or a timeout) that might eventually lead to resource exhaustion, with the device ceasing to answer new requests. A variation of this attack forges raw packets with a false sender, so that the SYN-ACK replies

also flood a target, via backscatter (similarly to a SMURF attack, that uses ICMP packets for such purpose). See Figure 77.

This can showcase the capabilities of NIDS, OCSVM and Shadow RTU.

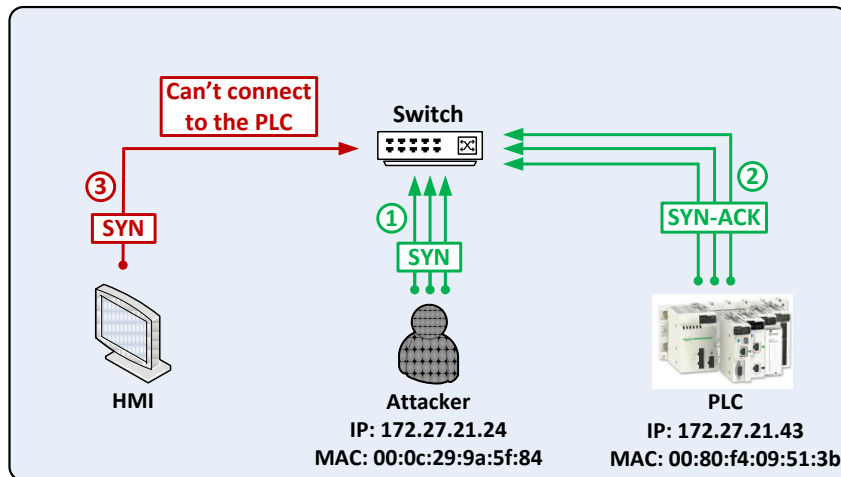


Figure 77 SYN Flood attack

4.3.2 DL operation

In this type of attacks several crafted packets are used in order to exhaust the resources of a specific host. In this case, the attacker sends flood of TCP SYN packets to the PLC/RTU. As a result the PLC/RTU becomes unavailable. The control of it by the HMI interface is not possible during the attack. In this case, both the NIDS and the OCSVM successfully detect an elevated number of packets and report them. Also, the Shadow RTU is able to detect this attack, as pictured in Figure 78.

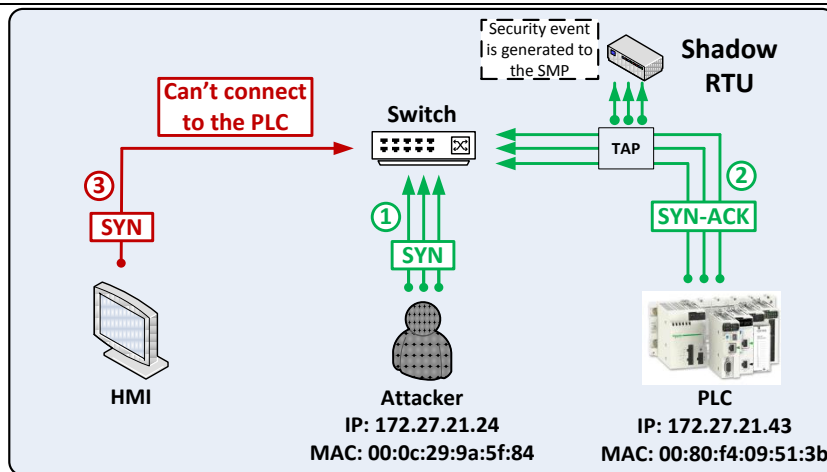


Figure 78 Shadow RTU detecting a SYN flood

To demonstrate the operation of the DL in the presence of this attack, two scenarios were considered: SYN flood against a PLC and against an HMI. For this purpose, the *hping* tool was used to generate a flood against the two targets (see Figure 79).

```
[root@attacker]# /home/coimbra/attack3.sh
HPING 172.27.46.43 (eth0 172.27.46.43): S set, 40 headers + 0 data bytes

--- 172.27.46.43 hping statistic ---
10000 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

[root@attacker]# /home/coimbra/attack2.sh
HPING 172.27.21.41 (eth1 172.27.21.41): S set, 40 headers + 0 data bytes

--- 172.27.21.41 hping statistic ---
10000 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figure 79 Generating SYN floods using *hping*

The captured network flows for the attacks are reported to both the NIDS and the OCSVM modules.

```
[root@nids-op]# tcpdump -nnNeti eth3
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1920 > 172.27.46.43.9999: Flags [S], seq 1745420796, win 512, length
0
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1921 > 172.27.46.43.9999: Flags [S], seq 1997731855, win 512, length
0
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1922 > 172.27.46.43.9999: Flags [S], seq 1769280820, win 512, length
0
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1923 > 172.27.46.43.9999: Flags [S], seq 204595012, win 512, length 0
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1924 > 172.27.46.43.9999: Flags [S], seq 1778959342, win 512, length
0
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1925 > 172.27.46.43.9999: Flags [S], seq 450781634, win 512, length 0
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1926 > 172.27.46.43.9999: Flags [S], seq 682166286, win 512, length 0
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1928 > 172.27.46.43.9999: Flags [S], seq 1051577808, win 512, length
0
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1927 > 172.27.46.43.9999: Flags [S], seq 94144950, win 512, length 0
00:50:56:bf:65:0e > 00:50:56:bf:7e:8d, ethertype IPv4 (0x0800), length 60:
172.27.46.46.1929 > 172.27.46.43.9999: Flags [S], seq 1700668879, win 512, length
0

[root@nids-field]# tcpdump -nnNeti eth0
```

```

00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2810 > 172.27.21.41.9999: Flags [S], seq 315664722, win 512, length 0
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2811 > 172.27.21.41.9999: Flags [S], seq 2121268813, win 512, length
0
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2812 > 172.27.21.41.9999: Flags [S], seq 1691969372, win 512, length
0
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2813 > 172.27.21.41.9999: Flags [S], seq 758246076, win 512, length 0
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2814 > 172.27.21.41.9999: Flags [S], seq 1873810841, win 512, length
0
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2815 > 172.27.21.41.9999: Flags [S], seq 2070606615, win 512, length
0
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2816 > 172.27.21.41.9999: Flags [S], seq 1869988535, win 512, length
0
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2817 > 172.27.21.41.9999: Flags [S], seq 1935549704, win 512, length
0
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2818 > 172.27.21.41.9999: Flags [S], seq 1333794662, win 512, length
0
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype IPv4 (0x0800), length 60:
172.27.21.24.2819 > 172.27.21.41.9999: Flags [S], seq 1480645771, win 512, length
0
  
```

As a result, the detection layer generates the adequate IDMEF messages (see Figure 80 and Figure 81), to be sent to the SMGW for propagation. These messages provide information about the time of detection, the source(s) and target of the attacks, and the ID of the correlator classification rule used to detect them.

```

<?xml version="1.0" ?>
<idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef">
  <idmef:Alert messageid="1233">
    <idmef:Analyzer name="snort" analyzerid="snort-agent-1">
      <idmef:Node category="unknown" ident="0">
        <idmef:location>Field Network</idmef:location>
        <idmef:name>Snort NIDS</idmef:name>
      </idmef:Node>
    </idmef:Analyzer>
    <idmef:CreateTime ntpstamp="0x54735f0e.0x893c1400">2014-11-
24T18:38:38.536073</idmef:CreateTime>
    <idmef:Source spoofed="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv4-addr" ident="0">
          <idmef:address>172.27.46.46</idmef:address>
        </idmef:Address>
      </idmef:Node>
    <idmef:Service ident="0">
      <idmef:port>3003</idmef:port>
    </idmef:Service>
  </idmef:Alert>
</idmef:IDMEF-Message>
  
```



```

</idmef:Source>
<idmef:Target decoy="unknown" ident="0">
  <idmef:Node category="unknown" ident="0">
    <idmef:Address category="ipv4-addr" ident="0">
      <idmef:address>172.27.46.43</idmef:address>
    </idmef:Address>
  </idmef:Node>
  <idmef:Service ident="0">
    <idmef:port>9999</idmef:port>
  </idmef:Service>
</idmef:Target>
<idmef:Classification text="Network SYN Flood" ident="0"/>
</idmef:Alert>
  
```

Figure 80 IDMEF message for flooding attack against a PLC

```

</idmef:IDMEF-Message>
<?xml version="1.0" ?>
<idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef">
  <idmef:Alert messageid="1233">
    <idmef:Analyzer name="snort" analyzerid="snort-agent-1">
      <idmef:Node category="unknown" ident="0">
        <idmef:location>Field Network</idmef:location>
        <idmef:name>Snort NIDS</idmef:name>
      </idmef:Node>
    </idmef:Analyzer>
    <idmef:CreateTime ntpstamp="0x547345a4.0x63ad6c00">2014-11-24T16:50:12.389365</idmef:CreateTime>
    <idmef:Source spoofed="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv4-addr" ident="0">
          <idmef:address>172.27.21.24</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>3156</idmef:port>
      </idmef:Service>
    </idmef:Source>
    <idmef:Target decoy="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv4-addr" ident="0">
          <idmef:address>172.27.21.41</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>9999</idmef:port>
      </idmef:Service>
    </idmef:Target>
    <idmef:Classification text="Network SYN Flood" ident="0"/>
  </idmef:Alert>
</idmef:IDMEF-Message>
  
```

```
</idmef:Alert>  
</idmef:IDMEF-Message>
```

Figure 81: IDMEF message for flooding attack against the HMI

4.3.3 SMGW operation

The SMGW receives the IDMEF file from the DL containing information regarding the flooding attacks against the PLC (Figure 82) and HMI (Figure 83), performs the storage in the local database and forwards them as soon as the IRP performs a GET request.

```
| 2014-11-24 19:12:39.35 | 1 |  
| 39 | 10 | DL1 | Power | <?xml version="1.0" encoding="UTF-16"?>  
<idmef:IDMef-Message xmlns:idmef="http://iana.org/idmef">  
  <idmef:Alert messageid="1233">  
    <idmef:Analyzer name="snort" analyzerid="snort-agent-1">  
      <idmef:Node category="unknown" ident="0">  
        <idmef:location>Field Network</idmef:location>  
        <idmef:name>Snort NIDS</idmef:name>  
      </idmef:Node>  
    </idmef:Analyzer>  
    <idmef:CreateTime ntpstamp="0x54735f0e.0x893c1400">2014-11-  
24T18:38:38.536073</idmef:CreateTime>  
    <idmef:Source spoofed="unknown" ident="0">  
      <idmef:Node category="unknown" ident="0">  
        <idmef:Address category="ipv4-addr" ident="0">  
          <idmef:address>172.27.46.46</idmef:address>  
        </idmef:Address>  
      </idmef:Node>  
      <idmef:Service ident="0">  
        <idmef:port>3003</idmef:port>  
      </idmef:Service>  
    </idmef:Source>  
    <idmef:Target decoy="unknown" ident="0">  
      <idmef:Node category="unknown" ident="0">  
        <idmef:Address category="ipv4-addr" ident="0">  
          <idmef:address>172.27.46.43</idmef:address>  
        </idmef:Address>  
      </idmef:Node>  
      <idmef:Service ident="0">  
        <idmef:port>9999</idmef:port>  
      </idmef:Service>  
    </idmef:Target>  
    <idmef:Classification text="Network SYN Flood" ident="0"/>  
</idmef:Alert>
```

Figure 82 – SMGW database dump in case of attack against the PLC

```

| 2014-11-24 18:12:39.35 | 1 |
| 38 | 9 | DL1 | Power | <?xml version="1.0" encoding="UTF-16"?>
</idmef:IDMEF-Message>
<?xml version="1.0" ?>
<idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef">
  <idmef:Alert messageid="1233">
    <idmef:Analyzer name="snort" analyzerid="snort-agent-1">
      <idmef:Node category="unknown" ident="0">
        <idmef:location>Field Network</idmef:location>
        <idmef:name>Snort NIDS</idmef:name>
      </idmef:Node>
    </idmef:Analyzer>
    <idmef:CreateTime ntpstamp="0x547345a4.0x63ad6c00">2014-11-
24T16:50:12.389365</idmef:CreateTime>
    <idmef:Source spoofed="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv4-addr" ident="0">
          <idmef:address>172.27.21.24</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>3156</idmef:port>
      </idmef:Service>
    </idmef:Source>
    <idmef:Target decoy="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv4-addr" ident="0">
          <idmef:address>172.27.21.41</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>9999</idmef:port>
      </idmef:Service>
    </idmef:Target>
    <idmef:Classification text="Network SYN Flood" ident="0"/>
  </idmef:Alert>
</idmef:IDMEF-Message>

```

Figure 83 – SMGW database dump in case of attack against the HMI

4.3.4 IRP operation

In this case, when the attack is detected we know that a particular RTU will not respond to commands. For this reason, when we set at 1 the DOS (Denial Of Service) attack for the attacked RTU, the availability of the device falls to zero. In addition, the DOS attack flag for R&F2 is activated to a value of 0.5 to point out that packets have been transmitted through it.

In Figure 84 the output of the console is reported when a SYN flood is received from the Detection Layer. In Figure 85 the result of the propagation in CISIA shows the RTUs that

could be compromised and in Figure 86 the console for the Electric Operator shows the RTUs status and which is the portion of the Electric Grid that is under attack.

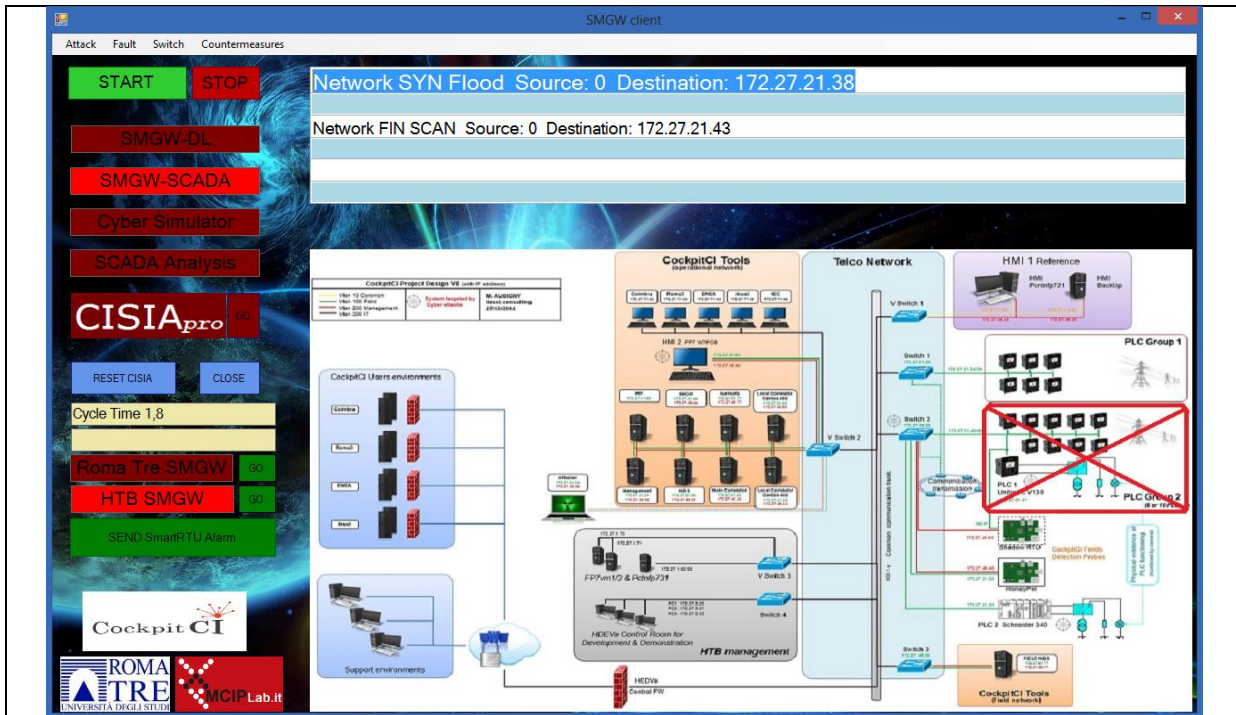


Figure 84 SYN FLOOD attack is collected from SMGW and elaborated from Cyber Simulator

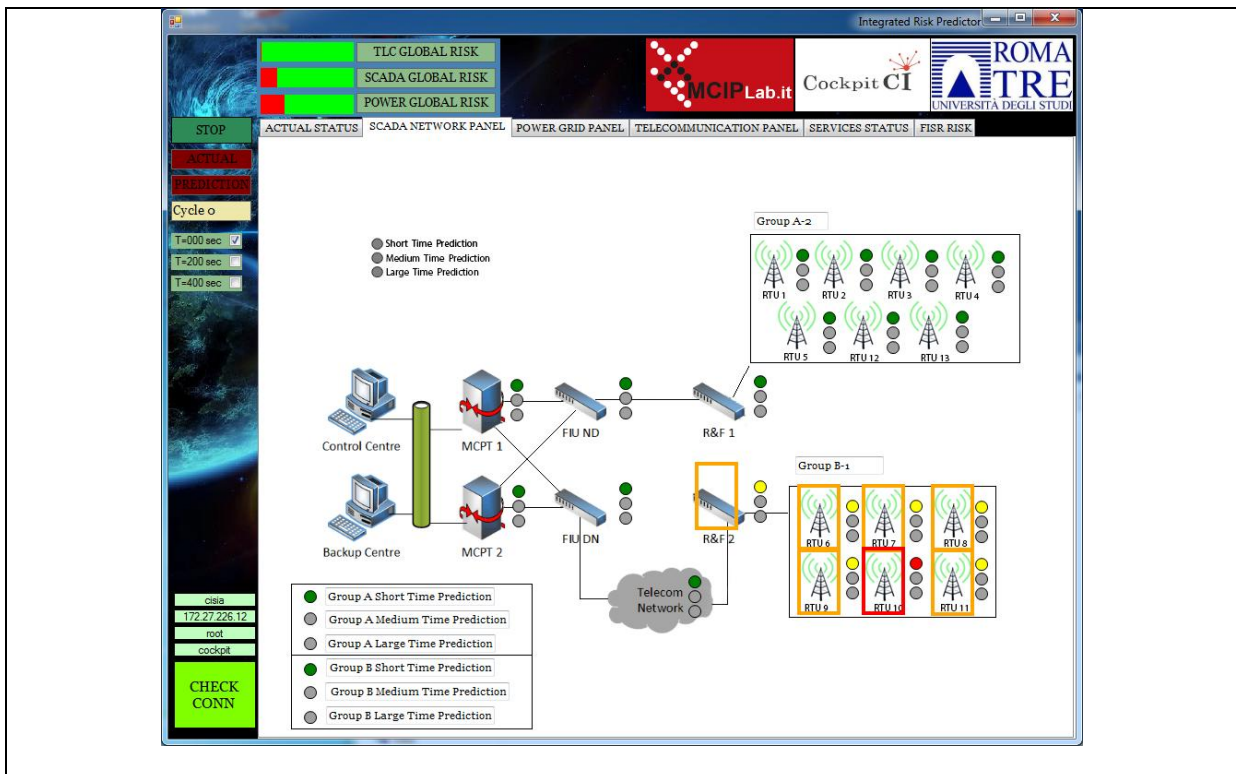


Figure 85 CISIA propagation of the attack

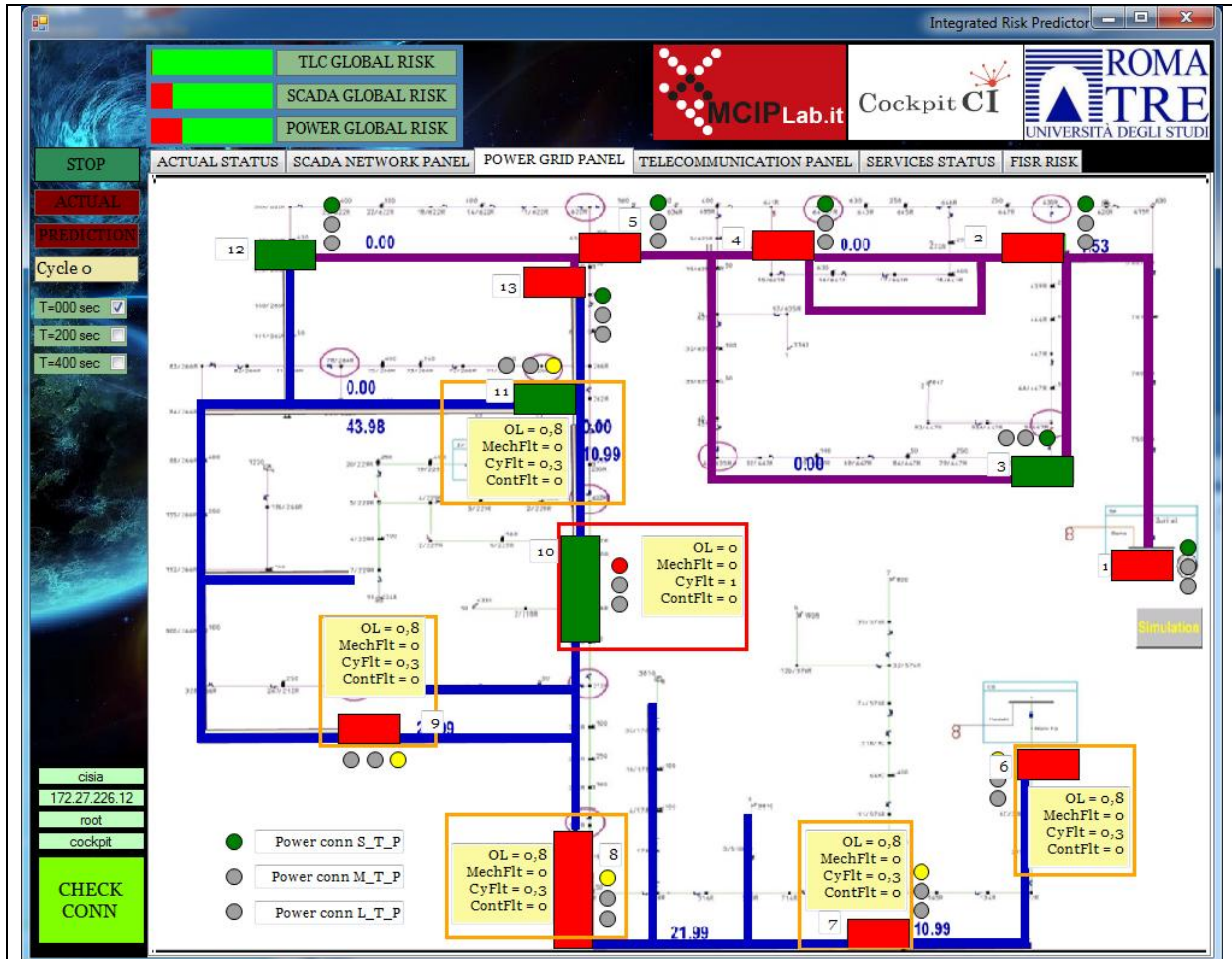


Figure 86 Propagation on the Electric Grid

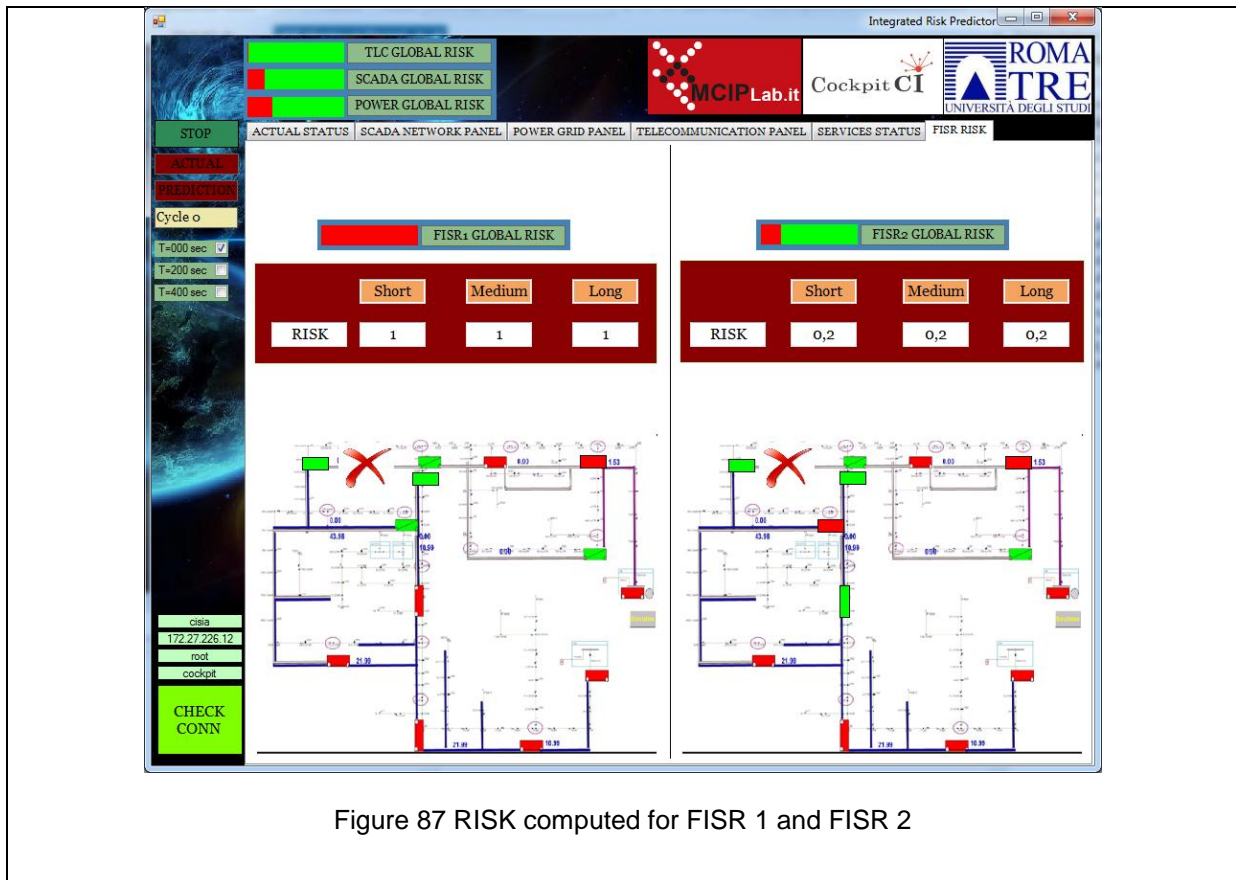


Figure 87 RISK computed for FISR 1 and FISR 2

In Figure 87 the risk of the two FISR procedures is computed and, as we can see, only the one on the right is acceptable.

4.4 Cyber-attack #3 spp_arp spoof:

4.4.1 ARP Cache Overwrite Attack; Source: 0 Destination: 172.27.21.43

In this section, we report the operation of the main components of the CockpitCI tool in presence of cyber-attack #3.

In the first stage of the ARP poisoning MITM, the attacker generates a series of unrequested ARP replies for both the HMI and the PLC, poisoning the local ARP caches. In this way, the MAC address of the attacker system becomes associated with the IP of the HMI for the PLC and the IP of the PLC for the HMI, respectively. This means that further interaction attempts from the HMI to the PLC will be redirected to the attacker system, and vice-versa.

In the second stage, the attacker provides a fake device for the HMI to interact with, using a Modbus simulator, programmed with information obtained from a previous survey or a script that replies with protocol interactions previously recorded, corresponding to a normal operation scenario. See Figure 88 for this attack.

This attack must display the malware detection capabilities of the CockpitCI tool, through the NIDS, OCSVM and Shadow RTU.

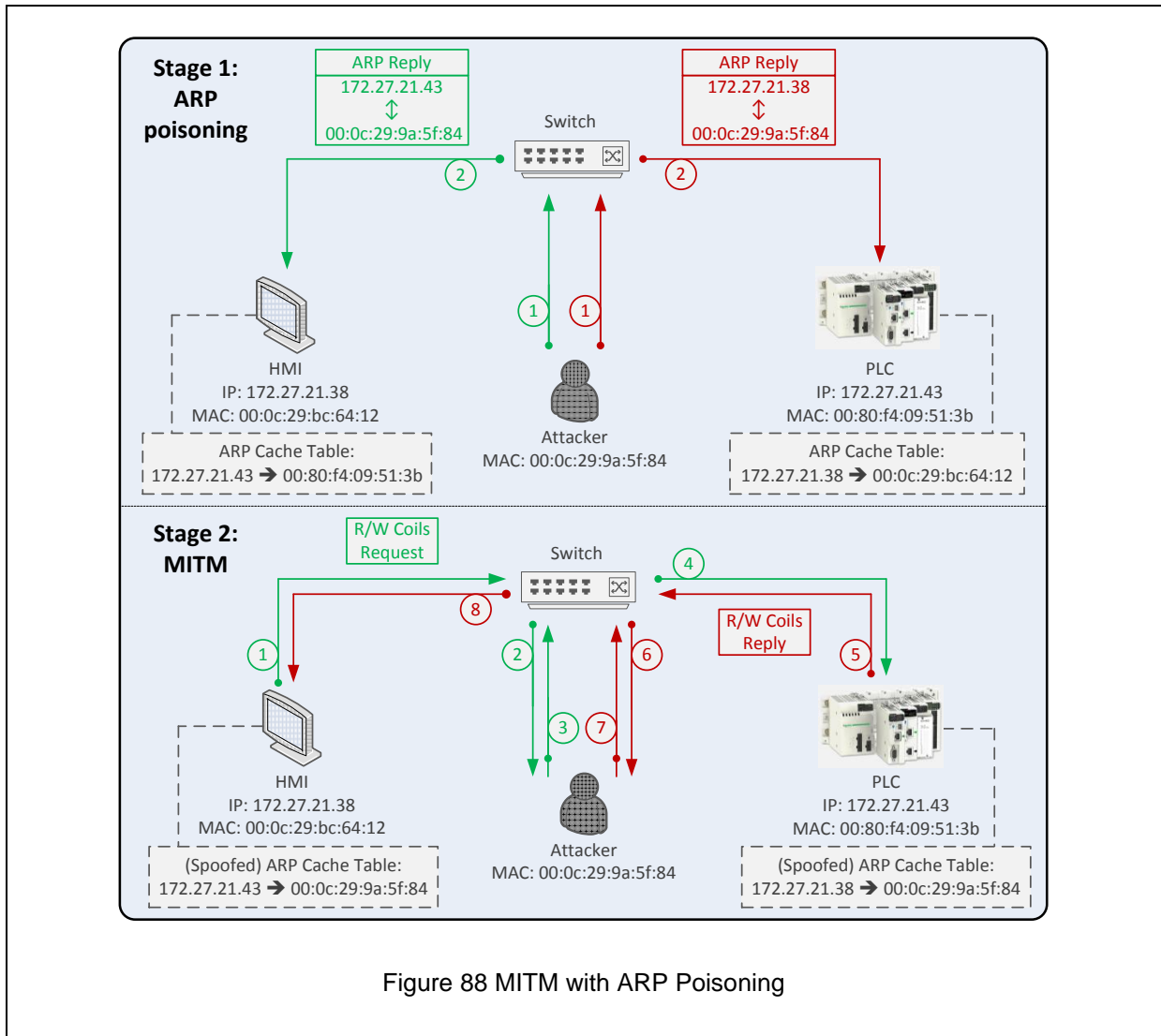
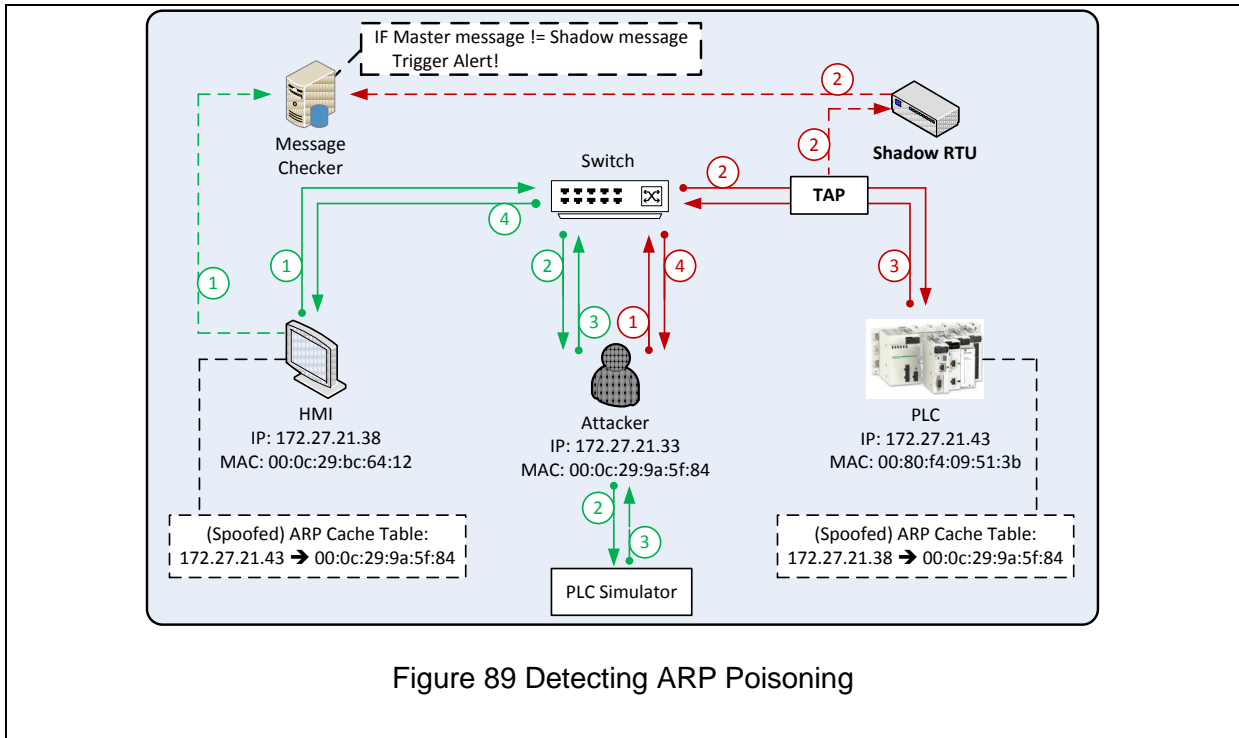


Figure 88 MITM with ARP Poisoning

4.4.2 DL operation

In this situation, the DL provides detection capabilities thanks to the OCSVM engine, the NIDS and the Shadow RTU.

Particularly, the Shadow RTU can be effective against this attack as it is configured with the IP/MAC addresses of the systems that are authorized to interact with the monitored device – this same strategy is equally effective for the real-time protocol manipulation scenario. In addition, thanks to the Shadow RTU Message Checker, it is possible to create a closed loop to monitor the integrity of command and control. The Shadow RTU provides a replica of all the interactions that arrive at the PLC back to a Message Checker system that also receives a copy of all interactions that are captured from HMI/Master Station side. Therefore, the Message Checker is able to verify the integrity of the control flow in an end-to-end basis, overcoming the most sophisticated MITM scenarios.



The attack is launched using a custom script (see Figure 90) that makes use of *ethercap*, together with *iptables* redirection rules.

```

[root@attacker]#/home/mitm.sh
0:50:56:bf:2b:cf 0:d:22:4:9:dd 0806 42: arp reply 172.27.21.21 is-at
0:50:56:bf:2b:cf
0:50:56:bf:2b:cf 0:50:56:bf:1:c3 0806 42: arp reply 172.27.21.41 is-at
0:50:56:bf:2b:cf
  
```

Figure 90: Launching the first stage of the MiTM attack – ARP poisoning

In the second stage, the attacker intercepts connections in real-time using a packet manipulation tool (such as SCAPY) to perform session hijacking on the TCP connection. In alternative, the attacker might provide a fake device for the HMI to interact with, using a Modbus simulator, programmed with information obtained from a previous survey or a script that replies with protocol interactions previously recorded, corresponding to a normal operation scenario – Figure 90 depicts such an attack, provided to showcase the Shadow RTU MITM detection capabilities. After the ARP poisoning is successfully implemented, the attacker system runs a PLC simulator to keep the HMI unaware of the real PLC status.

The captured network flows for the attacks are reported to both the NIDS and the OCSVM modules, as per Figure 91.

```

[root@nids-op]# tcpdump -nnNeti eth2
00:50:56:bf:2b:cf > 00:50:56:bf:01:c3, ethertype ARP (0x0806), length 60: Reply
172.27.21.41 is-at 00:50:56:bf:2b:cf, length 46

root@nids-field]# tcpdump -nnNeti eth0
  
```

```
00:50:56:bf:2b:cf > 00:0d:22:04:09:dd, ethertype ARP (0x0806), length 60: Reply
172.27.21.38 is-at 00:50:56:bf:2b:cf, length 46
```

Figure 91: Network trace captures for the ARP poisoning attempts

As a result, the detection layer generates the adequate IDMEF messages (see Figure 92), to be send to the SMGW for propagation. These messages provide information about the time of detection, the MAC source(s) and target of the attacks, and the ID of the correlator classification rule used to detect them.

```
<?xml version="1.0" ?>
<idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef">
  <idmef:Alert messageid="1233">
    <idmef:Analyzer name="snort" analyzerid="snort-agent-1">
      <idmef:Node category="unknown" ident="0">
        <idmef:location>Field Network</idmef:location>
        <idmef:name>Snort NIDS</idmef:name>
      </idmef:Node>
    </idmef:Analyzer>
    <idmef:CreateTime ntpstamp="0x54746dc1.0xe78ae400">2014-11-
25T13:53:37.904463</idmef:CreateTime>
    <idmef:Source spoofed="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv6-addr" ident="0">
          <idmef:address>0:0:0:0:0:0:0:0</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>0</idmef:port>
      </idmef:Service>
    </idmef:Source>
    <idmef:Target decoy="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv6-addr" ident="0">
          <idmef:address>0:0:0:0:0:0:0:0</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>0</idmef:port>
      </idmef:Service>
    </idmef:Target>
    <idmef:Classification text="spp_arpspoof: ARP Cache Overwrite Attack"
ident="0"/>
    <idmef:Assessment>
      <idmef:Impact type="other" severity="medium"/>
    </idmef:Assessment>
    <idmef:AdditionalData meaning="bad-unknown" type="string">
      <idmef:string>Potentially Bad Traffic</idmef:string>
    </idmef:AdditionalData>
  </idmef:Alert>
</idmef:IDMEF-Message>
```

Figure 92: IDMEF message for MiTM attack

4.4.3 SMGW operation

The SMGW receives the IDMEF file from the DL containing information regarding the flooding attack against the MiTM, stores it in the local database and forwards it as soon as the IRP performs a GET request. Figure 93 shows the dump of the SMGW local database containing the IDMEF.

```
| 2014-11-25 14:24:38.672 | 1 |
| 40 | 12 | DL1 | Power | <?xml version="1.0" encoding="UTF-16"?>
<?xml version="1.0" ?>
<idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef">
  <idmef:Alert messageid="1233">
    <idmef:Analyzer name="snort" analyzerid="snort-agent-1">
      <idmef:Node category="unknown" ident="0">
        <idmef:location>Field Network</idmef:location>
        <idmef:name>Snort NIDS</idmef:name>
      </idmef:Node>
    </idmef:Analyzer>
    <idmef:CreateTime ntpstamp="0x54746dc1.0xe78ae400">2014-11-
25T13:53:37.904463</idmef:CreateTime>
    <idmef:Source spoofed="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv6-addr" ident="0">
          <idmef:address>0:0:0:0:0:0:0:0</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>0</idmef:port>
      </idmef:Service>
    </idmef:Source>
    <idmef:Target decoy="unknown" ident="0">
      <idmef:Node category="unknown" ident="0">
        <idmef:Address category="ipv6-addr" ident="0">
          <idmef:address>0:0:0:0:0:0:0:0</idmef:address>
        </idmef:Address>
      </idmef:Node>
      <idmef:Service ident="0">
        <idmef:port>0</idmef:port>
      </idmef:Service>
    </idmef:Target>
    <idmef:Classification text="spp_arpspoof: ARP Cache Overwrite Attack"
ident="0"/>
    <idmef:Assessment>
      <idmef:Impact type="other" severity="medium"/>
    </idmef:Assessment>
    <idmef:AdditionalData meaning="bad-unknown" type="string">
      <idmef:string>Potentially Bad Traffic</idmef:string>
    </idmef:AdditionalData>
  </idmef:Alert>
</idmef:IDMEF-Message>
```

Figure 93 – Dump of SMGW database in case of ARP poisoning MITM

4.4.4 IRP operation

In this case, an ARP cache Overwrite Attack is detected and we know that a new attack could be run very soon. At least an external intruder could intercept the packets directed to a particular RTU and decide to block or modify them. In Figure 94 the attack is received from the Detection Layer and show in the attacks list, and passed to the Cyber Simulator implementing the vulnerability analysis.

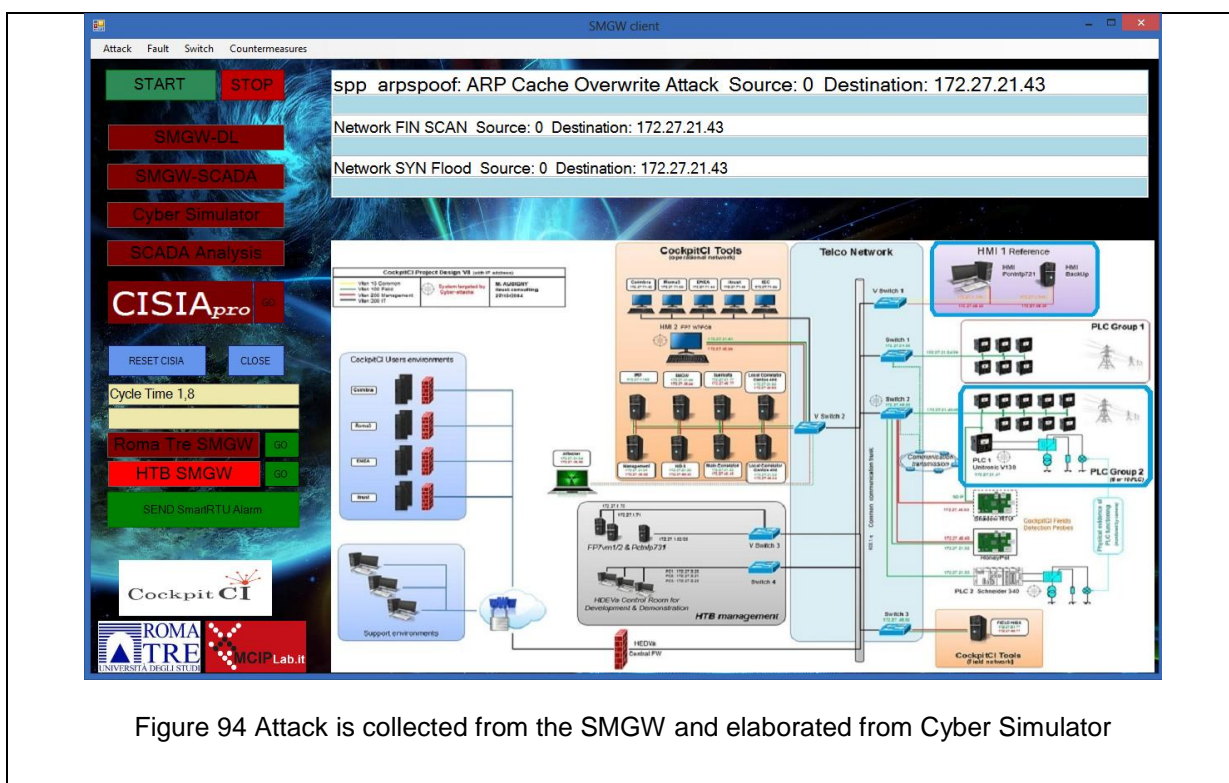


Figure 94 Attack is collected from the SMGW and elaborated from Cyber Simulator

In Figure 95 we see the effects on the other RTUs and on some nodes of SCADA TLC networks.

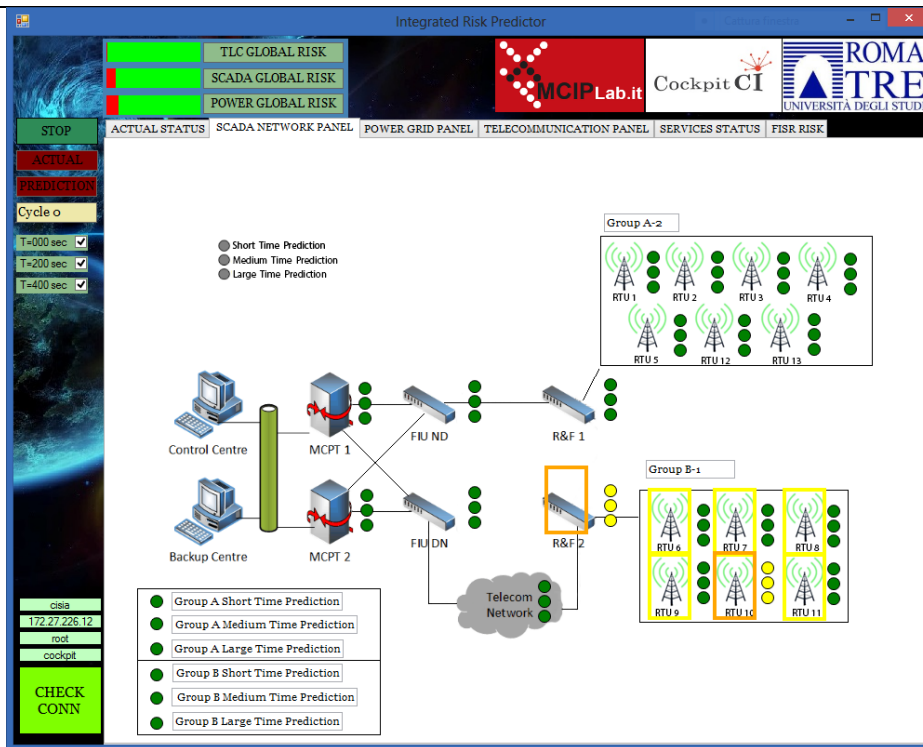


Figure 95 Propagation on the SCADA TLC network

In Figure 96, the effects on the Electric Grid is reported. We can see that the Operative Levels are set to 0.4 for the attacked RTU and 0.9 for the other RTUs on the same network segment.

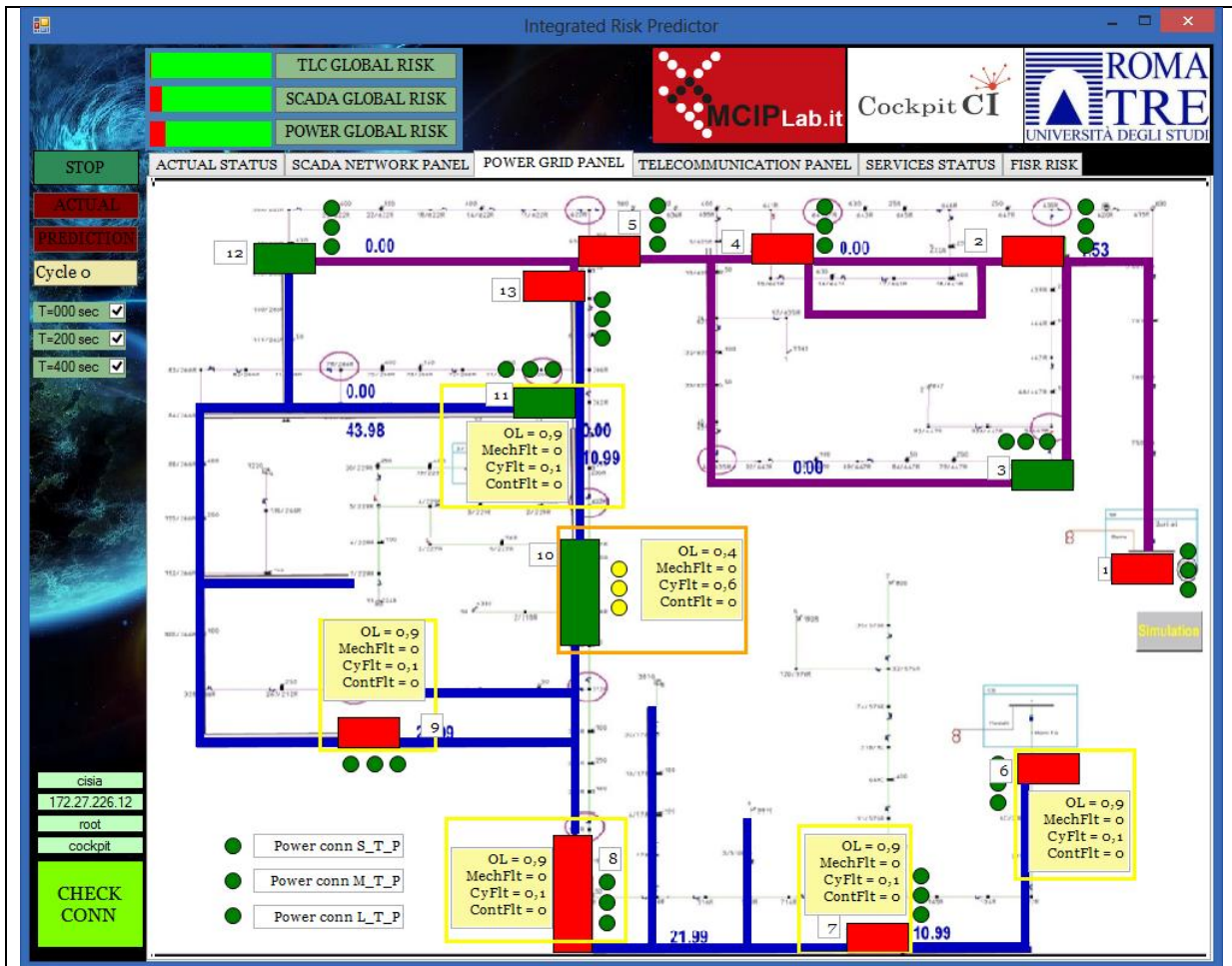


Figure 96 Propagation on the Electric Grid of ARP Poisoning

In Figure 97, the risks for the two FISRs is calculated. The second one seems to be more reliable.

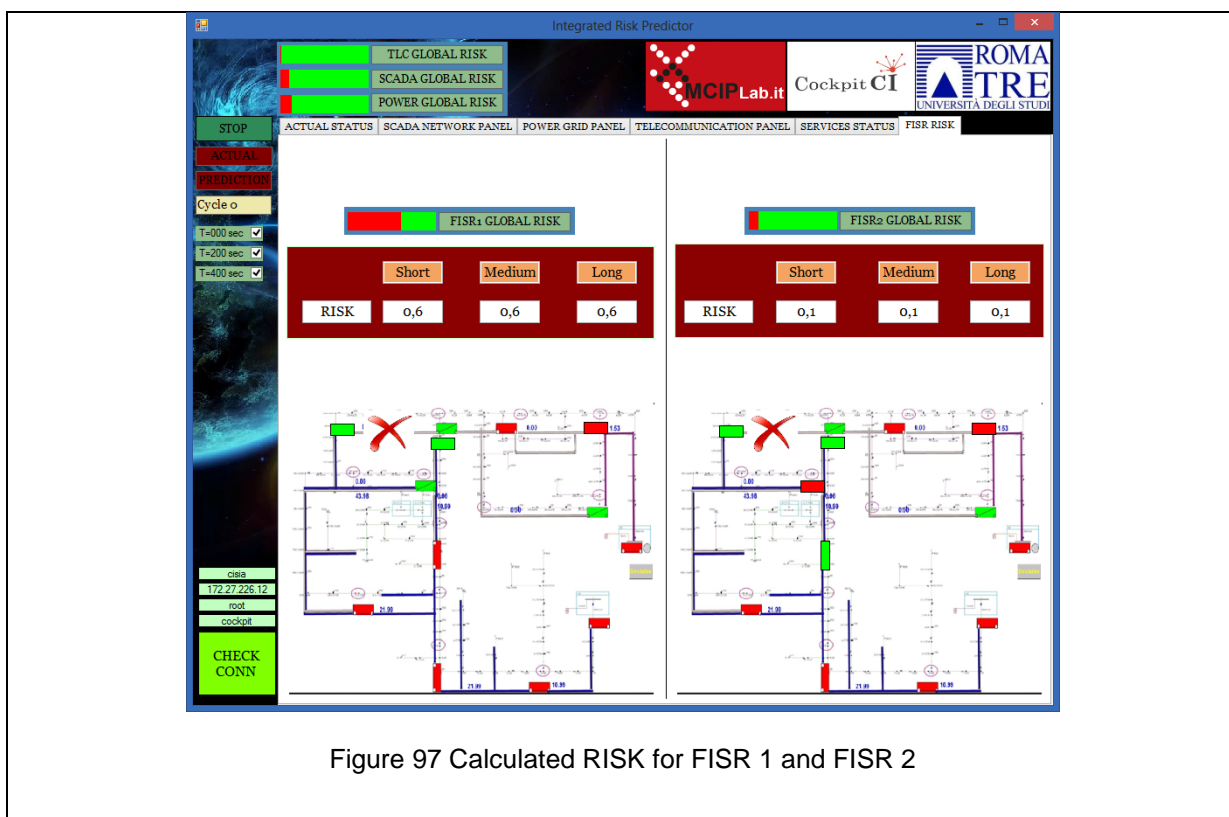


Figure 97 Calculated RISK for FISR 1 and FISR 2

4.5 Physical Fault #1

4.5.1 Door Open Source: 0 Destination: 172.27.21.37

This SCADA alarm is read from the field by the HMI and is reported by the SCADA adaptor to the SMGW. In this way, it can be collected and processed by the IRP. CISIA is initialized and run with this fault active. In Figure 98, the received door alarm is displayed before the propagation (that in this particular case is absent) and in Figure 99 and Figure 100 the effects on other RTUs and their Operative Levels is reported. Not much damage, but RTU 4 is possibly under a physical attack and should be monitored in a better way.

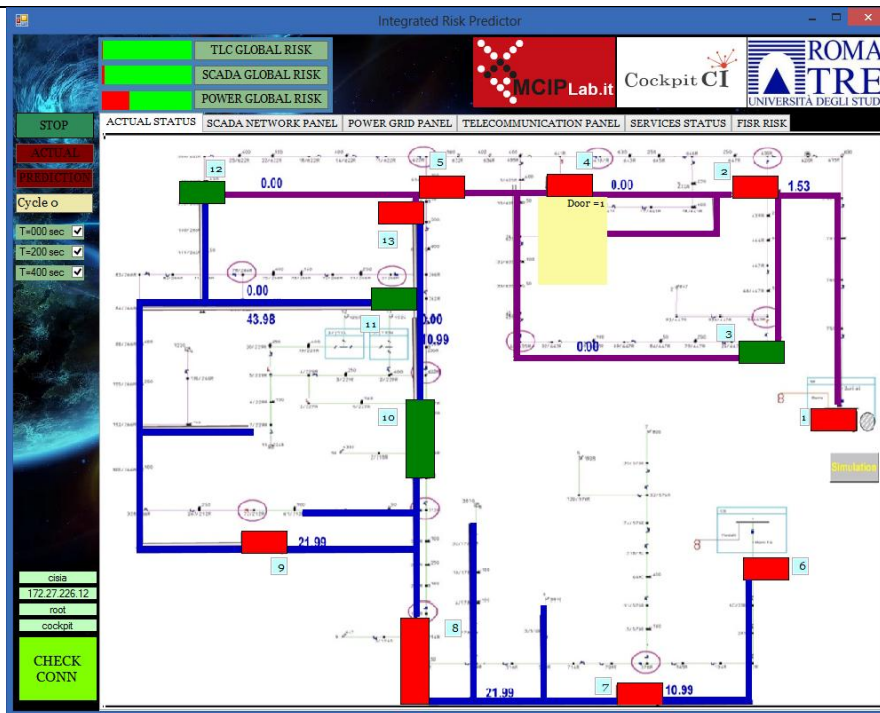


Figure 98 A Door alarm is generated in RTU 4

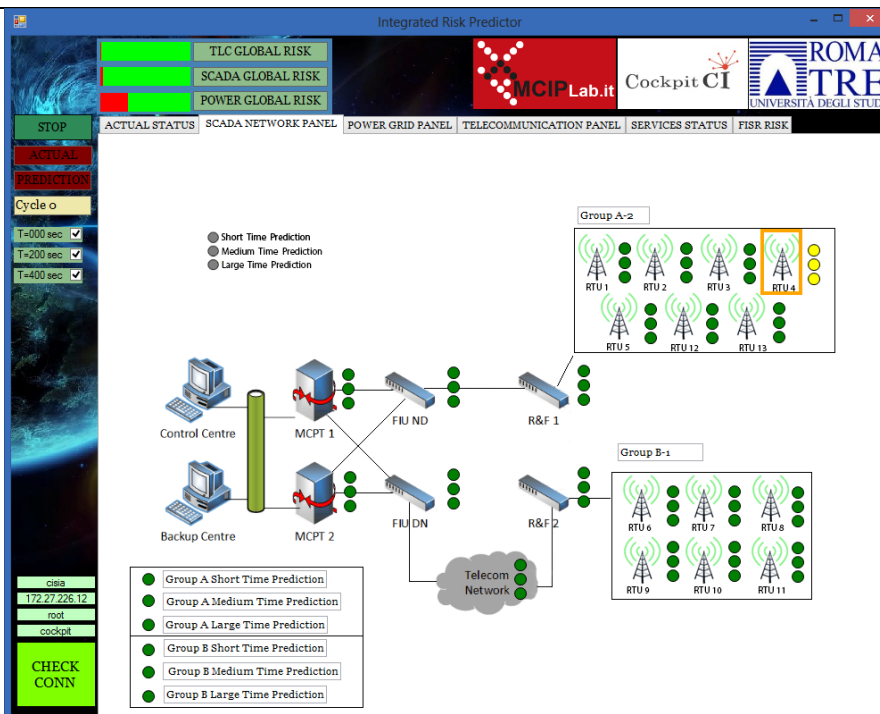


Figure 99 CISIA propagation on the SCADA TLC network is absent

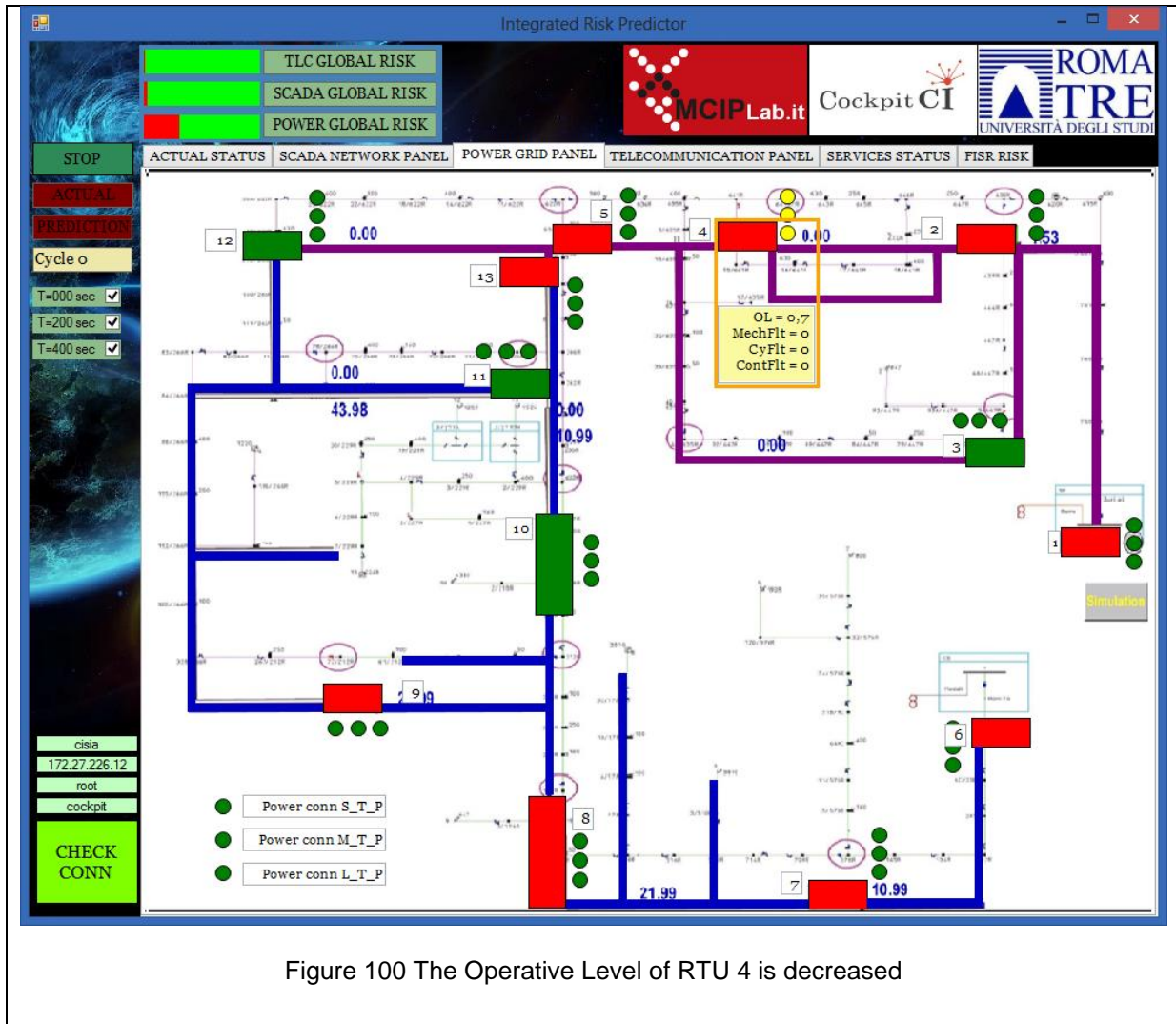


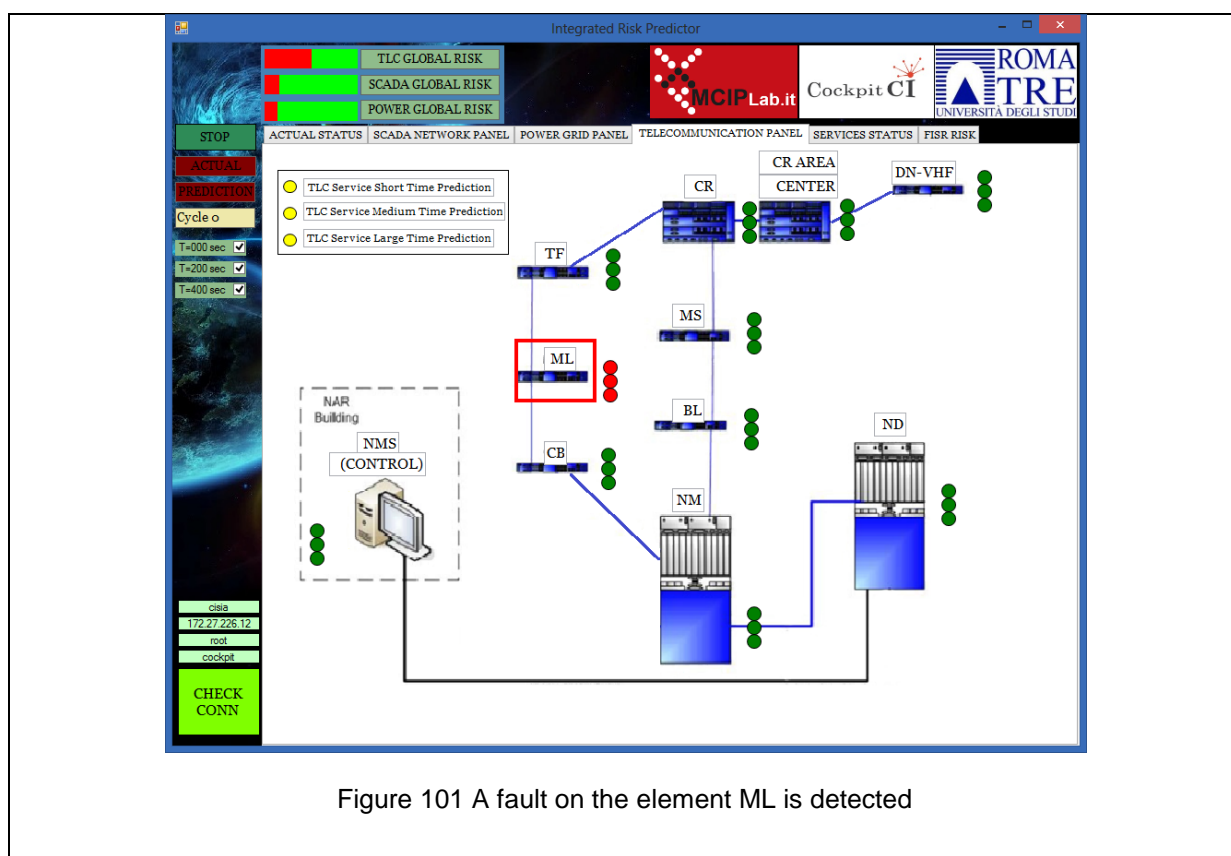
Figure 100 The Operative Level of RTU 4 is decreased

4.6 Cyber-Physical Fault

In this section the effects of a mechanical fault and two cyber-attacks in sequence is reported to show the possibility of having combinations of previous attacks and faults.

4.6.1 Mechanical fault on ML element of Optical ring node

First, a fault on an element of the Optical Ring is detected and fed to the CISIA simulator. The effects are reported in Figure 101 where the element ML is reported in red, and in Figure 102 where the effects on the SCADA TLC network are displayed.



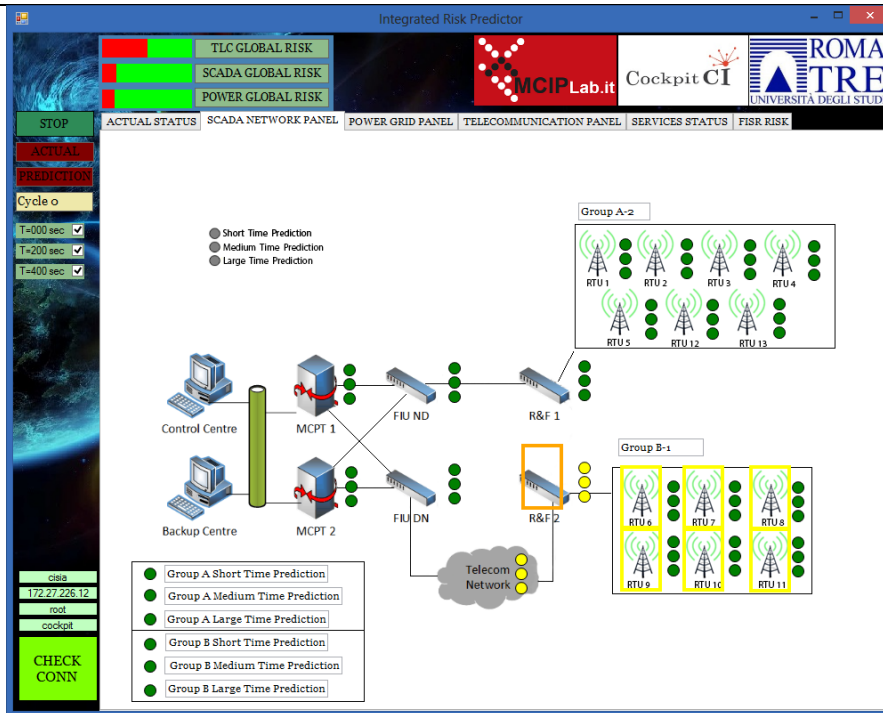


Figure 102 The TLC network will have a decreased Operative Level compromising the efficiency of some RTUs

In Figure 103, the Operative Levels of the RTUs are reported and the operator is aware that the situation is not so bad. The RISKS of the two FISRSs is low as can be seen from Figure 104.

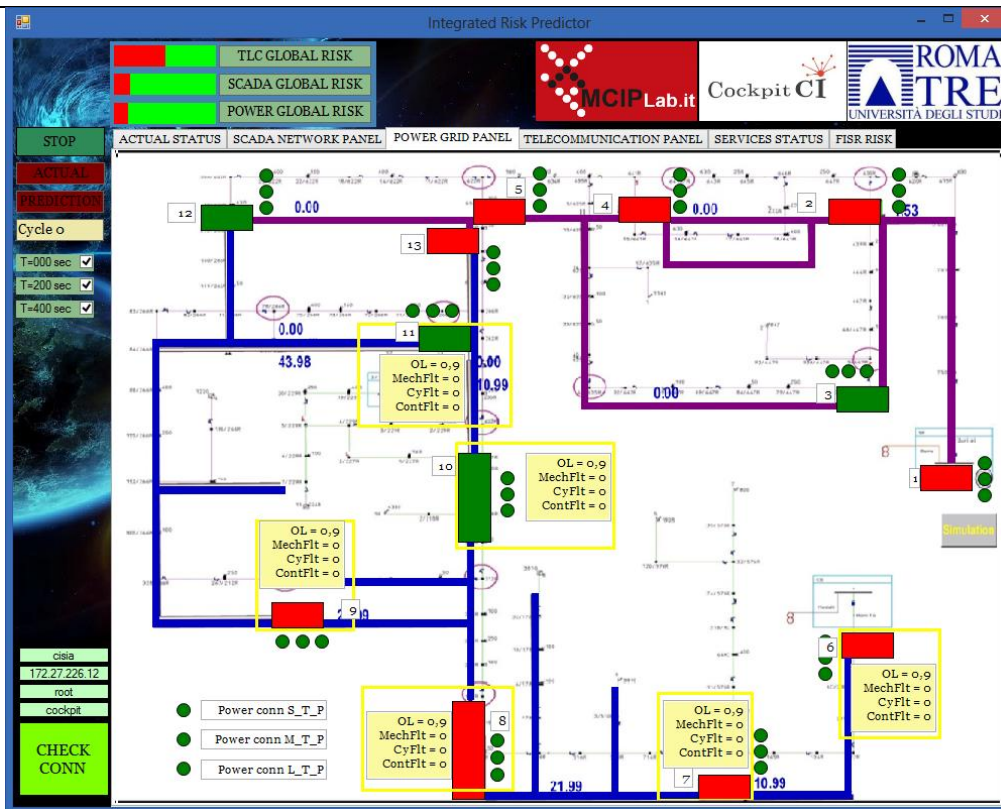


Figure 103 Distribution of Operative Levels among RTUs

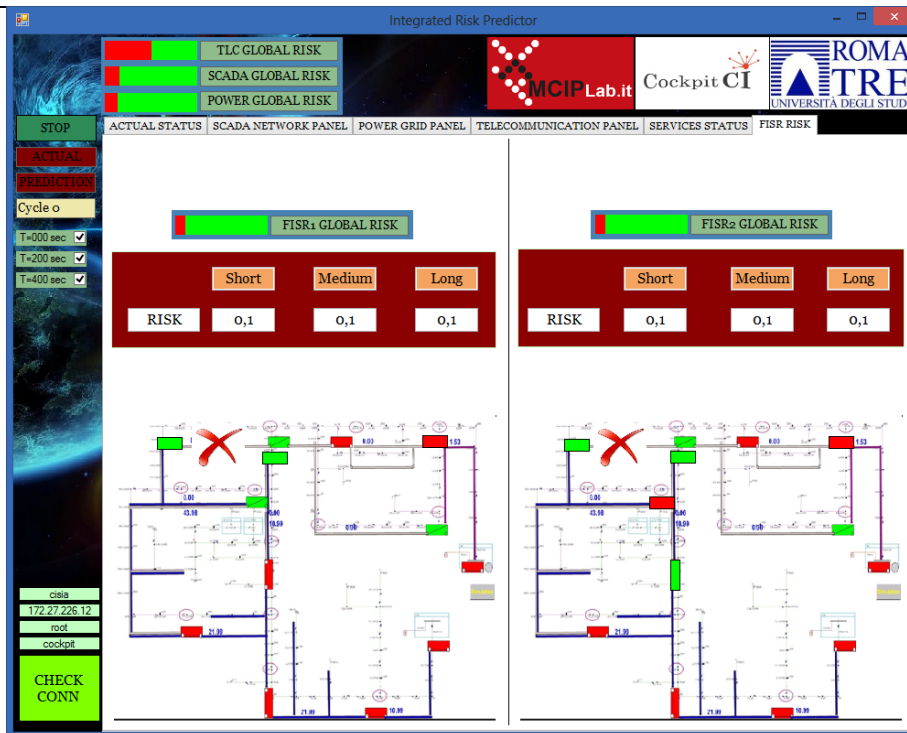


Figure 104 The risk of both FISRs is slightly increased

4.6.2 spp_arp spoof: ARP Cache Overwrite Attack Source: 0 Destination: 172.27.21.43

Then, an ARP poisoning attack is performed. In Figure 105, the cyber flag MITM is activated.

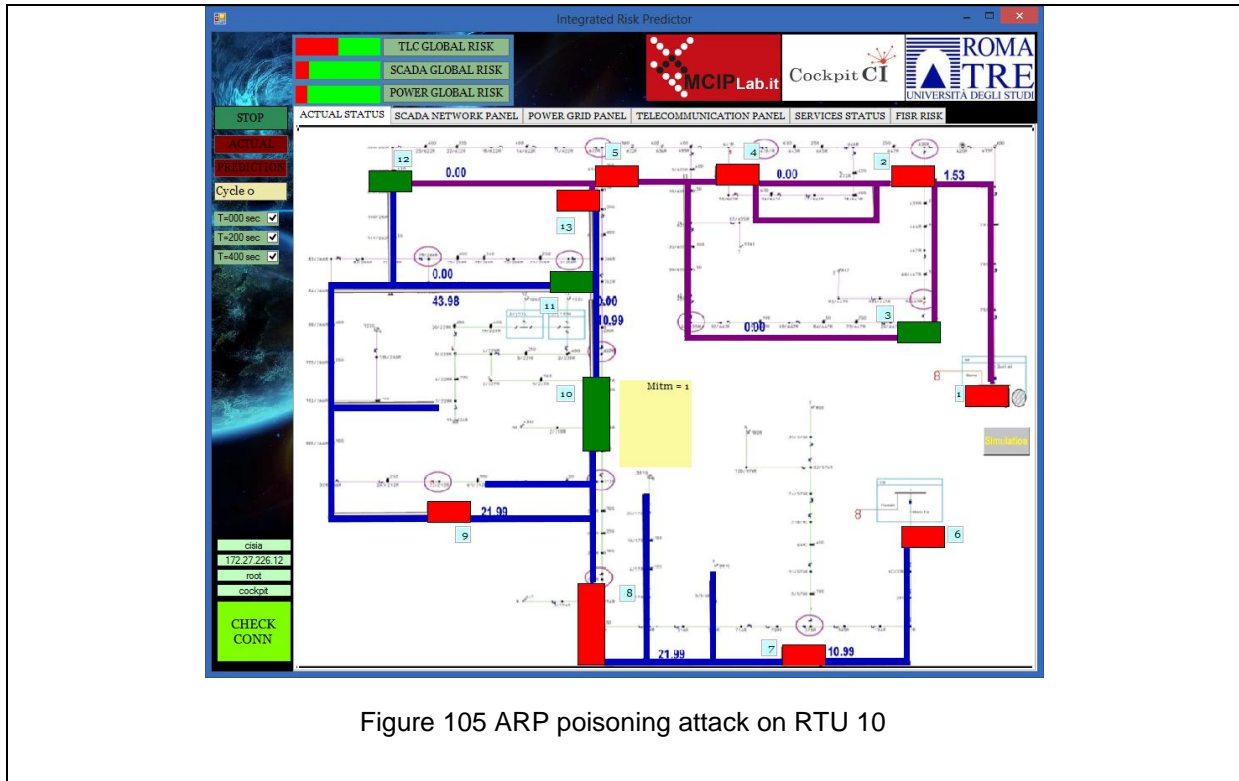


Figure 105 ARP poisoning attack on RTU 10

Now the Operative Level of RTU 10 is decreased as can be seen in Figure 106 and in Figure 107.

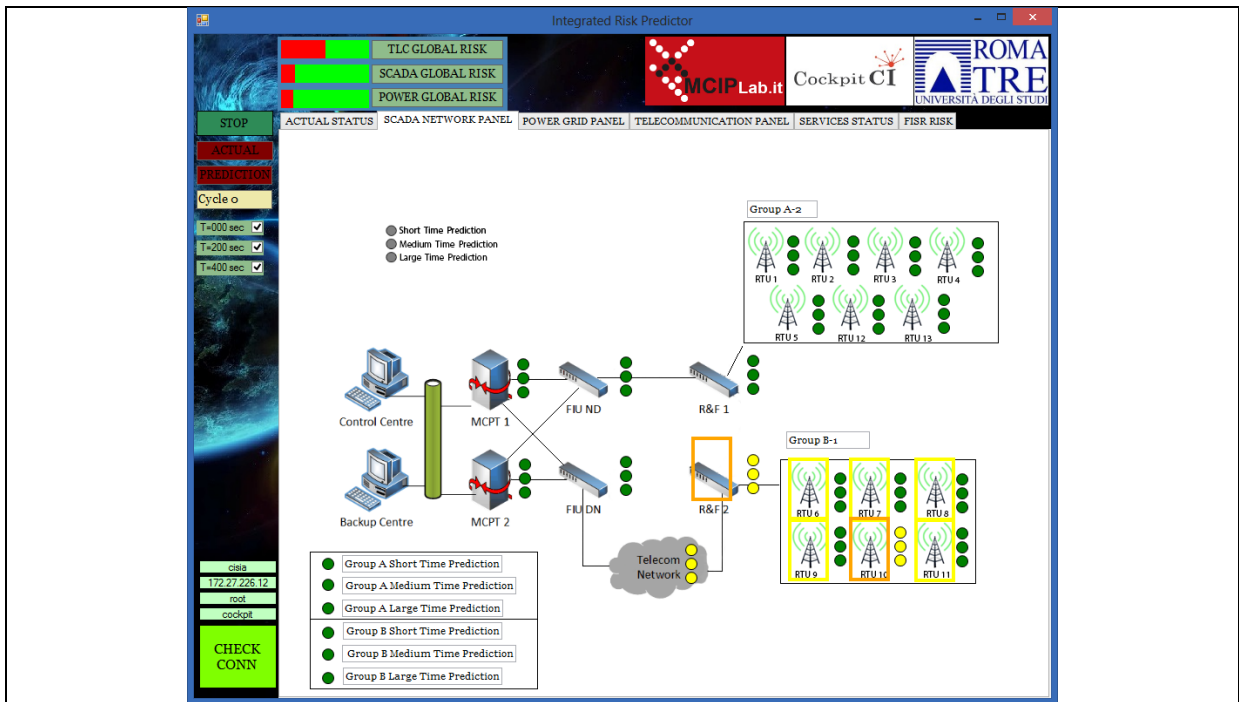


Figure 106 The Operative Level of RTU 10 is decreasing after a propagation

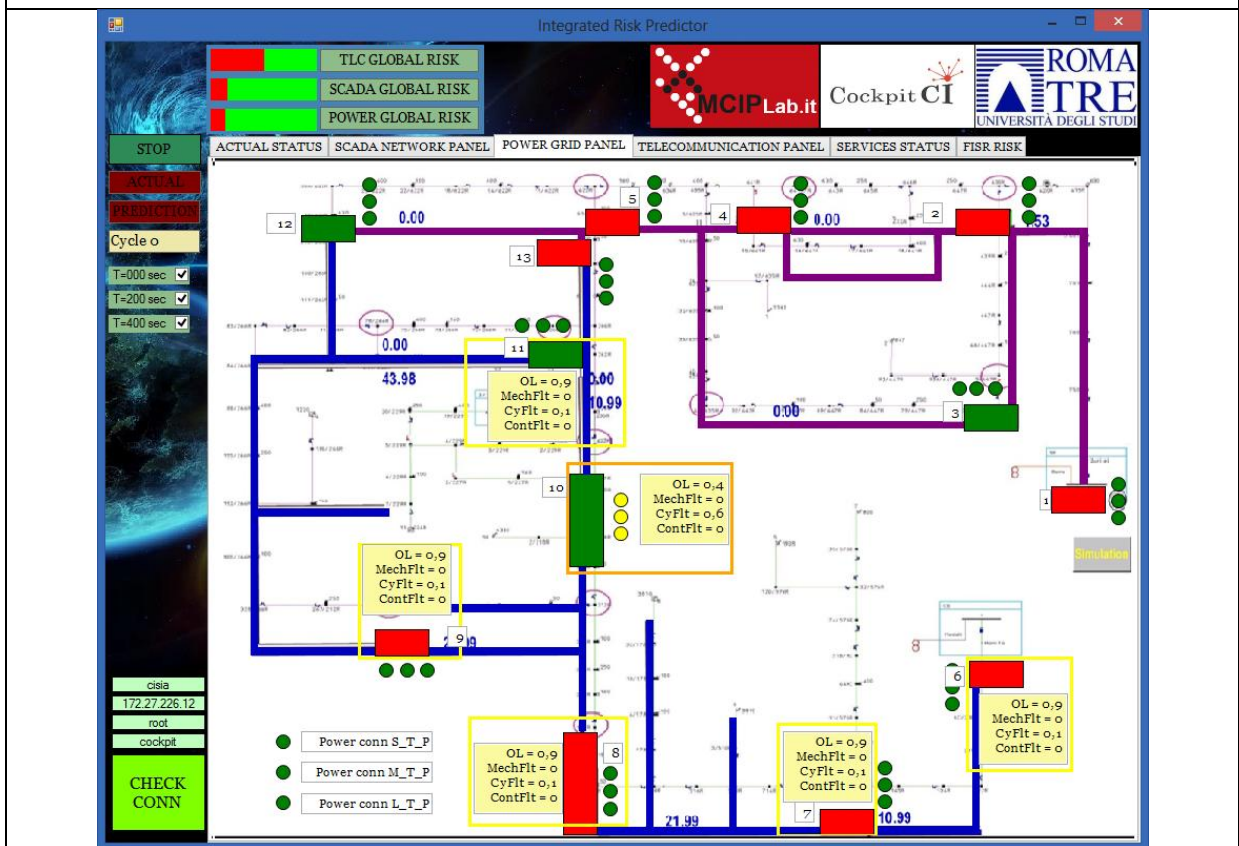


Figure 107 Distribution of OLs and Cyber Faults on the Electric Grid

The risk of the two reconfigurations is increasing as can be seen in Figure 108.

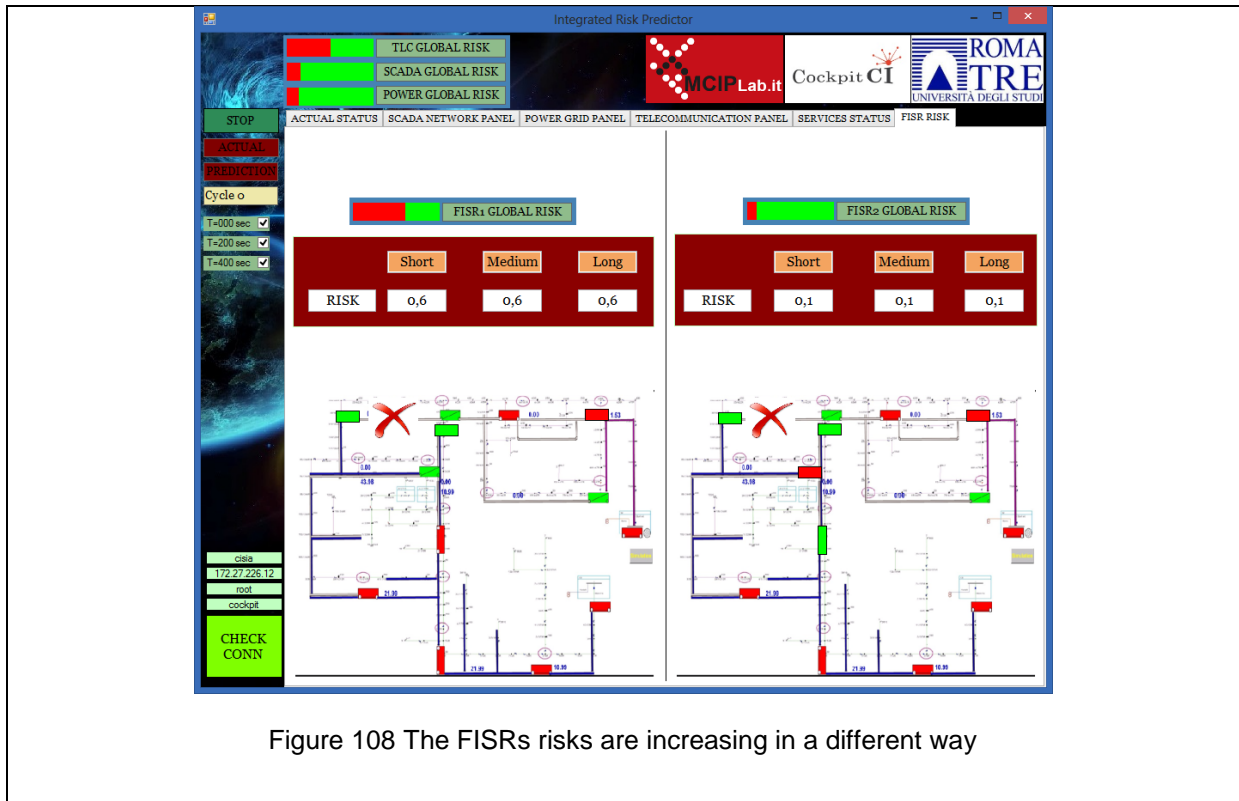


Figure 108 The FISRs risks are increasing in a different way

4.6.3 Network SYN Flood Source: 0 Destination: 172.27.21.44

Finally, a new attack is detected on RTU 11. Now the situation is getting worse. In Figure 109 the RTU11 is red to show that the risk of using it is very high.

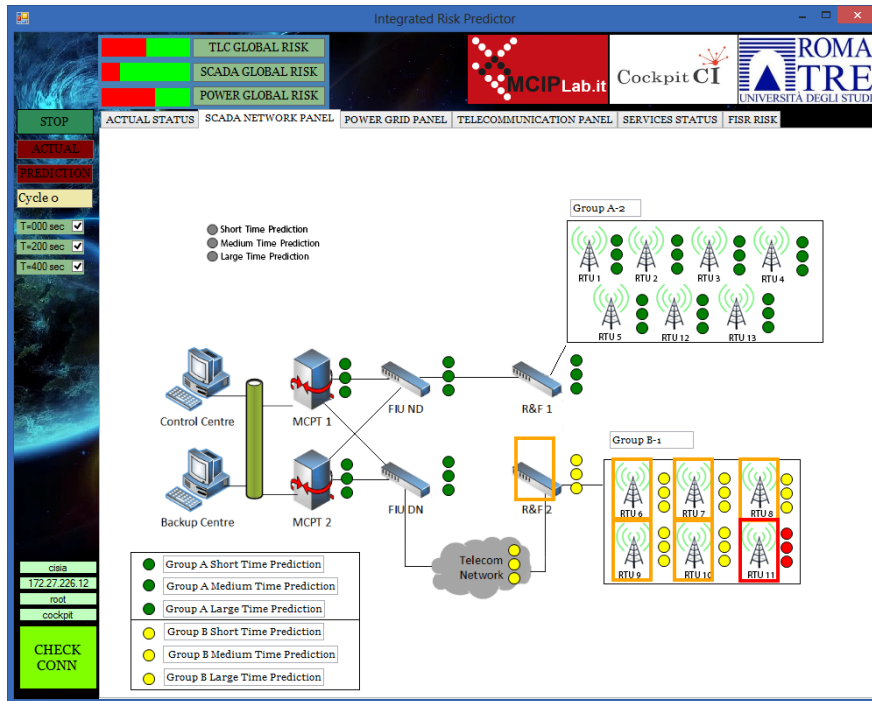


Figure 109 Cyber Propagation of both attacks on SCADA TLC network

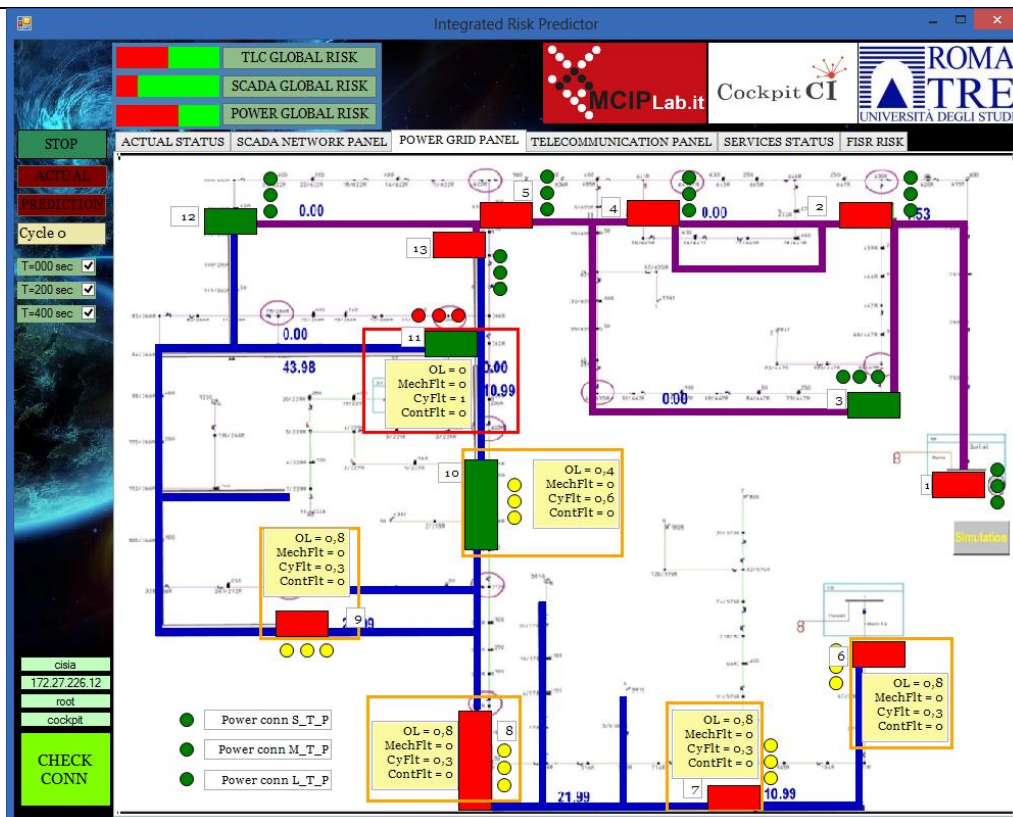


Figure 110 The Operative Levels of all RTUs and the Cyber Faults

In Figure 110, the Operative Levels of many RTUs are lowered but the one related to RTU11 is zero. In Figure 111, we find that only one FISR can be considered.

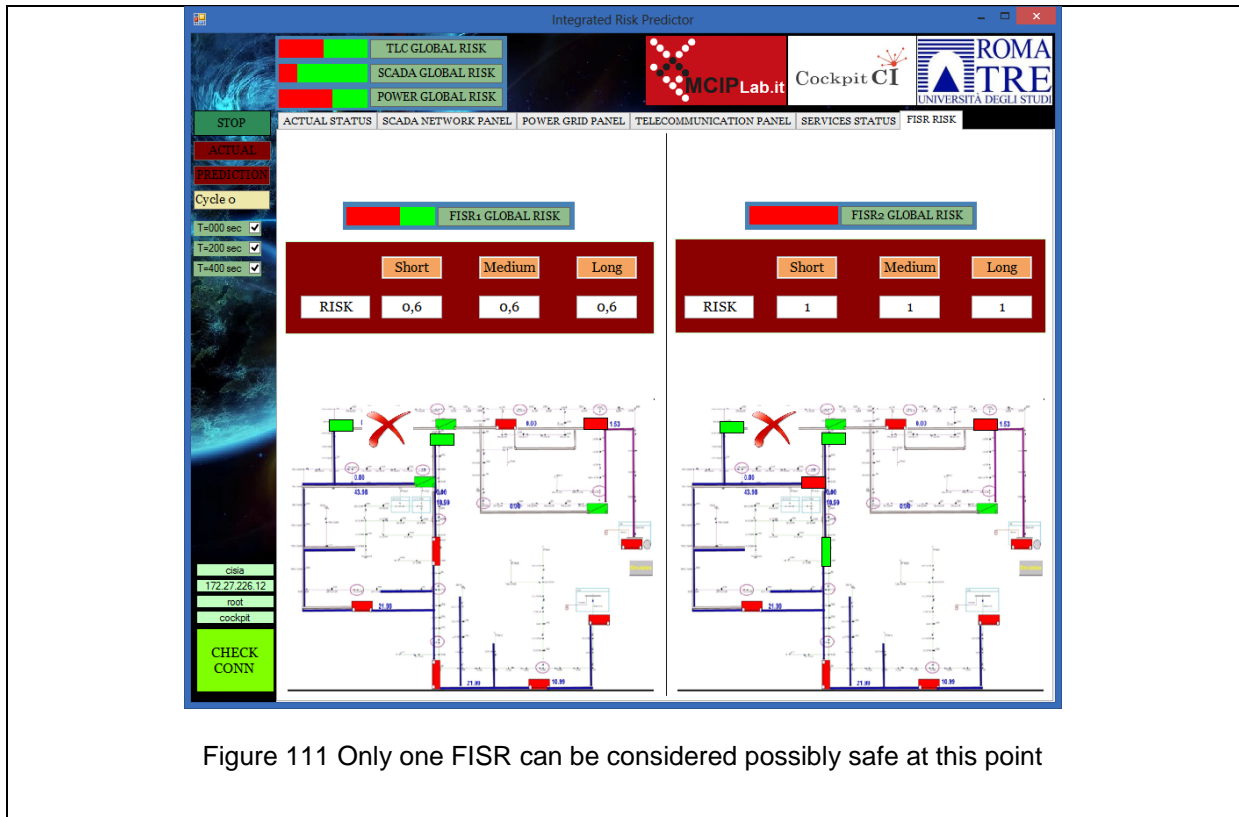


Figure 111 Only one FISR can be considered possibly safe at this point

4.7 Cyber-Physical Fault #1

In this section the effects of a mechanical fault and two cyber-attacks in sequence is reported to show the possibility of having combinations of previous attacks and faults.

4.7.1 Mechanical fault on ML element of Optical ring node

First, a fault on an Element of the Optical Ring is detected and fed to the CISIA simulator. The effects are reported in Figure 112 where the element ML is reported in red, and in Figure 113 where the effects on the SCADA TLC network are displayed.

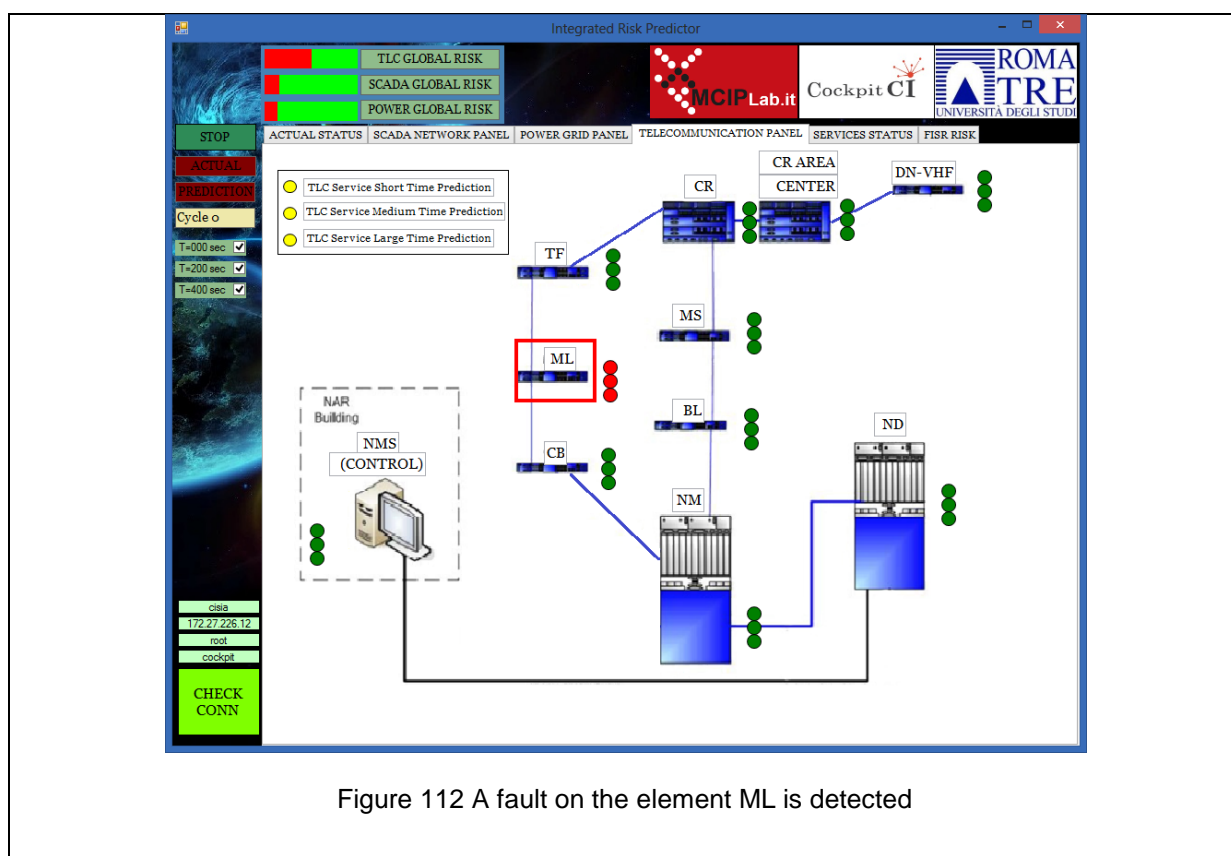


Figure 112 A fault on the element ML is detected

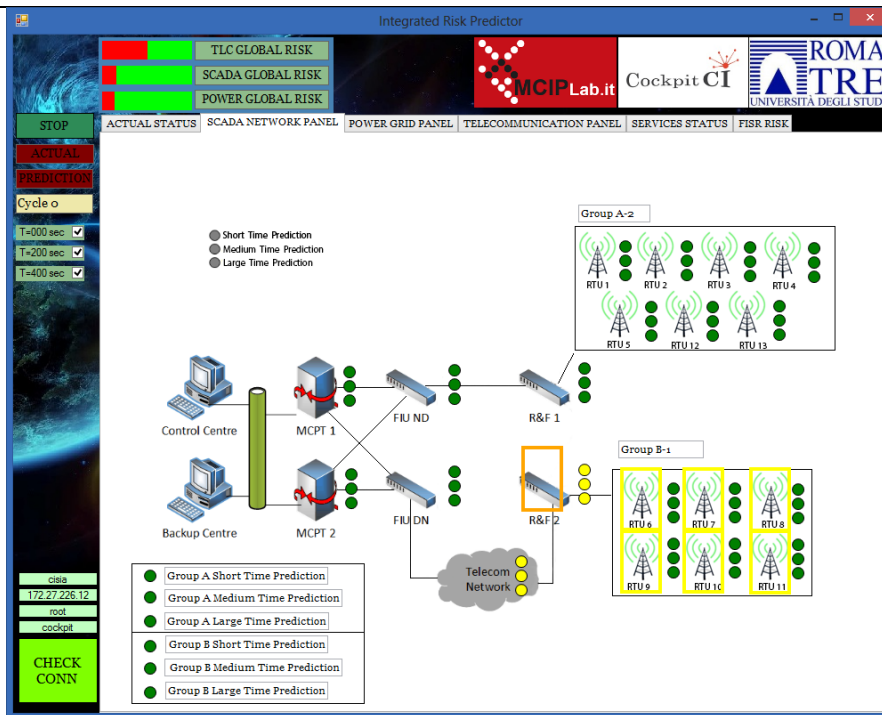


Figure 113 The TLC network will have a decreased Operative Level compromising the efficiency of some RTUs

In Figure 114, the Operative Levels of the RTUs are reported and the operator is aware that the situation is not so bad. The risk of the two FISRSs is low as can see from Figure 115.

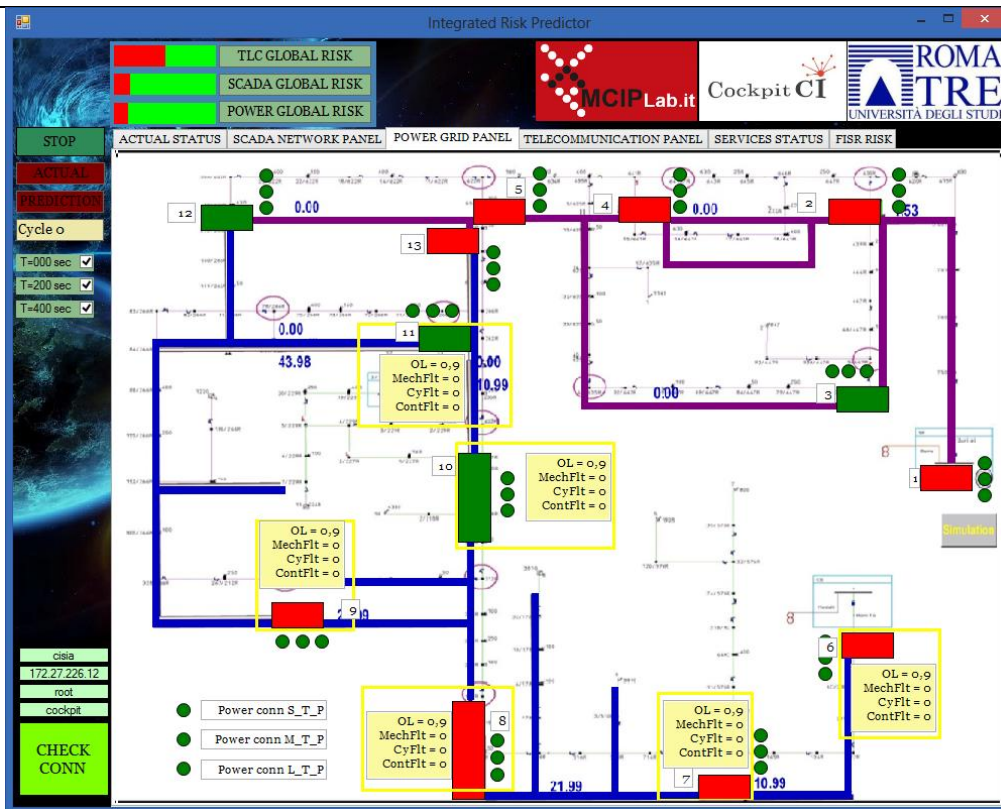


Figure 114 Distribution of Operative Levels among RTUs

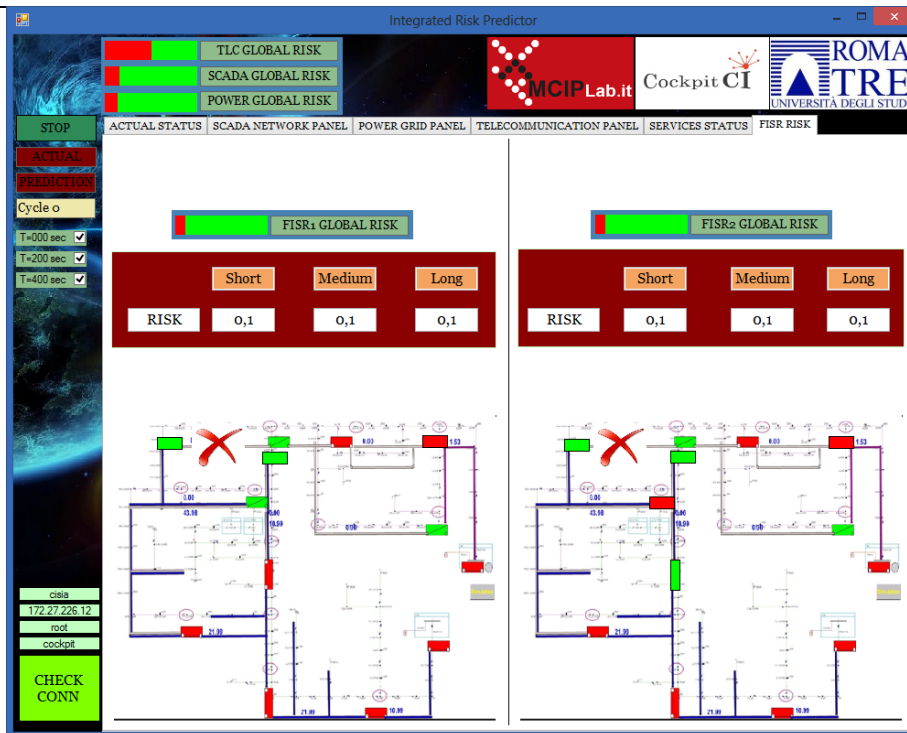


Figure 115 The RISK of BOTH FISRs is slightly increased

4.7.2 spp_arp spoof: ARP Cache Overwrite Attack Source: 0 Destination: 172.27.21.43

Then, an ARP poisoning attack is performed. In Figure 116, the cyber flag MITM is activated.

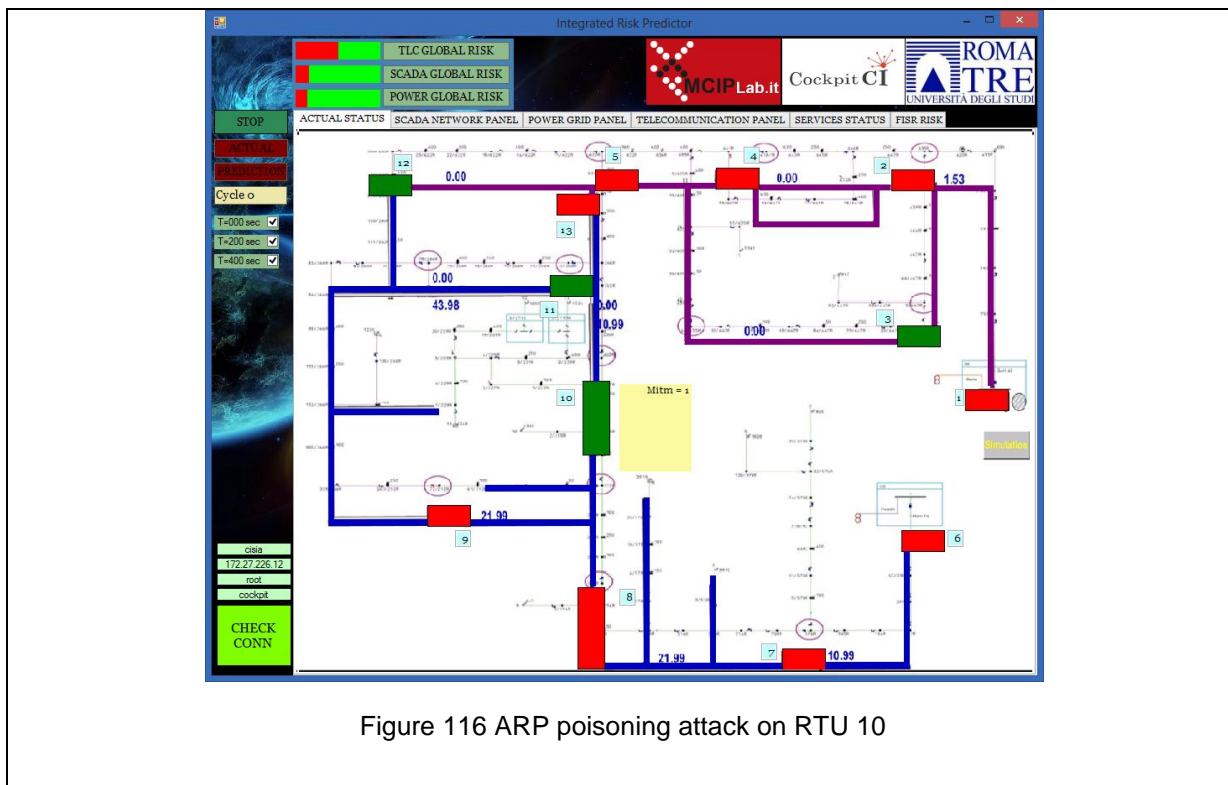


Figure 116 ARP poisoning attack on RTU 10

Now the Operative Level of RTU 10 is decreasing as can be seen in Figure 117 and in Figure 118.

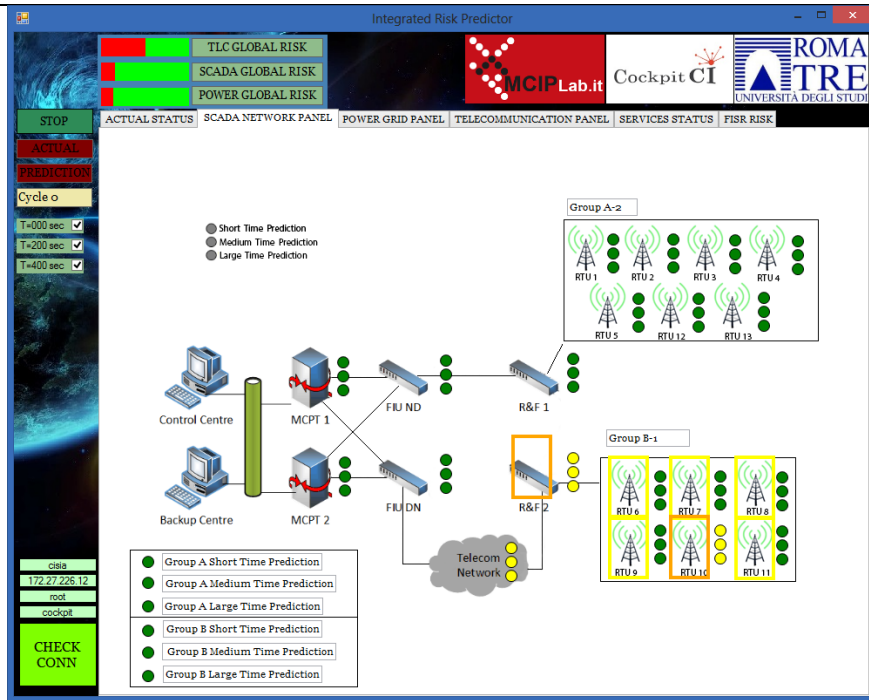


Figure 117 The Operative Level of RTU 10 is decreasing after a propagation

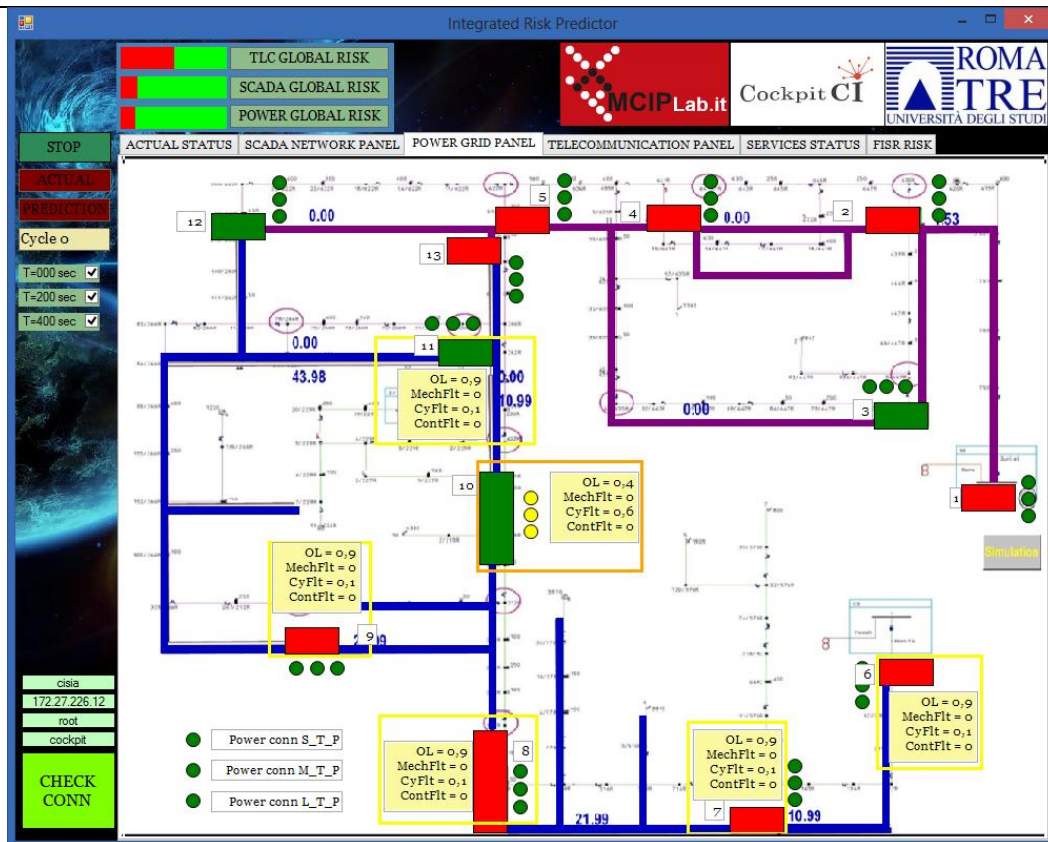


Figure 118 Distribution of OLs and Cyber Faults on the Electric Grid

The risk of the two reconfigurations is increasing as can be seen in Figure 119.

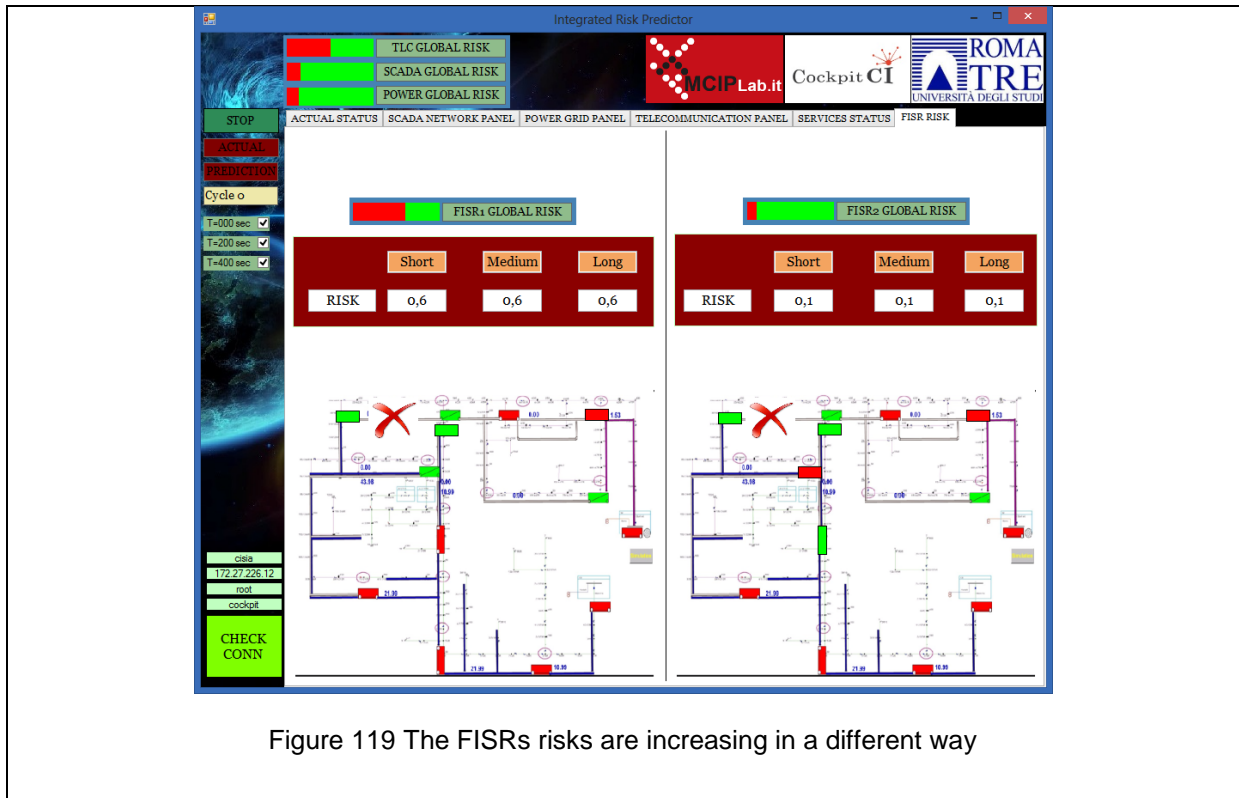


Figure 119 The FISRs risks are increasing in a different way

4.7.3 Network SYN Flood Source: 0 Destination: 172.27.21.44

Finally, a new attack is detected on RTU 11. Now the situation is getting worse. In Figure 120 the RTU11 is red to show that the risk of using it is very high.

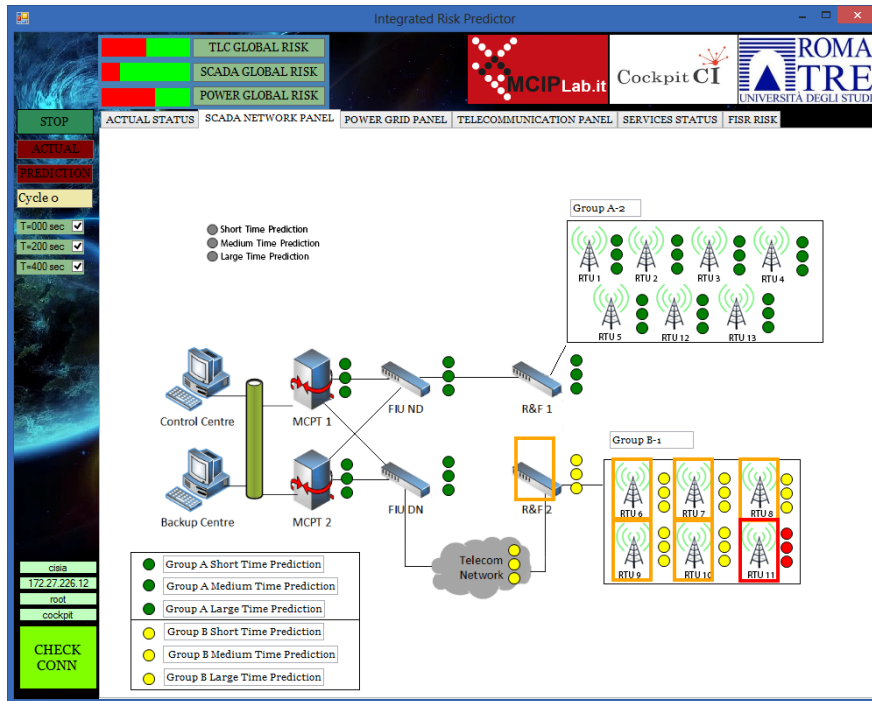


Figure 120 Cyber Propagation of both attacks on SCADA TLC network

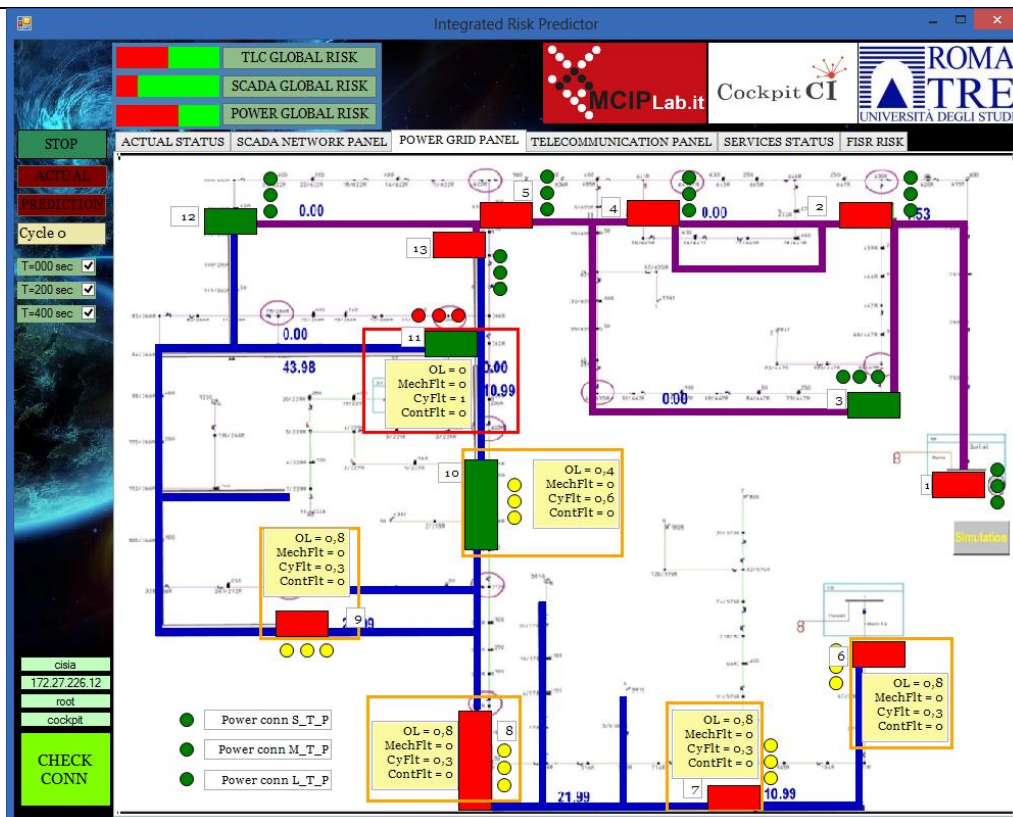


Figure 121 The Operative Levels of all RTUs and the Cyber Faults

In Figure 121, the Operative Levels of many RTUs are lowered but the one related to RTU11 is zero. In Figure 122, we find that only one FISR can be considered.

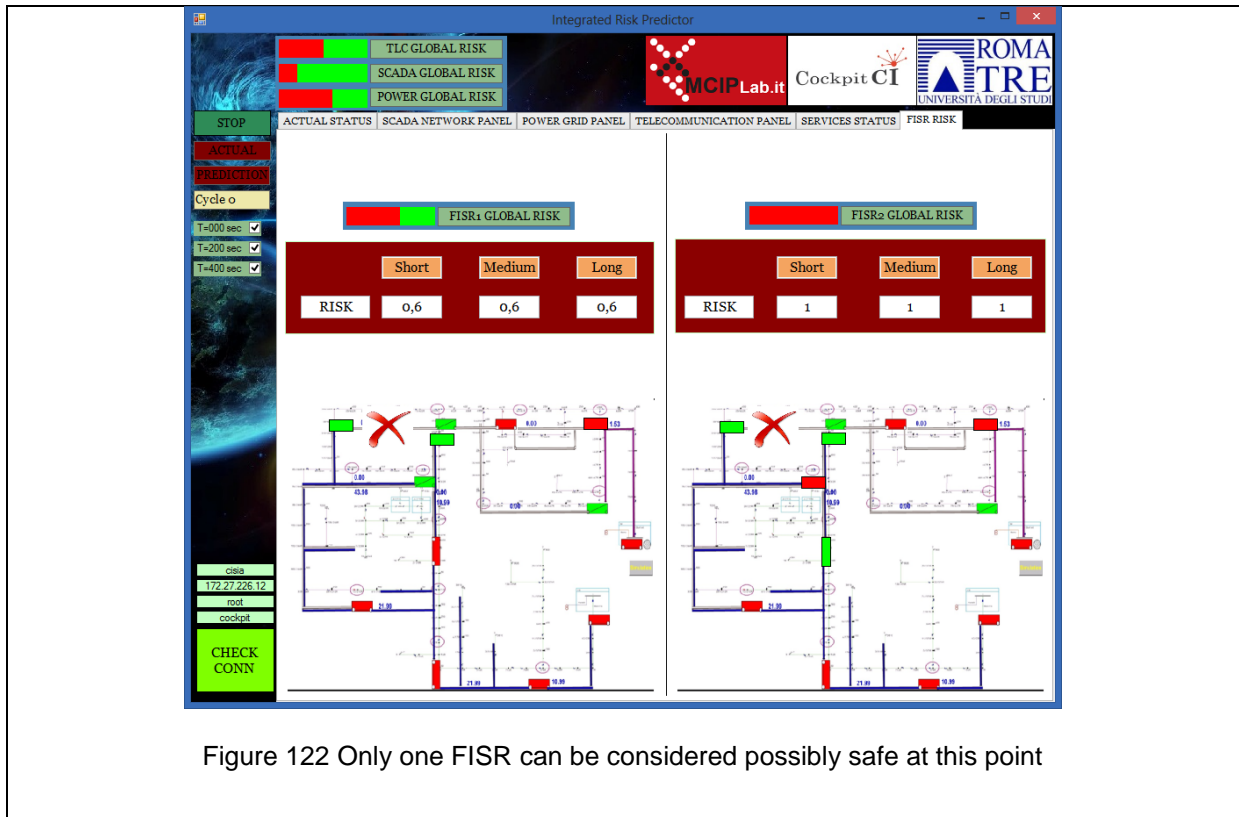


Figure 122 Only one FISR can be considered possibly safe at this point

5 Conclusions

In this deliverable the factory unit and integration tests using SMGW, DL, Scada Adaptor and IRP are provided by illustrating all the procedures that have been implemented.

The documents starts from the description of the main features of the SMGW providing also a guide addressed to describe how to get and install the machine. Then the security configuration is explained and a description of tools to perform the tests is provided.

In order to validate the deployed machine, a description of security tests is provided describing the results and the detected vulnerabilities. The tests have been carried out by adopting a black box approach. For the main vulnerabilities a short description of the problem is specified jointly with the indication of a solution to overcome each problem in future releases.

Also the exchange of messages between different SMGWs has been simulated in different conditions, both by using certificates and without certificates. In order to carry out this test, a configuration with two SMGWs on the same network has been simulated using VMware virtual machines.

The factory integration tests have been performed over the virtual LAN including DL, SCADA Adaptor, SMGW and IRP across the labs located in Coimbra and University of Roma TRE. All the messages have been delivered correctly in all simulations of attacks that have been carried out. In all cases the DL has detected the attack and has sent the correct information to the IRP using the SMGW and the IRP has provided alarms and indications for the end user over the graphical interfaces. Also scenarios with physical fault and a sequence of cyber attacks have been successfully tested.

In all cases the CockpitCI system was compliant with the requirements.

Requirement id	Short description	Implementation	Feature that must be fulfilled by the test	TEST RESULT
SMN_1.	The Secure Mediation Network <i>shall</i> provide near real-time secure, reliable and available data exchange between the Detection Layer and the IRP, as well as among different CIs	A SOA-based architecture for data exchange has been adopted to fulfil the requirement.	Delivery of IDMEF file end-to-end from DL to IRP	TO BE VERIFIED ON HTB
SMN_2	The Secure Mediation Network <i>shall</i> interface with the detection layer	A proper DL interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired in SMGW database from DL entity by means of a POST method.	OK
SMN_3	The Secure Mediation Network <i>shall</i> interface with the prediction tool	A proper IRP interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired from IRP by means a of GET method	OK
SMN_4	The Secure Mediation Network <i>shall</i> interface	A proper interface has been foreseen	A SOAP message should be sent	OK

	with external peer Secure Mediation Networks through an Internet connection	in the architecture of the SMN.	between SMGW approach using a publish / subscribe.	
SMN_5	The Secure Mediation Network <i>shall</i> support non real-time exchange of other kinds of information among CIs	A message broker with separate queue for each topic, each with its own priority, has been introduced in the architecture for InterSMGW communications.	The messages can be consumed by remote SMGW by using a publish subscribe mechanism. The remote invocation can consume messages or by selecting all messages in the database or requesting all not already accessed.	OK
SMN_6	The Secure Mediation Network <i>shall</i> acquire CI independent metadata from SCADA adaptors	A proper SCADA Adaptor interface has been foreseen in the architecture of the SMN.	An IDMEF file should be acquired in SMGW database form SCADA Adaptor by means of a POST method.	OK
SMN_7	The Secure Mediation Network <i>shall</i> store information obtained by all interfaced components in a dedicated database	Proper DB modules of the SMGW will manage exchange and storage of metadata.	Data arriving from DL and SCADA adaptor should be collected in dedicated databases.	OK
SMN_8	A specific framework shall be included in the Secure Mediation Network in order to allow local CockpitCI components and external SMGWs to retrieve metadata useful for their purposes	A message broker implementing a topic oriented, publish-subscribe mechanism has been adopted. Moreover, Web services implementing the request-response scheme have been introduced for metadata retrieval.	A metadata publisher should assure the capability to retrieve the metadata.	OK
SMN_9	The Secure Mediation Network <i>shall</i> perform information discovery at peer SMGWs to retrieve state information of interdependent CIs	SMN-SMN communication will be based on publish/subscribe paradigm, hence supporting the standard procedures such as service publishing, discovery, subscription, etc.	In message exchange the SMGW should check the list of subscribers and of subscribed services	OK
SMN_10	All ingoing and outgoing connections of the Secure Mediation Network <i>shall</i> be secure	Confidentiality, authenticity and integrity of ingoing and outgoing flows	At application level the security mechanisms should be adopted. The	OK

	connections	are provided by various security mechanisms, at both transportation and application level (IPSec, VPN, HTTPS, WS-Security, etc...) as detailed in the whole report.	web services should be exposed at least using https.	
SMN_1 1	The Secure Mediation Network <i>shall</i> disclose stored global awareness metadata of the local CI, to authorized subscribers, i.e. other SMGWs	SMN will disclose metadata provided by the IRP (e.g. current and predicted CI status) only to authorized subscribers (e.g. other SMGWs).	The SMGW should filter the notifications only to subscribers.	OK
SMN_1 2	The Secure Mediation Network <i>shall</i> accept subscriptions from peer SMGWs to be notified when updated metadata is available	SMN-SMN communication will be based on publish/subscribe paradigm, hence supporting the standard procedures such as service publishing, discovery, subscription, etc.	The SMN-SMN communication should be assured by means Web Services.	OK
SMN_1 3	The Secure Mediation Network <i>shall</i> perform client authentication on the basis of client profiles and certificates	The Web Server(s) providing REST and SOAP Web service will be configured in order to accept only requests from clients authenticated by means of X.509 certificates. Moreover, for each service request, requestor authorization is verified at application level.	Usage of X.509 certificates over SOAP and REST exposed methods.	OK
SMN_1 4	The Secure Mediation Network <i>shall</i> perform security auditing	The SMN provides a log capability, providing selective recording of all service requests and operations performed by each SMGW.	A security vulnerability scan should be performed.	OK
SMN_1 5	In order to guarantee a reliable and effective risk prediction, the Secure Mediation Network <i>shall</i> keep synchronized with	According to the publish/subscribe paradigm, whenever new relevant data are available at a	The notification messages should be sent to all subscribers.	OK

	remote CIs' metadata	remote CI side, all the subscribers to the particular data receive an update.		
SMN_16	The Secure Mediation Network <i>shall</i> provide a management interface allowing a certain degree of security and policies configuration	A control interface is foreseen (see Section 7).	Implemented at level of interchange SMGW-SMGW	OK
SMN_17	The Secure Mediation Network <i>should</i> provide the possibility to define who and in which way can access a certain piece of information	The definition and management of proper policies in the SMN will allow to achieve such a result.	The SMGW should filter the remote nodes on basis of an access list.	OK
SMN_18	The Secure Mediation Network <i>should</i> provide the possibility to define trust relations between different CIs	The definition and management of proper policies in the SMN will allow to achieve such a result.	The SMGW should establish the trust with remote nodes on basis of certificates.	OK
SMN_19	The Secure Mediation Network <i>should</i> enforce different communications protocols/technologies in each particular context	For each different communication, the SMN management functionalities allow the administrator to select the protocol /technology from a list of suitable choices .	Different interfaces should be implemented.	OK
SMN_20	The Secure Mediation Network <i>should</i> enforce Service Level Agreements (SLA) or Service Level Specifications (SLS) between CIs	The definition and management of proper policies in the SMN will allow to achieve such a result.	The management panel of the SMGW should assure the SLA on basis of predefined proprieties.	OK

Table 2 - Test summary

6 References

- [1] CockpitCI Consortium, *Cybersecurity on SCADA: risk prediction, analysis and reaction tools for Critical Infrastructures*, Project Proposal, PART B, 2011.
- [2] *CockpitCI – Deliverable 3.5 - Report on Implementation and Trials of the Detection Layers Components*
- [3] *CockpitCI – Deliverable D4.3 – Implementation and factory trials*
- [4] *CockpitCI – Deliverable D.5.3 – Secure Mediation network design and specification*
- [5] X.509 - <http://www.ietf.org/rfc/rfc5280.txt>
- [6] *Kali Linux – Reference Manual - <https://www.kali.org/official-documentation/>*
- [7] *OpenVas <http://www.openvas.org/>*
- [8] <http://www.metasploit.com/>
- [9] *Poster application <https://addons.mozilla.org/en-US/firefox/addon/poster/>*
- [10] WSO2 - <http://wso2.com/>

7 Appendix A – SMGW management panels

The functionalities related to management, operation and maintenance of the SMGW are supported by the console of the open source WSO2 oxygen middleware that offers a browser based interface for the setup of SMGW related to Apache stack that has been employed for the implementation. A set of Web Server applications, developed as Java Server Pages, provide all those functions required by the SWGW.

Among the functionalities provided by the wso2 oxygen console the main are:

- user management;
- key store Management;
- logging subsystem management;
- subscriber register.

Figure 123 shows the main console of the application server. The controls in the Management Console are usually self-explanatory. However, the context-sensitive help tips, accessible by clicking the 'Help' link at the top right corner of any page, are a fast and easy way to get more information, if required. The product documentation provides further information about the technology and configurations.

The console's menu items appear in the left hand side and they are divided as Main, Monitor, Configure and Tools. Each of these menus carries a list of sub menus.

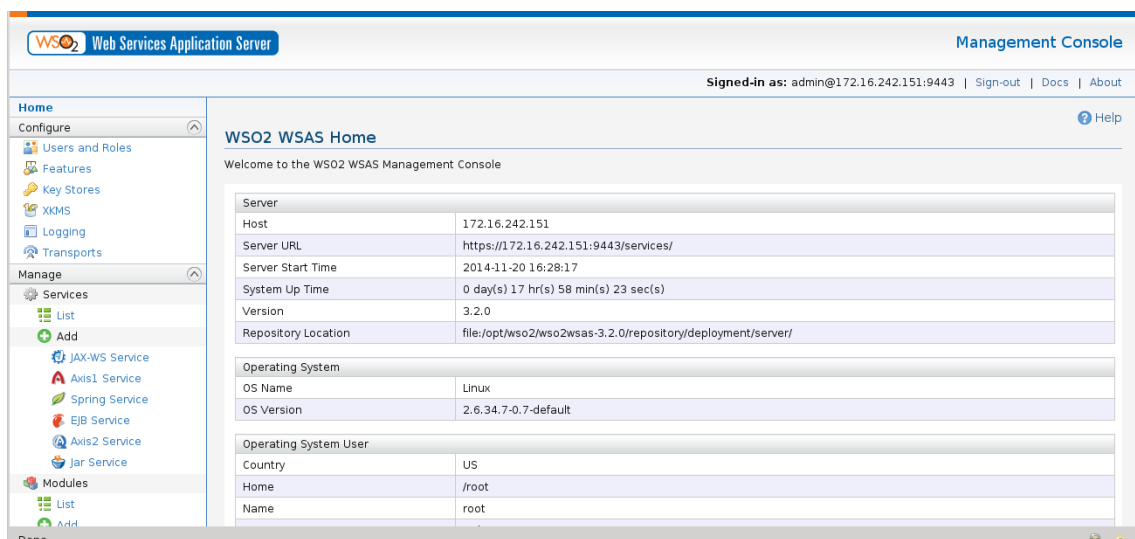


Figure 123 - WSO2 Application Server Management Console

Figure 125 shows the capability of WSO2 Carbon Keystore management to manage the keys. Keystores allows to manage the keys that are stored in a database. A keystore must contain a key pair with a certificate signed by a trusted Certification Authority (CA). A CA is an entity trusted by all parties participating in a secure communication. This entity will certify a trusted party's public keys by signing them. Since the certificate authority is a trusted one, it will accept the public key certificates signed by that particular CA as trusted.

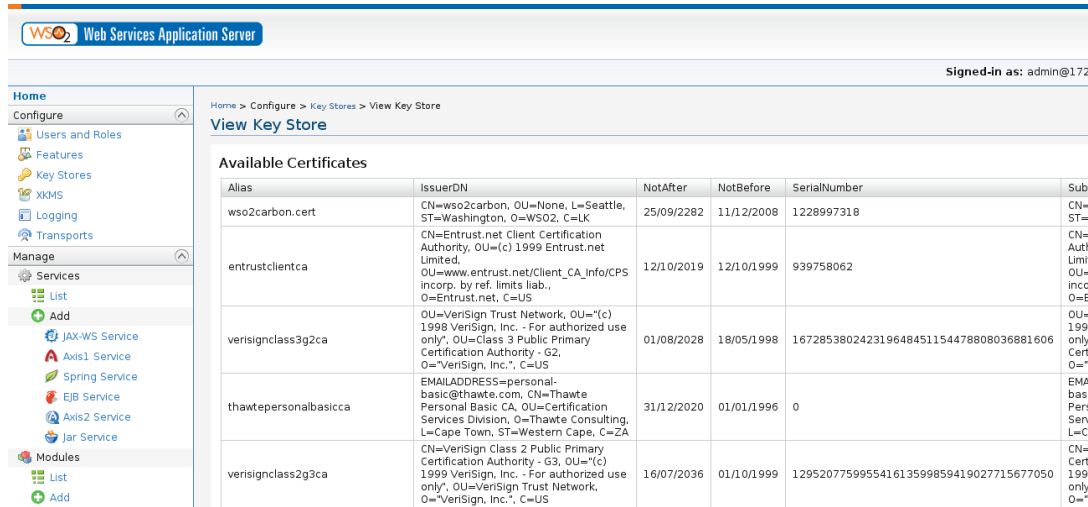


Figure 124 – WSO2 Application Server Key store management

Figure 125 shows the security server management panel where Web services security of SOAP messages can be set. WSO2 supports WS Security, WS-Policy and WS-Security Policy specifications. These helps the administrator to define the service-specific security policies.

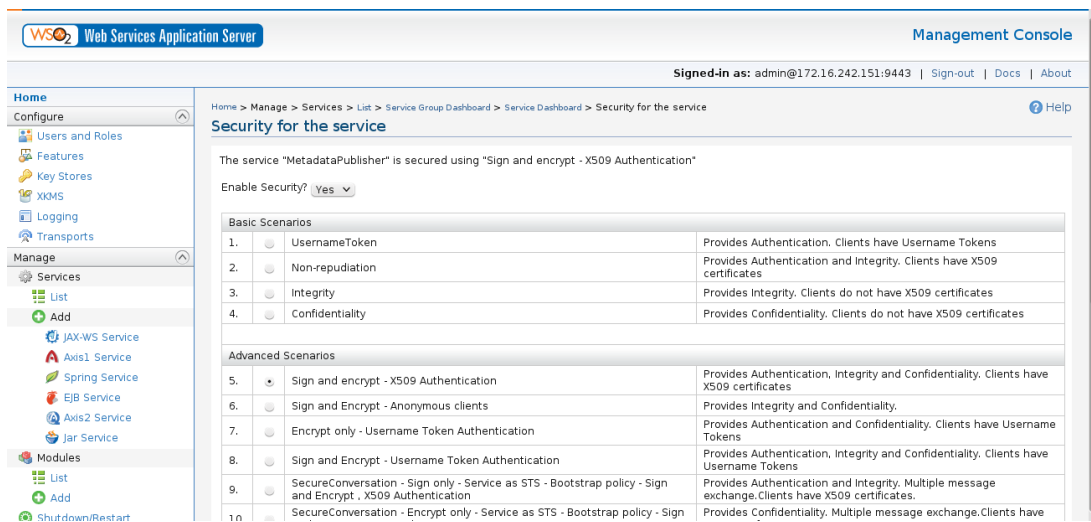


Figure 125 – WSO2 Application Server security management panel

8 Appendix B – SMGW Hardware specs and delivery details

The CockpitCI SMGW consists of a VMware virtual machine containing a Linux OS environment (Suse 11.3 distribution) with embedded all software components and applications required to run the SMGW included the Tomcat server, the WSO2 middleware and MySQL databases.

Virtual Machine VMware using about 9GB of disk space with the following features:

- CPU Information
 - Processor (CPU): Intel(R) Core(TM) i5-3427U CPU @ 1.80GHz
 - Speed: 2,293.98 MHz
 - Cores: 2
 - Memory Information
- Total memory (RAM): 2.0 GiB
 - Free memory: 939.9 MiB (+ 532.9 MiB Caches)
 - Free swap: 0.0 KiB
- OS Information
- OS: Linux 2.6.34.7-0.7-default i686
 - Current user: smgwroot@linux.site
 - System: openSUSE 11.3 (i586)
 - KDE: 4.4.4 (KDE 4.4.4) "release 3"
- Display Info
 - Vendor: VMWare, Inc.
 - Model: VMWARE0405
 - 2D driver: vmware
 - 3D driver: swrast (No 3D Acceleration) (7.8.2))

The SMGW was made available to the partners by means of a VMware image downloadable by an FTP server (Figure 126):



Figure 126 – Folder on FTP site

The image has been available both in .zip format or in OVF virtual machine open standard exchange format. The folders showed in Figure 126 contain the same virtual machine and they differ only for the release format.

Using .zip format, after download and unzip the CockpitCI_SMGW.zip file, the target directory contains the files shown in Figure 127. The virtual machine can start by double click on CockpitCI_SMGE.vmx file.

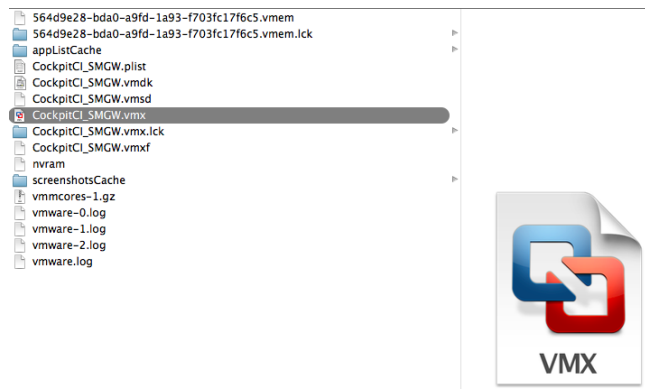


Figure 127 - Virtual machine directory

By using OVF format the whole folder is downloaded, in Figure 128 the content of the folder is shown, and the SMGW can be restored on a new hypervisor by selecting the CockpitCI_SMGW.ovf file.

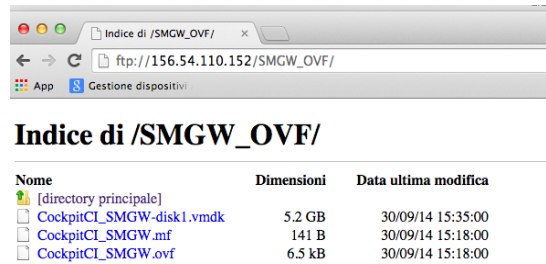


Figure 128 - SMGW repository using OVF standard

9 Appendix C - Installation of OpenVAS using Kali Linux

Kali Linux [6] is available in the following formats:

- ISO files based on system architecture (x86 and x64)
- VMware images
- ARM images

Kali can be either installed as a dual boot with your existing operating system, or it can be set up as a virtual machine. Basic requirements of the virtual machine are:

- Minimum 12 GB of hardware space
- At least 1 GB RAM for optimum performance
- Bootable device such as an optical drive or USB

Once requirements have been checked the bootable ISO can be downloaded from its official website: <http://www.kali.org/downloads>

Once the bootable media are ready, the system can be restarted and booted from our disk/USB. A screen similar to the following will appear (Figure 129):

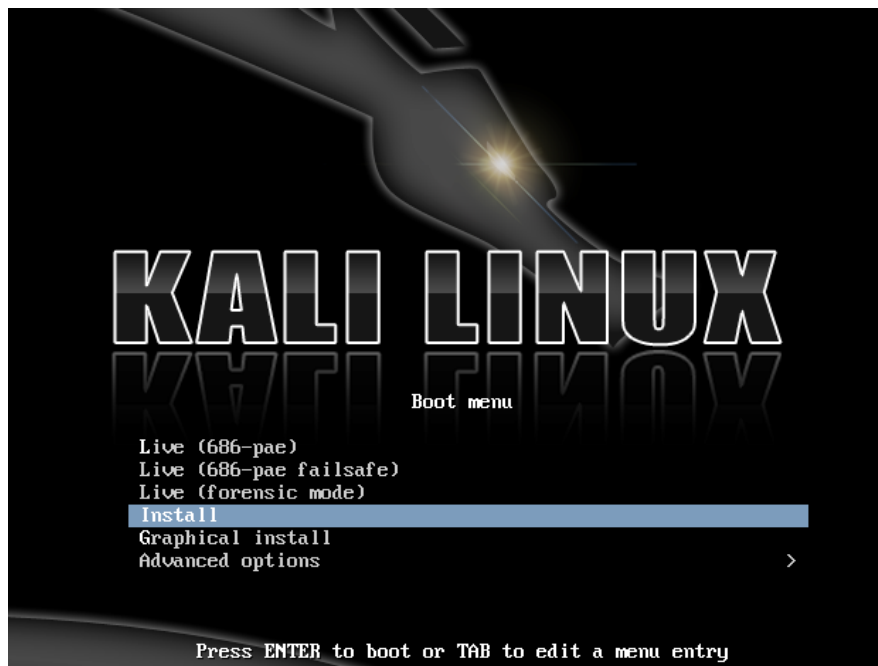


Figure 129 – Kali Linux start page

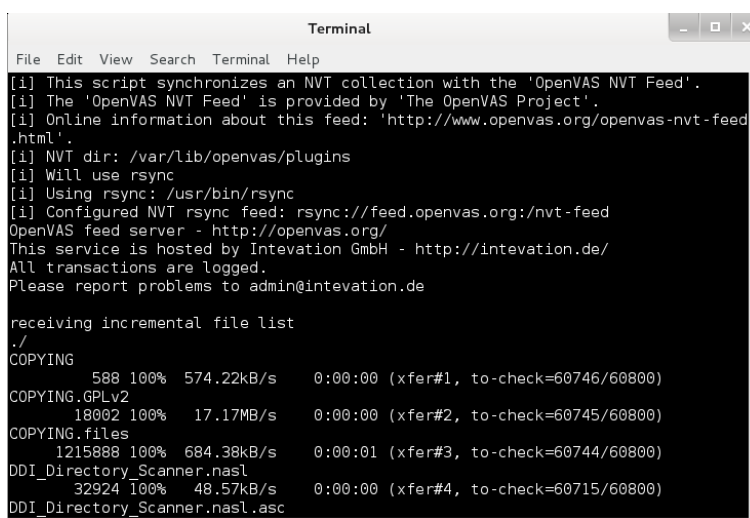
The installation will begin by selecting the Live boot option. The operating system will start loading and, within a few minutes, the desktop is loaded with the main menus:

- Applications
- System Tools
- Administration

Vulnerability assessment tools help a tester in analyzing vulnerabilities and weaknesses in the current system. Vulnerability assessment can be performed over a variety of services and software based on the requirement.

OpenVAS is an open source vulnerability-scanning framework specifically designed to dig out vulnerabilities under various scenarios. To start working with OpenVAS, browse to Applications|Kali Linux|Vulnerability Analysis|OpenVAS.

The first time OpenVAS starts the `openvas-setup` script provides the update of the software and launches all the required plugins and dependencies (Figure 130).



```
Terminal
File Edit View Search Terminal Help
[i] This script synchronizes an NVT collection with the 'OpenVAS NVT Feed'.
[i] The 'OpenVAS NVT Feed' is provided by 'The OpenVAS Project'.
[i] Online information about this feed: 'http://www.openvas.org/openvas-nvt-feed.html'.
[i] NVT dir: /var/lib/openvas/plugins
[i] Will use rsync
[i] Using rsync: /usr/bin/rsync
[i] Configured NVT rsync feed: rsync://feed.openvas.org:/nvt-feed
OpenVAS feed server - http://openvas.org/
This service is hosted by Intevation GmbH - http://intevation.de/
All transactions are logged.
Please report problems to admin@intevation.de

receiving incremental file list
./
COPYING
  588 100% 574.22kB/s 0:00:00 (xfer#1, to-check=60746/60800)
COPYING.GPLv2
 18002 100% 17.17MB/s 0:00:00 (xfer#2, to-check=60745/60800)
COPYING.files
 1215888 100% 684.38kB/s 0:00:01 (xfer#3, to-check=60744/60800)
DDI_Directory_Scanner.nasl
  32924 100% 48.57kB/s 0:00:00 (xfer#4, to-check=60715/60800)
DDI_Directory_Scanner.nasl.asc
```

Figure 130 - OpenVAS installation on Kali Linux

To begin the operations with OpenVAS a new user must be created by using the following command in the shell:

```
root@kali:~#openvas-adduser
```

The rule creation process can be skipped by pressing `Ctrl + D`. The following command updates the framework with new signatures and dependencies:

```
root@kali:~#openvas-nvt-sync
```

Browse to Applications|Kali Linux|Vulnerability Analysis|OpenVAS|openvas-gsd

This will launch the GUI (Figure 131) framework and prompt for the login details on url `https://localhost:9392`. The tester has to enter the credentials and provide the server address.

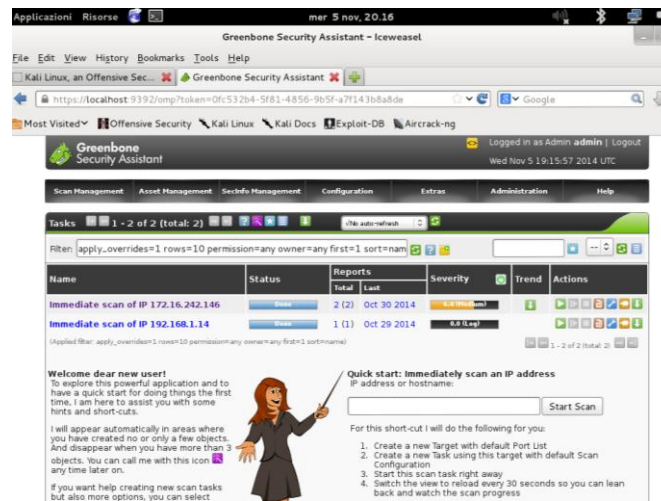


Figure 131 - Greenbone start page