A K/N Attack-Resilient ICT Shield for SCADA Systems, with State Based Attack Detection

I. Nai Fovino, A. Carcano, M. Guglielmi, M. Masera
Institute for the Protection and Security of the Citizen
Joint Research Centre, EU Commission
Via E. Fermi 1, 21027 Ispra, Italy
igor.nai@jrc.ec.europa.eu

Abstract—The security of Critical Infrastructures has become a prominent problem with the advent of modern ICT technologies used to improve the performance and the features of Process Control Systems. Several scientific works have showed how Supervisory Control And Data Acquisition Systems (SCADA), i.e. the systems that control industrial installations, are exposed to cyber-attacks. Traditional ICT security countermeasures (e.g. classic Firewalls, Antiviruses and IDS) fail in providing a complete protection to these systems since the needs of SCADA systems are different from those of traditional ICT for which security tools have been developed (Office PCs, TCP/IP communication protocols etc.). In this paper we present an innovative approach to the protection of SCADA systems based on three key concepts: Critical State based event correlation, SCADA protocols filtering and K-survivability.

Keywords: SCADA systems, security

I. INTRODUCTION

Modern Industrial systems (e.g. power plants, water plants, smart grids, chemical installation etc.) make large use of ICT technologies. In the last years, those systems started to use public networks (i.e. Internet) for system-to-system interconnection. As a result, thanks to this architectural advance, it has been possible to provide new services and features (implementation of the Energy Market, Energy Smart Grids, remote maintenance and optimization, self orchestrating distributed industrial systems etc.). However, this connectivity has exposed industrial installations to new sources of possible threats. As described by Nai et. al [5] [1], such infrastructures are exposed to ad-hoc created attacks aiming at interfering with, and in some case, taking the control of the process network of the industrial installation. The core of industrial installations is traditionally the “so called” SCADA (System Control and Data Acquisition). SCADA protocols and architectures are dedicated to very specific functions: those useful for controlling the operation of technical systems. Due to their characteristics they can be differentiated from classical ICT devices, protocols and systems. For that reason, at the present time, traditional ICT security technologies are not able to protect industrial installations in an adequate way against ad-hoc SCADA-tailored attacks. We defend that a new set of dedicated ICT security technologies needs to be designed in order to protect industrial critical installations. In this work, we focus our attention on a novel ICT security architecture which put together protocol filtering techniques, signature based verification and intrusion detection. In particular the intrusion detection technique used is the result of our recent studies in this field [13]. The paper is organized as follows: in section II we provide a brief overview of the state of the art in the field of SCADA security, while in section III we provide an overview of the vulnerabilities of SCADA systems that we want to address with the proposed work. In section IV we present the details of our ICT shield; in section V after describing our testbed, we present and discuss the experimental results we have obtained when testing an implementation of the proposed approach.

II. RELATED WORKS

As claimed in the introduction, only recently the security of SCADA systems assumed an ICT perspective. Adam and Byres [2] presented an interesting high level analysis of the possible threats affecting a power plant system, a categorization of the typical hardware devices involved and some high level discussion about intrinsic vulnerabilities of the common power plant architectures. A more detailed work on the topic of SCADA security, is presented by Chandia, Gonzalez, Kilpatrick, Papa and Shenoi [6]. In this work, the authors describe two possible strategies for securing SCADA networks, underlying that several aspects have to be improved in order to secure that kind of architecture. A relevant part of the vulnerabilities of SCADA systems is due to the specialized communication protocols they use to communicate with the field devices (e.g. Modbus, DNP3, Fieldbus etc.). Some work has been done about the security of such specialized communication protocols: for example, Majdalawieh, Parisi-Presicce and Wijesekera [7] presented an extension of the DNP3 protocol, called DNPsec, which tries to address some of the known security problems of such Master-slave control protocols (i.e. integrity of the commands, authentication, non repudiation etc.). at the same way, the DNP3 User group proposed a “Secure DNP3” implementing authentication mechanisms for certain type of commands and packets. This approach is extremely close to the one adopted in the IEC 62351-5 standard. Nai et al. [3] presented a secure implementation of the Modbus protocol aimed at introducing integrity, authentication and anti-replay mechanisms in the control flows based on the Modbus protocol. Similar approaches have been presented also by Heo, Hong, Ju, Lim and Hyun [8] while Mander,
Navhani and Cheung [9] presented a proxy filtering solution aiming at identifying and avoiding anomalous control traffic. The proposed solution is extremely interesting, however it does not protect the system against two particular scenarios: (1) The scenario in which an attacker is able to inject malicious packets directly in the network segment between the proxy and the RTUs, and (2) The scenario in which both the proxy and the master have been corrupted and collaborate in order to damage the process network. Finally, Nai et al. in [14], presented an architecture integrating a mesh of distributed packet filtering mechanisms based on signatures, with cryptography based integrity mechanisms. While on the one hand this architecture constitutes a first, significant improvement in the security of control systems, since it introduces the use of specialized firewalls able to analyze the Modbus protocol, on the other hand the filtering feature is only capable to block what we can define as “atomic attacks” constituted by a single not licit packet. The present work extends this architecture, introducing the concepts of Critical State based Filtering analysis and Multilayer Critical State Based and Proactive Monitoring.

III. SCADA VULNERABILITIES OVERVIEW

SCADA systems are widely used in industrial installations to control and maintain field sensors and actuators. The basic bricks of a SCADA system are:

- **Master Terminal Unit (MTU)**: The MTU presents data to the operator, gathers data form the remote PLCs and actuators site, and transmits control signals. It contains the high level logic of the industrial system under control.
- **Remote Terminal Unit (RTU)**: it acts as a slave in the master/slave architecture. Sends control signals to the device under control, acquires data from these devices, receives commands from the MTU and transmits the data gathered to the MTU. An RTU may be a PLC.

The core of the control flow of every SCADA system is the communication protocol (e.g. Modbus, Profibus, DNP3 etc.). By using these protocols it is possible, for example, to force the opening of a valve, etc. In this paper we concentrate our attention on two different industrial protocols: Modbus and DNP3. More specifically, we consider their most recent evolution, i.e. their TCP/IP version. In what follows we provide an overview of the typical vulnerabilities of these two protocols, which are, however, quite similar to the vulnerabilities affecting the other commonly used SCADA protocols (Profibus, Fieldbus etc.). The porting of Modbus and DNP3 over TCP/IP has introduced new layers of complexity for managing the reliable delivery of control packets in an environment with strong real time constraints. In addition, it has opened new possibilities to attackers motivated to cause damages to target industrial systems. In particular, those protocols:

1. **Unauthorized Command Execution**: The lack of authentication between Master and Slave can be used by attackers to forge packets which can be directly sent to a pool of slaves.
2. **SCADA-DOS**: On the basis of the same principle, an attacker can also forge meaningless Modbus/DNP3 packets, always impersonating the Master, and consume the resources of the RTU.
3. **Man-in-the-Middle attacks**: The lack of integrity checks allows attackers to access the production network for implementing typical Man-in-the-Middle (MITM) attacks, modifying the legal packets sent by the master.
4. **Replay-Attacks**: The lack of anti-replaying mechanisms allows attackers to re-use captured legitimate Modbus/DNP3 packets.

Finally, on top of these classes of attacks, since anti-repudiation mechanisms are not implemented, it is hard to proof the trustworthiness of malicious Masters, which could have been compromised.

IV. A SECURE SURVIVABLE ARCHITECTURE

As stressed in the previous section, the weaker point of a SCADA architecture is the communication channel between SCADA servers and PLCs. As showed for example in [1], by taking advantage of those vulnerabilities, a motivated attacker could be able to gain the control of the process network of an industrial installation. To mitigate this threat there are two possibilities: (a) to completely redesign the communication protocols, or (b) to wrap the current architecture into a sort of security shield, limiting as much as possible the impact of this new layer of the performance of the system.

![Diagram](image-url)
In this work we decided to follow the second option. In the following we briefly describe how the new proposed architecture (shown in figure 1) works, leaving to the next sections the detailed description of each element. When the master needs to send a command to a slave, instead of sending the normal SCADA protocol packet, it builds a new secure SCADA packet, containing a Time Stamp (to avoid replay-attacks) on top of the original SCADA packet, and signed with the private key of the Master. Since, on the other side of the communication channel, the slave will execute the command only if the signature of the master and the time stamp are valid, this mechanism prevents the risk of malicious packets injection by an attacker having access to the process network. However, if an attacker is able to directly corrupt a master, the authentication mechanism we have introduced will be easily circumvented. For that reason, we introduce a “filtering gap” between the Master and the slaves. This gap hosts a set of special filtering units (FU) we have designed and developed on the basis of a new concept of “Critical State based filtering”. Differently from the work presented in [14] by Nai et al., here the filtering units operates at two different levels: (1) signature based single packet filtering (to block single malicious packets), (2) Subsystem Level Critical State detection (which allows detecting complex attack patterns based on the use of chains of licit commands). As it is possible to see in figure 1, the filtering gap contains a set of different FU. In our architecture in fact, to mitigate the risk of collusion between a corrupted master and a corrupted FU, we introduce the concepts of system diversity and resilience. Roughly speaking, instead of having a single FU, in the filtering gap we introduce N different FU, where for different we intend that they work on different platforms, operating systems etc. The Master broadcasts the signed SCADA packets to all the FUs, which independently validate the signature of the Master, the time stamp, and then filter the packets according to their internal “single packet” and “Critical State” filters. If the packet passes all the checks, each FUs signs the original packet also with its own private key and forwards the packet to the proper destination. The slave verifies the signature of the FU, verifies the signature of the Master, and executes the command only if at least K/N FUs agree on the safeness of the packet. In this way an attacker, in order to make the system execute a malicious set of packets, has to corrupt the Master and at least K+1 filtering units.

Unfortunately, real industrial installations are composed of several subsystems, with different Masters and several slaves (up to thousands). Each subsystem might be in a safe state, but some of its configurations, as side effect, could put another subsystem into a critical state. In that scenario, the presented approach might fail. For that reason, taking advantage of our recent work in the field of multi-layer Intrusion Detection for industrial systems [13] [15], we have introduced in our architecture a specialized set of Critical State based IDS, one for each subsystem. They are able to monitor the state of each subsystem, share information and aggregate the local events, in order to predict whether a set of commands, which are safe for a target subsystem, can have a negative impact, as side effect, on another subsystem. If this is the case, the central aggregator, is able to act directly on a dedicated gateway to interrupt the potentially malicious flow.

In what follows we provide some details about the key elements of the proposed architecture.

A. Integrity-Authentication Layer

As underlined in the previous section, one of the most relevant weaknesses of the SCADA protocols is the lack of authentication and integrity mechanisms. Only recently, for example with the introduction of Secure DNP3, there is some improvement in this field. However, solutions such as Secure DNP3 modify the specification of the protocol, posing serious questions about their applicability on existing installations. On the other hand the use of typical ICT tunneling mechanisms (ipsec, security features of IPv6, SSL, TLS) could introduce latencies and interferences in the process flow, which might not be acceptable. For that reason we created a lightweight authentication mechanism which embeds the original SCADA packets into a minimal security envelop, which from one side make use of the classic authentication schema, and on the other uses only the minimum number of security features needed for our purposes. To explain our approach, we present the case of Modbus, but the same principle has also been applied to DNP3.

A SCADA system using Modbus TCP/IP embeds a standard Modbus data frame into a TCP frame, without the Modbus checksum. On top of the standard Modbus packet it is introduced a dedicated 7-byte header called MBAP (Modbus Application Protocol header), containing a transaction identifier, a protocol identifier, the packet length and a unit identifier. In the following we describe how we have introduced a “security layer” in this protocol, taking as entry point the security features we consider relevant to protect the process control flow.

![Fig. 2: Secure Modbus Application Data Unit](image)

- **Integrity**: the integrity is guaranteed by using a secure hashing function SHA2. The digest obtained is used to verify the integrity of the packet. For being able to verify it, the digest is sent with the original packet.
- **Authentication**: by adopting a signature scheme (e.g. a public/private key signature scheme), authentication will be enforced, since only the owner of the private key will be able to sign the digest. Of course, this assumption is weak when the owner of the private key does not adequately protect it.
• **Non-repudiation**: a side-effect of the use of a signature scheme in a communication protocol is the introduction of a non-repudiation mechanism. In theory only the owner of a private key can send a target message/command. Since this message will be processed only if the signature is valid, as a consequence a command will be executed only when its legitimate origin can be validated.

• **Anti-replay protection**: In order to avoid the scenario in which an attacker re-uses a “pre-captured” Modbus packet signed by an authorized actor of the communication, the protocol needs a method to discriminate between a “new packet” and an “used packet”. The lightest way to achieve this goal is the use of a *time-stamp* incorporated into the ADU, as shown in figure 2. The time-stamp will be used by the receiver of a target packet in combination with an internal “time-window” in order to check the validity of the packets.

The presented schema (figure 2) is able to guarantee a high level of security without impacting on the normal process control flow. Moreover, as it is possible to see in figure 1, the security envelope can be seamlessly applied by using user-space applications intercepting the Modbus packet (master-side) or by adopting a validation gateway connected transparently between the PLC and the rest of the network. This solution can be therefore applied on all the existing architectures without any significant impact.

### B. Critical State Filtering System

The introduction of the authentication and integrity layer is completely useless when one of the actors at the edges of the communication flow is corrupted; if an attacker takes the control of a master for example, he will be able to sign in a licit way malicious packets. For that reason we have introduced a new entity in the architecture that independently checks the safeness of the packets exchanged - which is based on the identification of the Critical States of the system. Traditional firewalls generally fail in fulfilling this task in the SCADA context for the following reasons:

- The SCADA protocols (in our case Modbus) are dedicated application level protocols. Traditional firewalls do not usually implement any analyzing function for the SCADA protocol payload.
- The heuristic engines of firewalls and IDSs have not been developed to identify malicious behaviors of SCADA_over_TCP protocols.

In any case even the development of a firewall able to analyze SCADA protocols might not be enough to protect process control systems. The following example clarifies this statement: consider a system with a pipe $P_1$ in which flows high pressure steam. The pressure is regulated by two valves $V_1$ and $V_2$. If an attacker is able to inject command packets in the process network, it could, for example, send a packet to the PLC controlling the valve $V_2$ to force its complete closure and a command to the PLC controlling the valve $V_1$ in order to maximize the incoming steam. These two operations, taken separately, are perfectly licit. However, if sent in sequence, they are able to put the system into a critical state since the pressure in the pipe $P_1$ will became soon too high and the pipe could explode. To solve this problem we developed an innovative filtering technique along the following lines:

1) The core of every industrial system is the *process network*, and the core of each industrial process network is the *SCADA system*. The SCADA system controls the process running inside the industrial system. In this way, by monitoring its activity it is possible – at least in principle – to control the activity of the entire industrial system.

2) Every industrial system is, when designed and deployed, well analyzed and all the possible “unwanted” states are usually identified. These unwanted states are what we identify as *critical states* i.e. system states which can be dangerous for the industrial system.

3) The data flowing among masters and slaves of a SCADA system can be used to reconstruct the *virtual image* of the state of the monitored system. We can thus compare such “virtual state” with the critical states to be avoided. Furthermore, upon tracing the evolution of the virtual state, we can predict whether the system is evolving into a critical state.

4) We model the industrial system in the following way: we identify a set of critical states for each subsystem composing the industrial system and we describe the dependencies among the different subsystems in such a way that we can then efficiently monitor the state of a (possibly very complex) system. In this way, we are able to detect many types of attacks. The effectiveness of this approach depends upon the granularity used in the representation of the virtual state and on the effects that such attacks can have on the evolution of those states.

Technically speaking, a CS-Filtering Unit is composed of the following elements:

- **Integrity-Authentication checker**: it verifies time-stamps and signatures of the analyzed packets.
- **A Virtual System**: a virtual system is a collection of software objects which simulate the active elements of the system monitored by the FU. It is built automatically by the FU on the basis of a system description (written using a formal language we have created for the purpose [15]). The Virtual System is kept alive using the data flowing between master and slaves.
- **Master Emulator**: To avoid the risk of divergence between the real system and the virtual system, the FU embeds a Master emulator able to periodically query the field network about its own state.
- **CS-State Checker**: it monitors the evolution of the virtual system in search for the occurrence of critical states triggered by a certain chain of packets.
- **Firewall**: it blocks data flows which have been detected as malicious by the CS-State Checker.
More details about this technique and the performance obtained can be found in [15]. As previously described, the use of a single filtering unit exposes the system to the risk of collusion between a corrupted master and a corrupted FU. For that reason in our architecture we have introduced the concept of diversity and \( \frac{K}{N} \) resilience. Instead of using a single filter, we use a mesh of independent filters, installed on machines using different operating systems and setups, in order to diminish the possibility of having an attacker able to corrupt simultaneously all the FUs. Moreover, the Master sends the request to all the FUs, and only if \( K \) of them agree on the safety of a packet, the slave will execute the command contained in the packet.

Here below we describe the different steps required by the introduction of the multiple FUs architecture:

1) The master (according to the new protocol specifications) composes the Modbus request \( M_{req} \) with the time stamp and the slave’s address.

2) The master calculates the digest through the hash function SHA2.

3) The Master signs the Modbus request/response digest with his private key \( PK_{m} \), and sends it to the \( N \) Filtering Unit

\[
M_{rd} = (TS|Modbus), Enc\{SHA2, PK_{m}\} \quad (1)
\]

\[
= (TS|Modbus), Enc\{C, PK_{m}\} \quad (2)
\]

4) Each FU, independently validates the Modbus request using the Master’s public key \( SK_{m} \)

\[
M_{r} = ((TS|Modbus), Dec\{C, SK_{m}\}) \quad (3)
\]

5) Each Filtering Unit (FU) analyzes the Modbus request destination and function; if the packets contains a forbidden address or a forbidden instruction, the FU adds it into a dedicated stack of “malformed packets”.

6) Each FU checks whether the command brings the virtual image of the system into a critical state. In this case it blocks the packet and sends an alert to the Critical State Correlation System

7) Each FU, taking as feed the analyzed traffic and by querying periodically the field network, keeps updated a digital representation of the system physical state.

8) If the packet is considered safe, each FU signs it with its private key \( PK_{f} \) and forwards the packet to the slave.

\[
M_{r,f} = (M_{r}, Enc\{SHA2(M_{rd}), PK_{f}\}) \quad (4)
\]

9) The slave (PLC) validates the Modbus request filtered \( (MrF) \) using the Filtering Unit Public Key \( SK_{f} \)

\[
M_{r} = Dec\{SHA2, SK_{f}\} \quad (5)
\]

10) The slave (PLC) validates the Modbus request/response \( (M_{r}) \) using the Master’s Public Key

\[
M_{req} = Dec\{SHA2, SK_{m}\} \quad (6)
\]

11) The slave stores the messages in a special stack for then executing the command if and only if it receives the same packet from \( K \) filtering units; otherwise after a predefined time it trashes the messages.

A key point for the robustness of this architecture is the minimum value \( K \) of agreeing responses which trigger the execution of a command. This value has to be tailored to the requirements of the different installations in which the architecture is inserted.

C. Critical State IDS proactive monitoring

An Intrusion Detection Systems (IDS) is generally composed of a set of distributed sensors, analyzing on the fly certain characteristics of the system being monitored, in search for evidence of attacks. The scientific literature of IDS is extremely prolific, and all the existing solutions can be grouped into two classes: (a) Anomaly Based IDS and (b)Signature Based IDS. The work of Vollmer and Manic [16] can be considered an example of the class (a), while Snort [11] is an example of the class (b); however, in the field of Industrial Control Processes and SCADA, it is practically non-existent. Only recently, Digitalbond [12] released the first set of signatures for Snort [11] to analyze SCADA traffic on the basis of single packets. Nai et al. in [13] and [15] presented an innovative approach to intrusion detection, based on the same concept of the Critical State Based Filtering Units. As described before, by providing an Network IDS sensor with a “virtual image” of the subsystem under control which is fed by the flow between SCADA master and slaves, it can directly monitor the state of an entire subsystem identifying possible complex attacks aiming at driving the sub-system into a critical state. In our case, we adopted this approach in a multi-layered configuration. In other words, each CS-IDS sensor monitors a subsystem controlled by a single Master, using as feed the traffic flowing from the Master to the filtering gap, from the filtering gap to the slaves and vice-versa. Each sensor, when detecting an operation that potentially might have an effect on other subsystems, sends an alert to a Critical State Correlation System (CSCS). This can be considered as a CS-IDS sensor containing a more abstract virtual-system that “captures” the causal relations between different subsystems. The CSCS, analyzing the evolution of the sub-system critical states, is able to detect Global Critical States, and on the basis of a set of Critical-State recovery rules, can directly take action to mitigate the effects of the attack. For instance, under some conditions, the CSCS can for example communicate with the PLC-Gateway, asking to interrupt a certain commands flow, or can directly send commands to the field network (emulating a SCADA master), to mitigate the effects of the attack. Moreover, since the CS-IDS sensor receives as input both the traffic entering into the filtering gap and coming out from it, by comparing the two flows it is able to detect whether a FU is corrupted or collaborating with a corrupted Master. The CS-IDS sensor is described more in detail in [13] while the formal language used to describe the high level critical states is described in [15].
D. CS Rules

The language we have designed to represent the critical states of the system (i.e., the rules of the IDS/Filtering unit) has the form \( \text{condition} \rightarrow \text{action} \), where \( \text{action} \) represents an alert. The remaining part of the rule, \( \text{condition} \), is a boolean formula, composed by conjunctions and/or disjunctions of predicates describing what values can be assumed by the different PLCs’ components.

The PLC’s elements taken into account by our rule language are coils \( C \), registers \( R \), digital inputs \( DI \), digital outputs \( DO \), analog inputs \( AI \) and analog outputs \( AO \).

The difference between traditional IDS rule languages and our language, is that in the last case, the predicates (upon which the condition that triggers the corresponding rule is formed) are defined over the states of the PLCs while in the traditional signature based rule languages, the content of a packet is represented.

In fact, upon interception of a packet, the IDS updates the SCADA system state, changing the parameters of the PLCs (or PLCs) to which the packet is addressed, according to the information contained in the packet payload.

Then, the IDS checks whether some PLCs’ configuration as represented in the resulting state triggers a rule. If this is the case, the corresponding action – prescribed by the rule – is performed. In the following we describe the rule language using the standard BNF notation:

\[
\text{(rule)} := \text{(condition)} \rightarrow \text{(action)} \; \text{(condition)} := \text{(predicate)} | \text{(predicate)} (\text{conn}) \text{(predicate)} \\
\text{(predicate)} := \text{(term)} \text{(relation)} \text{(term)} \\
\text{(term)} := \langle \text{PLCName} \rangle \; \text{(value)} \\
\langle \text{PLCName} \rangle := \text{PLC}(\text{number}).\langle \text{comp} \rangle \langle \text{number} \rangle \\
\langle \text{action} \rangle := \text{Alert} | \text{Log} | \text{Look}(\text{rule}) \\
\langle \text{conn} \rangle := \text{and} | \text{or} \\
\langle \text{relation} \rangle := \leq | \geq | < | > \; = \\
\langle \text{comp} \rangle := C | R | DI | DO | AI | AO \\
\langle \text{value} \rangle := 0|...|2^{16} - 1
\]

Consider the following rule stating that if coil \( C23 \) of PLC1 has value 0 and coil \( C17 \) of PLC2 has value 1 (corresponding, respectively, to open Valve 1 and close Valve 2), then the IDS performs an Alert action of the last packet that changed the state \( \text{PLC1}.C23 = 0 \text{ and } \text{PLC2}.C17 = 1 \rightarrow \text{Alert} \). Thus, a packet addressed to PLC1 containing a command for switching coil \( C23 \) to 0 – given that the other condition stated in the rule is satisfied – will trigger an Alert action.

Again, note that the switching request (local to PLC1) could be a perfectly legal one, but it could become critical when other, non-local conditions are tested.

V. EXPERIMENTAL TESTS

The described architectural approach has been fully implemented in a working prototype. We have developed the CS-filtering units, the CS-IDS sensors, the CS-correlation system. Moreover we have modified the TCP/IP stack to support the secure Modbus protocol. In what follows we describe the experimental facility used to test this architecture.

Additionally, in the following subsection we present the first results obtained.

A. Experimental Platform

The architecture presented was tested in the SCADA security laboratory of our research institute. This laboratory has a protected environment that reproduces the dynamics and the configurations of different industrial systems. In the configuration used for our tests, we reproduced the typical architecture of a turbo-gas power plant. Figure 3 shows the high level schema of the facility. Thanks to a large campaign of analysis of an existing turbo-gas power plant [4] the elements composing the experimental environment are very close the reality. The field network, i.e. the most physical part of the emulated plant, is composed of a mix of real PLCs (ABB AC800) and virtual PLCs (software PLCs which we have developed in order to increase the number of possible scenarios and system configurations). The emulated system is monitored by a set of sensors (host based and network based), for gathering as much information as possible during each experiment.

B. Experimental Results

The purpose of the experiments was to measure the impact of our architecture on the communications performance between a Master and a Slave when executing a Modbus transaction. A Modbus transaction is a request/response message exchange. The duration of a transaction is the time measured between the request sent by the master and the receipt of the slave response. The part of our architecture that affect more
the Master/Slave transaction is the "Integrity-Authentication Layer". For this reason we made the experiment described in Figure 4. The Master station sends a request message to the

Slave station which responds with the appropriate response message. In the middle there is one filtering unit that performs the authentication and integrity checks as described before. We used two simulators written in C# under Windows XP SP3 to simulate the master and the slave; all the machines involved mount an AMD Athlon(tm) 64 X2 Dual Core Processor 5800+ 3.01 Ghz and 3.25 GB of RAM. We measured the time elapsed when executing the steps described in section IV, those required to complete our secure transaction. The measured times are shown in Table I.

![Fig. 4: Test schema](image)

![Fig. 5: Experimental Tests](image)

The introduction of authentication and integrity affects the communications performance in terms of:

- **Packet size.**
- **Modbus transaction duration.**

Table II shows the packet size overhead introduced by the use of Secure Modbus.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Modbus size</th>
<th>Secure Modbus size</th>
</tr>
</thead>
<tbody>
<tr>
<td>M → FU</td>
<td>258 bytes</td>
<td>258 + 1 (ts) + 128 (sign M) = 387</td>
</tr>
<tr>
<td>FU → S</td>
<td>258 bytes</td>
<td>258 + 1 (ts) + 128 (sign M) + 128 (sign FU) = 515</td>
</tr>
</tbody>
</table>

**TABLE II: Modbus and Secure Modbus Packet Size**

The original Modbus packet size in Table II is 258 bytes because the packet sent contains the Function Code 15 Write Multiple Coils and the number of coils to read is 1968 (0x07B0 hex is the maximum value allowed according to the Modbus specification [10]). We chose this Function Code for two reasons:

1) It is one of the largest packets possible to build with the Modbus protocol (the maximum is 260 bytes according to the Modbus specification [10]), i.e. this is one of the worst cases in terms of packet size.

2) It involves many coils, so the Filtering Unit has to update the values of 1968 coils and this is also the worst case for the "Virtual System Update ".

As we claimed before, the introduction of authentication and integrity affects also the Modbus transaction duration. The time spent to complete a Modbus transaction is about 2 milliseconds in a network without background traffic. Using our architecture some delays are introduced:

- **Integrity-Authentication Delay:** it is the time spent to calculate the digest and to sign and verify the packet by the Master, Slave and Filtering Units.
- **Virtual System Update Delay:** it is the time spent to update the Virtual System used by each Filtering Unit.
- **Check Rules Delay:** it is the time spent to check the "Signature-Based” and “Critical State-Based” rules by each Filtering Unit.

The “Integrity-Authentication Time” introduces a very large delay, about 42 milliseconds. The “Virtual System Update Time” is a relatively small time and even in the worst case used
in our tests (1968 coils to update); it does not have significant effects upon the transaction time. Regarding the “Check Rules Time” we repeated the test using 6 different sets of rules (10, 50, 100, 500, 1000, 2000). The results of these experiments are shown in the Table III: the first column shows the time for a normal Modbus transaction, the second column shows the time for a secure Modbus transaction (with authentication and integrity checks), and the third column shows the time for a secure Modbus transaction with the filtering rules checks. The last column shows how the elapsed time increases with the number rules. In the chart (Figure 5) we compare the performance of the three architectures (only Modbus, Secure Modbus, Secure Modbus + filtering).

The line at the bottom represents the normal Modbus transaction time; the straight line at the top represents the secure Modbus transaction time and the line with a linear growth represents the time for a secure Modbus transaction with the rules checking. It is not possible to say whether the delay introduced by our architecture can actually affect or not the performance of SCADA network, because this depends on the type of process that the master is controlling. We plan to improve the performance of the presented architecture by using more advanced lightweight cryptographic protocols.

<table>
<thead>
<tr>
<th>Rules</th>
<th>Modbus</th>
<th>Secure Modbus</th>
<th>Secure Modbus + Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.6</td>
<td>43.6</td>
<td>43.6 + 0.007 = 43.67</td>
</tr>
<tr>
<td>10</td>
<td>2.6</td>
<td>43.6</td>
<td>43.6 + 0.016 = 43.76</td>
</tr>
<tr>
<td>50</td>
<td>2.6</td>
<td>43.6</td>
<td>43.6 + 0.058 = 44.18</td>
</tr>
<tr>
<td>100</td>
<td>2.6</td>
<td>43.6</td>
<td>43.6 + 1.09 = 44.69</td>
</tr>
<tr>
<td>500</td>
<td>2.6</td>
<td>43.6</td>
<td>43.6 + 2.66 = 46.26</td>
</tr>
<tr>
<td>1000</td>
<td>2.6</td>
<td>43.6</td>
<td>43.6 + 5.07 = 48.67</td>
</tr>
<tr>
<td>2000</td>
<td>2.6</td>
<td>43.6</td>
<td>43.6 + 9.99 = 53.59</td>
</tr>
</tbody>
</table>

VI. Conclusions

The security of SCADA systems requires new approaches that could afford a workable answer to the more urgent problems. In this paper we propose an architecture combining different security solutions, that altogether are able to respond to the main cyber vulnerabilities. The principle we followed is that it is not realistic to expect dramatic changes in the topological and functional part of the SCADA implementing the control functions. Therefore, any improvement of the security of existing SCADA should be attained by combining elements that make use of existing features and that can be easily integrated. The architecture we propose supplies the following capabilities: authentication and non-repudiation by means of a signature scheme, integrity protection guaranteed by a secure hash function, anti-replay protection from the use of time-stamps, an innovative filtering of potentially dangerous packets, and finally a state-based attack detection mechanism (that acts as an ad-hoc IDS), based on the proactive monitoring of the critical states of the system. The composition of all these security functions result in a security shield providing a complete defence of the SCADA system. For trying the solution, we implemented it and tested it in our experimental platform simulating a power plant. The paper presents in a brief manner some of the results of the tests, which clearly indicate that the proposed solution, while protecting the SCADA system, doesn’t introduce significant delays and therefore doesn’t affect the control functions. In the future we intend to improve the implementation of the architecture for reaching still better performance, and to extend the testing campaign for verifying its robustness against a wide set of attacks.

REFERENCES