SCADA Protocol Obfuscation: A Proactive Defense Line in SCADA Systems

Carlo Bellettini and Julian L. Rrushi
Università degli Studi di Milano
Dipartimento di Informatica e Comunicazione
Via Comelico 39/41, 20135 Milano, Italy.
E-mails: {carlo.bellettini, julian.rrushi}@dico.unimi.it

Abstract: This paper describes a defensive approach referred to as SCADA protocol obfuscation. Such an approach counters network attacks which remotely exploit vulnerabilities in target SCADA systems through malicious messages structured in compliance with an actually used SCADA protocol. SCADA protocol obfuscation is built in the form of a proactive structural intervention which we defined upon information provided by a new conceptual structure called attack requirements trees. An attack requirements tree is a structured means of providing conditions which must hold for a given attack to be feasible. The root node of this tree is the attack itself, while each other node is an attack requirement. We built the attack requirements tree for a SCADA protocol-based attack and identified a node whose denial could make such an attack unfeasible. This node represents the requirement that attack messages should be built according to the SCADA protocol in use, therefore we decided to introduce diversity into a SCADA protocol in order to deny to the attacker the knowledge required for building correct attack messages. While a strong cryptographic algorithm would accomplish this task, we had to investigate another solution as common SCADA field devices such as PLC's or RTU's have limited computational power. Consequently they have considerable difficulties running strong cryptographic algorithms. In this paper we propose SCADA protocol obfuscation as a candidate solution and describe it as applied to the Modbus TCP/IP protocol.

Keywords: SCADA security, attack requirements tree, SCADA protocol obfuscation, proactive intrusion prevention.

1 Introduction

SCADA systems rely on proprietary networks, communication protocols, hardware and operating systems. Furthermore, SCADA used to have limited communication between its components and often dedicated communication channels were used. Consequently such systems were thought of as not being subject to network attacks (Byres & Lowe, 2004). Nevertheless, SCADA networking has been evolving and the advent of low cost computing is leading companies into replacing their proprietary legacy hardware with modern more powerful devices. The interconnectivity between SCADA devices is relatively high and often SCADA systems use public communication channels.

SCADA systems are switching to Ethernet and TCP/IP networks, open standards such as DNP3, Modbus, IEC 60870-5, etc., and modern operating systems such as Windows

or Unix. Research in this field has shown SCADA systems actually may suffer from various kinds of vulnerabilities in their data, security administration, architecture, networks, and platforms (Stamp, Dillinger, Young & DePoy, 2003).

Attacks which remotely exploit arbitrary vulnerabilities in SCADA systems by sending malicious SCADA protocol messages represent a realistic threat to SCADA controlled critical infrastructure. We sought a countermeasure to these SCADA protocol-based attacks by first devising a general model which we refer to as an attack requirements tree whose purpose is to identify the basic requirements needed by an attack to successfully reach one of its objectives. The idea is to defend from an attack by denying one or more of its basic requirements without which such attack is not feasible. Although in the case of the SCADA protocol-based attack the basic requirements tree are quite obvious, we are confident that such a structured means of providing attack requirements may be helpful by assisting in the identification of the actions to take when devising intrusion prevention techniques.

We identified a lightweight dynamic scrambling method as an intervention candidate, which could deny a basic requirement of a SCADA protocol-based attack. We built such a scrambling method upon a parallelism with techniques such as address space layout randomization and instruction set randomization used by an operating system to protect from exploitation of low-level coding vulnerabilities. We experimented with this defensive approach as applied to the Modbus TCP protocol. We chose Modbus as its specification is publicly available. Furthermore there are publicly available Modbus implementations which may be run on publicly available embedded operating systems.

This paper is organized as follows. Section 2 is an overview of several models which like attack requirements trees provide a structured means for characterizing attacks, namely attack trees, fault trees and attack graphs. Section 2 is also gives an overview of research findings which like SCADA protocol obfuscation use a scrambling scheme for the purpose of securing information sent over a network. Section 3 gives an overview of the Modbus protocol which we used as a pattern protocol for applying our defensive approach, and defines what we are trying to protect SCADA devices from. Section 4 describes attack requirements trees, provides the attack requirements tree for a SCADA protocol-based attack and explains how attack requirements trees could be used as basis for building proactive intrusion prevention approaches. Section 5 describes the SCADA protocol obfuscation approach and presents the motivations which led us to devise such a candidate solution. Section 6 describes several techniques which we used to build a transformation scheme within SCADA protocol obfuscation. Section 7 defines the advantages and disadvantages of SCADA protocol obfuscation. Section 8 summarizes our findings and concludes the paper.

2 Related Work

Attack trees (Schneier, 1999) provide a structured means of defining a set of actions which lead to the achievement of an attack goal. The root node of an attack tree represents the goal of an attack. Such an attack goal may be achieved through different ways represented by the nodes which are children of the root node. In general each child node in an attack tree is an attack subgoal and branches of the tree go down till no

further subgoals are possible. A path from a leaf node to the root node is a way of performing the attack and consequently achieving the attack goal. The nodes of an attack tree may be AND nodes or OR nodes. For achieving the goal represented by an AND node all its subgoals must be achieved, while for achieving the goal represented by an OR node at least one of its subgoals must be achieved. Attack trees have been already employed as a systematic method for characterizing the security of SCADA systems.

In (Balducelli, 2003; Balducelli, Vicoli & Jin, 2006) attack trees are used to formalize the propagation paths of attacks for the purpose of proposing a sort of reference language to model and implement attack and fault scenarios in SCADA, and to support an attack tool platform. This also represents a formal strategy to elicit information about vulnerabilities in SCADA systems. In (Byres, Franz & Miller, 2004) the attack tree methodology is applied to SCADA systems based on the Modbus protocol stack. Through attack trees the authors identify attacker goals and vulnerabilities in both specification and typical deployments of such SCADA systems. (Convery, Cook & Franz) employ attack trees to describe possible vulnerabilities in the border gateway protocol. A fault tree (U.S. Nuclear Regulatory Commission, 1981) is similar to attack trees. The root node in a fault tree represents an undesired state and branches are ways which contribute to the undesired state. An attack graph (Scheyner, 2004) is organized like attack trees or fault trees. Nevertheless, an attack graph supports cyclic dependencies or merged states.

Attack requirements trees which we describe in this paper are conceptually similar to attack trees, fault trees and attack graphs as all these paradigms provide a structured means for characterizing attacks. But unlike attack trees, fault trees and attack graphs which reflect possible ways for achieving an attack goal, attack requirements trees provide the conditions for an attack to be possible. Furthermore, attack trees, fault trees and attack graphs are generally used for identifying attacker goals and related vulnerabilities, while attack requirements trees are used for identifying possible proactive interventions which could prevent an attack from taking place. In fact along with attack requirements trees in this paper, we propose SCADA protocol obfuscation or scrambling devised upon information held in an attack requirements tree. To the best of our knowledge no mechanisms functionally similar to attack requirements trees have been proposed in the security literature to date, and this paper is the first to propose a structure which reflects the requirements of a defined attack.

(Li, Ren, Ling & Liang, 2004) proposed a secure scrambling scheme to improve the physical layer built-in security of CDMA systems. Scrambling in wireless networks was initially devised for reducing interference between mobile nodes. Nevertheless, latter on scrambling was deemed adequate for adding security to the physical layer in such networks. The scrambling scheme proposed by (Li et al., 2004) is constructed through AES operations and consists in adding a scrambling sequence to the chip-rate spread signal. The performance cost of such a scheme is reported to be comparable to the performance of existing pseudo-random scrambling schemes used to secure the physical layer of CDMA systems. Scrambling has also been employed to secure speech signals. (Del Re, Fantacci & Maffucci, 1989), for example, proposed a two-dimensional

scrambling algorithm implemented by digital signal processing techniques and a digital signal processor.

The SCADA protocol obfuscation approach described in this paper is similar to the aforementioned research as it applies a scrambling scheme to secure information sent over a network. Nevertheless, unlike the work described in (Li et al., 2004; Del Re et al., 1989), which apply scrambling schemes at the physical layer of OSI stack, the SCADA protocol obfuscation scrambles information at the application layer.

3 Preliminaries

3.1 An Overview of the Modbus Protocol

Modbus is an application layer messaging protocol which enables SCADA devices to communicate with each other in a master-slave fashion within possibly different types of buses and networks (Modbus Organization, 2004). The Modbus protocol defines a Protocol Data Unit (PDU) independent of the underlying communication layers. A PDU is composed of two fields, namely a function code field and a data field. A function code field indicates to a slave what kind of action to perform. A function code field is coded in one byte and valid values are in the rage of 1 to 255 in decimal representation. A function code may come along with sub-function codes in order to define multiple actions. There are 127 function codes which belong to one of the three categories defined by Modbus, namely public function codes, user-defined function codes, and reserved function codes. A data field contains additional information such as register addresses, how many items are to be handled, or the number of bytes in the field, which slaves need to use in order to carry out the task specified by the function code. Nevertheless, in some defined requests the function code alone is sufficient for the slave to perform the specified action, therefore in such requests the data field is of zero length.

Modbus defines three PDUs; namely Modbus Request PDU which is a request message sent by a master to a slave during a transaction; a Modbus Response PDU which is a response message sent by a slave to a master during an error free transaction; and Modbus Exception Response PDU which is a response message sent by a slave to a master in a transaction where due to any reason the slave cannot handle the master's request. The employment of Modbus on specific buses or networks introduces some more fields in addition to the PDU, thus creating a Modbus frame referred to as Application Data Unit (ADU). In Modbus TCP/IP (Schneider Automation, 2004) an ADU is composed of a common PDU and a header defined by Modbus Application Protocol (MBAP). The MBAP header contains a transaction identifier field which identifies a Modbus request/response transaction, a protocol identifier field which is used for intra-system multiplexing and identifies the Modbus protocol, a length field which is a byte count of the following fields, and a unit identifier field which is used for intra-system routing and identifies a remote slave connected on a serial line.

3.2 Threat Model

The defensive approach we discuss in this paper counters SCADA protocol-based attacks, i.e. attacks which remotely exploit any kind of vulnerability in SCADA systems by operating through malicious messages built according to the SCADA protocol in use by target SCADA devices. In this context we define as a vulnerability also the lack of authentication as a consequence of which an attacker is able to send SCADA protocol messages to SCADA devices and succeeds in having them process such messages. The SCADA protocol-based attack deployment strategy varies upon the offensive capabilities of an attacker and particular needs for specialized attack tools. Nevertheless, such attacks are usually carried out either through a rogue device under the attacker's control or through a compromised SCADA field device. In the former case an attacker introduces a rogue device in a SCADA link media and uses it as an attack launching platform while making it appear as a legitimate SCADA device of a defined type. Alternatively, an attacker may physically disable a legitimate SCADA device and replace it with a rogue device, which mimics the behavior of the legitimate SCADA device, but in the meantime carries out the attacks.

SCADA protocol obfuscation is meant to protect a SCADA device from SCADA protocol-based attacks launched from a rogue device as described above. An attacker could gain physical access to a SCADA device such as a remote terminal unit (RTU) or a programmable logic controller (PLC) and use this SCADA device to carry out SCADA protocol-based attacks. We emphasize SCADA devices as potential victims of physical attacks considering that in reasonable SCADA deployments master stations and slave stations sit in highly secured rooms, therefore they may be considered as physically guarded. Nevertheless, we deem the defense from SCADA protocol based attacks launched from a compromised SCADA device falls within physical security competence. One of the most devastating SCADA protocol-based attacks consists in sending commands to SCADA devices, or corrupting a SCADA device with malicious response data. SCADA protocol-based attacks could also consist in sending abnormal or already used packets to exploit protocol level vulnerabilities in design or implementation of a SCADA protocol deployed in target SCADA systems. Further, by having target SCADA devices process malicious packets, an attacker could exploit low-level coding vulnerabilities such as stack-based buffer overflows, heap overflows, integer errors (integer sign errors and integer overflows), or format bugs in C/C++ implementations of a SCADA protocol.

4 Attack Requirements Trees

Intrusion prevention is often thought of as a combination of intrusion detection with intrusion response in this order. An intrusion detection system works on various kinds of data depending on if it is host intrusion detection or network intrusion detection, analyzes such data applying some anomaly or misuse algorithm, and in the case it deems an attack is taking place, triggers an action to counter the possible attack. As even the best intrusion detection systems are subject to false negatives and false positives such an approach of building intrusion prevention does not prevent undetected attacks and can

generate massive responses to non-existent attacks. Furthermore, the fact that response comes after detection could be a drawback, especially in process control systems. For instance, when attacks are detected with some delay due to computational costs or necessity to gather enough data before deciding whether a certain attack is taking place, the response may be activated too late to stop the attack from succeeding.

While an intrusion detection system remains a valuable defense instrument, a functionally independent and dedicated intrusion prevention system could perform better in countering attacks to SCADA systems and could complement the overall defense capability of SCADA systems. Along this line, intrusion prevention could be defined as a proactive structural intervention whose objective is to prevent an exploit from taking place (Bellettini & Rrushi, 2006). The idea behind such a definition is to structure SCADA in such a way that offensive operations are blocked since their very first stage, i.e. they are not allowed to take place at all. Thus, the defensive value provided by intrusion prevention designed in the form of a proactive structural intervention consists in prevention rather than response. We devised a conceptual structure called attack requirements tree which serves as basis for building proactive intrusion prevention approaches. An attack requirements tree is a structured means of providing conditions which must hold for a given attack to be feasible. The root node of this tree is the attack itself, while each other node is an attack requirement.

In this paper when we say that a node is true, we mean that the attack condition it represents holds. Similarly, when we say a node is false, we mean that the attack condition this node represents does not hold. With regard to the root node we say that this node is true if the attack it represents is feasible, and false if such attack is unfeasible. Except for the root node, each other node in an attack requirements tree is qualified either as an AND node or an OR node. Each node is true if all its AND children nodes (if any) are true, and at least one of its OR children nodes (if any) is true. For being able to carry out an attack such as a SCADA protocol-based attack, an attacker may be required to carry out successfully other support attacks such as breaking the physical security of a SCADA device, accessing a SCADA link media with a rogue device, sniffing a SCADA network, reverse engineering an unknown SCADA protocol, social engineering, etc. Consequently the attack requirements tree for an attack may incorporate, in the form of subtrees, the attack requirements trees for other support attacks. Thus, it may happen that an internal node in an attack requirements tree is true only if the root node of the attack requirements tree for some defined support attack is true.

An intrusion prevention approach against a defined attack could consist in a proactive structural intervention which denies, i.e. forces to be false, an internal node in the attack requirements tree of this attack in such a way that the root node of this tree is forced to be false. The arms race between defensive research and offensive research is reflected into efforts for forcing the root node to be false and true, respectively. An attack requirements tree provides for building a multi-layered defense against entire attack categories. As a matter of fact it is possible to force the root node to be false by denying several internal nodes in an attack requirements tree.

Figure 1 shows a sample attack requirements tree for a SCADA protocol-based attack. Dotted arcs denote subtrees which have not been included due to space limitations. From this tree we see that generally a SCADA protocol-based attack requires that the attacker transmits in the language of legitimate SCADA devices, i.e. that attack messages are fully compliant with the SCADA protocol which target SCADA devices are using for their communications.

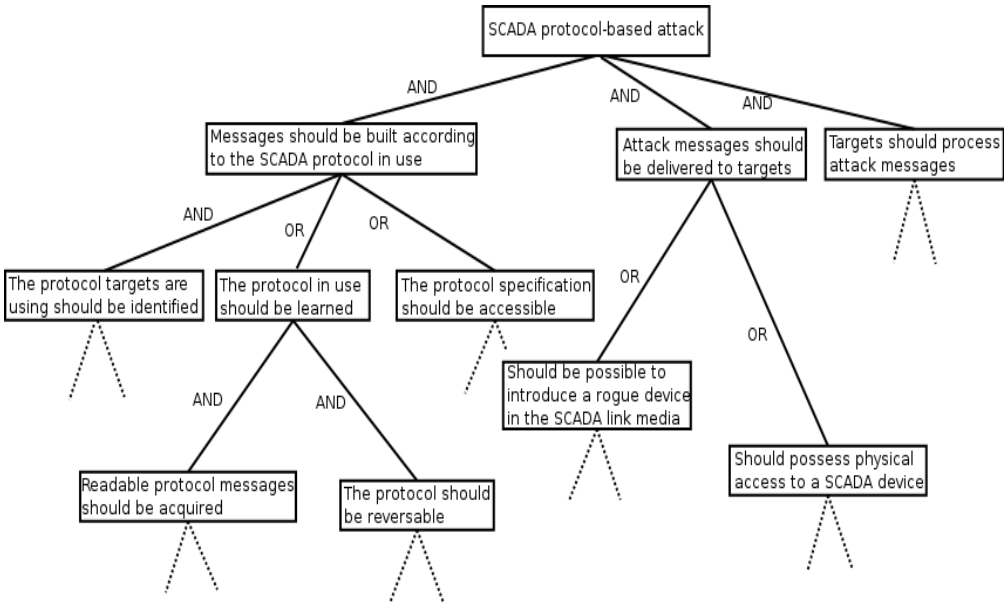

Figure 1 – An attack requirements tree for a SCADA protocol-based attack

In order to be able to construct attack messages in compliance with the SCADA protocol that target devices are using, an attacker first must either know or find out the identity of such protocol. Thus, an attacker is required to know whether the protocol in use is Modbus, DNP3, PROFIBUS, IEC 60870-5, etc., or a proprietary protocol, running over serial line or TCP/IP. Furthermore, an attacker should know the protocol in use by target devices. In order for an attacker to acquire this knowledge the protocol specification should be accessible to him, otherwise the attacker should have the possibility to learn the protocol itself. In the case the protocol specification is not available to the attacker the protocol specification accessibility requirement implies that root nodes of attack requirements trees for other attacks such as for example social engineering or physical intrusion are true. Similarly, the requirement of the possibility to learn the protocol in use by target SCADA devices implies that the root nodes of attack requirements trees for attacks such as sniffing and protocol reverse engineering are true. In general sniffing and reverse engineering along with operational data are used to build targeted attacks (Finco, Lee, Miller, Tebbe & Wells, 2006). Just like any other network attack a SCADA protocol-based attack also requires that an attacker feeds attack

messages to a target device, i.e. delivers through the network the attack messages to a target device. Such a requirement in turn requires that an attacker should either possess already physical access to a SCADA device, usually an IED considering that if an attacker controls a MS there is no need to carry out an attack, or the root node of the attack requirements tree of a physical attack against a SCADA device should be true. In alternative the root node of the attack requirement tree for the attack which consists in introducing a rogue device in the SCADA link media should be true. Nevertheless, building attack messages in compliance with the protocol in use and delivering them to target SCADA devices is not sufficient for a SCADA protocol-based attack to be feasible. In fact due to any reason a target SCADA device could drop attack packets. Therefore, in order for the root node of the attack requirements tree given in (Figure 1) to be true target SCADA devices should process attack messages.

Obviously the general attack requirements tree given in (Figure 1) could be further refined in front of relevant details of a defined SCADA environment and/or particular circumstances under which a SCADA protocol-based attack may be carried out, such as for example whether the SCADA network is wired or wireless, whether a default port is used for a defined protocol rather than custom ports, whether the operating system of SCADA devices provides for security including the physical one, whether the specification of a proprietary SCADA protocol is kept and handled securely, etc. There is a close relationship between attack trees and attack requirements trees. Most of the nodes in an attack requirements tree for a defined attack may be generated upon information contained in the attack tree of the aforementioned attack. On the other hand although attack requirements trees were designed to mainly provide the basis for proactive intrusion prevention approaches, they could help in assigning realistic difficulty values to several nodes in an attack tree. Furthermore, attack requirements trees complement attack trees in fully characterizing a given attack.

5 SCADA Protocol Obfuscation

5.1 The Approach

Seeking inspiration from biology, (Forrest, Somayaji & Ackley, 1997) investigatated the possible advantages of diversity in computing systems, considering that diversity is already a source of robustness in biological systems. (Forrest et al., 1997) followed several guidelines they devised for their ongoing research such as introducing diversity through randomization, introducing diversity in places that will be most disruptive to attacks, minimizing costs and preserving high level functionality. They concluded that diversity is valuable, and it can contribute to the development of more robust and secure computing systems.

SCADA protocol obfuscation is a proactive intrusion prevention approach which introduces diversity in the SCADA communication protocol employed by SCADA systems for supervisory and control. SCADA protocol obfuscation aims at diversifying a SCADA protocol in such a way that the actual obfuscated protocol is unique for each single group of legitimate SCADA devices, and this while preserving the communication functionality of the SCADA protocol.

The SCADA protocol obfuscation approach consists of a reversible transformation of each frame of a given SCADA protocol according to a defined obfuscation algorithm. The obfuscation algorithm is to be shared between all legitimate SCADA devices in a communication group and its implementation is composed of four components, namely a transformation function, a key, a transformation scheme generator, and a key generator (Figure 2). A transformation function obfuscates a regular SCADA protocol message into an obfuscated message or vice versa, i.e. deobfuscates an obfuscated message into a regular SCADA protocol message.

A transformation function is built through protocol obfuscation techniques such as frame layout randomization and field set randomization described in Section 6 possibly complemented by additional optional techniques. The logic of a transformation function optionally includes the use of a key generally when stream cipher scrambling (Section 6) is employed. The SCADA protocol obfuscation approach utilizes a transformation function which is intentionally simple in its functionality and implementable in a few lines of code. This is because a SCADA protocol obfuscation approach should require low computing power, a fact which as we will see latter in this section forms the basis of its very existence.

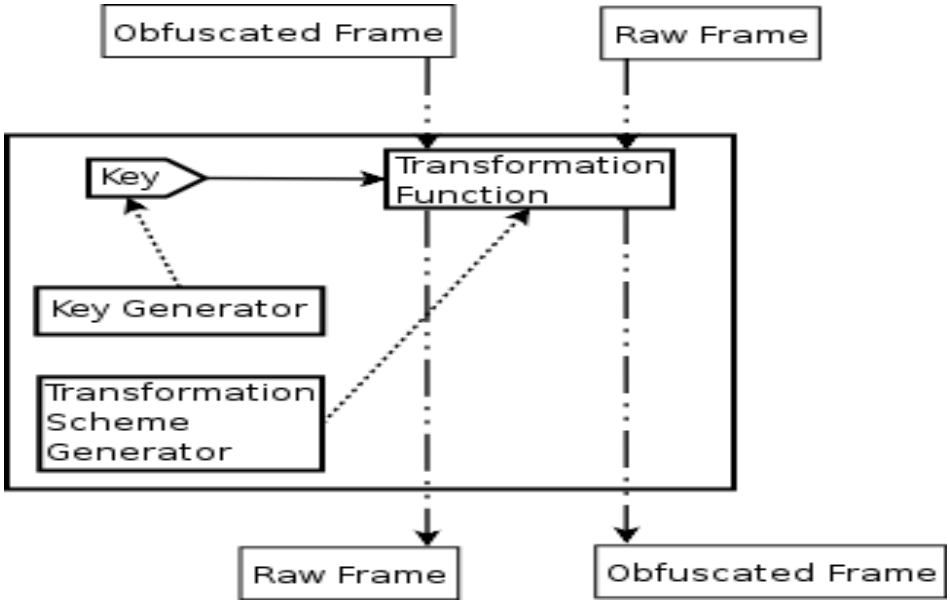

Figure 2 – Conceptual organization of a SCADA protocol obfuscation module

Due to its simplicity in both design and implementation, a transformation function provides an obfuscation scheme which is quite weak and whose breaking is just a matter of time. One of the main ideas behind SCADA protocol obfuscation approach which aims at overcoming such a weakness is to enable Master Stations and Slave Stations to

employ a certain transformation scheme for obfuscating their SCADA protocol messages during a limited amount of time only, say *t* minutes. After *t* minutes protected SCADA devices change the transformation scheme, thus for other *s* minutes they employ a different transformation function for obfuscating their SCADA protocol messages.

A transformation scheme can be simple to the degree of resisting for the limited amount of time during which it is employed for obfuscating SCADA protocol messages. Furthermore, the transformation function depends on a series of variables whose modification allows for substantially changing the obfuscation scheme implemented by such a function. The transformation scheme generator modifies the aforementioned variables each time a SCADA device needs to change obfuscation scheme.

In the case the transformation function employs stream cipher scrambling (Section 6), the key could be continuously varied for the purpose of making it less easy for an attacker to derive the obfuscation scheme. In our experiments, for example, we generated random key bytes by using an interface to the kernel's random number generator for each packet to be sent. Thus, the fundamental idea is to vary two structural components, namely a transformation function and a possible key in order to compensate for weak obfuscation schemes. Assuming an attacker will spend some time in breaking the obfuscation scheme and recovering the related key, if any, by the time he is prepared to attack the protected SCADA devices, it may have changed both the transformation scheme and the key.

We extend the Modbus messaging service conceptual architecture (Schneider Automation, 2004) by introducing an additional module which implements the SCADA protocol obfuscation approach, namely a Modbus obfuscation module (Figure 3). The Modbus client interface provides an interface to the user application to demand the creation of Modbus requests. The Modbus client builds a Modbus request as specified by the user application and passes it to the Modbus obfuscation module. The Modbus obfuscation module is an implementation of an obfuscation/deobfuscation algorithm and is responsible for obfuscating outgoing Modbus messages and deobfuscating incoming Modbus messages. After receiving the Modbus request from the Modbus client the Modbus obfuscation module obfuscates it and returns it back to the Modbus client which passes it to lower layers for transmission. The Modbus back-end interface is an interface from the Modbus server to the user application.

The Modbus server waits for an incoming Modbus request on a certain TCP port. The default TCP port for Modbus is 502. Upon receiving an incoming Modbus request the Modbus server passes it to the Modbus obfuscation module. The later deobfuscates the Modbus request and returns it to the Modbus server which in turn submits it to a Modbus PDU checking function. The Modbus PDU checking function is responsible for verifying that a Modbus message is fully compliant with the Modbus protocol. If the deobfuscated message is a valid Modbus message the Modbus server processes it and then possibly builds a Modbus response. If the deobfuscated message is not a valid Modbus message it is dropped and not treated by the Modbus server at all. If the Modbus server builds a Modbus response, such a response is first passed to the Modbus obfuscation module which obfuscates it and returns it back to the Modbus server which

in turn passes it to lower layers for transmission. The obfuscation capability would be inserted into Modbus speaking devices with a set of functions that can be called by the Modbus client and Modbus server. These functions will accordingly obfuscate/deobfuscate Modbus messages by calling the Modbus obfuscation module.

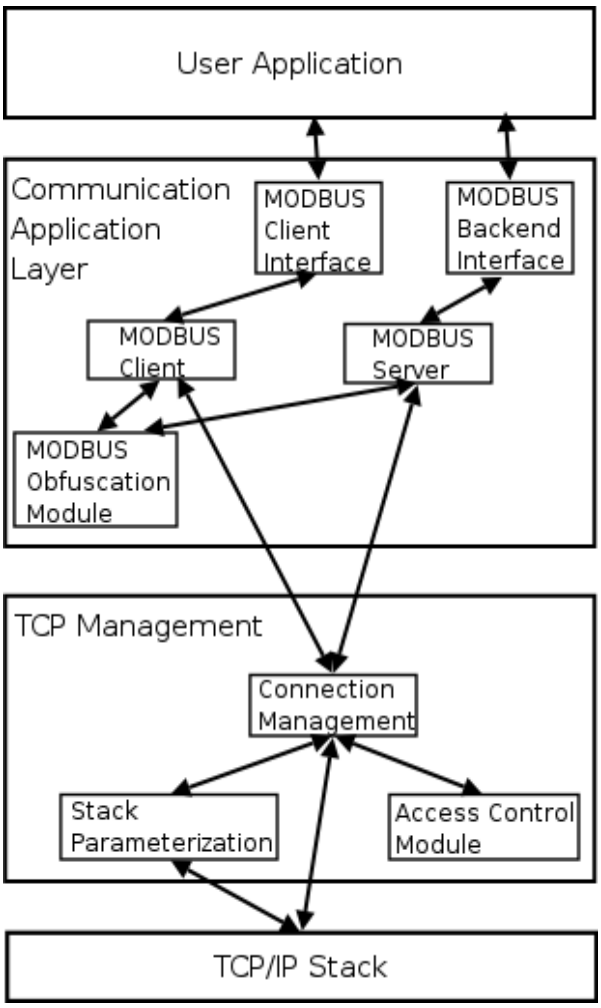

Figure 3 – The Modbus Messaging Service Conceptual Architecture extended to include SCADA Protocol Obfuscation capability

5.2 Motivations Behind SCADA Protocol Obfuscation

Our objective is to use the information held in the attack requirements tree of a SCADA protocol-based attack for the purpose of building a proactive intrusion prevention approach which could make such attack infeasible. For this purpose we need a mechanism for denying the AND node in the attack requirements tree of a SCADA protocol-based attack which represents the condition that attack messages should be built according to the SCADA protocol in use by target SCADA devices. A strong cryptographic algorithm would be ideal for such a task, definitely. For instance, legitimate SCADA devices could use asymmetric cryptography to exchange a cryptographic key which they can use along with a symmetric cryptographic algorithm to encrypt all their SCADA protocol messages. Although the key distribution in a SCADA network would have been an issue, strong encryption would have ensured that an attacker who does not know the encryption key would not be able to produce valid attack messages. Furthermore, taking into account the strength of a cryptographic algorithm, SCADA devices would have been reasonably protected from SCADA protocol-based attacks.

Nevertheless, the majority of SCADA field devices such as PLC's or RTU's have limited computational power, therefore such devices have noticeable difficulties in running strong cryptographic algorithms. Furthermore, in some cases the kind of processor and the amount of main memory in a SCADA field device does not allow for running a strong cryptographic algorithm at all. On the other hand SCADA communications should be real-time, therefore SCADA field devices are expected to carry out an action specified by a received request and send a response within a reasonable time fragment. We devised the SCADA protocol obfuscation approach to overcome such a problem, while accepting the fact that the cryptographic protection that may be provided by common processors in SCADA field devices is bounded, i.e. a cryptographic algorithm that can be run by SCADA field devices would have to be weak due to their limited computational power.

The SCADA protocol obfuscation approach provides a simple scrambling of SCADA protocol frames and it requires little computational power, therefore it is quite affordable by the kind of processors along with the amount of main memory in common SCADA field devices. This means that the "encryption" protection provided by SCADA protocol obfuscation will have to be weak, but we tried to deal with such a limitation to make it possible that such a weakness does not affect the robustness of the entire approach. The idea to cope with weak "encryption" protection consists in enabling SCADA devices to use a defined scrambling scheme within a limited time interval only, after which these devices change it. If an attacker will attempt to break a currently used scrambling scheme he will need some time to gather obfuscated protocol packets and analyze them. As the scrambling scheme is continuously changed, when an attacker has broken a certain scrambling scheme the actual scrambling scheme may be another scheme different than the broken scheme.

In the case of the SCADA protocol obfuscation approach, we may arguably talk about scrambling or obfuscation algorithm. Arguably, we do not encrypt, we just scramble to

keep aggressors off for a limited amount of time and we change scrambling scheme from time to time in order to limit the exposure window of a scrambling algorithm that has been broken. A SCADA protocol obfuscated through the approach described in this paper is intended to be unique for each group of legitimate SCADA devices, therefore an attacker who doesn't know the obfuscation scheme cannot produce valid messages. Our choice of introducing diversity exactly in SCADA protocols is motivated by the fact that such protocols are among the main instruments that attackers could use to disrupt the underlying critical infrastructure monitored and controlled by SCADA systems. Furthermore, each incoming message deobfuscated by legitimate SCADA devices equipped with obfuscation capability is checked for the purpose of verifying that it is fully in compliance with the SCADA protocol in use before being processed. These checks lead to denial of the AND node in the attack requirements tree of a SCADA protocol-based attack which represents the condition that targets should process attack messages.

6 SCADA Protocol Obfuscation Techniques

In this section, the obfuscation techniques we employ in building a transformation function within the SCADA protocol obfuscation approach are described. These techniques are described as applied to the Modbus TCP/IP protocol which has been a representative SCADA protocol for our general approach.

6.1 ADU Layout Randomization

ADU layout randomization was inspired by address space layout randomization (PaX Team; Bhatkar, DuVarney & Sekar, 2003; Xu, Kalbarczyk & Iyer, 2003) used by operating systems to counter attacks which exploit low-level coding vulnerabilities. Address space layout randomization consists in randomizing the base address of the memory area containing executable code, initialized data and uninitialized data, the base address of the memory area containing the heap, dynamic libraries, thread stacks and shared memory, the base address of the area containing the main stack, etc. An attacker would find it difficult to build such an exploit while he does not know where certain process components are located in main memory. If we build a parallelism between the address space of a given process and the Modbus ADU, and between process components and Modbus fields, we could follow the same approach by randomizing for example the order in which various Modbus fields are placed in an ADU (Figure 4). Generally the granularity at Modbus field level is not sufficient for building a robust transformation function. Thus, ADU layout randomization could be extended at a granularity of, for example, two bytes also for the data field while conserving the functional integrity of some defined bytes, such as the high and low parts of various addresses present in the data field.

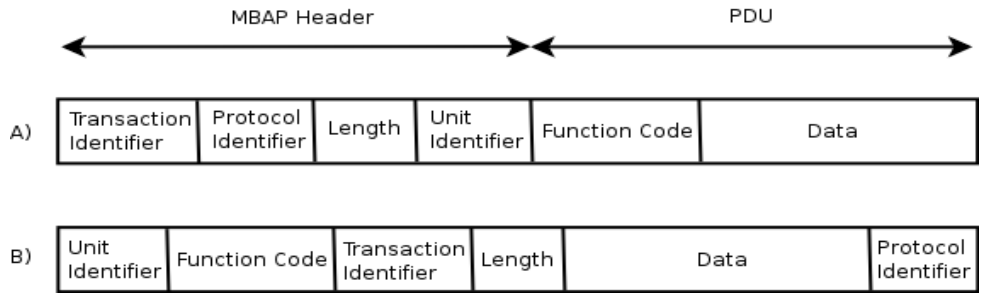

Figure 4 – Modbus TCP/IP request/response in A) an original layout and B) a randomized layout at the granularity of a Modbus field.

6.2 Modbus Field Set Randomization

Modbus field set randomization was inspired by instruction set randomization (Barrantes, Ackley, Forrest, Palmer, Stefanovic & Zovi, 2003; Kc, Keromytis & Prevelakis, 2003). Instruction set randomization is a technique devised to counter code injection attacks and aims at destroying the usability of binary code injected into the address space of a protected process. The main idea behind instruction set randomization consists in the creation of a randomized instruction set for each protected process. These process specific instruction sets are kept secret from attackers. If an attacker carries out a code injection attack against a protected process and doesn't know the instruction set used by that process, then he does not know how to produce machine code according to the instruction set of the protected process and will end up with injecting invalid binary code. We can build a parallelism between instructions in an instruction set used to talk to a machine and the Modbus fields used to talk to a Modbus device. Moving along this line we can randomize the Modbus fields for the purpose of achieving several randomized Modbus field sets.

Description	Function Code	Randomized Function Code
Read coils	0x1	0x17
Read discrete inputs	0x2	0x6
Read holding registers	0x3	0x0B
Read input registers	0x4	0x11
Write single coil	0x5	0x2B
Write single register	0x6	0x0F

Table 1 – An example of function code randomization

Similarly, if an attacker does not know the Modbus field set used by a group of devices in a SCADA network he cannot produce valid attack messages. The function code field for instance is suitable for being randomized (Table 1). The same holds for sub-function codes and error codes. Similarly addresses, quantities, byte counts, register and output values, etc., in the data field could be randomized as well. The transaction identifier could also help an attacker to identify the logic behind the transformation function as the value of this field in a request is simply copied into the corresponding response. Such a field is to be randomized into two values where one is to be used by the client and the other one is to be used by the server. The same holds for the unit identifier field. Furthermore, it is also necessary to masquerade some of the Modbus fields for the purpose of producing an acceptably resistant transformation function. The protocol identifier field for example in Modbus is always zero filled. If this field does not get masqueraded it will appear as 0x0000 in each Modbus message where evidently it is easily identifiable. By masquerading the protocol identifier field, we mean setting it to an arbitrary value, which during de-obfuscation at the other end is to reset to zero. The obfuscation algorithm however could use a single randomization map for all the fields to be randomized.

6.3 Other Obfuscation Techniques

In the case of Modbus requests where the data field is of zero length it may be much easier for an attacker to break the transformation function currently in use. In fact in this case there are only 8 bytes in the Modbus ADU, 7 bytes of MBAP header + 1 byte of function code. Thus, in such requests it may be constructive from the security point of view to pad the data field with randomly generated bits. Furthermore, before applying ADU layout randomization it may be helpful to allocate one or two bytes between Modbus fields to be filled with random values. For the purpose of deceiving an attacker and leading him to a wrong direction, it may be constructive from the security point of view to put on the communication link NULL Modbus requests and NULL Modbus responses, i.e. messages composed of randomly generated bits which cause no effect on the receiving SCADA device. These NULL messages may cause confusion in the attacker's analysis and consequently delay a possible break of the transformation function currently in use.

Another obfuscation technique which could be employed in the creation of a transformation function is stream cipher scrambling. It consists in introducing random bits into a Modbus frame and XOR'ing determined Modbus fields with determined random bits. Nevertheless, such a technique was considered too costly, therefore its employment is not always possible or advantageous.

7 Pro’s and Con’s of SCADA Protocol Obfuscation

The main advantage behind SCADA protocol obfuscation derives from the fact that it is a simple scrambling of SCADA protocol frames, consequently it requires little computational power. Furthermore, such an approach is generally composed of simple operations. The processors found in common SCADA field devices such as PLC's and RTU's along with the amount of main memory available in these devices can run a

reasonably structured instance of the SCADA protocol obfuscation approach. For instance, a transformation function constructed as a combination of ADU layout randomization and Modbus field set randomization imposes a performance cost around 0.8 % of the overall execution cost of a publicly available Modbus implementation in an embedded Linux operating system running on a processor emulator. Under the same testing platform a transformation function constructed as a combination of ADU layout randomization, Modbus field set randomization, data padding, and insertion of random bits between Modbus fields imposes a performance cost of around 1.3 % of the overall execution cost. Applying a granularity of two bytes also for the data field in the ADU layout randomization resulted in a performance cost between 1.4 % and 1.5 % of the overall execution cost.

Often an master station is configured to periodically send commands, which are specified in a configuration file, to slave stations. In these cases the key generator and the transformation scheme generator components in slave stations prepare their intervention to the transformation function during the period of time between the moment a request has been fully processed and a possible response has been built and sent, and the moment a successive request is received. Thus, generally only the cost of the transformation function weighs on the real time nature of the communication between SCADA devices. Furthermore, scrambling has some advantages with regard to intrusion detection as it reduces the time needed to "decrypt" an "encrypted" SCADA protocol packet for the purpose of analyzing it.

SCADA protocol obfuscation tries to make proper use of the low level of cryptographic protection allowed by limited computational power in SCADA field devices. In fact such an approach uses a certain transformation scheme only for a limited period of time, after which it changes transformation scheme. Therefore a weak transformation scheme does not necessarily imply weak protection against SCADA protocol-based attacks.

The main disadvantage of SCADA protocol obfuscation is that it considerably increases the network overhead in a moment when many SCADA networks have limited transmission capabilities. Obfuscation techniques such as data padding or insertion of random bits between Modbus fields extend possibly small Modbus messages into a full 253 bytes Modbus PDU. Furthermore, each NULL Modbus request or NULL Modbus response alone puts on the SCADA link exactly 260 bytes of a complete Modbus TCP ADU plus the number of bytes introduced by lower layers in the OSI stack. As a consequence SCADA protocol obfuscation is more suitable in TCP/IP networks where network bandwidth is not a critical issue rather than in serial line networks.

The application of a SCADA protocol obfuscation approach in the defense of SCADA devices from SCADA protocol-based attacks may be further constrained by the fact that many companies pay for each single byte of communication between their SCADA devices. Such is the case for example when their master stations communicate with field devices over satellite which may be operated by external parties.

SCADA protocol obfuscation does not add any value more than strong encryption does with respect to the high dependency on physical security in SCADA field devices. As a matter of fact, a physical compromise of a SCADA field device would turn into true the value of the node in the attack requirements tree which SCADA protocol obfuscation

aims at denying. An attacker may break the physical security of a slave station and carry out the attacks from there as the compromised PLC or RTU will obfuscate correctly the attack packets sent by the attacker.

Furthermore, SCADA protocol obfuscation has its own critical point, i.e. the transformation scheme generator. During our experiments we defined statically and randomly the periodic modifications to the transformation scheme. It could be possible to derive the next transformation scheme to be used by protected SCADA devices based on information gathered from previous transformation schemes. In this case the entire SCADA protocol obfuscation is considered broken.

8 Conclusion

In this paper we described a defensive approach, namely SCADA protocol obfuscation, to counter SCADA protocol-based attacks launched from a rogue device introduced in a SCADA network. Such an approach was structured as a proactive structural intervention that we built upon information held by a new conceptual structure we described in this paper as well, and which we refer to as attack requirements tree. This tree is a structured means of providing conditions which must hold for a given attack to be feasible, and each one of its nodes represents one of those conditions. We built an attack requirements tree for a SCADA protocol-based attack and tried to make this attack unfeasible by using SCADA protocol obfuscation to deny the node which represents the requirement that attack messages should be built according to the SCADA protocol in use. While a strong encryption algorithm could protect from SCADA protocol-based attacks much better, common SCADA field devices such as PLC's or RTU's have limited computational power and consequently are subject to considerable difficulties in running strong cryptographic algorithms.

The SCADA protocol obfuscation approach provides a simple scrambling of SCADA protocol frames. As a consequence such an approach requires little computational power and is quite affordable by the kind of processors and the amount of main memory found in common SCADA field devices. Due to the simplicity in both design and implementation the transformation scheme in our approach is quite weak from a security standpoint. In order to compensate for weak transformation schemes, SCADA protocol obfuscation provides the mechanism for changing a transformation scheme after a limited period of time. We build simple transformation schemes through techniques such as ADU layout randomization and field set randomization following a parallelism with address space layout randomization and instruction set randomization, respectively. These obfuscation techniques are actually complemented by data padding, field masquerading, insertion of random bits between Modbus fields, and when possible stream cipher scrambling for the purpose of strengthening the transformation scheme while remaining within the maximum possible level allowed by the limited computational power of SCADA field devices.

About the Authors – Carlo Bellettini works as an Associate Professor at Università degli Studi di Milano, Italy (Department of Computer Science and Communication) since 2002. He holds a Ph.D. in Computer Science from Università degli Studi di Milano, Italy (1998) and a laurea degree in Electronic Engineering from Politecnico di Milano, Italy (1992). His research interests include real-time systems, with particular reference to analysis of specification in high level Petri nets, object-oriented design approaches and object-oriented component reusability, network security and performance evaluation of computer systems and networks.

Julian L. Rrushi received a BS in Computer Science in 2003 and a MS in Information and Communication Technology in 2005 from the University of Milan. He is actually a second year PhD student of the University of Milan where he is doing research on SCADA security. While a Master student he was awarded a research scholarship by (ISC)² for a project on mobile code and security implications on servers, and an internship by the Joint Research Center of the European Commission for preparing his Master thesis. His research interests are system and network security.

References

AGA 12 Task Group. (2006). *Cryptographic Protection of SCADA Communications, Part 1: Background, Policies and Test Plan*. AGA Report Number 12.

Balducelli, C. (2003). *Modeling Attack Scenarios Against Software Intensive Critical Infrastructures*. In Proceedings of the 10th Annual Conference of the International Emergency Management Society, Sophia-Antipolis, Provence, France.

Balducelli, C., Vicoli, G., & Jin, Xuan. (2006). *Formalizing and Testing Attack Scenarios for Information Intensive Critical Infrastructures*. Methodologies for Emerging Technologies in Automation. University of Rome "La Sapienza", Rome, Italy.

Bhatkar, S., DuVarney, D., & Sekar, R. (2003). *Address obfuscation: An efficient approach to combat a broad range of memory error exploits*. In Proceedings of the 12th USENIX Security Symposium, (pp. 105 - 120). Washington D.C., U.S.A.

Barrantes, E.G., Ackley, D.H., Forrest, S., Palmer, T.S., Stefanovic, D., & Zovi D. (2003). *Randomized instruction set emulation to disrupt binary code injection attacks*, In Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS2003). (pp. 281 - 289). Washington, U.S.A.

Bellettini, C., & Rrushi, J.L. (2006). *FireBuff: A Defensive Approach Against Control-data and Pure-data Attacks*. Retrieved December 8, 2006, from http://homes.dsi.unimi.it/~rrushi/firebuff

Byres, E. J., Franz, M., & Miller, D. (2004). *The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems*, International Infrastructure Survivability Workshop, IEEE, Lisbon, Portugal.

Byres, E. J., & Lowe, J. (2004). *The Myths and Facts behind Cyber Security Risks for Industrial Control Systems*, VDE 2004 Congress, VDE, Berlin, Germany.

Convery, S., Cook, D. , & Franz, M. *An Attack Tree for the Border Gateway Protocol*, Retrieved December 8, 2006, from http://www.io.com/~mdfranz/papers/draft-convery-bgpattack-01.txt

Del Re, E., Fantacci, R., & Maffucci, D. (1989). *A New Speech Signal Scrambling Method for Secure Communications: Theory, Implementation, and Security Evaluation*. IEE Journal on Selected Areas in Communications, volume 7, number 4.

Finco, G., Lee, K., Miller, G., Tebbe, J., & Wells, R. (2006). *Cyber Security Procurement Language for Control Systems,* Version 1.5 Draft. Prepared by Idaho National Laboratory for the U.S. Department of Homeland Security, National Cyber Security Division. Retrieved December 1, 2006, from http://www.msisac.org/scada/documents/16nov06-scada-procurement_.pdf

Forrest, S., Somayaji, A., & Ackley, D. (1997). *Building diverse computer systems.* In Proceedings of the 6th Workshop on Hot Topics in Operating Systems. (pp. 67 - 72). Cape Cod, Massachusetts, U.S.A.

Kc, G.S., Keromytis, A.D., & Prevelakis V. (2003). *Countering code injection attacks with instruction set randomization*, In Proceedings of the 10th ACM Conference on Computer and Communications Security. (pp. 272 - 280). Washington, U.S.A.

Li, T., Ren, J., Ling, Q., & Liang, W. (2004). *Physical Layer Built-in Security Analysis and Enhancement of CDMA Systems*. Proceedings of 2004 Conference on Information Sciences and Systems, University of Princeton, Princeton, NJ.

MODBUS Organization. (2004). *MODBUS Application Protocol Specification V 1.1a*. Retrieved December 1, 2006, from http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1a.pdf

Schneider Automation. (2004). *MODBUS Messaging on TCP/IP Implementation Guide V1.0a*, Retrieved December 1, 2006, from http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0a.pdf

PaX Team, Documentation for the PaX project. Retrieved December 1, 2006, from http://pax.grsecurity.net/docs

Scheyner, O. (2004). *Scenario Graphs and Attack Graphs*. PhD thesis, Carnegie Mellon University. Retrieved December 8, 2006, from http://www.milena.org/thesis/sg-ag.pdf

Schneier, B. (1999). *Attack Trees*. Dr Dobbs Journal. Retrieved December 8, 2006, from http://www.schneier.com/paper-attacktrees-ddj-ft.html

Stamp, J., Dillinger, J., Young W., & DePoy, J. (2003). *Common Vulnerabilities In Critical Infrastructure Control Systems*, Sandia National Laboratories, Albuquerque, NM.

U.S. Nuclear Regulatory Commission. (1981). *Fault Tree Handbook*. Retrieved December 8, 2006, from http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/sr0492.pdf

Xu, H., Chapin, S.J. (2006). *Improving Address Space Randomization with a Dynamic Offset Randomization Technique*. (pp. 384 - 391). ACM Symposium on Applied Computing, Dijon, France.