Chapter 15

SECURITY ANALYSIS OF MULTILAYER SCADA PROTOCOLS: A MODBUS TCP CASE STUDY

Janica Edmonds, Mauricio Papa and Sujeet Shenoi

Abstract The layering of protocols in critical infrastructure networks – exemplified by Modbus TCP in the oil and gas sector and SS7oIP in the telecommunications sector – raises important security issues. The individual protocol stacks, e.g., Modbus and SS7, have certain vulnerabilities, and transporting these protocols using carrier protocols, e.g., TCP/IP, brings into play the vulnerabilities of the carrier protocols. Moreover, the layering produces unintended inter-protocol interactions and, possibly, new vulnerabilities. This paper describes a formal methodology for evaluating the security of multilayer SCADA protocols. The methodology, involving the analysis of peer-to-peer communications and multilayer protocol interactions, is discussed in the context of Modbus TCP, the predominant protocol used for oil and gas pipeline operations.

Keywords: Multilayer protocols, Modbus TCP, security analysis, formal methods

1. Introduction

Critical infrastructure systems, e.g., SCADA and public telephone networks, have traditionally employed specialized equipment and protocol stacks. Moreover, they were usually isolated from TCP/IP networks.

In recent years, however, the proliferation of TCP/IP networks, along with the advanced services they provide and the availability of inexpensive COTS equipment, have caused several critical infrastructure protocol stacks to be re-designed to use TCP/IP as a foundation for transport and network interconnectivity. For example, in the oil and gas sector, the original Modbus Serial protocol [14] is transported by TCP in the Modbus TCP variant [13], which provides increased network connectivity and multiple, concurrent transactions.

Similarly, in the telecommunications sector, the SS7 over IP (SS7oIP) protocol [6, 15] "floats" the top three layers of the SS7 protocol stack on IP using

another protocol (RUDP) as "glue." SS7oIP significantly reduces the operating costs of telecommunications carriers by migrating SS7 traffic from dedicated long-haul signaling links to inexpensive WAN backbones. Furthermore, combining cost-effective IP transport equipment and service-rich SS7 applications enables carriers to provide enhanced services (e.g., Short Message Service and Unified Messaging) and pursue new business opportunities.

The layering of protocols raises important security issues. The individual protocols (e.g., Modbus) or protocol stacks (e.g., SS7) have vulnerabilities. Transporting them over TCP/IP and RUDP brings into play the vulnerabilities of the carrier protocols. Meanwhile, layering multiple protocols produces unintended interactions between protocols and, possibly, new vulnerabilities.

This paper describes a methodology for evaluating the security properties of multilayer protocols used in critical infrastructure networks. The security analysis methodology involves the modeling of protocol communications, and the analysis and verification of protocols using formal methods. The methodology analyzes multilayer protocols from the point of view of single-layer protocol (peer-to-peer) communications and multilayer protocol interactions. A case study is discussed in the context of Modbus TCP, the predominant protocol used in the oil and gas sector. In the case study, a vulnerability is identified in the peer-to-peer Modbus protocol, a correction is proposed, and the security of the corrected protocol is verified. Next, a vulnerability involving inter-protocol interactions in the corrected peer-to-peer protocol is identified, and a correction is proposed, which is subsequently verified.

Several techniques have been proposed for modeling, analyzing and verifying the security properties of peer-to-peer protocols [1–4, 8, 11, 16–18]. This paper engages standard cryptographic protocol strategies for modeling and analyzing protocol communications and the knowledge held by communicating entities (principals and intruders). A verification tool, AVISPA [3, 18], is then used to identify violations of desired security properties and, subsequently, to verify the proposed corrections. The principal contribution of this work is the extension of peer-to-peer techniques to address protocol stack interactions, which is vital to securing critical infrastructure networks.

2. Modbus Protocol

Modbus is one of the oldest, but most widely used industrial control protocols [12–14]. Modbus' open specifications and TCP extension have contributed to its popularity, especially in the oil and gas sector, where it is the *de facto* standard for process control networks.

The Modbus protocol establishes the rules and message structure used by programmable logic controllers (PLCs) to communicate amongst themselves and with human machine interfaces (HMIs). The Modbus application layer defines the mechanisms by which devices exchange supervisory, control and data acquisition information for operating and controlling industrial processes.

Modbus engages a simple request/reply communication mechanism between a master unit and slave devices (Figure 1). For example, a control center

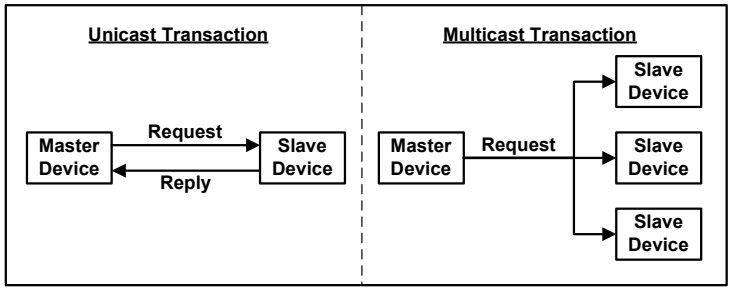

Figure 1. Modbus master-slave communications.

(master) might send a "read" message to a sensor (slave) to obtain the value of a process parameter (e.g., temperature). Alternatively, it might send a "write" message to an actuator (slave) to perform a control action (e.g., open a valve).

A unicast transaction involving the master and an addressed slave comprises two messages, a request message (e.g., for a temperature reading or to open a valve) and the corresponding response message (e.g., the temperature reading or an acknowledgment that the valve was opened). On the other hand, a broadcast transaction involves one request message from the master that is received by all the slaves; the slaves do not send response messages. An example broadcast transaction is a "write" message that resets all sensors and actuators.

Modbus communications occur over serial lines or, more recently, using TCP/IP as a transport mechanism. The following sections describe the serial and TCP variants of the Modbus protocol. For additional details, readers are referred to the protocol specification [12] and the guides for serial [14] and TCP/IP network [13] implementations.

2.1 Modbus Serial Protocol

In the Modbus Serial protocol, messages are transmitted over a serial line using the ASCII or RTU transmission modes. A Modbus Serial message has three components: (i) slave address, (ii) Modbus Application Protocol Data Unit (PDU), and (iii) an error checking field (Figure 2). The slave address in a request message identifies the intended recipient; the corresponding address in a response message is used by the master to identify the responding slave. A broadcast message has an address of 0. A unicast message has an address in the [1,247] range that identifies an individual slave in the network. Values in the [248,255] range are reserved addresses.

The Modbus PDU has two fields, a one-byte function code and function parameters (max 252 bytes). The function code specifies the operation requested by the master; it also conveys error information when an exception occurs in a slave device. The function parameters field contains data pertaining to function invocation (request messages) or function results (reply messages).

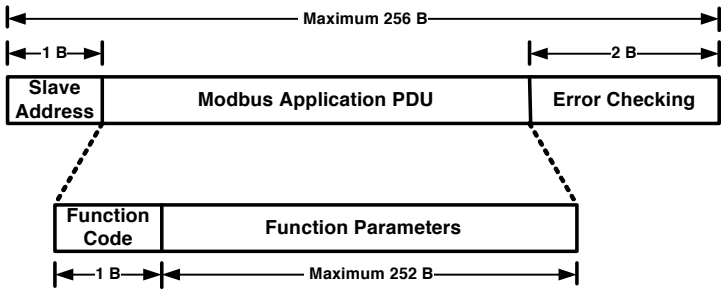

Figure 2. Modbus Serial message.

Modbus function codes specify a variety of read and write operations on slaves, as well as diagnostic functions and error conditions. Public codes correspond to functions whose semantics are completely defined or will be defined in the Modbus standard. Valid public codes fall in the non-contiguous ranges: [1,64], [73,99] and [111,127]. User-defined codes in the [65,72] and [100,110] ranges are not considered by the Modbus standard; their implementations are left to vendors and there are no guarantees regarding their compatibility. Reserved function codes are public codes that are reserved to ensure compatibility with legacy systems. Function code values in the unused range [128,255] are for indicating error conditions in response messages.

Response messages have the same structure as request messages. The Modbus specification defines positive and negative responses to request messages. A positive response informs the master that the slave has successfully performed the requested action; this is indicated by including the request message function code in the response. On the other hand, a negative or exception response notifies the master that the transaction could not be performed by the addressed slave. The function code for a negative response is computed by adding 128 to the function code of the request message; thus, function codes in the [128,255] range denote error conditions. A negative response also includes an exception code in the function parameters part of the response message that provides information about the cause of the error. The Modbus specification defines nine exception responses whose format and content depend on the issuing entity and the type of event producing the exception.

2.2 Modbus TCP Protocol

The Modbus TCP protocol provides connectivity within a Modbus network (a master and its slaves) as well as for TCP/IP interconnected Modbus networks (multiple masters, each with multiple slaves). Modbus TCP enables a master to have multiple outstanding transactions. Moreover, it permits a slave to engage in concurrent communications with multiple masters.

In Modbus TCP, slaves listen for incoming TCP connections on port 502 (IANA assigned port) and may optionally listen on additional ports. The device

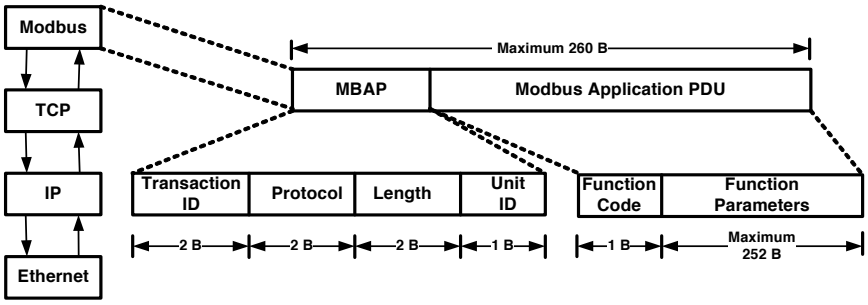

Figure 3. Modbus TCP message.

that performs the passive open operation on TCP (server) assumes the role of a Modbus slave. Similarly, the device performing the active open operation on TCP (client) assumes the role of a Modbus master. Once a TCP communication channel has been established, Modbus roles cannot be changed on that channel; however, multiple outstanding transactions can exist on the channel. Another communication channel must be opened if a device is to assume a different role.

Modbus TCP transactions are functionally equivalent to their serial counterparts: the master and slaves exchange PDUs, except that the transactions are encapsulated in TCP messages (Figure 3). Consequently, a Modbus TCP PDU includes the Modbus Application Protocol (MBAP) in addition to the Modbus Application PDU used in Modbus Serial.

The MBAP header has four fields: (i) transaction identifier, (ii) protocol identifier, (iii) length and (iv) unit identifier. The transaction identifier allows devices to pair transaction requests and replies. The protocol identifier indicates the application protocol encapsulated by the MBAP header (zero for Modbus). The length field indicates the length in bytes of the remaining fields (unit identifier and PDU). The unit identifier indicates the slave associated with the transaction (used only in the case of legacy implementations).

The Modbus TCP specification requires that only one application PDU be transported in the payload of a TCP packet. Since application PDUs have a maximum size of 253 bytes and the length of the MBAP is fixed at seven bytes, the maximum length of a Modbus TCP data unit is 260 bytes.

3. Security Analysis Methodology

This section describes the formalism used to model protocols. In addition, it discusses the technique used to identify violations of security properties and to verify corrections made to protocols.

3.1 Protocol Modeling

The first step in security analysis is to model the protocol. It is necessary to capture the roles of the participating agents, the initial knowledge associated

with each role, and the communications sequence. The roles indicate the parts that agents in the protocol can play. The initial knowledge associated with each role is the data that an agent must know in order to participate in the protocol. The communications sequence defines the messages (and their contents) that are sent and received by agents.

We illustrate the modeling process using the simple protocol:

$$\begin{array}{lll} A & \longrightarrow & B: \ \{ID_1, Amount_1\} \\ B & \longrightarrow & A: \ \{ID_1, Amount_1, Hash(ID_1, Amount_1)\}_{K_{ab}} \end{array}$$

The protocol has two agents (A and B) and a two-step communication sequence. Agents A and B assume the roles of initiator and responder, respectively. A knows the values ID_1 and $Amount_1$, which it sends to B. B receives the values from A, computes a hash of the two values, and sends all the information to A encrypted using a symmetric key K_{ab} known only to A and B.

In the following, we describe the modeling of agent communications and intruder attacks in more detail.

3.1.1 Agent Communications.

Agents communicate when a message sent by one agent matches the pattern exposed by another agent. We specify the constructs involved in modeling agent communications, including complex messages, encryption/decryption and pattern matching [16, 17].

Definition 1: A *key* ($key \in KEY$) is a public/private key (K_n/K_n^{-1}), a shared or secret key (K_n^s), or *nil*, for an unencrypted message:

$$key \quad ::= \quad K_n \mid K_n^{-1} \mid K_n^s \mid nil$$

We assume the existence of a basic type $NAME$ comprising an infinite set of *names*. This basic type is used to create unique keys, and data for messages and patterns. For example, n and m in Definition 1 are of type $NAME$ ($n, m \in NAME$).

A key matching operator is required to determine whether or not a key can be used to decrypt a message. The first two rules in the definition below are for asymmetric and symmetric encryption, respectively. The third rule is used for cleartext messages.

Definition 2: The key matching operator ($\overset{K}{\sim}$) is defined by:

$$\begin{array}{lll} K_a & \overset{K}{\sim} & K_b^{-1} \ \text{ iff } \ a = b \\ K_a^s & \overset{K}{\sim} & K_b^s \ \text{ iff } \ a = b \\ nil & \overset{K}{\sim} & nil \end{array}$$

Messages exchanged by agents are nested tuples of values encrypted under a key, where the values can be keys, messages, data, nonces or timestamps.

Definition 3: A *message* is a list of values $\{v_1, v_2, ., v_j\}$ encrypted under key, i.e., $\{v_1, v_2, \cdots, v_j\}_{key}$. A value is a key, a message, a name or a fresh name:

$$message = \{v_1, v_2, \cdots, v_j\}_{key}$$

$$value = key|message|n|\#n$$

Note that the names in a message represent message data. Fresh names refer to nonces and timestamps that are generated in a single run of a protocol.

Communication between agents occurs when the sending agent's message matches the receiving agent's pattern. The pattern exposed by the receiving agent reveals the knowledge that it has about the incoming message.

Definition 4: A *pattern* is a list of patterns $\{p_1, p_2, \cdots, p_j\}$ encrypted under key and denoted by $\{p_1, p_2, \cdots, p_j\}_{key}$ or a key, a wildcard or placeholder ($n?$) for capturing values or a name (n):

$$pattern = \{p_1, p_2, \cdots, p_j\}_{key}|key|n?|n$$

Finally, we specify the rules for matching messages and patterns.

Definition 5: The message-pattern matching operator $\sim$ is defined by the following rules ($v, v_i \in VAL$, $p, p_i \in PAT$, $key, key_i \in KEY$, $m \in MSG$ and $n \in NAME$):

$$\begin{aligned}
\{v_1, v_2, ..., v_j\}_{key_1} &\sim \{p_1, p_2, ..., p_j\}_{key_2} \text{ iff} \\
&\quad key_1 \overset{K}{\sim} key_2 \wedge v_i \sim p_i \ \forall i = 1..j \\
m &\sim n? \\
m &\sim key? \\
v &\sim n? \\
v &\sim n \text{ iff } v = n \\
key &\sim key?
\end{aligned}$$

Figure 4 presents a formal model of the two-step communication sequence:

$$\begin{aligned}
A &\longrightarrow B: \ \{ID_1, Amount_1\} \\
B &\longrightarrow A: \ \{ID_1, Amount_1, Hash(ID_1, Amount_1)\}_{K_{ab}}
\end{aligned}$$

The figure formally specifies the agents' roles and initial knowledge. Also, it illustrates the modeling of messages and patterns, and the exchange of information between agents.

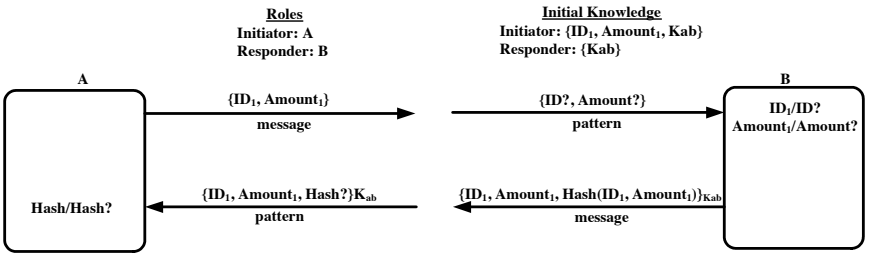

Figure 4. Modeling agent communications.

In the first step, A sends the message $\{ID_1, Amount_1\}$, which is matched by the pattern $\{ID?, Amount?\}$ exposed by B. The match causes the values ID_1 and $Amount_1$ to be bound to B's wildcard variables $ID?$ and $Amount?$, respectively. The binding is denoted by $ID_1/ID?$ and $Amount_1/Amount?$ in B. Thus, B obtains (knows) the values ID_1 and $Amount_1$.

After receiving ID_1 and $Amount_1$, B computes the hash value of ID_1 and $Amount_1$. In the second step, B sends ID_1, $Amount_1$ and the hash value to A in an encrypted message, i.e., $\{ID_1, Amount_1, Hash(ID_1, Amount_1)\}_{K_{ab}}$. Note that the message is encrypted under K_{ab}, a secret key shared by A and B. As shown in Figure 4, the encrypted message is matched by the pattern $\{ID_1, Amount_1, Hash?\}_{Kab}$ exposed by A. Thus, A's wildcard variable $Hash?$ receives the hash value computed by B.

3.1.2 Intruder Attacks. An intruder I may perpetrate an attack by exposing an appropriate pattern to capture information sent by agent A. Next, it creates a fabricated message, which it sends to B. The corresponding communication sequence involving agents A, I and B is:

$$\begin{aligned} A &\longrightarrow I: \ \{ID_1, Amount_1\} \\ I &\longrightarrow B: \ \{ID_1, Amount_2\} \\ B &\longrightarrow I: \ \{ID_1, Amount_2, Hash(ID_1, Amount_2)\}_{K_{ab}} \\ I &\longrightarrow A: \ \{ID_1, Amount_2, Hash(ID_1, Amount_2)\}_{K_{ab}} \end{aligned}$$

In the first step, I might expose the pattern $\{ID?, Amount?\}$ to capture both ID_1 and $Amount_1$ in the message sent by A. Alternatively, if I only intends to capture a value for $Amount?$, it might expose the pattern $\{ID_1, Amount?\}$ – this means that I has initial knowledge of ID_1. Of course, this pattern would not work if A were to send the message $\{ID_2, Amount_1\}$ because ID_2 in the message would not match ID_1 in the pattern $\{ID_1, Amount?\}$ exposed by intruder I (according to Definition 5).

In the second step, I sends the message $\{ID_1, Amount_2\}$, which matches B's exposed pattern $\{ID?, Amount?\}$, enabling B to obtain the values ID_1

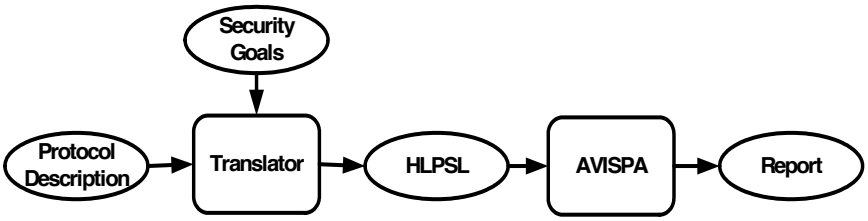

Figure 5. Analysis and verification framework.

and $Amount_2$. B computes the hash value, and sends an encrypted message, which is intercepted by I (Step 3). I obtains the encrypted message by exposing a pattern $\{Message?\}$. However, I is unable to unlock the message because it does not know key K_{ab}, so it can only forward the encrypted message to A (Step 4). Of course, when A unlocks the message and checks the hash, it notices that something is amiss. Thus, I's attack only has nuisance value.

Note that I is only able to change values that are eventually bound to wildcards (variables) in a pattern exposed by a receiver (e.g., $Amount?$, which receives the value $Amount_2$ instead of $Amount_1$ in the attack). The other values in the exposed pattern are determined by the receiver and cannot be changed by I.

3.2 Protocol Analysis and Verification

This section discusses the protocol analysis and verification methodology [7]. In particular, it describes the analysis and verification framework, the specification of security goals, and the application of AVISPA [3, 18], an automated tool for validating Internet security protocols.

3.2.1 Framework. The framework used for protocol analysis and verification is presented in Figure 5. A formal protocol specification (e.g., the example in Section 3.1) and security goals (described in Section 3.2.2 below) are translated into the High Level Protocol Specification Language (HLPSL) for use by the AVISPA tool [3, 18]. AVISPA produces a report that documents whether or not the security goals are satisfied. The report also describes the attacks (if any exist) that may be perpetrated by an intruder.

HLPSL is an expressive, modular role-based formal language that can be used to specify control flow patterns, data structures, alternative intruder models [5] and complex security properties, along with various cryptographic primitives and their algebraic properties. HLPSL models agent communications, including both messages and patterns, and security goals for protocols. Currently, the translation to HLPSL is performed manually. However, it is relatively simple to construct a compiler that automates the translation process.

The HLPSL file with protocol specifications and security goals is submitted to AVISPA for analysis and verification. AVISPA, which is described in more

detail in Section 3.2.3, has been tested on numerous protocols [3, 18]. We believe this is the first time AVISPA has been applied to analyze a SCADA protocol, including adapting it to examine multilayer protocol interactions.

AVISPA produces a formal report, which labels a protocol as `SAFE`, `UNSAFE`, `INCONCLUSIVE` or `ERROR`. `SAFE` indicates that no vulnerabilities were identified based on the specified security goals. `UNSAFE` means that AVISPA was able to find one or more attacks on the protocol; each attack is specified in the report as a message sequence chart. `INCONCLUSIVE` means that AVISPA was unable to reach any conclusions in a bounded number of iterations; in other words, the protocol is safe or additional iterations are required to identify an attack. `ERROR` implies that AVISPA was unable to correctly interpret the HLPSL file.

3.2.2 Security Goals. The security goals are the properties that must be satisfied by a protocol being analyzed by AVISPA, e.g., authentication and secrecy, key agreement properties, anonymity and non-repudiation [3]. Specific security goals that can be evaluated by AVISPA include:

- **Peer Entity Authentication:** Assuring one agent of the identity of a second agent through the presentation of evidence and/or credentials.
- **Data Origin Authentication:** Ensuring confidence that a received message or piece of data was created by a certain agent at some time in the past, and has not been altered or corrupted. This property is also called message authentication.
- **Implicit Destination Authentication:** Ensuring that a message is only readable by agents authorized by the sender.
- **Replay Protection:** Assuring that a previously authenticated message is not reused.
- **Key Authentication:** Ensuring that a particular secret key is limited to specific known and trusted parties.
- **Key Confirmation:** An agent has proof that a second agent has a particular secret key.
- **Fresh Key Derivation:** Dynamic key management is used to derive fresh session keys.
- **Identity Protection Against Eavesdroppers:** An intruder should not be able to establish the real identity of an agent based on communications exchanged by the agent.
- **Proof of Origin:** Undeniable evidence that an agent sent a message.
- **Proof of Delivery:** Undeniable evidence that an agent received a message.

AVISPA can reason about other security goals, but these may involve additional implementation efforts. Security goals are specified by augmenting the transitions of roles (messages and patterns) with *goal-facts*. HLPSL has built-in support for three primitive goals: secrecy, weak authentication and strong authentication. The expression $secrecy\_of\ X$ indicates that if an intruder obtains X, a violation of the specified security requirement has occurred. It is important to note that AVISPA assumes that all protocol roles with access to variable X will keep their respective copies a secret. The weak and strong authentication goals are modeled after Lowe's [10] notions of non-injective and injective agreement, respectively.

A weak authentication of Y by X on variable Z, expressed as $X\ weakly\ authenticates\ Y\ on\ Z$, is achieved if at the end of a protocol run initiated by X, both X and Y agree on the value Z. This definition does not guarantee a one-to-one relationship between the number of times X has participated in a valid protocol run and the number of times Y has participated, i.e., it is possible for X to believe that it has participated in two valid runs when Y has been taking part on a single run. A strong authentication goal, simply expressed as $X\ authenticates\ Y\ on\ Z$, requires that each protocol run initiated by X corresponds to a unique run of the protocol by Y. The HLPSL goal section is used to describe the combinations of facts that indicate an attack.

The internal representation of attack conditions is in terms of temporal logic [9]; additional properties must be expressed using temporal logic. Of the three main security goals for SCADA protocols, integrity is most easily verified using AVISPA.

In the example in Section 3.1, agent A might wish to ensure the integrity of the message from B. The following three statements would be introduced in the corresponding HLPSL file to check the integrity of the communications. The statement $witness(B, A, hash\_function, hash(ID_1,\ Amount_1))$ in B's role definition states that B has produced a certain hash value for A. The statement $request(A, B, hash\_function, hash?)$ in A's role definition states that A wants the hash value to be verified. The HLPSL goal statement $A\ authenticates\ B\ on\ hash\_function$ indicates that A and B must agree on the value of $hash\_function$.

3.2.3 AVISPA. AVISPA [3] was created to analyze Internet protocols and applications. It incorporates four separate backends: OFMC, CL-AtSe, SATMC and TA4SP. AVISPA has been tested on numerous industrial protocols, and has uncovered many known and unknown flaws [18].

To understand how AVISPA works, note that the values that an intruder I can assign to a variable (wildcard) depend on the values in I's knowledge base and the current state of the protocol run. These values must match the receiver's exposed pattern. If the values do not match, the message will be ignored or rejected, and the protocol run will not be completed. Knowledge about the specific values to be given to variables may come from information gathered about messages that are sent/received later in the protocol sequence.

Indeed, I requires knowledge about the protocol communication sequence to perpetrate an attack. AVISPA automates the process of searching for values assigned to variables that satisfy the protocol constraints. If an assignment of values results in a security requirement not being satisfied, AVISPA reports this fact and presents a sequence of communication exchanges as proof.

AVISPA provides a web interface that enables users to edit protocol specifications, and select and configure AVISPA backend tools. One or more backends may be used to evaluate a protocol. Input to all AVISPA backends must be in HLPSL.

AVISPA outputs a vulnerabilities report that lists the attacks that could be perpetrated by an intruder. An attack is detected when a communication sequence is found such that there is a successful protocol run and one or more security requirements are violated. Otherwise, it is assumed that an intruder is unable to generate an attack in the given scenario.

4. Modbus TCP Case Study

This section presents a Modbus TCP case study, which focuses on the authentication of Modbus agents and the origins of messages. The study shows that identifying and correcting flaws at the peer-to-peer (Modbus Serial) level are insufficient. It is imperative that security analyses of multilayer protocols, such as Modbus TCP, also be performed at the inter-protocol level.

4.1 Modbus Serial Analysis

Unicast transactions in the Modbus Serial protocol involve the exchange of request and reply messages. Figure 6(a) models a unicast transaction using the formalism discussed in Section 3. The $Master$ unit sends the message $\{Slave, FC_1, Data_1, CRC_1\}$ to $Slave$ requesting it to perform function FC_1 with parameters $Data_1$; CRC_1 is included to verify the integrity of the message. The $Slave$ listens for messages by exposing the pattern $\{Slave, FC_1?, Data_1?, CRC_1?\}$. The first field (with value $Slave$) indicates that $Slave$ only receives messages addressed to $Slave$ (see Definition 5 in Section 3). The remaining three fields $FC_1?$, $Data_1?$ and $CRC_1?$ are wildcards (variables) that are bound to the data values FC_1, $Data_1$ and CRC_1, respectively.

Upon executing the request, $Slave$ responds with the message $\{Slave, FC_2, Data_2, CRC_2\}$ indicating that it ($Slave$) is responding with status FC_2 ($FC_2 = FC_1$ or $FC_2 = FC_1 + 128$ for positive/negative replies [12]) as described by $Data_2$; CRC_2 is used to establish message integrity. $Master$ exposes the pattern $\{Slave, FC_2?, Data_2?, CRC_2?\}$ to receive this data from $Slave$.

A flaw was discovered after formally modeling $Master$-$Slave$ communications and using the protocol analysis technique described in Section 3. The flaw enables an $Intruder$ to intercept and modify messages sent by the $Master$ and $Slave$ (Figure 6(b)). In this attack, the $Intruder$ intercepts $Master$'s request message $\{Slave, FC_1, Data_1, CRC_1\}$ and sends the fabricated message $\{Slave, FC_2, Data_2, CRC_2\}$ to $Slave$. Upon receiving this message, $Slave$ per-

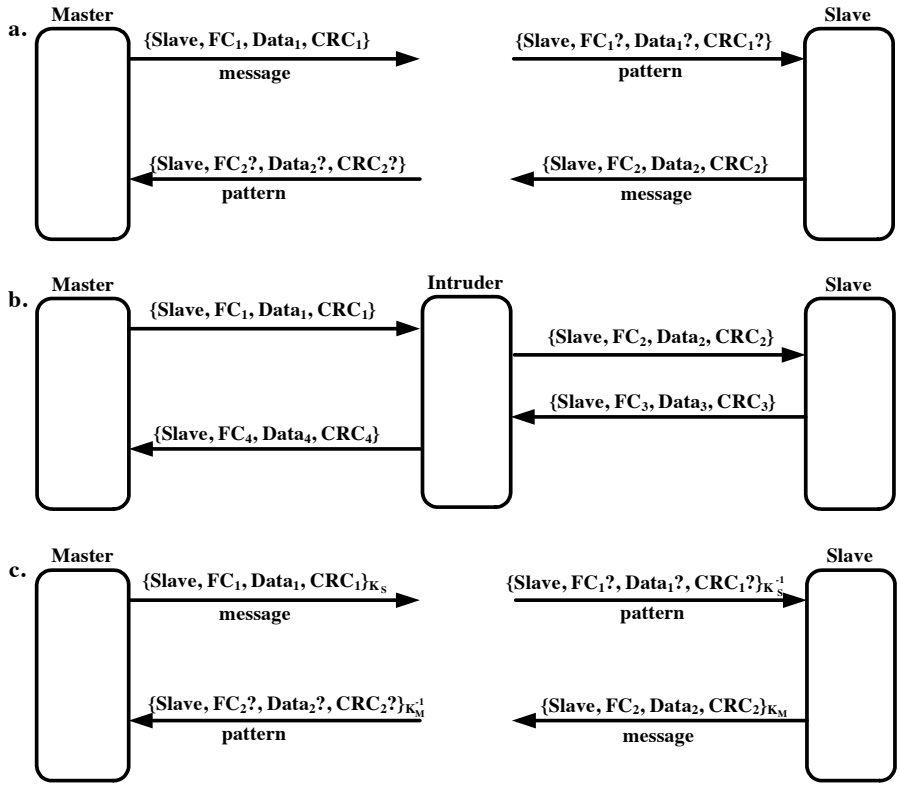

Figure 6. Modbus Serial attack and correction.

forms FC_2 with $Data_2$ instead of FC_1 with $Data_1$. Next, $Slave$ responds with $\{Slave, FC_3, Data_3, CRC_3\}$ where $FC_3 = FC_2$ or $FC_3 = FC_2 + 128$ for positive/negative replies.

At this point, as shown in Figure 6(b), $Intruder$ intercepts $Slave$'s message and sends the fabricated message $\{Slave, FC_4, Data_4, CRC_4\}$ where $FC_4 = FC_1$ or $FC_4 = FC_1 + 128$ to $Master$. Note that the fabricated message is constructed so that it matches the pattern exposed by $Master$ to receive a response message from $Slave$ regarding function FC_1.

One way to defeat this attack is to use asymmetric encryption for Modbus messages. The new specification of a Modbus transaction is shown in Figure 6(c). The private keys of the $Master$ and $Slave$ are K_M^{-1} and K_S^{-1}, respectively; the corresponding public keys are K_M and K_S. The $Master$ uses the $Slave$'s public key K_S to encrypt the request message $\{Slave, FC_1, Data_1, CRC_1\}_{K_S}$, and the $Slave$ exposes the pattern $\{Slave, FC_1?, Data_1?, CRC_1?\}_{K_S^{-1}}$. Matching is verified according to Definitions 2 and 5 (note that $K_S \overset{K}{\sim} K_S^{-1}$). The $Slave$ responds by using the $Master$'s public key K_M to encrypt the message: $\{Slave, FC_2, Data_2, CRC_2\}_{K_M}$, which is matched by the pattern $\{Slave,$

$FC_2?, Data_2?, CRC_2?\}_{K_M^{-1}}$ exposed by the $Master$. Subsequent analysis of the modified protocol reveals that the flaw is fixed.

4.2 Modbus TCP Analysis

Peer-to-peer analysis of Modbus neither ensures that the protocol is secure nor that is being used securely. Layering Modbus Serial on TCP requires that TCP flaws be considered in addition to flaws arising from negative interactions between the two protocols. TCP attacks, e.g., session hijacking, are well known and are not discussed here. In fact, modeling and analyzing TCP – as was done in the case of Modbus Serial – can reveal flaws and verify the security of protocol corrections. This section focuses specifically on a negative interaction produced by layering Modbus Serial on TCP.

Modbus TCP permits PDUs to be transported in TCP/IP messages. In Modbus TCP, master units are called "clients" because they send requests to slaves by initiating a connection using an active open operation on the TCP/IP stack. Slaves are called "servers" because they process requests and send the results to master units by listening, i.e., performing a passive open on the TCP/IP stack using a predefined port.

Figure 7(a) models the corrected Modbus TCP protocol, i.e., the corrected Modbus Serial protocol in Figure 6(c), which is transported by TCP. Note that to simplify the presentation, only relevant fields are shown in the protocol messages. The first four fields of each message capture essential IP data (required for proper processing) in the following order: source IP address, destination IP address, source port and destination port. Encrypting messages at the peer-to-peer level affects Modbus TCP because Modbus headers and PDUs are encrypted (Figure 7(a)). Note also that the Modbus specifications require that messages contain additional MBAP fields (e.g., $transID$, PID, $length$ and $unitID$).

Figure 7(b) presents an attack that exploits a flaw arising from negative interactions between layers of the Modbus TCP protocol. In the attack, the $Intruder$ intercepts the $Client$'s encrypted request message sent from $IP\text{-}C$ ($Client$'s IP address) to $IP\text{-}S$ ($Server$'s IP address). Since the $Intruder$ cannot decrypt the Modbus message (it does not know K_S^{-1}), it changes the TCP/IP data associated with the source of the IP datagram and forwards the Modbus content as is. The $Server$ responds to the $Intruder$'s IP address $IP\text{-}I$ with $\{transID, PID, length_2, unitID, FC_2, Data_2\}$ encrypted with its private key K_S^{-1}. The $Intruder$ decrypts the Modbus payload using K_S and creates a bogus (but valid) Modbus response by encrypting $\{transID, PID, length_3, unitID, FC_3, Data_3\}$ using the $Client$'s public key K_C.

Protocol analysis reveals that encryption – while it maintains message confidentiality – does not ensure message integrity. The attack in Figure 7(b) exploits the flaw, enabling the $Intruder$ to intercept and modify messages and make the $Server$ believe it is the $Client$ (master unit). One might observe that this attack is defeated if the $Server$ (a remote terminal unit) could be configured with the IP address of the $Client$ (the Modbus payload itself does

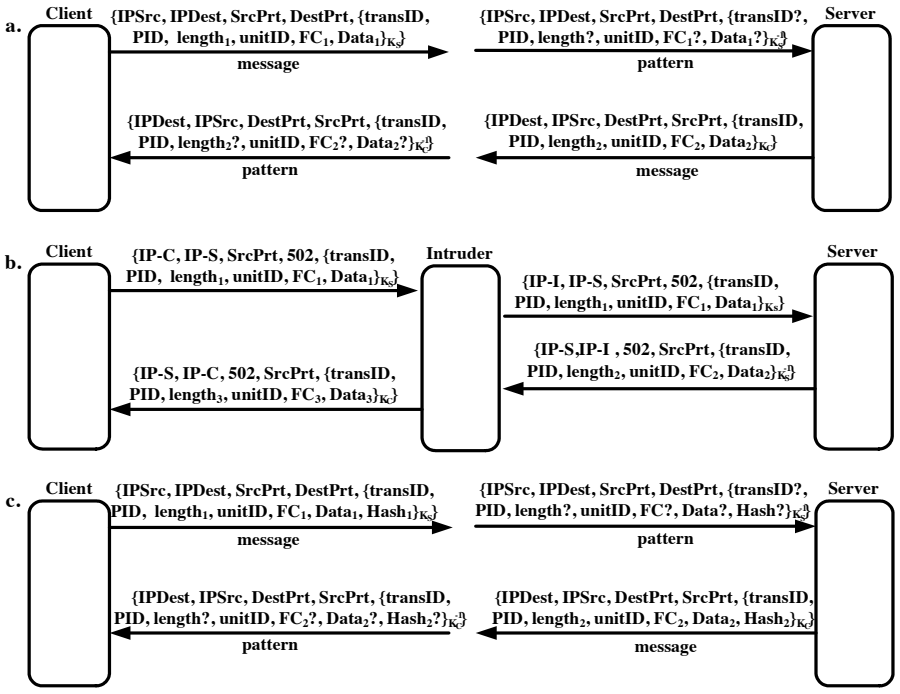

Figure 7. Modbus TCP attack and correction.

not contain any information identifying the *Client*). Clearly in this case, a message from *IP-I* would have been rejected and the attack would have failed. However, due to limited computational power, remote terminal units often deploy minimal TCP/IP stacks with few, if any, security settings.

The corrected protocol in Figure 7(c) appends a two-byte hash value of the IP source and destination addresses, the TCP source port and the Modbus transaction ID to the Modbus PDU. This has the effect of including an additional field (protected by encryption) that is used to establish integrity (by associating IP and Modbus-related data in a single field). Formal analysis of this corrected protocol reveals that it addresses the flaw by preserving Modbus agent authenticity and message integrity.

5. Conclusions

Using carrier protocols such as TCP/IP to transport control protocols in SCADA/distributed control systems, telecommunications and other critical infrastructure networks is efficient and cost effective. However, the layering of protocols raises serious security issues. The original control protocols were designed for use in isolated networks, not to be transported over WANs by carrier protocols. Protocol layering brings into play the vulnerabilities in the

individual protocols and carrier protocols, as well as unexpected vulnerabilities produced by negative interactions between protocol stack layers.

This paper has presented a methodology for formally modeling multilayer protocols and protocol stacks, and systematically analyzing peer-to-peer and inter-protocol interactions. The methodology engages AVISPA, a popular protocol validation tool, to identify flaws and verify that corrections satisfy the desired security goals. As such, the methodology is useful for analyzing the security properties of existing multilayer protocols as well as "secure" industry implementations involving legacy and transport protocols. With additional research and refinement, this methodology could serve as the foundation for designing inherently secure protocols and protocol stacks for next generation critical infrastructure networks.

Acknowledgements

This work was partially supported by the Institute for Information Infrastructure Protection (I3P) under Award 2003-TK-TX-0003 from the Science and Technology Directorate of the U.S. Department of Homeland Security.

References

[1] M. Abadi and B. Blanchet, Analyzing security protocols with secrecy types and logic programs, *Journal of the ACM*, vol. 52(1), pp. 102–146, 2005.

[2] R. Anderson and R. Needham, Programming Satan's computer, in *Computer Science Today: Recent Trends and Developments (LNCS Vol. 1000)*, J. van Leeuwen (Ed.), Springer-Verlag, Berlin, Germany, pp. 426–440, 1995.

[3] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P. Heam, O. Kouchnarenko, J. Mantovani, S. Modersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Vigano and L. Vigneron, The AVISPA tool for the automated validation of Internet security protocols and applications, *Proceedings of the Seventeenth International Conference on Computer-Aided Verification*, pp. 281–285, 2005.

[4] D. Basin, S. Modersheim and L. Vigano, An on-the-fly model checker for security protocol analysis, *Proceedings of the Eighth European Symposium on Research in Computer Security*, pp. 253–270, 2003.

[5] D. Dolev and A. Yao, On the security of public key protocols, *IEEE Transactions on Information Theory*, vol. 29(2), pp. 198–208, 1983.

[6] L. Dryburgh and J. Hewitt, *Signaling System No. 7 (SS7/C7): Protocol, Architecture and Services*, Cisco Press, Indianapolis, Indiana, 2005.

[7] J. Edmonds, Security Analysis of Multilayer Protocols in SCADA Networks, Ph.D. Dissertation, Department of Computer Science, University of Tulsa, Tulsa, Oklahoma, 2006.

[8] S. Escobar, C. Meadows and J. Meseguer, A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties, *Theoretical Computer Science*, vol. 367(1-2), pp. 162–202, 2006.

[9] L. Lamport, The temporal logic of actions, *ACM Transactions on Programming Languages and Systems*, vol. 16(3), pp. 872–923, 1994.

[10] G. Lowe, A hierarchy of authentication specifications, *Proceedings of the Tenth Computer Security Foundations Workshop*, pp. 31–44, 1997.

[11] G. Lowe, Casper: A compiler for the analysis of security protocols, *Journal of Computer Security*, vol. 6, pp. 53–84, 1998.

[12] Modbus IDA, MODBUS Application Protocol Specification v1.1a, North Grafton, Massachusetts (www.modbus.org/specs.php), June 4, 2004.

[13] Modbus IDA, MODBUS Messaging on TCP/IP Implementation Guide v1.0a, North Grafton, Massachusetts (www.modbus.org/specs.php), June 4, 2004.

[14] Modbus.org, MODBUS over Serial Line Specification and Implementation Guide v1.0, North Grafton, Massachusetts (www. modbus.org/specs.php), February 12, 2002.

[15] L. Ong, I. Rytina, M. Garcia, H. Schwarzbauer, L. Coene, H. Lin, I. Juhasz, M. Holdrege and C. Sharp, Framework Architecture for Signaling Transport, RFC 2719, October 1999.

[16] M. Papa, O. Bremer, J. Hale and S. Shenoi, Formal analysis of E–commerce protocols, *IEICE Transactions on Information and Systems*, vol. E84-D(10), pp. 1313–1323, 2001.

[17] M. Papa, O. Bremer, J. Hale and S. Shenoi, Integrating logics and process calculi for cryptographic protocol analysis, in *Security and Privacy in the Age of Uncertainty*, D. Gritzalis, S. De Capitani di Vimercati, P. Samarati and S. Katsikas (Eds.), Kluwer, Boston, pp. 349–360, 2003.

[18] L. Vigano, Automated security protocol analysis with the AVISPA tool, *Proceedings of the Twenty-First Conference on the Mathematical Foundations of Programming Semantics*, pp. 61–86, 2006.