MODBUS Covert channel

Carlos Leonardo (cal3678@rit.edu)[1] **and Daryl Johnson (dgjics@rit.edu)**[2]
[1] and [2] Department of Computing Security, Rochester Institute of Technology, Rochester, NY, United States

Abstract— *The security community already has seen some examples of actual attacks against real SCADA installations, like the Stuxnet case in 2010 [3]. MODBUS is one of the most used communication protocols in industrial control systems. Even though the protocol itself is known to lack basic security features; there is not much detail available about real world cases where MODBUS has been used as an attack vector. Covert channels have been mentioned several times as part of the security vulnerabilities of SCADA systems [1], [2] and [4]. This research targets the MODBUS protocol characteristics to introduce a covert channel. This covert channel allows information leakage from one device called covert Master to another one called covert Slave. The covert Master is supposed to be on the internal LAN, while the covert Slave is expected to be on a separate subnet. The Slaves subnet could be based on either Ethernet and TCP/IP or a serial BUS (RS485) using a media converter to reach the LAN.*

Keywords: SCADA, MODBUS, Covert Channels

1. Introduction

To appreciate the importance of the MODBUS communication protocol and the impact that a MODBUS covert channel could have, it is necessary to briefly describe SCADA Systems. SCADA stands for Supervisory Control and Data Acquisition. These systems usually control critical infrastructure for economy stability, such as power generation and distribution. MODBUS was created in 1979 by Modicon (now Schneider) and has been used since then as one of the main communication protocols for SCADA installations [1]. MODBUS is so important in first place because it is used to control and monitor critical infrastructure that is everywhere.

Being a layer 7 protocol, MODBUS was initially used on RS485 serial networks. However, it is now common to see MODBUS working over Ethernet networks with TCP/IP or in a combination of both. These recent configurations that integrate MODBUS on the LAN networks and internet have also opened the possibility of attacking the SCADA systems in similar ways that information systems are. This covert channel can use the MODBUS infrastructure in place to leak information from the LAN network, which is a valuable ability to have when approaching a target with valuable data. SCADA devices often are physically located outside of the protected building and gated environments. For example, an electrical power meter connected to the MODBUS network would be located on a remote sub-station or even a residencial area. This makes it a candidate for data exfiltration and infiltration.

Although some SCADA firewall solutions exist [5], the common approach is to block non valid or malformed MODBUS traffic as well as *write* transactions. This covert channel uses valid and *read-only* transactions to operate which makes it harder to detect or stop. Also, there are very low chances of having a SCADA system using such type of security implementation since those are still considered a new field compared with the time SCADA systems have been in use. MODBUS has been chosen for this covert channel research because of its importance and common use in SCADA systems as well as the lack of security in its design. SCADA security is still considered, as are covert channels, a novel subject that usually is not taken into account when implementing information security controls.

2. Related work

While the MODBUS protocol has been used for decades and SCADA systems are everywhere; it is not common to see MODBUS being used for purposes other than the monitoring and control of SCADA equipment. Some published papers and books have raised the flag indicating that SCADA systems need more attention from the information security community because of their importance [2] and [3].

A good example is the Stuxnet case, mentioned by Kim-Kwang [3] as a recent and high profile attack committed against SCADA systems. Other publications [4] also talk about the SCADA Security issue and how it has changed over the years. Originally, the security community was more worried about physical threats affecting the SCADA systems like sabotage; however, now the concern is also about a new wave of electronic and information based threats affecting them.

Knapp [1] even mentions that the MODBUS protocol lacks some basic security features such as authentication and encryption and says that SCADA Intrusion prevention systems (IPS) could monitor malicious activities using MODBUS signatures [1]. He also proposes that MODBUS sessions could be validated to ensure that MODBUS has not been "hijacked" and used for covert communication [1]. However, there is no evidence of an actual implementation of a MODBUS covert channel.

The SANS Institute (SysAdmin, Audit, Networking, and Security Institute) issued a document called “Using SNORT for intrusion detection in MODBUS TCP/IP communications“ describing a method of intrusion detection (IDS)

by developing rules for SNORT [8]. The objective of that document is to provide useful information about how to implement IDS systems for MODBUS by using open source technologies, instead of expensive and limited ones that are commercially available.

avtheir email addresses (unless they really want to.)

3. MODBUS Protocol and Data structure

The MODBUS protocol is well defined in the official documents [6] and [7] issued by the MODBUS Organization. This paper only exposes the protocol information that is closely related to the covert channel intended to be implemented.

MODBUS was originally created to work on serial networks like RS485 (figure 1) and that is why some terminology is different from what it is used in modern Ethernet and TCP/IP based networks (figure 2). In a MODBUS network there are always two types of devices: several field Slaves (called Servers in MODBUS TCP/IP) and a single Master (or several Clients as called in MODBUS TCP/IP). Only one Master can exist in a MODBUS RS485 serial network. Slaves generate errors if they receive requests from more than one master. This is obviously different in TCP/IP based networks where several clients can coexist as long as they are synchronized to avoid sending simultaneous requests. Client(s) and servers are usually in different subnets, the client being a computer using TCP/IP with SCADA software and the servers being field devices wired within a serial BUS (RS485) network and using a media converter to reach the LAN (figure 3). No matter what lower layer topology is in use, the protocol data structure changes minimally and it is relatively simple when compared with HTTP or other layer 7 protocols. We use Client for the Master device and Server for each Slave. Three different MODBUS networks layouts are shown below.

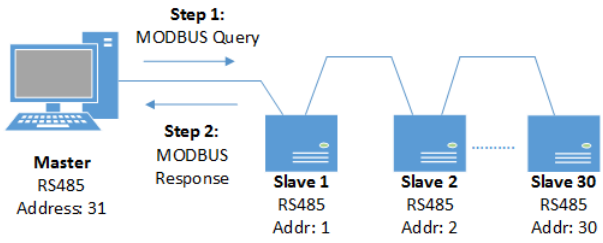

Fig. 1: Legacy MODBUS RTU working on a Serial BUS network (RS485).
One Master → Many Slaves.

The MODBUS servers are the field devices capable of measuring and/or controlling the environment by using inputs and outputs. The inputs can be digital (Discrete Inputs) and analog (Input Registers). Servers measure the

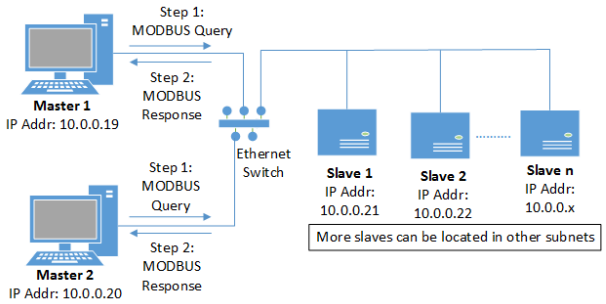

Fig. 2: New MODBUS TCP working on a TCP/IP network (used by this covert channel).
One / many TCP Clients → Many TCP Servers

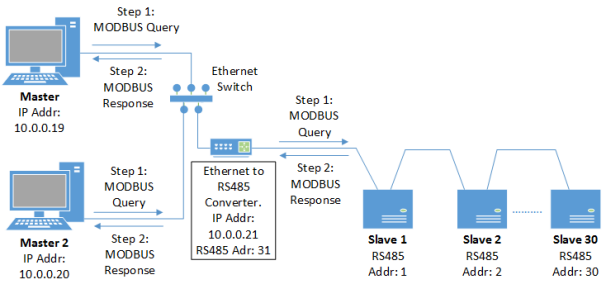

Fig. 3: Mixed MODBUS network (using media converters from RS485 to TCP/IP).
One / many TCP Clients → Converter → Many serial Slaves

environment variables by using these inputs and store their updated values in local memory tables (Table 1). The client can later request a server for the actual value of its inputs. The outputs can be also digital (Coils) and analog (Holding Registers). Coils are used to open or close a digital switch while holding registers are used to vary an analog output value within a range. The client can request a server to change the value of these outputs in the local memory tables (Table 1), thus controlling the environment. Not all servers have all types of inputs, outputs and functions since this is a vendor-specific decision.

Servers organize the data in four primary tables allocated for Discrete Inputs, Coils (digital outputs), Input Registers (analog inputs) and Holding Registers (analog outputs) [6]. Table 1 shows some details about each one of these primary tables [6]. The old version of the MODBUS protocol allows 9,999 data objects in each one of the four primary tables while the new versions of MODBUS allow 65,536 data objects. The proposed covert channel uses object numbers between 0 and 9,999 so it can be implemented in MODBUS networks using both schemas.

The MODBUS client device on the other hand only requests data from the servers and stores it in long term memory for future processing. Servers always wait for a client's request and never initiate a conversation because the

Primary Tables	Object Size	Object access	Comments
Discrete Inputs	Single bit	Read-Only	This data can be provided by an I/O system.
Coils	Single bit	Read-Write	This data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This data can be provided by an I/O system.
Holding Registers	16-bit word	Read-Write	This data can be alterable by an application program.

Table 1: Primary Tables in a MODBUS Server device.

protocol is based on requests and responses.

The MODBUS Application Data Unit (ADU) contains a simple Protocol Data Unit (PDU) and some additional fields introduced by the network topology in use [6] as illustrated in the figure 4:

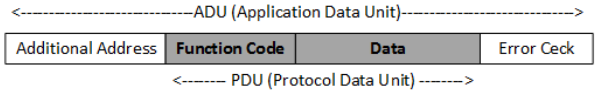

Fig. 4: MODBUS Application Data Unit (ADU)

The messages MODBUS client and servers use to communicate to each other contain a function code of 1 byte long and a data field of variable size. The function code specified in the client request specifies the action (read or write) and the type of object (digital or analog). The data field contains what object(s) will be affected by the function [6]. A complete list of Function codes and their meaning is available on the protocol standard document [6].

4. A Covert channel taking advantage of function codes

The MODBUS protocol uses two principal elements to establish communication between client and servers that are also used in this covert channel: Tables and Function Codes. The four primary tables (discrete inputs, coils, input registers and holding registers) are the four different sets of variables that can exist in a server device (see table 1). For each one of these types of variables, also called objects, there are up to 9,999 in the older MODBUS versions and 65,535 in newer MODBUS versions.

The second important element in the protocol is the set of function codes available that indicate what action is to be taken by both, the client and the server. The client uses different function codes to specify if it is going to read a discrete input, to write a coil, to read an input register, to write holding register, etc. The servers use function codes to indicate if they are sending a response with the value of a discrete input or a holding register, throwing an error exception code, etc. [6]. There are up to 127 function codes (1 byte value) divided in three groups: Public (1 to 64), User-defined and Reserved. Public function codes are guaranteed to be unique, publicly documented and widely used by the majority of devices [6].

5. Implementation

The covert channel proposed in this paper operates by using the read-only public function codes from 1 to 4 and the object numbers from 0 to 9,999 of MODBUS servers. The covert channel exists between a covert client and a covert server both connected to a TCP/IP network and using MODBUS TCP (figure 5); however, the channel is designed so it can also work with MODBUS RTU in a serial bus network (RS485). Both devices, client and server are separate hosts with a software-based MODBUS implementation. For the purpose of this work, the Python library *pymodbus* will be used [9]. This software library allows the instantiation of MODBUS servers and clients at will by using two different Python scripts. A script called *sender* resides on the client device while another script called *receiver* resides on the server device.

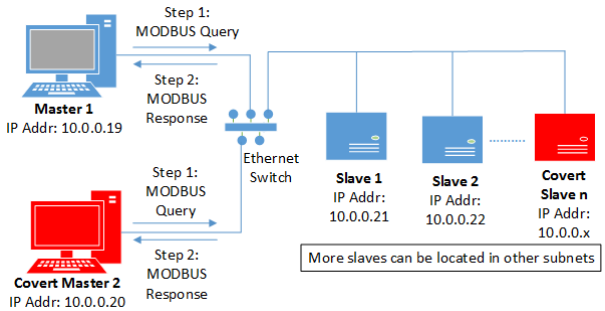

Fig. 5: Covert devices: They can be software based (*pymodbus*) and work in a RS485 or TCP/IP network

To establish the covert channel, a covert client sends a request asking to read a covert server's object (coil, discrete input, holding register or input register). Then, the covert server receives the request, verifying the function code and the object number that is requested. This object number is mapped to a pre-defined ASCII character. This pre-defined "covert" value is the value that the client was intending to send to the server.

To circumvent SCADA firewalls that might be in place, the covert client always uses a read-only function code like 01 to read coils, 02 to read discrete inputs, 03 to read holding registers or 04 to read input registers. The covert client also specifies the number of the object it wants to read (0 to 9,999), which is in fact the ASCII value it wants to transmit.

An example of a normal MODBUS transaction with two steps (Response and Request) is explained below:

1 - The Client sends a Request to the Server asking to read the object 110 (0x6E):

Function Code	Input Register number (8 bits long)
04	110 (0x6E)

2 - The Server receives the Request and sends back a Response with the value of the object number 110. If the value is "30", the Response will look like this:

Function Code	Input Register number	Input Register value (16 bits long)
04	110 (0x6E)	30 (0x001E)

When the covert channel is implemented, the server executes a third step without changing the Request/Response schema. In the example, what the covert channel requires is to have the server interpreting the requested object number 110 (0x6E) as an ASCII character, which is the character "n". The covert server also answers the request with the value of the register 110 to make this look like a legitimate MODBUS request/response message. However, the actual value stored in the object 110 is irrelevant to the covert channel. Every time the client is requesting to read the server's Input Register number 110 (0x6E), in fact, the covert client is *sending* the ASCII character "n" to the covert server (see section 8, alphabet in use).

This behavior is illustrated with the figure 6. The first and second steps represent a normal MODBUS transaction, while the third step on the right represents the covert behavior.

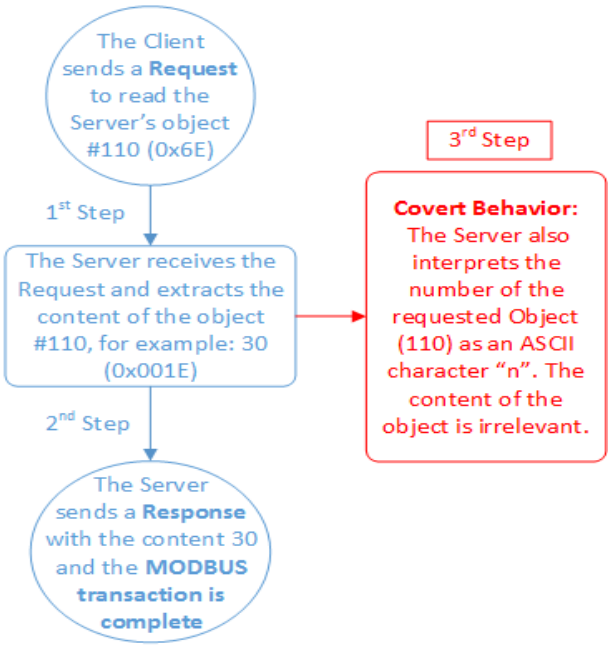

Fig. 6: First and second steps represent the normal behavior of a MODBUS transaction while the third step is the covert channel implemented

6. Challenges

The covert channel proposed in this paper is implemented by changing the way the server interprets the MODBUS PDU without changing the protocol structure. The covert channel consists of a series of MODBUS messages that are interpreted by the covert client and server. The messages are valid MODBUS client requests and server responses. Although SCADA systems are usually not taken into account when implementing security controls on the data networks [1]; there are solutions available to secure MODBUS installations.

Such controls represent a challenge for the covert channel and have been taken into account as an obstacle to overcome. As an example, it is important to consider that a SCADA installation could have a Tofino Firewall [5], SNORT IDS [8] or similar solutions implemented. These security solutions can look into the MODBUS PDU and block or alert of dangerous or abnormal messages. However, their common approach is to validate that the MODBUS requests and responses are well structured and valid, blocking the *write* requests as well as the malformed packets and non-MODBUS traffic. This covert channel has taken into account that some security measures might be in place and uses *read-only* and well-formed MODBUS requests that would be considered normal traffic.

7. Drawbacks

If an IDS solution is in place and filters all the commands sent to the SCADA network, this covert channel might have problems to send and receive all or some of the characters in the alphabet. However, for the SCADA system to work, whatever solution that is in place must allow the SCADA devices to communicate using the commands, device addresses and register numbers that are valid for that installation. Since the covert channel is aimed to use the same valid commands, addresses and register numbers, the covert channel should work properly.

8. Alphabet in use

The actual implementation of this covert channel uses the following alphabet structure:

Object_Number_X	=	ASCII_Character_X

How to use this alphabet is explained in more detail in the section 5 (implementation). This alphabet consists in a pure conversion from the object number to the ASCII character number. When the cover client sends a request, the covert server verifies the object number specified in the request and interprets what ASCII character corresponds to it. For example, the object number 110 (0x6E) maps to the ASCII character âĂŸâĂŹn". This direct mapping keeps the test alphabet simple while supporting all the ASCII characters. However, this is not necessary and it is possible to have a

different alphabet by using a substitution table to obfuscate the characters that are being sent. In that case, the object number 110 (0x6E) can be mapped to any other ASCII character. This model needs at least 256 objects in the covert server to be able to receive all the ASCII characters, including the non-printable and extended ones. The section 10 (future work) explores other schemas for the alphabet.

9. Covert Channel Classification

9.1 Type

The covert channel described in this paper is considered as a behavioral covert channel. This is because the covert data is not contained as a payload within the MODBUS Request itself, but it is extracted depending on how the Request is interpreted. The covert server extracts the covert data when it interprets the requested object number. The covert server takes the number of the requested object and looks for the ASCII character associated with that number. Depending on which object is requested, the covert server interprets the ASCII character that was sent.

9.2 Throughput

The throughput of this covert channel will depend on the characteristics of the targeted installation. Some SCADA installations only work with RS485 serial devices, which reduce the available bandwidth considerably. However, if 256 objects are used, a minimum of one byte can be transmitted per Request/Response transaction. It is normal to have one transaction every one or two minutes in the older installations that only use RS485 serial devices.

To support all the ASCII characters (one byte per transaction), the covert channel needs to be implemented in a MODBUS network where it is normal to have 256 different objects in the servers. In the future work section, a hexadecimal alphabet is considered, which would require only 16 different objects to work while it cuts the throughput in half.

It is important to understand that the receiver Python script, located on the server device and explained on the implementation section, allows the attacker to emulate up to 65,535 objects in the server. That number of objects can be used to create a bigger alphabet, which will allow transferring more data. However, this will increase the chances of being discovered, as explained in the detection section.

9.3 Robustness

Even though the majority of SCADA systems are usually not protected with security controls on the network level; there are commercial and open source solutions available to secure MODBUS networks. The MODBUS security solutions available can look into the PDU and block or alert of dangerous or abnormal transactions. This covert channel was designed considering these solutions as an obstacle to overcome. All the covert messages are legitimate Request/Response transactions that must be accepted by the security controls in place, if any, to allow the MODBUS devices to work, making the covert channel considerably robust.

9.4 Detection

To avoid detection, the attacker must know the normal behavior of the targeted network. Especially, it is critical to know how many objects are available in the servers of the network and use a similar number of objects in the covert server, which is emulated with *pymodbus* in the prove of concept. After taking care of this aspect, the critical point is to hide the sender and receiver scripts installed in the compromised devices. At this point, at least one client and one server are required, but in the future work section, another approach is considered to avoid using a compromised server, which will reduce the chances to be discovered.

Even though only 256 objects are required, the MODBUS original standard indicates that the servers could have up to 9,999 objects for each one of the four tables. If a server device with these characteristics is used, it is possible to send up to 9,999*4 = 39996 different values, but chances are the covert channel is found easily because those object numbers are not usually in use. The same happens if this is implemented in a modern MODBUS network, using a server device with 65,535 different objects per table. The cover client will be able to send up to 65,535*4 = 262140 different values but the communication can be suspicious.

9.5 Prevention

Measures that can be implemented to prevent this convert channel are related to how well the SCADA network configuration is documented and audited over time. If security controls like Tofino firewall or Snort IDS are implemented, they have to be configured to allow not only valid MODBUS objects, but only the object numbers that are used in that particular network. If the MODBUS network uses at least 256 objects in one sever device, then, that is enough to implement the covert channel. If the Hexadecimal alphabet is used as described in the future fork section, only 16 objects are enough. The physical security is extremely important, since the MODBUS network usually include devices installed in remote and unattended locations where contractors have access.

10. Future Work

A future implementation of this covert channel would use a slightly different alphabet that links an ASCII character with a combination of a Sever ID and an Object Number, thus, resulting in this structure:

“Sever_ID_Y + Object_No_X = ASCII_Char_Z”

In this new approach, the covert server is listening to all the requests sent by the covert client, including the ones sent to other severs in the network. Then, the covert sever interprets the ASCII character that the covert client is sending by combining the sever ID and the object number contained in the request. The advantage here is that the alphabet is "distributed", augmenting the stealth level. The 256 ASCII characters can be interpreted by using several severs IDs and the object of those severs instead of using 256 objects of one single server.

A future alphabet based on 16 characters would reduce the amount of objects needed from 256 (to send all the ASCII Characters individually) to only 16 (to send values from 0x0 to 0xF). The actual ASCII characters would be "assembled" by using two 4-bits values.

Another future implementation of this covert channel would work with serial devices (RS485) only instead of TCP/IP based systems, since an important number of MODBUS installations run over serial BUS networks. This can be done using the same Python library (*pymodbus*) used for this covert channel.

Since a MODBUS server can't initiate a communication (it just responds to requests), the two-way communication could be implemented by having a timing mechanism. For this to work; the covert client has to send a request to the covert server with a special pre-defined character (like 0x05, the Enquiry Character in ASCII). This character is used for asking if new data is available and ready to be sent from the covert server. If the covert server has data to send or not, it will respond with another pre-defined pair of special characters indicating so. The covert client will then keep requiring the next character from the server until it receives the last one indicating there is no more data (0x04 or End of Transmission character for example).

An interesting experiment would be implementing this covert channel as well as an IDS solution based on Snort as described by Díaz [8]. The experiment would consist in determining if it is possible to detect the covert channel with the filters proposed [8]. It would be useful to establish if the covert channel can overcome this security control and how it can be improved to leave a smaller footprint if necessary. The results could demonstrate how deep the SNORT rules have to look into the packets in order to detect the covert channel. The hypothesis is that the IDS rules should not interfere with the valid MODBUS device addresses and object numbers. If the rules interrupt the traffic, trying to block the covert channel, they will also make the SCADA system unusable. This is because the covert channel uses only valid device addresses and registry numbers to operate.

References

[1] Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and other Industrial Control Systems. Chapter 4 - Industrial Network Protocols. Eric Knapp. Syngress Publishing 2011. ISBN:9781597496452

[2] Cyber security risk assessment for SCADA and DCS networks. P.A.S. Ralstona, J.H. Grahamb, J.L. Hiebb. University of Louisville, JB Speed School of Engineering, Louisville, KY, United States. Department of Computer Engineering and Computer Science, University of Louisville, KY, United States. Available online 10 July 2007.

[3] The cyber threat landscape: Challenges and future research directions. Kim-Kwang Raymond Choo. School of Computer and Information Science, University of South Australia, Mawson Lakes campus (Room F2-28), Mawson Lakes, SA 5095, Australia.

[4] The SCADA challenge: securing critical infrastructure. By Steve Gold. Article from Network Security. August 2009.

[5] Tofino security Appliance home website. http://www.tofinosecurity.com/

[6] MODBUS Application Protocol Specification V1.1b by MODBUS Organization. December 28, 2006. Retrieved from http://www.Modbus.org

[7] MODBUS Messaging On TCP/IP Implementation Guide V1.0a by MODBUS Organization. June 4, 2004. Retrieved from http://www.modbus.org

[8] Using SNORT for intrusion detection in MODBUS TCP/IP communications. By Javier Jiménez Díaz (javier.jimenez@coit.es) and Robert Vandenbrink. SANS Institute InfoSec Reading Room. December 7th, 2011.

[9] pymodbus Python Library. Retrieved from https://pymodbus.readthedocs.org/en/latest/index.html

[10] Covert Channels in the HTTP Network Protocol: Channel Characterization and Detecting Man-in-the-Middle Attacks. By Erik Brown (erik.t.brown@gmail.com), Bo Yuan (bo.yuan@rit.edu), Daryl Johnson (daryl.johnson@rit.edu), Peter Lutz (peter.lutz@rit.edu). Rochester Institute of Technology, Rochester, NY, USA