A Modbus command and control channel

Conference Paper · April 2016

DOI: 10.1109/SYSCON.2016.7490631

CITATIONS
15

READS
537

3 authors, including:

Jose Manuel Fernandez
Polytechnique Montréal
82 PUBLICATIONS **1,248** CITATIONS

SEE PROFILE

Scott Knight
Royal Military College of Canada
51 PUBLICATIONS **386** CITATIONS

SEE PROFILE

A Modbus command and control channel

Antoine Lemay, *École Polytechnique de Montréal* Scott Knight, *Royal Military College of Canada* José M. Fernandez, *École Polytechnique de Montréal*

Abstract

Since the discovery of Stuxnet, it is no secret that skilled adversaries target industrial control systems. To defend against this threat, defenders increasingly rely on intrusion detection, but no good ICS attack data sets exist to evaluate the effectiveness of those solutions. This paper presents a command and control (C&C) covert channel over the Modbus/TCP protocol that can be used to generate attack data sets representing the next logical step for the attackers. The channel stores information in the least significant bits of holding registers to carry information using Modbus read and write methods. This offers an explicit tradeoff between the bandwidth and stealth of the channel that can be set by the attacker.

1. Introduction

In this information age, it would come as no surprise to hear that most of our industrial control systems (ICS) are vulnerable to cyber attacks by skilled adversaries. The discovery of Stuxnet [1] revealed to the world that such attacks were not only possible, but could also remain undetected for a long time.

To address this issue, a number of researchers have started to look at intrusion detection for SCADA systems. However, this research is hobbled by the fact that few data sets of SCADA traffic exist and even fewer data sets of SCADA attacks are available. These attack data sets usually contain specific network exploits, often relying on malformed packets for which signature-based IDS and anomaly detection work very well. Non-SCADA specific attack tools produce network traffic that, while sufficiently stealthy to hide in protocol diverse customer-based IT networks, stands out like a sore thumb against the background of the regular machine-to-machine communication of Operational Technology (OT) networks.

As detection capability increases, it is unlikely that attackers will continue using overt communication channels to maintain command and control (C&C). As such, it is important to test our defensive tools against threats that represent reasonable next steps for attackers. One of these steps is increased stealth in C&C communication for attacks against OT. In particular, the use of covert channels on SCADA protocols, meaning a command and control channel that is hiding its presence within an existing ICS communication protocol.

This paper presents a command and control covert channel over the Modbus protocol. It starts by presenting a brief overview of the operational scenario and some background information on OT systems. Then, relevant previous work in the field of information hiding is surveyed. This is followed by a presentation of our Modbus command and control channel and by a performance analysis of that channel. Finally, a brief conclusion summarizes the results and present avenues for future work.

2. Operational scenario

For an attacker aiming to create physical impacts through cyber attacks, it is necessary to access the control system directly. For most control systems, which are not directly accessible from the Internet, a significant investment on the part of the attacker is required to establish a foothold on the control network. For example, the attacker may have to entice an operator to a watering hole site, infect a machine through physical access or perform a pivot through a third party providing remote support.

Once inside the control network, the attacker can start to work toward his objective. However, it is unlikely that his initial point of presence will provide the attacker the access he requires to fulfill his goals. The attacker will need to expand his presence from his initial foothold to reach the machine controlling the piece of equipment enabling the correct physical impact and gain control of that machine.

Having gained control of his ultimate target (or targets) inside the ICS network, the attacker needs to maintain his presence inside the network. In most attack scenarios, maintaining the presence requires frequent communications. For example, the compromised machine must frequently report the status of the implant to the person controlling it. Additionally, the attacker might want to update the implant with new order (i.e. commands) or new functionalities. This requires an active channel between the attacker and the target.

This is usually done by establishing a direct command and control channel between the target and the attacker. For example, through the establishment of a reverse shell. However, this is often impractical in an ICS network because the ultimate targets, the machines controlling the physical systems, are not designed to have human operators and may be forbidden to directly call out to the Internet. Furthermore, if they tunnel back through the initial point of presence inside the ICS network on an arbitrary TCP port, they are at risk of revealing the initial compromise, forcing the attacker to expend significant resources to establish a new one.

In this situation, we expect the attacker to invest in stealth to preserve the investment necessary to establish the initial point of presence. This is particularly true of command and control traffic which must occur frequently. The repeated nature of these communications increase the risk of detection even if the overall amount of traffic sent is low or if the traffic is sent over a long period of time.

3. Background

In order to build a command and control channel for an industrial control system network, background information on these networks and on the protocols they use is required. This section will present a brief summary of the general architecture of an industrial control network. Then an overview of the relevant features of the Modbus protocol will be detailed.

3.1. Industrial control network

An industrial control network is a distributed implementation of a basic control loop. A desired state is specified by an operator in a Human Machine Interface (HMI) sitting on an operator workstation. A host of remote sensor devices estimate the state of the process. A central controller, called the Master Terminal Unit (MTU), and/or a number of Remote Terminal Units (RTU) and distributed Programmable Logic Controllers (PLC) evaluate the distance between the estimated state and the desired state, and then send commands to actuator devices to reduce the distance between the two states.

Typically, the PLCs and the sensor/actuator devices are located close to the physical equipment being controlled. This can be explained by the fact that the sensors and actuators are required to be physically attached to the physical equipment. In turn, the sensors and actuators have to be connected to the PLC or RTU to allow remote operation. This connection is typically implemented through short-range communication technologies such as serial cables or wireless networks. The PLC and RTU are commonly interconnected on a local network using standard TCP/IP protocols, called the plant LAN.

On the other hand, the machines that perform central monitoring have no requirement to be located in proximity to the industrial process. In fact, being close to the industrial process could even be dangerous for the human operators. As such, the machines associated with the supervisory capacity are typically concentrated in a single control center, on the control center LAN, and connect remotely to other devices. The network presented in Figure 1 exemplifies what a typical industrial control network would look like.

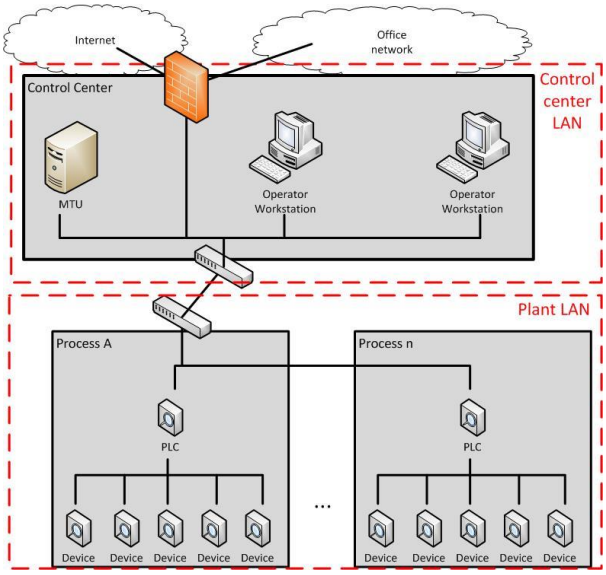

Figure 1: Example of industrial control network

In common attack scenarios, there is a need for the attacker to establish communications between the two LAN. A typical attacker will either gain control of an operator workstation through human interaction or will gain control of a PLC through lax physical security in the remote site or via a USB infection. While no explicit restrictions on communications are present, these networks contain very little traffic other than process control traffic. As such, if an attacker requires stealth, regular communication between the two zones has to resemble process control traffic.

3.2. Modbus protocol

Modbus TCP [2] is an example of an industrial control process protocol. Its main purpose is to harmonize the global state, kept in the master units, with all the local states, kept in the controllers. Two main types of exchanges are used for this purpose: state update requests and state alteration commands.

Because Modbus TCP is derived from the serial version of the Modbus protocol, it follows a master-slave archi-

tecture. This means that the communications are always initiated by the master unit, i.e. the MTU. This is the case for both communications exchanges covered in this section.

In a state update exchange, the master unit will send a request to read the values stored in one or more memory fields on the controller. The memory fields can be of four types:

- Coil : A settable binary field;
- Binary Input : a read-only binary field;
- Holding Register : a settable 2 bytes register field;
- Input Register: a read-only 2 bytes register field.

This request will take the form of a Modbus packet with the appropriate function code, the memory address of the value(s) being read and the number of values read. For example, to read a coil at offset 0, a packet with function code 1 (read coils), offset 0 and word count 1 will be sent. To read 4 holding registers at offset 10, a packet with function 3 (read holding registers), offset 10 and word count 4 will be sent.

To conclude the exchange, the controller will send a response packet. The response packet will include the function code that was used in the request and a series of fields containing the values. If binary values are being read, the fields will be 1 byte long and contain up to 8 values. If register values are being read, the fields will be 2 bytes long and will contain a single 16 bit unsigned integer.

In addition to sending a request, a master unit can also send a command to change the value contained in one or more memory fields on the controller. Because some of the memory fields are read only, only two types of memory fields are considered:

- Coil : A settable binary field;
- Holding Register: a settable register field.

The command will take the form of a Modbus packet with the appropriate function code, the memory address of the value(s) being written, the number of values being written and the content being written. Function code 5 will write a single coil at a given offset and function code 15 will write multiple adjacent coils. Function code 6 will write a single register at a given offset and function code 16 will write multiple adjacent registers.

To conclude the exchange, the controller will send a response packet. This packet will contain all the information included in the packet sent by the master unit and serves as a confirmation that the appropriate actions have been carried out.

4. Previous work

Research on covert communication and information hiding has a long and prolific history. Since the seminal work of Shannon in estimating channel capacity, numerous researchers have developed methods to carry information between an emitter and a receiver, whether that channel is overt or covert.

The development of covert communication channels over networks is a more focused field. Zander *et al.* [3] present a survey of covert channels and countermeasures in network protocols. In their work, we can observe that one foci of their research is on finding unused portions of headers where information can be stored. For example, the TCP sequence number could be customized to store information or a given sequence of HTTP headers could correspond to a certain bit value.

This kind of information hiding is dependent on finding specific protocol idiosyncrasies, dubbed vulnerabilities in the information hiding literature, that can be exploited to store information. If the number of vulnerabilities is low, only a small amount of information can be carried per packet. Additionally, some of these vulnerabilities are not reliable. Values such as TTL fields can be altered by network equipment in transit, destroying the information stored within. The work of Smith and Knight [4] have adressed the issue by proposing a predictable design methodology, but at the cost of adding error correction, further reducing the channel bandwidth.

Instead of hiding information in header fields, some researchers have resorted to hide information within the payload of the packets. In particular, the goal is to hide information of a certain type in the payload of a packet pretending to carry a different type of information. For example, tunneling TCP communications in the payload of an ICMP packet, as is the case in the Ping Tunnel tool [5].

In addition to header fields and packet payload, it is also possible to abuse protocol specifications. For example, OzymanDNS [6] abuses recursive DNS queries to tunnel TCP communications over DNS. As is the case with Ping Tunnel, because network operators do not expect this kind of traffic over DNS, that information is in effect covert.

However, both types of communication significantly alter the information properties of the channel they use. As an example, the payload of ICMP echo request packets is commonly fixed to a set value. Introducing

information in that payload instantly reveals the existence of the channel. Additionally, in our operational context of a SCADA network, it is not possible to rely on a large volume of ICMP and DNS traffic to hide our communication. A dedicated approach leveraging a SCADA protocol is needed.

To create a channel on a SCADA protocol, we could either find a vulnerability in the protocol headers that could be exploited to store information, or store information in the payload. However, if we want to avoid distorting the information properties of the protocol, we need an information hiding method that preserves to a certain degree the structure of the content of the payload.

Looking at more traditional information hiding research, it is possible to find such a method. The field of steganography presents a number of ways to hide information in non-relevant parts of signals. For example, the S-tools toolset [7] enables users to hide information in the least significant bits of bitmap pixels. In a bitmap file, the color of each pixel is represented by a number of bits. The exact bit value represents the position in a color palette. As such, altering the least significant bits moves the color to neighboring colors on the palette, typically indistinguishable from the original color. Similar tools exist to hide information in non-relevant portions of other file types such as JPG images [8] or MP3 files [9. However, this type of steganography does not provide an inherent transport mechanism for the information. The file must be manually carried over the network by a traditional communication protocol, requiring human interaction which is not practical in our operational context.

If we could build a tool that hides information in irrelevant parts of the data used by a SCADA protocol, we could ge the convenience of a network protocol channel with the stealth of traditional steganography.

5. Modbus command and control

In order to maintain stealth, our command and control channel should not produce unexpected traffic. In that sense, it is reasonable to implement the command and control channel on the Modbus protocol which will already be present on the industrial control network. This section explains the various design choices involved in the production of the Modbus command and control channel.

5.1. General approach

In order to build a channel, we must first find a method to move bits around. The most obvious solution available is to use the least significant bits of read and write messages as a storage-based covert channel. This solution has multiple advantages. First, it is easy to push data using write messages and to pull data using read messages. This facilitates bidirectional communication. Second, the bandwidth of the channel can be arbitrarily increased by using a larger number of bits per register or just adding more registers.

Consider a scenario where a malicious program behaving as the Modbus master is using a covert command and control channel to execute remote commands via a malicious trojan on a Modbus slave. A benefit of using Modbus messages as a storage channel is that the Modbus traffic profile shows roughly the same emitter-receiver orientation as a traditional command and control channel: a small number of uploads (sending commands) and a large number of downloads (downloading results). As such, a sufficiently patient attacker would be able to fit the traffic profile of normal Modbus operations. Even more, this ability for stealth is not made at the expense of responsiveness as would be the case in a beaconing approach.

However, using this method introduces a limitation. Because a Modbus slave cannot initiate a communication, all the traffic must be managed by the master unit. This introduces an overhead for signaling between the machine acting as the channel server, behaving as a Modbus slave, and the machine acting as the channel client, behaving as a Modbus master.

5.2. Client and server design

To overcome the limitation of the unidirectional nature of Modbus communications, we decided to use a state machine design for the client and server. In order to synchronize the state machines, a few coils are used as storage channels for signaling purposes.

In order to avoid interfering with normal operation, both the holding registers and the coils used for the channel are added to the Modbus slave. Because these registers are added, we could theoretically store arbitrary values of up to 16 bits. However, in order to be stealthy, the new coils and holding registers must present information properties that resemble those of coils and holding registers used in normal operation.

The channel client handles all the communications. As such, it is not necessary for the client to communicate its state to the server. Only the server state has to be communicated to the client. The server can be in one of four states:

- Awaiting instructions;
- Reading from buffer;
- Writing to buffer;

- Completing a command.

This requires 2 bits of storage to preserve the state. Additionally, another bit of storage is used as a reading status bit to manage flow control. This bit is turned on when there is a chunk to read and turned off when the chunk has been read, signaling readiness to read the following chunk.

In addition to the three coils used to maintain state, a register is used to transfer commands from the client to the server. Each command has an integer number associated with it. When the client wants to send a command to the server, it send a WRITE_SINGLE_REGISTER packet to write the value corresponding to the command in the register. The server then processes the command and toggles the value back to the integer associated with the idle command once the task is done.

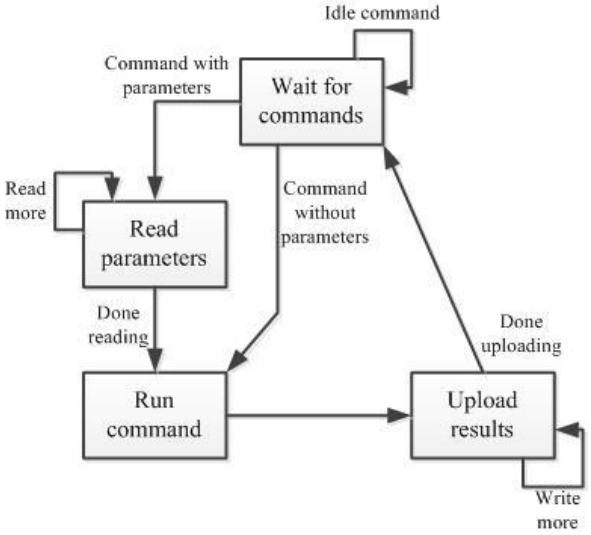

Figure 2: State machine for the server

The server state machine starts in the "Wait for commands" state. If no command is forthcoming, the machine idles. Once a command is registered, if the command does not require parameters, the machine switches to the "run command" state and executes the command. If the command requires a parameter, the machine goes to the "read parameter" state instead. The machine then loops on that state until the entire content of the parameters is read. It then moves to the "run command state". Once the command is completed, the machine is placed in the "upload results" state. The results are then chunked and stored in the least significant bits of registers until the client reads the results. Once the client has read all the chunks from the results, the machine returns to the "Wait for commands" state. The resulting state machine is illustrated in Figure 2.

The client state machine starts in the "Wait for commands" state. Once the user types in a command, if the command does not require parameters, the machines switches to the "Upload command" state and sends the command to client. Once the command is sent, the machines enters the "Wait for results" state. If the command require parameters, the machines switches to the "Upload command" state and sends the command to client. Then, the machines passes to the "Upload parameters" state and the parameters are chunked and sent to the client. Only once the client has received all the parameter chunks will the machine move to the "Wait for results" state. In the "Wait for results" state, the client polls the server to determine when the command is done. Once the server has completed the command, the machine goes into the "Download results" state and reads the results from the server. Once all the chunks of the results are read, the results are displayed and the machine returns to the "Wait for commands" state. The resulting state machine is illustrated in Figure 3.

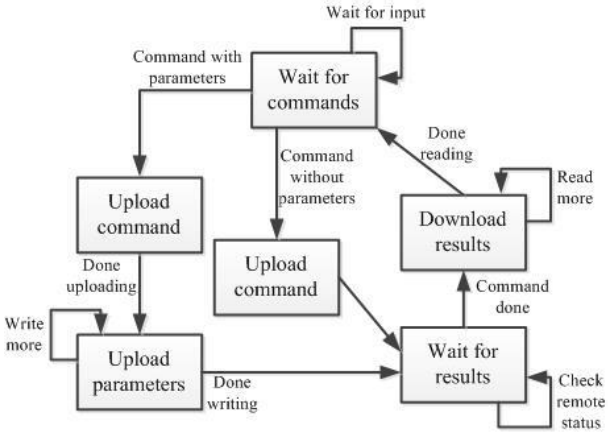

Figure 3: State machine for the client

5.3. Bandwidth available

To estimate the available bandwidth, we have to consider three parameters:

- n : Number of bits used per register;
- R : Number of registers used as storage;
- f: Frequency of messages.

The bandwidth B of the channel can be calculated from Equation 1.

$$\mathbf{B = f \cdot R \cdot n} \qquad \text{(Eq. 1)}$$

So, if we send up to 10 read or write messages per second, use 10 registers as storage and use the 8 least significant bits of each register, we obtain a channel with a theoretical bandwidth of 800 bits per second.

Obviously, the higher the proportion of the 16 bits available per register that are used as storage, the higher the bandwidth of the channel will be. However, it is important to remember that the content of typical Modbus registers have low entropy. After all, the contents of these registers are supposed to reflect the state of a physical process. If a register is supposed to represent the voltage value on a line it would be unusual for that value to swing between arbitrary discrete values outside the normal operating limits of the system. As such, much like the bitmap steganography case, there is an explicit tradeoff between the bandwidth and the stealth of the channel.

In terms of the number of registers used as storage, the use of more registers translates into more bandwidth. However, unlike the number of storage bits used, the tradeoff between stealth and bandwidth is less explicit. A great number of controllers are in charge of a large number of field devices and each of those devices may be associated with multiple registers. As such, depending on the particular control architecture used, it is possible to use a large number of registers as storage without arousing suspicion.

As for the frequency of messages, the ability to send more messages positively impacts the bandwidth. Even if the full 16 bits of the register are used, this represents only two ASCII characters per register. Even with a large number of registers, we can expect a message to require multiple packet exchanges. However, in normal Modbus communications, packet exchanges are rare. Read exchanges occur only as part of scheduled polling, with a frequency in the order of seconds for SCADA networks. Write exchanges occur only as part of deliberate actions by operators to respond to a change in the state of the physical system, which occur on a frequency in the order of minutes to hours. An attacker which is very concerned by stealth would be required to match these frequencies, severely limiting the bandwidth and diminishing the responsiveness of the command channel.

6. Performance analysis

This section presents the performance analysis of the Modbus storage-based covert channels. It starts by presenting the experiment design, then follows with the results obtained and a brief discussion on their interpretation.

6.1. Experiment design

In order to test the validity of our approach to implement a covert channel, we implemented the Modbus command and control channel described in section 4. The implementation was done in Python 2.7 with the use of the Modbus-tk library [10] to implement the Modbus communications. For the purpose of the experiment, the code was instrumented to run benchmarking tests. The instrumented code is then used to run multiple transfers of a compressed file from the server to the client. This would be the equivalent of exfiltrating a compressed data archive. This action is selected as the most suitable for the benchmarking test because a compressed file maximizes the bandwidth utilization of our channel.

Using this implementation, we want to test the bandwidth and the stealth of a number of configurations. Looking at Equation 1, we see that three parameters have an influence on the bandwidth: frequency, number of holding registers used for storage and number of bits used per register. Because delays in processing on the server can affect the effective frequency, the frequency will be fixed for the entire experiment. The number of registers and the number of bits used per register will be changed one at a time and used as control variables for our experiment.

The tests are made using VMWare. One Windows workstation image is running the client and another is running the server. Both images are on the same virtual network. While this significantly reduces the variability of the results, it represents the most typical use-case where an operator workstation is compromised through human interaction and then used to pivot onto controllers on the plant LAN.

Table 1: Results - Channel throughput experiment

Digits	**2**	**3**	**4**	**5**	**4**	**4**	**4**	**4**	**4**	**4**
Store	**3**	**3**	**3**	**3**	**1**	**2**	**5**	**7**	**9**	**12**
Throughput (Avg) (bits/s)	75.9	113.5	164.7	203.9	54.1	110.2	268.0	378.3	480.4	645.0
Std. Dev.	0.3	0.4	0.3	0.5	0.2	0.2	1.4	1.0	1.7	2.1
Runs	10	10	10	10	7	10	10	10	10	10

For each run, the time required to complete the transfer is measured and stored. We then divide the size of the transferred file by the time of completion to obtain the throughput of our covert channel. This metric presents an estimator of the bandwidth available for the attacker.

Additionally, for each run group, a packet capture is made using Wireshark. Then, the packet capture is analyzed to extract the IP payload of Modbus packets. This is done by extracting all Modbus packets using a display filter and then dumping the TCP payload via tshark. The pseudo-entropy of each individual packet's TCP payload is calculated and stored. The sampling window of a single packet was chosen for convenience. Even if this sampling size cannot be used as an indicator of the true source entropy, it is still sufficient to observe the contribution of the channel. This means that the entire Modbus content, including the Modbus headers, are included in the pseudo-entropy measurements.

Based on the measurements, we graph the distribution of individual packets' pseudo-entropy. This will enable us to compare the distribution of packet pseudo-entropy for our two control variables. Normally, as we add information to the channel, we expect more packets to contain more information and thus raise entropy. This metric represents an estimator of stealth. The more the distribution of pseudo-entropy resembles the distribution of pseudo-entropy of natural Modbus traffic with no additional information, the stealthier the channel will be.

Starting with a base configuration case where the four least significant digits are used as storage, i.e. the covert content can take the values of 0000 to 9999 in the 16 bit unsigned integer, and 3 storage registers, we alter the number of digits used or the number of storage register used. This enables us to plot the value of throughput and pseudo-entropy for both control variables and quantify the explicit trade-off between stealth and bandwidth for our channel. For each value of the independent variables, 10 file transfers are made and the average and standard deviation are calculated.

6.2. Results

The results for the channel throughput measurement experiment are presented in Table 1.

The digit row lists the number of least significant digits used for the experiment. The Store row lists the number of storage registers were used in the experiment. The *Throughput (Avg)* row lists the average throughput collected from all the runs. The *Std. Dev*. row lists the standard deviation for the experiment runs and the *Runs* row lists the number of completed runs for the experiment.

As we can see in the table, the throughput increases linearly with both the number of least significant digits used and the number of storage registers used. Figures 4 and 5 illustrate these relationships.

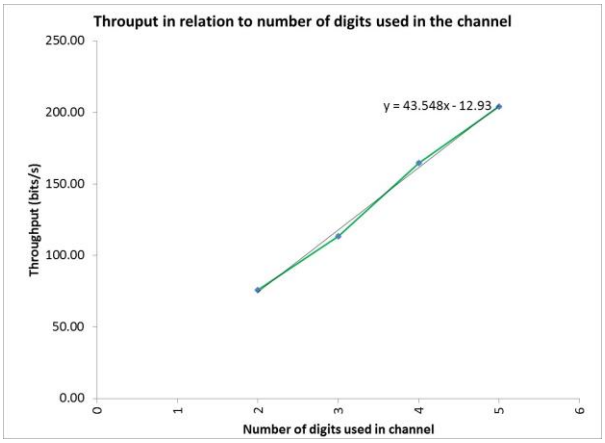

Figure 4: Throughput relative to the number of registers

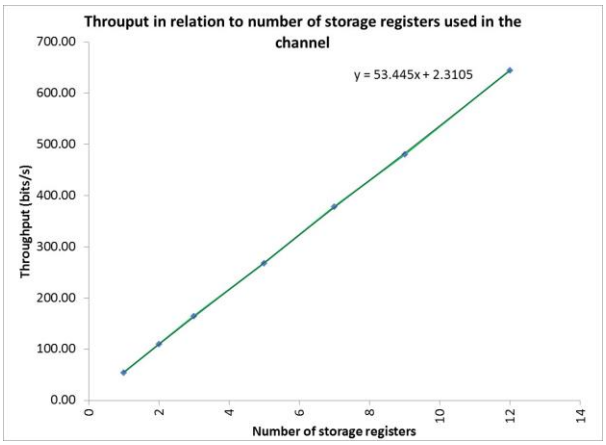

Figure 5: Throughput relative to the number of digits

Linear regression show that adding a digit adds around 43 bits per second of throughput with 3 storage registers and that adding a storage register at four digits adds around 53 bits per seconds. Each additional digit adds between 3 and 4 bits per storage channel, depending on the need to drop fractions of bits. With the current experiment configuration, this amounts to 9 to 12 bits added per read operation. As for storage registers, at 4 digits, each storage register holds 13 bits. Therefore, each time a register is added, 13 bits are added to each read operation. This slightly better performance in terms of number of bits added per modification is the likely explanation for the greater coefficient observed experimentally. As such, we feel that this experiment confirms the expected value presented in Equation 1.

The results for the channel stealth measurement experiment are presented in Figures 6 and 8, with a zoomed in display of the contribution of the channel presented in Figures 7 and 9.

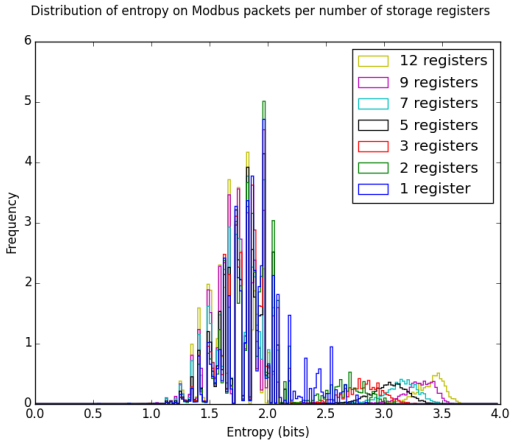

Figure 6: Pseudo-entropy per number of registers

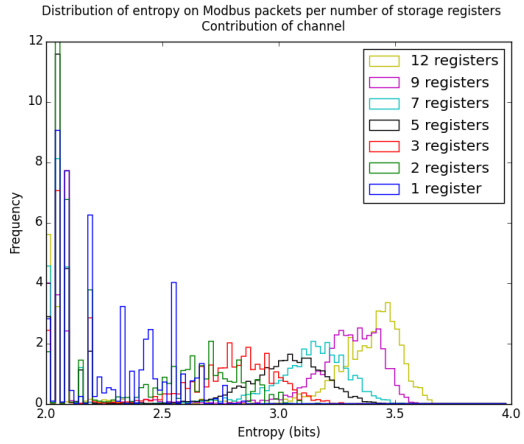

Figure 7: Zoomed-in entropy per registers

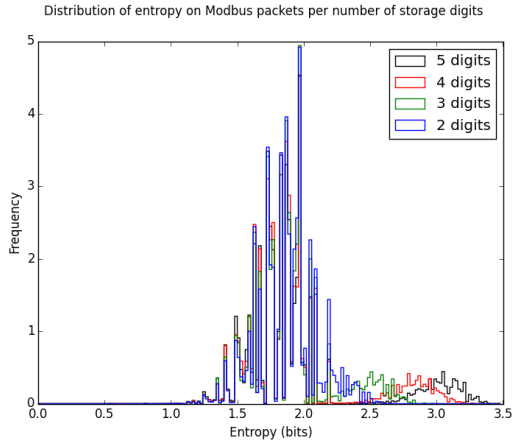

Figure 8: Pseudo-entropy per number of digits

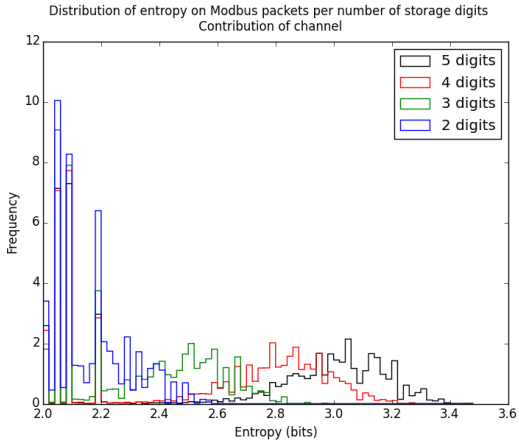

Figure 9: Zoomed-in entropy per digits

As expected, the figures show that, as information is added to the channel, a more significant number of packets have higher observed pseudo-entropy. Furthermore, the more information that is injected in the channel, whether it is through increasing the numbers of digits used or increasing the number of storage registers used, the further the contribution of the channel is from the center peak, which represents the distribution of pseudo-entropy in signaling packets. The pseudo-entropy obtained from the signaling most likely emanates from the Modbus headers, which are contained in the IP payload. For a given Modbus server, the headers are mostly fixed, which explains the relatively tight distribution excluding the channel contribution.

Normally an attacker would want to maximize his bandwidth, given stealth constraints. In our case, this would mean maximizing the number of digits used and the number of storage registers used. However, both of these actions increase the distance of the pseudo-entropy of these packets from the signaling peak. On the other hand, packets from real production systems, where the physical components add entropy to the packet, would also show some distance from the signaling peak. The exact quantity of information that is carried depends on the exact physical system used and the amount of noise present in that system. For example, the reading of the state of a line relay that very seldom triggers has low entropy while the reading of the voltage of a power line includes a lot of noise caused by natural phenomena. As such, it would be necessary for attackers concerned about stealth to characterize the exact entropy distribution of the targeted system and minimize the distance within that particular distribution.

7. Conclusion

In this paper, we presented a Modbus covert channel that can be used for remote command and control. This channel uses the least significant bits of holding regis-

ters to store and transfer information between a client and a server. While the channel adopts a client-server architecture for request, the transfer of information is bidirectional, allowing users to upload commands and parameters and receive results. This emitter-oriented approach, which is a constraint of Modbus architecture, delivers a responsive command channel for the user.

The channel can be optimized for bandwidth or stealth by tweaking the number of least significant digits and the number of holding registers used to carry information. The more digits or registers used, the faster information can be carried. However, this speed comes at the expense of stealth. This provides a range of attacker profiles for the generation of attack data sets.

Nonetheless, the Modbus channel presented in this paper does not cover all possible implementations of SCADA covert channels. For example, due to the predictable timing of polling, the investigation of a time-based channel seems profitable. Similarly, the Modbus header contains a host of fields that could be investigated for the presence of storage vulnerabilities. Also, channels for other SCADA protocols which follow the same design as Modbus, such as DNP3, could be implemented following an approach similar to the one used in this paper. Finally, once data sets for these types of channels are available, revisiting SCADA network intrusion detection work to investigate their performance against these types of channels would be worthwhile.

8. Acknowledgements

This research was funded in part by the Center for Security Science.

9. References

[1] N. Falliere, L. O. Murchu and E. Chien, "W32.Stuxnet Dossier Version 1.4," Symantec Security Response, 2011.

[2] Modbus organisation, "Modbus Application Protocol Specification v1.1b," 28 December 2006. [Online]. Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf. [Accessed 22 July 2013].

[3] S. Zander, G. Armitage and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *Communications Surveys & Tutorials,* vol. 9, no. 3, pp. 44-57, 2007.

[4] R. W. Smith and S. Knight, "Predictable design of network-based covert communication systems," in *Security and Privacy, 2008. IEEE Symposium on*, Oakland, 2008.

[5] D. Stødle, "Ping Tunnel," 5 September 2011. [Online]. Available: http://www.cs.uit.no/~daniels/PingTunnel/. [Accessed 11 May 2015].

[6] D. Kaminsky, "Black Ops of DNS," 29 July 2004. [Online]. Available: www.blackhat.com/presentations/bh-usa-04/bh-us-04-kaminsky/bh-us-04-kaminsky.ppt. [Accessed 11 May 2015].

[7] K. Magee, "CISSP – Steganography, An Introduction Using S-Tools," 14 March 2011. [Online]. Available: http://resources.infosecinstitute.com/cissp-steganography-an-introduction-using-s-tools/. [Accessed 6 May 2015].

[8] A. Westfeld, "F5—A Steganographic Algorithm High Capacity Despite Better Steganalysis," in *Information Hiding, 4th International Workshop, IHW 2001*, Pittsburgh, 2001.

[9] F. Petitcolas, "MP3stego," June 2006. [Online]. Available: http://www.petitcolas.net/steganography/mp3stego/. [Accessed 11 May 2015].

[10] L. Jean, "Python Software Foundation - modbus_tk 0.4.3," 3 November 2014. [Online]. Available: https://pypi.python.org/pypi/modbus_tk. [Accessed 26 January 2015].