Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Assessment of Hidden Channel Attacks: Targetting Modbus/TCP

Kevin Lamshöft * **Jana Dittmann** *

* *Research Group Multimedia and Security , Institute of Technical and Business Information Systems , Department of Computer Science , Otto-von-Guericke-University of Magdeburg (e-mail: name.surname@ovgu.de).*

Abstract: Recent findings in research on malware threats indicate an increasing use of information hiding techniques as a novel approach for compromising IT-Systems by using covert functions and hidden channels. Especially in the context of covert intrusion and data exfiltration, networks of Industrial Control Systems (ICS) are a valuable target for information hiding based attacks. In this paper we discuss how 18 known patterns of information hiding in networks can be applied to protocols found in ICS networks and demonstrate how information can be covertly embedded and retrieved at the example of Modbus/TCP to achieve an overall protocol-compliance by also studying the embedding capacity. Additionally we provide a first tendency of warden- compliance, if the warden has a suspicion that there is a hidden message (conspicuousness). For a practical analysis we introduce an evaluation framework based on open-source software enabling assessment and quick implementation of hidden channels in common protocols found in ICS networks. In combination with a pattern-based approach for the identification of hidden channels, we show how this framework can be utilized as a systematic approach to identify and evaluate plausible hidden channels in industrial communication protocols. From the 14 identified patterns, in this paper we use the introduced framework to implement and evaluate two exemplary timing and storage channels in Modbus/TCP. Our results show 14 protocol-compliant patterns of information hiding based attacks in the context of Industrial Control Systems as well as the necessity of more research in this particular field, especially in terms of further plausible combination of pattern, warden-compliance, detection and mitigation strategies.

Keywords: Industrial Control Systems, Cyber Physical Systems, Hidden Communication, Steganography, Information Hiding.

1. INTRODUCTION

In contrast to classical desktop malware, information hiding based malware uses unsuspicious data as plausible cover data or hidden channels which are then utilized for stealthy intrusion, installation, persistence and activation of malicious code. Malware which uses information hiding is often considered stealthy malware or obfuscated malware. This new trend is increasingly utilized by actual malware found the the wild in different areas. The findings and research of the Criminal Use of Information Hiding Initiative (CUIng) (Europol's European Cybercrime Centre EC3 (2018)), which are working on this specific topic, show the present need for action. For example StuxNet uses root kit characteristics in order to hide its intrusion into the system (Falliere et al. (2011)). StuxNet uses lnk-files in the file system as cover data, and uses in-memory code injections to keep malware code blocks persistent in the system space of the RAM. Another example is the so called SyncCrypt Ransomware which hides its malicious code in images as cover data on the target system in order to avoid detection by antivirus software (Abrams (2017)).

As the recent attack on an Indian nuclear power plant (as reported by Raza (2019) and Dragos Inc (2019)) demonstrates, targeted attacks on industrial facilities become more common. In the context of espionage, sabotage and cyber-physical attacks, adversaries have the uprising need for covert intrusion and stealthy exfiltrations. With this motivation in mind the use of information hiding techniques in network data is growing as well.

While there is research on many methods of network steganography, research on ICS-related specifics or protocols are missing. This paper addresses this issue by giving a first assessment on a common protocol found in ICS networks. In this paper, we take a first look at Modbus/TCP as plausible cover for hiding messages in networks of Industrial Control Systems (ICS). As Modbus/TCP is one the most distributed protocols for industrial communication, especially due to its broad compatibility, it is a valuable target for future attacks.

We use the extended taxonomy for network information hiding patterns by Mazurczyk et al. (2018) in order to systematically analyze the Modbus/TCP protocol in regards to plausible covert channels. We take a look on how information can be embedded and retrieved in an industrial context while at the same time estimating the resulting capacity in regards to its detectibility.

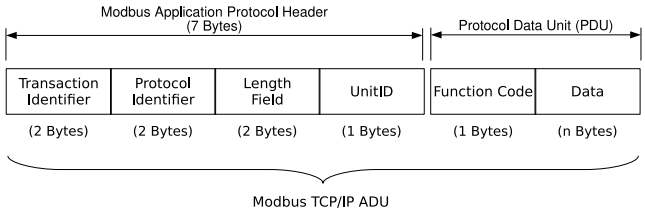

Fig. 1. Modbus/TCP Frame with Application Data Unit (ADU) and Protocol Data Unit (PDU) as specified in the Modbus/TCP Specification Modbus Organization (2006)

In the following Chapter 2 we give a brief overview on the known pattern based taxonomy of Mazurczyk et al. (2018) as well as Modbus/TCP. In Chapter 3 we describe our concept of embedding and retrieval of hidden messages for Modbus/TCP and by orientating at the known patterns, we do a theoretical cover and capacity analysis for Modbus/TCP. Chapter 4 introduces our evaluation framework which implements 6 selected covert channels in order to evaluate the plausibility and capacity of the previously identified covert channels. Chapter 5 concludes our results and gives a brief outlook on future work.

2. STATE OF THE ART

Recent research in the context of network information hiding include general considerations, such as the pattern based taxonomy of Wendzel et al. (2015) and its later extension by Mazurczyk et al. (2018). Specific methods of information hiding in the Internet Protocol are recently shown in IPv4 by Caviglione et al. (2019) as well as the application of covert channels to IPv6 in real world scenarios by Mazurczyk et al. (2019).

In this paper we use the extended pattern based taxonomy by Mazurczyk et al. (2018), which was originally introduced by Wendzel et al. (2015). The pattern based taxonomy is a unified approach for modeling and describing covert channels as part of information hiding in network communication. It is based on the analysis of over one hundred techniques, and unifies them into 18 general patterns. In the extended taxonomy there are 8 timing patterns and 6 storage patterns. With the pattern based taxonomy it is possible to describe covert channels of different protocols in a unified language as each method can be linked to one of the 18 patterns. We take this approach and reverse it by applying each pattern theoretically to Modbus/TCP in order to systematically identify plausible channels in Modbus/TCP and test them practically with our assessment framework.

As stated in the specification by the Modbus Organization (2012), Modbus is an application layer (OSI Layer 7) messaging protocol which is based on a client/server model which enables communication between devices connected on different types of buses or networks. Modbus was introduced in 1979, became quickly the defacto industry standard for serial communication in ICS networks and was later ported to TCP/IP (Modbus Organization (2006)). Modbus uses a request/reply model and specifies function codes for access to certain services of a device. The PDU (Protocol Data Unit) is independent of the underlying layers and contains a function code, which describes which action should be performed and the actual payload data. As shown in figure 1, around the PDU lays the ADU (Application Data Unit) which in the case of Modbus over TCP consists of the MBAP (Modbus Application Protocol) Header and the PDU (function code plus data). The MPAB Header contains four fields: a Transaction Identifier (Request/Response), a Protocol Identifier (static for Modbus), the length and a Unit Identifier (to address remote Slaves).

3. HIDDEN CHANNELS IN MODBUS/TCP

In this chapter we use the extended patterns by Mazurczyk et al. (2018) as a starting point to evaluate whether Modbus/TCP is usable as a plausible cover for hiding messages and which patterns can be plausibly applied. As patterns can be implemented in various ways and even be combined for more advanced methods, we focus on probable implementations to evaluate if the pattern is applicable in a plausible way. In extension to the patterns provided by Mazurczyk et al. (2018) we estimate the capacity of the pattern when using a (basic) plausible implementation. For the analysis we take a look on embedding messages by **modification** of present network data, either single data packets or flows of packets. For that we put a focus on the specific Modbus layer and leave the TCP/IP layer out of this analysis. In the following, patterns are also evaluated in terms of applicability to the kind of device and if active or passive information hiding methods can be used.

The plausibility of a covert channel can be described by two conditions: **(C1) protocol-compliance** and **(C2) warden-compliance**. A covert channel is **protocol-compliant**, when a modification of a packet or flow does not break the protocol in a way that the recipient would not receive, accept or process the packet. For **warden-compliance** we can differentiate three levels (based on probabilities): (1) the message is hidden in a way that a potential warden has no knowledge of the existence of a hidden message (inconspicuous), (2) the warden has a suspicion that there is a hidden message but can not access it and (3) the warden can identify and access but not reconstruct the hiddden message. In this paper we focus and test for protocol-compliance and give a first indication of warden-compliance, if the warden has a suspicion that there is a hidden message (resulting conspicuousness).

As Modbus differentiates between Servers and Clients, or more specific Slaves and Masters, the capabilities between those two vary as well. In the Modbus protocol only

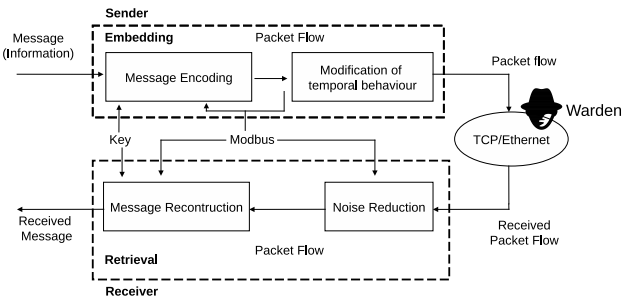

Fig. 2. Process steps for embedding and retrieval of timing channels.

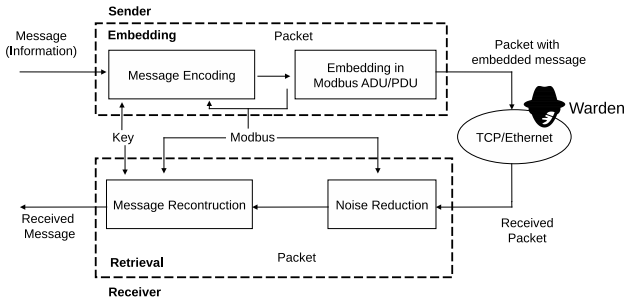

Fig. 3. Process steps for embedding and retrieval of storage channels.

clients (Masters) are able to initiate a communication. PLCs usually work mainly as Slaves, whereas HMIs and Engineering Workstations generally work as Masters to query I/Os or set parameters of PLCs. PLCs might also act as Masters when communicating with other PLCs. Therefore in the analysis we will differentiate whether only the Slave, the Master or both are able to embed or retrieve hidden information in a specific covert channel.

3.1 Embedding and Retrieval by Modification

In Network Information Hiding two categories of hidden channels are distinguished: On the one hand there are timing channels, which modulate the temporal behavior of a specific packet flow to encode a hidden message or information. On the other hand there are storage channels, which modify contents of specific messages to embed a secret message or information.

Figure 3 shows the general process of embedding and retrieval for storage channels. Before the actual message can be embedded into a specific (cover) packet, the message must be encoded with a key which is known by both the sender and the receiver. The encoding is done in such a way that a modification of the packet represents the information or a part of the message. In many cases a message is encoded over multiple packets, e.g. one modification per packet representing one bit of the message. The packet with the embedded message then gets sent over the usual communication channel or carrier to the receiver. The transmission might induce some noise, which is then reduced or removed as part of the retrieval process. With the pre-known key and knowledge about the encoding, the message can then be reconstructed. In the case of the so-called reversible information hiding, for example as shown by Caviglione et al. (2019) the covert packages would include information to enable the receiver to restore the original message as well as the hidden information. As shown in figure 2 timing channels modulate an information over the temporal behavior of a defined packet flow rather than modifying certain attributes of single packets.

3.2 Active & passive Information Hiding

Modbus/TCP can usually be found in the communication between (1) PLCs, (2) Engineering Workstations, and (3) Human-Machine-Interfaces. Therefore we consider these devices as potential sender and receiver of hidden information. As network elements like (4) Switches, (5) Hubs and (6) Firewalls obviously have access to the network traffic as well, they are considered, too.

If the embedding and retrieval solely takes place at the original sender and receiver, e.g. a PLC and an HMI, we consider this as *active information hiding* (see figure 4). If the embedding and retrieval is done solely by other parties that take only a passive part in the communication, for example switches or firewalls, we consider this as *passive information hiding*. If the embedding takes place on an end device, like a PLC or HMI and the retrieval is done by a network element, we consider this as *semi-active information hiding*. Vice-versa, embedding by a network element and retrieval by an ICS element is considered *semi-passive information hiding*.

3.3 Timing Channels

In the following we apply the 8 timing channels (T1-T8) to Modbus/TCP, test for compliance conditions C1, check which Modbus role (Master/Slave/Network element) in combination with active/passive Information Hiding (IH1-IH4) could implement the pattern, and estimate a plausible payload capacity in regards to the resulting conspicuousness.

T1. Inter-packet Times Pattern T1 alters the timing intervals between network packets of a flow (inter-arrival times) to encode a message or hidden information. As the communication in ICS networks is mainly based on the cycle times of the PLCs, the timings between each packets and packet flows are usually well defined, constant and vary only in a small delta. This suits information hiding well as the noise level is low which results in a more stable covert channel. An attacker can leverage this property by delaying certain packets of a flow in order to encode a message. The limitation is given by the timeout of the communication partner which then classifies the packet as not-received and will then send re-transmissions. Many re-transmissions are conspicuous and may lead to an detection by a warden. Depending on the timeout

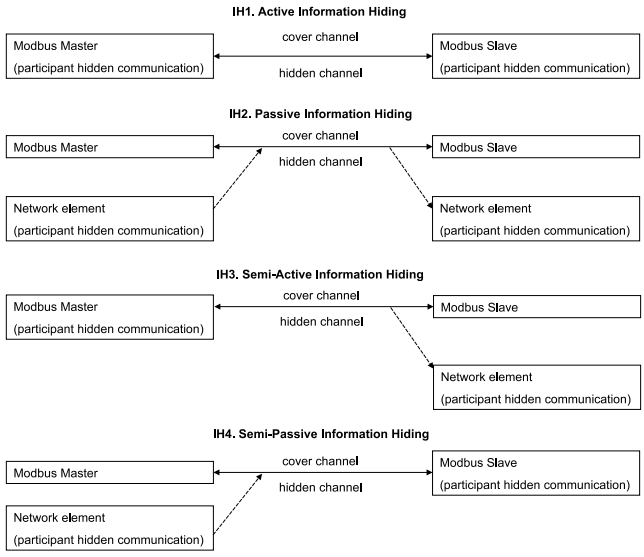

Fig. 4. Illustration: Our extension of the active/passive information hiding differentiation as published by Dittmann et al. (2005).

settings, even more symbols could be encoded by delaying packets in different timing windows. The modulation of inter-packet times can be performed by Modbus Slaves as well as Masters with the limitation that only Masters can initiate a communication. Therefore, a transmitting Slave has to "wait" for an incoming communication. The same holds true for any network devices that route or forward the traffic. The **capacity** of the channel depends on the throughput of regular packets, the encoding, and the specified timeout settings. In a simple approach, where each packet is either delayed or not the **capacity** would be 1 bit/packet.

T2. Message Timing The Message Timing pattern encodes data in the timing of message sequences within a flow, e.g. sending a command x times. For Mobdus/TCP this patterns applies only to Modbus Master devices as they initiate a communication flow. A man-in-the-middle network device would be capable as well by simply duplicating packages of a Master device. A Slave device that would change the timing of messages within a flow would violate the specification (protocol-compliance) and therefore raise suspicion (warden-compliance). A reasonable scenario would be an HMI sending an I/O read or write request two or three times instead of only once in order to encode a message. As the traffic would derive significantly from the usual traffic flow, a detection of the covert channel seems highly probable. Therefore, an application of this pattern in a real-world scenario seems unlikely.

T3. Rate/Throughput In the rate/throughput pattern, the sender alters the data rate of a packet flow that is directed to the covert receiver. Usually the data rate of flows between Modbus devices are stable over time, for example an HMI querying and setting I/Os multiple times per second. An attacker might encode a hidden message by modulating this rate by applying a multiplicand k, in the form of $encode(bit) = (x * k)packets/second$, where for example $k = 2$ could encode a binary one and $k = 1/2$ a binary zero respectively. For Modbus/TCP this is only applicable to devices that act as a Modbus Master, or take part in the communication as a network device. The **capacity** of this channels depends on the scale of the data rate which is modulated. An attacker could use packets per second as shown in the example, or use longer terms, for example packets per hour or day, which would result in lower bit rates but lower detection probability as well. Whereas in the example only one bit per flow is encoded, a higher bandwidth can technically be achieved by using more multiplicands to encode a message. As the data rate usually keeps in a well-defined window, a detection seems likely.

T4. Artificial Loss As the name suggests, the pattern of artificial loss makes use of dropping or corrupting frames to encode a message. While still applicable to Modbus devices, this pattern is especially useful for passive information hiding done by network elements. As frame drops or packet losses induce retransmissions, this pattern raises suspicion when used with a high bandwidth. An attacker would have to calculate the noise level (the probability of random retransmissions) and only slightly increase this noise level by conducting additional artificial losses in order to encode a message while preventing easy detection.

T5. Message Ordering The message ordering pattern encodes information by altering or synthesizing the packet or message order in a flow. This pattern is one of the most promising candidates as we typically have flows of packets with a specific order - for example an HMI which queries first the status of multiple coils and then the registers. An attacker can leverage this typical behavior by switching these two packets. As no data is being altered or timings shifted, this is an inconspicuous pattern when implemented in Modbus/TCP. The pattern can be implemented as an active information hiding technique by Modbus Masters as well as Slaves, by altering the order of reply packets. It can also be implemented by network elements for a passive approach or as an hybrid. The **capacity** depends on the amount of packets in a flow. By using different orders of the packet flow, multiple bits can be represented. In the simplest scenario, when the message order is either correct or altered, one bit per flow is transmitted.

T6. Retransmission Pattern T6 uses artificial retransmissions by sending previously sent or received messages of a flow. This pattern can be used by any element that takes part in the communication as it depends only on previously sent packages. Retransmissions tend to offer only a low bandwidth as they usually do not appear often in regular traffic flows and can therefore only be used over longer periods of time in order not to rise any suspicion. A simple method to encode a bit might be implemented by e.g. defining a threshold for how many retransmissions appear per minute. Using a threshold is especially useful to ensure that regular retransmissions do not disturb the covert channel.

T7. Frame Collisions In Pattern T7 a sender uses artificial frame collisions to encode a hidden information. In the case of full-duplex Ethernet switches frame collisions generally do not happen, therefore this patterns seems not to be applicable in most scenarios.

T8. Temperature The idea of the temperature pattern as published by Mazurczyk et al. (2018) is to influence the CPU temperature of a (third-party) device by e.g. sending a high amount of packets in a short term which results in a higher CPU load and by that a rise of the CPU temperature. The temperature has a direct effect on the clock skew which will differ depending on the current temperature of the CPU. By communicating with the (third-party) device the receiver can observe the clock skew and by that decode the hidden information. This pattern is an interesting scenario for ICS networks especially in order to bypass firewalls. A plausible scenario would be for example an HMI or PLC which encodes a message by sending packet bursts to a connected firewall. Devices on the other side of the firewall could then decode the message by observing differences in the clock skew in the communication with the firewall. By this method communication between separated security zones might be bypassed. While message bursts tend to be conspicuous, the actual hidden channel (the clock skew) is inconspicuous. Therefore we can not give a final verdict, as it heavily depends on the message

bursts. The **capacity** might vary as well, depending on how much the temperature (and by that the clock skew) can be modulated. In the case of a noticeable clock skew shift caused by a message burst, an encoding of 1 bit per burst can be achieved.

3.4 Storage Channels

In the following we apply the 6 storage patterns (S1-S6) and the four payload storage patterns (S7-S10) to Modbus/TCP, test for compliance conditions C1, check which Modbus role (Master/Slave/Network element) in combination with active/passive Information Hiding (IH1-IH4) could implement the pattern, and estimate a plausible payload capacity in regards to the resulting conspicuousness.

S1. Size-Modulation The size-modulation pattern modifies the size of flow meta data, e.g. the PDU size to embed a hidden message. In Modbus/TCP this pattern is applicable to the length field of the MBAP, which describes the amount of the following bytes (see figure 1). The length field values can vary between two bytes (Unit ID and Function Code) to a maximum of 253 bytes, leading to a high (theoretical) **capacity**. The modification can only take place on Masters or network elements in the request packets, as the response packet must use the same length as the request packet. A modification by a Slave is technically possible though violating the specification and by that making a detection by a warden more likely. The most plausible scenario is a small variation in the length field, e.g. to achieve odd and even numbers of bytes to encode **one bit per packet**.

S2. Sequence Modulation The sequence pattern alters the sequence of elements on the meta data. This pattern is **not** applicable in a **protocol-compliant** way as the sequence is given by the specification. One way though to use this pattern is by using a passive or hybrid **reversible** information hiding approach, where the sequence of the MBAP Header elements could be changed by a Modbus or network device - but the modified packet would then be sent to or via a (rogue) network element which decodes the hidden message and reverses the packet to its to original state before it gets forwarded to the (original) receiver (Modbus device). This would allow for a data exfiltration strategy and only violating the specification in between certain paths and not at the receiving node. As there are six fields in the Modbus ADU the **capacity** this approach would be $6! = 720$ **options** (to define an alphabet) **per packet**. Depending on where the warden is observing the communication an altered sequence would be **obtrusive**. This pattern is promising for the Modbus Security protocol (see Modbus Organization (2018)) though, as this extension uses TLS to encrypt the packet which would obfuscate the modified sequence for any warden who does not have the secret keys.

S3. Add Redundancy The redundancy pattern adds additional, unused information to the metadata. Due to the limited options available in Modbus, this pattern is **not directly applicable**. A variation of the original pattern is plausible though: Modbus has different function codes for accessing either only one single coil/register or multiple coils/registers at once. Given the scenario, that a Modbus Master sends a request to access a specific register, an adversary could use the function code to access multiple I/Os but instead specifying only one register to be accessed. This leads to same result as using the function code for single access but uses a different function code. This circumstance can be used to encode a hidden message. The resulting **capacity** is **one bit per request**. This method is only suitable for active information hiding with a Modbus Master sending a request, as a response with a different function code would not be processed by the receiver (**not protocol-compliant**). A reversible variation though could be implemented in a passive approach where two network elements would change the function code in the communication between those devices and reverse it back before forwarding the packet to the original recipient.

S4. Random Value As there are no random values used in the Modbus specification, this pattern is **not applicable** to Modbus/TCP.

S5. Value Modulation The value modulation pattern modifies one of n values that a metadata element can contain. This pattern has two subpatterns: Case and Least Significant Bit. As Modbus does not have any literals and uses only 16 bit integers, case variation is **not applicable** and LSB modification of element of the MBAP header would break (or alter significantly) the functionality of the original packet.

S6. Reserved/Unused This pattern uses reserved or unused fields of the metadata to embed a hidden message. In Modbus/TCP this pattern can be applied to the Unit ID field in the MBAP header, which is only used when remote Slaves are addressed that are connected to the target devices. To our knowledge this feature is only rarely used but takes up one byte in the MPAB header. Therefore, if used by an adversary for information hiding the **capacity** is **8 bit per packet**. As many devices do not use or even ignore this field, it is applicable for Masters, Slaves, network elements and in active, passive and hybrid scenarios. If a warden is not directly scanning for those variations (for example in scenario where the unit id is used from time to time or only between specific devices) the channel is, as it is **protocol-compliant**, **inconspicuous**.

S7. Payload Field Size Modulation The Payload Field Size Modulation Pattern is a derivation of Storage Pattern S1, as the modification of the length field in the metadata is used to modulate the payload field size. As described before the payload can be extended to a maximum of 253 bytes, therefore leading to a **high capacity** but **conspicuousness** as well. Though, when used for example to add one additional byte per packet, this methods would still have a **capacity** of **8 bit per packet** and not being that conspicuous. This pattern is applicable mainly to Masters and network elements in active, passive and hybrid methods (see Pattern S1).

S8. User-data Corruption In the user-data corruption pattern an adversary inserts (blindly) covert data into the payload. For Modbus/TCP this would result in wrong values for reading and writing coils and registers. As this has a direct influence on the physical processes controlled by the Modbus devices this approach is **obtrusive**. When

Table 1. Overview of our findings regarding the applicability of the network information hiding storage and timing patterns (*simplified to T:Timing, S:Storage*) of Mazurczyk et al. (2018) to Modbus/TCP. The *Master*, *Slave* and *network* (element) column indicate whether the pattern is plausible applicable (✓) to the specific type of device or not (✗). *Active*, *Passive* and *Hybrid* (○ indicates semi-active) indicate the applicability of the information hiding approach. The *capacity* is a rough estimation based on the findings in chapter 3. The *conspicuousness* (tendency of warden-compliance) is estimated on a binary scale (high/low). Four practical tested patterns are marked in bold.

Patterns by Mazurczyk et al. (2018)		Our findings for Modbus/TCP							
ID	Pattern	Master	Slave	Network	Active	Passive	Hybrid	Capacity	Conspic.
T1	**Inter-Packet Times**	✓	✓	✓	✓	✓	✓	**1 bit/packet**	**low**
T2	Message Timing	✓	✗	✗	✓	✗	○	n bit/flow	high
T3	Rate/Throughput	✓	✗	✗	✓	✗	○	n bit/flow	high
T4	**Artificial Loss**	✓	✓	✓	✓	✗	✓	**1 bit/packet**	**high**
T5	Message Ordering	✓	✓	✓	✓	✓	✓	1 bit/flow	low
T6	Retransmission	✓	✓	✓	✓	✓	✓	1 bit/time unit	high
T7	Frame Collisions	✗	✗	✗	✗	✗	✗	✗	✗
T8	Temperature	✓	✓	✓	✓	✓	✓	to be tested	to be tested
S1	Size Modulation	✓	✓	✓	✗	✓	✓	8 bit/packet	low
S2	Sequence Modulation	✓	✗	✓	✓	✓	✓	720 options/packet	high
S3	Add Redundancy	✓	✗	✓	✓	✓	✗	1 bit/packet	low
S4	Random Value	✗	✗	✗	✗	✗	✗	✗	✗
S5	Value Modulation	✗	✗	✗	✗	✗	✗	✗	✗
S6	**Reserved/Unused**	✓	✓	✓	✓	✓	✓	**8 bit/packet**	**low**
S7	Payload File Size Modulation	✓	✗	✓	✓	✓	✓	8 bit/packet	low
S8	User-data Corruption	✓	✓	✓	✓	✓	✓	16 bit/packet	high
S9	Modify Redundancy	✗	✗	✗	✗	✗	✗	✗	✗
S10	**Value Modulation & Reserved/Unused**	✗	✓	✓	✓	✓	✓	**7 bit/packet**	**low**

used in a very low frequency though and for example for only one specific sensor reading (or similar) a warden might not detect this corruption as a cover for a hidden message, as the value might have been corrupted by the sensor itself. Therefore, this pattern can be implemented by all devices in active, passive and hybrid scenarios. The **capacity** depends on the payload length. A plausible corruption of one register would lead to a **capacity** of **16 bit/packet**.

S9. Modify Redundancy The modify redundancy pattern for the payload uses compression to free space which is then used to embed a hidden message. As Modbus uses 16 bit integers for the payload, this is pattern **not applicable**.

S10. User-data Value Modulation and Reserved/Unused This pattern modifies the payload in a way that the interpretation of the data does not differ significantly from the original packet, e.g. by altering the least significant bits or hiding data in unused or reserved bits. In Modbus/TCP this pattern can be implemented in multiple ways. Depending on the encoding, a modulation of the least significant bit of register values can encode a hidden message while leading to only minor variations in the values, e.g. manipulating the last digit of a sensor reading. This would lead to a **capacity** of **1 bit per packet** and applies to Modbus Slaves for active information hiding as well as network elements as part of passive or hybrid embedding method. Another way of applying this pattern is to use unused fields or bits in the payload. As Modbus is byte-oriented there are several scenarios, in which unused bits of a byte are zero-filled. For example, if a Master is querying a single coil or discrete input of a Slave, only one bit of the payload byte is used - the other seven bits can be used by an attacker to embed a hidden message, resulting in a **capacity** of **7 bits per packet**. The embedding can take place on a Slave or network elements.

4. PRACTICAL FRAMEWORK

In the previous chapter we used the pattern based approach to identify plausible covert channels in Modbus/TCP on a theoretical basis. As these are considerations based on the pattern based approach and the Modbus specification, a practical evaluation is needed to validate plausibility. To evaluate plausibility in regards to protocol- and warden compliance as well as the interplay between capacity and conspicuousness, we developed an evaluation framework called eFrameSS (Evaluation Framework for Stealth Scenarios). The idea is to have one framework which consists of a set of integrated tools and is designed in a modular way to support all patterns as well as the possibility to go beyond Modbus and easily implement and test other protocols in the same fashion.

To achieve this design goal we decided to use passive information hiding by using a man-in-the-middle like approach. Active information hiding would need the manipulation of the communicating (end-) devices, like PLCs and HMIs. This is especially hard to achieve when using proprietary, closed-sourced devices. eFrameSS makes use of a combination of iptables rules, the Linux netfilter(-queue) and the scapy packet manipulation framework (Biondi (2019)) to manipulate packets on the fly. It can be used on any Linux

based machine and can be adapted for network elements like firewalls and switches, as well as Microsoft Windows based machines. This gives the opportunity to run the framework on software-based PLCs, like OpenPLC (Alves (2019)), and common HMIs or to use additional hardware (e.g. a Raspberry Pi with two network interfaces) as a man-in-the-middle device to embed or retrieve hidden messages. This approach allows for extensive yet fast prototyping and testing of protocols as the packets can be directly manipulated on the fly while the originating devices do not need to be tampered. eFrameSS uses iptables on target devices to manipulate the communication flows within a devices. Packets of local processes (e.g. OpenPLC), which go through the output chain and are to be sent to other devices, are redirected into the netfilter-queue of the linux kernel by using iptable rules. The eFrameSS core, which is based on scapy accesses this queue and processes each packet. Each processed packet can be either accepted or dropped. This procedure is similar to the functionality of firewalls. eFrameSS parses the packets and, depending on the used pattern, modifies certain attributes of the packet to embed a hidden message. The packet with the embedded message then gets forwarded to the post-routing chain and after that gets sent via the regular network interface. At the receiving node the process is similar. Incoming packets from the input chain get redirected into the framework, where the extraction takes place. After the extraction the packet gets passed to the local process. This procedure comes especially handy when implementing reversible approaches.

Due to time and space constraints we choose 4 out of the 14 protocol-compliant patterns (two timing and two storage patterns) to implement and test. In the following we evaluate the protocol-compliance, the capacity and conspicuousness of certain implementations for these patterns.

4.1 Test Setup

In the test setup we use two Raspberry Pi 3B+, where one acts as a Modbus Master (Client) and the other as a Modbus Slave (Server). The Modbus Slave is implemented by using OpenPLC, an open-source PLC software. In this case, OpenPLC is running a simple project with each one coil, discrete input and holding register. The Modbus Master on the other Raspberry Pi is simulating an HMI by reading every second available I/Os, and after getting the responses of the Slave, writing arbitrary data on the outputs. To implement the HMI we use pymodbus, an open-source Python library for Modbus/TCP communication (Riptide (2019)). The eFrameSS framework runs directly on the Raspberry Pis, therefore simulating an infected PLC and HMI which are communicating with each other. Depending on the used pattern the HMI and PLC act both as embedding and receiving nodes. The Raspberry Pis are connected via Ethernet cables over a Linksys LRT214 switch.

4.2 First Results regarding Modbus/TCP

From the 14 theoretically identified protocol-conform pattern, we choose four patterns to be implemented and tested practically for protocol-compliance in our framework: (1) Interpacket Times T1, (2) Artificial Loss T4, (3) Reserved/Unused S6 and (4) User-data Value Modulation & Reserved/Unused S10.

T1: Interpacket Times To implement the Interpacket Times pattern we define two types of Modbus packets (based on the function code) which are artificially delayed to encode a message. For this implementation we use the ReadCoils request and the WriteSingleCoil requests of the Modbus Master to encode one bit per packet. The Master sends the requests every 1000ms, in order to encode a binary zero eFrameSS delays the ReadCoil requests by 250ms, to encode a binary one WriteSingleCoil requests are delayed. By using two different types, the start and end of a message are more easy to detect and therefore provide more robustness. As the modification takes place on the request the timeout settings of the Slave to do not affect this channel and therefore are within the **protocol-compliance**. This simple technique provides a **capacity** of **1 bit/packet** while being rather **inconspicuous** (depending on the delay). To decrease the conspicuousness/detectibility the information hider has to set the delay as low as possible while staying above usual fluctuations in the interpacket times.

T4: Artificial Loss To implement the Artificial Loss pattern we use the same procedure as in the Interpacket Times pattern, but drop the frame instead of delaying it. The **capacity** remains the same (**1 bit/packet**), but as regular packet losses are unlikely in our test setup, this method is **obtrusive** and easy to detect. Though, packet losses might be considered to be caused by an issue with the hardware or software rather than suspecting an hidden channel.

S6: Reserved/Unused (UnitID) In a first implementation we use the UnitID modulation approach (S6: Reserved/Unused) as it is easy to implement, has a high capacity while (depending on the scenario) not being highly conspicuous. To encode a message we use four UnitIDs that are not used in the usual communication flow (default is $0x00$): The first UnitID ($0x01$) is used to mark the beginning of message, the second one ($0x02$) is used to encode a binary zero, the third one to encode a binary one ($0x03$) and the fourth UnitID ($0x04$) is used to mark the end of the message. The actual message is encoded as a bitstream and than embedded bit-wise per packet (1 bit/packet). To increase robustness we added a cyclic redundancy check (CRC-8) to the message at the end. We tested two embedding positions: The first uses requests to read coils (of a Modbus Master) for embedding, the other one uses the responses of the Modbus Slave. With these two positions we can simulate infiltration, exfiltration and command-and-control scenarios. In our test setup the modulation of UnitIDs does not change the interpretation of the data and is processed in a regular way by OpenPLC and pymodbus - therefore fulfilling the **protocol-compliance** criteria. Further tests with proprietary hardware will be conducted in the future. The **capacity** of this implementation is **1 bit/packet**, though by adding the CRC each message needs eight additional packets. In our test setup this pattern is **conspicuous** and easily detectable as the UnitID is regularly not used. In a setup where UnitIDs are used though the modulation is

harder to detect and needs understanding of the regular communication partners.

S10: User-data Value Modulation & Reserved/Unused We implement the S10 pattern by using unused bits in the responses of ReadCoils requests. When the PLC is responding to a request of the HMI regarding the status of one single coil only one bit of the payload byte is used. This gives the option to use the remaining seven bits to encode a message. In our tests these seven bits are not interpreted by the HMI and OpenPLC. Even in wireshark the modification is only visible when inspecting the bitstream. Therefore, this method is **inconspicuous**. Though violating the specification with this modification, the packet gets interpreted as if it was not modified. Therefore this method is still considered **protocol-compliant**. The method can be applied to the Modbus functions "Read Discrete Inputs" and "Read Coils". The capacity depends on the number of queried coils or inputs as one byte can hold up for up to eight coils or discrete inputs. The capacity can be described as: $8 - [(\text{number of I/Os}) \bmod 8]$ in bits per packet. In case of one queried coil/input this means the **capacity** is **7 bits/packet**.

5. CONCLUSION & FUTURE WORK

In this paper we applied the pattern-based taxonomy for network information hiding by Mazurczyk et al. (2018) to Modbus/TCP. This allows for a systematic and comprehensive approach to identify potential plausible hidden channels. We analyzed those channels in regards to their applicability to Modbus and network devices, and did an estimation on possible capacities in regards to conspicuousness. Our findings are that 14 out 18 patterns are plausibly applicable and offer various options for an adversary to embed and retrieve hidden information. We introduced a modular evaluation framework which allows for comprehensive yet fast prototyping and testing of network hiding patterns. By its modular design it can be adapted to other network protocols as well. In our evaluation we implemented 4 of the 14 theoretical protocol-compliant patterns and tested them for practical protocol-compliance, capacity and conspicuousness in a testing environment. Our four selected test cases show that these patterns can be successfully applied in a protocol-compliant way. In respect to warden-compliance two patterns are probably difficult to detect: (1) Interpacket Times T1 and (4) User-data Value Modulation & Reserved/Unused S10, whereas two patterns are conspicuous and easily detectable: (2) Artificial Loss T4 and (3) Reserved/Unused S6. More research is required to further test alternate implementations of those patterns and especially combining them to either achieve more capacity or reduce detection probability. Furthermore, tests regarding warden-compliance are required to fully evaluate applicability and consequences in a real-world scenario. Still, our results show the general applicability and potential of hidden communication in Modbus/TCP and demonstrate it as a viable threat for networks of Industrial Control Systems.

ACKNOWLEDGEMENTS

The presented work is funded by the German Federal Ministry of Economic Affairs and Energy (BMWi, project no. 1501589A) in the framework of the German reactor safety research program. The authors thank all project partners and reviewers for their helpful comments.

REFERENCES

Europol's European Cybercrime Centre EC3 (2018). Criminal use of information hiding cuing initative. https://cuing.org/.

Abrams, L. (2017). Synccrypt ransomware hides inside jpg files, appends .kk extension. https://www.bleepingcomputer.com/news/security/synccrypt-ransomware-hides-inside-jpg-files-appends-kk-extension/.

Alves, T. (2019). Openplc project. https://www.openplcproject.com/.

Biondi, P. (2019). Scapy - packet crafting for python2 and python3. https://scapy.net/.

Caviglione, L., Mazurczyk, W., Szary, P., and Wendzel, S. (2019). Towards reversible storage network covert channels. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 69. ACM.

Dittmann, J., Hesse, D., and Hillert, R. (2005). Steganography and steganalysis in voice-over ip scenarios: operational aspects and first experiences with a new steganalysis tool set. In *Security, Steganography, and Watermarking of Multimedia Contents VII*, volume 5681, 607–618. International Society for Optics and Photonics.

Dragos Inc (2019). Assessment of reported malware infection at nuclear facility. https://dragos.com/blog/industry-news/assessment-of-reported-malware-infection-at-nuclear-facility/.

Falliere, N., Murchu, L.O., and Chien, E. (2011). W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6), 29.

Mazurczyk, W., Powójski, K., and Caviglione, L. (2019). Ipv6 Covert channels in the wild. In *Proceedings of the Third Central European Cybersecurity Conference*, CECC 2019, 10:1–10:6. ACM, New York, NY, USA. doi:10.1145/3360664.3360674. URL `http://doi.acm.org/10.1145/3360664.3360674`.

Mazurczyk, W., Wendzel, S., and Cabaj, K. (2018). Towards deriving insights into data hiding methods using pattern-based approach. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 10. ACM.

Modbus Organization (2006). Modbus messaging on tcp/ip implementation guide v1.0b. http://modbus.org/specs.php.

Modbus Organization (2012). Modbus protocol specification v1.1b3. http://modbus.org/specs.php.

Modbus Organization (2018). Modbus/tcp security protocol specification. http://modbus.org/specs.php.

Raza, A. (2019). Indian nuclear power plant confirmed to be infected with malware. https://news.beincrypto.com/.

Riptide (2019). pymodbus - a full modbus protocol written in python. https://github.com/riptideio/pymodbus.

Wendzel, S., Zander, S., Fechner, B., and Herdin, C. (2015). Pattern-based survey and categorization of network covert channel techniques. *ACM Computing Surveys (CSUR)*, 47(3), 50.