Side Channel Attacks over Encrypted TCP/IP Modbus Reveal Functionality Leaks

Nikolaos Tsalis, George Stergiopoulos, Evangelos Bitsikas,
Dimitris Gritzalis and Theodore Apostolopoulos
Information Security & Critical Infrastructure Protection (INFOSEC) Laboratory,
Dept. of Informatics, Athens University of Economics and Business, Greece

Keywords: Modbus, Protocol, Side, Channel, Attack, Decision, Tree, Sequence, Unpadded, Cryptography, Scada, TCP.

Abstract: With HMI systems becoming increasingly connected with the internet, more and more critical infrastructures are starting to query PLC/RTU units through the Web through MODBUS ports. Commands sent from such interfaces are inevitably exposed to potential attacks even if encryption measures are in place. During the last decade, side channels have been widely exploited, focusing mostly on information disclosure. In this paper, we show that despite encryption, targeted side channel attacks on encrypted packets may lead to information disclosure of functionality over encrypted TCP/IP running MODBUS RTU protocol. Specifically, we found that any web interface that implements unpadded encryption with specific block cipher modes (e.g. CFB, GCM, OFB and CTR modes) or most stream ciphers (e.g. RC4) to send MODBUS functions over TCP/IP is subject to differential packet size attacks. A major cause for this attack is the very small number of potential MODBUS commands and differences in packet sizes, which leads to distinctions in traffic. To support the importance of these findings, we conducted research on Shodan looking for relevant devices with open MODBUS ports over TCP/IP that utilize encrypted web traffic. The result was that a significant amount of web interfaces communicate with MODBUS ports and many use unpadded ciphers and SSL with AES-GCM or RC4. We also implemented a PoC on a simulated architecture to validate our attack models.

1 INTRODUCTION

Human-machine interfaces (HMIs) control devices over SCADA networks and allow remote administration of systems in various critical infrastructures. During the last decade, HMI systems are becoming increasingly connected with the internet. Today, these interfaces are using the Web to provide remote administration of systems not usually located within a predefined control area. However, unlike traditional isolated SCADA networks, exposing the HMI to the internet and allowing for a browser-server component structure, inevitably exposes information transferred over the internet to potential attacks. To mitigate such risks, deployed systems sometimes utilize encryption over TCP/IP when communicating MODBUS commands to terminal units (Modbus TCP, 2006).

A quick check in shodan.io (Matherly, 2009) reveals many open Modbus ports currently utilized for multiple reasons, sometimes for controlling industrial control systems (ICS) and respective remote units. Some of these interfaces provide web platforms to exchange protocol commands.

During the last decade, side channels have been widely exploited, focusing mostly on information disclosure (Comey, 2014). A side channel attack is an attack that aims to retrieve secret information using attributes and factors outside normal computation and algorithmic attacks; such as timing information, power analysis and electromagnetic leaks or differential size analysis (McLaughlin et al., 2016). Concerning side channel attacks on encrypted traffic, previous research was successful in exploiting characteristics in various cases like SSH (Song, 2001), VoIP (Wright et al., 2008), identifying web pages through their loading sizes (Danezis, 2009) (Sun et al., 2002) etc. Researchers in (Chen et al., 2010) showed how to exploit fundamental characteristics of web applications like stateful communication, low entropy input and traffic distinctions to leak information concerning sensitive personal information of users.

Tsalis, N., Stergiopoulos, G., Bitsikas, E., Gritzalis, D. and Apostolopoulos, T.
Side Channel Attacks over Encrypted TCP/IP Modbus Reveal Functionality Leaks.
DOI: 10.5220/0006832700530063
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, pages 53-63
ISBN: 978-989-758-319-3

The Modbus protocol is a real-time industrial communication protocol often used to connect supervisory systems with remote terminal units and data acquisition systems (NI, 2017) (Modbus TCP, 2006). Its master/slave concept is simple without excessive overhead. Since Modbus lacks of authentication and encryption external security implementations, e.g., TCP/IP encryption, IPsec VPN are often used to protect communication (Knapp and Langill, 2014).

1.1 Contribution

In this paper, we present similar findings on HMIs that utilize TCP/IP encryption over MODBUS to control remote units in SCADA systems. In some modern encryption algorithms, differential packet size analysis attacks can reliably leak information about the functionality of the SCADA system down to small granularity levels, like leaking specific functionality/ task executions through detected sequences of MODBUS commands. The contributions of this paper are the following:

- Modbus command leaks and SCADA functionality extraction by exploiting packet sizes between web HMIs and remote units. We present in detail how analysis of requests and responses can exploit packet sizes to leak information about the underlying functionality in web HMIs that implement unpadded encryption with specific block cipher modes (e.g. CFB, GCM, OFB and CTR modes) or stream ciphers (e.g. RC4). We also show that HMI like these are currently interact with MODBUS ports over encrypted TCP/IP.
- Prediction Models for MODBUS commands and ICS functionality. We model MODBUS functions according to discriminatory characteristics and develop two worst-case scenario prediction models capable of extracting task/routine functionality in infrastructures according to sequences of packets sent and received. Notice that presented models assume worst-case scenarios: All probable collisions between packet-sizes are taken for granted. In real life implementations, detection is even better than the one presented in our generalized models since industrial implementations use different number of registers/coils in functions.
- Analysis and attack mitigation solutions. We present and analysis of potential solutions and implementations to avoid the presented attacks.

Section II of this paper presents related work on the subject and compares it with our research and experiments. Section III describes how we model MODBUS functions used in SCADA systems into mathematical formulas for discriminating them based on classification of packet request and response sizes. Section IV describes the prediction models (general and specific) constructed by the aforementioned formulas; the main contribution of our paper upon which we based our implementation and tests. Section V briefly presents an implementation executed to test the prediction models while Section VI concludes and proposes mitigation techniques.

2 RELATED WORK

Side Channels attacks can be broadly defined as attacks on systems using information gathered from unintended output channels (Zhou and Feng, 2005). Kocher's paper back in 1996 was one of the first publications to present that that non-constant execution of encryption algorithms can leak information about the keys (Comey, 2014).

2.1 Side Channel Attacks on Encrypted Traffic

Packet timing and sizes are significant factors contributing to the execution of a side-channel attack in encrypted communications. It has been shown that a network eavesdroppers may be able to break cryptographic schemes or infer keystrokes in SSH (Song, 2001). Concerning side channel attacks on encrypted traffic, other research was very successful in exploiting characteristics in various cases like VoIP (Wright et al., 2008), identifying web pages through their loading sizes (Danezis, 2009) (Sun et al., 2002). Brumley-Boneh (Beresford, 2011) illustrate a timing attack against OpenSSL to extract the secret RSA keys.

Encrypted web communications share similar issues. Research from Microsoft and Indiana's University (Chen et al., 2010) shows that attackers fingerprinting web pages to collect sensitive properties can be later used as prior knowledge for side channel attacks on packet sizes not only to identify visited web pages, but also to determine user input data. Moreover, anonymity in encrypted implementations, such as TOR, is questioned when side-channel attacks are carried out, as presented in (Acromag, 2005) and Danezis (Danezis, 2009) in respective research papers.

Our work is motivated by these researches, but it differs sufficiently. Even though, we share a common techniques with the aforementioned papers, the application of our attacks and the methods of fingerprinting used are based on industrial systems and protocols; specifically on MODBUS. In addition, our focus

on the cryptographic schemes is slightly different, which, once more, depends on the MODBUS protocol and its properties.

2.2 Side Channel Attacks on ICS

ICS systems are affected by similar threat vectors as with normal IT systems. Software errors, bugs, malware and relevant cyber-attacks affect ICS systems (Kaspersky, 2014). The Stuxnet (Langner, 2011) event, Flame (Munro, 2012) and Idaho's demonstration of a cyber-attack gaining control of physical components of the electric grid, are such examples (Meserve, 2007).

As stated in (McLaughlin et al., 2016), ICSs have specific types of vulnerabilities, such as the use of micro-based controllers, the adoption of communication standards and protocols and the complex distributed network architecture. Moreover, types of attacks on ICS are wide and they can be broken down into specific layers, namely the process layer, the network, the software, the firmware and the hardware layer (McLaughlin et al., 2016).

Concerning network layer attacks in ICS, which is the scope of this paper, vulnerabilities can manifest in multiple ways; most of which are similar threat vectors with traditional IT systems. A PLC protocol, ISO -TSAP was found vulnerable to replay attacks due to lack of proper session handling (Beresford, 2011). MODBUS implementation rarely use any form of encryption. Instead, they deliver commands through unencrypted channels. Some attempts to encrypt MODBUS traffic involve Modbus TCP/IP (also Modbus-TCP), which is simply the Modbus RTU protocol with a TCP interface that runs on Ethernet (Acromag, 2005). These implementation involves encrypted TCP/IP traffic that transfers MODBUS commands to SCADA systems. DCS and SCADA server software is, also, often out of date or misconfigured and hence can be exploited (Nan et al., 2012).

2.3 Non-padded Cryptographic Schemes

Symmetric encryption utilizes padding for block ciphers, since blocks need to be multiples of specific block sizes. Examples range from older triple-DES up to AES-CBC and relevant encryption schemes. There are, however, modes that do not require padding due to effectively using block ciphers as stream ciphers. For instance, CFB, GCM, OFB and CTR modes, used by AES, do not require any special measures to handle messages whose lengths are not multiples of the block size. These modes work by XOR-ing the plaintext with the output of the block cipher. The last partial block of plaintext is XOR-ed with the first few bytes of the last keystream block, producing a final ciphertext block that is the same size as the final partial plaintext block. In addition, RC4 (which is still widely used despite being reported as potentially vulnerable under various attacks shares the same qualities with the modes above, which means that the size of the input is equal to the size of the output. These characteristics make the above mentioned schemes suitable (i) for applications that require identical sizes in plaintext and encrypted data, and (ii) for applications that transmit data in streaming form where it is inconvenient to add padding bytes (Stallings, 2016).

However, these attributes also jeopardize the security and the privacy of the encrypted communication. For example, RC4, an algorithm used extensively in web-based applications (Chen et al., 2010), can cause a leakage based on the length of the encrypted packets, considering it vulnerable to side-channel attacks (Imperva, 2015).

2.4 Existing Web HMIs with Encrypted TCP MODBUS Ports

To support our arguments regarding the significance of leaking industrial functionality over encrypted TCP/ IP MODBUS executions, we opted to search for MODBUS protocol systems (Zhou and Feng, 2005) that may utilize unpadded encryption between a web interface (HMI) and relevant MODBUS ports.

Shodan is a search engine that lets users search and identify devices and systems connected to the internet (Matherly, 2009). Shodan collects data mostly on web servers (HTTP/HTTPS-port 80, 8080, 443, 8443), as well as FTP (port 21), SSH (port 22), Telnet (port 23), SNMP (port 161), SIP (port 5060), RTSP (port 554). Using Shodan, we mapped ICS devices that actually listen to the Modbus port 502 for MODBUS commands. In addition, we were able to identify and classify the type of encryption-schemes used by these devices for protection of data sent to them over the internet.

Extensive research showed some interesting results: Out of hundreds detected, approximately 50% of machines listening for MODBUS commands used web interfaces requiring usernames and passwords without encryption; listening on HTTP ports. 27% used SSH with a mixed cipher suite including block and stream ciphers, 16% used SSL/TLS encryption and 7% used other encryption schemes such as VPN. Moreover, most of the SSH's ciphers were AES-CBC, RC4, AES-GCM, 3DES-CBC which are all unpadded

versions (Brown, 2007). Also, SSL/TLS utilized AES-GCM instead of AES-CBC in most cases also unpadded. This information was gathered by examining the connection properties and the required certificates between provided interfaces, web ports 80/8080, and relevant MODBUS ports.

3 MODELING MODBUS

In this section we present an analysis of MODBUS packet sizes in requests and their responses. A general mathematical formula is calculated for use with prediction models in the following Sections.

Worst Case Scenario. Notice that presented models assume worst-case scenarios: All probable collisions between packet-sizes are taken for granted. In real life implementations, detection is even better than the one presented in our generalized models since industrial implementations use different number of registers/coils in functions.

3.1 Modbus Functions

The Modbus protocol utilizes eight different kind of functions. Functions are direct instructions to the PLCs. Each function is a different set of instructions; in deployed ICS, their execution frequencies and payload properties depend on industrial production policies (Modbus, 2017) (Modbus, F.A.Q., About the Protocol, 2017).

Table 1: Modbus functions and codes.

FCODE	FUNCTION
01	Read coil status
02	Read input status
03	Read holding registers
04	Read input registers
05	Write single coil
06	Write single register
15	Write multiple coils
16	Write multiple registers

3.2 Discriminating Functions by Sizes

Modbus uses the master/slave scheme. Each request is accompanied by a response for each function. The presented attacks need to capture both to analyse the traffic properly.

The discrimination process starts by defining the payload to analyse in requests and responses. The payload must be the same for both types. So, we begin by decapsulating the headers, the CRC and the slave address, leaving only important information for the main payload-PDU (Modbus, 2017). This part consists of the function code and the data field. The data field comprises the addresses of the registers/inputs/coils, the actual data and other instrumental fields. By default, MODBUS functions form payload in a very specific way. Since payload sizes differ, differential analysis on sequences captured allows pattern detection due to low entropy.

Notice that Modbus TCP/IP does not use FCS like Modbus RTU. TCP has the responsibility of delivering the packets to the target unspoiled. This means that checksum is generated at lower layers, not the application layer (Modbus, 2006).

For each function we provide an analysis of its payload. Each presented type will be used later on in decision trees for traffic discrimination. We should note here that variable x used to model function below, represents the registers and, indirectly, the coils. Input is always positive integer.

3.2.1 FC5 Write Single Coil and FC6 Write Single Register

The payload of these functions has always the same number of bytes. Every FC5 request has 5 bytes and gets 5 bytes response. So, if we capture $\rho = 5$ bytes first and then capture another $\gamma = 5$ bytes packet, there is significant possibility that the function is either FC5 or FC6. Modelling it into a mathematical function:

$$\gamma = \rho = 5 \quad (1)$$

where S is the number of bytes of the request payload and the response payload (since request − response bytes = 0). In this case, $\gamma = 5$. The size of data and of register do not affect the overall size of the payload. Generally, the fields are always constant values with no added fields.

3.2.2 FC3 Read Holding Registers

The request payload always has a size of 5 bytes. However, the response payload varies. The following possibilities exist:

- request < response, for $\chi \geq 2$
- request > response, for $\chi = 1$

where x is the number of registers.

The size of data and number of registers affect the size of the response. Modeling the response payload:

$$\gamma = 2 * \chi + 2 \quad (2)$$

χ = number of registers. Every request is 5 bytes with y bytes response. +2 value represents the number of bytes to follow; always 2 bytes.

3.2.3 FC2 Read Discrete Inputs

The request payload always has a size of 5 bytes. However, the size of the response payload varies. The following possible cases occur:

- request > response for 1-16 Inputs or for χ=1 and χ=2
- request = response for 17-24 Inputs or for χ=3
- request < response for 25+ Inputs or for $\chi \geq 4$

The mathematical representation of the response payload is:

$$\gamma = (\text{inputs} / 8 \text{ bits}) + 2 = \chi + 2 \quad (3)$$

where χ is the number of registers. We notice that this function uses inputs instead of registers. Thus, this mathematical model *must be converted* to use the same variables. Assuming that one register has 8 inputs, the form of the mathematical function is $\gamma = \chi + 2$. Table 2 shows correlations between registers and inputs.

Table 2: Input - register correlations.

FCODE	FUNCTION
1-8 inputs	1 register
9-16 inputs	2 registers
17-24 inputs	3 registers
25-32 inputs	4 registers
…	…

Coils and inputs use roundup policies, so we have to adhere to these rules. These roundup policies cannot disrupt or stop the attacking process in any way, due to the above extensive analysis (examples will prove this statement later.) Notice also that every request 5 bytes yields y bytes in response.

3.2.4 FC4 Read Input Registers

Once more, the request size is 5 bytes and the response payload varies. So:

- request < response for $\chi \geq 2$
- request > response for χ=1

The model is defined the same way like before, χ is the number of registers. Every 5 bytes request yields y bytes response as follows:

$$\gamma = 2 * \chi + 2 \quad (4)$$

3.2.5 FC1 Read Coil Status

The size of the request payload is 5 bytes. However, the response payload has the following variations:

- request > response for 1-16 Coils or for χ=1 and χ=2
- request = response for 17-24 Coils or for χ=3
- request < response for 25+ Coils or for $\chi \geq 4$

We have to convert the mathematical formula, since this function uses coils. Therefore:

$$\gamma = (\text{num of coils} / 8 \text{ bits}) + 2 = \chi + 2 \quad (5)$$

χ is the number of registers. Considering that one register has 8 coils, the conversion is exactly the same as before. Roundup policies apply to this function too. Table 2 shows correlations in registers and coils:

Table 3: Coil - register correlations.

FCODE	FUNCTION
1-8 coils	1 register
9-16 coils	2 registers
17-24 coils	3 registers
25-32 coils	4 registers
…	…

3.2.6 FC15 Force Multiple Coils

In this case, unlike the others, the size of the response payload is always 5 bytes and the size of the request payload varies. The mathematical representation of the payload is:

$$\gamma = (\text{num of coils} / 8 \text{ bits}) + 6 = \chi + 6 \quad (6)$$

where χ is the number of registers. +6 represents the fixed fields in the request packet. This mathematical formula follows the conversion process, as it uses coils instead of registers. In addition, it conforms to a specific limitation:

$$\min | \text{FC15} | = \min | \text{coils} / 8\text{bits} + 6 | = \min |\text{coils} / 8 \text{ bits}| + 6 = 1 + 6 = 7 \text{ bytes}$$

The minimum value of the response size is 7 bytes. This has some interesting characteristics: The

size of request is always bigger than the size of the response. Thus, this function can be easily distinguished from the functions that we have described so far.

3.2.7 FC16 Preset Multiple Registers

Like FC15, the response size is 5 bytes and the request size varies. The mathematical model of the request payload is:

$$\gamma = 2 * \chi + 6 \quad (7)$$

where χ is again the number of registers. +6 represents the fixed fields in the request packet. This function also conforms to a limitation:

$$\min |FC16| = \min |2 * \chi + 6| = \min| 2 * \chi | + 6 = 2 + 6 = 8 \text{ bytes}$$

The minimum value of the response size is 8 bytes, meaning, that:

1. size of request > size of response
2. This function can be easily distinguished from functions FC1, FC2, FC3, FC4, FC5 and FC6.

This function can be distinguished from function FC15 if only the request size is 7 bytes.

3.3 ICS Tasks as Functional Sequences

All eight functions can be distinguished and grouped based on their utility. FC5 (write single coil) and FC6 (write single register) are grouped since they perform a write procedure on a single element. FC3 (read holding registers), FC2 (read discrete inputs), FC1 (read coil status) and FC4 (read input registers) perform a read procedure. FC15 (force multiple coils) and FC16 (preset multiple registers) perform a write procedure one or more elements. A functionality or even routine or task in ICS may consist of multiple serial executions of sequences of Function Codes (Modbus, 2017) (National Instruments, 2017).

To model functionality flows as sequences of executed MODBUS functions, we introduce the term *Functional Sequences*. A functional sequence is a sequence of dependent FC functions for the purpose of executing specific tasks or routines. It is a chain of combined processes. An example is:

$$\text{F.S.1} = \text{FC3 read holding registers} \rightarrow \text{FC6 write single register}$$

The functional sequence F.S.1 carries out a procedure that reads a number of registers and then writes one of them with a specific value. This procedure may represent a routine designed for the PLCs/RTUs. Notice that t is has to complete two steps before it terminates successfully. A functional sequence may have different functions and different number of steps depending on the policies of the industry. Therefore, the functional sequence takes the following form:

$$\text{Functional Sequence} = \text{Function 1} \rightarrow \ldots \rightarrow \text{Function N} \quad (8)$$

N is the number of steps. Notice that a functional sequence may have one step; one function. This complies, because this certain function cannot be combined with any other function in any way.

If an attacker has knowledge of the business/function flow of the logic ladder and business logic, he can define a number of functional sequences that are executed (Erickson, 2016). Knowing that, we show that he is capable of performing classification in each step and continuously reduce the uncertainty of identifying functions. This procedure is illustrated further in the second example of this paper.

4 PREDICTION MODELS

We now use the aforementioned mathematical representations to construct two prediction models. Each model consists of a *decision tree* that utilizes previous formulas and detected restrictions.

Worst-case Scenario. Notice that presented decision trees assume the worst-case implementation scenario: All probable collisions between packet-sizes are taken for granted but, in present industrial implementations, detection is leaking executed functions may be significantly easier.

Implementations often use different number of registers and coils in functions, leading to different packet sizes in responses (e.g. (Acromag, 2005) (Siemens, 2009). For example, generally FC1 and FC2 read functions can have colliding response packet sizes, since both sizes are calculated using equation $\gamma = (\chi/8 \text{ bits}) + 2$.

It is important to state here that variable χ may diversify in real-world (as shown in our PoC below). Modbus implementations often utilize different number of coils or registers (different χ), leading to different sizes and ultimately no collision.

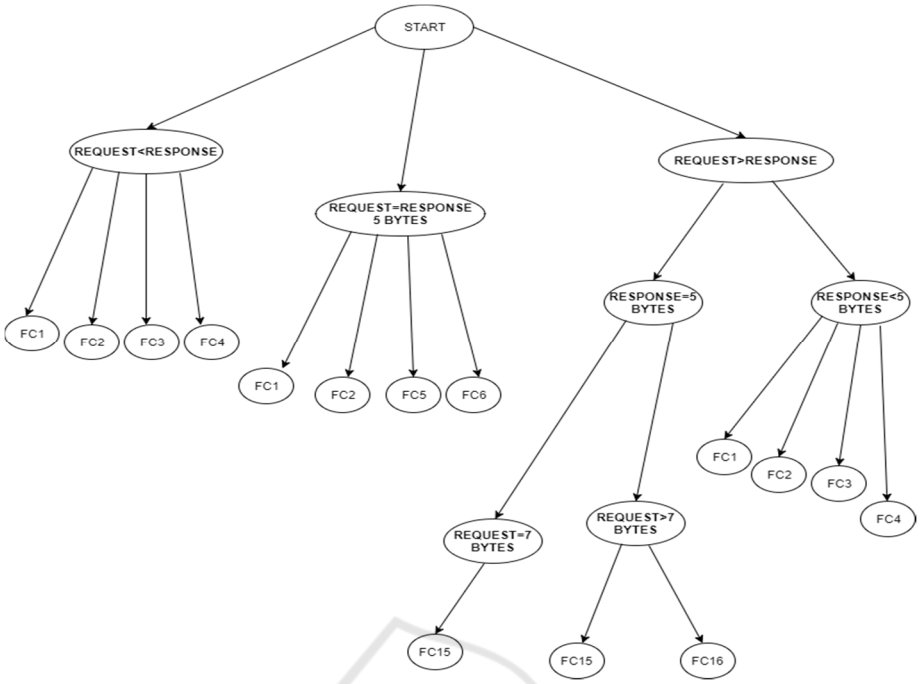

Figure 1: Decision Tree (no prior knowledge).

4.1 Decision Tree

The general decision tree uses every limitation and condition presented in the previous section. Each path is a different condition that leads to a subset of functions. When the attacker sniffs both the encrypted request packet and the response packet, he determines the request and response by relying on the sequence numbers in the headers. Then, he uses the tree to determine the function or the subset of possible functions. The general decision tree is built based on worst-case scenarios: All FC functions whose packet sizes may collide in real-world, are thought to collide.

At this point, attack scenarios have no prior knowledge of ICS functionality. Each function (leaf) in each subset (tree branch) has the same possibility of occurrence. For example, the attacker starts the packet sniffing, captures the first two packets and decapsulates them. He determines that the size of the encrypted request payload is 5 bytes and response is 8 bytes. He knows that the 5 bytes packet belongs to request and the 8 bytes packet belongs to response, because he can validate it using the sequence numbers inside the packet header (for example, TCP header, ESP header, etc.).

For example, if we capture two encrypted TCP payloads with Modbus request and response commands and determine that "request < response" in packet sizes, then FC1, FC2, FC3 and FC4 each have 25% possibility to be the ones executed (fig. 1). For any potential execution in typical Modbus, the worst case in this example is to have 4 out of 15 possible instructions executed. Generally, in the best case we achieve 100% successful execution leak (e.g. for FC15 and FC16). Still, these predictions can be improved.

In addition, the attacker can use the mathematical formulas/representations from the chapter 2.2 in order to determine the possible functions precisely:

- If it is FC1 then: $y = x + 2 \Rightarrow 8 = x + 2 \Rightarrow x = 6$ registers. But FC1 uses coils, so the conversion is $6*8 = 48$ coils. Therefore, the instruction is: FC1 Read Coil Status with coils from 41 to 48.
- If it is FC2 then: $y = x + 2 \Rightarrow 8 = x + 2 \Rightarrow x = 6$ registers. But FC2 uses inputs, so the conversion is $6*8 = 48$ inputs. Thus, the instruction is: FC2 Read Discrete Inputs with inputs from 41 to 48.
- If it is FC3 then: $y = 2*x + 2 \Rightarrow 8 = 2*x + 2 \Rightarrow x = 3$ registers. Therefore, the instruction is: FC3 Read Holding Registers with 3 registers.
- If it is FC4 then: $y = 2*x + 2 \Rightarrow 8 = 2*x + 2 \Rightarrow x = 3$ registers. Therefore, the instruction is: FC4 Read Input Registers with 3 registers.

4.2 Experimental Attack Scenarios

4.2.1 Packet Density

As in (Chen et al., 2010), we can utilize the term Packet Density to show that it is possible to

discriminate possibilities of detected encrypted traffic. In our case, packets represent a single Modbus function. The bytes used, are either the size of the request payload or the size of the response payload depending on the varying part of the function.

Packet Density = Nun of distinct packets/(max(bytes) – min(bytes))

where N∈Z and N>0. In Modbus protocol however as mentioned, there are collisions between some packet sizes. So, we have to adhere to the following rules:

- Distinct packets must be ≥ 2
- Omit duplicate packets (preserve only 1 of them)

According in (Chen et al., 2010), we define "density($\wp$) = |$\wp$| / [max($\wp$)-min($\wp$)], as the average number of packets for every possible packet size. If the value of the packet density is below 1, then the traffic can be easily distinguished" (Chen et al., 2010). In our case, there are only 8 different functions and sizes vary (Comey, 2014; Zhou and Feng, 2005) with only one collision (FC05 and FC06), i.e., Packet Density = 7 / (20 bytes – 4 bytes) = 0,43 < 1. Since packet density is below 1 and this means that the functions of the sample are distinguishable.

4.2.2 PoC Implementations

A PoC simulation was conducted to support these scenarios. We built a client-server system in C++ that simulated an HMI sending MODBUS RTU commands over TCP/IP encrypted with RC4. Simulations utilized the entire MODBUS protocol, simulating DWORDS, coils and registers.

When unpadded encryption is used, results support the predicted attack models and detection rates. Detection rates are even better in industrial appliances where FC1, FC2 and FC4 all have different packet sizes. This can lead to full execution leaks for sequences longer than three different FC functions. Even in worse scenarios, collisions detected between FC5 and FC6 and FC1 with FC3 (4 bytes) did not seem to directly affect information leakage when multiple instructions were executed sequentially.

4.2.3 Example Attack on Industrial Implementation

In this scenario, we simulate an example attack on an implementation using Function Codes. Notice that this is just an example of how Function Codes can be implemented in a way that avoids packet size collisions. We will show that eavesdropping encrypted TCP MODBUS commands can lead to full execution leaks, even in the presence of collisions (through statistical analysis).

All real-world ICS utilize at least 6 different sequences of instruction executions to support their functionality and these mostly include two or three different instructions executed per sequence). For this worst-case scenario, we make some realistic assumptions:

- Similarly with Microsoft (Chen et al., 2010), we allow the attacker to have some prior knowledge about executions that are utilized in ICS functionality (see Table 4), still without knowing the full functionality implemented scenarios (i.e. entire business logic and series of multiple functional sequence executions that comprise a full ICS functionality execution). Prior knowledge of the business/ function flow or fingerprinted ICS command executions for a time period can allow the attacker to recreate the general decision tree each time he observes a new payload size, thus eliminating potential execution sequences until he is certain of the sequence of executed MODBUS functions.
- The attacker does not know how the functional sequences are lined up or their frequency of execution (knowledge of ICS functionality execution for limited scenarios).
- The functional sequences execute a routine or a task. This is a simple example, which illustrates that our prediction models work even in cases with little functionality and collisions present.
- The more the number of functional sequences, the more diversity appears in the sample. This means that the chances of a successful prediction are higher.
- Knowing potential sequences of instruction executions allow us to develop an extended decision tree which further reduces chances of collisions (see Fig. 2).

Table 4: Example Function Sequences from MODBUS routines.

Functional sequence	Step 1	Step 2	Step 3
F. Sequence 1	Read FC1	Write FC5	-
F. Sequence 2	Read FC1	Write FC15	-
F. Sequence 3	Read FC3	Write FC6	-
F. Sequence 4	Read FC3	Write FC16	-
F. Sequence 5	Read FC4	Read FC3	Write FC16
F. Sequence 6	Read FC2	-	-

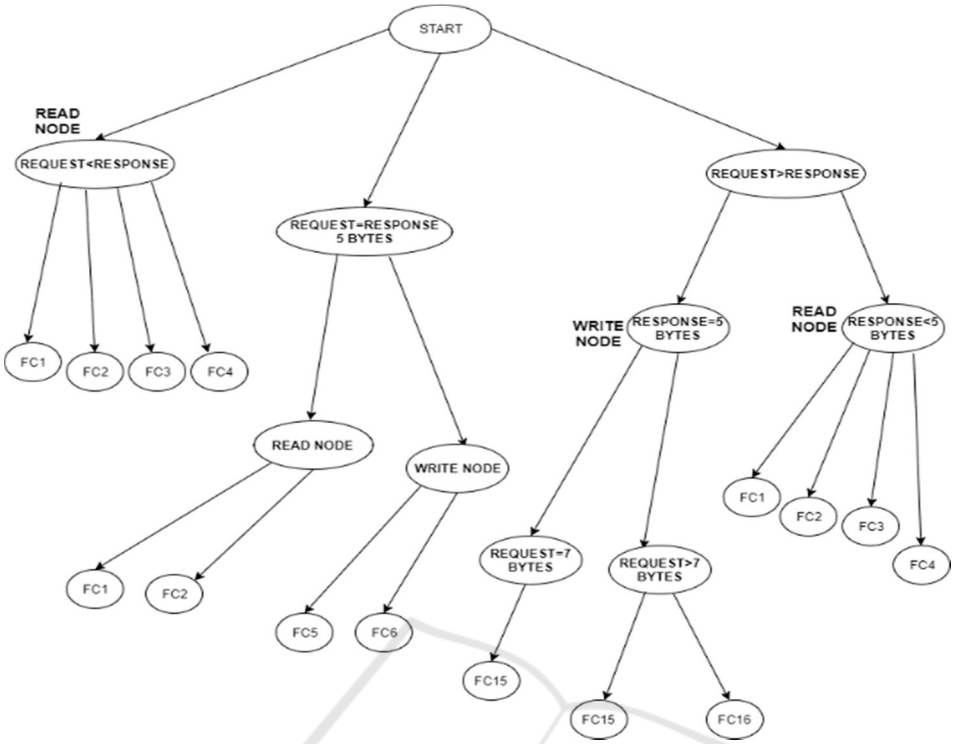

Figure 2: Decision Tree (with observed prior knowledge of ICS).

Suppose one has observed executions of sequences in Table 4 for a given short period of time. Frequencies below show the number of times that each functional sequence was executed during the observed period; i.e. what commands were sent to a PLC to complete a specific routine or task.

- F. sequence 1 executed 10 times
- F. sequence 2 executed 12 times
- F. sequence 3 executed 5 times
- F. sequence 4 executed 3 times
- F. sequence 5 executed 6 times
- F. sequence 6 executed 2 times

With this knowledge, an attacker can later either fully disclose what is being executed, or use probabilistic models to enhance predictions in worst-case scenarios where collisions occur.

Example Scenario. An attacker sniffs the first couple of packets and decapsulates. Request payload is 5 bytes so is response.

Using the special decision tree and following the path request = response and then the read node, he deduces that possible functions are FC1, since in the s7300 implementation FC1 and FC2 have different payload sizes. Specifically, it is FC1 then: $y = x + 2 \Rightarrow 5 = x + 2 \Rightarrow x = 3$ registers. Since FC1 uses coils, FC1 has coils from 17 to 24.

Next couple of payloads that are sniffed have a request payload of 7 bytes and a response payload of 5 bytes. Using the Decision Tree, he deduces that the function that he is looking for is FC15 (request > response and response = 7 bytes). For FC15: $y = x + 6 \Rightarrow 7 = x + 6 \Rightarrow x = 1$ register.

Since FC15 uses coils, so the conversion is 1*8 = 8 coils. Thus, the instruction is: FC15 Write Multiple Coils with coils from 1 to 8. Notice that the 2nd step consists of both read and write functions. So, the special decision tree cannot be used in this case, because it is unclear which direction to take.

Worst case Scenario. Collisions exist between F. Sequence 1 and 3, where both executed sequences of commands can have the same payload sizes. In other cases, an attacker is mostly able to leak executed functionality (sequences of instructions) since payload sizes differ. In reality, extended sequences of executed functions almost always leak executed functionality. Still, even in the presented worst case, leaks can be achieved statistically.

- The attacker starts the packet sniffing, captures the first couple of packets and decapsulates them. He determines the following payloads: Request payload is 5 bytes and response is 6 bytes. The

decision tree path is the node:"request < response". Possible functions: FC1, FC2, FC3 and FC4.

- At this stage, the likelihood of each instruction is calculated as P(FC1)=57%, P(FC2)=5%, P(FC3)= 21% and P(FC4)=15,7%.
- Next, the attacker sniffs payloads: 5 bytes request payload and 5 bytes response payload. Based on the first step and the general decision tree, possible instructions are either FC5 or FC6.
- In this case, the attacker *prunes* potential functionality and deduces that either sequence FC01-FC15 or FC3-FC6 has been executed.

Still, fingerprinting executions has turned up that Sequence 1 is used for monitoring values and is executed significantly more often than the second one. If we calculate probabilities according to fingerprinted executions of the systems logic:

- F. Sequence 1: P(FC1 | FC5) = 0.66 = 66%
- F. Sequence 3: P(FC3 | FC6) = 0.33 = 33%

Therefore, there is a statistical deviation of 33% in favor of F. Sequence 1 being executed. In any other scenario, full execution leakage is achieved. Tests have shown that as sequences become longer, they become more unique and collisions are minimized.

5 CONCLUSIONS

With appropriate measures the success of this type of attacks can be reduced dramatically. The most effective technique is the padding applied inside the data field. There are two types of padding: (i) roundup padding and/or (ii) random padding. Padding alters the original size giving a uniformity in the packets of the network, making any analysis of network traffic relatively weak (Stallings, 2016).

i. In roundup padding, block ciphers (such as AES) require blocks to have the size as the key used before the process of encryption. This means that blocks smaller than the length of the key will follow the procedure of roundup (Stallings, 2016).
ii. Random padding acts as a supplement to each packet, filling it with random bytes until it reaches a default size. The size may even be the maximum size which can be sent in Modbus protocol. In IPsec VPN and SSH, random padding can be used. It is mainly designed to hide possible leaks that may occur from side channel attacks.

These two cases of padding may be used at the same time as for example in the IPsec VPN, where the roundup padding is always used by AES and the random padding is an optional field (Kent, 2005).

Although it is true that padding can in many cases be the main measure against these types of side channel attacks, in practice it may create some overhead on the network traffic. Also, depending on implementation, integrity checks must run on all industrial instructions, since consistency in execution of functionality and performance is paramount. Potential delays or malfunction will be considered as a serious issue. It should be noted that padding may consume even 1/3 of the network bandwidth without fully subduing the information leaks (Chen et al., 2010). Depending on implementation, this may or may not be an issue in padding MODBUS communications. In realistic scenarios, and in order to protect the significant performance, low to moderate level of padding aggressiveness should be applied. However, this measure is not enough to protect the data.

REFERENCES

Modbus, F.A.Q., About the Protocol, 2017. *FAQ*. Modbus Organization inc. Viitattu.

Chen, S., Wang, R., Wang, X. and Zhang, K., 2010, May. *Side-channel leaks in web applications: A reality today, a challenge tomorrow*. In Security and Privacy (SP), 2010 IEEE Symposium on (pp. 191-206). IEEE.

Comey J., 2014. *FBI: Protecting critical infrastructure and the importance of partnerships*. FBI.

McLaughlin, S., Konstantinou, C., Wang, X., Davi, L., Sadeghi, A.R., Maniatakos, M. and Karri, R., 2016. *The cybersecurity landscape in industrial control systems*. Proceedings of the IEEE, 104(5), pp.1039-1057.

NI, 2017. *The Modbus Protocol In-Depth*, 2017. National Instruments (Online: accessed Mar 7 2017).

Knapp, E. D. and Langill, J. T., 2014. *Industrial Network Security: Securing critical infrastructure networks for smart grid, SCADA, and other Industrial Control Systems*. Syngress.

Modbus, 2017. *Simply Modbus*: Modbus TCP/IP. (accessed October 2017).

Modbus TCP, 2006. *Modbus Messaging on TCP/IP Implementation Guide* v1.0b. Modbus Organization, (accessed June 2017).

Erickson K., 2016. *Controllers, Programmable Logic. An Emphasis on design & application*. Dogwood Vally Press, LLC. Third edition.

Song, D., 2001. Timing analysis of keystrokes and SSH timing attacks. In *Proc. of 10th USENIX Security Symposium*, 2001. Usenix.

Kent S., IP, 2005. *Encapsulating Security Payload (ESP)*. RFC 4303. BBN Technologies.

Stallings W., 2016. *Cryptography and network security: Principles and practices*. Pearson.

Wright, C. V., Ballard, L., Coull, S. E., Monrose, F. and Masson, G. M., 2008, May. Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on* (pp. 35-49). IEEE.

Imperva, 2015. Attacking SSL when using RC4: Breaking SSL with a 13-year-old RC4 weakness. Imperva USA.

Matherly, J., 2009. Shodan search engine. *Available at [Online]: https://www. shodan. io*.

Brown, M., 2007. System Administration Toolkit: Set up remote access in UNIX through OpenSSH. *IBM, published Feb*, *13*.

Danezis, G., 2009. *Traffic Analysis of the HTTP Protocol over TLS*.

Sun, Q., Simon, D.R., Wang, Y.M., Russell, W., Padmanabhan, V.N. and Qiu, L., 2002. Statistical identification of encrypted web browsing traffic. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on* (pp. 19-30). IEEE.

Zhou, Y. and Feng, D., 2005. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. *IACR Cryptology ePrint Archive*, *2005*, p.388.

Langner, R., 2011. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, *9*(3), pp.49-51.

Munro, K., 2012. Deconstructing flame: the limitations of traditional defences. *Computer Fraud & Security*, *2012*(10), pp.8-11.

Kaspersky, 2014. *Cyperthreats to ICS systems*. *Available at [Online]:* http://media.kaspersky.com/en/business-security/critical-infrastructure-protection/Cyber_A4_Leaflet_eng_web.pdf .

Meserve, J., 2007. *Mouse click could plunge city into darkness, experts say*. CNN. com, 27.

Beresford, D., 2011. *Exploiting siemens simatic s7 plcs*. Black Hat USA, 16(2), pp.723-733.

Nan, C., Eusgeld, I. and Kröger, W., 2012, September. Hidden vulnerabilities due to interdependencies between two systems. In *International Workshop on Critical Information Infrastructures Security (pp. 252-263)*. Springer, Berlin, Heidelberg.

Acromag, 2005. Introduction To Modbus TCP/IP, Acromag Inc., USA. *Available at [Online]:* www.acromag. com/pdf/intro_modbus_TCP_765a.pd.

Siemens, 2009. *S7-300/S7-400 Loadable Driver for Point-to-Point CP*. Siemens Manual.