



Lost in Translation: When Industrial Protocol Translation Goes Wrong

Marco Balduzzi, Luca Bongiorno, Ryan Flores,
Philippe Z Lin, Charles Perine, Rainer Vosseler

TREND MICRO LEGAL DISCLAIMER

The information provided herein is for general information and educational purposes only. It is not intended and should not be construed to constitute legal advice. The information contained herein may not be applicable to all situations and may not reflect the most current situation. Nothing contained herein should be relied on or acted upon without the benefit of legal advice based on the particular facts and circumstances presented and nothing herein should be construed otherwise. Trend Micro reserves the right to modify the contents of this document at any time without prior notice.

Translations of any material into other languages are intended solely as a convenience. Translation accuracy is not guaranteed nor implied. If any questions arise related to the accuracy of a translation, please refer to the original language official version of the document. Any discrepancies or differences created in the translation are not binding and have no legal effect for compliance or enforcement purposes.

Although Trend Micro uses reasonable efforts to include accurate and up-to-date information herein, Trend Micro makes no warranties or representations of any kind as to its accuracy, currency, or completeness. You agree that access to and use of and reliance on this document and the content thereof is at your own risk. Trend Micro disclaims all warranties of any kind, express or implied. Neither Trend Micro nor any party involved in creating, producing, or delivering this document shall be liable for any consequence, loss, or damage, including direct, indirect, special, consequential, loss of business profits, or special damages, whatsoever arising out of access to, use of, or inability to use, or in connection with the use of this document, or any errors or omissions in the content thereof. Use of this information constitutes acceptance for use in an "as is" condition.

Published by

Trend Micro Research

Written by

Marco Balduzzi

Luca Bongiorno

Ryan Flores

Philippe Z Lin

Charles Perine

Rainer Vosseler

Stock image used under license from
Shutterstock.com

For Raimund Genes (1963 – 2017)

Contents

4

Introduction

6

Protocol Gateways

11

Modbus and Protocol Gateways

14

Security Testing

34

Device Vulnerabilities

39

Denial of Service

41

Cloud Support

43

Other Findings

44

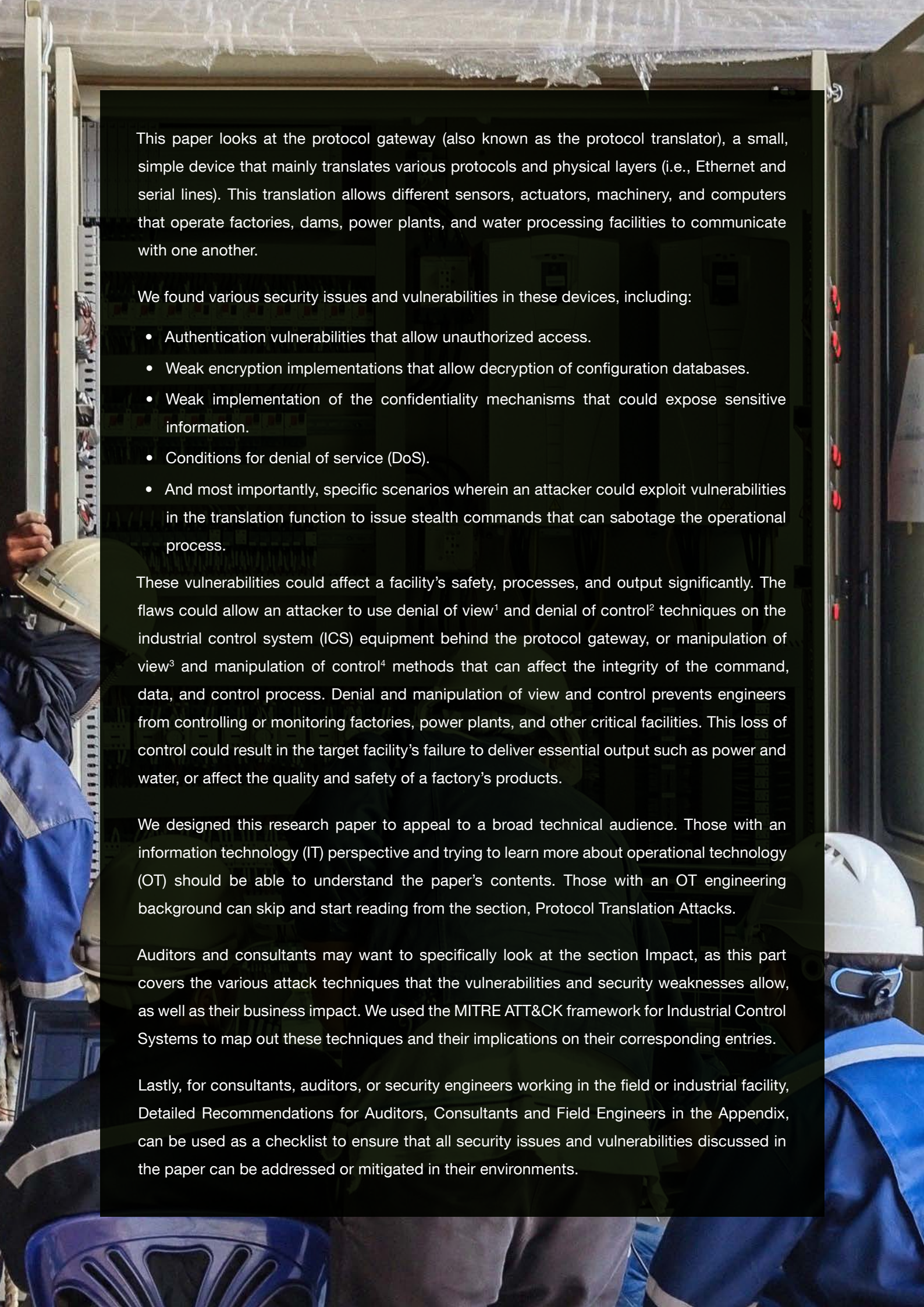
Impact

48

Discussion and Recommendations

50

Related Work



This paper looks at the protocol gateway (also known as the protocol translator), a small, simple device that mainly translates various protocols and physical layers (i.e., Ethernet and serial lines). This translation allows different sensors, actuators, machinery, and computers that operate factories, dams, power plants, and water processing facilities to communicate with one another.

We found various security issues and vulnerabilities in these devices, including:

- Authentication vulnerabilities that allow unauthorized access.
- Weak encryption implementations that allow decryption of configuration databases.
- Weak implementation of the confidentiality mechanisms that could expose sensitive information.
- Conditions for denial of service (DoS).
- And most importantly, specific scenarios wherein an attacker could exploit vulnerabilities in the translation function to issue stealth commands that can sabotage the operational process.

These vulnerabilities could affect a facility's safety, processes, and output significantly. The flaws could allow an attacker to use denial of view¹ and denial of control² techniques on the industrial control system (ICS) equipment behind the protocol gateway, or manipulation of view³ and manipulation of control⁴ methods that can affect the integrity of the command, data, and control process. Denial and manipulation of view and control prevents engineers from controlling or monitoring factories, power plants, and other critical facilities. This loss of control could result in the target facility's failure to deliver essential output such as power and water, or affect the quality and safety of a factory's products.

We designed this research paper to appeal to a broad technical audience. Those with an information technology (IT) perspective and trying to learn more about operational technology (OT) should be able to understand the paper's contents. Those with an OT engineering background can skip and start reading from the section, Protocol Translation Attacks.

Auditors and consultants may want to specifically look at the section Impact, as this part covers the various attack techniques that the vulnerabilities and security weaknesses allow, as well as their business impact. We used the MITRE ATT&CK framework for Industrial Control Systems to map out these techniques and their implications on their corresponding entries.

Lastly, for consultants, auditors, or security engineers working in the field or industrial facility, Detailed Recommendations for Auditors, Consultants and Field Engineers in the Appendix, can be used as a checklist to ensure that all security issues and vulnerabilities discussed in the paper can be addressed or mitigated in their environments.

Introduction

Everyday transactions like ordering food or asking for directions could be frustrating if the parties involved only spoke and understood different languages. A translator who speaks both languages would clear up any difficulties, much to the relief of those involved.

This demonstrates the importance of translation. On an individual level, one would need reliable translation when traveling abroad, reading foreign websites, or submitting legal documents to a foreign country or embassy. On a global level, documents for critical treaties and agreements involving peace, trade, and the environment are translated to the official languages of signatory countries.

Translation plays an important role in critical situations on a personal and global level. For example, one could get lost in a foreign country if the directions were incorrectly translated, or a person may unknowingly order and eat food that they were prohibited from eating because of a mistranslated menu. On a global scale, world leaders may sign a treaty that is disadvantageous to their people, the environment, or their territory because the translation failed to reflect important nuances.

This paper delves into the vital role of translation in industrial facilities by looking at the protocol gateway, also known as the protocol translator. The protocol gateway is an unassuming device that provides critical translation for machinery, sensors, actuators, and computers that operate factories, dams, power plants, and water processing facilities.

If protocol gateways fail, then the communication between the control systems and machinery would stop. The operators would not have visibility, rendering them unable to tell if machines or generators are running properly and within safety limits. Even when something is visibly wrong, it can also prevent the operator from issuing commands to start or stop processes.

This is precisely what happened in the Ukrainian power grid attack of December 2015. Attackers were able to access the power grid controls and issue commands to open the circuit breakers, causing a power outage. The attackers also disabled the protocol gateways in the substations by uploading corrupted firmware. This deliberate action effectively blocked recovery efforts made by the power grid engineers as commands from control systems to close the circuit breakers could not be transmitted due to the disabled protocol gateways. This prolonged the power outage and made recovery efforts much more difficult.⁵

In a report done by SANS ICS and the Electricity-Information Sharing Analysis Center (E-ISAC) about the incident, they called the firmware corruption of the protocol gateways “blowing the bridges.”⁶ It’s an apt description, as the attackers destroyed the protocol gateways—the translators that act as a bridge between the controllers and substations.

In this paper, we share our findings on the various security weaknesses and vulnerabilities of protocol gateways. Moreover, we will share findings for scenarios where attackers would not be “blowing the bridge,” but “using the bridge” instead to stealthily carry out malicious commands affecting the sensors, equipment, and machinery behind the protocol gateway.

Protocol Gateways

In an interconnected digital world, computers and machines use “languages,” or protocols in computing terms, to communicate with each other. But just like with people, a certain set of machines can only talk and understand their native language.

Many industrial machines, controllers, sensors, and actuators are designed to work together if they speak the same language. But an industrial environment is not always homogenous, or use the same protocol. Devices may come from different manufacturers and use different protocols, or different languages. As an analogy, it's like having one device that can only speak Japanese and another that only speaks English. A protocol gateway bridges the gap between the two devices by being the Japanese-to-English and English-to-Japanese translator that enables both devices to understand each other.

Protocol gateways are also necessary in Industry 4.0, which connects traditionally separate OT and IT networks. This gave rise to the scenario where older OT equipment, which can only communicate using OT protocols transmitted over serial cables, now needed to communicate with IT equipment over Ethernet cables, Wi-Fi, or mobile networks. As an analogy to describe how difficult this scenario is, we can say that OT equipment, which knows only braille, would need to communicate with IT equipment that only knows spoken English. A protocol gateway bridges the gap between the two by converting serial OT protocols (braille) into their TCP/IP (Transmission Control Protocol/Internet Protocol) equivalents (spoken English).

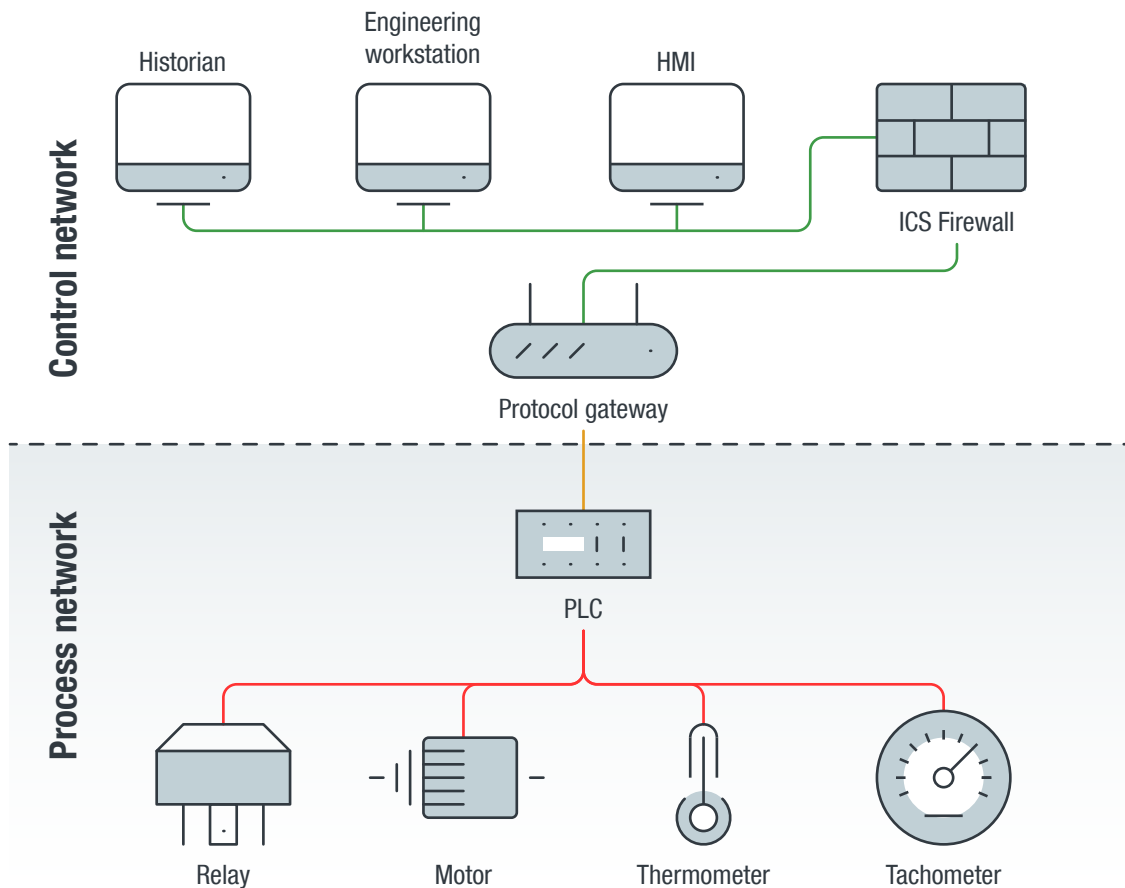


Figure 1. The typical position of a protocol gateway lies at the bottom of the control network, directly before the process network.

The process network in Figure 1 contains devices such as relays, motors, switches, and other sensors that are connected to a legacy programmable logic controller (PLC), which can only speak the Modbus RTU (remote terminal unit) protocol. These devices need to either send data (e.g., temperature reading from the thermometer, RPM from the tachometer) via the PLC to the Human Machine Interface (HMI), Historian, or Engineering Workstation in the control network. In return, an engineer or operator can also send instructions (e.g., open or close a valve, change a threshold value in the PLC) from the HMI. In Industry 4.0, HMIs are commonly in a separate network and would use Modbus TCP, the TCP/IP equivalent of the Modbus RTU protocol. For the devices to receive instructions from the HMI, and for the devices to send data to the HMI, a protocol gateway is needed to translate Modbus RTU to Modbus TCP and vice versa.

Figure 2 shows the Purdue Architecture Model commonly followed by industrial networks. Figure 1 zooms into levels 0, 1, 2 and 3, of this model and focuses on where protocol gateways sit in relation to the sensors, actuators, PLCs, HMIs, historians, and others.

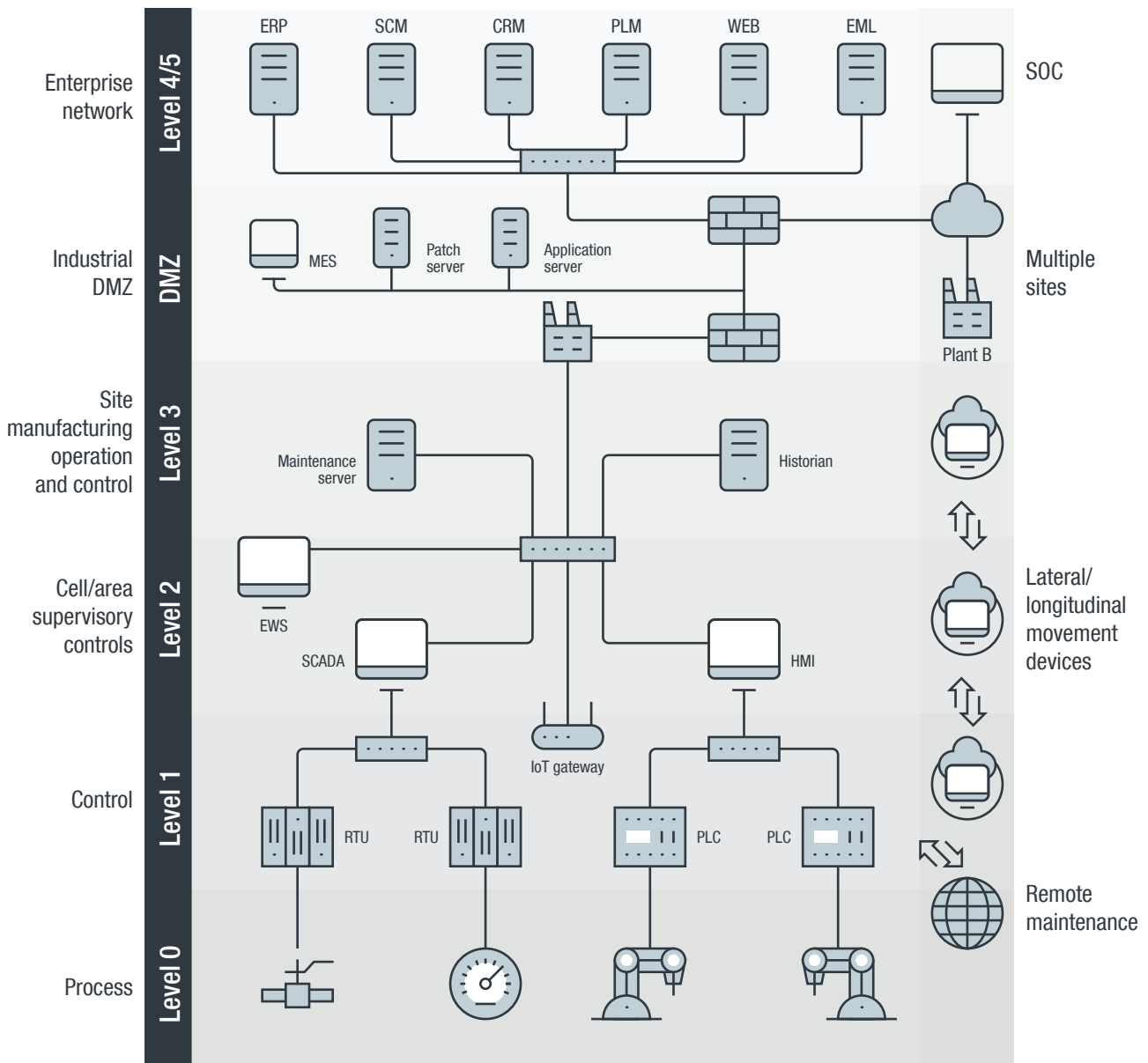


Figure 2. The Purdue Model example of a factory network shows which levels the industrial network maps to

As one might realize by now, any disruption or compromise to the protocol gateway can cripple the control network.

Protocol gateways are small devices usually no bigger than a home router. They cost anywhere from US\$300 for basic models to US\$1,200 for fully-featured ones. Most vendors that manufacture industrial equipment, such as Schneider Electric and Rockwell Automation, have a protocol gateway device in their catalog. Some smaller, emerging players, such as Nexcom, also sell protocol gateways with more features at lower prices to compete with the larger vendors.

While conducting this research, we discovered that protocol gateways can be categorized into two macro-categories.

1. Real-time gateways translate traffic in real time where every incoming packet is immediately evaluated, translated, and forwarded. How real-time gateways operate is similar to how sign language interpreters translate news during a live broadcast.
2. Data stations adopt an offline translation approach, where the translation mechanism operates asynchronously. For example, data stations do not wait for a read request to fetch the data from a connected PLC, but regularly query the PLC for updates and keep an internal cache to serve the data upon request.

As a result of this categorization, real-time gateways translate the packets on-the-fly (upon validating and parsing them according to protocol specifications), while data stations match the incoming packets against a translation table that users are asked to configure in the gateway manually. This table, normally called the I/O mapping table, operates similarly to a routing table, which indicates how the inbound requests need to be routed to the final peer and in which way.

Another important aspect of protocol gateways is the type of protocols that they support and convert. For simplicity, the different devices available on the market can be grouped into three.

1. Gateways that translate within the same protocol (e.g., Modbus) and across different physical layers (e.g., TCP to RTU). Analogous to translating spoken English to English braille.
2. Gateways that translate within the same physical layer and across different protocols (e.g., Modbus RTU to Profibus, both serial protocols). Analogous to translating German braille to English braille.
3. Gateways that translate across different protocols and physical layers (e.g., Modbus TCP to Profibus). Analogous to translating English to German braille.

In this research, we decided to focus on the first group of devices. The last two, which support translation across different protocols, we leave to explore in future works.

The following table summarizes the protocol gateways from different manufacturers that we considered in our research and believe to be representative of what can be found in real installations.

Gateway	Text acronym	Country of vendor	Price range	Interfaces	Type	OS
Nexcom NIO50	NIO50	Taiwan	US\$200	Ethernet, serial (RS232/422/485), wireless	Real-time gateway	FreeRTOS
Schneider Link 150	Link150	France	US\$600	Ethernet, serial (RS232/485)	Real-time gateway	ThreadX
Digi One IA	Digi One	USA	US\$350	Ethernet, serial (RS232/422/485)	Real-time gateway	Vendor-specific embedded Linux
Red Lion DA10D	DA10D	USA	US\$650	Ethernet, serial (RS232/422/485)	Data station	Vendor-specific embedded Linux
Moxa MGate 5105-MB-EIP	MGate 5105	Taiwan	US\$500	Ethernet, serial (RS232/485)	Data station	Embedded Linux for ARM EABI5

Table 1. A high-level summary of the industrial protocol gateways considered in our research.

Just like human language translators, some translators are only bilingual. However, other translators are polyglots who understand and speak multiple languages. Table 2 lists each of the devices' supported protocols.

Gateway	Supported protocols	Supported translations
NIO50	Modbus TCP, Modbus RTU, Modbus ASCII*, MQTT	Transparent, Modbus TCP (master/slave), Modbus RTU (master/slave), Modbus TCP (master/slave) to MQTT, Modbus RTU (master/slave) to MQTT
Link 150	Modbus TCP, Modbus RTU, Modbus ASCII, JBUS, powerlogic	Modbus TCP (master/slave), Modbus RTU (master/slave),
Digi One	Modbus TCP, Modbus RTU, Modbus ASCII	Transparent, Modbus TCP (master/slave), Modbus RTU (master/slave),
DA10D	300 protocols including Modbus, MQTT	All combinations
MGate 5105	Modbus TCP, Modbus RTU, Modbus ASCII, EthernetIP, MQTT, cloud services	All combinations including Modbus TCP (master/slave), Modbus RTU (master/slave), ethernetIP (adapter/scanner), MQTT and cloud services (client)

Table 2. Gateways and their supported protocols and translations

In our analysis, we chose to focus on Modbus translation as Modbus is one of the first and most widely used OT protocols globally. Modbus is an open standard, and its usage is royalty-free. It's non-vendor specific and allows interoperability between various equipment from different manufacturers. Our choice of devices to test and purchase reflects this focus; all of them support Modbus. The most expensive device in table 2, the DA10D, supports 300 protocols, including Modbus. This gives an idea of the number of protocols that industrial networks use.

Other protocol gateways offer additional features such as MQTT (Message Queuing Telemetry Transport), which is supported by two more expensive gateways, the DA10D and MGate 5105, and by NIO50, the least expensive but an emerging player in the market.

To test the security of the protocol gateways, we did a basic analysis of various authentication mechanisms used to control and operate the protocol gateway. More importantly, we set out to test the protocol gateways' translation capability, which is the novelty of this research.

Reliable language translators should provide not only fast and accurate translation but also be able to handle grammatical errors gracefully and adjust for speakers who are difficult to understand. In terms of secure protocol gateways, they should be able to handle malformed packets that do not follow protocol specifications (similar to understanding the meaning of a sentence despite its grammatical errors), and also perform well when handling a large amount of traffic (similar to catching up with a fast speaker).

To fully understand how we conducted our analysis and the impact of the vulnerabilities we discovered, the next section is a quick primer on the Modbus protocol.

* Modbus ASCII is a less used variant of Modbus RTU in which the payload information is encoded in ASCII format. The packet structure and specifications are the same.

Modbus and Protocol Gateways

Modbus is an application-layer messaging protocol that provides client/server communication across intelligent devices in industrial networks. The protocol was standardized in 1979 and rapidly became a de facto industrial standard for serial communication (e.g., over RS-232 and RS-485).⁷

The Modbus TCP has since been introduced to enable the communication over TCP/IP stack, on port 502. Contrary to common understanding, a Modbus master operates as a TCP/IP client and issues requests to a Modbus slave, which acts as a server.

Figure 3 shows a sample configuration of a protocol gateway. The gateway is configured on the Ethernet interface as Modbus TCP slave (labeled in green) and responds to queries from an HMI that is acting as the control server and Modbus master. The gateway is configured on the serial interface as the Modbus RTU master (labeled in orange) and translates the requests to a PLC acting as the Modbus slave.

PLCs supervise different devices. In our example, it supervises a relay, a motor, a thermometer, and a tachometer.

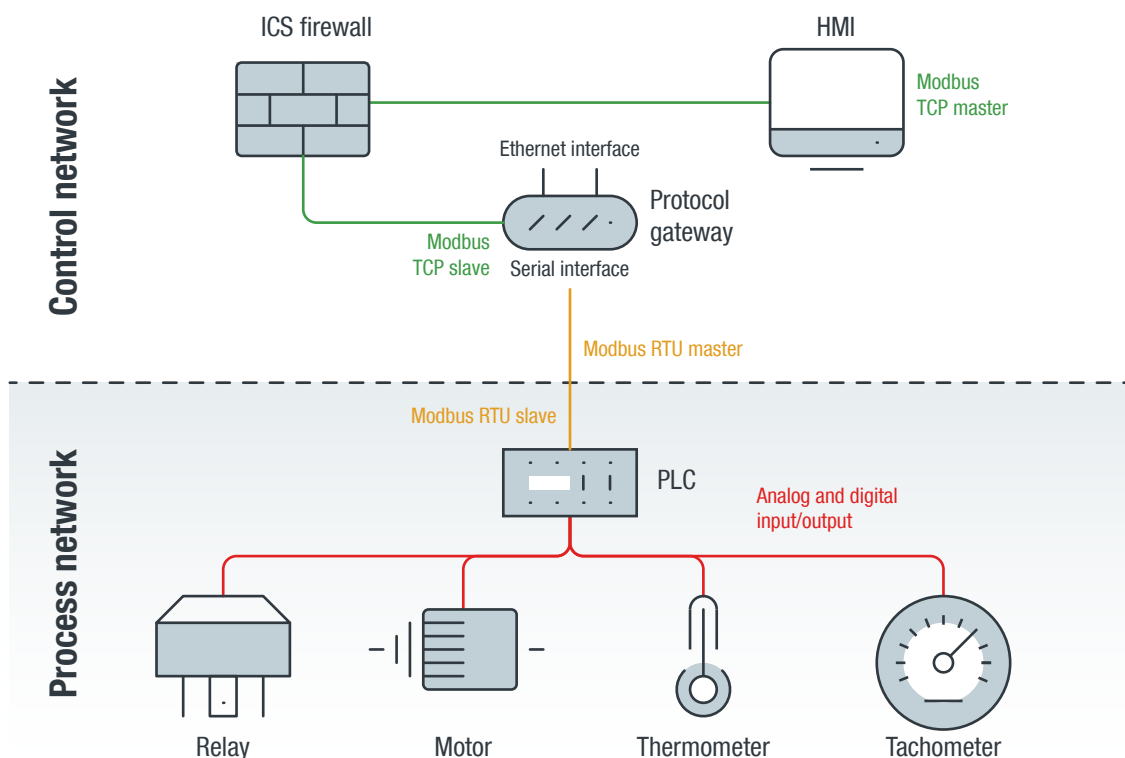


Figure 3. The translation between Modbus TCP and Modbus RTU. In this setup, the gateway is configured to operate as slave on the Ethernet interface and master on the serial interface. An HMI (master) dialogs with a legacy PLC (slave) via the protocol gateway.

Modbus messages are composed of mandatory information. These are:

- Unit ID: The recipient's identifier or to whom the message is for
- Message length: How long the message is
- Function code: Instructions for the recipient or what needs to be done. Refer to Table 3 for the common Modbus function codes.

Function code	Message type	Type of object
1	Read Coils	Binary (1 bit)
2	Read Discrete Inputs	Binary (1 bit)
3	Read Holding Registers	Words (16 bit)
4	Read Input Registers	Words (16 bit)
5	Write Single Coil	Binary (1 bit)
6	Write Single Holding Register	Words (16 bit)
15	Write Multiple Coils	Sequence of Coils (N*1 bit)
16	Write Multiple Holding Registers	Sequence of Words (N*16 bit)

Table 3. Summary of the commonly used Modbus function codes⁸

As shown in Table 3, the common Modbus functions are either reading from or writing to Coils or Registers. Coils are switches that are either on or off. Registers hold data, which can be a measurement of a sensor, a threshold, or a configuration value. For example, the temperature reading of a thermometer is stored in a register.

To put this into use, here is an example. A requesting message having a function code of 5 indicates a write single coil request. This message is sent by a master to set the binary coil (a switch) of a slave. This message includes, on top of the mandatory information previously enumerated, the address of the coil to be set and the corresponding value (0xFF00=ON, 0x0000=OFF). In a successful scenario, the slave responds with a message which has similar fields to the original requesting message, indicating which coil has been set and the new value.

While Modbus TCP and Modbus RTU may look similar, they have a few interesting differences. One of these differences lies in the way they run on different layers. Modbus TCP runs at the application layer of the TCP/IP stack (layer 7 of the ISO/OSI model), while Modbus RTU directly operates on serial lines. Another difference: Modbus TCP includes a Modbus Application Layer consisting of a transaction identifier, a protocol identifier, and the message length field indicating the length of the payload (e.g., the request). Modbus RTU also uses a checksum (CRC16) suffix for data integrity checks, which Modbus TCP does not do.

In a real-world scenario, where a master node communicates with a slave node by means of a protocol gateway translating Modbus TCP to Modbus RTU and vice-versa, these are the messages being exchanged for reading a coil, such as to tell whether a motor is turned on. An HMI would produce the following request message:

	Modbus application header				Modbus protocol data unit			
	Transaction ID	Protocol ID	Length	Unit ID	Function code	Starting address	Quantity of coils	CRC
Modbus TCP	0001	0000	0006	01	01	0001	0001	

The protocol gateway translates the message above to the message below

Modbus RTU				01	01	0001	0001	AC0A
------------	--	--	--	----	----	------	------	------

If everything was normal, this is the response:

	Transaction ID	Protocol ID	Length	Unit ID	Function code	Byte count	Coil status	CRC
Modbus RTU				01	01	01	01	9048

The protocol gateway translates the message above to the message below

Modbus TCP	0001	0000	0004	01	01	01	01	
------------	------	------	------	----	----	----	----	--

Figure 4. An example translation of an order from the HMI and the response

Note that the payload in Figure 4 (indicated in bold), also known as PDU (protocol data unit), is the same in both Modbus TCP and Modbus RTU. Therefore, an imprudent gateway can simply forward the payload by adding/removing the Application Header and CRC, while a carefully designed gateway may check for the validity of the unit ID, length, starting address, and packet structures as dictated by the function codes' specification. We will discuss this aspect in detail later.

Security Testing

When a security analyst is asked to evaluate a technology they are not familiar with, for example, a technology that looks like a standard desktop application, he may run into different unprecedented challenges. Common best practices, like logging, debugging, or automated analysis, may indeed get complicated when dealing with embedded or proprietary devices.

In our scenario, we wanted to understand and investigate the ability of protocol gateways as a technology to correctly translate industrial protocols, specifically when they operate in difficult situations that involve heavy or malformed traffic, from a malicious actor.

We began by evaluating the gateways' capabilities in detecting and dropping malformed packets, or packets that do not comply with the protocol specifications. It is expected that such devices behave intelligently, especially in the context of modern, smart, and complex Industry 4.0 networks. These devices are expected to be able to understand the protocol format and take appropriate translation strategies — based on the protocol's semantics — rather than blindly forwarding the traffic from one interface to another. In other words, protocol gateways are expected to implement appropriate filtering capabilities like an application firewall does, to translate securely and properly.

We dug deeper into the implementation of the protocol translation process and researched the conditions in which the gateways may introduce errors that have an impact on the device they communicate with, such as a PLC connected to the serial interface. This is the equivalent of testing if a language translator can correctly translate sentences with mismatched tenses, subject-verb agreement errors, and misplaced or missing punctuation. A reliable translator will either correct the sentence if the context is obvious enough or refuse to translate if the message is unclear in its present form.

In our evaluation, we adopted a black-box approach wherein we compared the translated traffic of a network gateway with network traces that were generated and fed to the gateway. This strategy was dictated by the fact that the gateways that we considered in our analysis do not publicly disclose information on their design or implementation. To this end, we made use of an automated testing and analysis framework (depicted in Figure 5) to test all gateways under the same conditions successfully, and to exhaustively cover the largest number of potential corner cases in terms of protocol specifications.

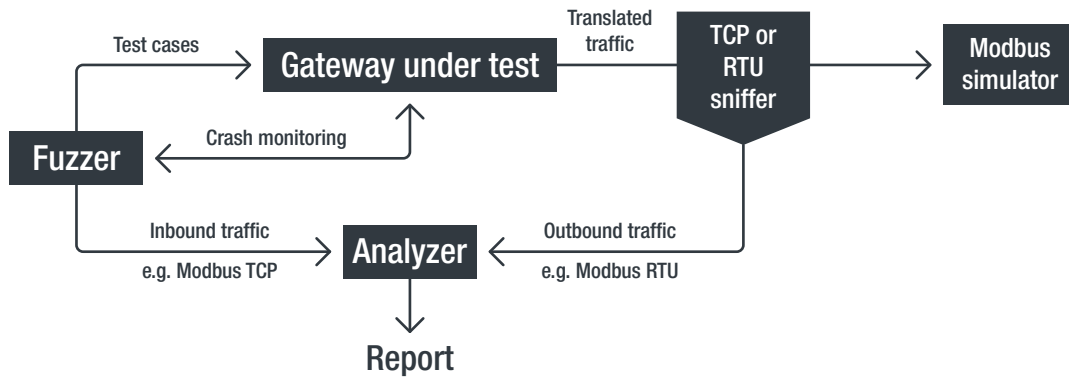


Figure 5. The architecture of our testing & analysis framework

As depicted in Figure 5, our framework consists of the following components:

- A fuzzer that generates the inbound traffic for the gateway under test. For example, when testing the translation from Modbus TCP to Modbus RTU, the fuzzer generates Modbus TCP test cases.
- A simulator that simulates the receiving station, such as a PLC implementing a Modbus RTU Slave. The simulator is needed because the protocol gateways often operate incorrectly (or not operate at all) if they are not connected to certain devices.
- A sniffer that collects information on the outbound traffic (i.e., the translated protocol).
- An analyzer that collects both inbound and outbound traffic for the analysis.

The gateway can operate as Modbus master or Modbus slave, and can translate from Modbus TCP or Modbus RTU, giving it four configuration options. All our gateways support these four main configurations, on top of additional ones like transparent translation (forwarding) or cloud connectivity.

Our framework can operate on all four configurations with appropriate configuration of the components. For example, to test the translation from Modbus TCP (master) to Modbus RTU (slave), the fuzzer is instructed to generate Modbus TCP master requests and the simulator to behave as Modbus RTU slave that responds to the requests.

In our test lab, we used a networked power switch,⁹ which allowed us to reboot the test device under test when the device stopped responding. Serial traffic was captured in IONinja¹⁰ using either the included software-based capture driver or an EZ™ Tap Pro.¹¹ Figure 6 shows our test setup.

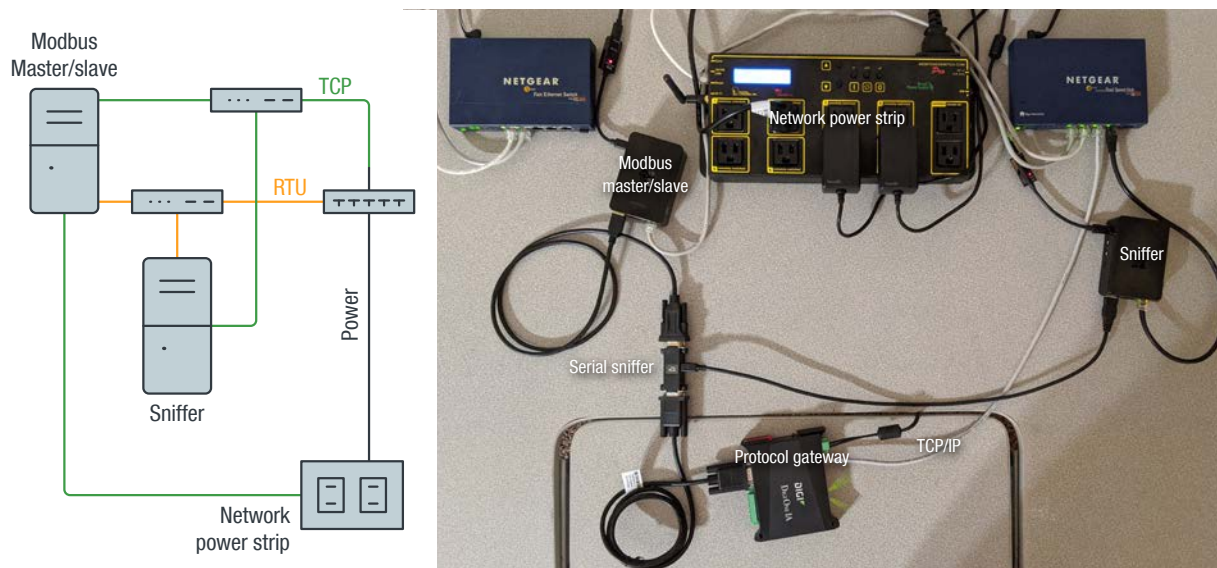


Figure 6. The diagram and actual test setup.

When it comes to the implementation details, we used the open-source QModMaster software¹² to simulate a Modbus master node and pyModSlave¹³ to simulate the slave. We adapted the software to our needs, such as for acquiring the data from /dev/ttyUSB0 serial.

To capture the translated traffic from the sniffer, we used Wireshark¹⁴ for Modbus TCP and IONinja¹⁵ for Modbus RTU. We wrote dedicated parsers to convert the outputs of the two programs to a common syntax that our analyzer would understand.

Figure 7 shows a Modbus TCP to Modbus RTU conversion session. The first field indicates the timestamp.

```

1574704509.746888,TCP,00:01:00:00:00:06:01:00:00:01:00:01
1574704509.770035,RTU,01:00:00:01:00:01:91:CA
1574704511.271468,TCP,00:02:00:00:00:06:01:01:00:01:00:01
1574704511.289164,RTU,01:01:00:01:00:01:AC:0A
1574704512.802031,TCP,00:03:00:00:00:06:01:02:00:01:00:01
1574704512.875859,RTU,01:02:00:01:00:01:E8:0A
1574704514.328139,TCP,00:04:00:00:00:06:01:03:00:01:00:01
1574704514.343510,RTU,01:03:00:01:00:01:D5:CA
1574704515.860150,TCP,00:05:00:00:00:06:01:04:00:01:00:01
1574704515.878557,RTU,01:04:00:01:00:01:60:0A

```

Figure 7. An example of a Modbus TCP to Modbus RTU conversion session

When it comes to the fuzzer, two main categories of fuzzers are known to researchers. The first category is called generation-based, which works well with protocols with specifications that are known to the public, such as Modbus. In this category we reference BooFuzz¹⁶ and Sulley.¹⁷ Opposite to these, mutation fuzzing is used to learn from a protocol such as proprietary ones to generate permutations such as Radasma¹⁸ and PropFuzz.¹⁹

Our fuzzer is built around BooFuzz, but we also integrated part of the boofuzz-modbus project distributed under Apache license.²⁰ We developed our fuzzer to make it portable to different ICS protocols, and used it to test several Modbus implementations. The snippet shows an example routine for generating permutations of Modbus TCP's write coil requests.

```
def write_single_coil(session):
    s_initialize('write_single_coil')
    with s_block('adu'):
        s_incr('transId')
        s_word(0x0000, name='protoId', endian=cfg.endian, fuzzable=cfg.fuzz_proto_id)
        s_size('pdu', length=2, offset=1, name='length', endian=cfg.endian,
fuzzable=cfg.fuzz_length)
        s_byte(cfg.slave_id, name='unitId', fuzzable=cfg.fuzz_slave_id)
    with s_block('pdu'):
        s_byte(0x05, name='write_single_coil', fuzzable=False)
        s_word(0x0001, name='address', endian=cfg.endian, fuzzable=cfg.fuzz_address)
        if cfg.random_coil_value:
            s_word(0x0000, name='outputValue', endian=cfg.endian, fuzzable=True)
        else:
            s_group(name='outputValue', values=['\x00\x00', '\xFF\x00'])
            if cfg.trailing_garbage:
                s_random('', cfg.gmin, cfg.gmax, num_mutations=cfg.gmut,
name='trailing_garbage')
            session.connect(s_get('write_single_coil'))
```

Figure 8. An example routine for generating permutations of Modbus TCP's write coil requests

The fuzzer also operates a monitor routine aimed at detecting unexpected conditions such as a device crash caused by a distributed denial of service (DDoS) attack. Thanks to monitoring, we automatically discovered a few of the problems discussed in the section Denial of Service.

The detection of errors in the protocol is instead left to a dedicated component, the analyzer. The analyzer compares inbound and outbound traffic for inconsistencies. In our example of conversion from Modbus TCP to Modbus RTU, a protocol gateway is supposed to at least remove the Modbus Application Header (ADU), then compute and append the Modbus RTU's CRC. The payload (PDU) is not supposed to be altered. However, corner cases may result in faulty behaviors of the gateway. The analyzer is instructed to look for these cases using a series of heuristics that we manually developed.

This comparison performed by the analyzer is based on the packets' timestamp as inbound packets are translated sequentially (inbound 1, outbound 1, inbound 2, outbound 2, etc.). However, this is not always the case for data stations where packets might be translated asynchronously (inbound 1, inbound 2, outbound 2). In this last case, the fuzzer includes a nonce (semi-random values) in the generated requests' payload to help the analyzer match an inbound packet to its translated one. Also, for data stations, we limited our analysis to only write functions (i.e., 5, 6, 15, 16); read requests are not translated since the data stations regularly poll the Modbus slave.

Protocol Translation Attacks

As with our real-life language translator example, the protocol gateway is not the star of the show, but it can cause a lot of problems if its translation fails. Depending on the situation, the protocol gateway can become the weakest link in the industrial facility's chain of devices, and a sophisticated attacker may target such devices for two important reasons:

1. Protocol gateways are unlikely to fall under the inventory of critical assets to be monitored by a security agent or logging system; the attack is less likely to be noticed.
2. Translation issues are difficult to diagnose by nature. As we show in the rest of the document, especially in this section, design errors in protocol gateways may allow an attacker to run very stealthy attacks, presenting an interesting possibility for advanced attackers.

Real-Time Gateways

A secure protocol gateway should not only provide fast and accurate translation between protocols and physical layers, but also handle packets that do not conform to protocol specifications and provide the correct command and proper values to the receiving master or slave.

To evaluate the gateways' capability to detect and discard malformed packets, we used our fuzzer to generate 5,078 invalid Modbus TCP packets and 1,659 invalid Modbus RTU packets. We fed them to the tested gateways and calculated the drop rate or percentage of packets that the device discarded.

Note that only the real-time gateways (NIO50, Link150, Digi One) have been tested in this section, because data stations employ different methods to adhere to the protocol specifications.

From this initial evaluation, we can report that:

- The Link 150 filtered out the most invalid packets with a drop rate of over 30% for Modbus TCP and over 20% for Modbus RTU.
- The Digi One had similar results on Modbus TCP but only discarded 9% of the Modbus RTU packets.
- The NIO50 performed well on Modbus RTU but failed to filter out any of the malformed Modbus TCP packets.

Protocol	NIO50	Link 150	Digi One
Modbus TCP	0.00%	58.13%	55.64%
Modbus RTU	19.55%	19.75%	8.85%

Table 4. Initial evaluation of the gateways' firewalling capabilities

As the NIO50 seemed to operate poorly with respect to the handling of malformed Modbus TCP traffic, we decided to dig deeper and inspected the translation handling of this device.

In particular, we observed that 2,454 of the packets sent to this device had been improperly translated and not filtered. These Modbus TCP packets were constructed by the fuzzer to be purposely malformed, such that the message length in the application header is different from the calculated length of the Modbus payload. In fact, they all violate the message length specifications.

Table 5 shows a *write multiple registers (function code 0x10)* message that requests the writing of two registers. However, the message length field indicates a packet length of 9 bytes instead of 11 bytes (0B in hexadecimal).

Transaction ID	Protocol ID	Message length	Unit ID	Function code	Starting address	Number of registers	Byte count	Register values	
0001	0000	0009 (correct is 000B)	01	10	0000	0002	04	0001	0005

Table 5. Example of the *write multiple registers* message having a wrong message length field

When one of these invalid packets reaches the gateway, the device forwards the packet as is (no translation is made), instead of dropping the packet or correcting its length, which a secure and reliable protocol gateway should do. As a result, the gateway pollutes the serial bus with Modbus TCP packets, while only packets that are compliant with the Modbus RTU specification are supposed to exist on this bus.

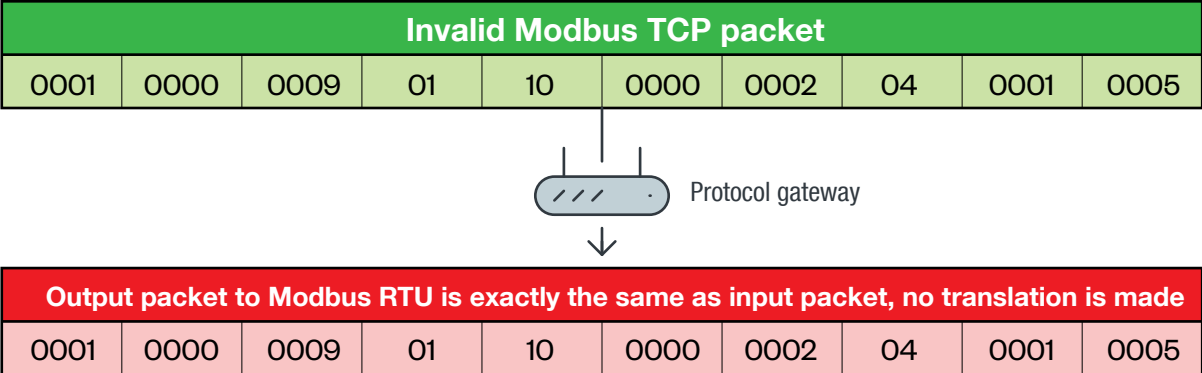


Figure 9. Illustration of how the transmitted message is the same for both outbound and inbound signals

The other gateways that we have tested handled this case correctly, in particular, by adopting one of the following strategies. They either dropped these malformed packets or fixed them before translating (e.g., by reducing the length to match the length specified in the header’s field, or adjusted the field’s value to the real length). We have reported the translation vulnerability that we found on NIO50 to the vendor, as ZDI-CAN-10485.

The vulnerability might not trigger any alarm bells as it is just a blind forwarding of invalid packets. However, we can carefully design a packet with an incorrect length in Modbus TCP while being valid and meaningful in Modbus RTU. When the packet gets to the other end (Modbus RTU), it is structurally valid and has a meaningful data structure.

A word can have the same set of characters and spelling, but can have widely different meanings depending on the recipient’s language. An example of this is the word “gift,” which means “present” in English; “married” in Danish, Norwegian, and Swedish; but means “poison” in German.

To test the NIO50 translation vulnerability, we then asked ourselves if there was any condition where the protocol gateway blindly forwards the word “gift” from an English speaker, without realizing that the recipient, which speaks German and will translate it as “poison.”

To do that, we need to design packets with the following characteristics:

- It will trigger the blind forwarding vulnerability of the protocol gateway.
- It will be a valid Modbus RTU packet, even though it is sent as a Modbus TCP packet initially.
- If read as a Modbus RTU packet, it will have a completely different meaning when compared to its Modbus TCP equivalent.

The packet proposed in Figure 10 is one of the many packets an attacker can design to meet the three criteria outlined above.

Modbus TCP	Transaction ID		Protocol ID	Message length	Unit ID	Function code	Starting address	Number of registers
Packet	01	0F	0000	0011	03	04	D10E	0070
Modbus RTU	Slave ID	Function Code	Starting address	Number of coils	Byte count	Data		CRC

Figure 10. Attack packet and semantic for Modbus TCP (read input registers) and Modbus RTU (write multiple calls)

When this packet is parsed with the semantic of Modbus TCP, it gets interpreted as read input registers (Function Code 04) from unit ID 3. But when the semantic of Modbus RTU is applied, it is interpreted as write multiple coils (Function Code 15 and 0F in hexadecimal notation) to unit ID 1.

This vulnerability is significant. In fact, a legitimate, innocent read message becomes a write request because the protocol gateway mishandles the translation. An advanced attacker may exploit this vulnerability to circumvent ICS Firewalls that block writing functions requested from the IP that is not in the whitelist.

Figure 11 shows a scenario where an attacker has gained access to a historian and is sending a *read input registers* message within the semantic of Modbus TCP. The request is a valid read of 0x70 (112) registers, beginning with address 0xD1CE. As the message is sent, it passes the ICS firewall – where it is considered a legitimate request – and reaches the protocol gateway. However, due to the way the gateway handles Modbus TCP messages with incorrect message length (0x11 instead of 0x06), the request is forwarded to the PLC without a proper translation. As a result, the PLC interprets the message in the context of Modbus RTU (i.e., as a *write multiple coils* request). As depicted in Figure 11, the request is translated as a write of three-byte coils: 0x04D1CE. In particular, the first byte (0x04, 0000 0100 in binary) is used to control the PLC’s devices and produces the following result: turn off the tachometer (address 1), turn off the thermometer (address 2), turn on the motor (address 3) and turn off the relay (address 4).

With a single command, the attacker can deactivate the critical sensors for monitoring the motor’s performance and safety (temperature and tachometer), while keeping the motor running. If unnoticed by field engineers and operators, the motor could already be exceeding the safe operating conditions, however, it won’t be visible or trigger any alarms because the sensors (thermometer and tachometer) have been disabled.

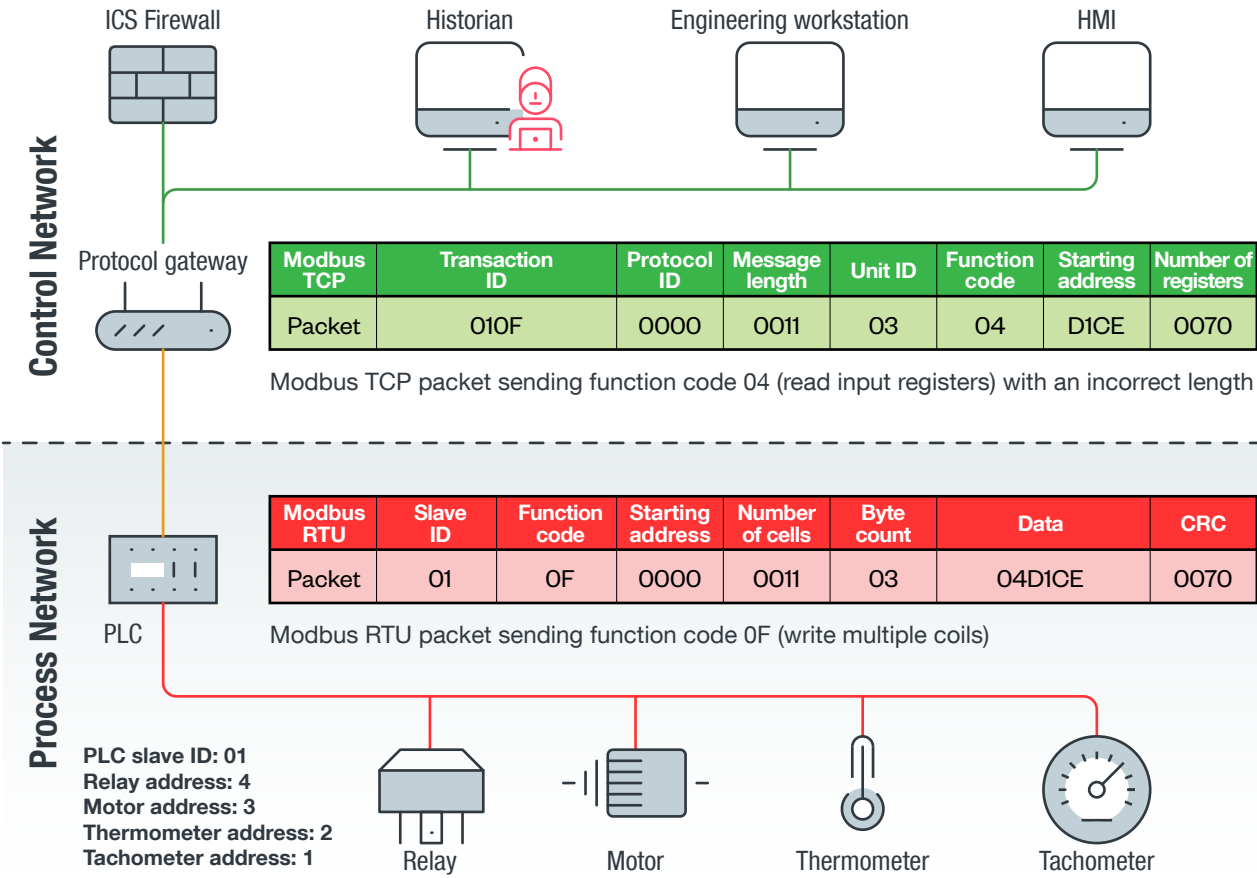


Figure 11. Attacks that can make use messages with invalid length

In this scenario, an attacker has gained access to the historian. A historian is a software program that keeps historical records of critical operating parameters such as machine temperature and RPM. Historians are mostly likely the part of systems to have connections to the corporate network, as the data it contains is used for operational and business decisions. This function makes historians useful targets for attackers who wish to jump from the corporate LAN to the OT network.

This vulnerability has been reported to Nexcom via the ZDI disclosure program on Feb. 10, 2020, and has been assigned ZDI-CAN-10485. Nexcom response stated that the NIO50 is considered an end-of-life product and would no longer receive any updates to fix this vulnerability. The other vulnerabilities affecting NIO50, which we detailed in the Cloud Support section, will also not be patched.

As far as we know, the NIO50 was still available for sale in early 2020. The NIO51, which directly replaced the NIO50, has not been tested for the same vulnerability.

Data Stations

So far, we have seen how an attacker can potentially leverage a translation vulnerability in one real-time gateway to perform arbitrary command conversions, from a legitimate-looking *read* message to a *write* request that would eventually affect the operation of an industrial process. But what about the second type of protocol gateways, data stations?

This section discusses how translation vulnerabilities in data stations are, to a certain extent, similar to the way it was described so far for real-time gateways, but also differs in the way the attack is mounted and its final impact. The data stations that we have analyzed do not allow a conversion from a read to a write command, but they interestingly rely on the I/O mapping table concept, which may lead to the conversion from a write coil to a write register. The translation of device ID also causes a loss in context that prevents an ICS firewall from protecting a particular device.

In this section, we present the analysis of two data stations: The MGate 5105 and DA10D. We used the same setup described in Figure 3.

I/O Mapping Table and Translation Routines

Unlike real-time gateways, data stations do not translate inbound to outbound protocols on the fly. Before a data station can be deployed, a field engineer would first need to configure a data station appropriately, so that it knows which coil, register, and command maps to which switch, sensor, or device on the outbound interface.

As mentioned earlier, the resulting mapping of coils, registers, and commands is known as an I/O mapping table.

Using the DA10D as an example, the I/O mapping table is configured via Red Lion’s management software Crimson version 3.1, which is the latest version as of this writing.²¹ This software is used across the majority of Red Lion’s suite of industrial products, and then updated to the Red Lion data station. Figure 12 is an example of how the Modbus master coils on the DA10D, configured for Modbus TCP, are mapped to the Modbus slave coils on the DA10D, which is configured for Modbus RTU. The DigitalCoilWrite gateway block containing four coils (000001 – 000004) on MBSlave (Modbus slave) is mapped to the MBMaster (Modbus master) coils (MBMaster.000001 – MBMaster.000004), an internal memory block representing an actual coil on the PLC connected to the data station. Sending a Modbus TCP write coil command to coil 1 on the DA10D will be received by the DigitalCoilWrite 000001, then sent to MBMaster.000001. That value will then be sent to the device connected to the PLC connected via Modbus RTU.

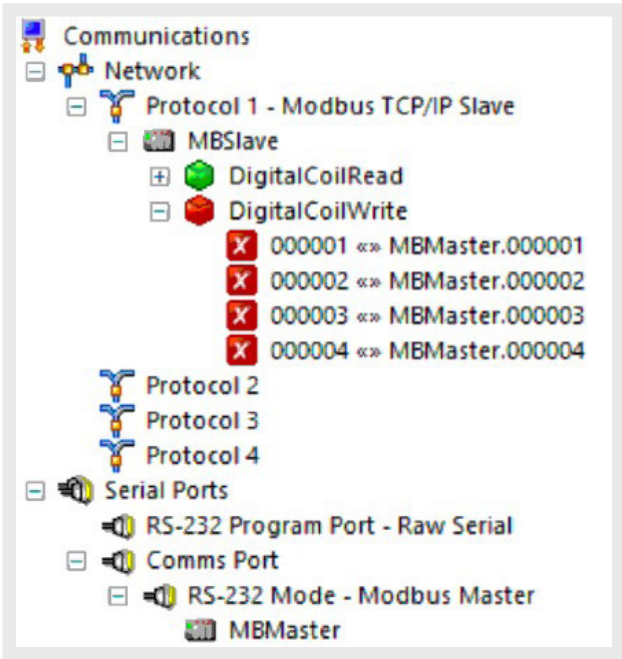


Figure 12. Example of I/O mapping table as defined by Red Lion’s Crimson software

The use of I/O mapping tables optimizes certain operations. For example, an incoming write request is first parsed and stored in the internal memory before being written to the target device. This asynchronous design enables some data stations to use the serial bus more efficiently, as several write coil requests to adjacent addresses can be compiled and issued to just one “write multiple coils.” Similarly, if a coil is switched off twice, the “switch off” command is issued only the first time.

With regards to read requests, the data station immediately replies to a read request by reading the values from the internal memory without the need to poll the values from the actual slave’s coils or registers. In order to keep the data in the internal memory synchronized to real-world values, a second routine periodically scans whether values are changed in the internal memory and generates one command – or a series of commands— to write the values to corresponding devices.

Figure 13 depicts the two routines operating concurrently in a data station and how they're used to keep the values in the I/O mapping table updated. This asynchronous operation made our security testing more complicated. In fact, as we mentioned already, we had to enable our framework to successfully correlate the outbound traffic of the data station with the inbound one, for example, when multiple write requests get aggregated in a single one, or when the messages do not get translated in order.

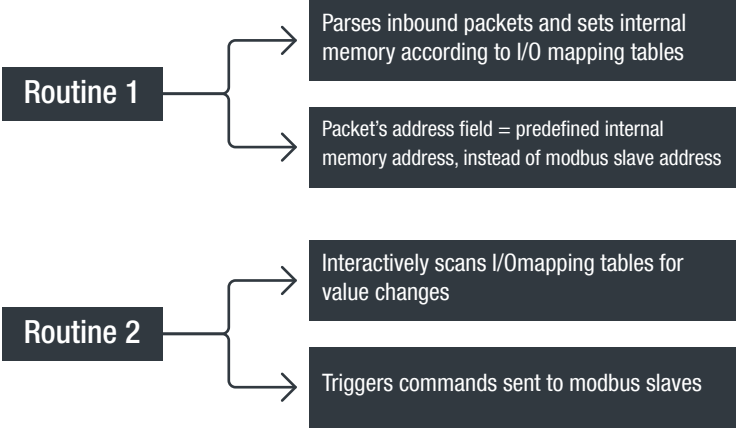


Figure 13. Diagram depicting the high-level routine of data stations

While I/O mapping tables offer the same functionalities, different products may adopt different design strategies. A good correlation is one with the routing tables of different network implementations. Unlike the previously shown table for DA10D, MGate 5105 adopts a two-stage design, as shown in Figure 14.

Mapping address arrangement Automatic

Your device :
Modbus TCP Client

write

write

Your device :
Modbus RTU/ASCII Slave

Role 1 of MGate 5105-MB-
EIP :
Modbus TCP Server

Name	Unit ID	TCP Port	Modbus Address
RelaySwitch	1	502	4x0001-4x0001
MotorSwitch	1	502	4x0001-4x0001
ThermSwitch	1	502	4x0002-4x0002
EnTachometer	1	502	4x0002-4x0002
SetRPM	1	502	4x0003-4x0003
SetDevID	1	502	4x0004-4x0004

Role 2 of MGate 5105-MB-
EIP :
Modbus RTU/ASCII Master

Name	Function	Internal Address	Quantity
RelaySwitch	5	0 .. 0	1 bytes
MotorSwitch	5	1 .. 1	1 bytes
ThermSwitch	5	2 .. 2	1 bytes
EnTachometer	5	3 .. 3	1 bytes
SetRPM	6	4 .. 5	2 bytes
SetDevID	6	6 .. 7	2 bytes

Figure 14. I/O mapping table on MGate 5105 (stage 1). Maps requests to internal memory addresses.

The table on the right side of Figure 14 specifies which “command” needs to be executed in case of a change in any of the internal addresses defined in the rows. The table can be read as:

- Row 1: Any changes to the value at Internal Address 0, trigger command “RelaySwitch”
- Row 2: Any changes to the value at Internal Address 1, trigger command “MotorSwitch”
- Row 3: Any changes to the value at Internal Address 2, trigger command “ThermSwitch”
- Row 4: Any changes to the value at Internal Address 3, trigger command “EnTachometer”
- Row 5: Any changes to the value at Internal Address 4-5, trigger command “SetRPM”
- Row 6: Any changes to the value at Internal Address 6-7, trigger command “SetDevID”

The table on the left reports how these commands map to Modbus TCP addresses. This table only serves as a reference and has no actual impact on address translation. In fact, the address assigned in Modbus TCP’s PDU is the internal memory address on MGate 5105, causing a special vulnerability. The section Arbitrary R/W Vulnerability (MGate 5150) will cover this in-depth.

Figure 15 shows stage 2 of Mgate 5105’s mapping table. This table defines which slave ID, function code, address, and the number of coils/registers to request for each “command” when a specific command is triggered.

For example, a Modbus TCP request “write coil to slave 1 (= data station) address 0 turn ON” will change the value of the internal address 0 (internal address begins from zero). Therefore, RelaySwitch will be triggered, and the Modbus RTU request “write coil (function 5) to slave 2 address 1 turn ON” will be made. This way, the translation from Modbus TCP to Modbus RTU is completed.

Index	Name	Slave ID	Function	Address/quantity	Trigger	Poll interval	Endian swap
1	RelaySwitch	2	5	Write address 1, Quantity 1	Data change	N/A	None
2	MotorSwitch	3	5	Write address 101, Quantity 1	Data change	N/A	None
3	ThermSwitch	4	5	Write address 1, Quantity 1	Data change	N/A	None
4	EnTechometer	5	5	Write address 1, Quantity 1	Data change	N/A	None
5	SetRPM	3	6	Write address 401, Quantity 1	Data change	N/A	None
6	SetDevID	4	6	Write address 101, Quantity 1	Data change	N/A	None

Figure 15. (stage 2) I/O mapping table on address 1 turn ON will be made. This way, the translation from Modbus TCP to Modbus RTU is completed.

Malicious Extraction of the I/O Mapping Table

The I/O mapping table contains confidential information. In fact, should hackers gain access to it, they can then derive contextual information they can use to formulate more targeted attacks, such as identify the ones listed here.

- Coils to write to shut down a motor
- Holding register to write to override a temperature threshold
- Holding register to write to slow down a centrifuge
- Coils to write to reverse a conveyor belt

Therefore, I/O mapping tables can be a crucial source of information during the attack development and tuning phase and may provide the key piece of information an attacker is looking for to bring the facility down. In addition, any unauthorized modification to the I/O mapping table will tamper with the operation of the HMI, PLCs, and devices connected to the data station.

The two data station gateways we looked at had some security measures in place to protect the I/O mapping table from unauthorized access. However, we found the implementation of the security measures to be weak. We will discuss this in the next subsections.

Credential Re-use and Decryptable Configuration (MGate 5105)

The MGate 5105 protects the I/O mapping table from unauthorized access, so a malicious actor would need to obtain valid login credentials to read the table. To the best of our knowledge, there are no hardcoded credentials.

Although the default configuration enables HTTP and Telnet (i.e., the password is transmitted in clear text), the field engineer can disable them or use HTTPS. They can even upload an X.509 certificate to make the HTTPS connection resistant to man-in-the-middle (MiTM) attacks.

However, we have found multiple strategies to overcome these limitations for malicious attackers. One option is to use the credential re-use attack described in the section, Credential Reuse and Decryptable Configuration, while another one is the Privilege Escalation vulnerability discussed in the Privilege Escalation section. After dumping the configuration or gaining access to the root shell, an attacker would be able to read the I/O mapping table stored in a SQLite3 database.

Arbitrary R/W Vulnerability (MGate 5150)

As we said, an attacker that gains access to the I/O mapping table will have complete visibility over the ecosystem of the data station, enabling them to conduct more targeted and precise attacks. Throughout this section, we rely on a common scenario in which a temperature control system is used in production and show how an attacker can negatively target it.

Figure 16 shows a simple HMI for monitoring a temperature reading (shown on display) and a critical threshold temperature that should not be changed. The system contains a PLC that reads the current temperature from a thermometer, stores the critical temperature in a holding register, and runs a ladder logic that turns on both alarm and emergency cooling procedures when the current temperature exceeds the critical temperature. This is a normal setup in production environments where the temperature could negatively affect production and should be constantly kept under control.

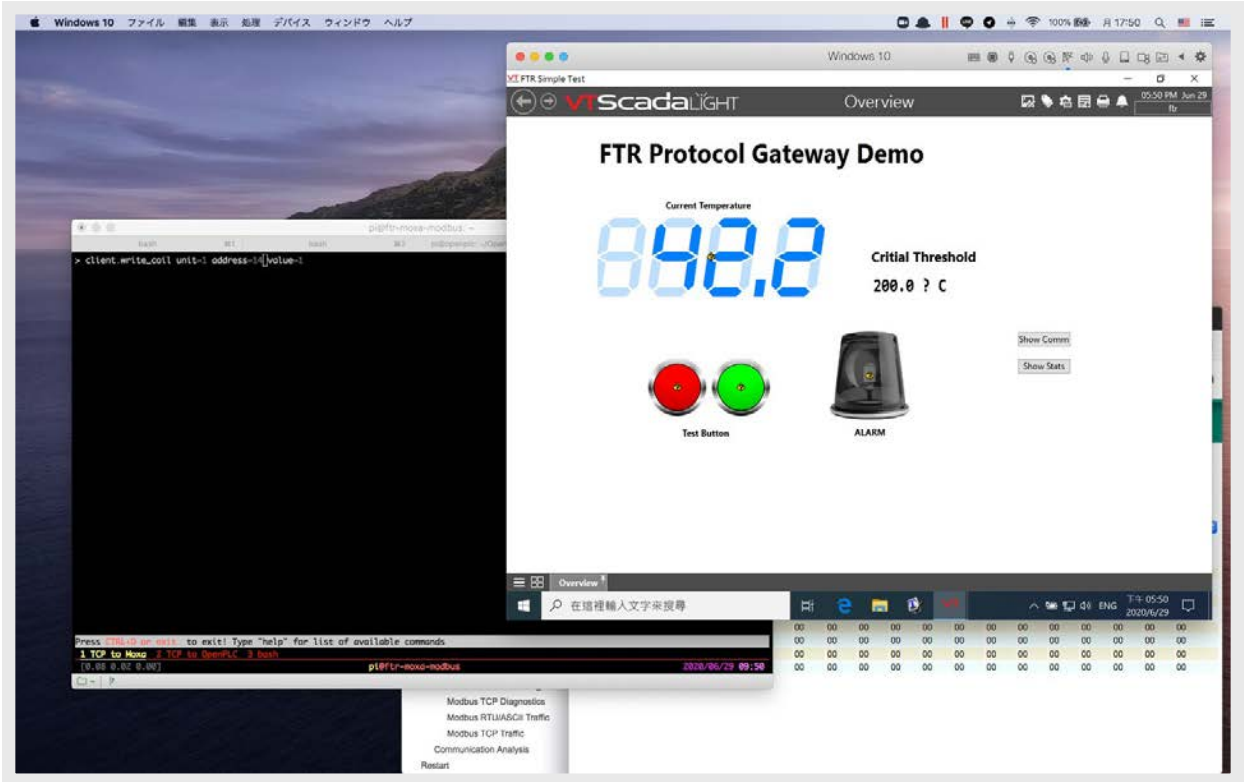


Figure 16. An example of HMI with one switch, one alarm and two temperatures (current and critical)

As a safety feature, the HMI provides a manual way to test if the alarm is operational through periodic checks to make sure it is functioning or needs repair. When a human operator clicks on the Red Test Button (defined as SetSW1), the HMI produce a “write a single coil” request on Modbus TCP (05 00 00 FF 00, function 5, address 0, ON [FF00], the address starts at zero). The data station parses the request, changes the internal memory (set address 0 bit 0 = 1, as defined in the I/O mapping table below), and responds with “OK” to the HMI.

Meanwhile, since the value of this address was changed, SetSW1 is triggered, and the MGate 5105 translates the command as a Modbus RTU request 05 00 01 FF 00 DD FA to turn on SW1 in PLC. Notice that the address begins from 1, depending on the PLC, and DD FA is the checksum. When the PLC receives the request, it responds with “OK” back to the MGate 5105, and the ladder logic would turn on the Alarm light. As shown on both the HMI and the I/O mapping table, there is only one switch in the system, SW1.

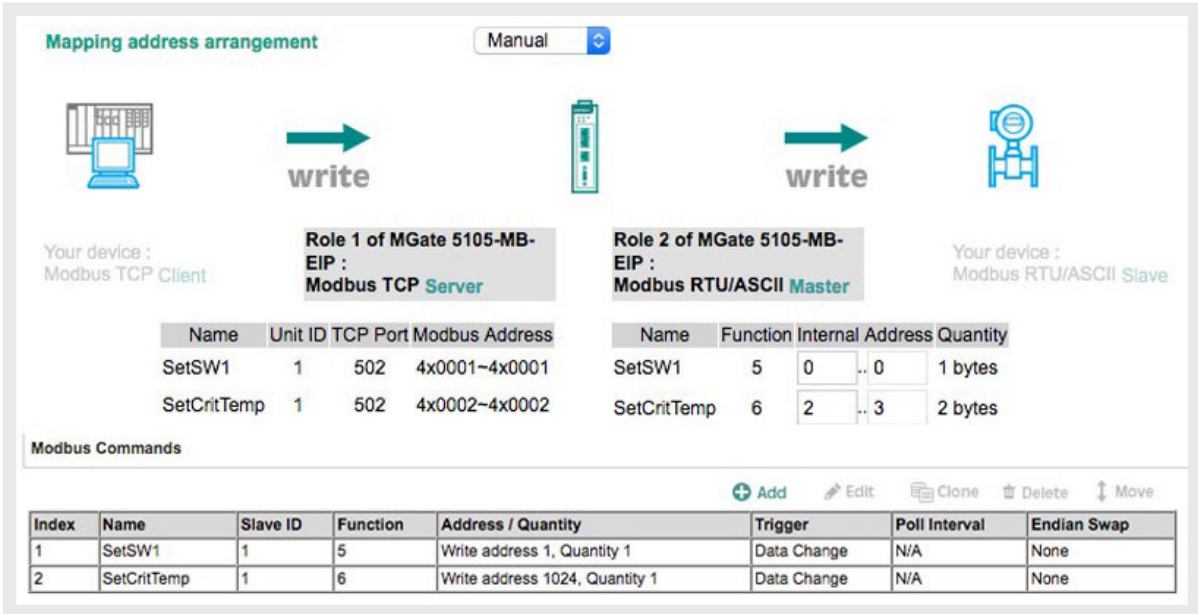


Figure 17. I/O mapping table used to test the arbitrary R/W vulnerability

According to the Modbus specification, a coil is represented by one bit: 1 means on and 0 means off. Therefore, one byte of internal memory can host eight coils. In other words, the internal address 0 in our example can serve eight potential switches.

SW8	SW7	SW6	SW5	SW4	SW3	SW2	SW1
OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON

Figure 18. An illustration showing how a single internal address can serve eight switches

We found a design issue in this data station, which the vendors have acknowledged. Since we have only defined one switch (SW1), only SW1 should be turned on and off. If a hacker wants to modify or manipulate a hypothetical non-existing switch SW5, we believe that the system should return a “switch not found” error. However, the MGate 5105 accepts the command to turn on SW5, causing the internal memory to be changed to 0001 0001 (i.e., from 0x00 to 0x11).

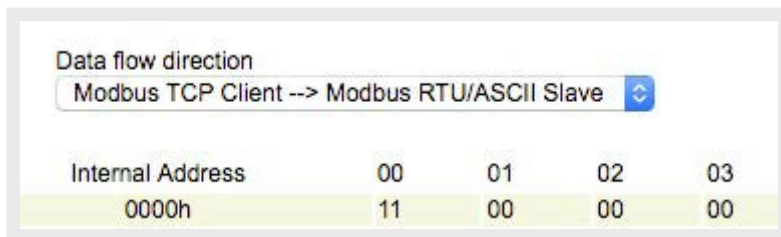


Figure 19. Internal memory 0000h changed from 0x01 to 0x11, as a non-existing SW5 is turned on

As a result, internal memory address 0 is changed, thus triggering SetSW1 again. It also switches SW1 to on again, despite it already being on.

This may not cause any issue in a real-world setting, because a real switch cannot be turned ON twice. However, a logical switch inside a PLC might work differently, depending on the actual design of the ladder logic. Our concern for such a scenario is the implication that a hacker could set any arbitrary bit to 1 in the internal memory.

In this scenario, the holding register for SetCritTemp is mapped to the internal address 2 to 3 (as shown in Table 6), and represents the critical temperature threshold that should not be changed by an authorized person or system. A secure OT network setup will have an ICS firewall that blocks all “write register” requests, unless the IP that issues the request belongs to the principal engineer or authorized HMI. Therefore, a command like this one should have been blocked.

Transaction ID	Protocol ID	Message length	Unit ID	Function code	Register address	Register value
0001	0000	0006	01	06	0001	47D0

Table 6. How a hacker might change the critical threshold by Function code 6 (write a single register): Command translates to set a single register on address 0001 to value 47D0, wherein address 0001 stores the critical temperature value in memory

If the original critical temperature is 200.0°C (0x07D0 in hex equals to decimal 2000), this command will set it to 1838.4°C (0x47D0). This command should be blocked unless the principal engineer is really sure of what they are doing when changing these values. However, we have found an arbitrary read/write vulnerability, through which a hacker can still write the register by flipping the non-existing switches.

When the value of internal address 2 to 3 is changed, a Modbus RTU request is sent to slave ID 1 function 6 (write a register) address 1024, quantity 1. We dumped the I/O mapping table and therefore knew that threat actors could take advantage of the table by changing the value of address 2 to 3. A hacker can therefore bypass the ICS firewall and issue this request:

Transaction ID	Protocol ID	Message length	Unit ID	Function code	Output address	Output value
0001	0000	0006	01	05	0016	FF00

Table 7. Command translates to set a single coil on address 0016 (22 in decimal, which is bit 23 in the internal address) to value FF00 (turns the switch ON; FF00 = ON, 0000 = OFF)

The Modbus TCP request in Table 7 is an example of how an attacker can maximize the attack in one request. We changed the leftmost bit of internal address 1 by turning on the non-existing switch 23, as illustrated in the table below. Since the leftmost bit of internal memory address 1 was set from 0 (off) to 1 (on), the value changed from 0x07D0 (200.0°C) to 0x47D0 (1838.4°C). In effect, MGate 5105 translated the “write coil” to “write register,” specifically 06 04 00 47 D0 (write register 1024 to 0x47D0), thus invalidating the safety threshold. The temperature can keep increasing to the point where a safety system interferes and starts the shutdown process.

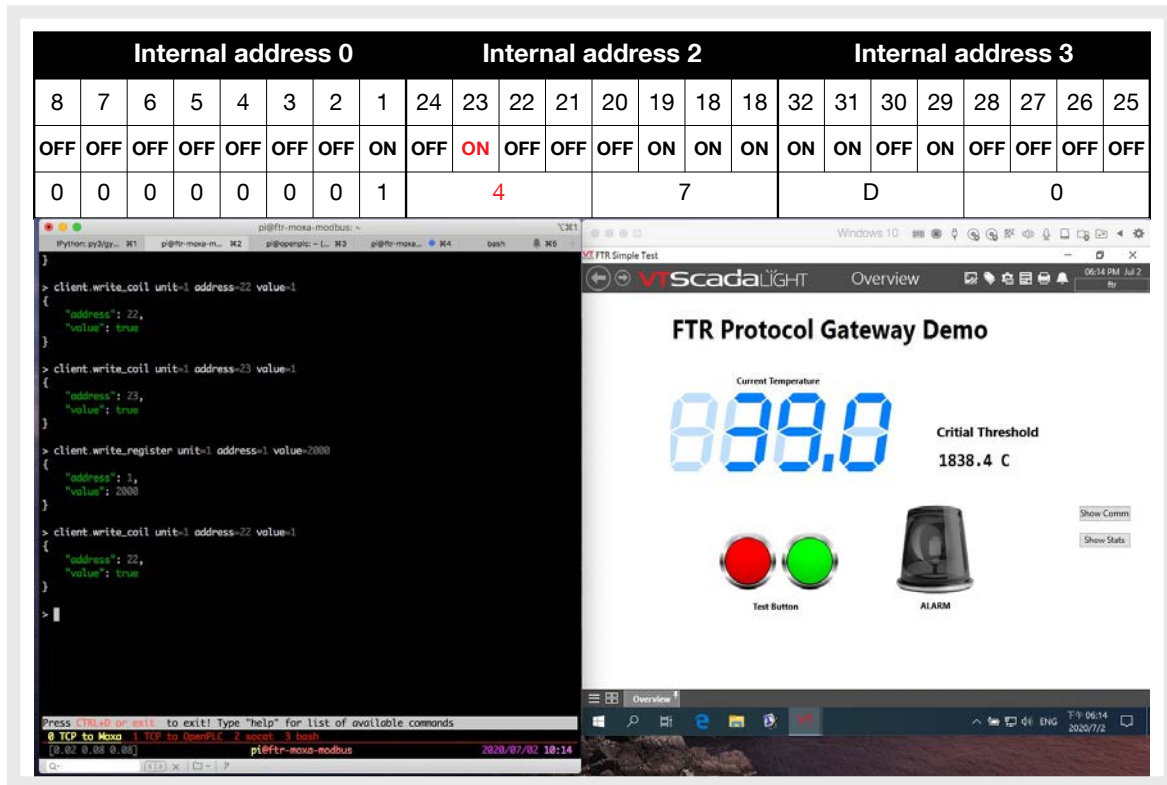


Figure 20. Images show how an attacker can modify the value of the critical temperature threshold. The bottom image shows that it is now at a very high 1,838.4 degrees Celsius.

To summarize, the vulnerability exists because of the following design issues:

- A non-existing switch should not be turned on (highlighted in red in the first image of Figure 20)
- Internal memory for coils should be different from the internal memory for registers

The other data station we looked at, the DA10D, separates the internal memory of coils from the internal memory of registers. It is thus not affected by this vulnerability.

We have reported to Moxa the possibility of their design being misused in this way. Moxa replied that it is working as designed.

Context Lost in Translation

Not all network configurations involve a protocol gateway and a PLC. In cases where a PLC is not needed, a protocol gateway can be directly connected via Modbus RTU to other serial devices, as illustrated in Figure 21.

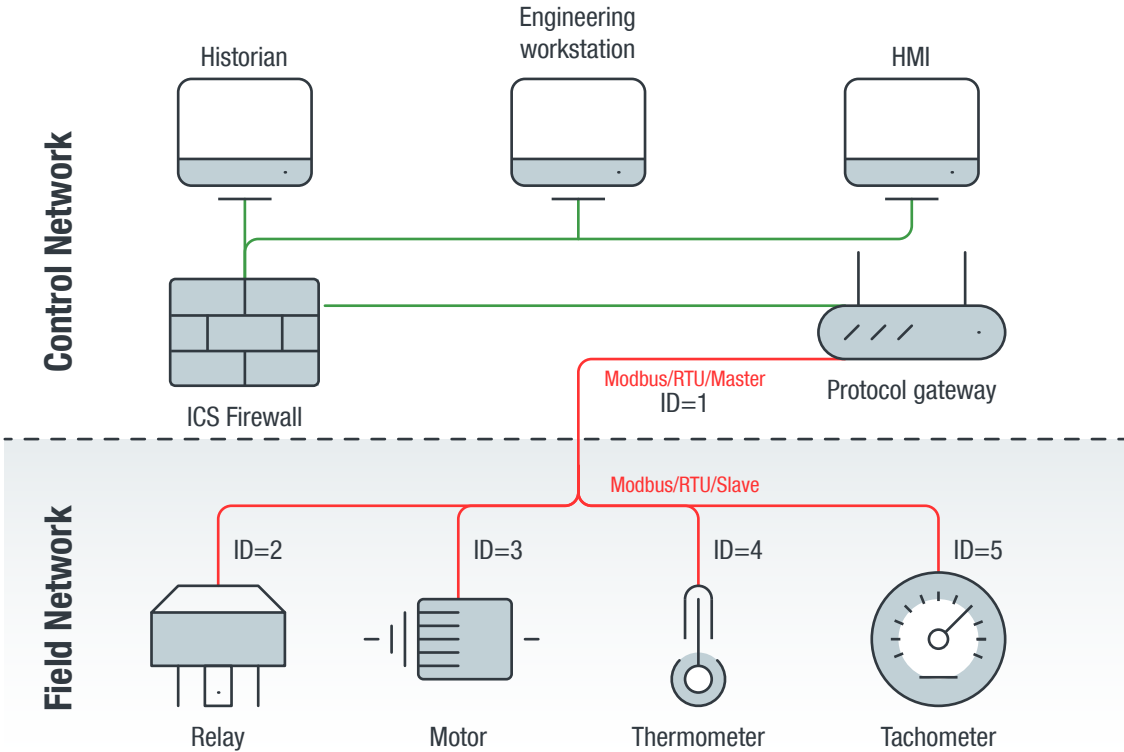


Figure 21. A protocol gateway is shown to be directly connected to Modbus RTU slaves

As Modbus TCP (green line) and Modbus RTU (red line) are two separated buses, all requests to Modbus RTU Slaves have to be written to the protocol gateway at ID=1 in Modbus TCP before they are translated and transmitted to Modbus RTU. Figure 22 is an example of I/O mapping table.

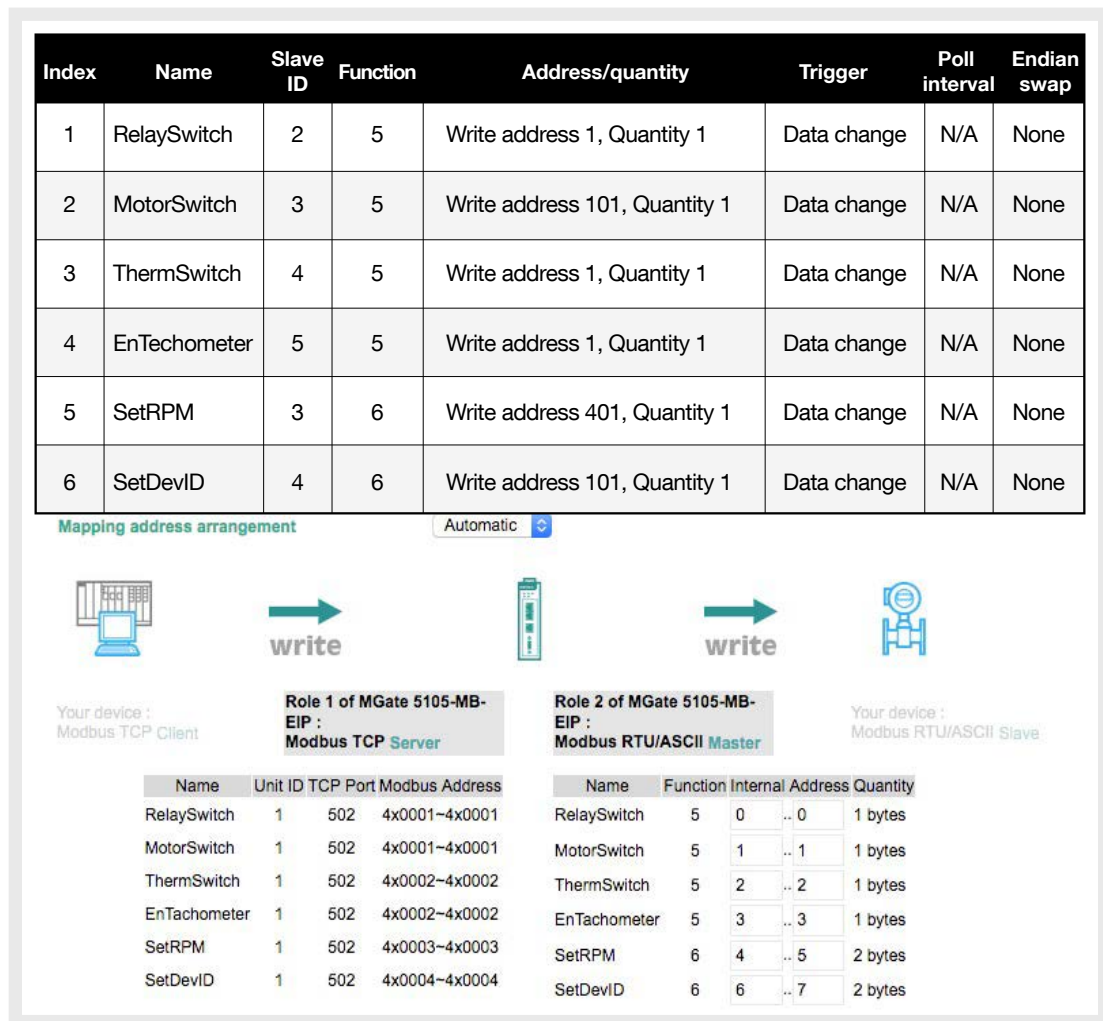


Figure 22. The I/O mapping table of a setup that connects multiple slaves to Modbus RTU

The functions of the settings are

- RelaySwitch: turns the relay on/off
- MotorSwitch: turns the motor on/off
- ThermSwitch: turns the thermometer on/off
- EnTachometer: turns the tachometer on/off
- SetRPM: sets the RPM of the motor
- SetDevID: sets the device ID of the thermometer

We placed several requests and compared the Unit ID before and after the translation.

Request	Before translation (TCP)	After translation (RTU w/o checksum)
Turn on the relay	01 05 00 00 FF 00	02 05 00 01 FF 00
Turn on the motor	01 05 00 08 FF 00	03 05 00 65 FF 00
Turn on the Thermometer	01 05 00 10 FF 00	04 05 00 01 FF 00
Set RPM to 1000	01 06 00 04 03 E8	03 06 01 91 03 E8

Table 8. A comparison of Unit ID (in bold) before and after the translation

Although we are using the I/O mapping table of the MGate 5105 as an example, the context of unit ID lost in translation is not limited to it. All requests are issued to unit ID 1 (the protocol gateway) and translated to different unit IDs. An ICS firewall is not aware of the I/O mapping table; therefore, it is not able to specifically prevent a special device from being requested.

Traffic Amplification

As the data stations asynchronously translate among protocols, it is possible to merge multiple “write single coil” requests into one “write multiple coils” request and vice versa to use the serial bus more efficiently. Take the example of Figure 23; when a hacker requests function 15 (write multiple coils), it will be translated to 1 writing to ID=2, 1 to ID=4, 1 to ID=5, 1 to ID=6. Therefore, one write on Modbus TCP becomes four writes on Modbus RTU, thus causing minor congestion on the serial bus.

```
TCP: 01 0F 00 00 00 30 03 FF FF FF ID=1, write 48 coils, addr=0, ON ON ON ON ...
RTU: 02 05 00 01 FF 00      ID=2, write single coil, addr=1, ON
RTU: 03 05 00 65 FF 00      ID=3, write single coil, addr=101, ON
RTU: 04 05 00 01 FF 00      ID=4, write single coil, addr=1, ON
RTU: 05 05 00 01 FF 00      ID=5, write single coil, addr=1, ON
```

Figure 23. Image depicting how one write on Modbus TCP becomes four writes on Modbus RTU

The attack above is not limited to the MGate 5105. However, the arbitrary R/W vulnerability in MGate 5105 could allow a hacker to cause more congestion on the serial bus by requesting “write multiple registers” with random values. That way, the internal memory is randomized, and all commands set in the I/O mapping table could be triggered. For example,

Function code	Starting address	Quantity of regs	Byte count	Reg values
10	00 00	00 7B	F6	11 22 33 44 55 66 77 88 ...

Table 9. Example values should a hacker request to write multiple registers

It should be noted that such amplification might not necessary cause a denial of service (DoS), however a congested RS-485 bus can still cause abnormal behaviors.

Device Vulnerabilities

In the previous sections, we showed how subtle flaws in the protocol translation’s implementation or design could result in significant translation vulnerabilities. Architectural errors, for example, in the design of a data station’s mapping table, can be leveraged to conduct stealthy attacks that are difficult to detect.

This section presents several new vulnerabilities that we discovered during our research. These four are privilege escalation, credential reuse, decryptable configuration, and memory leakage vulnerabilities. We further show how they could affect the ability of an attacker to mount a translation attack. In other words, the presence of these vulnerabilities will amplify the feasibility of a protocol translation attack, and largely increment the risk associated with the use of a protocol gateway as part of a larger attack.

Credential Reuse and Decryptable Configuration

Moxa uses a proprietary protocol when communicating with the remote management software called MGate Manager. When launching MGate Manager, a field engineer is prompted for their username and password to access the protocol gateway, after which MGate Manager automatically dumps the configuration so that the field engineer could change settings on the user interface. When the field engineer finishes setting the protocol gateway and clicks on “Exit,” the configuration is compressed, encrypted, and uploaded to the gateway.

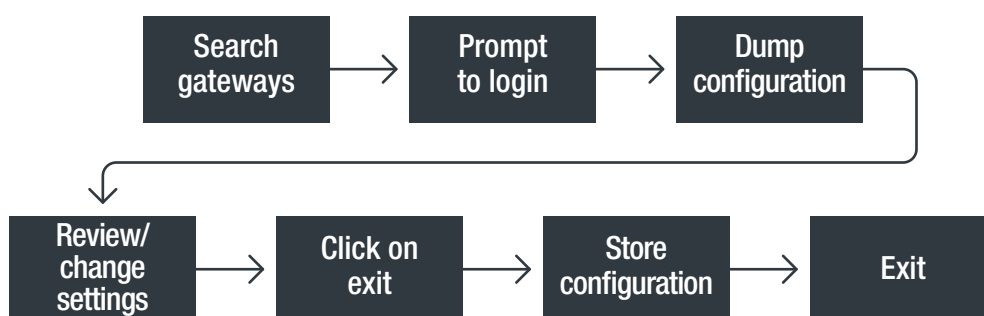


Figure 24. A summary of the steps for setting the Moxa protocol gateway

However, there are two security weaknesses in this procedure that can be abused:

1. **Credential Reuse:** When the field engineer is asked to log in, a cipher is transmitted to MGate Manager in order to hash the password, such that the password is not transmitted in cleartext. The cipher can be randomized by setting it to rand(), but the random seed must be set so that every time the internal service program in the protocol gateway restarts, the random ciphers are reused. The firmware version being tested did not set the random seed. As a result, the ciphers would look predictable from a hacker’s perspective. The service program restarts when the protocol gateway reboots and when a new configuration is uploaded. Therefore, the cipher is recycled each time the field engineer exits the user interface. A hacker can replay the same encrypted password to log in as the field engineer who usually has administrator privilege, without knowing the password in cleartext. This vulnerability was reported to Moxa via the ZDI disclosure program on March 18, 2020, and has been assigned CVE-2020-15493.²² Moxa has already released a patch on July 10, 2020, that addresses this vulnerability.²³
2. **Decryptable Configuration:** Even if the random seed is correctly set, the second vulnerability still allows us to dump the I/O mapping table. The encrypted configuration being transmitted over Ethernet contains the encryption key and thus can be decrypted. A configuration dump that a hacker intercepts and reconstructs from the control network looks like the configuration shown in Figure 25.

Offset	Hex	Label
00000000	01 00	Fixed value
00000002	60 e9 07 00	Length of the original file
00000004	44 45 42 41 32 42 45 45 32 35	Encrypted configuration
00000006	44 32 44	
00000008	46 42 45	
0000000a	39 46 37 36 34 37 30 31	
0000000c	31 36 37 45 45 32 31 38	
0000000e	33 39 34 38 36 32 30 37	
00000010	41 31 30 31 33 38 35 38	
00000012	35 31 34 37 33 37 43 33	
00000014	41 33 34 33 41 38 44 35	
00000016	44 45 42 41 32 42 45 45 32 35	
00000018	44 32 44	
0000001a	46 42 45	
0000001c	39 46 37 36 34 37 30 31	
0000001e	31 36 37 45 45 32 31 38	
00000020	33 39 34 38 36 32 30 37	
00000022	41 31 30 31 33 38 35 38	
00000024	35 31 34 37 33 37 43 33	
00000026	41 33 34 33 41 38 44 35	
00000028	44 45 42 41 32 42 45 45 32 35	
0000002a	44 32 44	
0000002c	46 42 45	
0000002e	39 46 37 36 34 37 30 31	
00000030	31 36 37 45 45 32 31 38	
00000032	33 39 34 38 36 32 30 37	
00000034	41 31 30 31 33 38 35 38	
00000036	35 31 34 37 33 37 43 33	
00000038	41 33 34 33 41 38 44 35	
0000003a	44 45 42 41 32 42 45 45 32 35	
0000003c	44 32 44	
0000003e	46 42 45	
00000040	39 46 37 36 34 37 30 31	
00000042	31 36 37 45 45 32 31 38	
00000044	33 39 34 38 36 32 30 37	
00000046	41 31 30 31 33 38 35 38	
00000048	35 31 34 37 33 37 43 33	
0000004a	41 33 34 33 41 38 44 35	
0000004c	44 45 42 41 32 42 45 45 32 35	
0000004e	44 32 44	
00000050	46 42 45	
00000052	39 46 37 36 34 37 30 31	
00000054	31 36 37 45 45 32 31 38	
00000056	33 39 34 38 36 32 30 37	
00000058	41 31 30 31 33 38 35 38	
0000005a	35 31 34 37 33 37 43 33	
0000005c	41 33 34 33 41 38 44 35	
0000005e	44 45 42 41 32 42 45 45 32 35	
00000060	44 32 44	
00000062	46 42 45	
00000064	39 46 37 36 34 37 30 31	
00000066	31 36 37 45 45 32 31 38	
00000068	33 39 34 38 36 32 30 37	
0000006a	41 31 30 31 33 38 35 38	
0000006c	35 31 34 37 33 37 43 33	
0000006e	41 33 34 33 41 38 44 35	
00000070	44 45 42 41 32 42 45 45 32 35	
00000072	44 32 44	
00000074	46 42 45	
00000076	39 46 37 36 34 37 30 31	
00000078	31 36 37 45 45 32 31 38	
0000007a	33 39 34 38 36 32 30 37	
0000007c	41 31 30 31 33 38 35 38	
0000007e	35 31 34 37 33 37 43 33	
00000080	41 33 34 33 41 38 44 35	
00000082	44 45 42 41 32 42 45 45 32 35	
00000084	44 32 44	
00000086	46 42 45	
00000088	39 46 37 36 34 37 30 31	
0000008a	31 36 37 45 45 32 31 38	
0000008c	33 39 34 38 36 32 30 37	
0000008e	41 31 30 31 33 38 35 38	
00000090	35 31 34 37 33 37 43 33	
00000092	41 33 34 33 41 38 44 35	
00000094	44 45 42 41 32 42 45 45 32 35	
00000096	44 32 44	
00000098	46 42 45	
0000009a	39 46 37 36 34 37 30 31	
0000009c	31 36 37 45 45 32 31 38	
0000009e	33 39 34 38 36 32 30 37	
000000a0	41 31 30 31 33 38 35 38	
000000a2	35 31 34 37 33 37 43 33	
000000a4	41 33 34 33 41 38 44 35	
000000a6	44 45 42 41 32 42 45 45 32 35	
000000a8	44 32 44	
000000aa	46 42 45	
000000ac	39 46 37 36 34 37 30 31	
000000ae	31 36 37 45 45 32 31 38	
000000b0	33 39 34 38 36 32 30 37	
000000b2	41 31 30 31 33 38 35 38	
000000b4	35 31 34 37 33 37 43 33	
000000b6	41 33 34 33 41 38 44 35	
000000b8	44 45 42 41 32 42 45 45 32 35	
000000ba	44 32 44	
000000bc	46 42 45	
000000be	39 46 37 36 34 37 30 31	
000000c0	31 36 37 45 45 32 31 38	
000000c2	33 39 34 38 36 32 30 37	
000000c4	41 31 30 31 33 38 35 38	
000000c6	35 31 34 37 33 37 43 33	
000000c8	41 33 34 33 41 38 44 35	
000000ca	44 45 42 41 32 42 45 45 32 35	
000000cc	44 32 44	
000000ce	46 42 45	
000000d0	39 46 37 36 34 37 30 31	
000000d2	31 36 37 45 45 32 31 38	
000000d4	33 39 34 38 36 32 30 37	
000000d6	41 31 30 31 33 38 35 38	
000000d8	35 31 34 37 33 37 43 33	
000000da	41 33 34 33 41 38 44 35	
000000dc	44 45 42 41 32 42 45 45 32 35	
000000de	44 32 44	
000000e0	46 42 45	
000000e2	39 46 37 36 34 37 30 31	
000000e4	31 36 37 45 45 32 31 38	
000000e6	33 39 34 38 36 32 30 37	
000000e8	41 31 30 31 33 38 35 38	
000000ea	35 31 34 37 33 37 43 33	
000000ec	41 33 34 33 41 38 44 35	
000000ee	44 45 42 41 32 42 45 45 32 35	
000000f0	44 32 44	
000000f2	46 42 45	
000000f4	39 46 37 36 34 37 30 31	
000000f6	31 36 37 45 45 32 31 38	
000000f8	33 39 34 38 36 32 30 37	
000000fa	41 31 30 31 33 38 35 38	
000000fc	35 31 34 37 33 37 43 33	
000000fe	41 33 34 33 41 38 44 35	
00000100	44 45 42 41 32 42 45 45 32 35	

Figure 25. The encrypted configuration contains the AES key when being transmitted. This is akin to sending a password-protected Zip file, but the password is in the filename.

For the decryption, we leveraged the proprietary decryption library that we extracted from the device’s firmware, which we obtained online.²⁴ The configuration contains configuration files, databases, and a Secure Shell (SSH) key. Below is an example of a decrypted configuration that we “intercepted” from our own protocol gateway.

```
etc key

./etc:
devsvr      ieg_system.log localtime      passwd      shadow
ethCert.pem ieg_system_idx loginLog.conf resolv.conf  snmpd.conf
gsd         iptable.allow  network      server.pem   snmpdv3.conf

./etc/devsvr:
aliyun.db3  cloud.db3      eip.db3       modbus_tcp.db3 proto_cfg.dat
azure.db3   cmap.db3       modbus_ser.db3 mqtt_common.db3 system.db3

./etc/gsd:
slave
```

Figure 26. A successfully decrypted configuration file

The decrypted configuration contains the protocol gateway’s private RSA keys and several databases. The databases are not encrypted and can be dumped and modified with SQLite3. The I/O mapping table, user table (password is hashed), Modbus configurations, and even cloud configurations (such as Azure connect string) and MQTT credentials are all stored in the dump.

To summarize, a malicious actor would be able to conduct a replay attack to the MGate Manager and decrypt the system configuration, including the I/O table, if they have access to the network between the MGate Manager and the MGate 5105. Moreover, it is possible to add an admin account by changing system.db3, then repack, compress, encrypt and upload the configuration by reusing the credential. Once the credential reuse vulnerability is fixed, it is still possible to intercept and hijack the uploading process from MGate Manager if the hacker can run a program on the network switch or the ICS firewall.

We have a successful proof of concept that simulates such an attack using a Raspberry Pi between MGate Manager and the MGate 5105.

This vulnerability was reported to Moxa via the ZDI disclosure program last March 18, 2020, and has been assigned CVE-2020-15494.²⁵ Moxa has already released a patch on July 10, 2020, that addresses this vulnerability.²⁶

Privilege Escalation

We found that the MGate 5105-MB-EIP is vulnerable to a privilege escalation vulnerability that was communicated to and patched by the vendor under CVE-2020-8858.

The vulnerability allows an unprivileged user to execute privileged commands due to an unfiltered input within the diagnostic functionality offered by the device’s web interface, as shown in Figure 27. As a result, an unprivileged user can launch a Telnet daemon in the context of the root user via a simple HTTP GET request. This allows the unprivileged user to gain full remote access (i.e., a root shell) to the device.

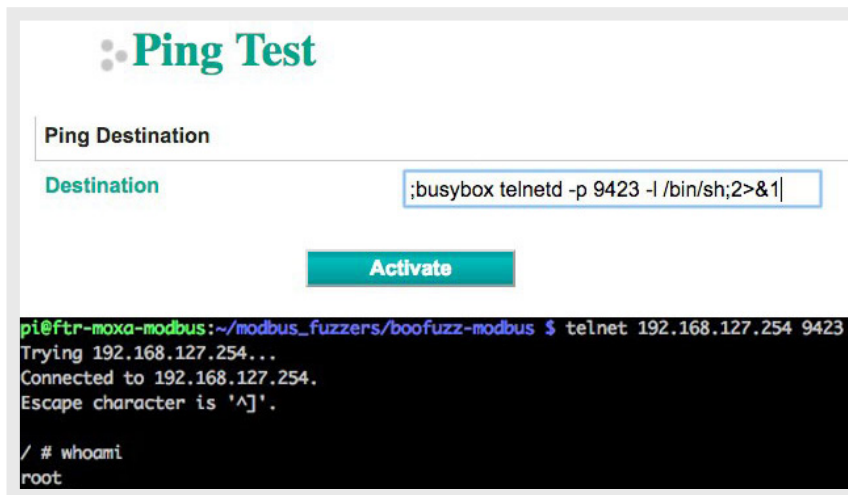


Figure 27. Post-authentication privilege escalation on MgATE 5105 (CVE-2020-8858)

This vulnerability was reported to Moxa via the ZDI disclosure program on Oct. 14, 2019, and has been assigned CVE-2020-8858.²⁷ Moxa has since fixed the issue and released a patch.

Memory Leakage

During testing, we also found that DA10D leaks memory contents whenever:

- The data station is configured to convert between Modbus TCP and Modbus RTU
- It receives a “Write Multiple Registers” message (function code 16) command having the byte count field set to 0.

In this evaluation, the data station was set up as a Modbus TCP Slave on Protocol 1, using the Ethernet interface, and as a Modbus Master on RS232. The Modbus Slave was set up with two gateway blocks — one for reading Holding Registers and one for writing Holding Registers. On the DA10D, gateway blocks represent the different types of data, such as coils, registers, digital inputs, and others, that are being translated.

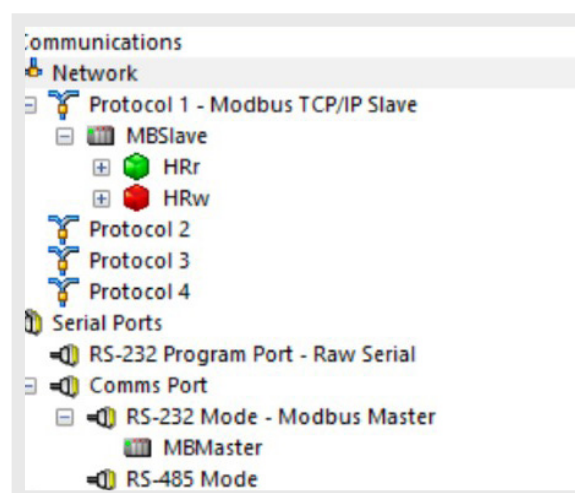


Figure 28. DA10D settings vulnerable to data leakage

A normal “Write Multiple Registers” payload (Function Code 16) looks like the following:

Modbus TCP	Slave ID	Function code	Starting address	Number of registers	Byte count	Data	
Packet	01	10	0000	0002	04	1234	4321

The attack message triggering the vulnerability looks like the following:

Modbus TCP	Slave ID	Function code	Starting address	Number of registers	Byte count	Data	
Packet	01	10	0000	0008	00		

Table 10. A comparison of normal and attack “write multiple registers” payload. In the attack version, the number of registers to write is not 0 (x0008), and the number of data bytes to follow is 0 (x00).

The problem occurs when the number of registers to write is between x0001 and x0008, and the number of data bytes is set to x00. When this occurs, the DA10D leaks memory data in the form of write register requests being sent to the Modbus RTU Slave (one or more, with the sum of the data written equal to the data being leaked as per attack message). The data being leaked is a data tag called HoldingRegister.hr40563. At that point, the DA10D automatically reads the data back to sync its internal mapping registers with the values that have been written on the slave. The attacker accesses the leaked data via read messages sent to the DA10D.

From our analysis, we can confirm that the amount of the memory data leaked is equal to the value specified in the “number of registers to write” field multiplied by two. The maximum amount of data that can be leaked at once seems to be 16 bytes.

The address of the memory data that is leaked is derived from the “starting address” field and is predictable. As a result, an attacker could leak arbitrary memory locations, including configuration and code.

The problem occurs in write multiple coils messages as well (function code 15). This vulnerability was reported to Red Lion via the ZDI disclosure program on April 5, 2020, and has been assigned ZDI-CAN-10897. Disclosed in line with the ZDI guidelines, Red Lion is currently working on a fix for this vulnerability.

Denial of Service

Protocol gateways play a crucial role in the network connectivity of industries, and a fault in one of these devices, such as one caused by a DoS attack, would affect the operation of a factory or a similar industrial facility.

Using the initial metaphor of language translators: A failure by protocol gateways is similar to a translator being unable to keep up with a speaker because the speaker is too fast or too difficult to understand, preventing the translator from translating correctly.

To answer our concerns on the ability of these embedded devices to handle handling large chunks of network streams, we instructed our fuzzer to generate, among others, large or complex packets. These packets would eventually cause out-of-bounds reads or trigger resource exhaustion conditions for protocol gateways. In this operation mode, the fuzzer monitors the gateway for its status (e.g. if the device is still operating properly), and communicates with the analyzer in case of an error.

As a result of this experiment, all three real-time gateways exhibited resource exhaustion problems. On the Digi One, NIO50 and Link150, the protocol translation service stopped working after the fuzzer sent about 100 packets, 2,000 packets, and 3,000 packets, respectively (at a time interval of 0.5 seconds between packets). Interestingly, all three devices kept powered on, suggesting that the resource exhaustion only affected the translation process.

While we were able to reproduce the problems consistently, we also noticed that the number of packets triggering the DoS was, to a certain extent, variable. This is an expected and normal behavior for this class of vulnerability, even though we could not conduct more exhaustive debugging due to the lack of logs.[†] As an aside, it is interesting to note that the lack of logs is a general limitation for embedded devices and complicates forensic activities in real-world attacks.

[†] Logs were collected either via serial port (RS232) or JTAG port, depending on the gateway. Note that most of the devices did not have such ports directly exposed, therefore we had to manually expose them by tapping to wire, where possible.

However, DoS vulnerabilities are not limited to real-time protocol gateways. DA10D was confirmed to be vulnerable to DoS attacks where an attacker can remotely trigger a device reboot by sending specially crafted malicious Modbus TCP packets, in particular:

- Packets with function code equal to 1 (read coils) and the number of coils to be read is zero
- Packets with function code equal to 2 (read discrete inputs) and the number of inputs to be read is zero
- Packets with function code equal to 3 (read holding registers) and the address of the first register to read is equal to zero
- Packets with function code equal to 4 (read input registers) and the address of the first register to read is equal to zero

This vulnerability is caused by a lack of sanitization of the inbound packets whose values are erroneously stored in the internal I/O mapping table. As a result, the data station reads invalid memory addresses when accessing the table in the act of retrieving updated data from the serial bus. This causes a crash in the code running at kernel space and, consequently, a reboot of the device. An attacker can use this vulnerability to reboot a targeted device repeatedly.

This vulnerability was reported to Red Lion via the ZDI disclosure program on March 23, 2020, and has been assigned ZDI-CAN-10804. Disclosed in line with the ZDI guidelines, Red Lion is currently working on a fix for this vulnerability.

Cloud Support

Three of the devices we tested feature support for cloud interfaces, either via MQTT protocol or cloud-specific APIs like Amazon AWS or Microsoft Azure. With the advent of Industry 4.0, remote devices can be controlled or monitored through the cloud. For example, a control server can use MQTT to push requests to the internet and a protocol gateway to deliver the commands to a PLC. The communication can also be reversed with PLCs collecting sensor data and uploading it to the cloud for storage in databases or post-processing.

The NIO50 offers MQTT translation support in the form of Modbus TCP (or RTU) to MQTT. In other words, this protocol gateway can be used to upstream Modbus data to the cloud, such as commands produced by a control server. MQTT is a lightweight messaging protocol relying on the publish-subscribe design pattern. When the translation from Modbus to MQTT is enabled in the gateway, the device subscribes itself to the configured MQTT broker. At that point, all inbound Modbus commands and data are translated and forwarded to the broker. During our evaluation, we identified the following security flaws that we reported to the vendor and are currently under responsible disclosure:

- The gateway does not support encryption, such as TLS/SSL (Transport Layer Security/Secure Sockets Layer). There is no option to enable a form of encryption, so it always forwards data in cleartext. As a result, an adversary such as an insider would be able to access unauthorized and private information, including usernames and passwords, via sniffing. This is even more true when the data upstream is performed via the wireless interface, as is often the case in real-world installations with legacy devices that were remotely distributed and then connected to ethernet networks via protocol gateways.

This vulnerability was reported to Nexcom via the ZDI disclosure program on Feb. 10, 2020, and has been assigned ZDI-CAN-10486.

- The gateway always transmits a null username (0x00000000), even when a login username is configured via the web console. As a result, an attacker can configure a rogue MQTT broker and, by enabling “anonymous login” (or disabling authentication), intercept all traffic translated by a targeted gateway, violating privacy and confidentiality.

This vulnerability was reported to Nexcom via the ZDI disclosure program on Feb. 10, 2020, and has been assigned ZDI-CAN-10487.

- The gateway does not validate the input before forwarding it. In a translation from Modbus TCP to MQTT, correct handling consists of verifying that the messages to be upstreamed complies with the modbus specifications, which means the modbus payload holds valid function codes and messages (e.g., function 0x01 for “read coil”). Instead, this gateway up-streams any messages, allowing an adversary to inject malicious payloads that can potentially trigger vulnerabilities exposed in the backend, such as the service collecting the traffic received from the gateway via MQTT. As a proof-of-concept example, we have been able to trigger an SQLi vulnerability using the protocol gateway’s translation as the attack vector.

This vulnerability was reported to Nexcom via the ZDI disclosure program on Feb. 10, 2020, and has been assigned ZDI-CAN-10487.

As mentioned earlier on the NIO50’s protocol translation vulnerability, the NIO50 is considered an end-of-life product, and Nexcom will no longer release a fix for the authentication and MQTT vulnerabilities.

The DA10D supports MQTT-over-TLS. However, this configuration is not enabled by default when using Generic MQTT and Sparkplug MQTT. The default configuration transmits usernames and passwords (together with other sensitive information like data payloads) in cleartext.

Other Findings

Up to this point, we have shown how erroneous design or implementation translations can open the door to advanced and difficult-to-detect attacks. While conducting our research on protocol translation problems, we have encountered a series of additional problems that we believe are worth mentioning as they could expose devices to risks or easily abused.

- MGate offers the ability to change the default IP address of the Ethernet interface through the use of a magic packet, which any user can transmit to the gateway to request an IP change. While this feature is only enabled on the default IP within the first 600 seconds, an attacker can potentially abuse this feature to disrupt the gateway's functions.
- Modbus TCP employs a unit ID field to indicate the message recipient. This information is especially important when gateways are used to integrate networks, as they may include information of different peripherals or peripherals with different unit IDs. While the Modbus specifications reserve one byte for this field (0-255), some vendors adopt a different implementation. For example, if a packet with a unit ID is greater than 127 is sent to the Digi One, the gateway will subtract 127 from the unit ID. As a result, a message sent to a device with unit ID 128 will actually be routed to a device with unit ID 1. This could create inter-communication problems and potential faults, such as packet losses or overflows.
- The MGate 5105 uses a hardcoded symmetric password to decrypt the firmware. A check revealed that this hardcoded password is not used across multiple Moxa devices. The hardcoded password allows an attacker to decrypt the firmware and conduct a thorough study of the firmware.

Impact

Protocol gateways, by their nature, are deployed in industrial environments where critical data and instructions need to be relayed in a timely and proper manner. This does not only ensure the continuous operation of the facility; it also makes sure the final product is of the intended quality.

In these applications, a temperature threshold is not just an arbitrary number. A temperature threshold may have several implications. Consider these examples:

- The necessary temperature for a manufacturing process, such as plastic extrusion: If the temperature is too high, the plastic polymers will break down. If the temperature is too low, the plastic will not extrude at all.
- The ideal temperature range for a food production process, like canning: If the temperature is below the threshold, microorganisms in the food product will not be destroyed, leading to food safety and health issues.

Therefore, in order to ensure a facility's continuous operation and product quality, a field engineer in an industrial environment would need to be able to see the data, trust that the data is correct, and in certain cases (such as the temperature exceeding or failing to reach the threshold), be able to take action.

An adversary's objective, on the other hand, is to either steal confidential or proprietary information, or sabotage the operation. If an attacker's objective is to sabotage the operation, the attacker can do so by compromising the integrity of the reported data, the operators' ability to view data, and their ability to take action.

To further explain how the vulnerabilities and security weaknesses we discovered in protocol gateways affect an industrial environment, we mapped the scenarios to MITRE's ATT&CK for Industrial Control Systems ICS²⁸ and corresponding impact.

In all, we identified four ways an insecure protocol translation or protocol gateway could affect a facility, namely – denial of view, denial of control, manipulation of view, and manipulation of control.

Below are the impact definitions, based on MITRE.

Denial of View²⁹

Malicious attackers may cause a denial of view to disrupt and prevent operator oversight on the true status of an ICS environment. This scenario typically results in temporary disruption or communication failure between a device and its control, which eventually recovers once the interference ceases.

Attackers may also attempt to deny visibility by preventing status reports and messages from reaching operators. Such actions will leave operators unable to notice changes and anomalous behavior in the system. In such a scenario, the environment's data and processes could still be operational, but functioning in an unintended or adversarial manner.

Denial of Control³⁰

Malicious attackers may temporarily prevent operators and engineers from interacting with process controls. In such a scenario, operators would be denied access to process controls, causing a temporary loss of communication with the control device.

Manipulation of View³¹

Attackers may attempt to manipulate the information that operators or controllers receive from machinery and sensors. The report an operator receives would not be reflective of the actual process.

Without the proper information, operators might make the wrong decisions and inappropriate sequences.

Manipulation of Control³²

Threat actors might manipulate physical process controls in the industrial environment. Methods of doing so can include making changes to setpoint values, tags, or other parameters. Threat actors can also manipulate control of system devices — or even leverage their own — to communicate with and command physical control processes. The manipulation can be temporary or sustained, depending on the time it takes for the operators to detect malicious activity.

Impact of Protocol Gateway Vulnerabilities

In this subsection, we map out how an attacker could use the vulnerabilities and security weaknesses we have discovered to result in a corresponding impact. We used the MITRE ATT&CK framework for Industrial Control Systems (ICS) to map out the specific technique the vulnerability or security weakness can be used for, and its corresponding impact.

Real-Time Protocol Translation Vulnerability (Nexcom NIO50)

The NIO50 packet forwarding vulnerability, where a read command can be translated into a write command, can be used by an attacker to issue a stealth unauthorized command message,³³ resulting in manipulation of control.

Due to the way the protocol translation works, it would make tracking down the offending system more difficult for a team doing incident response and mitigation.

ATT&CK for ICS Technique	ATT&CK for ICS Impact
Unauthorized command message	Manipulation of control

I/O Mapping Vulnerability (MGate 5105) and Malicious Extraction of I/O Mapping Table (MGate 5105 and DA10D)

I/O mapping is the core of protocol translation for data station protocol gateways. The ability to configure the protocol gateway with an I/O mapping table allows for increased versatility. However, we have detailed an attack chain for both the MGate 5105 and DA10D wherein an attacker can manipulate the I/O Image³⁴ to perform a surgical attack, resulting in manipulation of control.

As for the MGate 5105 Arbitrary R/W Vulnerability, it allows an attacker to issue an unauthorized command message to modify parameter³⁵ (the temperature threshold in the example) resulting in manipulation of view (modified temperature threshold).

ATT&CK for ICS Technique	ATT&CK for ICS Impact
Manipulate I/O image	Manipulation of control
Unauthorized command message	Manipulation of view
Modify parameter	

Impact of Device Vulnerabilities - Denial of Service (DoS)

Our tests found that all three of the real-time gateways (NIO50, Link150, Digi One) and the DA10D were susceptible to DoS attacks that prevented the transmission of messages (block command message,³⁶ block reporting message,³⁷ block serial COM³⁸), resulting in denial of control and denial of view.

ATT&CK for ICS Technique	ATT&CK for ICS Impact
Block command message	Denial of control
Block reporting message	Denial of view
Block Serial COM	
Denial of service	

Impact of Cloud Vulnerabilities - Nexcom NIO50 MQTT

Lack of Encryption

Due to the lack of encryption on the NIO50, an attacker could observe or manipulate the communications, compromising the integrity of the data. If the MQTT data is only for monitoring the communication with the PLC, the observable data could give an attacker information about the process. It could also allow an attacker to manipulate the data being sent to the MQTT server, resulting in manipulation of view. If MQTT is used to send commands to the PLC, the commands could be manipulated, preventing the correct action from being performed or performing an alternate action, resulting in manipulation of control.

ATT&CK for ICS Technique	ATT&CK for ICS Impact
Unauthorized command message	Manipulation of control
	Manipulation of view

Other Vulnerabilities and Security Issues

Throughout this research, we have discovered several other vulnerabilities and security issues (refer to the Device Vulnerabilities and Other Findings sections), that by themselves do not pose a high risk.

However, adversaries interested in sabotaging industrial facilities are mostly advanced attackers. They will likely take their time to learn about the network, devices, and processes, and formulate a highly specific attack based on the gathered information.

Therefore, minor vulnerabilities can become part of an attack chain designed by an advanced adversary to achieve their goal. This is called a complex process attack.

We presented an example of a complex process attack earlier in the paper (see Malicious Extraction of I/O Mapping Table), the minor security issues in the MGate 5105's user authentication and data transmission that can be leveraged by an attacker to read and manipulate the I/O mapping.

Discussion & Recommendations

Over the last decade, industrial networks have become more interconnected, thanks to several abstraction layers enabling control servers to communicate with PLC-controlled machines located on separate or remote facilities. Operational technology networks that were traditionally isolated, and designed and operated by field engineers, are now interacting with information technology ecosystems, making visibility and management easier. Unfortunately, this also exposes the ecosystem to more vulnerabilities and makes it more prone to errors.

While this evolution brings a lot of additions and benefits, like the possibility to control larger industrial processes through a centralized control center, it also adds complexity. One aspect of this complexity is dictated by heterogeneous networks that need to communicate using the same language. Within this new, interconnected ecosystem, protocol gateways play a crucial role. While these devices often go unnoticed because of their small size, or because it is not part of the process inventory (as opposed to a piece of machinery), these devices play an active part in the communication chain, such as in a control server delivering the order for a new piece. Requests and responses should be translated and delivered by protocol gateways properly.

We focused on protocol translation in this research paper, because we believe that this is a topic overlooked not only by the security community but also by those who are in the OT and engineering fields. We wanted to find out how a small error in the protocol translation's design could result in a critical issue for the whole production or process network. For example, we showed how a malicious actor could potentially set an out-of-bound value on a motor's engine, or a heating system, by requesting what appears to be an innocent read request. We believe that these subtle attacks could easily go unnoticed and therefore require more attention from the security and engineering staff. In a period in which advanced, highly motivated cybercrime is a given fact, we should assume that the entire production chain needs to be secured. The security coverage should also include these tiny, overlooked devices operating the connectivity between networks.

Based on what we have learned from our research, here are a number of useful suggestions and recommendations for vendors, installers, or end-users of industrial protocol gateways.

1. Even though different protocol gateways operate similarly, devices from different vendors handle invalid packets differently. Some of the devices considered in our research do not offer adequate packet filtering capabilities and, therefore, were more prone to translation errors or behaved unexpectedly after they received a malformed packet. This also includes resource exhaustion problems: while some devices operate correctly under normal circumstances, others are more prone to a DoS attack if they receive a purposely crafted malicious packet (for example, without proper process memory segmentation). As we already wrote in the paper, protocol gateways play a crucial role in the operation of industrial networks, and even a small fault could have a significant impact, like an interruption of the production. Given these considerations, one should appropriately and carefully consider these design aspects when evaluating products during the procurement process.
2. A protocol-aware ICS firewall is useful in two ways: It detects packets that do not comply with protocol standards and enforces administrative access to protocol gateways only from authorized endpoints. Having an ICS firewall helps ensure the integrity of the traffic while enforcing secure access from allowed devices. Organizations can also take advantage of cybersecurity solutions designed for ICS environments. Trend Micro's TXOne Networks offers both network- and endpoint-based products to help provide real-time, in-depth defense to OT networks and mission-critical devices. However, ICS firewalls only cover the Ethernet side of the network³⁹. As far as we know, there are no serial-based ICS firewalls. Depending on the risk assessment and security requirements, mission-critical facilities may opt to create an in-house monitor on the serial side to ensure the integrity of control, commands, and data.
3. Allocate appropriate time for configuring and protecting the gateway — this is even more true for data stations. Our tests showed how easy it is to misconfigure an I/O mapping table, which can provide a malicious actor a platform for conducting stealthy attacks. This careful approach to configuration includes well-known security best practices such as using strong credentials, disabling unnecessary services (one of the gateways we tested had anonymous FTP upload enabled by default), and enabling encryption where supported, e.g., in the cloud integration (MQTT).
4. Consider the protocol gateways as a critical OT asset, if this is not already the case. This implies the application of security management procedures, like regular security assessments for identifying potential vulnerabilities or misconfiguration, and application of security patches that complies with the organization's patching policy for OT systems. Even though applying security patches (or firmware updates in general) is sometimes a cumbersome activity for embedded devices, modern gateways are paired with management software that provides simplified mechanisms for firmware updates.

For convenience, we added more recommendations for auditors and consultants in the Appendix, Detailed Recommendations for Auditors, Consultants.

Related Work

In our research, we investigated the risks connected with the translation of industrial protocols, in particular, Modbus. While a certain extent of research has already been conducted in the domain of ICS and networks, no previous research on protocol translation is known to us.

In addition, while a significant amount of research used simulators to evaluate potential security risks, few have conducted empirical analyses with real devices and installations. Testbeds consisting of simulation models were used as a practical alternative to the latest hardware and the number of different technologies employed in the ICS world.

Holm et al. conducted research that aimed to improve the study of ICS environments by surveying several ICS testbeds.⁴⁰ Meanwhile, Amin et al. used such mathematical models to show the effects of DoS attacks against control systems,⁴¹ and Mo et al. analyzed the effects of false data injection on simulation.⁴² While this approach is interesting, it does not explore the risks connected with errors introduced in the implementation phase of a product.

More on the practical aspects of security, Niedermaier et al. studied the effect of DoS attacks on real-world PLCs.⁴³ By relying on an exhaustive evaluation made on 16 devices from six different vendors, the researchers confirmed that PLCs are susceptible to network flooding and, in worse scenarios, service disruption. Kalluri et al. arrived at a similar conclusion.⁴⁴ Closer to our work on protocol gateways, Thomas Roth showed how these embedded devices expose a number of vulnerabilities that malicious actors could potentially exploit.⁴⁵ While this work is certainly of interest, it does treat protocol gateways as standalone devices while investigating the risks related to translating, for example, commands issued from a control server located on one interface to a PLC on a second interface of the gateway.

Appendix

List of Vulnerabilities Discovered in this Research

#	Gateway	Name	ID	Reporting date
1	NIO50	Protocol translation bypass	ZDI-CAN-10485	Feb 10, 2020
2		Unencrypted MQTT	ZDI-CAN-10486	Feb 10, 2020
3		Authentication bypass	ZDI-CAN-10487	Feb 10, 2020
4		Unsanitized MQTT upstream	ZDI-CAN-10488	Feb 10, 2020
5	MGate 5105	Information disclosure through proprietary commands	CVE-2020-15494	Mar 18, 2020
6		Credential reuse through proprietary commands	CVE-2020-15493	Mar 18, 2020
7		Post-auth root shell and persistence	CVE-2020-8858	Oct 14, 2019
8	DA10D	Modbus read denial of service	ZDI-CAN-10804	Mar 23, 2020
9		Arbitrary memory leakage	ZDI-CAN-10897	Apr 5, 2020

Summary of Vulnerabilities Discovered and Reported

Detailed Recommendations for Auditors, Consultants and Field Engineers

Recommendations for protocol translation vulnerabilities

I/O Mapping Vulnerabilities – MGate 5105 and DA10D

The difficulty with the I/O mapping vulnerabilities is that the I/O mapping table is the core of how the data station devices function. A valid user could accidentally trigger one of these vulnerabilities and affect the process.

- Monitor the device for modifications of the configuration
- Ensure change management process is used
- Review the configuration periodically
- Password-protect configurations (if applicable)
- Restrict access to the programming port or service to specific systems (e.g., engineering workstations).

Moxa has also reached out to us and recommended the following:

1. Disable Moxa Command after the first installation of MGate 5105
2. If the user still needs to use Moxa Command after the first installation, it is recommended to restrict access to the MGate 5105 through “Accessible IP List setting” to prevent unauthorized users or hosts from getting the device information.

- Monitor traffic to the device, especially programming traffic
- Nexcom packet forward vulnerabilities
- Monitor traffic sent to the protocol gateway

- Use a firewall that validates the data
- Patch when available
 - Work with the vendor
 - Test the patch in a lab
 - Move to production as soon as possible

Recommendations for device vulnerabilities

Denial of service

- Monitor traffic sent to the protocol gateway for packets that may trigger a DoS
- Monitor for increased bandwidth to the protocol gateway to prevent resource exhaustion
- Patch when available
 - Work with the vendor
 - Test the patch in a lab
 - Move to production as soon as possible

Privilege escalation

- Implement a strong password policy
- Patch when available
 - Work with the vendor
 - Test the patch in a lab
 - Move to production during the next scheduled downtime

Credential reuse

- Restrict access to the programming port or service to specific systems (such as engineering workstations)
- Patch routers and switching hubs, and avoid using unsecure or outdated routers.
- Patch when available
 - Work with the vendor
 - Test the patch in a lab
 - Move to production during the next scheduled downtime

Memory leak

- Monitor traffic for packets that trigger the memory leak
- Patch when available
 - Work with the vendor
 - Test the patch in a lab
 - Move to production during the next scheduled downtime

Recommendations for cloud vulnerabilities

Lack of encryption

- Patch when available
 - Work with the vendor
 - Test the patch in a lab
 - Move to production during the next scheduled downtime
- Create a VPN tunnel to cloud service

Null username

- Restrict access to MQTT server
- Create VPN tunnel to cloud services
- Patch when available
 - Work with the vendor
 - Test the patch in a lab
 - Move to production during the next scheduled downtime

Blind forward

- Validate all data coming from the Nexcom MQTT client before use in another application or database
- Patch when available
 - Work with the vendor
 - Test the patch in a lab
 - Move to production during the next scheduled downtime

Recommendations for other issues

MGate 5105 change IP with a magic packet

- Do not use the default IP address
- Change the IP address before deploying to the network

Modbus unit ID roll-over

- Monitor traffic sent for Modbus IDs over 127
- Use a firewall that validates packets

MGate 5105 firmware hardcoded password / password reuse

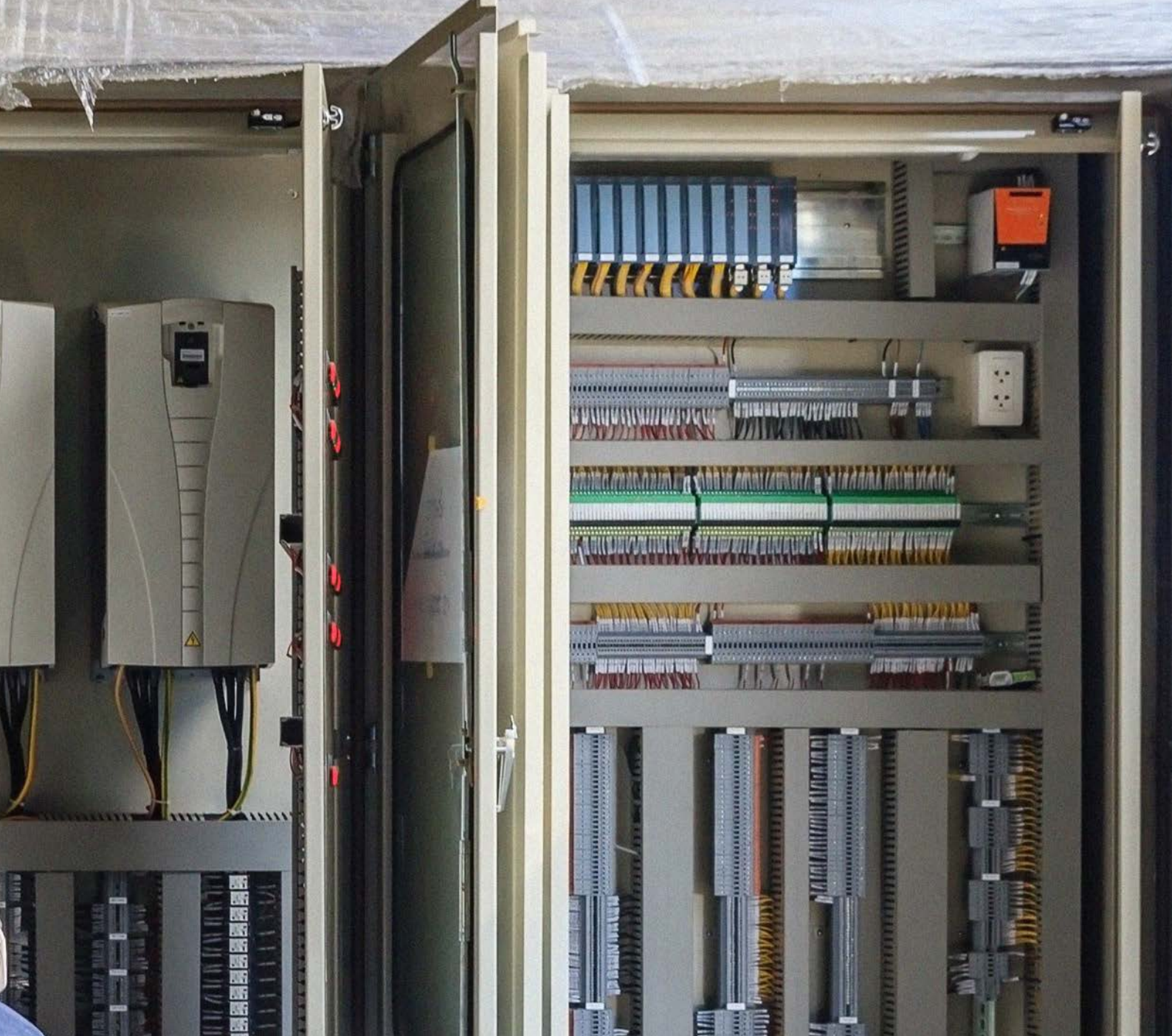
This is an issue that only the vendor can address. Signing firmware updates should have a higher priority than encryption of firmware updates though they can typically be implemented together.

- Properly sign firmware updates
- Properly encrypt firmware updates if encryption is necessary
- Use industrial standard asymmetrical encryption and protect the private key in a secure element.

References

- 1 MITRE. (n.d.). *MITRE*. “Denial of View.” Accessed on July 15, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T815>.
- 2 MITRE. (n.d.). *MITRE*. “Denial of Control.” Accessed on July 15, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T813>.
- 3 MITRE. (n.d.). *MITRE*. “Manipulation of View.” Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T832>.
- 4 MITRE. (n.d.). *MITRE*. “Manipulation of Control.” Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T831>.
- 5 CISA. (Feb. 25, 2016). *CISA*. “ICS Alert (IR-ALERT-H-16-056-01): Cyber-Attack Against Ukrainian Critical Infrastructure.” Accessed on July 15, 2020, at <https://us-cert.cisa.gov/ics/alerts/IR-ALERT-H-16-056-01>.
- 6 SANS and E-ISAC. (March 18, 2016). *SANS*. “Analysis of the Cyber Attack on the Ukrainian Power Grid.” Accessed on July 15, 2020, at https://www.nerc.com/pa/CI/ESISAC/Documents/E-ISAC_SANS_Ukraine_DUC_18Mar2016.pdf.
- 7 Modbus. (April 26, 2012). *Modbus*. “Modbus Application Protocol Specification v1.1b3.” Accessed on July 16, 2020, at www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.
- 8 Modbus. (April 26, 2012). *Modbus*. “Modbus Application Protocol Specification v1.1b3.” Accessed on July 16, 2020, at www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.
- 9 Digital Loggers. (n.d.). *Digital Loggers*. “Pro Switch.” Accessed on July 17, 2020, at <https://dlidirect.com/products/new-pro-switch>.
- 10 IO Ninja. (n.d.). *IO Ninja*. “Introducing IO Ninja.” Accessed on July 16, 2020, at <https://ioninja.com/>.
- 11 Stratus Engineers. (n.d.). *Stratus Engineers*. “EZ-Tap™ Pro RS-232 Passive Tap Module – Bus Analyzer / Protocol Analyzer.” Accessed on July 17, 2020, at <https://www.stratusengineering.com/product/ez-tap-pro/>.
- 12 Elbar. (May 26, 2020). *SourceForge*. “QModMaster.” Accessed on July 16, 2020, at <https://sourceforge.net/projects/qmodmaster/>.
- 13 Elbar. (May 23, 2020). *SourceForge*. “PyModSlave.” Accessed on July 16, 2020, at <https://sourceforge.net/projects/pymodslave/>.
- 14 Wireshark. (n.d.). *Wireshark*. “Wireshark.” Accessed on July 16, 2020, at <https://www.wireshark.org/>.
- 15 IO Ninja. (n.d.). *IO Ninja*. “Introducing IO Ninja.” Accessed on July 16, 2020, at <https://ioninja.com/>.
- 16 Joshua Pereyda. (n.d.). *GitHub*. “BooFuzz fuzzing framework.” Accessed on July 16, 2020, at <https://github.com/jtpereyda/boofuzz>.
- 17 Pedram Amini, Aaron Portnoy, and Ryan Sears. (n.d.). *GitHub*. “Sulley Fuzzing Framework.” Accessed on July 16, 2020, at <https://github.com/OpenRCE/sulley>.
- 18 Aki Helin. (n.d.). *GitLab*. “Radamsa fuzzing test case generator.” Accessed on July 16, 2020, at <https://gitlab.com/akihe/radamsa>.
- 19 Matthias Niedermaier, Florian Fischer, and Alexander von Bodisco. (Sept. 2017). *IEEE Xplore*. “PropFuzz — An IT-security fuzzing framework for proprietary ICS protocols.” Accessed on July 16, 2020, at <https://ieeexplore.ieee.org/document/8053600/>.
- 20 Boofuzz. (n.d.). *GitHub*. “boofuzz-modbus.” Accessed on July 16, 2020, at <https://github.com/youngcraft/boofuzz-modbus>.
- 21 Red Lion. (n.d.). *Red Lion*. “Crimson 3.5.” Accessed on July 16, 2020, at <https://www.redlion.net/red-lion-software/crimson/crimson-31>.
- 22 Common Vulnerabilities and Exposures. (July 1, 2020). *Common Vulnerabilities and Exposures*. “CVE-2020-15493.” Accessed on July 17, 2020, at <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15493>.
- 23 Moxa. (July 10, 2020). *Moxa*. “MGate 5105-MB-EIP Series Protocol Gateways Vulnerabilities.” Accessed on July 30, 2020, at <https://www.moxa.com/en/support/support/security-advisory/mgate-5105-mb-eip-series-protocol-gateways-vulnerabilities>.
- 24 stacksmashing / Ghidra. (Mar. 14, 2019). *YouTube*. “Reverse engineering with #Ghidra: Breaking an embedded firmware encryption scheme.” Accessed on July 17, 2020, at <https://www.youtube.com/watch?v=4urMITJKQQs>.

- 25 Common Vulnerabilities and Exposures. (July 1, 2020). *Common Vulnerabilities and Exposures*. "CVE-2020-15493." Accessed on July 17, 2020, at <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15494>.
- 26 Moxa. (July 10, 2020). Moxa. "MGate 5105-MB-EIP Series Protocol Gateways Vulnerabilities." Accessed on July 30, 2020, at <https://www.moxa.com/en/support/support/security-advisory/mgate-5105-mb-eip-series-protocol-gateways-vulnerabilities>.
- 27 National Vulnerability Database. (Feb 14, 2020). *National Vulnerability Database*. "CVE-2020-8858." Accessed on July 17, 2020, at <https://nvd.nist.gov/vuln/detail/CVE-2020-8858>.
- 28 MITRE. (n.d.). MITRE. "ATT&CK for Industrial Control Systems." Accessed on July 17, 2020, at https://collaborate.mitre.org/attackics/index.php/Main_Page.
- 29 MITRE. (n.d.). MITRE. "Denial of View." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T815>.
- 30 MITRE. (n.d.). MITRE. "Denial of Control." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T813>.
- 31 MITRE. (n.d.). MITRE. "Manipulation of View." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T832>.
- 32 MITRE. (n.d.). MITRE. "Manipulation of Control." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T831>.
- 33 MITRE. (n.d.). MITRE. "Unauthorized Command Message." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T855>.
- 34 MITRE. (n.d.). MITRE. "Manipulate I/O Image." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T835>.
- 35 MITRE. (n.d.). MITRE. "Modify Parameter." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T836>.
- 36 MITRE. (n.d.). MITRE. "Block Command Message." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T803>.
- 37 MITRE. (n.d.). MITRE. "Block Reporting Message." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T804>.
- 38 MITRE. (n.d.). MITRE. "Block Serial COM." Accessed on July 17, 2020, at <https://collaborate.mitre.org/attackics/index.php/Technique/T805>.
- 39 Trend Micro. (n.d.) *TXOne Networks*. "HYPERLINK "https://www.txone-networks.com/en-global" TXOne Networks." Accessed on July 23, 2020 at <https://www.txone-networks.com/en-global>.
- 40 Hannes Holm et al. (2015). *Springer*. "A Survey of Industrial Control System Testbeds." Accessed on July 17, 2020, at https://doi.org/10.1007/978-3-319-26502-5_2.
- 41 Saurabh Amin, Alvaro Cárdenas, and Shankar Sastry. (2010). *Springer*. "Safe and Secure Networked Control Systems under Denial-of-Service Attacks." Accessed on July 17, 2020, at https://doi.org/10.1007/978-3-642-00602-9_3.
- 42 Yilin Mo and Bruno Sinopoli. (2010). *Springer*. "False data injection attacks in control systems." Accessed on July 17, 2020, at https://www.researchgate.net/publication/228859026_False_data_injection_attacks_in_control_systems.
- 43 Matthias Niedermaier et al. (2018). *Usenix*. "You Snooze, You Lose: Measuring PLC Cycle Times under Attacks." Accessed on July 17, 2020, at <https://www.usenix.org/system/files/conference/woot18/woot18-paper-niedermaier.pdf>.
- 44 Rajesh Kalluri et al. (2016). *IEEE Xplore*. "Simulation and impact analysis of denial-of-service attacks on power SCADA." Accessed on July 17, 2020, at <https://ieeexplore.ieee.org/document/7858908>.
- 45 Thomas Roth. (Aug. 7, 2018). *Conference Cast*. "Breaking the IIoT: Hacking industrial Control Gateways." Accessed on July 17, 2020, at <https://www.conferencecast.tv/talk-17877-breaking-the-iiot-hacking-industrial-control-gateways>.



TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threat techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com



| research 