Global Information Assurance Certification Paper

Vulnerabilities on the Wire: Mitigations for Insecure ICS Device Communication

GIAC (GRID) Gold Certification

Author: Michael Hoffman, mjhoffman80@gmail.com
Advisor: David Fletcher

Accepted: 2/6/2020

Abstract

Modbus TCP and other legacy ICS protocols ported over from serial communications are still widely used in many ICS verticals. Due to extended operational ICS component life, these protocols will be used for many years to come. Insecure ICS protocols allow attackers to potentially manipulate PLC code and logic values that could lead to disrupted critical system operations. These protocols are susceptible to replay attacks and unauthenticated command execution (Bodungen, Singer, Shbeeb, Hilt, & Wilhoit, 2017). This paper examines the viability of deploying PLC configuration modifications, programming best practices, and network security controls to demonstrate that it is possible to increase the difficulty for attackers to maliciously abuse ICS devices and mitigate the effects of attacks based on insecure ICS protocols. Student kits provided in SANS ICS515 and ICS612 courses form the backdrop for testing and evaluation of ICS protocols and device configurations.

1. Introduction

Modbus, an industrial protocol used for server to client communication, has been used for over 40 years and is still widely deployed in new ICS installations (Mostia, 2019). Modbus can be transported over serial mediums of RS232, RS485, or it can be wrapped in an IEEE 802.3 TCP segment. Within TCP, the typical implementation is Modbus Remote Terminal Unit (RTU) contained in the TCP/IP stack Application layer, which can be easily viewed in Wireshark (Sanchez, 2017). Modbus uses simple function calls combined with data range requests to read and write bits, called coils. Additionally, it can also read and write integers or floats, called registers. When engineers were encapsulating Modbus within TCP, cybersecurity concerns were nonexistent and, therefore, Modbus RTU does not have any built-in security mechanisms (Rinaldi, n.d.). From an ICS security perspective, Modbus is rife with many vulnerabilities and is subject to Probe, Scan, Flood, Authentication Bypass, Spoof, Eavesdrop, Misdirect, Read/Copy, Terminate, Execute, Modify, and Delete attacks (Draias, Serhrouchni, & Vogel, 2015)

1.1. Where Insecure ICS Protocols are Commonly Found

In an ICS environment, the Modbus TCP protocol is often found near the physical processes. As shown in Figure 1, advanced field devices at level 0, such as mass flow meters and analytical instrumentation, may have Modbus TCP capability.

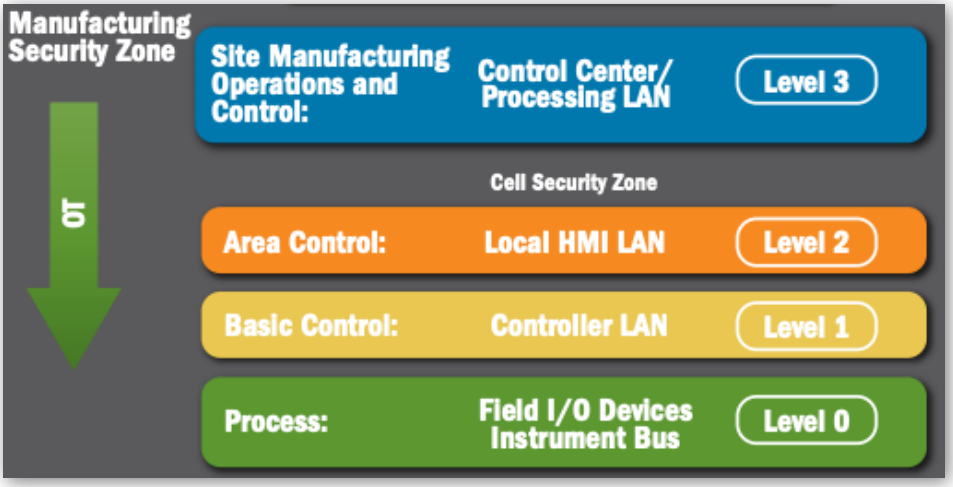


Figure 1. ICS zone segmentation (CSIA, 2016)

Michael Hoffman, mjhoffman80@gmail.com

Programmable Logic Controllers (PLCs) or RTUs comprise functional level 1, where they are used as masters or slaves based on the Modbus communication hierarchy. These devices normally house the logic that controls the physical process. Supervisory Control and Data Acquisition (SCADA), master stations, or communication gateways, such as Modbus to Open Platform Communication (OPC) servers, comprise functional level 2 and provide oversight and input to lower-level devices. Therefore, Modbus covers IEC 62443 functional layers 0 to 2 (CSIA, 2016).

1.2. ICS Attacks Leveraging Insecure Protocols

Although the Modbus protocol is plagued with many vulnerabilities and is easy to exploit on its own, the difficulty in carrying out attacks on ICS systems lies within the loose relationship between Modbus data and the ICS device logic and programming. Modbus does not associate data with an information model as other protocols or frameworks do (OPC Foundation, 2020). For instance, the values at coil 50 or register 120 could be implemented entirely differently between two PLCs mounted side-by-side in a control panel. It depends on the developed logic and association between Modbus mapping tables. Therefore, without viewing a project file, PLC program, or HMI screen, limited contextual information is available to an attacker without significant dwell time in the ICS environment to watch and learn. Modbus registers can represent a range of process or control variables. These may include tank level, flow rate, temperature, pressure, voltage, setpoint, or control output. Modbus coils, on the other hand, offer less context and are more difficult to decipher: a value of 0 or 1, for example, could indicate pump status, breaker position, valve position, or can be used for control of those items.

Despite the difficulty of gaining process and control system understanding, the 2014 German Steel Mill Cyber Attack illustrated that adversaries learned sufficient details about the environment, leveraged specialized ICS knowledge, and caused multiple control system failures that ultimately led to a plant outage and consequential physical equipment damage (Lee, Assante, & Conway, 2014). The 2015 Ukraine power grid attack is a further example of adversaries learning the environment and leveraging trusted communications to cause an electrical grid outage. The second attack on the Ukraine power grid, which happened in 2016, utilized extensible ICS specific malware that is known as CRASHOVERRIDE. The malware contained an IEC104 protocol module,

among others, that leveraged capabilities to pull RTU configurations in the ICS environment. The IEC104 module then used this implementation-specific knowledge to manipulate breaker positions, leading to a power outage (Lee R. M., 2017). As Slowik suggests, ICS attacks appear to be increasing both in relative frequency and severity (2019). Therefore, when considering the increasing threat to the ICS environment, it becomes imperative for asset owners and operators to leverage proper device configuration, programming methodologies, and network security barriers to increase the security posture of ICS devices that use insecure protocol communication.

1.3. ICS Security Challenges and Potential Solutions

The Stuxnet malware that infected Siemens PLCs to perform centrifuge cascade overpressure and centrifuge rotor speed operational manipulations at the Natanz Fuel Enrichment Plants marked a decisive change in the history of the ICS community (Langer, 2013). Since Stuxnet, ICS owners, operators, and vendors have begun to take notice of their vulnerable systems and incorporate increased security controls. As an example, Rockwell Automation ControlLogix PLCs now can encrypt their programs and lock access to routines (Allen-Bradley, 2018). Additionally, they have included new CIP security enhancements in their communication modules to encrypt EtherNet/IP protocol traffic between PLCs and drives (Rockwell Automation, 2019). For their S7 PLC product line, Siemens has implemented password block protection and three staggered CPU protection levels (Siemens, 2016).

Despite these improvements, researchers are still at work uncovering many vulnerabilities in ICS devices. For the year of 2018, Dragos indicated that 17 ICS vulnerability advisories are reported monthly, on average, and of those advisories, 28 percent are related to PLCs and industrial equipment (Dragos, 2018). For the asset owners and operators, however, PLCs and industrial equipment are the same devices used in control systems of physical processes and cannot be easily patched without causing a process disruption. For specific industries and processes, an average span of five to six years is prevalent between maintenance windows where firmware can be upgraded. These devices also are at a functional location in the Purdue Model where the responsibility of work blurs between various staff, including instrumentation technicians,

electricians, automation engineers, ICS security engineers, and vendors. Therefore, coordination with operations personnel and working closely with technical staff is necessary to perform device-level firmware upgrades. Nevertheless, apart from keeping devices patched, many of these devices are still insecure by design and can be manipulated by an attacker without leveraging any unpatched vulnerability (Langner, 2019).

Still, a variety of potential controls and protective measures do exist for these "insecure-by-design" systems. Many vendors offer embedded PLC and RTU security configuration settings. Other potential defenses include properly checking range values in the PLC code itself, which is not unlike conventional standard programming best practices (McConnell, 2004). Additional essential controls include establishing startup or "default" values in the PLC for proper recovery (Shearer, Dely, Conway, & Robinson, 2019). Finally, installing an Application layer ICS firewall to permit only those PLC functions necessary for the intended program, is yet another mitigating control (Bodungen, Singer, Shbeeb, Hilt, & Wilhoit, 2017). Therefore, a holistic effort is needed for ICS device-level security, as shown in Figure 3, starting with applying the latest tested firmware, disabling services unneeded, and enabling security controls already supplied by the vendor.

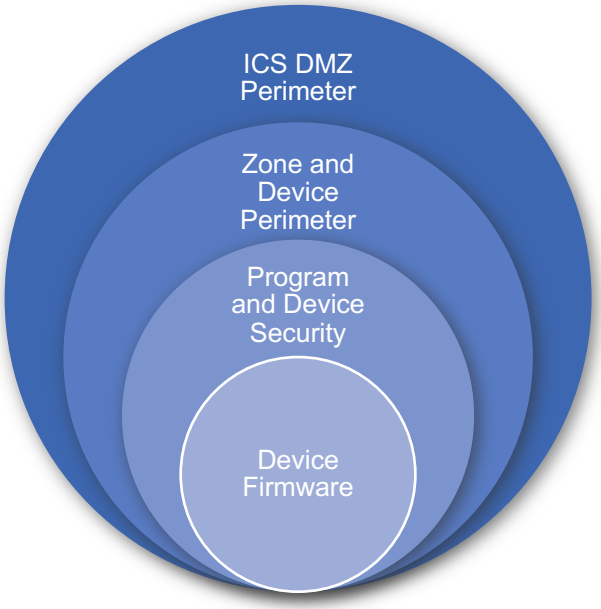


Figure 3. Layers ICS device protection

The next layer of security is achieved by implementing secure programming practices, such as treating any external data as suspect and ensuring that if devices do fail or are compromised, they can be restarted with necessary default values for safe operation. The final layers of Device, Zone, and Parameter protection provide facilities to tightly restrict communication at ICS protocol level and protect the lower functional networks from those levels above them.

This research will demonstrate that it is possible to increase the level of effort required for an attacker to maliciously abuse ICS devices that use insecure protocols through the implementation of configuration, programming, and network security controls. With these potential mitigation solutions reviewed, the remaining sections cover assessments leveraging both the SANS ICS515 and ICS612 course student kits.

2. Research Method

The criteria for researching the proposed mitigations was to leverage ICS devices provided to SANS ICS course students, ensure assessments could be repeated by other researchers, and provide techniques that could be tailored and extended to various ICS systems. As shown in figure 2, the components in the ICS VLAN include student PLC kits from SANS ICS515 and ICS612 courses and a MOXA EDR-G903 Modbus Application layer firewall. For the supervisory VLAN, a workstation VM was used for both PLC and CybatiWorks development. Additionally, the ctmodbus tool was utilized from the ControlThingsIO Platform VM to generate Modbus packets in an attempt to overwrite coils and registers in the ICS VLAN (ControlThings I/O, 2020). A separate VM was deployed to run Wireshark to monitor traffic from either the ICS VLAN or between the CLICK PLC and Modbus firewall. A Cisco IE-3010 switch was used to route cross-VLAN communication and was configured to mirror all ICS VLAN traffic coming through the switch out to the Wireshark VM.

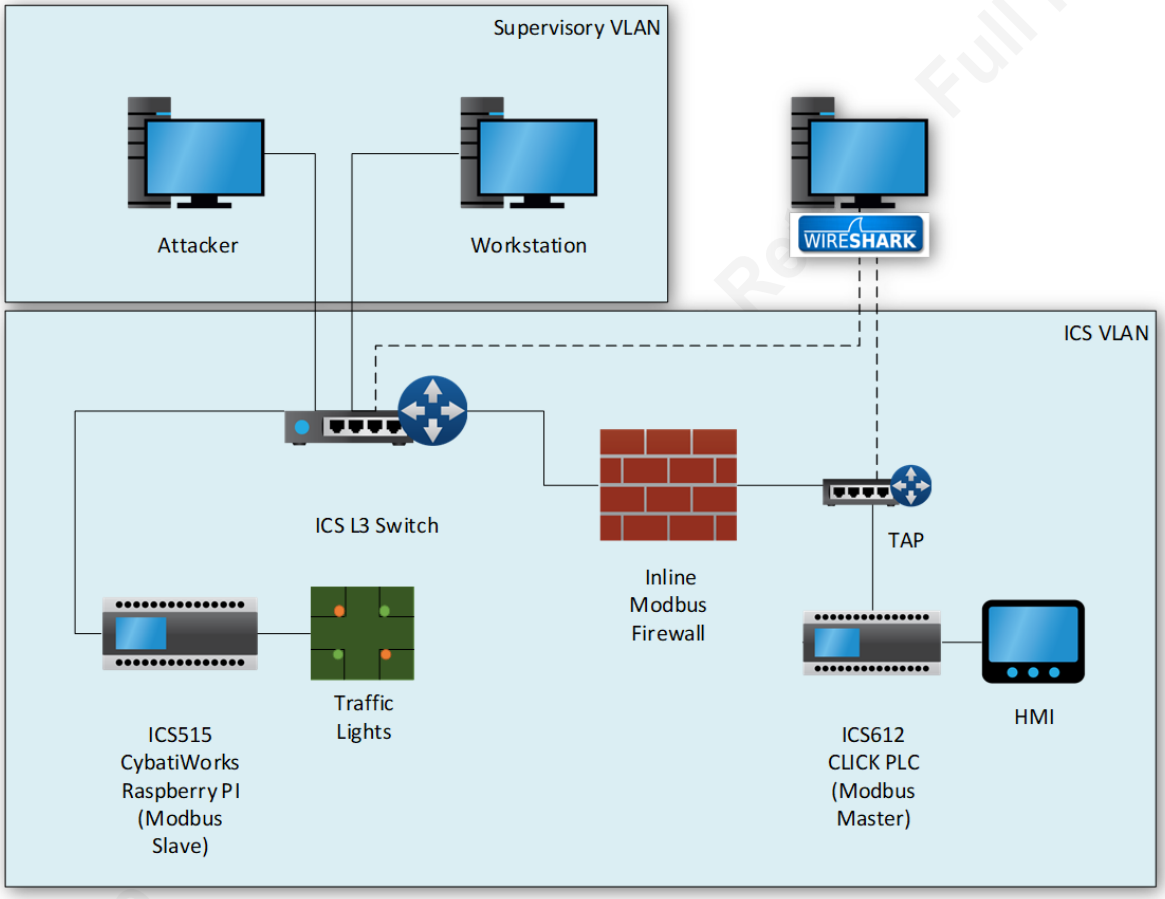


Figure 2. ICS Lab Setup

The ICS515 package incorporated a modified version of a traffic light program, used in the course, to simulate a four-way stoplight. The ICS612 CLICK PLC programming was modified to read and write Modbus registers from the ICS515 CybatiWorks Raspberry PI. The updated CLICK PLC program displayed the traffic lights status, the Raspberry system time on the local HMI, and had the capability to control the traffic lights and update the Raspberry system time from the HMI. This configuration is like many PLC-to-PLC communications used in various ICS environments. In this instance, the CybaitWorks Raspberry PI is functioning as a Modbus Slave, and the CLICK PLC is functioning as a Modbus Master with full read/write capability.

The first assessment consisted of configuring internal security controls in the CLICK PLC processor for communication session handling to determine if it would be effective against the remote attackers Modbus tool. Although this does test the effectiveness of the

CLICK PLC security controls, the researcher acknowledges that it likely has limited portability to other PLCs manufactures, such as Rockwell and Siemens, which include different security measures in their products. The second assessment included adding ladder logic code to separate running logic from communication logic. This step encompasses adding input filtering, which is essentially adopting and porting standard computer programing best practices to PLC ladder logic programming (OWASP Top Ten, 2020). Part of this process/step included adding default startup values to the code to bring the PLC back to a safe operational state in the event of a successful attack. The final test added a MOXA secure router in front of the CLICK PLC with a custom configuration to allow only the required Modbus register types and ranges between the two devices.

3. Findings and Discussion

Throughout the testing, it was evident that the devices under consideration are designed to operate with known "good" parameters, settings, and communications. During the firewall assessment, the CLICK PLC Modbus master and slave ladder logic blocks froze due to dropped Modbus connectivity and had to be restarted. Likewise, during the same assessment, the REXYGEN Modbus server component in the Raspberry PI froze from multiple dropped sockets and required a service restart to get communications working. Additionally, the CybatiWorks unit experienced an SD card corruption early in the testing. In order to get the Raspberry PI operational, the system was rebuilt using a new Raspbian Linux image and the latest version of the REXYGEN Core package. These issues highlight the importance of assessing ICS systems in a safe lab environment where device failures have a limited impact on the physical environment.

The following assessments carried out on the CLICK PLC, programs, CybatiWorks Raspberry PI, and Moxa firewall, take the approach of applying security controls from the inside out. This method consists of starting with device security, then focusing on programming controls, and finally, providing a secure network perimeter. The

assessments began with updating device firmware to the latest version to ensure the most recent vendor security settings and controls were available for testing.

3.1. PLC Security Controls Assessment

The CLICK programming software and PLC firmware version, which are often aligned, was at version 2.40. At the time of assessment, version 2.51 was available and subsequently installed. An upgrade to the PLC firmware followed with a project file upgrade. Upon initial review, the default configuration of the PLC was unprotected and allowed the PLC to be entirely manipulated by anyone with local or remote network access. Program uploads and downloads, operational changes, and data could be all manipulated.

After reviewing the security settings, password protection features were enabled for the project file and system configuration. Enabling these settings protects both the PLC program (ladder logic) and CPU (IP address, client/server settings, etc.) from being read or manipulated without a password. As shown in Figure 4, password protection settings for data reads and writes were also enabled to lock down remote Modbus interactions to the PLC acting as a Modbus slave. EtherNet/IP was out of scope for the assessment and disabled in a configuration section, not shown.

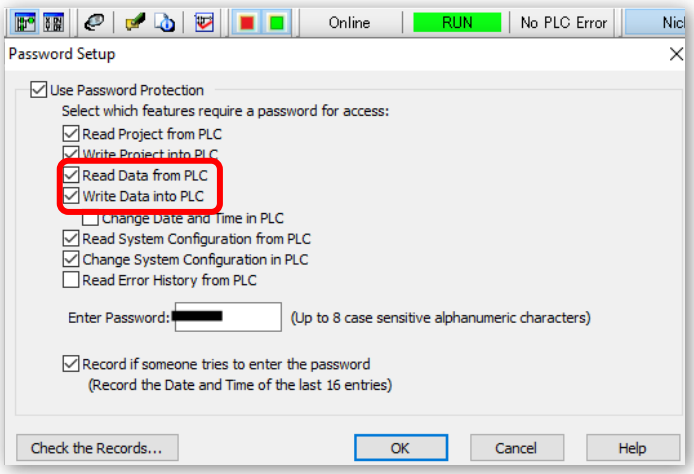


Figure 4. PLC password security settings

With password security enabled, the program was downloaded to the PLC. Initial testing revealed that Modbus read and write functionality could still be performed using the ctmodbus tool. Troubleshooting revealed that for the settings to take effect, the PLC

needed to be rebooted. After rebooting the PLC, any remote Modbus TCP read requests failed, with the slave device failure error, as shown in Figure 5. It is important to note, however, that if an attacker had access to the same VLAN where ICS is connected, they could perform a layer two attack and capture the password during the initial authentication challenge.

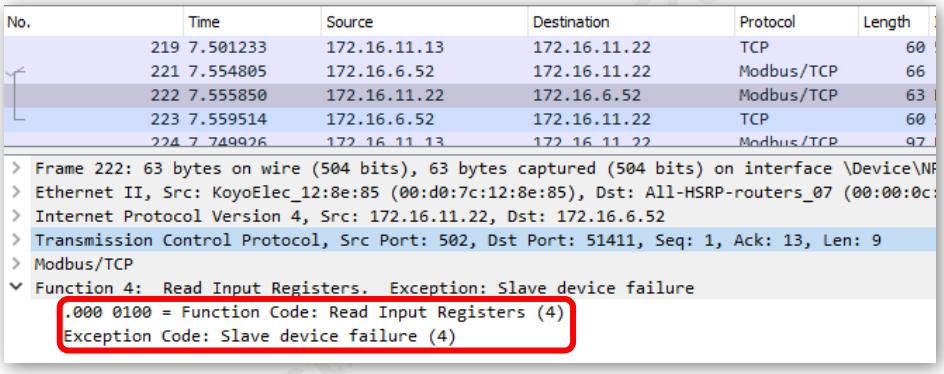


Figure 5. Wireshark capture of a Modbus read failure

The password security control for data is a global setting --affecting not only Ethernet communication but also serial communication, which is an essential consideration for enabling such a security control. The local HMI, which is part of the ICS612 student kit, is connected to the second port on the CLICK PLC, which is an RS-232C serial port and uses Modbus to read and write values. With the data security setting enabled, the HMI displayed a data access error as it could not read PLC values. Testing further commenced for Modbus writes, and those also failed, as shown in Figure 6.

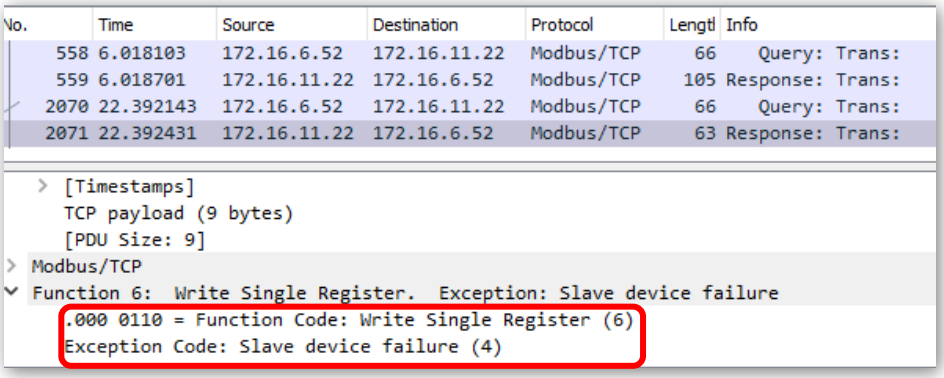


Figure 6. Wireshark capture of a Modbus write failure

Michael Hoffman, mjhoffman80@gmail.com

Based on the results above, the device controls effectively worked by limiting Modbus connectivity to an external Modbus master device irrespective of the transport protocol. However, despite Modbus slave isolation, the PLC continued to work correctly as a Modbus master by reading values from the ICS515 CybatiWorks modified traffic light program.

After verifying modus slave data password enforcement for reading and writing, the security settings were reverted. As shown in Figure 7, by enabling Modbus reads in the PLC security settings, the ctmodbus tool was able to read the PLC values once again.

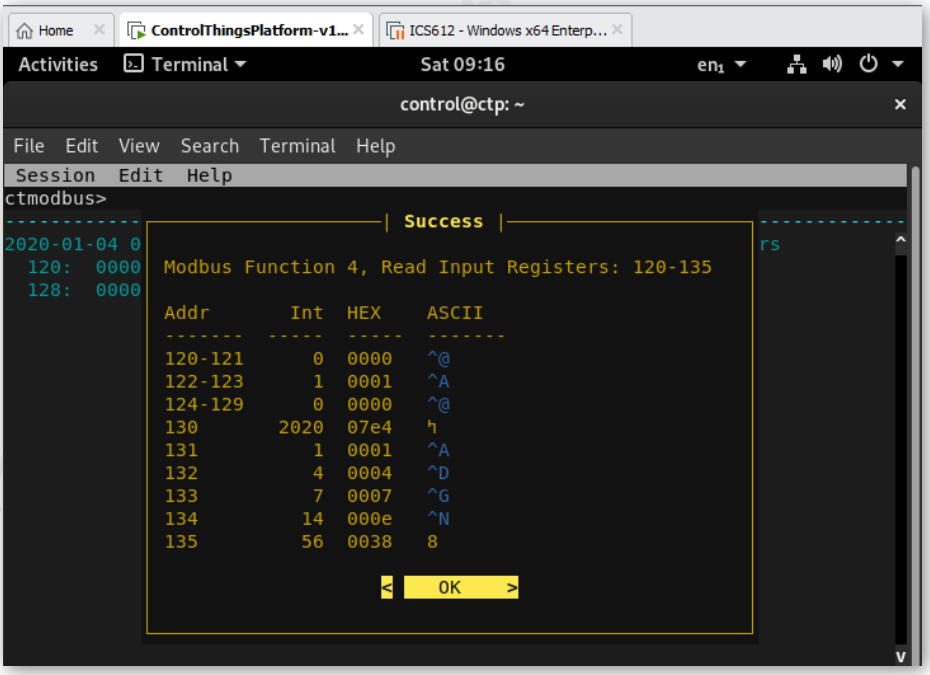


Figure 7. ctmodbus tool illustrating Modbus reads

Figure 8 shows the traffic generated by the ctmodbus tool, which is precisely the same, and further reinforces that Modbus values are transmitted in the clear and can easily be deciphered.

Michael Hoffman, mjhoffman80@gmail.com

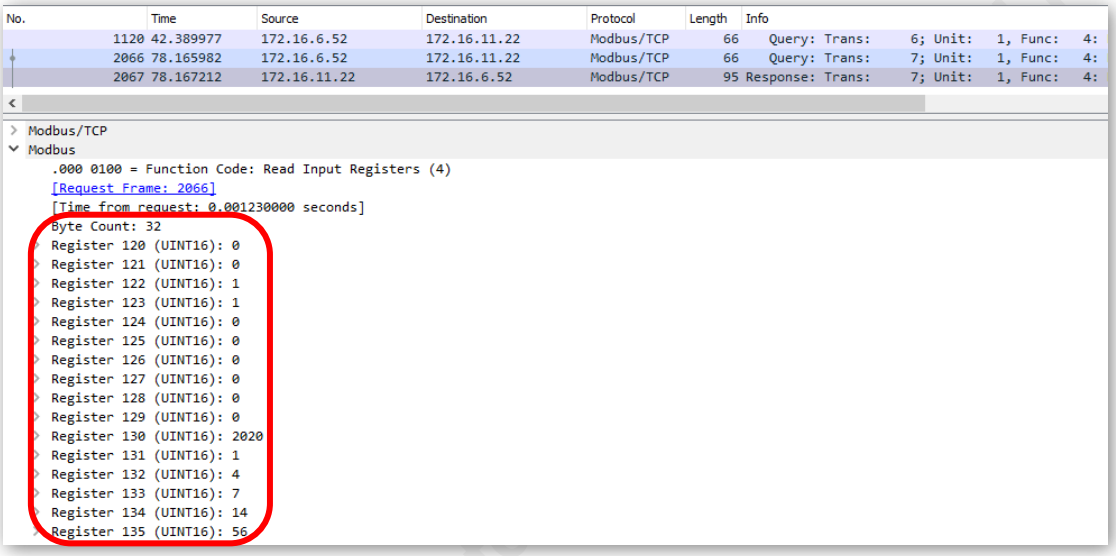


Figure 8. Modus reads captured by Wireshark

Beyond password security controls tested, the CLICK PLC did not offer restricted access on PLC tags themselves, although it did indicate tag access, as shown in Figure 9.

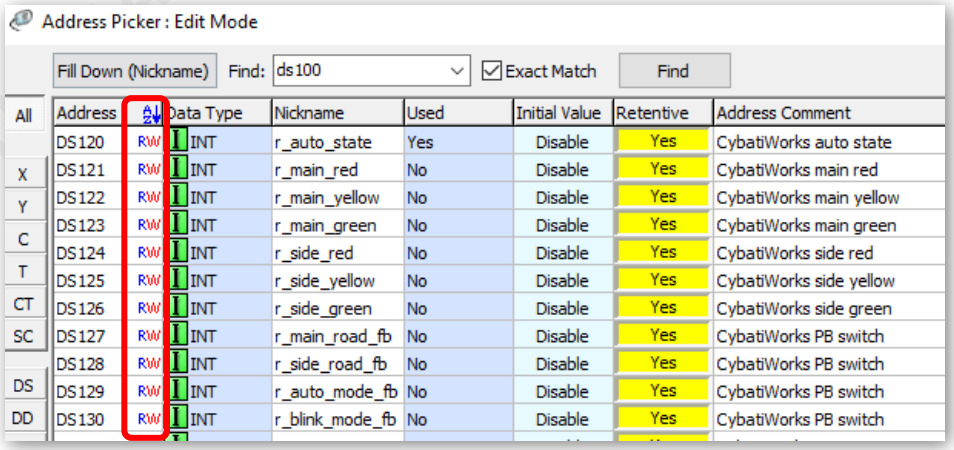


Figure 9. CLICK PLC tags showing Read/Write (RW) attributes

In contrast, Rockwell PLCs do offer external access attributes to control how external applications interact with the tag, which provides yet another means to restrict external communication at the tag value level (Allen-Bradley, 2018, p. 64). External access settings can be Read/Write, Read-Only, and None, which means that external applications, such as HMI, PLCs, or Historians, would not be able to interact with the tag.

Michael Hoffman, mjhoffman80@gmail.com

Similarly, the CybatiWorks REXYGEN application running on the Raspberry PI did have the ability to restrict Modbus access at the tag level, as shown in Figure 10.

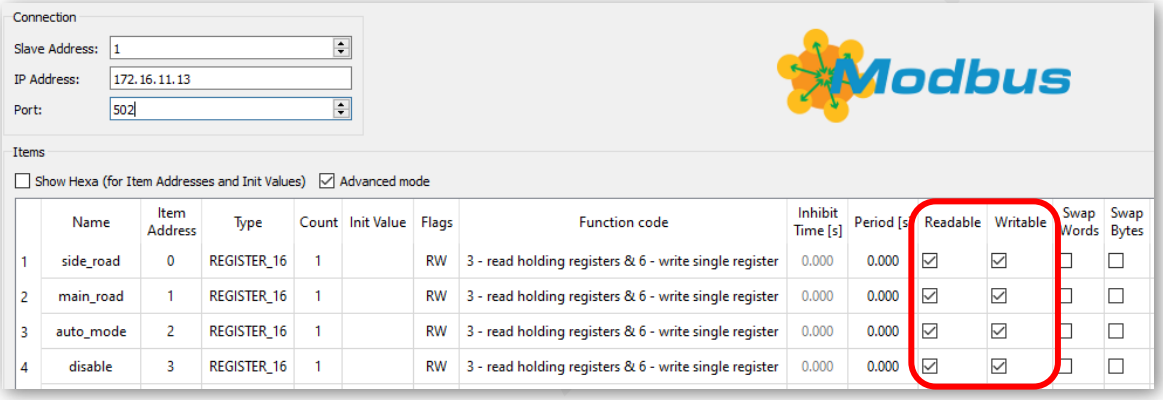


Figure 10. REXYGEN Modbus Slave configuration with read/write restriction

The auto_mode tag controls the traffic light program; toggling the bit will cause the traffic lights to stop or start the cyclic program operation. With the value set to both readable and writable, the ctmodbus tool was able to read the register at the time 16:51:23 with a value of one and then wrote over the register with a zero and subsequently re-read the register with a value of zero at 16:51:49, as shown below in Figure11.

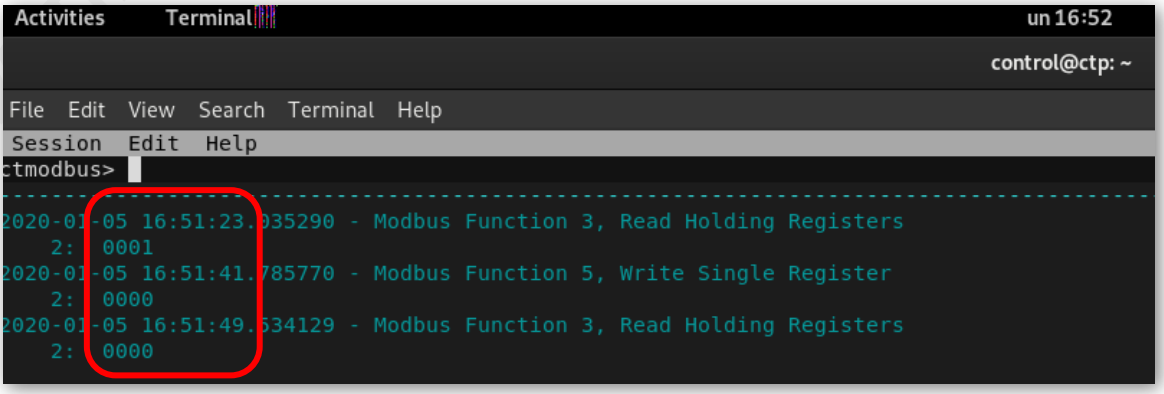


Figure 11. ctmodbus tool modifying auto_mode tag on the Raspberry PI

By making a configuration change to the auto_mode tag to select only writeable (this is per documentation for a REXYGEN Modbus slave settings and not a typo), the auto_mode value was able to be read but not written. As shown in Figure 12, the Write

Single Register 2 with a value of 1 did not affect the register value, and the subsequent Read Holding register still showed a value of zero.

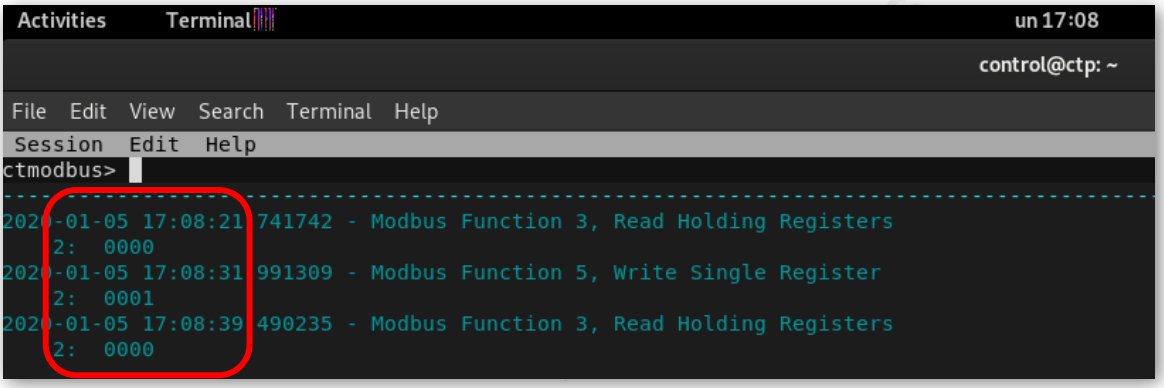


Figure 12. ctmodbus unable to modify auto_mode tag on the Raspberry PI

Therefore, this assessment indicates that the REXYGEN Modbus server can restrict access at the granular tag level. Restricting read and write access to Modbus registers and coils not required for the control implementation reduces the attack surface to the device. Furthermore, it is crucial to restrict access to functionality not required and apply standard security practices such as those given in CIS Control 5 (Center for Internet Security, 2020). If a PLC offers services such as a web portal, FTP host, or SSH host, these services and protocols should be thoroughly scrutinized and disabled if possible, to decrease the attack surface for the respective PLC controller and given control scenario. Once these device-level mitigations are employed, the next step to holistically secure these devices is to evaluate potential PLC program vulnerabilities and review their mitigations.

3.2. PLC Programming Assessment

Beyond leveraging security controls that vendors have built into their products, the actual implementation of programs and logic running in PLC devices affect the overall security of their implementation. Despite running legacy protocols with no built-in security, asset owners and operators can leverage programming best practices that have been common in the IT community inside their PLC and RTU programs. Some of these areas include properly defined interfaces between logic and functional blocks. Others

include data type and range validation. Although these techniques are not a one-to-one mapping between conventional programming and PLC ladder logic/function blocks, many of the constructs do carry over.

One specific security construct in computer programming that carries over to PLC programming is the notion of limiting the scope, or access, to variables. In referring to minimizing scope, McConnell states, "…you should declare each variable to be visible to the smallest segment of code that it needs to see it" (Code Complete, 2004, p. 251). Vendors, such as Rockwell, have implemented this security practice by providing the option to configure Tag variable scope. For Rockwell PLCs specifically, tag scope can be set to Controller Level or Program Level, which determines if the tag value can be read and written from within a set of programs running on a PLC. Program scope tags, which have local scope, cannot be used for external communication directly, such as with EtherNet/IP or Modbus TCP, and depend on Controller Scope tags to interact outside of their program (Allen-Bradley, 2018, p. 25). Upon review of the CLICK PLC, tag scope could not be restricted or segmented to various programs, and therefore this feature was not tested as all memory locations are globally accessible.

Validating inputs is a critical component in secure computer programming. With regards to PLC ladder logic, function blocks, or structured text, this concept is no different. According to OWASP, "Input validation should happen as early as possible in the data flow, preferably as soon as the data is received from the external party" (OWASP, 2020). For external connectivity, this translates to validating external data from a Modbus TCP connection. As shown in Figure 13, the addresses of DS131 and DS132 are populated directly by external data from the CybatiWorks kit.

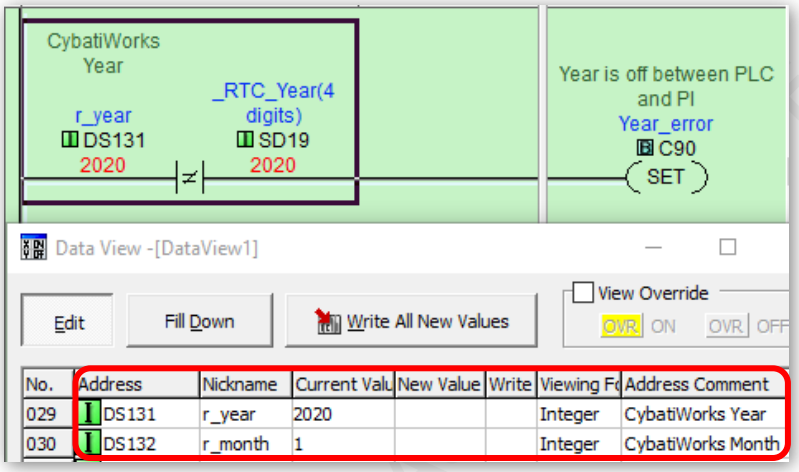


Figure 13. PLC logic with no input testing

Because the tags can be written from an external source, the data value of DS131, which is always being overwritten by the Modbus read block, can also be overwritten by an attacker. In this example, a latching "SET" is used to capture the event due to how fast the values are being overwritten from the CybatiWorks data. By leveraging the ctmodbus tool again, the Modbus address of 130 (displayed as 131 in Figure 13) is manipulated by forcing a value from the remote laptop, as shown in Figure 14.

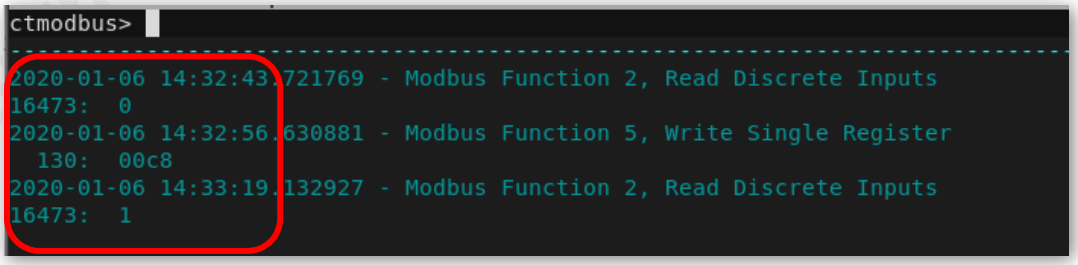


Figure 14. ctmodbus tool used to manipulate a register holding year data

The Modbus register mapped for the SET instruction at address C90 is 16474, but the actual register area in memory is at 16473, which is most likely due to a one-off implementation difference with the address labels starting at one versus the actual memory addresses starting at 0. After hexadecimal value of 00c8 (200 in decimal) is written to 130 (DS131 in Figure 13), the compare logic sees a difference in values and sets the internal C90 bit to True, which is reflected in the second Read Discrete Inputs

Michael Hoffman, mjhoffman80@gmail.com

request return value of 1, and is further illustrated in Figure 15 with SET output highlighted in blue.

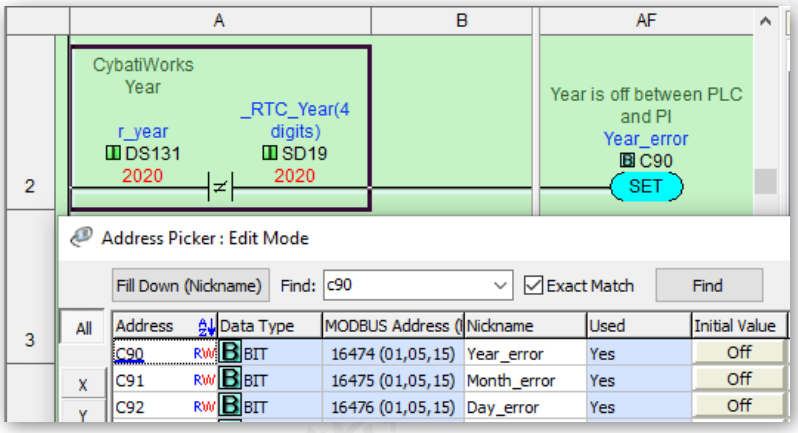


Figure 15. PLC program manipulation due to an external overwrite

A possible mitigation for this attack is to provide input range checking for the values. If a malicious individual or program exploits a vulnerable protocol, the values passed by the input validation will at least be in the range that the PLC program is designed to handle. As shown in Figure 16, the CybatiWorks hour value is read into register DS134 and verified to ensure the value is greater than or equal to 0 and less than or equal to 24. If the value falls between these ranges, it is copied over to a separate register (DS87 in this case) for use in the PLC logic.

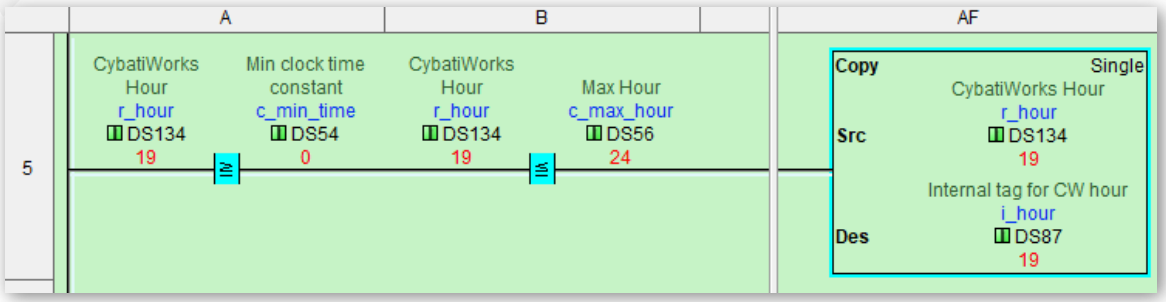


Figure 16. PLC inputs with range checking

As shown in Figure 17, the range validation assessment is carried out by first reading the value of DS87, which is input register 86. Upon the first read request, the register provides a value of Hex of 13, or Decimal 19.

Michael Hoffman, mjhoffman80@gmail.com

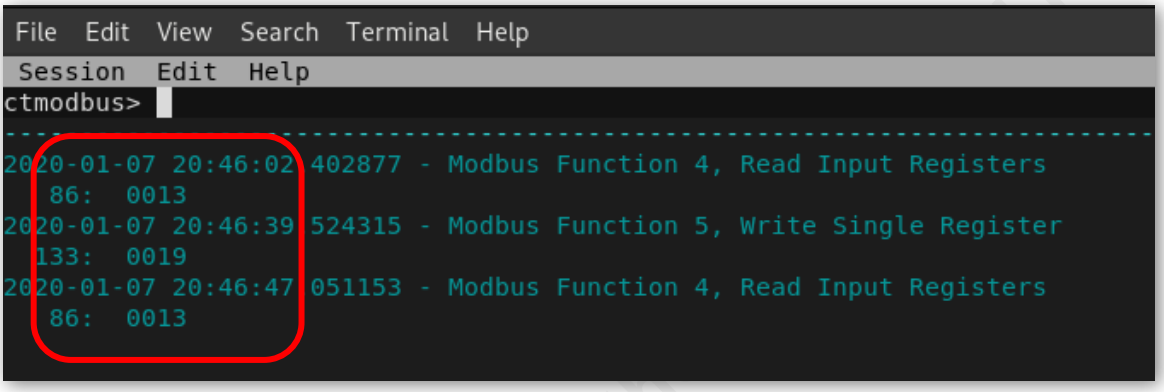


Figure 17. Testing input validation against external overwrite

Register 133, which is DS134, is then overwritten by a value of Decimal 25, or Hex 19, and then DS87 is re-read to verify that the value is not changed, thus proving that the input validation logic is working. Despite still having the ability to overwrite a value within the input range validation area remotely, the program logic will be able to handle an input over-range or under-range condition, thereby avoiding a logic error or PLC Processor fault.

Another essential reason to separate inputs from logic is to provide a snapshot of the values for orderly program flow and execution. PLCs have historically operated in a synchronous, cyclic fashion where inputs are read, logic solved, and outputs are written. Nevertheless, many newer PLCs can communicate asynchronously between input, communication, and output modules during program execution. This topic is referred to as *buffering I/O data,* and as Scott describes, "…input and output values can change in the middle of a program scan and [can] put the program in an unpredictable state (2015, p. 77)." Therefore, it is crucial to take a snapshot of all values used for program logic to ensure they remain unchanged, and then validate inputs before using them in PLC logic, as Figure 18 illustrates.

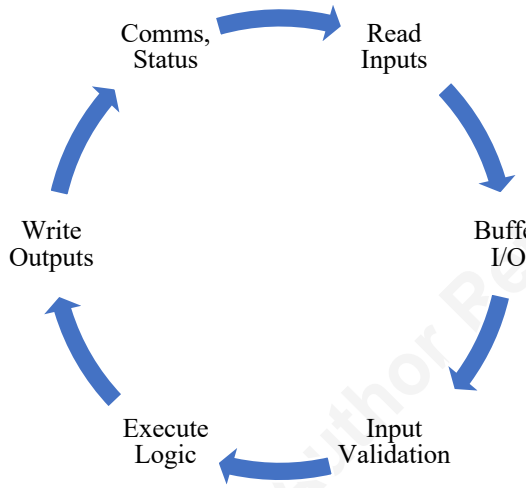


Figure 18. Example PLC Cycle with added buffering and validation

Despite the effort to buffer input data and validate range, an attacker could still overwrite values using insecure ICS protocols within the acceptable analog value range or with coils where range validation would still accept a 0 or 1. If an ICS cyber event were to occur, it could cause a PLC logic error, CPU fault, or, worse yet, affect the physical process under control. Therefore, a recovery function is necessary to restore operations after a cybersecurity event (NIST, 2018). As discussed by Shearer, Dely, Conway, and Robinson, the specific recovery, in this case, is to provide an initialization routine that injects safe startup values for the respective logic and process (2019). One method to carry out this feature is to leverage a first scan bit, which is energized only upon the first PLC scan cycle, as shown in Figure 19.

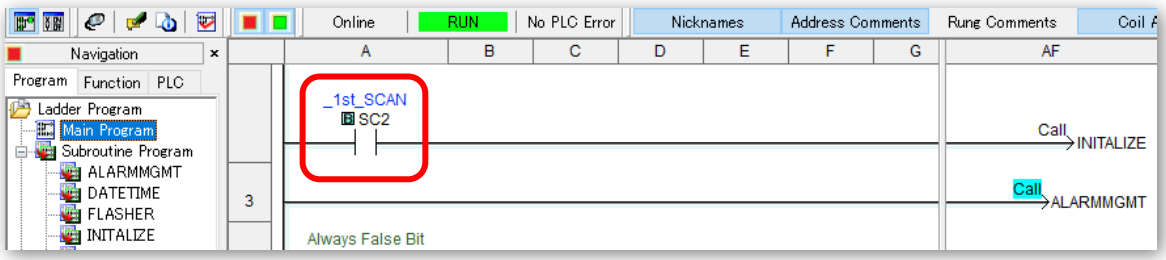


Figure 19. PLC First Scan Bit used to call the INITIALIZE subroutine

Depending on the PLC manufacture, the first scan bit can be set using the programming software or a local mode switch on the PLC processor. The CLICK PLC provides a RUN/STOP switch to change PLC modes. However, cycling power to the processor will

always cause the first scan bit to transition from false to true (0 to 1, respectively) for one cycle. Because the Main Program is continuously executed, utilizing a first scan bit to trigger the initialization subroutine is an effective way to compartmentalize code and ensure the subroutine is only executed once. As shown in Figure 20, the initialization subroutine both resets and sets latching outputs.

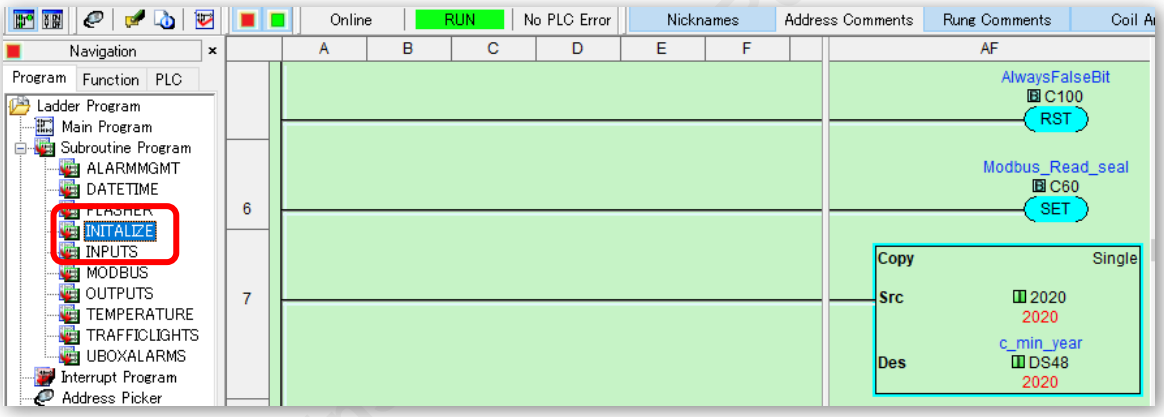


Figure 20. INITIALIZE subroutine example

In PLCs, a latching output is held in a true or false condition for an indefinite period until a logical condition causes a state change. In this example, the initialization subroutine is also leveraged to move constant integer values into memory locations for range checking or other critical program functions. These pre-engineered values injected from the initialization program ensures safe process startup conditions.

Although the discussed programming practices are necessary for secure program design and implementation, they do nothing to protect the ICS device perimeter from malicious abuse of ICS protocols or specially crafted packets. Therefore, an ICS-specific Application layer firewall is the next logical mitigation step to protect ICS devices using vulnerable communication protocols.

3.3. ICS Application Layer Firewall Assessment

Firewalls are border protection devices and control communication between different levels of trust (CISA, 2020). In the ICS space, firewalls are commonly used to define the boundary between the corporate network and ICS network, create the ICS DMZ, and further segregate levels of trust deeper in the control network (NIST, 2015, pp.

5-12). Firewalls inspect and control traffic at different layers of the TCP/IP stack, and thus can have varying degrees of granularity in their applicability. Packet filtering firewalls are stateless and inspect at the Internet layer for source and destination IP addresses. Session-based firewalls maintain a session table and inspect/control communication at the Transport layer where TCP and UDP protocols are found, which includes the session, port number/protocol, and flags -- among other information. At the Application layer, a firewall inspects and controls the actual application communication. Modbus TCP commands, for example, are found in the Application layer. Therefore, to provide granular inspection and control for Modbus TCP communication, an Application layer firewall is required.

The level of trust is reduced as communication extends beyond the boundary of an ICS device. The amount of risk involved in this reduction of trust is mostly dependent on the security and medium of the communication protocol but is also dependent on the controlled physical process. If a PLC or RTU is used for read-only monitoring of a natural gas well, for example, the risk of compromise is less than it would be for a PLC or RTU that controls well pressure or flow rate. To mitigate the risk of ICS systems that are controlling critical processes, and yet are still using legacy protocols, an ICS Application layer firewall can be used to allow only required commands per the control application.

To understand and assess how Modbus can be restricted, a Moxa EDR-G903 model secure router was leveraged. Initially, the device was loaded with firmware version 5.0. Upon checking with the vendor's website, version 5.4 was available and subsequently installed. To restrict communication to the CLICK PLC specifically, the Moxa was placed in front of the PLC using the LAN port on the device and was set to bridge mode, also called transparent mode, which makes it appear like a switch on the network. The Moxa WAN1 port was connected to the upstream Cisco switch with WAN2 port disconnected. Application layer firewall functionality in the Moxa comes with the capability to inspect the Modbus commands and registers themselves. The device can support Layer 2 Policy, Layer 3 Policy, and a Modbus Policy. The firewall assessment began by configuring Layer 3 rules to allow unobstructed Modbus communication and

enable programming software from the ICS612 VM to communicate with the CLICK PLC, as shown in Figure 21.

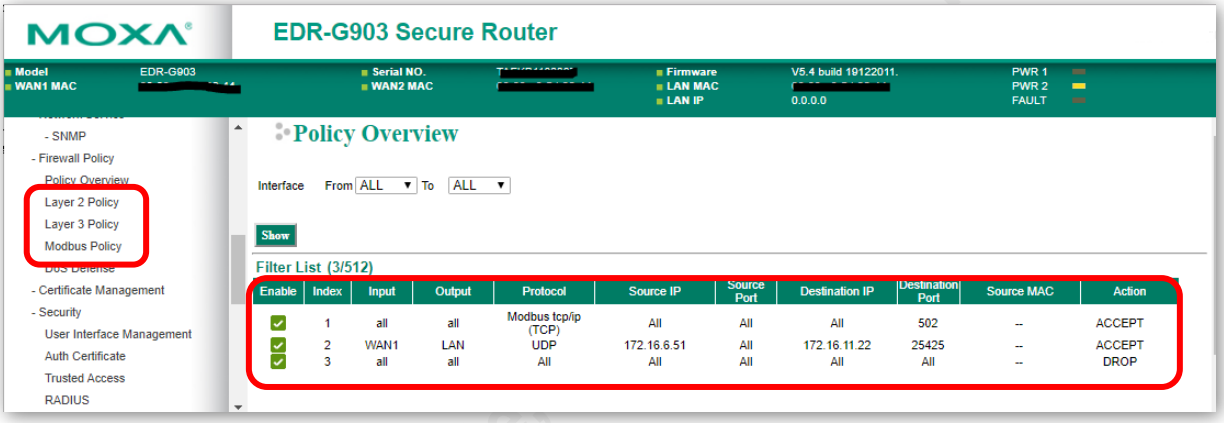


Figure 21. Initial settings applied to the Moxa firewall

As expected, the ctmodbus tool reveals that read/write functionality from an external network is possible. Figure 22 shows a successful Read Input Register at memory location 103, which returns a value of 07e4 hex value (2020 decimal). The value of 2019 (07e3 hex) is written back to the register, and then the value is re-read as 07e3, which indicates the value was changed.

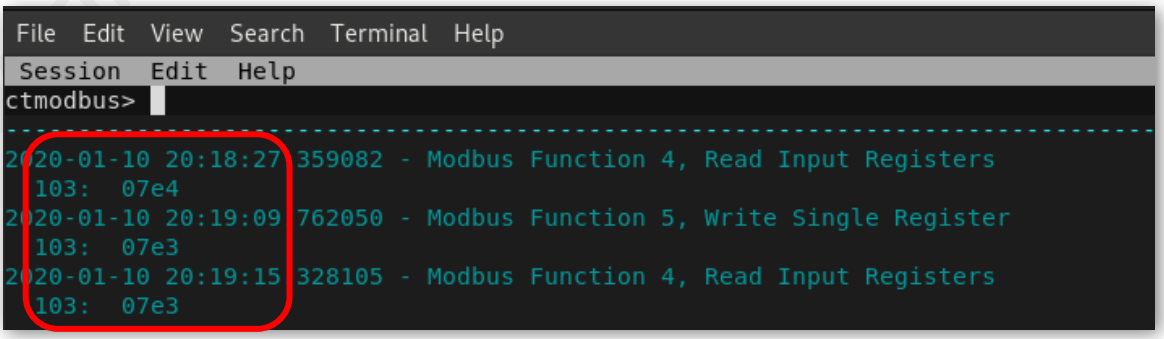
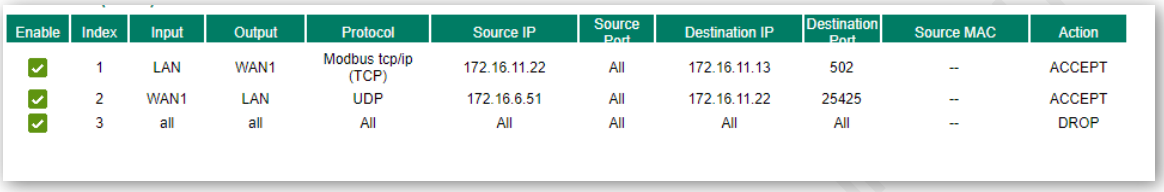


Figure 22. External register manipulation

The next assessment was to tighten up the connection setting and only allow port 502 connectivity between the CLICK PLC at 172.16.11.22 and CybatiWorks PI kit at 172.16.11.13 address, as shown in Figure 23.

Michael Hoffman, mjhoffman80@gmail.com



Enable	Index	Input	Output	Protocol	Source IP	Source Port	Destination IP	Destination Port	Source MAC	Action
☑	1	LAN	WAN1	Modbus tcp/ip (TCP)	172.16.11.22	All	172.16.11.13	502	--	ACCEPT
☑	2	WAN1	LAN	UDP	172.16.6.51	All	172.16.11.22	25425	--	ACCEPT
☑	3	all	all	All	All	All	All	All	--	DROP

Figure 23. Additional settings applied for Layer 3 firewall configuration

With firewall source/destination physical ports and source/destination IP addresses assigned on rule 1, the connectivity was no longer able to be established from the ctmodbus application. This is illustrated by Wireshark capturing TCP SYN packets sent from the ControlThingsIO VM at 172.16.6.52 to the PLC at 172.16.11.22, but there was no connection established as shown in Figure 24.

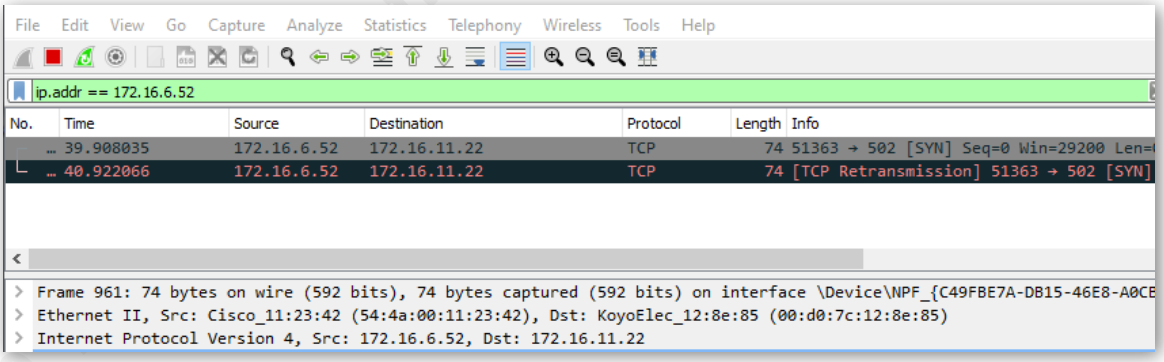
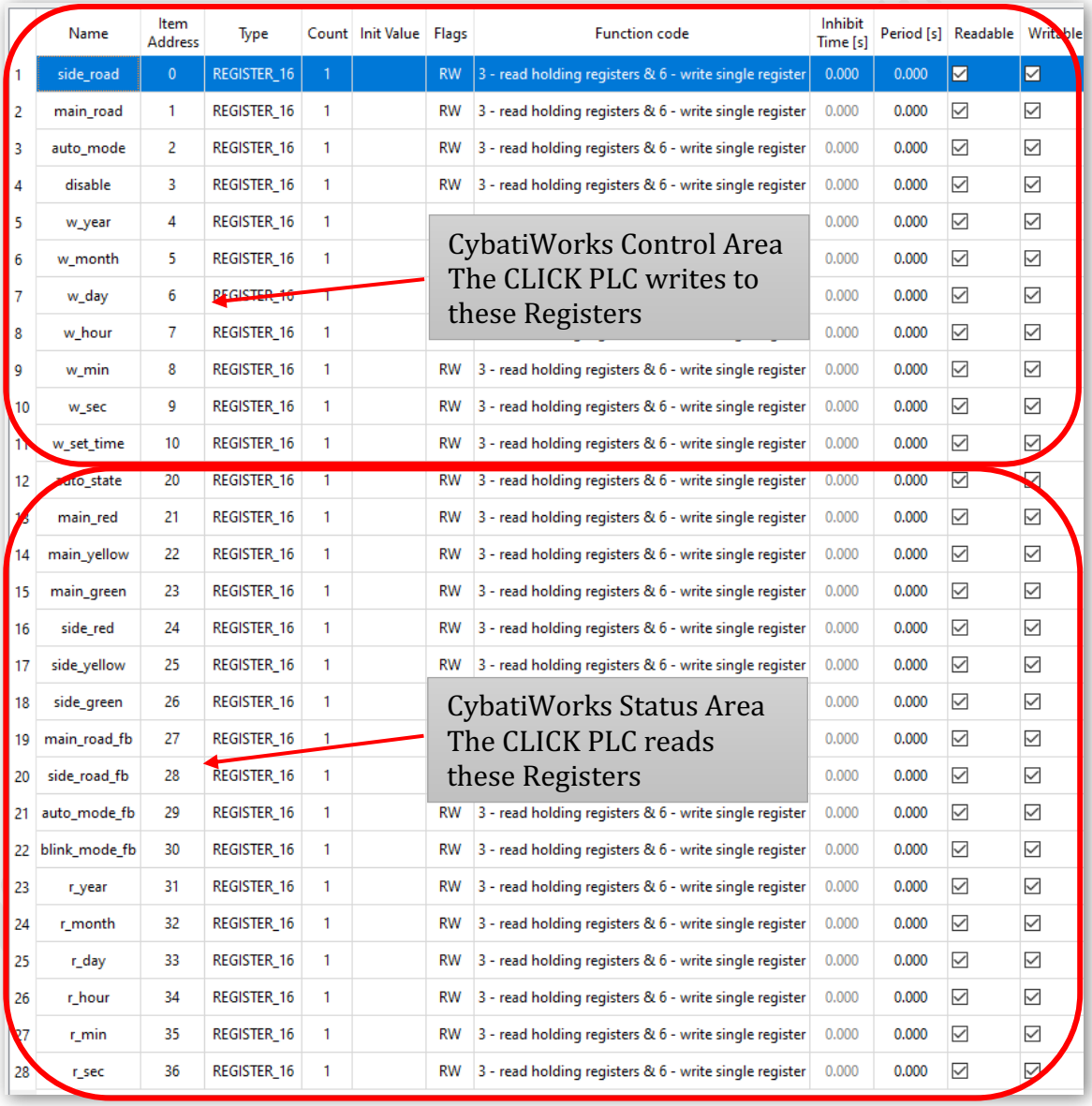


Figure 24. Failed TCP session creation due to tightening firewall rules.

Although a Man In The Middle (MITM) attack on Modbus TCP, as discussed by Sanche, could still be leveraged with this connection, the placement of the remote laptop is outside of the ICS VLAN (2017). Therefore, a MITM attack will not natively work. If an attacker was able to pivot down to a functional level 1 zone or abuse a trusted connection between a Modbus master station and slave, additional firewall restrictions would be required to protect the ICS device endpoint(s).

Applying additional protections for Modbus TCP is performed at the Application layer. However, before creating a firewall ruleset, the underlying function codes and ranges for the given ICS devices must be understood. For Modbus TCP, this information is typically found by reviewing Modbus master pulling configuration, and Modbus slave mapping tables. For the systems under assessment, this information was found by

reviewing the modified CybatiWorks Modbus mapping table, as shown in Figure 25.



	Name	Item Address	Type	Count	Init Value	Flags	Function code	Inhibit Time [s]	Period [s]	Readable	Writable
1	side_road	0	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
2	main_road	1	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
3	auto_mode	2	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
4	disable	3	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
5	w_year	4	REGISTER_16	1				0.000	0.000	☑	☑
6	w_month	5	REGISTER_16	1				0.000	0.000	☑	☑
7	w_day	6	REGISTER_16	1				0.000	0.000	☑	☑
8	w_hour	7	REGISTER_16	1				0.000	0.000	☑	☑
9	w_min	8	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
10	w_sec	9	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
11	w_set_time	10	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
12	auto_state	20	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
13	main_red	21	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
14	main_yellow	22	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
15	main_green	23	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
16	side_red	24	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
17	side_yellow	25	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
18	side_green	26	REGISTER_16	1				0.000	0.000	☑	☑
19	main_road_fb	27	REGISTER_16	1				0.000	0.000	☑	☑
20	side_road_fb	28	REGISTER_16	1				0.000	0.000	☑	☑
21	auto_mode_fb	29	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
22	blink_mode_fb	30	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
23	r_year	31	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
24	r_month	32	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
25	r_day	33	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
26	r_hour	34	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
27	r_min	35	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑
28	r_sec	36	REGISTER_16	1		RW	3 - read holding registers & 6 - write single register	0.000	0.000	☑	☑

Figure 25. CybatiWorks REXYGEN modified Modbus configuration

Mapping tables are used in ICS devices as a bridge between Modbus register or coil addresses and underlying logic or programs. To incorporate the principle of least privilege in the Moxa firewall ruleset, the CybatiWorks Modbus write and read register ranges of 0-10 and 20-36, respectively, are mirrored. The resulting firewall rules configured in the Moxa are shown in Figure 26.

Michael Hoffman, mjhoffman80@gmail.com

Modbus List (5/256)

Index	Enable	Input	Output	Protocol	Source IP	Destination IP	Slave ID	Function Code	Address	Action
1	✓	LAN	WAN1	TCP	172.16.11.22	172.16.11.13	1	3: Read Holding Registers	20~36	ACCEPT
2	✓	WAN1	LAN	TCP	172.16.11.13	172.16.11.22	1	3: Read Holding Registers	--	ACCEPT
3	✓	LAN	WAN1	TCP	172.16.11.22	172.16.11.13	1	16: Write Multiple Registers	0~10	ACCEPT
4	✓	WAN1	LAN	TCP	172.16.11.13	172.16.11.22	1	16: Write Multiple Registers	--	ACCEPT
5	✓	ALL	ALL	All	--	--	0	All	--	DROP

Figure 26. Moxa Firewall configuration Modbus specific registers

A unique requirement for the Moxa Modbus Application Layer rules is that a response from the slave device must be explicitly listed on a reverse rule -- despite having an underlying established TCP session. The first rule at Index 1 is for a Read Holding Register request from the CLICK PLC at 172.16.11.22 with an address range of 20-36. The slave response is listed at Index 2, with the traffic direction reversed but using the same function code. For Modbus writes, Index 3 allows function code 16, or Write Multiple Registers, with an address range of 0-10. This rule matches both the CybatiWorks Modbus configuration and the Click PLC Modbus write Configuration, shown in Figure 27. Due to the one-off addressing of the PLC, write register addressing starts at 400001 with 11 master addresses. This translates to a range of 0-10, or 11 addresses starting at address 0. The same reverse rule requirement holds for Modbus writes, and is the purpose for the rule at Index 4.

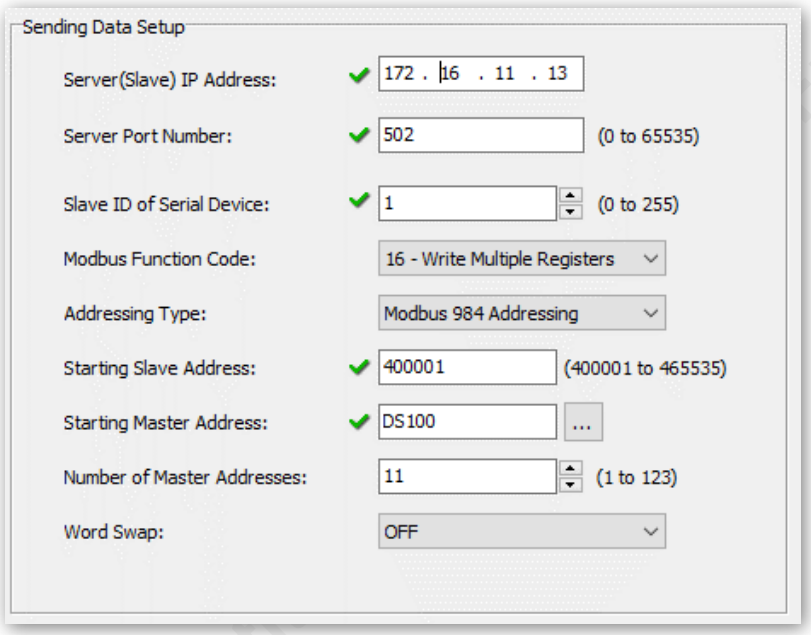


Figure 27. Click PLC Modbus Write block configuration

An important area to note is that the firewall rule(s) must incorporate the entire range of coils or registers requested or written. If a Modbus firewall inspection rule range is shorter than the Modbus function range requested by the master, the firewall will drop the entire packet. This is illustrated by increasing the number of master address from 17 to 18, as shown in Figure 28.

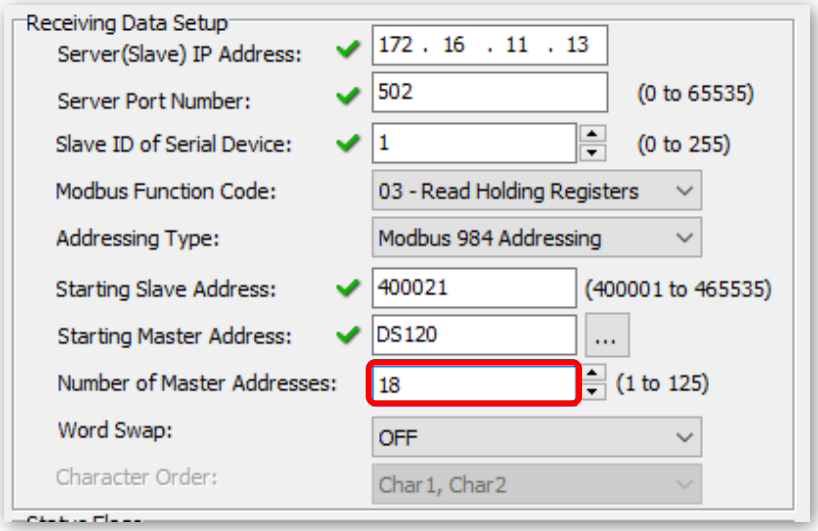


Figure 28. Click PLC Modbus Write block configuration

Michael Hoffman, mjhoffman80@gmail.com

Before this assessment, however, the Wireshark connection was moved to the network tap between the CLICK PLC and Moxa firewall to capture the firewall response to the modified Modbus request values. As shown in Figure 29, when the request word count was changed from 17 to 18 in the PLC configuration, the Moxa firewall detects this change and closes the connection to the CybatiWorks PI. The closed connection was also immediately evident on the PLC HMI as the traffic light screen stopped updating.

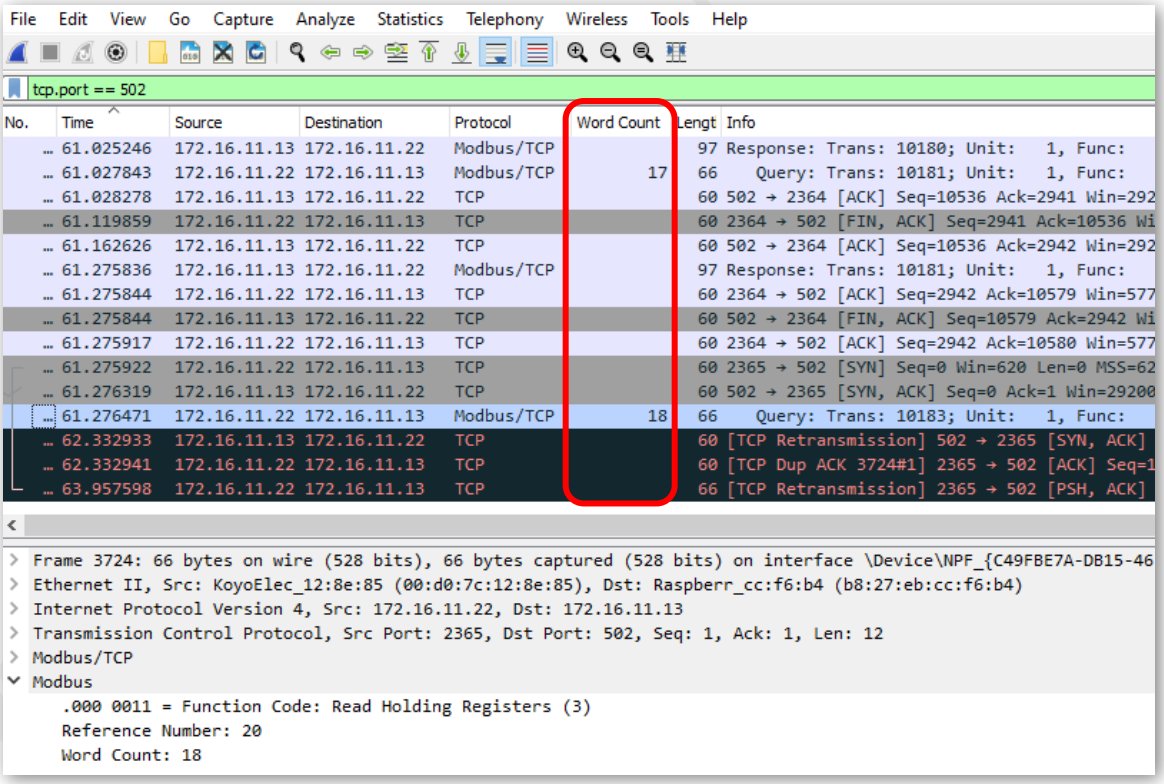


Figure 29. TCP session termination from word count change.

This assessment shows that by leveraging ICS protocol aware application layer firewalls and configuring granular rule sets, the control of insecure ICS protocol device communication and, therefore, the security of ICS devices themselves is significantly increased. However, increased security is only fully realized when a proper reference architecture, such as the Purdue Model, is implemented, that provides layers of security controls to protect these critical devices from systems and communication above the control network and beyond the ICS.

Michael Hoffman, mjhoffman80@gmail.com

4. Recommendations and Implications

The mitigating controls discussed in this paper are focused on preventative and, in some small way, recovery controls to deal with the prevalent use of insecure ICS protocols in lower functional levels of the Purdue Model. A commonly heard motto in the security community says, "prevention is ideal, but detection is a must." However, detection builds on the foundation of prevention. Therefore, architectural and passive defenses, including device hardening, patching/updating firmware, implementing secure logic and programs, employing network and zone segmentation, among others, provide a higher value of return and lay the foundation for detection capabilities and ultimately Active Defense (2018). Lee describes, "..Active Defense is more achievable and efficient when done in an environment with proper Architecture and Passive Defenses (2015, p. 4)." With preventative and architectural controls designed and adequately maintained, defenders can then leverage tools such as Network Security Monitoring (NSM) to detect adversarial behaviors in ICS networks and respond to attempted attacks.

4.1. Recommendations for Practice

Many of the mitigation controls discussed can be performed on operational environments, but there is always risk involved with making any changes to PLCs, RTUs, or other ICS devices. Additionally, making network changes can disrupt communication that could cause loss of view or control. Therefore, the most opportune time to make ICS devices or network modifications is during planned operational outages with scheduled maintenance windows. However, this does not prevent a thorough assessment of the asset owner's and operator's ICS devices and systems to understand their security posture and threat landscape and begin to plan and take necessary mitigating actions.

For new installations or upgrades, ICS devices should be implemented with vendors recommended security practices for hardening, programming, and network restrictions. Finally, if devices have the option of using a more secure protocol, it should be reviewed to determine applicability for the environment and used if possible.

4.2. Implications for Future Research

The ICS security community would benefit from researching device security settings and controls, programming and logic, and network perimeter security -- each in more detail to further understand vulnerability and mitigations and feed those learnings back to the community. Additionally, this research reviewed only one PLC vendor, and thus, a further refined study could be carried out for other vendors as well. Finally, although Modbus TCP is widely used and easy to assess, other protocols, such as EtherNet/IP or DNP3, would be good candidates to review and contrast with their secure versions.

5. Conclusion

Despite the increasing trend of attacks on ICS systems, vulnerable ICS protocols are still in use and are even leveraged in new ICS projects. However, ICS owners and operators do have options to mitigate vulnerabilities inherent in these insecure protocols. Research performed on ICS device security settings, secure programming practices, and deploying network security barriers using student kits from SANS ICS515 and ICS612 courses show that it is possible to increase the security posture of ICS devices and their respective Modbus TCP communications. Although these are preventative mitigations, they enable a strong foundation to build a defensible approach in ICS environments that directly impacts the controlled process. Deploying these mitigations is ultimately dependent on the level of risk in a respective ICS environment. However, asset owners and operators are encouraged to assess their environment to understand the state of ICS devices, protocols, and communication flows and carry out any remediation activities to ensure critical infrastructure is kept operational today and into the future in light of the increasing cyber threats to ICS.

Michael Hoffman, mjhoffman80@gmail.com

6. Appendices

The supporting device configurations used throughout this research are posted in GitHub and can be found at the following link:

https://github.com/Eirene77/ICSProtocolLab

References

Allen-Bradley. (2018, February). *Documents.* Retrieved from Rockwell Automation: https://literature.rockwellautomation.com/idc/groups/literature/documents/pm/1756-pm004_-en-p.pdf

Allen-Bradley. (2018, November). *Literature.* Retrieved from Rockwell Automation: https://literature.rockwellautomation.com/idc/groups/literature/documents/pm/1756-pm016_-en-p.pdf

Assante, M. J., & Lee, R. M. (2015, October). *Reading Room.* Retrieved from SANS: https://www.sans.org/reading-room/whitepapers/ICS/industrial-control-system-cyber-kill-chain-36297

Bodungen, C. E., Singer, B. L., Shbeeb, A., Hilt, S., & Wilhoit, K. (2017). *Hacking Expose, Industrial Control Systems: ICS and SCADA Security Secrets & Solutions.* New York: McGraw Hill Education.

Center for Internet Security. (2020, January 4). *Secure Configuration for Hardware and Software on Mobile Devices, Laptops, Workstations and Servers*. Retrieved from Center for Internet Security: https://www.cisecurity.org/controls/secure-configuration-for-hardware-and-software-on-mobile-devices-laptops-workstations-and-servers/

CISA. (2020, January 11). *Control System Firewall*. Retrieved from US-CERT: https://www.us-cert.gov/ics/Control_System_Firewall-Definition.html

ControlThings I/O. (2020, January 2). *Tools*. Retrieved from ControlThings.io: https://www.controlthings.io/tools

CSIA. (2016, September). *Improving Industrial Control System Cybersecurity with Defense-in-Depth Strategies.* Retrieved from US-CERT: https://www.us-cert.gov/sites/default/files/recommended_practices/NCCIC_ICS-CERT_Defense_in_Depth_2016_S508C.pdf

Dragos. (2018). *Year In Review 2018.* Retrieved from Dragos: https://dragos.com/wp-content/uploads/yir-ics-vulnerabilities-2018.pdf

Draias, Z., Serhrouchni, A., & Vogel, O. (2015, July). *Taxonomy-of-attacks-on-Industrial-control-protocols.* Retrieved from Research Gate: ttps://www.researchgate.net/profile/Zakarya_Drias2/publication/283045654_Taxo

nomy_of_attacks_on_Industrial_control_protocols/links/568b808108ae1e63f1fce049/Taxonomy-of-attacks-on-Industrial-control-protocols.pdf

Langer, R. (2013, November). *Resources.* Retrieved from Langner: https://www.langner.com/wp-content/uploads/2017/03/to-kill-a-centrifuge.pdf

Langner. (2019, February 12). *THE FIVE THINGS YOU NEED TO KNOW ABOUT OT/ICS VULNERABILITY AND PATCH MANAGEMENT*. Retrieved from Langner: https://www.langner.com/2019/02/the-five-things-you-need-to-know-about-ot-ics-vulnerability-and-patch-management/

Lee, R. M. (2015, September 1). *The Sliding Scale of Cyber Security.* Retrieved from SANS: https://www.sans.org/reading-room/whitepapers/ActiveDefense/sliding-scale-cyber-security-36240

Lee, R. M. (2017). *White Papers.* Retrieved from Dragos: https://dragos.com/wp-content/uploads/CrashOverride-01.pdf

Lee, R. M. (2018). *ICS515 | ICS Active Defense and Incident Response.* Bethesda: SANS Institute.

Lee, R. M., Assante, M. J., & Conway, T. (2014, December 30). *Industrial Control Systems Library.* Retrieved from SANS: https://ics.sans.org/media/ICS-CPPE-case-Study-2-German-Steelworks_Facility.pdf

McConnell, S. (2004). *Code Complete.* Redmond: Microsoft Press.

Mostia, W. L. (2019, January 4). *Introduction to Modbus.* Retrieved from Control Global: https://www.controlglobal.com/articles/2019/introduction-to-modbus/

NIST. (2015, May). *Guide to Industrial Control Systems (ICS) Security.* Retrieved from NIST: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf

NIST. (2018, April 16). *Cybersecurity Framework.* Retrieved from NIST: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf

OPC Foundation. (2020, January 4). *OPC UA Information Models*. Retrieved from OPC Foundations: https://opcfoundation.org/developer-tools/specifications-opc-ua-information-models

OWASP. (2020, January 6). *Input Validation Cheat Sheet*. Retrieved from OWASP: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

OWASP Top Ten. (2020). *Top 10 Web Application Security Risks*. Retrieved from OWASP: https://owasp.org/www-project-top-ten/

Rinaldi, J. S. (n.d.). *Modbus.* Retrieved from Real Time Automation: https://www.rtautomation.com/technologies/modbus/

Rockwell Automation. (2019, July 22). *Press Release Details*. Retrieved from Rockwell Automation: https://ir.rockwellautomation.com/press-releases/press-releases-details/2019/New-Communication-Module-With-CIP-Security-Can-Reduce-Risks-Boost-Performance-/default.aspx

Sanchez, G. (2017, October 20). *Man-In-The-Middle Attack Against Modbus TCP Illustrated with Wireshark.* Retrieved from SANS Reading Room: https://www.sans.org/reading-room/whitepapers/ICS/man-in-the-middle-attack-modbus-tcp-illustrated-wireshark-38095

Scott, A. (2015). *Learning RSLogix 5000 Programming.* Birmingham, UK: Packt Publishing.

Shearer, J., Dely, J., Conway, T., & Robinson, C. (2019). *ICS612 | ICS Cyber Security In-Depth.* Bethesda: SANS Institute.

Siemens. (2016, March 20). *Product Support.* Retrieved from Siemens: https://cache.industry.siemens.com/dl/files/010/90885010/att_959937/v1/77431846_Security_SIMATIC_DOKU_V20_en.pdf

Slowik, J. (2019). *Whitepapers.* Retrieved from Dragos: https://dragos.com/wp-content/uploads/Evolution-of-ICS-Attacks-and-the-Prospects-for-Future-Disruptive-Events-Joseph-Slowik-1.pdf