CSEC 793 Capstone in Computing Security
Project Report

ASSESSING SECURITY OF CONTEMPORARY INDUSTRIAL CONTROL SYSTEMS: INVESTIGATED THROUGH MODBUS HIJACKING

April 27, 2023

Christopher Tremblay
Department of Computing Security
College of Computing and Information Sciences
Rochester Institute of Technology
cst1465@rit.edu

1 Abstract

Industrial Control Systems (ICS) security is a critical concern in the context of Industry4.0, where increased connectivity and data-driven practices have revolutionized modern factories. This paper examines two common ICS structures, one involving a simple network of sensors and a Programmable Logic Controller (PLC) communicating via Modbus/TCP, and the other involving an additional software platform, Ignition by Inductive Automation, integrated with the network. The paper demonstrates specific security exploits that can be carried out against these structures, highlighting the risks introduced by poorly implemented networking practices. The results of this paper provide valuable insights into Industry4.0 vulnerabilities and reinforce the need for strong network security practices.

2 Introduction

In the last century, strides have been made in how production lines are controlled within factories. Industrial Control Systems (ICS) originally were complex circuits of relays wired to one another and to different parts of the machine to facilitate basic automation. In the late 1960s, Modicon created the "Standard Machine Controller"[8] that aimed to replace large relay cabinets with a single microcontroller, known today as Programmable Logic Controllers (PLCs). These highly specialized industrial devices are designed to make quick decisions based on machine signals and typically come with networking capabilities.

With the growing need to log and analyze data, factories are increasingly interconnecting PLCs and similar devices. Networking these devices can provide benefits such as faster alarming when something is wrong, better data analytics, and off-site monitoring. However, if not implemented correctly, these practices can also introduce security risks that need to be addressed. The purpose of this paper is to exploit two different ways ICSs can be structured in practice. The first structure is a simple network composed of a PLC that communicates via Modbus/TCP, and sensors. The second one is a PLC that communicates via Modbus/TCP, sensors, and Ignition[15]. The main contribution of this paper will be demonstrating an attack on Ignition, as it is a relatively new technology with limited literature on vulnerabilities.

Ignition is a powerful industrial application software platform that is used to build Human Machine Interfaces (HMIs) and Supervisory Control and Data Acquisition (SCADA) systems. This software platform has gained widespread popularity in recent years due to its flexibility, scalability, ease of use, and ability to leverage contemporary computing techniques. However, as with any critical infrastructure component, it is essential to evaluate the security of Ignition to ensure that it is not vulnerable to cyber attacks. In particular, Ignition is often used as the front-end for critical infrastructure systems, making it a target for attackers seeking to gain access to sensitive data or disrupt operations. Therefore, in this paper, I will demonstrate a successful Man In The Middle (MITM) cyber attack on an Ignition system to highlight the importance of securing this critical component of an ICS.

3 Background

3.1 Modbus Protocol

The Modbus protocol is a straightforward method of communication between client and server devices. In industrial automation, servers often consist of sensing devices, while clients are controllers interested in the values of those sensors. Modbus can be used to communicate in two different ways: through a query/response method for communication between client and server, or through broadcast communication, where a client sends a command to all servers [1]. To distinguish between commands sent to a client, Modbus employs function codes. For example, the function code `0x01` corresponds to the ReadCoils function code, which is used to retrieve the status of specified coils in the remote device. A coil in Modbus refers to a digital output on the client device that controls a physical component, such as a valve or a motor. In practice, a coil is a wire that can be toggled on or off to activate or deactivate the associated device. In other words, a coil is a switch that sends a signal to the sensing device to perform a specific action. In response, the remote device echoes back the same function code, along with ON/OFF values of the coils encoded as single bits.

Modbus was first published in 1979 when industrial control systems were isolated, and networking them was impractical. As a result, the designers of Modbus did not include basic security features such as encryption, authentication, and integrity in the protocol's design [1]. Today, Modbus and similar IoT/SCADA protocols, such as Profibus and DNP3, are vulnerable to attacks due to changes in the threat landscape since their inception. These protocols were created during an era before security was a major concern.

Overall, while Modbus was not originally designed with security in mind, steps can be taken to secure Modbus communications. For example, using virtual private networks (VPNs) can encrypt and authenticate communication between client and server devices, while firewalls can restrict access to the Modbus network. Additionally, secure Modbus variants, such as Modbus/TCP Secure and Modbus Secure, offer encryption, authentication, and integrity protection. Regular security assessments and penetration testing can also help identify and address potential vulnerabilities in the Modbus network. Nonetheless, Modbus was chosen for this project due to its popularity in industrial settings, its open-source nature, and its availability at no cost.

3.2 Ignition

Ignition is a powerful industrial automation software platform designed to help businesses create and deploy custom industrial applications. The platform offers a suite of tools for developing, deploying, and managing a wide range of automation solutions, including SCADA, HMI, MES, and Industrial IoT (IIoT) applications. Ignition provides a modular, flexible, and scalable architecture that allows users to build applications tailored to their specific needs, while also offering robust data analytics and visualization capabilities.

Ignition also integrates with a wide range of industrial devices, systems, and protocols, making it a versatile and powerful solution for modern industrial automation needs.

The platform also includes a Python scripting engine that allows users to develop custom scripts and applications for their industrial automation needs. This provides a high degree of flexibility and customization, allowing users to create automation solutions tailored to their specific requirements. The Python scripting engine can be used for a variety of tasks, such as data analysis, machine learning, and advanced control algorithms, making Ignition a versatile platform for industrial automation.

Ignition was chosen for this project due to its increasing popularity among factories, and ease to get running quickly.

4 Related Work

There are many infrastructures beyond the scope of factories that rely on Microcontroller Units (MCUs) and sensors to automate processes. Since the `Stuxnet` worm [2], the presence of industrial-based cyberweapons has become a significant threat that requires attention to protect critical processes and supply chains. Controls and industrial networks are highly customized solutions that typically use proprietary software, hardware, and protocols, making it challenging to find literature about generic attacks. Most of the literature is very specific or generic, without much middle ground. Therefore, the reviewed literature fell into two categories: Surveys and Taxonomies or Specific Attacks.

4.1 Surveys and Taxonomies

In their paper [4], Zhu et al. classified attacks into three broad categories: cyber attacks on hardware, cyber attacks on software, and attacks on the communication stack. The hardware attack section was brief, mentioning that an attacker may have physical or remote access to change set points and disable alarms. Access control through passwords and physical barriers was suggested as a solution. The attacks on software section had three subsections: lack of privilege separation in embedded operating systems, buffer overflows, and SQL injections. One important takeaway was that many PLCs and higher-end MCUs run versions of VxWorks, the most popular embedded device operating system. Another takeaway was that SQL injections can be particularly dangerous since these systems often store and load process parameters from databases, providing a path for changes to be made through SQL injection. The final attacks discussed were DoS attacks on TCP and UDP, along with a brief survey of protocols such as Modbus and DNP3. This paper provided a good introduction to industrial security attacks, without going into too much detail on any one topic. It was particularly relevant at the time it was written, as industrial attacks were still relatively new in 2011.

The paper by Ghosh and Sampalli [9] is a refinement of Zhu et al.'s work, with a better taxonomy and additional sections on quantum computing attacks and modern standards

of SCADA security. They also provide a comprehensive list of standards from different organizations for security in industrial environments. The taxonomy of attacks is classified by two metrics: which layer in the five-layer OSI model is being attacked and which aspect of security is being attacked (confidentiality, integrity, availability, non-repudiation). The paper also discusses modern standards of SCADA security, such as current detection and prevention-based standards that are being used today. Overall, the paper is an excellent resource for organizing thoughts on which attacks are viable for which aspect of SCADA when crafting an attack.

In [14], Duggan provides a high-level overview of how to begin penetration testing (pen-testing) an industrial network. While the general framework of the pen-test is the same, there are differences in how one must perform those steps in industrial environments. For example, instead of a ping sweep, a pen-tester might examine SAT/CAM tables on switches or chase wires. Traditional pen-test methods may not be applicable in the environment because they may lack support for a ping sweep, or it may disrupt and cost the company money in lost productivity. Duggan also describes a ping sweep that caused a factory to hang and caused up to $50K of damage, highlighting the importance of considering the potential impact of pen-testing on industrial systems. Thus, this paper provides new ways of scanning and fingerprinting hosts when traditional means may not be the correct approach, emphasizing the importance of adapting pen-testing methodologies for industrial environments.

Sayegh et al., the authors of [6], propose a survey that focuses solely on internal attacks on SCADA systems. This paper assumes that the attacker already has access to the network and can monitor traffic to and from the PLCs. The authors discuss network and application layer attacks such as replay attacks, UDP reflection attacks, and fragmentation attacks, most of which target availability and cause DoS attacks. Although this paper provides valuable insights into these attacks, my paper focuses more on gaining control of the machine rather than simply taking it down. However, this paper provides useful implementation details that enable attackers to execute these attacks with relative ease. This was particularly beneficial as other papers mainly focused on theoretical frameworks with few implementation details.

The Department of Homeland Security (DHS) also has a survey of cyberattacks in ICS systems [3] that seems to be a response to the `Stuxnet` attack since it was written in 2011. In this paper, the authors describe common ICS vulnerabilities, configuration weaknesses, and security weaknesses. For vulnerabilities, these would include things like poor code quality, improper validation, poor credential management, and more. Configuration weaknesses would include poor permissions and access controls, improper authentication, and poor planning/policies/procedures. Finally, security weaknesses would include weak firewalls, common network design weaknesses, and poor security audits. Two things this paper had that others did not was weak network designs and best practices to mitigate risk. This is useful because the other papers didn't include any content on weak network design, which is part of the scope of the project. The mitigations they provided were quite comprehensive

as well. These did point to other government documents, but it was much more content than the other papers had given.

4.2 Specific Attacks

Parian et al. [10] present a Man-in-the-Middle (MITM) attack on Modbus. The authors describe the process of ARP poisoning the master and slave to reroute all traffic through the attacker's machine. This allows the attacker to easily forge communications by modifying packets and sending fraudulent responses back to the master due to the predictability of the protocol. The paper details the implementation and provides examples of how the attack can be executed. This paper is useful for understanding the specific techniques used in this type of attack and the potential impact on a SCADA system.

Chen et al. in [7] discuss a Modbus MITM attack and a TCP SYN Flood DoS attack in their paper. They did a similar attack where ARP poisoning was used of the router and master devices. They did not go into the implementation details like Parian et al. did, however, they went over some network and coil results. For example, what traffic looks like when the attacker is initially setting up the ARP poisoning. They also went over a TCP SYN Flood DoS attack where the attacker just starts flooding the master with TCP SYN requests.

Mohammed et al. in [11] were among the few authors who actually used a physical PLC to perform their attacks, while other papers used virtualized environments. The use of a higher-end PLC, such as a `Siemens Logo!`, allows us to observe how an actual network created by a vendor stack would react under such circumstances instead of relying on a virtualized environment. This paper summarized the implementation of a simple attack that involves sending forged packets to the HMI and PLC. This attack does not involve any sophisticated techniques like ARP poisoning or MITM, but rather relies on sending properly formatted packets to the devices and manipulating coils.

The related works on Modbus and ICS security attacks provided valuable insights into the possible attacks that can be conducted on industrial systems and aided in crafting the attacks detailed in this paper. The study conducted by Zhu et al. [4] provided an important entry-level understanding and served as an entry point for crafting the attacks once the systems and protocols were decided upon. Since the Modbus protocol is the primary focus of this paper, more specific papers and attacks, such as Mohammed et al. [11], helped find targeted attacks for the scope of the project. By analyzing and building upon other works in the field in tandem with the Modbus Specification [5], a viable cyberattack was developed and is described in detail in this paper.

5 Project Implementation

The exploits developed in this project were written in Python3 and predominantly used `scapy` and NetfilterQueue in tandem with the Linux utilities `iptables` and

`libnetfilter_queue`. The first exploit written was a basic Modbus gratuitous request attack to showcase the lack of authentication in the Modbus protocol. The second exploit was a MITM attack against Ignition developed by Inductive Automation[15] with a Modbus/TCP device connected. The code can be found in the Github repository[20].

5.1 Gratuitous Request

The gratuitous request exploit consists of crafting a completely legitimate Modbus packet and sending it from a malicious host. There is no security mechanism for authentication for client-to-server communication so the server device will blindly accept the packet performs the request by the client, and send a response back.

This section of the project was implemented in Python3 using the `pymodbus`[13] library. Its functionality was straightforward; a `ModbusTCPClient` connection was opened up to a server, and the library `write_coil()` function was used to send a completely legitimate packet from a malicious host.

5.2 MITM Attack on Ignition

This attack was more complex than the previous one and comprised of a 3 stage attack pipeline. The 3 stages were a CAM Overflow attack, ARP Poisoning attack, and a Modbus Hijacking attack. Each one of these was packaged into a separate Python module, and then tied together into one final exploit module.

5.2.1 CAM Overflow Module

The CAM Overflow module would craft custom ARP requests and send them into the switch to overflow the switch's CAM table. This would result in the switch acting like a hub, and sending all traffic as a broadcast instead of a unicast.

To do this the module generates a random MAC. It then creates an ARP packet and sends it through the switch so that a new entry is created in the CAM table. This is done over and over again until all the memory has been used, and no more MACs can be stored on the switch.

5.2.2 ARP Poisoning Module

The ARP Poisoning Module is what facilitates the traffic between the Ignition Gateway and PLC to be passed through the adversary's computer. It does this by sending gratuitous ARP Replies. For example, if the adversary wants to sniff traffic from the PLC to the Ignition Gateway, they would craft an ARP Reply that has the destination IP and MAC of the Ignition Gateway, the source IP from the PLC, and the source MAC as the adversary's MAC. This creates or overwrites the entry in the ARP cache of the device that maps the Ignition Gateway IP to the adversary's machine.

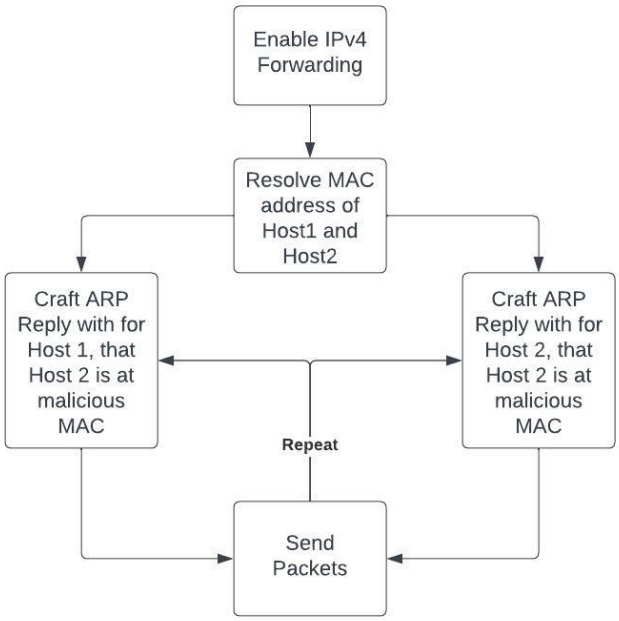

Figure 1: ARP Poison Module Flowchart

Figure 1 shows the flow of operations in the module where Host1 and Host2 are the Ignition Gateway and PLC, respectively.

5.2.3 Modbus Hijacking Module

The Modbus Hijacking Module is responsible for processing the packets and deciding which ones to drop, forward, and craft. The program starts first by creating an `iptables` rule that routes all traffic into a NetfilterQueue. Each packet is taken from the NetfilterQueue and is goes through 3 checks: does it have an IP header, does it have a Modbus/TCP payload, is the Modbus/TCP Payload have a WriteCoil function code? This can be seen in Figure 2. If the answer to all those questions was "yes" then the packet is dropped, since the goal is to interrupt control requests. The packet is accepted in every other case.

When the packet has been discovered to have a WriteCoil function code, the payload is saved in a Python `dict()` keyed by the TransactionID. The program then extracts what the WriteCoil request is doing, and crafts a new payload that does the opposite. For example, if Ignition requests to turn a coil ON, then the new packet will have a crafted payload to turn the coil OFF. The crafted packet is then sent off to the PLC.

Once the PLC receives the Modbus request, it will perform the WriteCoil request and send a response back to Ignition. The NetfilterQueue will catch this response and drop it

the same was as it dropped the request. Note, in Modbus the request and response share the same TransactionID, and the WriteCoil response just echoes the payload of the original request. The TransactionID is then cross-referenced in the `dict()` to get the original payload sent by Ignition and that original payload is loaded into a new packet and sent back to Ignition.

In summary, this module intercepts and drops a WriteCoil packet. The payload is saved, and a new payload is crafted to do the opposite action (turn a coil off, when it was requested on) and is put in a new packet sent to the PLC. Upon reception of the response, the response is dropped and the original payload is loaded into a new packet and sent to Ignition to make it appear the device echoed back the original request. This makes Ignition think the WriteCoil request was completed successfully.

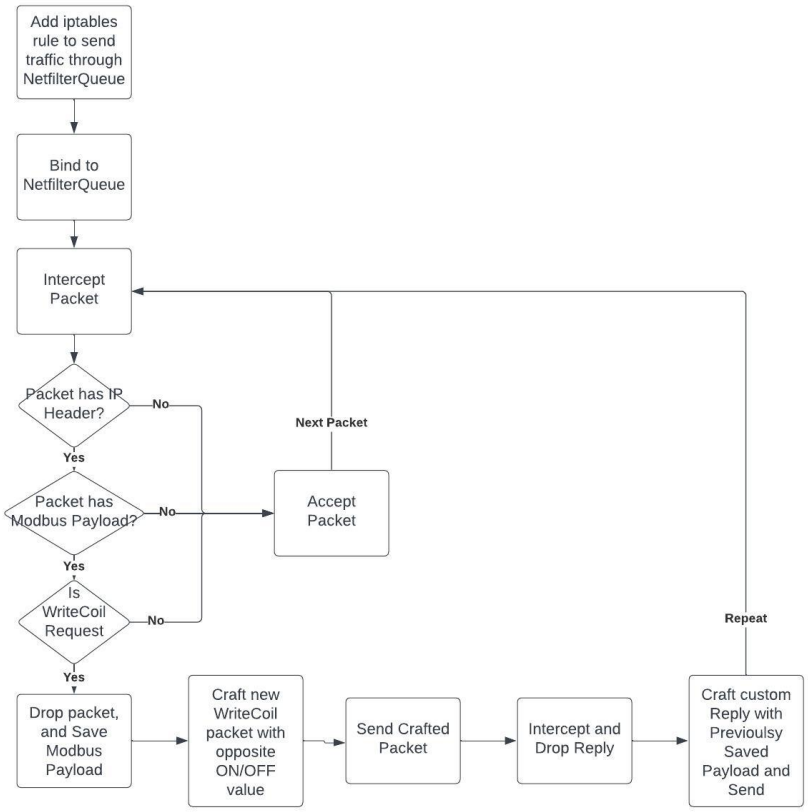

Figure 2: Modbus Hijacking Module Flowchart

6 Testbed Environment

The setup used for the validation of this exploit was comprised of 3 machines. First, a Windows 10 machine that simulated a production line, which included simulation software (FactoryIO v2.5.4[17], OpenPLC Runtime v3[12]) for sensors and a PLC. Second, a Windows 10 machine that acted as the Ignition Gateway (v8.1.26[15]). Third, a Linux machine was used as the adversary. These 3 devices were all networked together using a Cisco 3550 Multi-Layer Switch running on factory default settings. The setup can be seen in Figure 3. The addresses and descriptions of the machine can be seen in Table 1.

Figure 3: Lab Environment. Production Line PC (left), Ignition Gateway PC (right), adversary (middle).

OS	IP	MAC	Description
Windows 10	192.168.1.2	d8:bb:c1:7e:3c:a9 (Micro-St_7e:3c:a9)	Simulated Production line
Windows 10	192.168.1.3	d8:bb:c1:7c:8e:15 (Micro-St_7c:8e:15)	Ignition Gateway Server
Arch Linux	192.168.1.254	00:00:1b:3c:05:86 (NovellNo_3c:05:68)	Adversary

Table 1: Addresses of Lab Machines

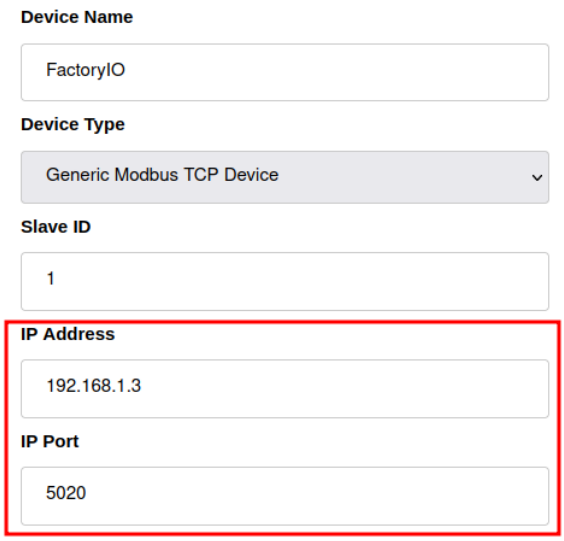

Figure 4: FactoryI/O Slave Device Configuration in OpenPLC Runtime

6.1 Windows 10 PC: Production Line

The first Windows 10 machine simulated a production line. To do this 2 pieces of software were needed. One to simulate sensors, and one to simulate a PLC.

When choosing how to evaluate sensors for output different solutions were considered MSP-430 MCUs, RaspberryPis, and simulation software. Simulation software was ultimately chosen for ease of demonstration of the project. Factory I/O v2.4.5 [17] was chosen for its robustness in protocols, and for how visually it can demonstrate factory operation.

When deciding on PLC simulation software, the goal was to have it closely resemble commercial-grade development and deployment software used in practice today. Some examples of commercial-grade development software include Siemens TIA Portal[19] and Studio 5000[18]. Features such as programming in ladder logic and the ability to compile and upload to a runtime were evaluated when searching. OpenPLC v3[12] was chosen as the PLC simulation software because of its open-source nature, and ease of use. In the current state of PLC simulation software, it is probably the best simulation software that most closely resembles high-end PLC development. It allows for development in ladder logic and compilation to a runtime software that communicates via Modbus/TCP. A simple solution to the Assembler scene included in FactoryI/O was developed in OpenPLC Editor and used in testing. The solution is available on the project GitHub repository[20].

In the testbed, OpenPLC was configured with FactoryI/O as a slave device with network settings shown in Figure 4, and Modbus addressing settings shown in Table 2. To avoid conflicts with the Ignition Gateway server, the default port of 502 was changed to 5020.

Address Type	Start	Size
Discrete Inputs (%IX100.0)	0	16
Coils (%QX100.0)	0	12
Input Registers (%IW100)	0	0
Holding Registers - Read (%IW100)	0	0
Holding Registers - Write (%QW100)	0	1

Table 2: FactoryI/O Modbus Addressing in OpenPLC Runtime

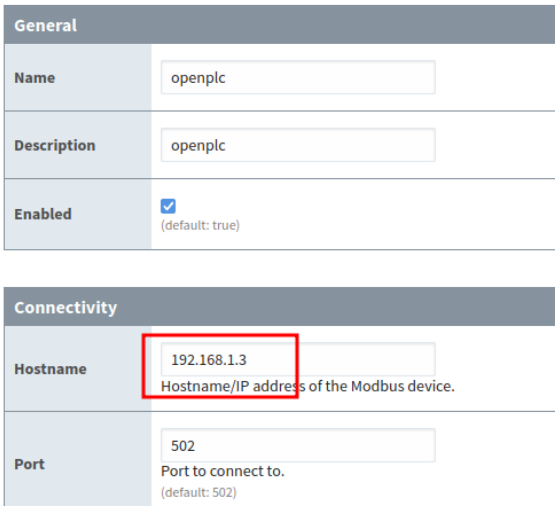

Figure 5: OpenPLC Runtime Network Settings in Ignition

6.2 Windows 10 PC: Ignition Gateway Server

The Ignition Gateway Server v8.1.26[15] was configured with the OpenPLC Runtime as a Modbus/TCP device pictured in Figure 5. A simple project developed in Ignition Designer was used in testing. The project had an assortment of buttons that actuated different parts of the FactoryI/O Assembler machine. For example, buttons to turn the conveyor belts on and off. The Ignition project, tag import file, and OpenPLC Runtime address configuration files can all be found on the project GitHub[20].

6.3 Linux Machine: Adversary

The Linux machine used was running Arch Linux. The only special configuration to this machine was setting its IP statically to the one listed in Table 1.

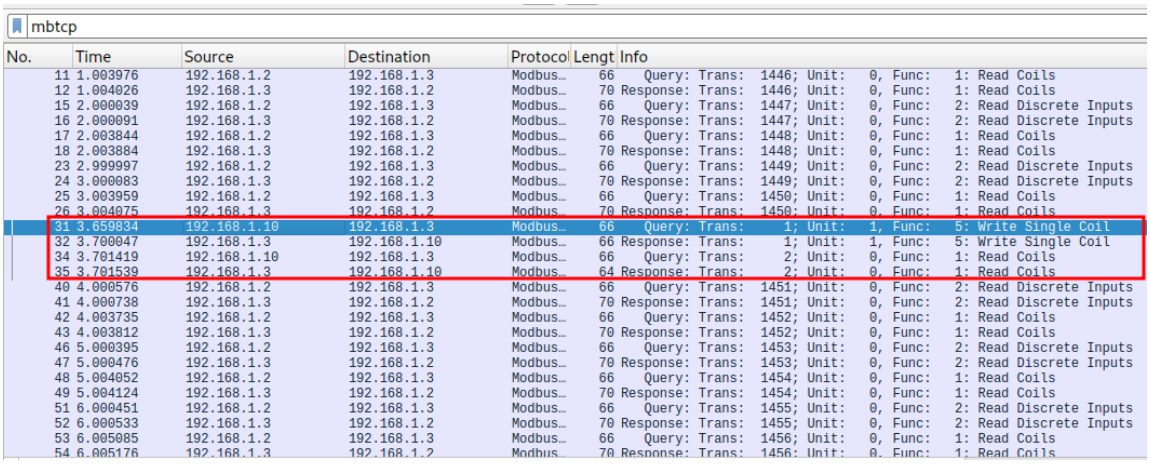

Figure 6: Gratuitous Request Wireshark Capture

7 Results

7.1 Gratuitious Request Exploit

This exploit was successful within the lab environment. This is a very well-known vulnerability within Modbus. In Figure 6, the Wireshark capture depicts normal traffic and then in the middle of the traffic a gratuitous request boxed in red.

7.2 MITM Attack on Ignition

This exploit was successful within the lab environment. The ARP Poisoning attack can be seen working in Figure 7 and Figure 10. Figure 7 is a Wireshark capture on the Ignition Gateway Server Windows machine. It can be seen that the destination MAC address changes from the PLC MAC to the adversary MAC. Similar results can be seen in Figure 10, which depicts a Wireshark capture on the Simulated Production Line machine. It is also shown in Figures 8 and 9 that show the change in ARP caches for the the Ignition Gateway PC (192.168.1.3) before and after the ARP Poisoning attack. Figures 11 and 12 show the same for the Production Line PC (192.168.1.2).

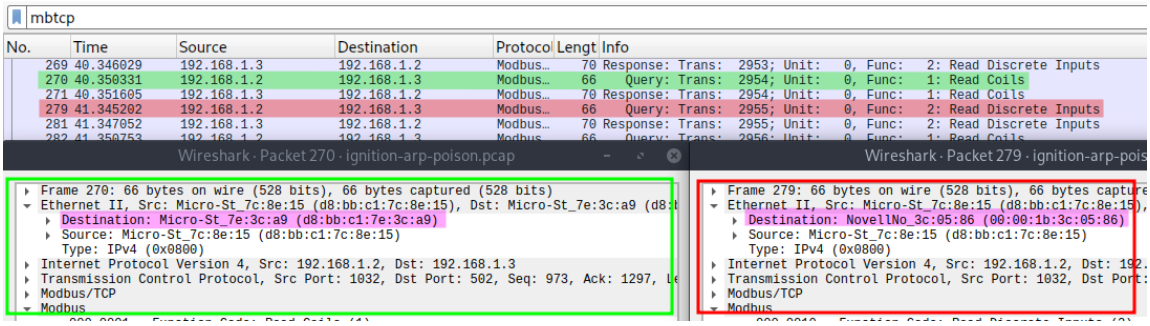

Figure 7: Wireshark Capture of ARP Poisoning on Ignition Gateway Server Windows PC

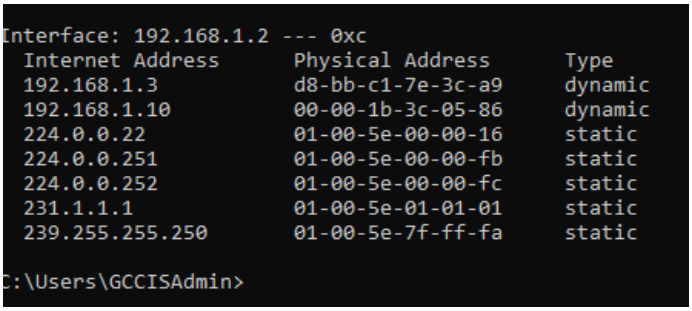

Figure 8: Ignition Gateway PC Before ARP Poison. Note the `192.168.1.3` MAC change to Adversary.

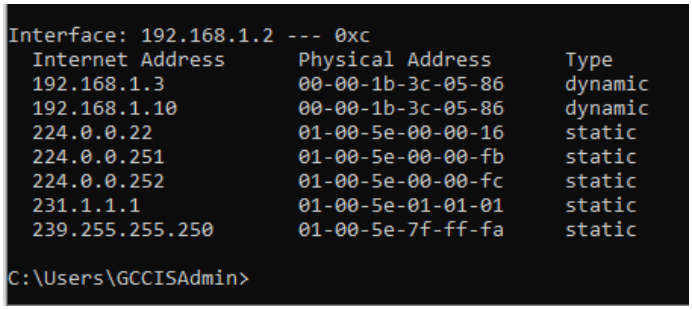

Figure 9: Ignition Gateway PC During ARP Poison. Note `192.168.1.3` MAC change to Adversary.

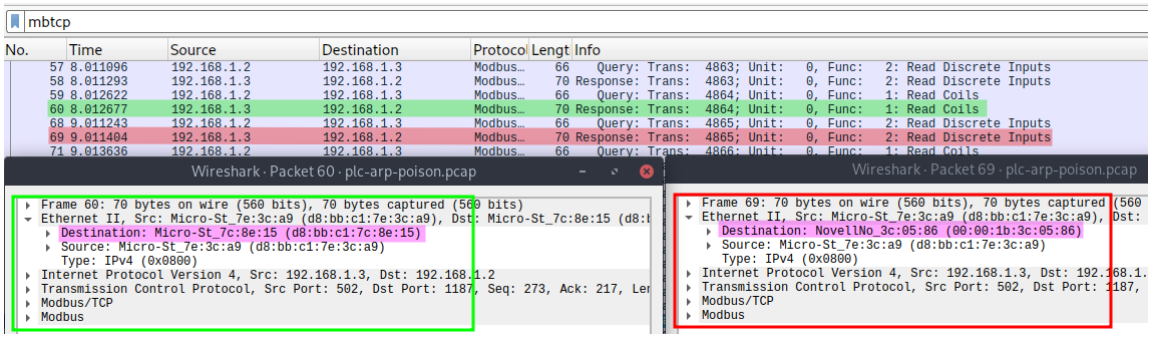

Figure 10: Wireshark Capture of ARP Poisoning on Production Line Windows PC

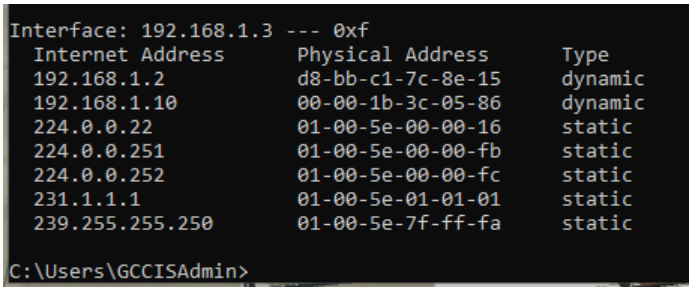

Figure 11: Production Line PC Before ARP Poison. Note `192.168.1.2` MAC change to Adversary.

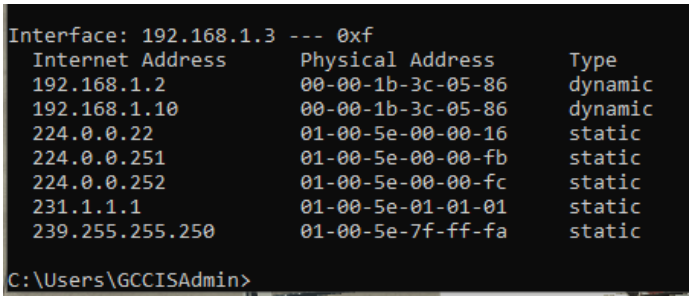

Figure 12: Production Line PC During ARP Poison. Note `192.168.1.2` MAC change to Adversary.

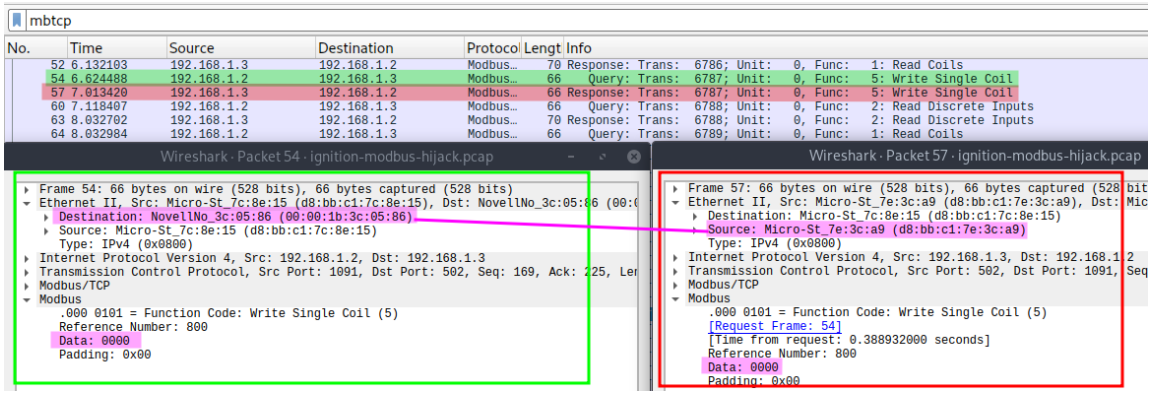

Figure 13: Wireshark Capture of Modbus Hijacking on Ignition Gateway Server PC

The Modbus Hijacking attack can be seen working in Figure 13, 14, and 15. Figure 13 shows the Ignition Gateway Server sending a Modbus WriteCoil request to the PLC and receiving a response. Nothing is corrupted enough to make Ignition throw an error however, there are artifacts of ARP Poisoning within the Modbus request Ethernet frame. It can be seen that the destination of the request (green in Figure 13) is the adversary machine, but the response (red) appears to come from the actual destination. This is a result of the ARP poisoning and the adversary crafting custom packets. It should also be noted that the request has a data field of `0x0000` since this is changed by the adversary. Figure 14 shows the PLC receiving the request and sending back the responses. Note that the request WriteCoil request value is opposite to what the Ignition Gateway request in Figure 13. Similar to the Ignition Wireshark capture, the request source (green in Figure 14) appears to be the actual Ignition Gateway but the destination of the response (red) is the adversary machine. It can also be seen that the adversary successfully manipulated the request to have a data field of `0xFF00`. Figure 15 shows depicts the adversary machine dropping, crafting, and forwarding the packets.

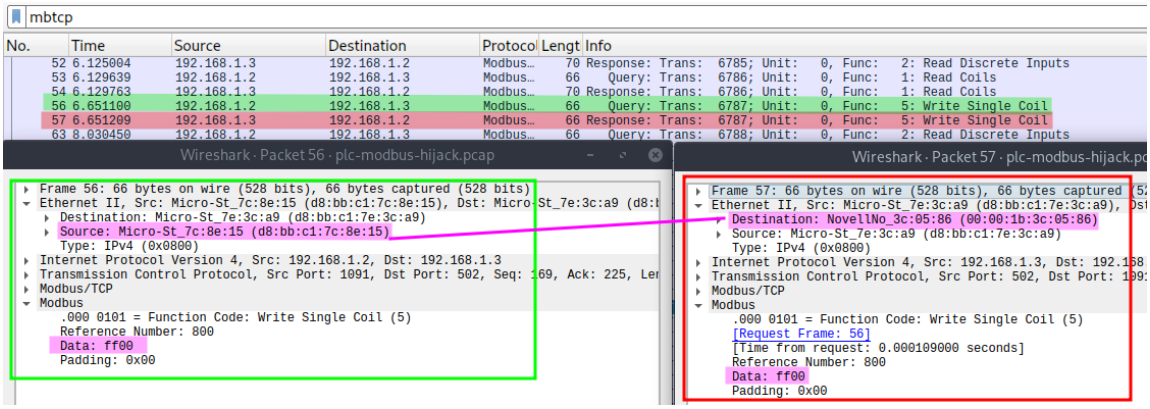

Figure 14: Wireshark Capture of Modbus Hijacking on Production Line PC

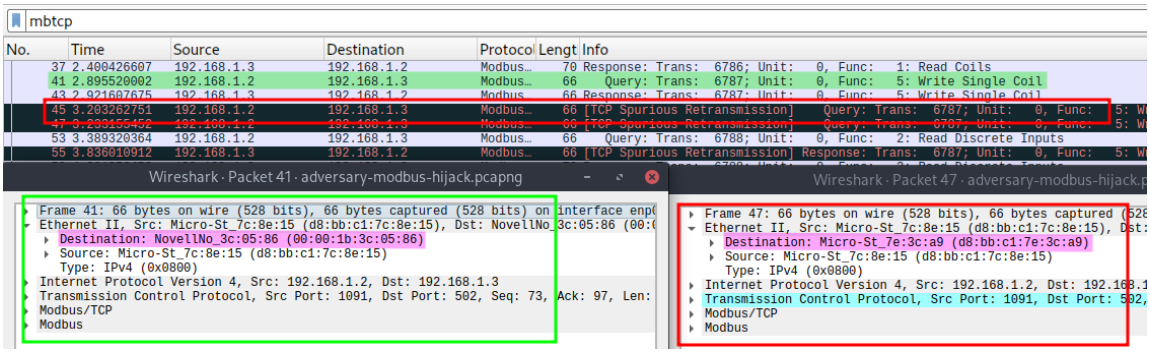

Figure 15: Wireshark Capture of Modbus Hijacking on Adversary Linux Machine

8 Challenges

Despite all of the preparation and planning, a couple of unexpected challenges emerged during testing and validation. These issues stemmed from the intricacies of the Linux kernel and its handling of network traffic, which required additional investigation into kernel documentation and experimentation to resolve. The first challenge that appeared was during ARP Poisoning. Packets were not being forwarded properly to the machines once they were poisoned. The Ignition Gateway Server was poisoned to have the PLC IP (and vice versa for the PLC to Ignition) mapped to the adversary MAC, causing the traffic to be dropped upon reception in the adversary machine. To resolve this Kernel IPv4 Forwarding[16] was enabled for the packets to be sent to the correct host. Since packet crafting does not happen at the ARP Poisoning stage, there is no code that manipulates the packets to correct the destination fields. However, this caused some odd ICMP Redirect packets to appear on the network. Due to time constraints, this was the solution chosen, and porting it over to a NetfilterQueue would be a better solution to avoid the unusual ICMP traffic.

Another challenge faced was properly dropping or accepting packets once the Modbus Hijacking module started. To do this, a NetfilterQueue was used and packets were inspected to see if they were WriteCoil requests. The packets were dropped to prevent unnecessary forwarding due to the IPv4 Forwarding setting.

9 Conclusion

The results of the project show that contemporary ICSs are still susceptible to various types of attacks. The experiments within the lab environment were successful in demonstrating that these attacks are still applicable despite advances in the industrial automation field. This particular attack is very harmful when it is hard to physically validate whether or not the device is accurately receiving the commands. For example, if Ignition is being used to communicate over large geographical distances, or the devices are in hard-to-reach spaces. Future work to make this project better would be to expand the Modbus hijacking module to include support for more than just the WriteCoil function code. For example, keeping state of the WriteCoil requests affects on coils so that the ReadCoil transactions can be spoofed based on previous transactions.

References

[1] Igor Nai Fovino et al. “Design and Implementation of a Secure Modbus Protocol”. en. In: *Critical Infrastructure Protection III*. Ed. by Charles Palmer and Sujeet Shenoi. IFIP Advances in Information and Communication Technology. Berlin, Heidelberg:

Springer, 2009, 83â96. ISBN: 978-3-642-04798-5. DOI: 10.1007/978-3-642-04798-5_6.

[2] Nicolas Falliere, Liam O Murchu, and Eric Chien. *W32.Stuxnet Dossier*. Feb. 2011.

[3] Department of Homeland Security. *DHS_Common_Cybersecurity_Vulnerabilities_ICS_2010.pdf*. May 2011.

[4] Bonnie Zhu, Anthony Joseph, and Shankar Sastry. "A Taxonomy of Cyber Attacks on SCADA Systems". In: *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*. Oct. 2011, 380â388. DOI: 10.1109/iThings/CPSCom.2011.34.

[5] *MODBUS APPLICATION PROTOCOL SPECIFICATION*. V1.1b3. Modbus Organization, Inc. Apr. 2012. URL: https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

[6] Naoum Sayegh et al. "Internal security attacks on SCADA systems". In: *2013 Third International Conference on Communications and Information Technology (ICCIT)*. June 2013, 22â27. DOI: 10.1109/ICCITechnology.2013.6579516.

[7] Bo Chen et al. "Implementing attacks for modbus/TCP protocol in a real-time cyber physical system test bed". In: *2015 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*. May 2015, 1â6. DOI: 10.1109/CQR.2015.7129084.

[8] Mallikarjun G. Hudedmani et al. "Programmable Logic Controller (PLC) in Automation". en. In: *Advanced Journal of Graduate Research* 2.11 (May 2017), 37â45. ISSN: 2456-7108. DOI: 10.21467/ajgr.2.1.37-45.

[9] Sagarika Ghosh and Srinivas Sampalli. "A Survey of Security in SCADA Networks: Current Issues and Future Challenges". In: *IEEE Access* 7 (2019), 135812â135831. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2926441.

[10] Christopher Parian, Terry Guldimann, and Sajal Bhatia. "Fooling the Master: Exploiting Weaknesses in the Modbus Protocol". en. In: *Procedia Computer Science* 171 (2020), 2453â2458. ISSN: 18770509. DOI: 10.1016/j.procs.2020.04.265.

[11] Abubakar Sadiq Mohammed, Neetesh Saxena, and Omer Rana. "Wheels on the Modbus - Attacking ModbusTCP Communications". en. In: *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. San Antonio TX USA: ACM, May 2022, 288â289. ISBN: 978-1-4503-9216-7. DOI: 10.1145/3507657.3529654. URL: https://dl.acm.org/doi/10.1145/3507657.3529654.

[12] Alves, Thiago. *Open-source PLC software*. Version v3. URL: https://openplcproject.com/.

[13] Collins, Galen and Iversen, Jan. *Pymodbus*. Version 2.5.3. URL: https://pypi.org/project/pymodbus/.

[14] David P Duggan. “Penetration Testing of Industrial Control Systems”. en. In: ().

[15] Inductive Automation. *Ignition*. Version 8.1. URL: https://inductiveautomation.com/.

[16] *ip-sysctl.txt*. URL: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt.

[17] Real Games. *Factory I/O*. Version 2.5.4. URL: https://factoryio.com/.

[18] Rockwell Automation. *Studio 5000 Design Software*. URL: https://www.rockwellautomation.com/en-us/products/software/factorytalk/designsuite/studio-5000.html.

[19] Siemens. *TIA Portal*. URL: https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html.

[20] Tremblay, Christopher. *Capstone Project*. URL: https://github.com/cstremblay2000/Capstone_Project.