MODELLING CYBER-ATTACKS ON MODBUS TCP PROTOCOL IN INDUSTRIAL CONTROL SYSTEMS

Final Report

AUTHOR – KALVIN WILLIS C1914891
SUPERVISOR – NEETESH SAXENA MODERATOR – YUHUA LI
School of Computer Science and Informatics 2022
BSc Computer Science with Security and Forensics
One Semester Individual Project (40 Credits)

Acknowledgements

I would like to thank my supervisor Neetesh Saxena for all of his support throughout this project. I have been kept motivated through our weekly meetings and the encouragement and enthusiasm I have received around my ideas has been so helpful.

I would like to thank Abubakar Mohammed for providing me insight into other Modbus attacks, how they operate and giving me a deeper understanding of real time scenarios and the true impact of attacking critical systems. It has made me appreciate the work I am doing more and showing me that it is relevant to the landscape we live in today.

I would also like to thank my parents for keeping me on track. Working on this project has been a test to my resolve, and they have made sure to keep me going even when it got difficult.

Abstract

The Modbus protocol was designed and developed to be used with Programmable Logic Controllers. Over time, as technology has progressed, the original protocol has been redeveloped to transmit data across ethernet/wireless connections rather than serial cables. This led to the creation of the Modbus TCP protocol, which achieves everything the original protocol did but across modern infrastructure.

It is now a de-facto standard in communication across Programmable Logic Controllers, which are commonly used in Industrial Control Systems. However, a major problem lies in the distinct lack of security within the protocol.

This project looks to demonstrate vulnerabilities in the Modbus protocol through the creation of an attack that exploits a gap in the current research into the protocol. Through background research, it became evident that Man in the Middle attacks hadn’t been looked at in too much detail.

As a part of this project, an automated tool was created to host a man in the middle attack. Once this tool was created, it was demonstrated against a python implementation of the Modbus TCP protocol and a discussion into how the attack would fair against known security measures was opened, considering the different solutions found through background research. The main conclusion taken away is that there are solutions available to prevent Man in the Middle attacks, however in most cases these will only ever detect an attack and will do little to prevent one.

Table of Contents

Table of Figures

Tables

1 Introduction

1.1 What is Modbus TCP?

The Modbus protocol was designed and developed by Modicon in 1979 for use with Programmable Logic Controllers (PLCs), commonly seen in Industrial Control Systems (ICS). Since then, it has become a de facto standard protocol, and is widely used to connect electronic devices in industry [1]. It is one of hundreds of Supervisory Controls and Data Acquisition (SCADA) protocols that have been created in the past three decades but is one of the most common found today due to its simplistic nature and design.

Modbus Transmission Control Protocol (TCP), alongside its predecessor Modbus, allows for communication between a Modbus Master, and Modbus Slaves, both of which are traditionally PLCs. It operates at level 7 of the OSI model [2], with the original Modbus protocol mainly transmitting data and information through physical serial lines. In the modern world, the original Modbus protocol has been superseded by Modbus TCP in the majority of control systems, where being connected with the more up to date systems will increase efficiency and mean less reliance on older, harder to maintain hardware.

1.2 Security issues with Modbus

Since the protocol is so common, if an attack is found against it that is difficult to defend against, most industry that uses PLCs could be impacted. An example of this is that the Modbus protocol transfers data with no encryption, meaning everything gets sent in plain text [3]. This has serious concerns for confidentiality, especially as critical infrastructure could be communicating using this protocol.

The majority of the hardware that runs both the original Modbus and Modbus TCP is so out of date that the security solutions present in the modern world are not compatible with it. For the world to move forward and exist without the need for insecure protocols, the hardware would have to catch up.

Between January 2021 and September 2021, there was an increase of 2,204% in Internet scanning of TCP port 502 connections [15], which is the port that uses Modbus. This shows that attackers are becoming increasingly aware of the vulnerabilities around the Modbus protocol and are taking the steps to start abusing them.

This project will attempt to solve a problem that is presented through background research carried out around currently existing cyber-attacks on the Modbus TCP Protocol. From this research, a research gap will be chosen in an area which needs deeper understanding or poses a larger threat within the protocol. The research gap chosen will then allow me to design and implement an attack within an automated tool that fills it. This will be done against a python simulation of the protocol in action, specifically pymodbus [4].

1.3 Project Aims and Objectives

The project has a wide scope that can be met with the resources available, meaning there will be little extra software, funding or approval needed. Within the Gantt chart provided in my initial plan, there are three objectives that will be met at clearly stated intervals, these being the following:

1 – Find a gap in the current research on the Modbus protocol that could have an attack developed against it.

2 – Implement an attack on the Modbus protocol to attack a simulated ICS. This should be within the research gap identified in objective one. The simulation will be a client/server pair, with the main target being the server.

3 – Develop a tool that can automate the attack developed previously and evaluate how effective it is compared to existing methods. This tool could include the functionality to produce attack performance statistics such as time to attack completion, or even comparing the attack to others that are mounted on the tool.

1.4 Project Stakeholders

The final product is specifically targeted towards researchers, as it should fill a gap in current research. Along with this, it will be beneficial towards industry, as the final tool I create will highlight vulnerabilities of the Modbus TCP protocol that would need to be addressed.

As mentioned previously, where this project will be beneficial for research and not released as a consumer product, I will not have to engage in ethical approval as there will be no user-testing or interaction with potential customers, nor will there be handling of personal data

1.5 Challenges

Within the project, there will be certain challenges that present themselves. Through the planning stage, it has been identified that Modbus is not a traditional protocol. It is typically only seen in ICS' and not in more common devices that would be more familiar. To complete this project to a satisfactory level, the protocol will need to be understood to a high degree, including but not limited to the effects changing a single bit in transmission can do. Along with this, it will need to be understood how pymodbus implements the protocol and if there are any major differences in its operation compared to the real version, as this could affect how the final tool would be transposed into a real environment.

1.6 Motivation

I have chosen this project as I have a deep-rooted interest in cyber security, specifically penetration testing. This project will improve my skills in this area, along with taking advantage of my understanding of python and further pushing me to understand and create more complicated code. I also want to demonstrate the importance of security around the Modbus protocol. It is a de-facto standard protocol for communication between PLCs and is used extensively. Cyber-attacks targeting it would be unstoppable without some form of security on top of it, and I hope to present that through this project, both through the research I conduct, and the final tool I develop.

1.7 Summary

This project will be attempting to demonstrate the vulnerabilities present in the Modbus TCP protocol. Several aims and objectives have been outlined along with the potential challenges that could hinder progress. The main motivation behind taking on a project like this has been discussed also.

2 Background and Related Work

2.1 Overview

In this chapter, we will look deeper into the underlying problems with the Modbus protocol, looking closely at its use in Industrial Control Systems (ICS), the packet structure, existing attacks and more. From this research, we will identify a research gap that this project will be used to aid.

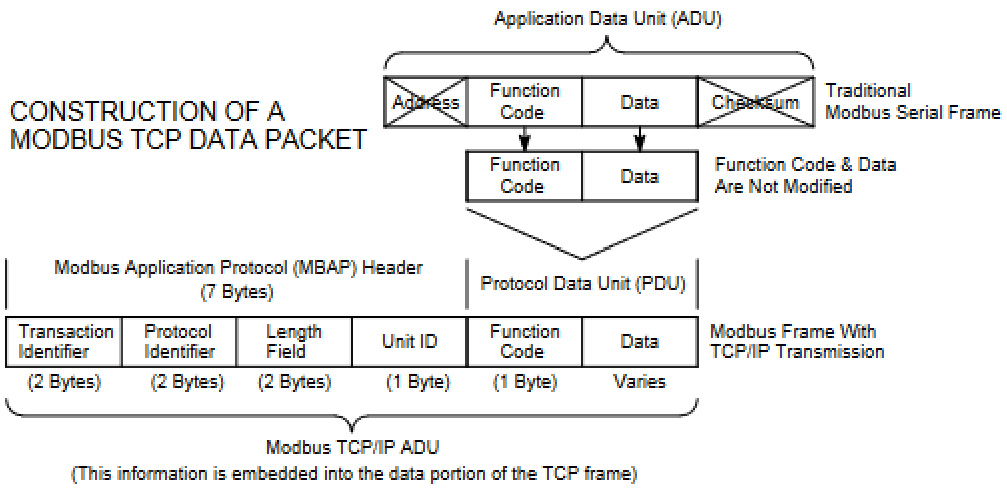
2.2 Industrial Control Systems and SCADA

Supervisory Control and Data Acquisition (SCADA) systems are commonly used within ICS', typically where humans cannot manage them alone as they are too complex [5]. In recent times, as newer systems have been designed, security concerns have been addressed which protects the Modbus protocol.

However, the majority of ICS' were designed 30 years ago, where the main goal was simply efficiency. Security issues are a lot more prevalent in these older systems. Along with this, it is very expensive to replace the existing systems. This means that legacy systems are still vulnerable, hence the need to demonstrate vulnerabilities to raise awareness with stakeholders.

2.3 The Modbus TCP Packet

Modbus TCP is the same protocol as Modbus; however, it has been applied to a TCP interface that works using Ethernet connections.



Application Data Unit (ADU)
CONSTRUCTION OF A MODBUS TCP DATA PACKET
Address
Function Code
Data
Checksum
Traditional Modbus Serial Frame
Function Code
Data
Function Code & Data Are Not Modified
Modbus Application Protocol (MBAP) Header (7 Bytes)
Protocol Data Unit (PDU)
Transaction Identifier
Protocol Identifier
Length Field
Unit ID
Function Code
Data
Modbus Frame With TCP/IP Transmission
(2 Bytes)
(2 Bytes)
(2 Bytes)
(1 Byte)
(1 Byte)
Varies
Modbus TCP/IP ADU
(This information is embedded into the data portion of the TCP frame)

Figure 1 - Modbus TCP – a TCP frame which has a Modbus data frame embedded into it. Source [1]

Because the data is stored within a TCP frame, data integrity is guaranteed because of the checksum used within a TCP packet, meaning that the standard Modbus checksum is not used in transmission. All of the data being transmitted within the Modbus data frame is stored in the data section of the TCP frame.

2.3.1 Vulnerabilities with the Modbus TCP Packet

Like traditional Modbus, there is no data security outside of what TCP provides, which is very little. This means that all of vulnerabilities that exist for each section of the original Modbus packet will exist for the TCP variant, along with vulnerabilities that already existed in TCP. A key weakness with TCP packets lies with the flags that can be set in packets, along with the creation of new connections.

For instance, new TCP connections will send a SYN segment to the recipient to initiate the communication, which forces the recipient to respond with either an SYN-ACK segment, (to acknowledge the connection), or an RST segment if the port is closed. If a device were to have all open ports flooded with TCP SYN segments, and the attacking device were to ignore every SYN-ACK received in response, the ports affected would be stuck in a buffer waiting for the final ACK segment. This can be used to effectively close machines from receiving new connections. [30]

If an attacker is able to get a shell open on a device that has TCP connections, the connection can be forced to close by sending TCP packets with the FIN flag, which will indicate that the sender has finished transmitting data, or with the RST flag, which indicates that the device was not expecting the packets it received. [31]

Within the Modbus section of the packet, the data portion is transmitted in plaintext. This means that any network sniffing software, like Wireshark, can be used maliciously to see what typical communication between master and slave devices in a particular setup would look like. Along with this, Modbus communications do not have any authorization whatsoever, meaning connections can be initiated by any device at any time so long as it has a network connection. [32]

Because of this, the Modbus data itself can be used to hold a malicious payload, using the TCP packet as its method to be planted within a target device. The Modbus command carried on the packet could be used to modify coils and registers within the target slave device, the consequences of which vary wildly depending on what the PLCs are controlling, be it a power plant or a turbine.

2.4 Existing attacks on the Modbus TCP Protocol

As a part of this project, background research on existing attacks and solutions for attacks on the Modbus TCP Protocol has been conducted. From this research, three main areas of interest appear to be repeated in what attacks target and what solutions look to prevent.

These are the following:

- Loss of Confidentiality

- Loss of Control
- Loss of Awareness

Within the research, three papers stood out for their approach to different attacks and solutions;

According to the paper Attack taxonomies for the Modbus protocols [7], some of the possible attacks on Modbus systems include the periodical disruption of devices attached to the network, along with a complete loss of control if the Modbus master is spoofed. Primary targets would typically be the master, field devices and the communication routes for the protocol. Within this, it is easy to intercept, modify, fabricate and interrupt the protocol in action.

The experiments were run on the premise of what could be accomplished by having a Modbus Sniffer and a packet injector available – essentially allowing the attacker to have the full ability to modify, create their own or completely block messages. The full collection of attacks found are collated in the Problem Space table below.

The paper Assessment of Hidden Channel Attacks: Targeting Modbus/TCP [8] shows that methods of attack that take advantage of the Modbus protocol can use hidden channels as a means to embed any extra data that would be necessary to facilitate an attack.

Hidden channels have to be plausible enough to be of any use for an attacker in the real world, that is to say, it meets two conditions. The first of these conditions is protocol compliance, where the modification of a packet does not break the protocol in any noticeable way that the recipient would be able to identify, or would cause the recipient to be unable to process the packet. The second condition, warden compliance, can be broke down into three different levels, which are determined by different probabilities. The first is that the message is hidden in such a way that a potential warden cannot suspect there is a hidden message. The second is that a potential warden is suspicious of a hidden message but has no ability to access it, and the third is a potential warden is aware of a hidden message, can access it, but cannot read it.

The experiments within this assessment were carried out using a pattern based taxonomy for hiding network information [14] which allowed the authors to identify hidden channels within Modbus TCP, and how easily they could be exploited. The findings show that the majority of the patterns examined could be applied to Modbus TCP to embed information that could be retrieved at a later date, whilst maintaining secrecy from the intended target. In most cases, the patterns would be protocol compliant, however in fewer cases they would be warden compliant also. This suggests that attacks that use hidden channels within Modbus TCP are possible, however if the system being attacked is being monitored by a human rather than an automated program, chances are it may be detected.

From the first paper, it became clear that flooding attacks were extremely dangerous to systems that used Modbus, as there was nothing in place to stop them being overwhelmed by commands. The paper Practical modbus flooding attack and detection [9] shows how the Modbus protocol is vulnerable to flooding attacks, which it states are attacks involving the injection of commands that interrupt the normal running of the system the Modbus protocol is on.

This paper ran their experiments on the premise that flooding attacks were still an open area in vulnerability assessment on the Modbus protocol. Through use of a Java application, the target PLC simulation was quickly overran by allowing the genuine commands in, alongside many more fake generated commands that were in place to slow the system down to the point where only a handful of genuine commands could be executed in-between all of the illegitimate ones.

Their testing proved this through attacking a simulated water-pump that was controlled by PLCs – in all cases the flooding attack was more than able to stop the pump in action through constant state 0 commands. This shows that, whilst there were methods to prevent the attacks, which are listed below in the Solutions table, flooding attacks that are unprevented are very powerful in their offensive capabilities.

Below are summary tables reflecting the research completed as a part of this project:

2.4.1 Problem Space

Table 1 – Problem Space within Modbus TCP, including existing attacks

Existing Attack	Attack Target & Description
Broadcast Message Spoofing [7]	Targets slave devices by sending fake broadcast messages to them. Very difficult to detect as slaves do not respond to broadcast messages. Two variations, one which modifies the targeted device and one which interrupts its operation.
Baseline Response Replay [7]	Targets a master device by recording genuine communications, and then sending the recorded messages back to the master. This can be used to create fake slave devices.
Direct Slave Control [7]	Targets a master device by locking it out of the network, then assuming

	control in its place. It is one of the most threatening attacks to the Modbus protocol, as posing as a Master device in a network gives the attacker full control.
Modbus Network Scanning [7]	Communicates with all possible addresses on the network, giving the attacker information about the slave devices connected.
Passive Reconnaissance [7]	Monitor the communications across a network between the Master and Slave devices. This is usually used in tangent with other attacks, such as Baseline Response Replay.
Response Delay [7]	Delays communications from slave devices, so the master device only ever receives out of date information. This is another dangerous attack, especially towards critical ICS'
Rogue Interloper [7]	Works as a 'Man in the Middle' attack, attaching a separate computer to the network through an unprotected communication link, such as an ethernet port. It would be able to intercept and perform any action to Modbus messages, for example, it could simply monitor messages like in the passive reconnaissance attack, or it could pose as a master device, like in direct slave control.
Irregular TCP Framing [7]	This attack targets either the master or slave device by either creating and then sending or modifying existing messages to create improperly framed messages, which can cause the target device to close the connection.
TCP Fin Flood [7]	This attack targets either the master or slave device by transmitting a fake TCP packet that has the FIN flag set. A FIN flag indicated the end of a transmission, and closes the TCP connection, meaning it can be used following a legitimate Modbus message to kill the connection.
TCP Pool Exhaustion [7]	This attack targets either the master or slave device by exhausting both the priority and non priority connection

	pools through opening massive amounts of TCP connections. To further this attack, if network activity is preserved, it could be used to execute a denial of service attack.
TCP RST Flood [7]	This attack targets either the master or slave device by transmitting an illegitimate TCP packet with the RST flag set. A RST flag is used to terminate a connection if the sender believes the connection should not exist, meaning it can be used following a legitimate Modbus message to kill the connection.
Hidden Channels [8]	Rather than being a direct attack, Hidden Channels can be used to deliver information to a target covertly, potentially delivering a malicious payload alongside a legitimate packet or transmission.

2.4.2 Existing Tools

Table 2 – Existing tools on the market that protect devices using Modbus TCP

Tools	What does it solve/protect?	Cost
Tofino Security Appliance [12]	Acts as a physical firewall in the network, with a list of predefined rules on how the PLCs in the network can communicate.	~£900
Using a VPN [23]	Prevent access to the Modbus device from the internet by adding extra layers of encryption and authentication	Free, up to £100p/a~ [24]
Modbus Transfer Service (MBTS) [25]	Securely transfers existing data from the registers to a data diode outside of the existing network alongside protecting the existing networks from cyber attacks	Varies dependant on existing network.

SolarWinds NetFlow Analyzer [26]	Performs packet inspection on a large, deep scale, allowing any fraudulent packets to be detected with ease.	£812

2.4.3 Solutions (outside the market)

Table 3 – Solutions that are freely available to use

Solution	What does it solve/protect?
Use the Modbus/TCP Security Protocol [10] instead of the original Modbus TCP.	The Modbus/TCP Secuity protocol integrates Transport Layer Security (TLS) [11], which is a protocol built to provide security to communications between applications on a network. It uses encryption for privacy, meaning it's a lot more difficult to intercept any messages. In the case of Modbus, the Modbus master and slave would be authenticated before communication starts.
Use OPC Unified Architecture [13]	OPC Unified Architecture has built-in security that prevents an attack from the internet. However, it is still vulnerable to attacks that are from within the local network.
Anomaly-based change detection algorithm [9]	This is an intrusion detection technique that detects attacks based on changes to the system by any unauthorised actions. This is done through using a technique called Exponentially Weighted Moving Average (EWMA), which is a robust algorithm used for detecting high intensity attacks, like flooding.
Signature-based detection [9]	This is an intrusion detection technique that captures network traffic, decodes it and generates alerts based on detection rules that have been predefined. Snort is an example of this. [17]

2.5 Research Gap Chosen

The research gap chosen is loss of control, as from the sources I have researched it is clear that whilst attacks exist, loss of control is the least explored research area regarding cyber-attacks towards the Modbus TCP protocol, along with arguably being the most damaging to those it targets.

2.6 Project Constraints

As I am working with a Python simulation, I am limited by what I can achieve with code written in Python. Therefore, some attacks may not be possible to recreate/create myself. Along with this, a Python simulation presents difficulties in modelling a real-world scenario. The majority of industry PLCs that use the Modbus protocol will use the version designed for serial connections, whereas the implementation I am using is designed for ethernet connections. However, as industry catches up with more modern technology, my project will then become a lot more prevalent.

2.7 Methods and Tools to attack the research gap

Targeting control means removing the control that the master device has over the slave device. However, like all cyber-attacks, being covert as the attack executes is highly important, therefore the best attack to carry out to exploit this will be a man in the middle attack.

From this, I can effectively cause a loss of control, along with a bonus of loss of confidentiality, as I could change messages as they are sent between master and slave.

2.8 Related Work

In this section, we will look at the related work I will need to carry out to complete a Man in the Middle attack, and look at what preventative measures exist, along with how these can be circumvented in my attack. In some cases, due to the lack of security in the Modbus protocol, it may be the case that preventative measures can be overlooked.

2.8.1 Man in the Middle Attack

A man in the middle attack looks to complete a malicious task completely covertly, that is to say, anything monitoring the target would be unable to recognise that a malicious event is happening.

An example of this happening in the real world is in 2017, the credit score giant Equifax was forced to remove its apps from both the Google Play store and the Apple App store after the app was revealed to not use HTTPS (the secure variant of the HTTP protocol) consistently. This meant that attacker could implement a MITM attack to acquire account data as user's logged in to the app, as without HTTPS, the data transmission was not secure.

In any scenario where two devices are communicating, if the transmission is not secure in any instance it can be targeted by a MITM attack.

In the case of a MITM attack against a Modbus TCP installation, the attacker would have to place themselves between a slave and master, intercepting communications between the two. The master device would be deceived into thinking that the slave was not in danger, leaving the attacker free to modify the slave device.

2.8.2 Preventing a MITM Attack

One of the biggest problems when faced with a MITM attack is detecting it, as they are designed to be as covert as possible until it is too late to do anything to stop it. [33]

Since it is so difficult to stop attacks whilst they are happening, the best approach to take is to stop the attacks before they happen. [34] Good practices include having strong encryption on wireless access points to prevent unwanted users gaining access to a network, as if an attacker has gained access to a network, they can unleash a man in the middle attack and little can stop them.

On top of using encryption with access points, it is important to make sure a network's router's credentials are secure. Man in the middle attacks can be launched easily from within a router, as routers contain a network's primary routing table which is used to update a computer's ARP table, which controls network transmission sources and destination.

A big issue with a man in the middle attack occurring is validating that the devices communicating on the network are real and haven't been spoofed. A method of doing this is through public key-based authentication, such as RSA, which can be used to ensure that even if a device has been spoofed in an ARP table or otherwise, the user's device can still verify whether the destination device is legitimate.

2.9 Summary

This chapter has looked at the background and related work surrounding modelling cyber-attacks on the Modbus TCP protocol. Many vulnerabilities exist which there are a multitude of attacks that target them, however, security measures and detection techniques do exist which will make combatting attacks easier.

Man in the middle attacks against the Modbus protocol are rare. Since these kind of attacks are interesting, covert and powerful, this project will look at creating one.

3 Methodology

3.1 Overview

This chapter will look into the different methodologies used within this project, including the research methodology, attack method and how the final results and evaluation will be carried out.

3.2 Research Methodology

For this project, the research that has been carried out is qualitative rather than quantitative. [36]

This is because qualitative research does not use measurable data, as quantitative research would. Qualitative research is useful here as it can be used to aid concepts which I may not have fully understood before taking on this assignment, such as the problem space around the Modbus TCP protocol. Where the research I am carrying out has no need to show any trends or connections, I do not need qualitative research. Qualitative research is also very flexible, meaning my project scope can change as I gather more research, and I can also return to the research stage at any point to collect more information if it is needed.

As shown in section 3.3, I hope to produce results that can be used in qualitative research also, as again I believe in this area it is more important to understand why these attacks occur rather than the trends they produce.

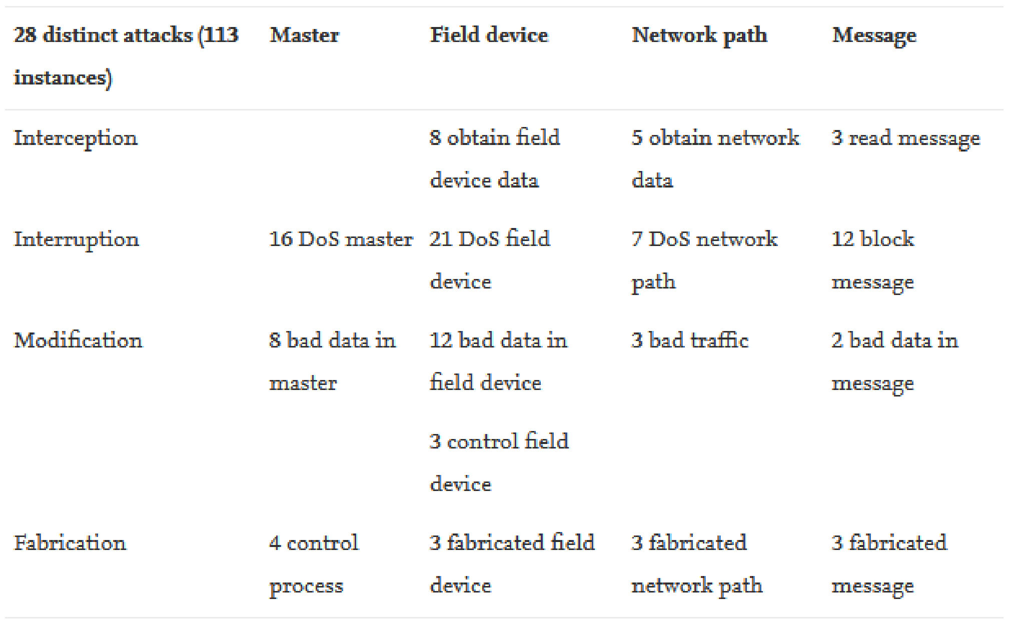
3.3 Attack Method

The attack will be made under the assumption that an attacker has already gained access to a network and gained any and all necessary levels of access to deploy an attack. This means that we won't have to consider any preliminary examination or attack to gain access, and we can use all of the tools available on Linux and Windows.

Once the attack is built, I will look to produce some statistics that show how quickly my attack achieved its intended goals. These will include time for the attack to fully execute, time taken from the main python file being launched and damage caused, if applicable.

The papers discussed above have used similar methods when conducting their experiments. All would use tables to present their data, and all would go through their experiments in a linear fashion, similar to what we would want to produce. From this, in order to show how the attack created with this paper performs, we will need to compare it to other existing attacks on similar characteristics, for example what impact the attacks have and what in particular is impacted.

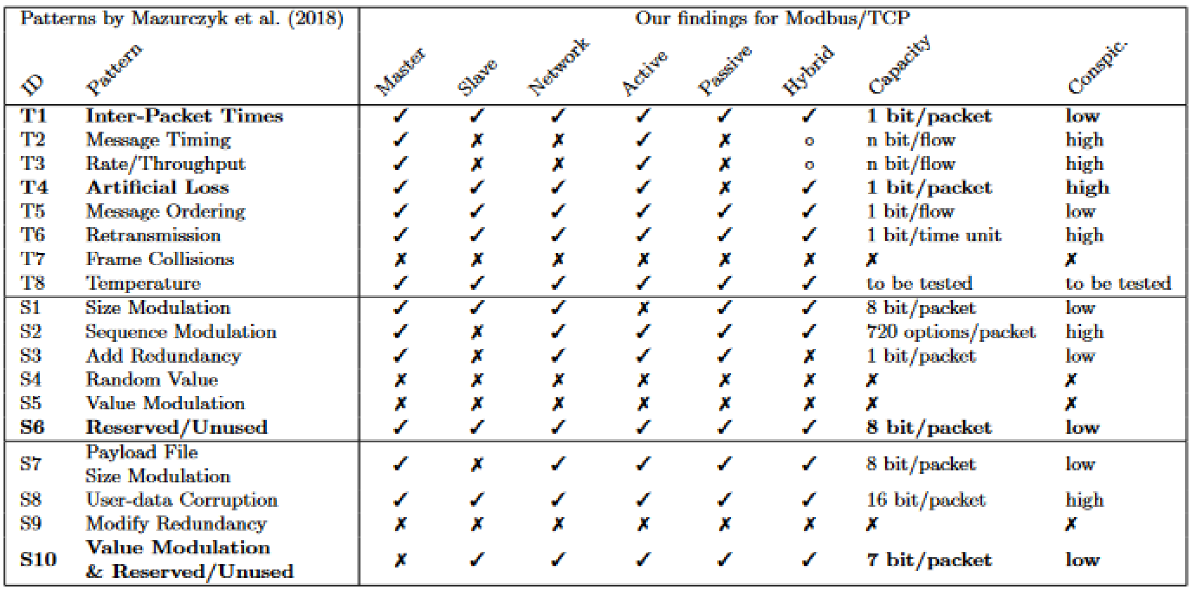
3.4 Evaluation and Results



28 distinct attacks (113 instances)	Master	Field device	Network path	Message
Interception		8 obtain field device data	5 obtain network data	3 read message
Interruption	16 DoS master	21 DoS field device	7 DoS network path	12 block message
Modification	8 bad data in master	12 bad data in field device	3 bad traffic	2 bad data in message
		3 control field device		
Fabrication	4 control process	3 fabricated field device	3 fabricated network path	3 fabricated message

Figure 2 – Table used in [7] to present the impact of the different attack taxonomies tested

Shown above is an example of what results that could be produced after the attack is implemented. This way, the attack created within this project could be compared to what previously researched attacks do, or if time allows, further attacks could be built and compared to the main one created in the project.



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

Figure 3 – Table from [8] that shows what each Hidden Channel pattern can achieve across a network that uses Modbus TCP

Another method of displaying the results is shown above, where the author has compared each pattern used in their experiment by showing what it can be used against, how it functions and the payload capacity of the attack.

Based on the above, once the attack is designed, created and tested, it will be evaluated against the different solutions by explaining what would happen in the hypothetical situation where the attack is deployed against a Modbus installation with the protection in place. This information will be displayed in a table similar to Figure 2 and 3. If time permits, we can then replicate the table for attacks that are already well documented and compare them to my own, like those found in the problem space shown in section 2.4.1.

3.5 Summary

The research methodology being used for this project is qualitative, as it will be more useful to use existing research to aid understanding around the vulnerabilities of the Modbus protocol. The attack created will be run under the assumption an attacker has full access to a network, and has carried out all the pre-required surveillance. Similar to previous studies into the protocol, the attack's performance against the existing defences will be recorded in a table and evaluated.

4 Design

4.1 Overview

This chapter will look into designing a man in the middle attack against the Modbus protocol. Requirements will be covered, along with the user interface and the route the user will through the system. This will include key algorithms through pseudocode.

4.2 Man in the Middle Attack:

To perform this attack, I have broken down my approach to it into several functional and non-functional requirements:

4.2.1 Functional Requirements

- Prevent messages being received by the Slave device.
- Return confirmation messages based off of what commands were originally sent. This could be achieved through generating messages, or using a Pymodbus server on the attacking device
- Send user-written commands to the Slave device.

4.2.2 Non-Functional Requirements

- The attack should be simple enough to deploy once it is created and should not take an unreasonable time to execute

4.3 Automated Tool:

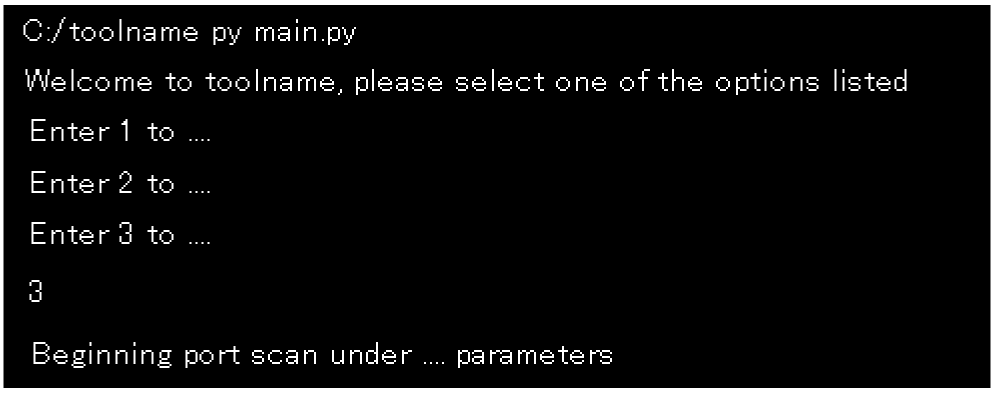
4.3.1 Functional Requirements

- Give the user the option to find the server across all ports, common Modbus ports (502, 5020, 50200) or a specific port.
- Once a server has been found, allow the user to choose one of the mounted attacks. (The original version of the tool will only have the attack created within this project available.)
- Deploy the attack, printing statistics such as time to execution post-deployment
- Return output from the attack target to the user if any output exists.

4.3.2 Non-Functional Requirements

- The tool should be simple to use, offer help where necessary and should have a clear route to attack deployment.

4.4 User Interface



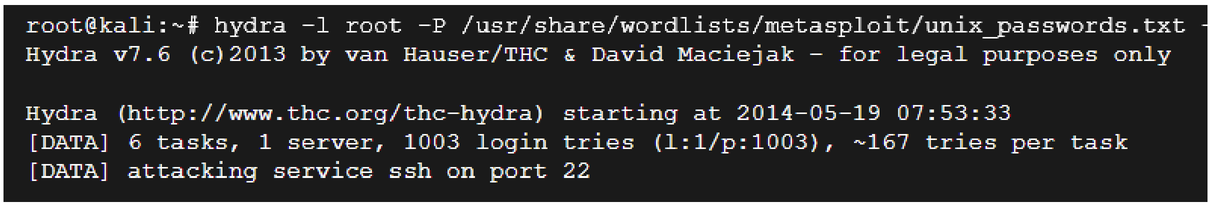
```
C:/toolname py main.py
Welcome to toolname, please select one of the options listed
 Enter 1 to ....
 Enter 2 to ....
 Enter 3 to ....
 3
 Beginning port scan under .... parameters
```

Figure 4 - Example UI with example input and output

Because the final tool needs to be simple to use, I have decided it would be best to use a command line interface, as that way there will not be complicated graphics or anything that could possibly overwhelm the user. Instead, the user will be confined to the inputs the program would need to be able to function properly, and at each stage every option available to the user would be given to them, making it difficult to unintentionally break the software.

Alongside this, it is very simple to read user input from the command line in python, making it the ideal choice where there isn't much input from the user outside of choosing options. Where user interaction is concerned also, since this tool is being designed for researchers, it is safe to presume that the majority of users will be computer-literate, and will have no issue using the command line.

Furthermore, most attack tools use the command line as their interface of choice. This can be seen with Hydra [16], which is present in Kali Linux, which is one of the more popular offensive tools used in vulnerability research.



```
root@kali:~# hydra -l root -P /usr/share/wordlists/metasploit/unix_passwords.txt -
Hydra v7.6 (c)2013 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2014-05-19 07:53:33
[DATA] 6 tasks, 1 server, 1003 login tries (l:1/p:1003), ~167 tries per task
[DATA] attacking service ssh on port 22
```

Figure 5 - Example of Hydra using a Command Line Interface as its User Interface

Because it appears to be a standard form for offensive tools built for research to use the command line, and the other reasons mentioned above, I believe it to be the best choice for the user interface for my tool.

4.5 Class Diagram for Tool

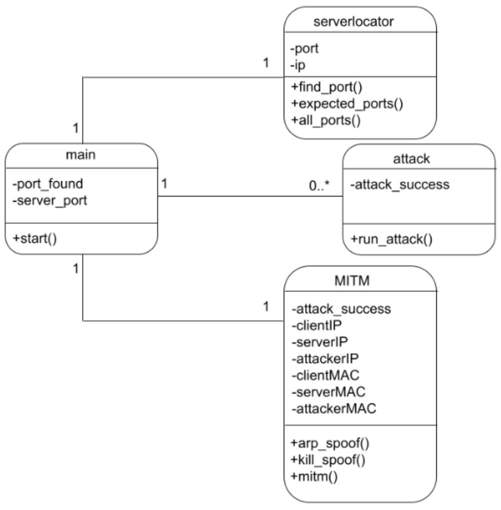
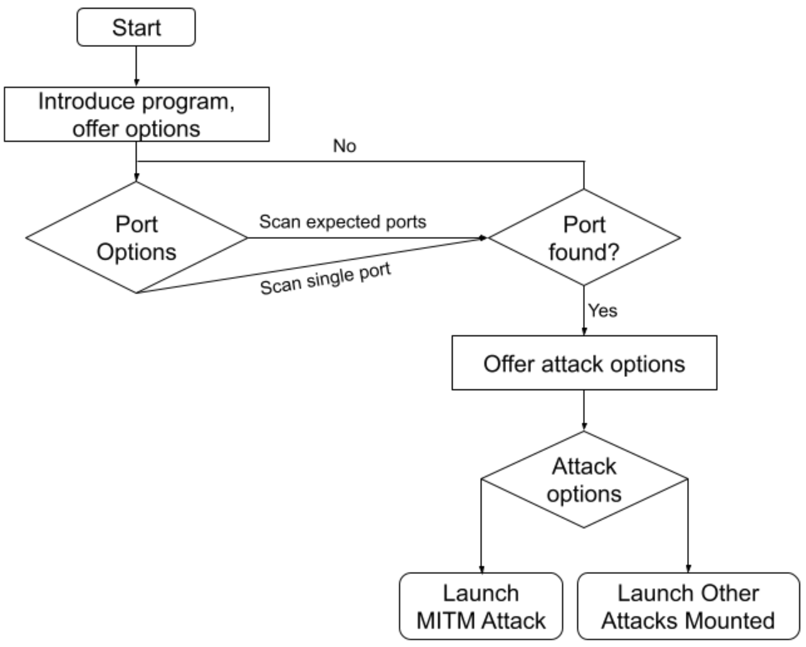


Figure 6 – UML Class Diagram showing how the tool should look

The UML diagram above shows how the tool should look once it has been implemented. There will be a class for finding the server given an IP, there will be the Man in the Middle attack developed as a part of this project and then if any other attacks are added to the tool, they have been included.

4.6 Flow Chart for Tool and MITM Attack



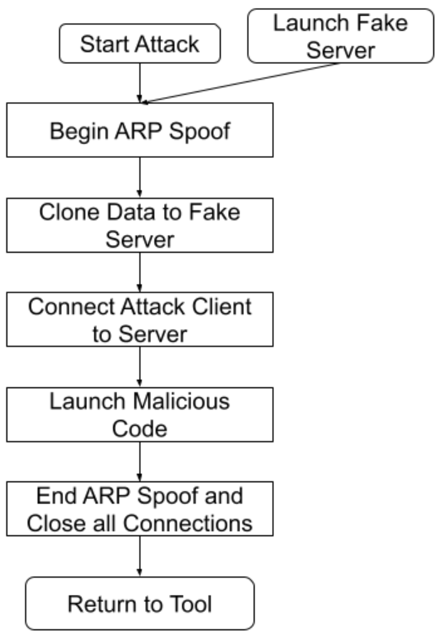

Start Attack
Launch Fake Server
Begin ARP Spoof
Clone Data to Fake Server
Connect Attack Client to Server
Launch Malicious Code
End ARP Spoof and Close all Connections
Return to Tool

Figure 7 – Flow chart showing how a user will move through the tool

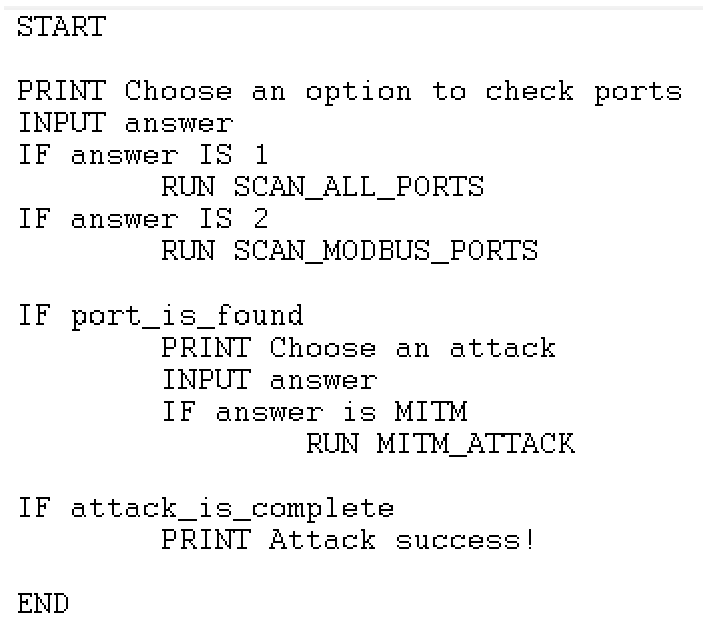
The two flow charts above show the path the user will navigate to perform the MITM attack. Once the tool has been launched, it should introduce itself, and move directly into locating the server. The tool is meant to be used after gaining access to a network, meaning all IP addresses and MAC address needed would have already been found, so the tool will not need the facilities to do this.

When a port is found, it should then list the attacks that are built into it. Currently, the aim is to have attacks hardcoded as options, but in the future it would be ideal to expand into using polymorphism to change a generic attack class into more specific attacks, should there be enough similarities between attacks created in python.

When the MITM attack is launched, an ARP spoof will start, tricking the server and client into believing the attacking device is what it is connected to or receiving connections from. Data from the real server will be cloned into a fake server to make sure the client is unable to recognise an attack is occurring. At this point in the attack, malicious commands can be injected into the real server, following which the connections can be closed and the ARP spoof can be ended.

4.7 Pseudocode of Tool and MITM Attack

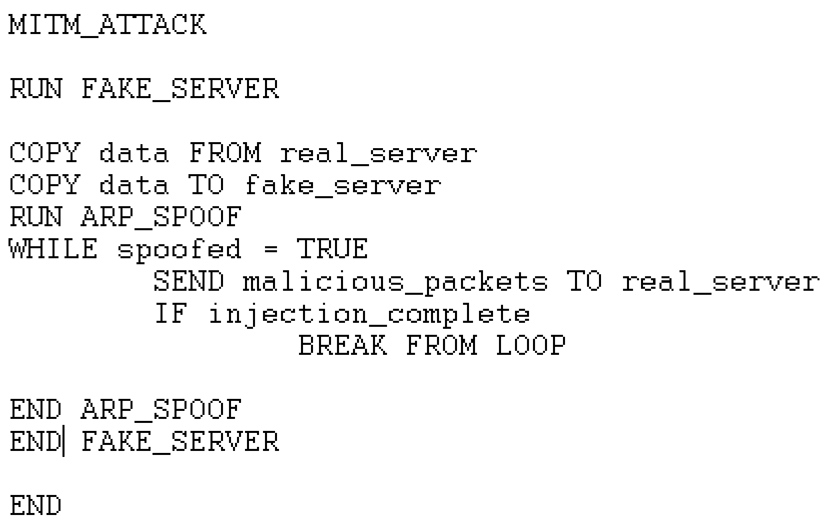
In order to accurately show how the attack and tool will operate, pseudocode has been produced to demonstrate the functionality of both.



```
START

PRINT Choose an option to check ports
INPUT answer
IF answer IS 1
        RUN SCAN_ALL_PORTS
IF answer IS 2
        RUN SCAN_MODBUS_PORTS

IF port_is_found
        PRINT Choose an attack
        INPUT answer
        IF answer is MITM
                RUN MITM_ATTACK

IF attack_is_complete
        PRINT Attack success!

END
```

Figure 8 – Pseudocode of Main Tool

The main tool will have two stages – finding the Modbus Server on a given IP address, and then launching an attack against it. There will be several options available for the user for finding the server, and eventually the tool could be expanded to also host several other attacks. Once the attack has been chosen, it will be run, at which point the program will jump to the attack. When the attack is complete, the user will be informed and the program will end.



```
MITM_ATTACK

RUN FAKE_SERVER

COPY data FROM real_server
COPY data TO fake_server
RUN ARP_SPOOF
WHILE spoofed = TRUE
        SEND malicious_packets TO real_server
        IF injection_complete
                BREAK FROM LOOP

END ARP_SPOOF
END FAKE_SERVER

END
```

Figure 9 – Pseudocode of Man in the Middle attack

Shown in the figure above is how the man in the middle attack will operate. A big part of every man in the middle attack is intercepting data during transmission, which will be achieved here through an ARP spoof. Before the spoof happens, a copy of the real-time data will be placed in a fake server on the attacking device, which the real client will be connected to during the attack. Once the spoof happens, the malicious commands can be sent to the real server. Once the packets are fully sent, the loop will break, the spoof will end and the fake server will be closed. At this point, the program will return to the main tool.

4.8 Required Hardware and Software

For the attack planned as a part of this project, there are two options available to me. Either a virtual implementation of the attack can be used, hosted on one device, or a physical implementation of the attack can be used, hosted across three.

To help visualise the attack whilst I make it, I have opted to use three different physical devices, one to host the Pymodbus client, one to host the Pymodbus server, and one to run the attack.

Both the client and server device will be running Windows 10 and have python installed in order to host the Pymodbus installation, and both will have Wireshark installed to monitor the packets being received.

The attacking device will be running Ubuntu 20.0.4, with python installed to launch the attack and Wireshark will be installed to monitor the packets leaving the device.

4.9 Final System Model

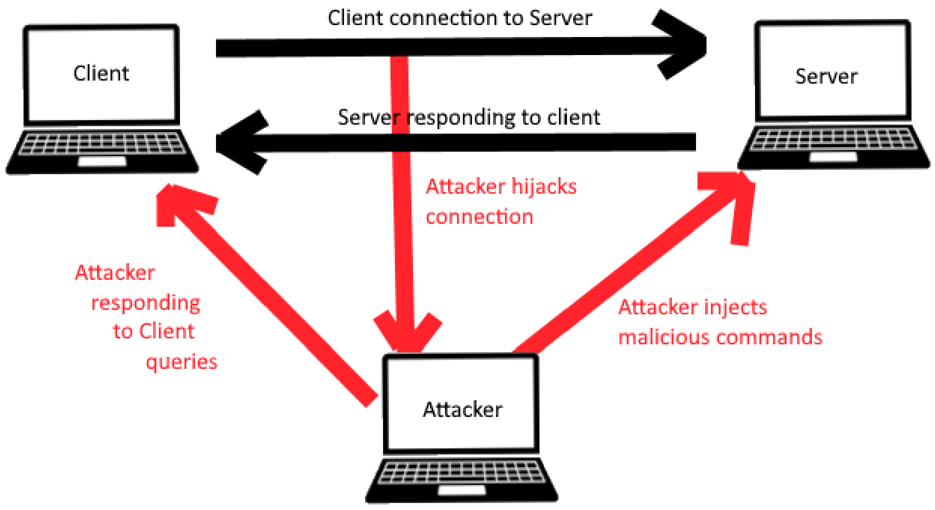

Client connection to Server
Client
Server
Server responding to client
Attacker hijacks connection
Attacker responding to Client queries
Attacker injects malicious commands
Attacker

Figure 10 – System model showing how the Man in the Middle attack will operate

Figure 10 above shows how the man in the middle attack will operate. Originally, the client and server will be communicating with each other, with the client issuing read requests to the server's registers and coils.

Once the attack starts, the attacker will use ARP spoofing to hijack the connection the client has to the server and will feed it fake data that is has cloned from the server. The server will no longer be responding to the client, which will be unable to tell the connection has been replaced with a fake server. At this point, the attacker would be free to inject whatever malicious commands they please.

4.10 Summary

The final attack has been designed with the user in mind, being encased within an automated tool that does most of the work for the user. All the user will need to do is enter numbers to pick the options for checking ports and which attack to load – although within the boundaries of this project the only attack available will be the man in the middle.

5 Implementation – Attack & Tool

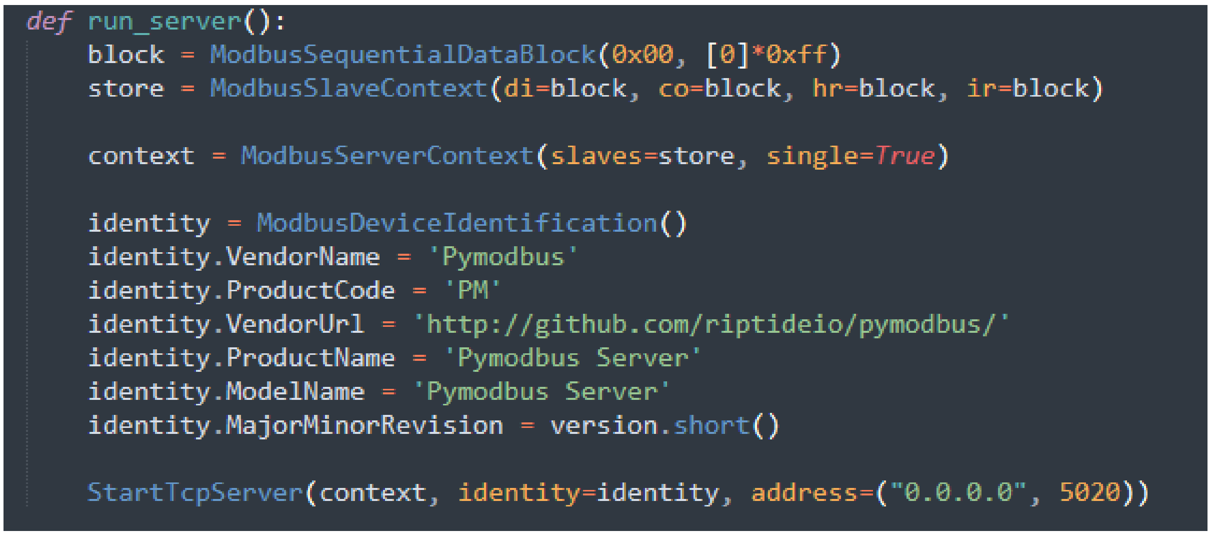
5.1 Overview

This chapter covers the implementation of the man in the middle attack and the automated tool that will be used to deploy it.

5.2 Client / Server Combination

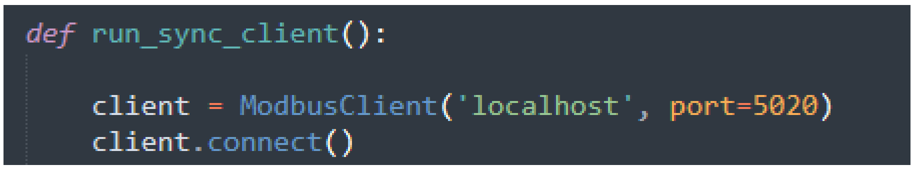
To begin work on an attack, I first had to implement a simulation of Modbus TCP. For this, as mentioned previously, I will be using pymodbus, which is a full implementation of the Modbus TCP protocol in python.

Pymodbus has several different implementations of servers and clients, and I have chosen the synchronous version:



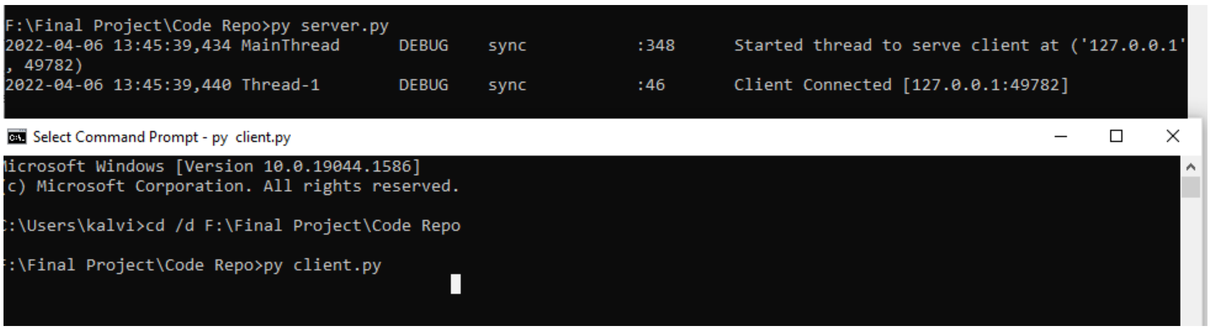
```python
def run_server():
    block = ModbusSequentialDataBlock(0x00, [0]*0xff)
    store = ModbusSlaveContext(di=block, co=block, hr=block, ir=block)

    context = ModbusServerContext(slaves=store, single=True)

    identity = ModbusDeviceIdentification()
    identity.VendorName = 'Pymodbus'
    identity.ProductCode = 'PM'
    identity.VendorUrl = 'http://github.com/riptideio/pymodbus/'
    identity.ProductName = 'Pymodbus Server'
    identity.ModelName = 'Pymodbus Server'
    identity.MajorMinorRevision = version.short()

    StartTcpServer(context, identity=identity, address=("0.0.0.0", 5020))
```

Figure 11 - Code for creating a synchronous pymodbus server hosted on the localhost port 5020



```python
def run_sync_client():

    client = ModbusClient('localhost', port=5020)
    client.connect()
```

Figure 12 - Code for connecting a synchronous client to the synchronous server



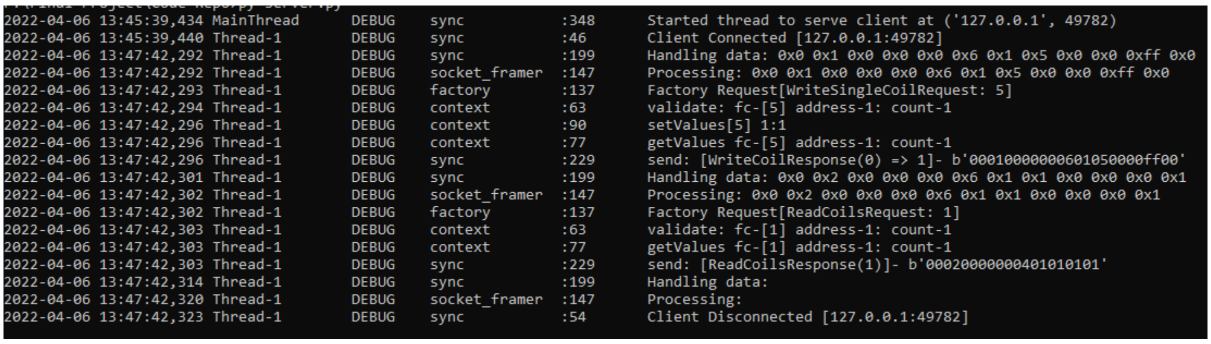
```
F:\Final Project\Code Repo>py server.py
2022-04-06 13:45:39,434 MainThread      DEBUG    sync          :348      Started thread to serve client at ('127.0.0.1'
, 49782)
2022-04-06 13:45:39,440 Thread-1        DEBUG    sync          :46       Client Connected [127.0.0.1:49782]
```

Select Command Prompt - py client.py

```
icrosoft Windows [Version 10.0.19044.1586]
c) Microsoft Corporation. All rights reserved.

:\Users\kalvi>cd /d F:\Final Project\Code Repo

:\Final Project\Code Repo>py client.py
```

Figure 13 - Synchronous server running with the client connected.

In practice, the client can stay connected to the server for as long as possible until it is either out of commands or the user terminates the connection. This can be seen in the figure below, where the data in the server's coils has been modified:



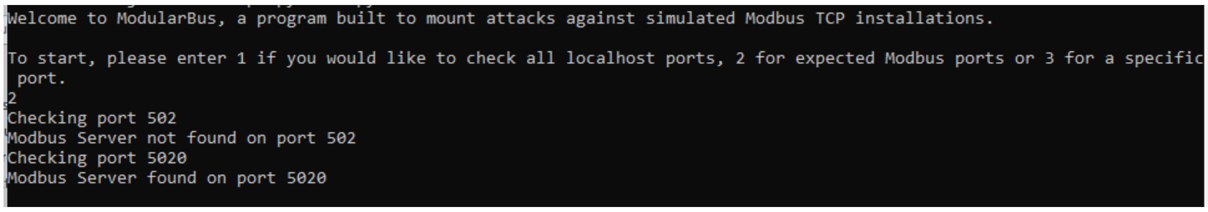
```
2022-04-06 13:45:39,434 MainThread      DEBUG    sync           :348      Started thread to serve client at ('127.0.0.1', 49782)
2022-04-06 13:45:39,440 Thread-1        DEBUG    sync           :46       Client Connected [127.0.0.1:49782]
2022-04-06 13:47:42,292 Thread-1        DEBUG    sync           :199      Handling data: 0x0 0x1 0x0 0x0 0x0 0x6 0x1 0x5 0x0 0x0 0xff 0x0
2022-04-06 13:47:42,292 Thread-1        DEBUG    socket_framer  :147      Processing: 0x0 0x1 0x0 0x0 0x0 0x6 0x1 0x5 0x0 0x0 0xff 0x0
2022-04-06 13:47:42,293 Thread-1        DEBUG    factory        :137      Factory Request[WriteSingleCoilRequest: 5]
2022-04-06 13:47:42,294 Thread-1        DEBUG    context        :63       validate: fc-[5] address-1: count-1
2022-04-06 13:47:42,296 Thread-1        DEBUG    context        :90       setValues[5] 1:1
2022-04-06 13:47:42,296 Thread-1        DEBUG    context        :77       getValues fc-[5] address-1: count-1
2022-04-06 13:47:42,296 Thread-1        DEBUG    sync           :229      send: [WriteCoilResponse(0) => 1]- b'000100000006010500000ff00'
2022-04-06 13:47:42,301 Thread-1        DEBUG    sync           :199      Handling data: 0x0 0x2 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0x0 0x0 0x1
2022-04-06 13:47:42,302 Thread-1        DEBUG    socket_framer  :147      Processing: 0x0 0x2 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0x0 0x0 0x1
2022-04-06 13:47:42,302 Thread-1        DEBUG    factory        :137      Factory Request[ReadCoilsRequest: 1]
2022-04-06 13:47:42,303 Thread-1        DEBUG    context        :63       validate: fc-[1] address-1: count-1
2022-04-06 13:47:42,303 Thread-1        DEBUG    context        :77       getValues fc-[1] address-1: count-1
2022-04-06 13:47:42,303 Thread-1        DEBUG    sync           :229      send: [ReadCoilsResponse(1)]- b'00020000000401010101'
2022-04-06 13:47:42,314 Thread-1        DEBUG    sync           :199      Handling data:
2022-04-06 13:47:42,320 Thread-1        DEBUG    socket_framer  :147      Processing:
2022-04-06 13:47:42,323 Thread-1        DEBUG    sync           :54       Client Disconnected [127.0.0.1:49782]
```

Figure 14 - Example of Client interacting with Server through modifying coils data values

Figure 14 above shows how the client will terminate the connection when it is out of commands to run. This can either be achieved by manually ending the connection using one of the included methods, or by a timeout.

5.3 Automated Tool

This section covers the automated tool produced as part of this project. It operates through running the user through key questions to determine whether or not it needs to check ports for the server, what attack the user would like to run and then finding the server (if it exists within the user's defined parameters) and then deploying the chosen attack.



```
Welcome to ModularBus, a program built to mount attacks against simulated Modbus TCP installations.

To start, please enter 1 if you would like to check all localhost ports, 2 for expected Modbus ports or 3 for a specific
 port.
2
Checking port 502
Modbus Server not found on port 502
Checking port 5020
Modbus Server found on port 5020
```

Figure 15 - Tool running and a port scan across expected Modbus port being run.

5.4 Man in the Middle Attack

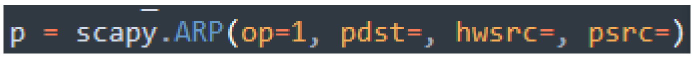
This section covers the attack that I have produced as a part of this project. There are three methods within this file that will carry out the main portion of the attack.

5.4.1 arp_spoof()

In most man in the middle attacks, arp spoofing [19] is used to forge connections between devices that are typically meant to communicate with each other. This is done by modifying the ARP cache on hosts, which store the different links between the IP addresses and MAC addresses of devices on a network. By modifying this table, it is possible for an attacker to effectively pretend to be any device on the network.

In the case of the simulation, I want the attacking device to pretend to be both the server and the client. This means that when the client attempts to communicate with the server, it will be re-rerouted to a fake server hosted on the attacking device. Alongside the fake server, I will also have a client on my device that will take the original client's place.

The arp_spoof method will, once a simple iptables rule is in place, send ARP packets to the server and client to change their ARP cache to point towards the attacking device. This completes the man in the middle attack, and will allow the attacker to deliver whatever payload they like to the server. It will run on a continuous loop until the attack is complete, keeping the spoof running as long as needed.



```
p = scapy.ARP(op=1, pdst=, hwsrc=, psrc=)
```

Figure 16 – Empty scapy packet.

The package scapy [20] has been used here to forge the spoof ARP packets. This is because it is one of the most well documented packages for packet creation and decoding available for free. Alongside this, it was very easy to use and implement into my attack.

A separate method, end_spoof, is used to end the spoof following the attack, removing any footstep we had on the system.

5.4.2 intercept()

This method is used after arp_spoof has been executed for the first time to create a running replica of the data that should be expected from the server.

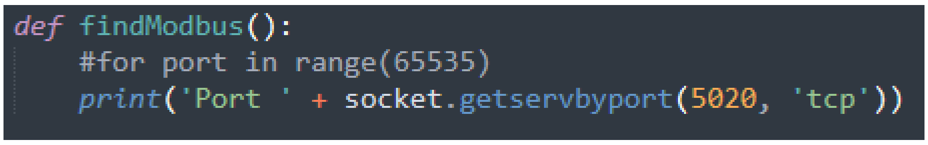
```
coil_data = attack_client.read_coils(address = 0x00, count = 0x05, unit = 0x01).bits
print("Reading Coils: " + str(coil_data))
arp_spoof()
reg_data = attack_client.read_holding_registers(address = 0x00, count = 5, unit = 0x01).registers
print("Reading Registers: " + str(reg_data))
arp_spoof()
intercept_client.write_coils(0, coil_data[:], unit=1)
print("Cloning Coil data to Attack Server")
arp_spoof()
intercept_client.write_registers(0, reg_data[0:1], unit=1)
print("Cloning Register data to Attack Server")
```

Figure 17 – Extract from the intercept() method

The fake server that is created and hosted on the attacker's device will be constantly populated with real-time data from the real server. If somebody were to be monitoring the real server, (during the attack the output from the fake server would be seen, not the real one), there would appear to be nothing wrong as the coils and registers within the fake server would appear to be legitimate.

5.4.3 find_server3()

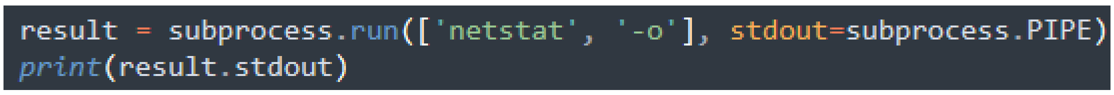
This method is the third iteration of a method used to find a Modbus server on a network device. Because this attack assumes information like IP addresses for the devices involved is known, there is no functionality to check individual IP addresses at present, but in the future, this could be included.



```
def findModbus():
    #for port in range(65535)
    print('Port ' + socket.getservbyport(5020, 'tcp'))
```

Figure 18 – Extract from the first attempt at finding a modbus server

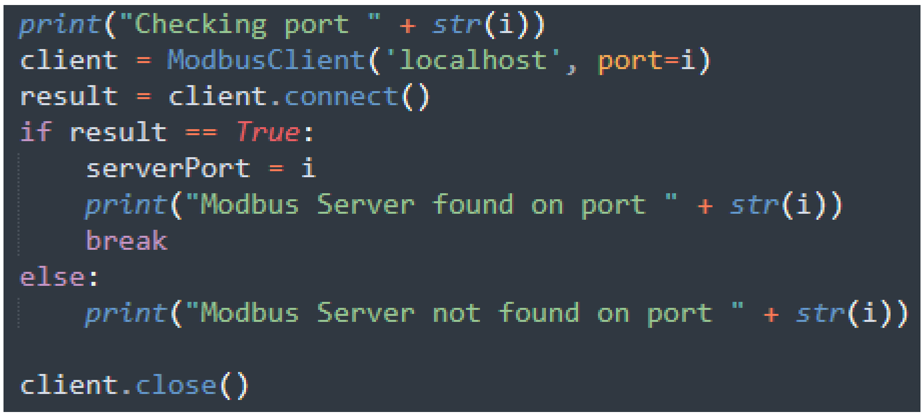
In the first attempt, I wanted to use the socket library to find the service name from the port number. Originally, this would've been done using a loop to check every port, but during testing it was only necessary to test the port the server was hosted on. The issue remained where this method didn't work as the socket function wouldn't want to test a port that had something hosted on it.



```
result = subprocess.run(['netstat', '-o'], stdout=subprocess.PIPE)
print(result.stdout)
```

Figure 19 – Extract from the second attempt at finding a modbus server

The second attempt involved using the subprocess library to run the command necessary to find the server. Again, following the same kind of method as the previous attempt, this would find the process names and list them. The main issue came from the output from the command, where the full result would be returned as one string, which would be too long a complex to pick out the information we need.



```
print("Checking port " + str(i))
client = ModbusClient('localhost', port=i)
result = client.connect()
if result == True:
    serverPort = i
    print("Modbus Server found on port " + str(i))
    break
else:
    print("Modbus Server not found on port " + str(i))

client.close()
```

Figure 20 – Extract from find_server3

For the final method, I decided to use the pymodbus library itself to check ports. Whilst this way was more invasive, it would be guaranteed to work if it found the correct port during its checking. There are three options the user can pick, as shown in Figure 19 above. The user could either check a port of their choice, typical ports a Modbus TCP server could exist on or every port. However, due to the invasiveness and the amount of time it would take, it wouldn't be ideal checking every port.

This would work by generating a client and then attempting to connect with the selected port(s). If the port has a modbus server present on it, a message will be returned to user stating it. For each port that doesn't have a server on it, a message is also returned to the user.

This tool would be used after the user has gained access to a system, so at that point it would most likely be known which port the server was on, meaning that the option to check every port isn't necessary, but it has been included for the sake of covering all bases.

5.5 Unforeseen Problems

During the implementation of this attack, there were several issues encountered which hindered progress and changed the timeline used.

5.5.1 Client/Server

To test anything that I created both during implementation and in the testing phase, I would need to have an implementation of the Modbus protocol available to attack.

Because there is a lack of documentation available around the implementation of pymodbus, I found it difficult to get the client/server pair working. This meant that the project faced delays, as I couldn't begin the creation of an attack without a working simulation of Modbus TCP. I attempted to use a different library to accommodate this issue, however I struggled with that and reverted back to pymodbus, which I eventually got working. However, the delays in getting the client/server working meant the rest of the project had less time available to it.

5.5.2 MITM – iptables

In order to pull off the man in the middle attack, I would need to use ARP spoofing in order to trick the 'real' client with authentic responses to whatever commands are sent across the network for the server. The easiest method of doing this involves the linux command iptables, which would allow me to set rules for transmission between ports across the network – in essence I could reroute all transmissions intended for the port the server is hosted on to the device the attack is being run on.

Through research, there are several solutions that exist on the Windows 10 operating system, for example netsh interface portproxy [18], however though my attempts to use them none of them worked. This has meant that I have had to repurpose old hardware to create a linux workstation in my network for this project, which added a small delay to the attack's creation.

Once the iptables command has been executed (which reroutes data across the network heading to the port found in find_server3) the attack can continue as expected.

5.6 Summary

This chapter has covered all of the code created as a part of this project. It targets the research gap uncovered during the Background and Related work stage, and exploits the lack of security in the Modbus TCP protocol.

6 Results

6.1 Overview

This chapter looks into the real world impact of the man in the middle attack created in this project, examining three different scenarios that if targeted with no preventative measure could have disastrous consequences. Alongside this, there is a full demonstration of the attack in operation against the Pymodbus installation of Modbus TCP.

6.2 Real World Scenarios

In order to understand the potential impact of an attack like the one demonstrated below, it is important to consider the consequences of the attack when it targets both critical infrastructure and non-essential infrastructure. The following section looks at what would happen if the attack as demonstrated above was used in three different scenarios.

6.2.1 Water Treatment Plant

Water treatment plants use different chemicals, like aluminium sulphate (alum) [27], to clump particles in un-treated water together to form large, gelatinous particles. These chemicals are often used in very trace amounts to make sure that the levels present in the water would be non-fatal, and could easily pass through the body without causing any damage.

If the treatment plant used PLCs to monitor the chemical input to the water, and control the amount of chemicals being added to clean it, that would mean the chemical content of the water could be targeted by an attack like the one created in this project.

The attack demonstrated above changes the contents of the holding registers of the targeted device – if the device were to measure the chemicals it adds to the water in micrograms, and the safe amount if 50 micrograms, the attack could change the value stored in the registers to 10000, which would have a dangerous effect to the intestines of any person who drank the water that was affected [28].

6.2.2 Centrifuge to create Reactor Fuel

A centrifuge can be used to create low-enriched uranium which is suitable for nuclear reactors [29]. However, an alternative use of a centrifuge is to produce highly-enriched uranium, which is the kind used in nuclear weapons. If a centrifuge's PLCs were to be changed through an attack like the one above, the uranium created to eventually be used to fuel nuclear reactors could be the kind which is used within weapons.

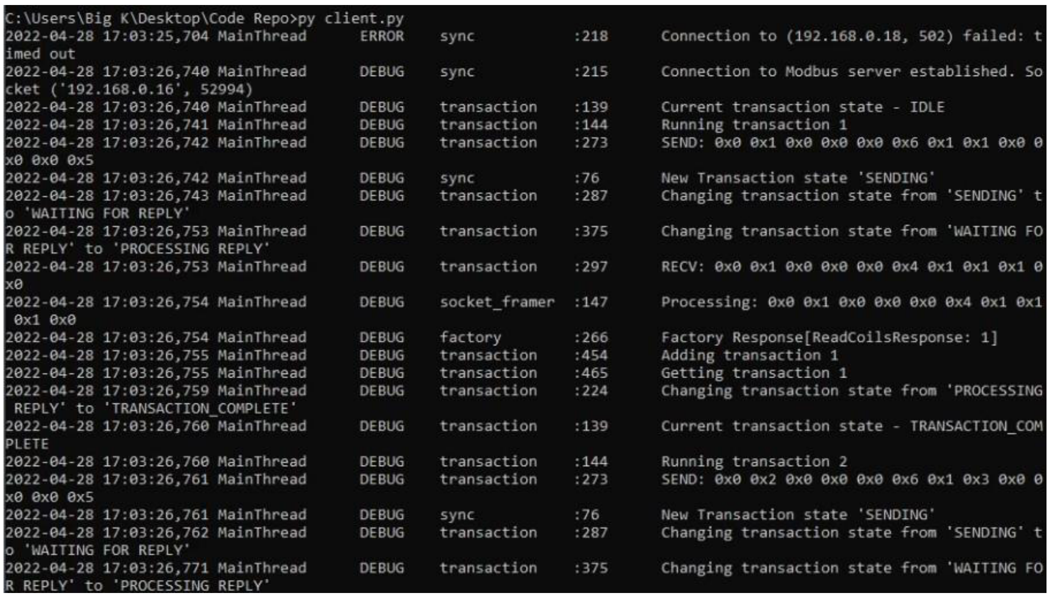
This would mean that when the fuel is added to a reactor, it would become unstable quickly and possibly cause the reactor to melt-down, which would lead to an event similar to Chernobyl occurring, the effects of which would be devastating, especially if it were to happen in a densely populated area.

6.2.3 Manufacturing Machines

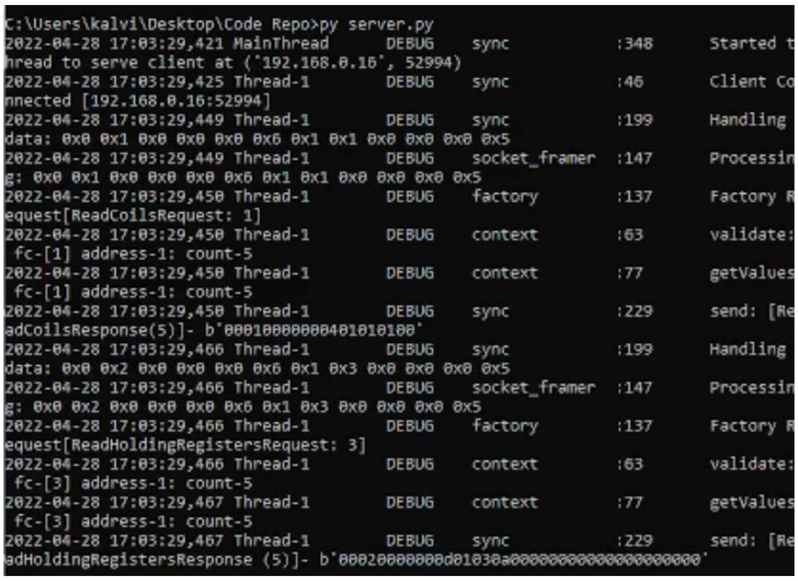
A less lethal target would be factories, or manufacturing centres. The majority of these system would be ICS', meaning they would be one of the traditional targets to be picked when targeting devices that use Modbus. Like above, the attack would again target the holding registers in the slave devices, which in the case of a manufacturing centre would most likely control operating speed. This normally could be safely modified in small amounts to increase or decrease production speed at different times of year, such as Christmas. By modifying this value to beyond safe limits, the machines would most likely break down as they will not work beyond the specified limits.

6.3 Demonstration of MITM Attack

Within this demonstration, it is presumed that the attacker has gained full access to the system, along with knowing all of the information required to pull off the attack, like IP addresses and MAC addresses.



```
C:\Users\Big K\Desktop\Code Repo>py client.py
2022-04-28 17:03:25,704 MainThread      ERROR    sync          :218      Connection to (192.168.0.18, 502) failed: t
imed out
2022-04-28 17:03:26,740 MainThread      DEBUG    sync          :215      Connection to Modbus server established. So
cket ('192.168.0.16', 52994)
2022-04-28 17:03:26,740 MainThread      DEBUG    transaction   :139      Current transaction state - IDLE
2022-04-28 17:03:26,741 MainThread      DEBUG    transaction   :144      Running transaction 1
2022-04-28 17:03:26,742 MainThread      DEBUG    transaction   :273      SEND: 0x0 0x1 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0
x0 0x0 0x5
2022-04-28 17:03:26,742 MainThread      DEBUG    sync          :76       New Transaction state 'SENDING'
2022-04-28 17:03:26,743 MainThread      DEBUG    transaction   :287      Changing transaction state from 'SENDING' t
o 'WAITING FOR REPLY'
2022-04-28 17:03:26,753 MainThread      DEBUG    transaction   :375      Changing transaction state from 'WAITING FO
R REPLY' to 'PROCESSING REPLY'
2022-04-28 17:03:26,753 MainThread      DEBUG    transaction   :297      RECV: 0x0 0x1 0x0 0x0 0x0 0x4 0x1 0x1 0x1 0
x0
2022-04-28 17:03:26,754 MainThread      DEBUG    socket_framer :147      Processing: 0x0 0x1 0x0 0x0 0x0 0x4 0x1 0x1
 0x1 0x0
2022-04-28 17:03:26,754 MainThread      DEBUG    factory       :266      Factory Response[ReadCoilsResponse: 1]
2022-04-28 17:03:26,755 MainThread      DEBUG    transaction   :454      Adding transaction 1
2022-04-28 17:03:26,755 MainThread      DEBUG    transaction   :465      Getting transaction 1
2022-04-28 17:03:26,759 MainThread      DEBUG    transaction   :224      Changing transaction state from 'PROCESSING
 REPLY' to 'TRANSACTION_COMPLETE'
2022-04-28 17:03:26,760 MainThread      DEBUG    transaction   :139      Current transaction state - TRANSACTION_COM
PLETE
2022-04-28 17:03:26,760 MainThread      DEBUG    transaction   :144      Running transaction 2
2022-04-28 17:03:26,761 MainThread      DEBUG    transaction   :273      SEND: 0x0 0x2 0x0 0x0 0x0 0x6 0x1 0x3 0x0 0
x0 0x0 0x5
2022-04-28 17:03:26,761 MainThread      DEBUG    sync          :76       New Transaction state 'SENDING'
2022-04-28 17:03:26,762 MainThread      DEBUG    transaction   :287      Changing transaction state from 'SENDING' t
o 'WAITING FOR REPLY'
2022-04-28 17:03:26,771 MainThread      DEBUG    transaction   :375      Changing transaction state from 'WAITING FO
R REPLY' to 'PROCESSING REPLY'
```

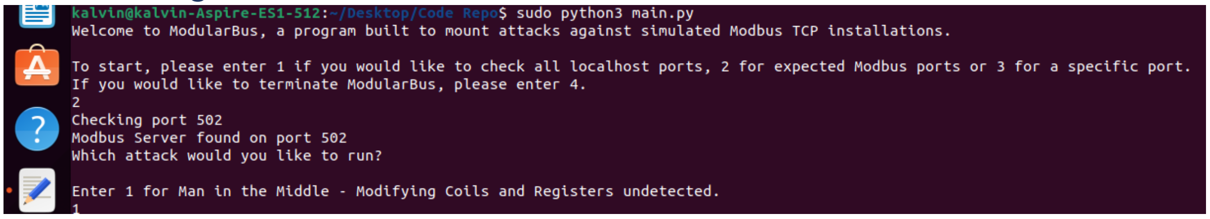


```
C:\Users\kalvi\Desktop\Code Repo>py server.py
2022-04-28 17:03:29,421 MainThread      DEBUG    sync           :348      Started t
hread to serve client at ('192.168.0.16', 52994)
2022-04-28 17:03:29,425 Thread-1        DEBUG    sync           :46       Client Co
nnected [192.168.0.16:52994]
2022-04-28 17:03:29,449 Thread-1        DEBUG    sync           :199      Handling
data: 0x0 0x1 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0x0 0x0 0x5
2022-04-28 17:03:29,449 Thread-1        DEBUG    socket_framer  :147      Processin
g: 0x0 0x1 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0x0 0x0 0x5
2022-04-28 17:03:29,450 Thread-1        DEBUG    factory        :137      Factory R
equest[ReadCoilsRequest: 1]
2022-04-28 17:03:29,450 Thread-1        DEBUG    context        :63       validate:
 fc-[1] address-1: count-5
2022-04-28 17:03:29,450 Thread-1        DEBUG    context        :77       getValues
 fc-[1] address-1: count-5
2022-04-28 17:03:29,450 Thread-1        DEBUG    sync           :229      send: [Re
adCoilsResponse(5)]- b'000100000004010101 00'
2022-04-28 17:03:29,466 Thread-1        DEBUG    sync           :199      Handling
data: 0x0 0x2 0x0 0x0 0x0 0x6 0x1 0x3 0x0 0x0 0x0 0x5
2022-04-28 17:03:29,466 Thread-1        DEBUG    socket_framer  :147      Processin
g: 0x0 0x2 0x0 0x0 0x0 0x6 0x1 0x3 0x0 0x0 0x0 0x5
2022-04-28 17:03:29,466 Thread-1        DEBUG    factory        :137      Factory R
equest[ReadHoldingRegistersRequest: 3]
2022-04-28 17:03:29,466 Thread-1        DEBUG    context        :63       validate:
 fc-[3] address-1: count-5
2022-04-28 17:03:29,467 Thread-1        DEBUG    context        :77       getValues
 fc-[3] address-1: count-5
2022-04-28 17:03:29,467 Thread-1        DEBUG    sync           :229      send: [Re
adHoldingRegistersResponse (5)]- b'00020000000d01030a0000000000000000000000'
```

Figure 21 – Client and Server combination running. The client is connecting from the IP 192.168.0.16

As mentioned previously, this attack will be focusing on the Pymodbus implementation of a Modbus installation. The above figure shows the server, which represents Modbus slaves, and the client, which represents the master device, running synchronously. In this scenario, the client is running read requests every five seconds on the coils and registers of the first unit (slave) in the server. The attack will be targeting the values within the registers and coils, and modifying them to unrealistic and unexpected values.

6.3.1 Running the Tool



```
kalvin@kalvin-Aspire-ES1-512:~/Desktop/Code Repo$ sudo python3 main.py
Welcome to ModularBus, a program built to mount attacks against simulated Modbus TCP installations.

To start, please enter 1 if you would like to check all localhost ports, 2 for expected Modbus ports or 3 for a specific port.
If you would like to terminate ModularBus, please enter 4.
2
Checking port 502
Modbus Server found on port 502
Which attack would you like to run?

Enter 1 for Man in the Middle - Modifying Coils and Registers undetected.
1
```

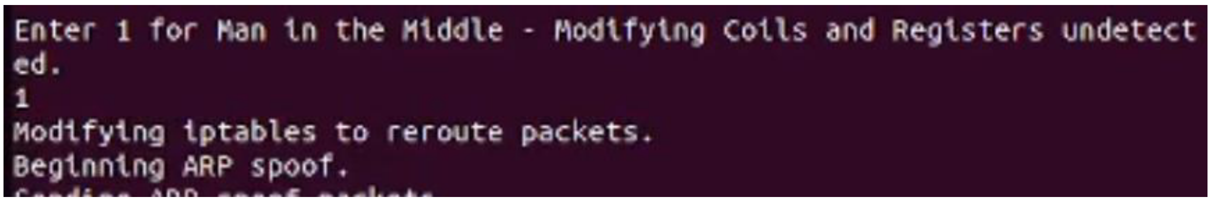
Figure 22 – Running main.py on the attacking device and choosing option 2 to check expected Modbus ports.

On the attacking device, Figure 22 shows what happens when main.py is running. Because the attacker is assumed to have access to the network and would have found all the information needed for the attack, there is no functionality provided for finding IP addresses or MAC addresses within the tool yet. This information is hardcoded into the program for this demonstration, but in a real scenario, in the current version of the tool, the attacker would add this information themselves based on their own surveillance of the network.

The user is given four options after running the script; they can check all of the local ports on the IP address suspected to have the server on it, they can check ports that would typically use Modbus, such as 502, they can check a single port or they can quit. The more ports checked, however, the more invasive the attack is, meaning it would be better to use option 2 most of the time to avoid detection, as the majority of the time Modbus installations will use port 502.

In this example, the attacker has chosen option 2. In order, port 502, 5020 and 50200 will be checked, and in this scenario, the Modbus server is being hosted on port 502.

Once the port has been found, the tool will progress to asking which attack the user would want to launch. Where the tool is meant to be easy to add to, in the future it could be expanded upon with different attacks. At the moment, there is only the Man in the Middle attack developed as a part of this project, so the user can only select that.

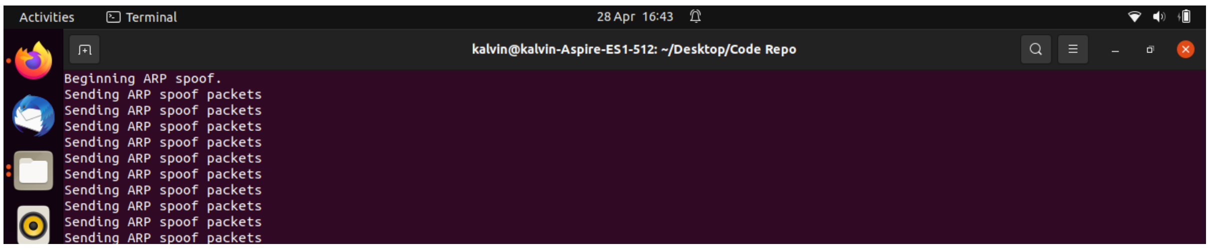


```
Enter 1 for Man in the Middle - Modifying Coils and Registers undetect
ed.
1
Modifying iptables to reroute packets.
Beginning ARP spoof.
```

Figure 23 – Output showing that iptables is being modified

6.3.2 ARP Spoof

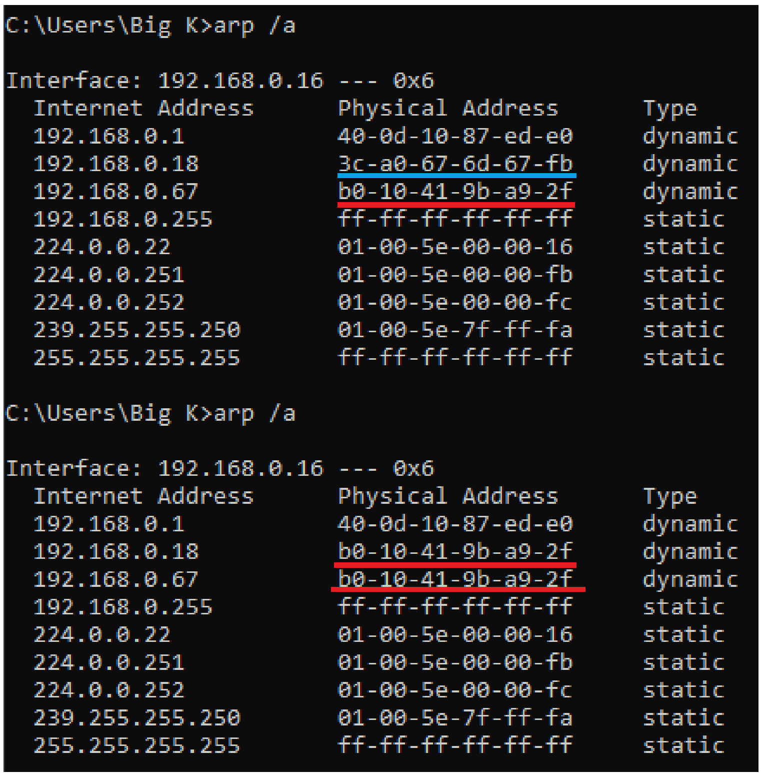
For the ARP spoof to work successfully, a rule needs to be added to iptables which will manage and reroute packets between the client and server device for the duration of the attack. Figure 22 shows the user confirmation that this has happened.



```
Beginning ARP spoof.
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
```

Figure 24 – ARP Spoof output between Client and Server device

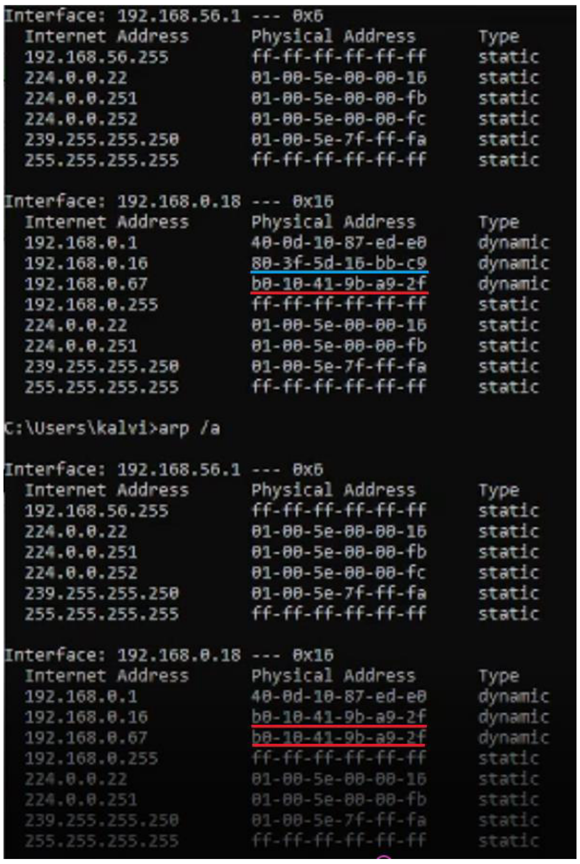
Once the user starts the MITM attack by pressing 1, ARP packets are created using Scapy that are sent to both the Client and the Server device. These packets update the ARP table in the client device, linking the attacking device's MAC address to the server's IP address. This will mean that when packets are being routed to the server, instead they will be forced to the attacker's device, and the client device wouldn't be able to realise that what was happening was wrong.



```
C:\Users\Big K>arp /a

Interface: 192.168.0.16 --- 0x6
  Internet Address      Physical Address      Type
  192.168.0.1           40-0d-10-87-ed-e0     dynamic
  192.168.0.18          3c-a0-67-6d-67-fb     dynamic
  192.168.0.67          b0-10-41-9b-a9-2f     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Users\Big K>arp /a

Interface: 192.168.0.16 --- 0x6
  Internet Address      Physical Address      Type
  192.168.0.1           40-0d-10-87-ed-e0     dynamic
  192.168.0.18          b0-10-41-9b-a9-2f     dynamic
  192.168.0.67          b0-10-41-9b-a9-2f     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

Figure 25 - ARP Table in the Client Device before and after the ARP spoof starts.

The same happens to the server, linking the attacking device's MAC address to the client's IP address. In both cases, unless there was a system in place to monitor either the network traffic or the ARP tables of both the client and server device, this would happen undetected.

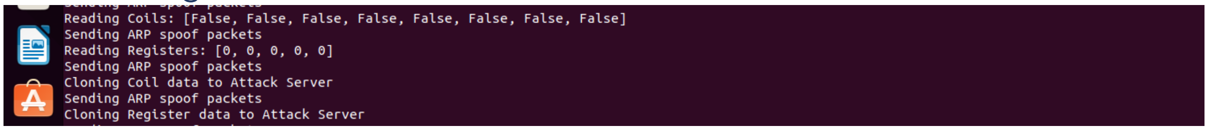


```
Interface: 192.168.56.1 --- 0x6
  Internet Address      Physical Address      Type
  192.168.56.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

Interface: 192.168.0.18 --- 0x16
  Internet Address      Physical Address      Type
  192.168.0.1           40-0d-10-87-ed-e0     dynamic
  192.168.0.16          80-3f-5d-16-bb-c9     dynamic
  192.168.0.67          b0-10-41-9b-a9-2f     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Users\kalvi>arp /a

Interface: 192.168.56.1 --- 0x6
  Internet Address      Physical Address      Type
  192.168.56.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

Interface: 192.168.0.18 --- 0x16
  Internet Address      Physical Address      Type
  192.168.0.1           40-0d-10-87-ed-e0     dynamic
  192.168.0.16          b0-10-41-9b-a9-2f     dynamic
  192.168.0.67          b0-10-41-9b-a9-2f     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

Figure 26 – ARP Table in the Server device before and after the ARP spoof starts.

In both Figure 25 and 26, the MAC address of the attacking device is underlined in red and the MAC address of the affected device, be it the server or client, is underlined in blue. Once the ARP spoof starts, for both the server and client, the ARP table is updated to show the MAC address of the attacking device.

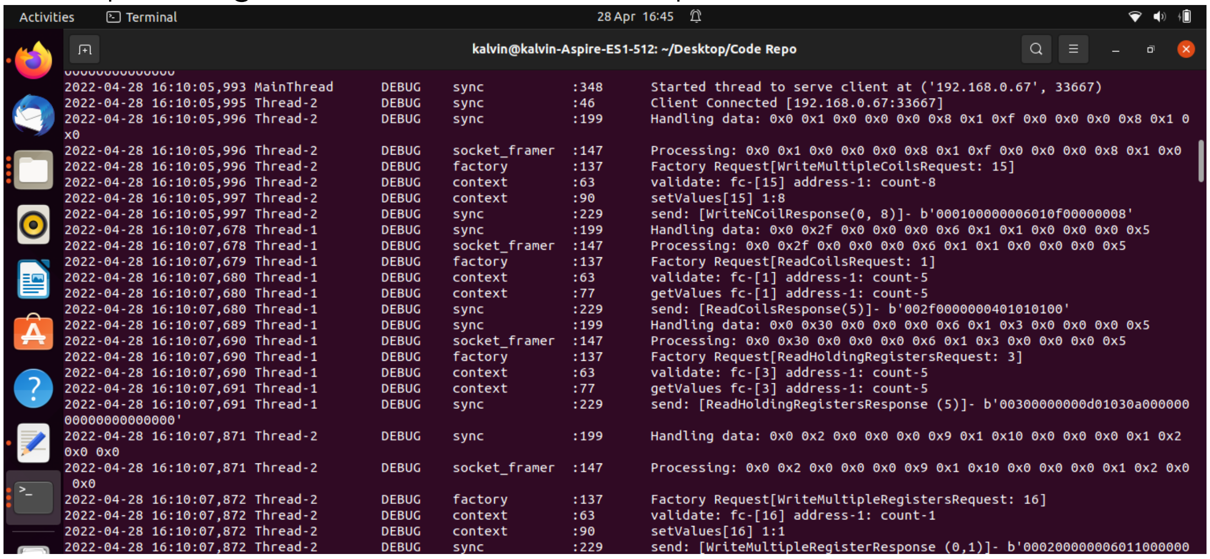
6.3.3 Cloning Data to Attack Server



```
Reading Coils: [False, False, False, False, False, False, False, False]
Sending ARP spoof packets
Reading Registers: [0, 0, 0, 0, 0]
Sending ARP spoof packets
Cloning Coil data to Attack Server
Sending ARP spoof packets
Cloning Register data to Attack Server
```

Figure 27 – Output showing the current values stored in the real server, and confirmation that the data has been sent to the fake server.

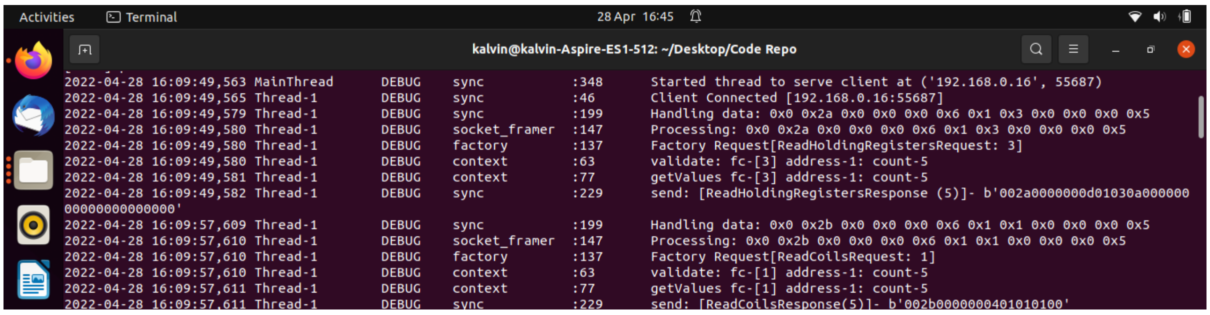
Using a pymodbus client, the attacking device is able to connect to the real server temporarily and copy the real data across to the fake server before it is modified by the attack. That way, when the real client connects to the fake server because of the ARP spoof, the data being read will be exactly the same, providing the attacker with cover to perform the rest of the attack.



```
2022-04-28 16:10:05,993 MainThread      DEBUG    sync           :348      Started thread to serve client at ('192.168.0.67', 33667)
2022-04-28 16:10:05,995 Thread-2        DEBUG    sync           :46       Client Connected [192.168.0.67:33667]
2022-04-28 16:10:05,996 Thread-2        DEBUG    sync           :199      Handling data: 0x0 0x1 0x0 0x0 0x0 0x8 0x1 0xf 0x0 0x0 0x0 0x8 0x1 0
x0
2022-04-28 16:10:05,996 Thread-2        DEBUG    socket_framer  :147      Processing: 0x0 0x1 0x0 0x0 0x0 0x8 0x1 0xf 0x0 0x0 0x0 0x8 0x1 0x0
2022-04-28 16:10:05,996 Thread-2        DEBUG    factory        :137      Factory Request[WriteMultipleCoilsRequest: 15]
2022-04-28 16:10:05,996 Thread-2        DEBUG    context        :63       validate: fc-[15] address-1: count-8
2022-04-28 16:10:05,997 Thread-2        DEBUG    context        :90       setValues[15] 1:8
2022-04-28 16:10:05,997 Thread-2        DEBUG    sync           :229      send: [WriteNCoilResponse(0, 8)]- b'000100000006010f00000008'
2022-04-28 16:10:07,678 Thread-1        DEBUG    sync           :199      Handling data: 0x0 0x2f 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0x0 0x0 0x5
2022-04-28 16:10:07,678 Thread-1        DEBUG    socket_framer  :147      Processing: 0x0 0x2f 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0x0 0x0 0x5
2022-04-28 16:10:07,679 Thread-1        DEBUG    factory        :137      Factory Request[ReadCoilsRequest: 1]
2022-04-28 16:10:07,680 Thread-1        DEBUG    context        :63       validate: fc-[1] address-1: count-5
2022-04-28 16:10:07,680 Thread-1        DEBUG    context        :77       getValues fc-[1] address-1: count-5
2022-04-28 16:10:07,680 Thread-1        DEBUG    sync           :229      send: [ReadCoilsResponse(5)]- b'002f0000000401010100'
2022-04-28 16:10:07,689 Thread-1        DEBUG    sync           :199      Handling data: 0x0 0x30 0x0 0x0 0x0 0x6 0x1 0x3 0x0 0x0 0x0 0x5
2022-04-28 16:10:07,690 Thread-1        DEBUG    socket_framer  :147      Processing: 0x0 0x30 0x0 0x0 0x0 0x6 0x1 0x3 0x0 0x0 0x0 0x5
2022-04-28 16:10:07,690 Thread-1        DEBUG    factory        :137      Factory Request[ReadHoldingRegistersRequest: 3]
2022-04-28 16:10:07,690 Thread-1        DEBUG    context        :63       validate: fc-[3] address-1: count-5
2022-04-28 16:10:07,691 Thread-1        DEBUG    context        :77       getValues fc-[3] address-1: count-5
2022-04-28 16:10:07,691 Thread-1        DEBUG    sync           :229      send: [ReadHoldingRegistersResponse (5)]- b'00300000000d01030a000000
00000000000000'
2022-04-28 16:10:07,871 Thread-2        DEBUG    sync           :199      Handling data: 0x0 0x2 0x0 0x0 0x0 0x9 0x1 0x10 0x0 0x0 0x0 0x1 0x2
0x0 0x0
2022-04-28 16:10:07,871 Thread-2        DEBUG    socket_framer  :147      Processing: 0x0 0x2 0x0 0x0 0x0 0x9 0x1 0x10 0x0 0x0 0x0 0x1 0x2 0x0
 0x0
2022-04-28 16:10:07,872 Thread-2        DEBUG    factory        :137      Factory Request[WriteMultipleRegistersRequest: 16]
2022-04-28 16:10:07,872 Thread-2        DEBUG    context        :63       validate: fc-[16] address-1: count-1
2022-04-28 16:10:07,872 Thread-2        DEBUG    context        :90       setValues[16] 1:1
2022-04-28 16:10:07,872 Thread-2        DEBUG    sync           :229      send: [WriteMultipleRegisterResponse (0,1)]- b'000200000006011000000
```

Figure 28 – Data being cloned to the fake server using the values pulled from the real server

Figure 28 shows the data that was extracted from the real server previously being copied into the registers and coils of the fake server. This way, when the real client is connected to the fake server and begins reading data again, there will be no discernible difference, which should keep the attack covert.

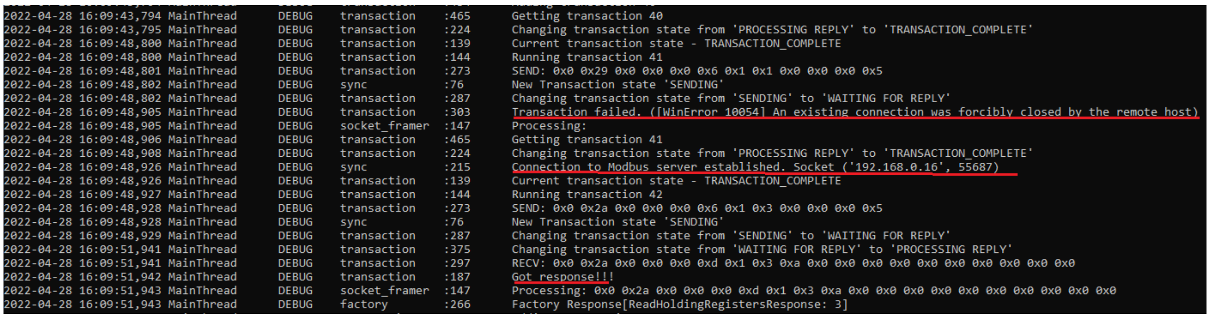


```
2022-04-28 16:09:49,563 MainThread      DEBUG    sync           :348      Started thread to serve client at ('192.168.0.16', 55687)
2022-04-28 16:09:49,565 Thread-1        DEBUG    sync           :46       Client Connected [192.168.0.16:55687]
2022-04-28 16:09:49,579 Thread-1        DEBUG    sync           :199      Handling data: 0x0 0x2a 0x0 0x0 0x0 0x6 0x1 0x3 0x0 0x0 0x0 0x5
2022-04-28 16:09:49,580 Thread-1        DEBUG    socket_framer  :147      Processing: 0x0 0x2a 0x0 0x0 0x0 0x6 0x1 0x3 0x0 0x0 0x0 0x5
2022-04-28 16:09:49,580 Thread-1        DEBUG    factory        :137      Factory Request[ReadHoldingRegistersRequest: 3]
2022-04-28 16:09:49,580 Thread-1        DEBUG    context        :63       validate: fc-[3] address-1: count-5
2022-04-28 16:09:49,581 Thread-1        DEBUG    context        :77       getValues fc-[3] address-1: count-5
2022-04-28 16:09:49,582 Thread-1        DEBUG    sync           :229      send: [ReadHoldingRegistersResponse (5)]- b'002a0000000d01030a000000
00000000000000'
2022-04-28 16:09:57,609 Thread-1        DEBUG    sync           :199      Handling data: 0x0 0x2b 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0x0 0x0 0x5
2022-04-28 16:09:57,610 Thread-1        DEBUG    socket_framer  :147      Processing: 0x0 0x2b 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0x0 0x0 0x5
2022-04-28 16:09:57,610 Thread-1        DEBUG    factory        :137      Factory Request[ReadCoilsRequest: 1]
2022-04-28 16:09:57,610 Thread-1        DEBUG    context        :63       validate: fc-[1] address-1: count-5
2022-04-28 16:09:57,611 Thread-1        DEBUG    context        :77       getValues fc-[1] address-1: count-5
2022-04-28 16:09:57,611 Thread-1        DEBUG    sync           :229      send: [ReadCoilsResponse(5)]- b'002b0000000401010100'
```

Figure 29 – Real client connected to the fake server hosted on the attacking device.

6.3.4 Client Connected to Attack Server

Once the iptables rule is in place, and the ARP spoof has started, the real client will suffer a temporary disconnection before reconnecting to the fake server which has been populated using data taken from the real server. The client will continue running read requests against the coils and registers from the fake server, and will continue receiving responses from what is shown to be the real server containing real data.

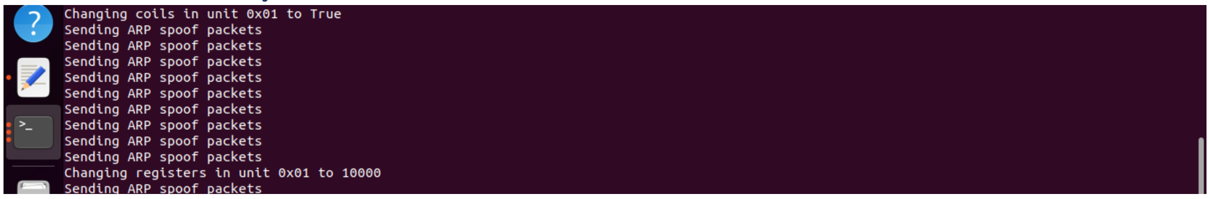


```
2022-04-28 16:09:43,794 MainThread     DEBUG    transaction    :465      Getting transaction 40
2022-04-28 16:09:43,795 MainThread     DEBUG    transaction    :224      Changing transaction state from 'PROCESSING REPLY' to 'TRANSACTION_COMPLETE'
2022-04-28 16:09:48,800 MainThread     DEBUG    transaction    :139      Current transaction state - TRANSACTION_COMPLETE
2022-04-28 16:09:48,800 MainThread     DEBUG    transaction    :144      Running transaction 41
2022-04-28 16:09:48,801 MainThread     DEBUG    transaction    :273      SEND: 0x0 0x29 0x0 0x0 0x0 0x6 0x1 0x1 0x0 0x0 0x0 0x5
2022-04-28 16:09:48,802 MainThread     DEBUG    sync           :76       New Transaction state 'SENDING'
2022-04-28 16:09:48,802 MainThread     DEBUG    transaction    :287      Changing transaction state from 'SENDING' to 'WAITING FOR REPLY'
2022-04-28 16:09:48,905 MainThread     DEBUG    transaction    :303      Transaction failed. ([WinError 10054] An existing connection was forcibly closed by the remote host)
2022-04-28 16:09:48,905 MainThread     DEBUG    socket_framer  :147      Processing:
2022-04-28 16:09:48,906 MainThread     DEBUG    transaction    :465      Getting transaction 41
2022-04-28 16:09:48,908 MainThread     DEBUG    transaction    :224      Changing transaction state from 'PROCESSING REPLY' to 'TRANSACTION_COMPLETE'
2022-04-28 16:09:48,926 MainThread     DEBUG    sync           :215      Connection to Modbus server established. Socket ('192.168.0.16', 55687)
2022-04-28 16:09:48,926 MainThread     DEBUG    transaction    :139      Current transaction state - TRANSACTION_COMPLETE
2022-04-28 16:09:48,927 MainThread     DEBUG    transaction    :144      Running transaction 42
2022-04-28 16:09:48,928 MainThread     DEBUG    transaction    :273      SEND: 0x0 0x2a 0x0 0x0 0x0 0x6 0x1 0x3 0x0 0x0 0x0 0x5
2022-04-28 16:09:48,928 MainThread     DEBUG    sync           :76       New Transaction state 'SENDING'
2022-04-28 16:09:48,929 MainThread     DEBUG    transaction    :287      Changing transaction state from 'SENDING' to 'WAITING FOR REPLY'
2022-04-28 16:09:51,941 MainThread     DEBUG    transaction    :375      Changing transaction state from 'WAITING FOR REPLY' to 'PROCESSING REPLY'
2022-04-28 16:09:51,941 MainThread     DEBUG    transaction    :297      RECV: 0x0 0x2a 0x0 0x0 0x0 0xd 0x1 0x3 0xa 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0
2022-04-28 16:09:51,942 MainThread     DEBUG    transaction    :187      Got response!!!
2022-04-28 16:09:51,943 MainThread     DEBUG    socket_framer  :147      Processing: 0x0 0x2a 0x0 0x0 0x0 0xd 0x1 0x3 0xa 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0
2022-04-28 16:09:51,943 MainThread     DEBUG    factory        :266      Factory Response[ReadHoldingRegistersResponse: 3]
```

Figure 30 – Real client losing connection to the real server and reconnecting to the fake server

The three underlined sections in Figure 30 show what happens in stages when the ARP spoof takes place. First, the client's most recent 'transaction' (command sent to the server) will fail, since the connection to the real server has been blocked. It will then attempt to re-establish the connection, which it does successfully, leading to what is shown in Figure 26 occurring. Once the connection is re-established and it has confirmed that the connection is working, it will print a 'Got response!!!' message to show that this is the case, and normal function can resume.

6.3.5 Malicious Injection



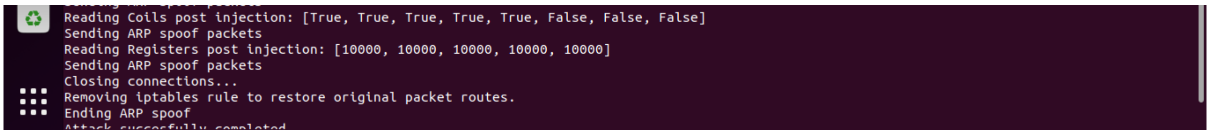
```
Changing coils in unit 0x01 to True
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Sending ARP spoof packets
Changing registers in unit 0x01 to 10000
Sending ARP spoof packets
```

Figure 31 – Modifying the data in the coils and registers of the real server.

At this stage of the attack, the attacker is in complete control of the server and what happens to it. Without external assistance, there would be little the client could do to realise it has been spoofed.

In this attack, it was important to make sure that the attacker could be in and out of the server as fast as possible, so it was clear that the way to cause the most damage quickly was to inject unrealistic values into the registers and coils. Like mentioned previously, there would be little way of knowing what would happen by doing this in a real scenario without further knowledge of the devices under attack, but it is a safe assumption that doing this would cause catastrophic damage.

In this example, the attack writes the value 10000 to all the registers in the first slave device, and changes some of the coils to True. However, it is important to remember that at this stage, this is not the only thing the attacker could do. The attack could easily be modified to do more



```
Reading Coils post injection: [True, True, True, True, True, False, False, False]
Sending ARP spoof packets
Reading Registers post injection: [10000, 10000, 10000, 10000, 10000]
Sending ARP spoof packets
Closing connections...
Removing iptables rule to restore original packet routes.
Ending ARP spoof
```

Figure 32– Output showing the result of the attack

Following the injection, the tool shows the attacker what values have been added to the registers and coils. After this, the connections to the fake server and real server are closed, the iptables rule is deleted and the ARP spoof is ended. At this stage, the attack is successfully complete. The real client will then re-establish its connection to the real server, at which point it will start reading the injected data. At that moment, it would then become clear an attack has taken place.

6.3.6 Video Demonstration

A full demonstration with an audio commentary of this attack happening against the same scenario is available at https://youtu.be/BkxGO5zLyhQ

6.4 Summary

The attack created within this project can be deadly if there are no secure measures in place, especially in critical environments like a centrifuge. It functions as intended, hijacking a client's connection to the Modbus server and injecting whatever malicious commands have been coded into it.

7 Evaluation

7.1 Overview

This chapter will evaluate the man in the middle attack, comparing the final product to the original aims and objectives set. The tool itself will be tested, and its performance will be evaluated against existing solutions and protective measures. I will then discuss out of the solutions the most effective prevention, taking into consideration everything a company would need to think about when choosing defence options.

7.2 Comparison to Aims and Objectives

The original aims of this project were as follows:

1 – Find a gap in the current research on the Modbus protocol that an attack could be developed to take advantage of.

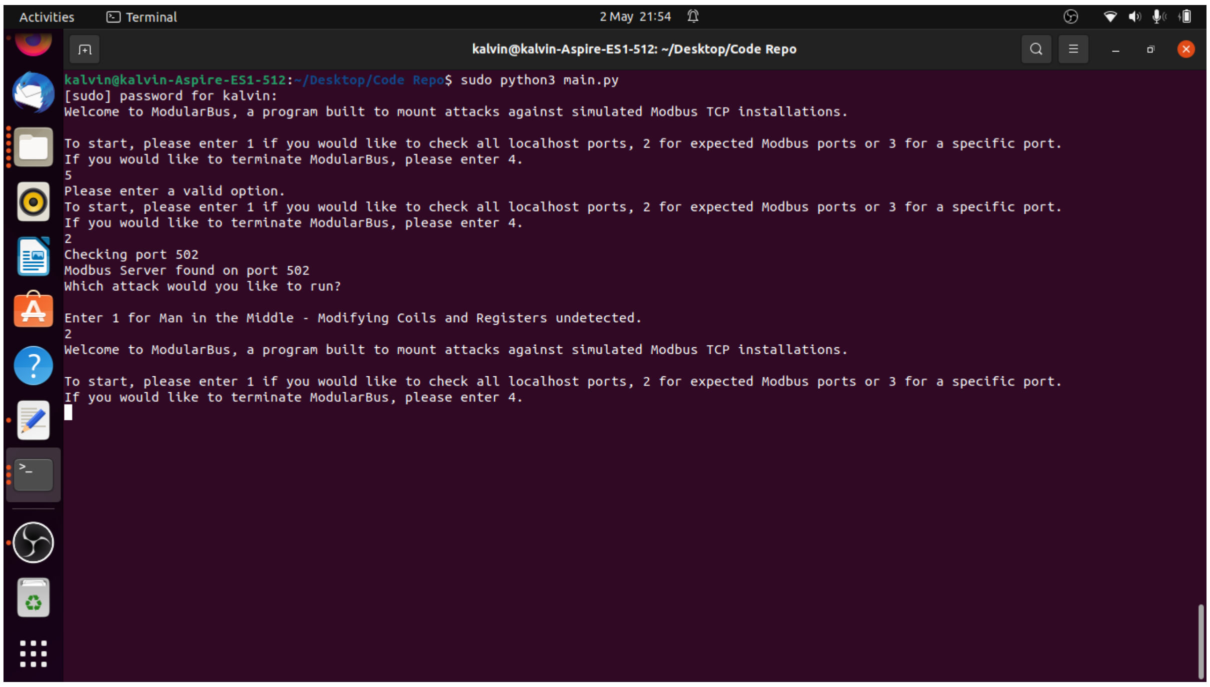
2 – Implement an attack on the Modbus protocol to attack a simulated ICS. This should be within the research gap identified in objective one. The simulation will be a client/server pair, with the main target being the server.

3 – Develop a tool that can automate the attack developed previously and evaluate how effective it is compared to existing methods. This tool could include the functionality to produce attack performance statistics such as time to attack completion, or even comparing the attack to others that are mounted on the tool.

Within this project, the deliverables created meet aims one and two, and partially meet aim 3. Due to time constraints, automatic attack statistics have not been implemented into the tool, however this is something that could be added should this project be expanded on in the future.

7.3 Testing the Tool

Through testing the final tool and attack, it does what it intends to do, and injects whatever commands the attacker would like into a Modbus server without being detected, assuming that the attacker has already infiltrated the network and has acquired the necessary information to carry out the attack prior. From this, I am confident that the final product of this project is a fully functioning attack, again presuming the same preliminaries I did in the demonstration. In the future, the tool could be expanded to include the ability to find IP and MAC addresses to reduce the need for the attacker to intrude on the network more than necessary. The tool also cannot take invalid options in the stages of choosing an option for the port checker and the stage in which the user chooses the attack to mount, so user error should be very unlikely, and in each stage the user is told exactly what options are available to them, meaning that the tool should not have errors occur unless the user manages to input an invalid option at the start.



```
kalvin@kalvin-Aspire-ES1-512:~/Desktop/Code Repo$ sudo python3 main.py
[sudo] password for kalvin:
Welcome to ModularBus, a program built to mount attacks against simulated Modbus TCP installations.

To start, please enter 1 if you would like to check all localhost ports, 2 for expected Modbus ports or 3 for a specific port.
If you would like to terminate ModularBus, please enter 4.
5
Please enter a valid option.
To start, please enter 1 if you would like to check all localhost ports, 2 for expected Modbus ports or 3 for a specific port.
If you would like to terminate ModularBus, please enter 4.
2
Checking port 502
Modbus Server found on port 502
Which attack would you like to run?

Enter 1 for Man in the Middle - Modifying Coils and Registers undetected.
2
Welcome to ModularBus, a program built to mount attacks against simulated Modbus TCP installations.

To start, please enter 1 if you would like to check all localhost ports, 2 for expected Modbus ports or 3 for a specific port.
If you would like to terminate ModularBus, please enter 4.
```

Figure 33 – Showing what occurs when invalid options are entered in main.py

As shown above, entering invalid options at any point will return the user to the start of the tool.

7.4 Attack Performance

From the video created as a part of this report, the attack itself after the user has selected options takes fifty seconds. Getting to the attack is as fast as the user wants it to be. For example if the user were to click 2, and then 1 as fast as possible, they will have launched the attack within seconds provided the server is being hosted on an expected Modbus port. On average, attackers will launch an attack every 39 seconds [21], meaning the attack built as a part of this project falls short of the execution time required to achieve this.

Even though the attack is not as fast as it could be, the potential damage caused is limitless. Depending on the system that is using Modbus that is being attacked, the damage could be irreparable and very deadly. For instance, there are 153 devices recognised by Modbus.org [22], ranging from data acquisition devices to centrifuges. Attacks like the one created here that inject random data would have random, potentially fatal consequences which could be further amplified if the attack was optimised to damage its target device as much as possible.

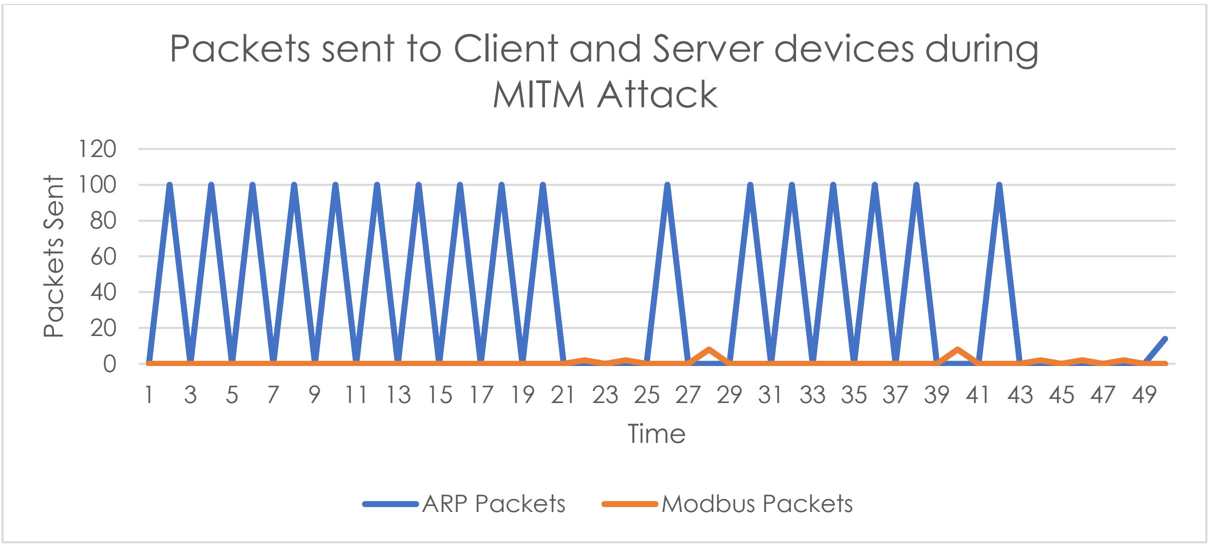


Figure 34 – Graph showing network activity during the attack

Figure 34 shows the presence the attacker will have on the network across the duration of the attack. Preventative measures that monitor the network would be able to uncover the attack within the first few seconds, due to the amount of ARP packets being used. However, to guarantee that the ARP spoof takes place without being interrupted by either the client or server sending its own ARP requests, a large amount is needed.

7.5 MITM vs Existing Tools and Solutions

The table below shows how my attack would operate against the solutions found during the Background stage in the project. For each solution, it is presumed again that the attacker has full access to the network, and all data needed to perform the attack is available to them. It is also presumed that the only defensive mechanism present on the network is the solution in question.

Table 4 – Solutions compared against the MITM Attack

Solution	Would the MITM succeed?	Discussion
Tofino Security Appliance [12]	No	The Tofino Security Appliance regulates all of the Modbus messages transmitted across the network. Because the Man in the Middle attack sends Modbus messages in order to inject malicious data, it would be stopped immediately.
Using a VPN [23]	Yes, under certain conditions	If a VPN is only being configured to prevent attacks from the Internet, then this attack will still work. However, if the VPN is being used as

		a subnet which only authorised devices can access, this attack will not succeed.
Modbus Transfer Service (MBTS) [25]	No	MBTS protects the existing network from attacks by transmitting all data into a separate data diode securely. This connection itself would be extremely difficult to intercept and take advantage of through a man in the middle attack, and makes attacking the connection between the master and slave devices futile as MBTS protects them.
SolarWinds NetFlow Analyzer [26]	Yes	This tool itself is only a deep packet analyser, meaning it would need to be combined with a method of closing connections when fraudulent packets are detected. However, should the ARP spoof be carried out correctly, the NetFlow Analyzer should be tricked. This would need to be tested.
Use the Modbus/TCP Security Protocol [10] instead of the original Modbus TCP.	Yes	The Modbus TCP Security protocol only encrypts data. As the Man in the Middle attack is not concerned with extracting information, only replicating the existing server and then transmitting commands of its own, the attack would still succeed. However, if the attacker had a secondary intention of reading the real commands as the attack happened, then the attack would not meet all of the requirements.
Use OPC Unified Architecture [13]	Yes, under certain conditions	OPC Unified Architecture prevents attacks from occurring from the internet. Therefore, if this attack was launched physically within the network, it would work, but if any attempt was made from outside the network, chances are it would fail.
Anomaly-based change detection algorithm [9]	Yes, but it will be detected	This method of prevention will not directly prevent the attack, but rather alert relevant authorities that an attack is happening. The ARP spoof is intensive as it will flood both the client and server machines with

		ARP packets, which would be detected by this algorithm. If the attacker isn't concerned by this, then the attack could proceed as normal, however staying covert is essential then another attack should be used.
Signature-based detection [9]	Yes, but it will be detected	Like the above algorithm, the attack will still work, however it will be obvious an attack is still happening. This algorithm monitors network traffic, which means it would detect both the ARP spoof and the malicious Modbus commands being transmitted. If the attacker is not concerned by this, then the attack can happen, but again like the above algorithm if being covert is essential then this approach should not be taken.

7.6 Preventing the MITM Attack

The Tofino Security Appliance by far would be the best solution to prevent the Man in the Middle attack from occurring, however the biggest issue with it is the cost. Small companies may not be able to spare the money to use it, in which case it would be better to use one of the detection algorithms combined with a method to cut off all network traffic. That way, operations could temporarily halt until the attacker is located and removed from the network. Alternatively, using a VPN could also hinder the attack, as if it is configured correctly, the attacker would not be able to gain access to the network unless they were using an authorised device – however, if the attacker had access to an authorised device, they wouldn't need to perform a man in the middle attack.

Yet against the majority of the solutions, when they are the only defensive mechanism on the network, the attack should work. In most cases it will be detected, but the malicious effect of the attack will still succeed, disrupting or damaging the target as intended.

Overall, from the above points, the most viable solution, considering all factors that may affect businesses trying to protect themselves and cost, would be using the signature detection combined with a method to prevent network transmissions, like activating a full firewall on both the client and server and suspending all of their transmissions. Likewise, a correctly configured VPN would also be a wise decision, if a company was on a low budget, as that way their entire network would be secure, protecting the devices from not only a man in the middle attack, but countless others also.

7.7 Summary

In summary, the attack created can be stopped should the network it is on be locked down. However, in most scenarios, the attack would remain undetected on a network. The recommendation given is that an intrusion detection technique would be an effective measure to see a man in the middle attack happening. If it were to be combined with a method of locking down the network, it would be a strong countermeasure.

8 Future Work

This project is fortunate in that it can be expanded in several different directions in the future. If time allowed, I would have liked to test the attack against more different installations of the Modbus TCP protocol to see if it is as effective against more than just a python simulation. The attack itself could become more modular, being able to tailor itself to cause the most damage against a server based on whatever kind of information is stored within the registers.

Furthermore, it would have been useful to create several other attacks and compare them to the attack created as a part of this project. That way, the automated tool could transition into a tool that compares the effectiveness of attacks, rather than just a tool to mount attacks on.

Within the code itself, the attack could have been written more efficiently, bringing down the attack time to below the 39 seconds referenced earlier. This would mean the attack itself would have less chance of being detected in a system without any protective solution, as the attacker would have less of a presence in the network. Furthermore, the tool could be developed to use polymorphism and inheritance. This way, a generic attack class could be used as a basis for all created attacks, meaning there would be less duplicated code, reducing the overall size of the tool. Another addition to the tool would be removing the need to have MAC addresses and IP addresses hardcoded into the attacks and the server finding algorithm. That way, the tool becomes even more automated, reducing the need for potentially invasive surveillance on a network in which the attacker may not even have the time to perform any.

The tool could eventually be recreated with a graphical user interface, allowing users to interact more freely with the tool, rather than using the command line as the main source of input and output. Whilst I believe my reasoning behind using the command line/terminal as my user interface was justified as it is the more common choice for attack software, I think where the final product is meant to be a tool for research, a more intuitive and user-friendly interface might encourage the tool to be used more.

9 Conclusion

In the initial plan of this project, I set out to create an automated tool that could host an attack that filled a research gap. I believe that I have achieved this. Even though I have not been able to include every feature I originally intended, I have successfully met every point in the original aims.

The attack itself does what it is meant to do; infiltrate a python simulation of Modbus TCP undetected, unleash malicious commands into the server and leave no trace of the attacker ever being on the network. ARP spoofing achieves all of this for the attack. Both the tool and attack meet all of the functional and non-functional requirements set out in the design, apart from the production of attack statistics during execution, but the main attack and tool itself is complete in requirements and functionality.

Furthermore, it's clear to see that the attack would be devastating in the real world, if it targeted a company that hadn't implemented any security around its PLCs. In a real-world scenario, the attack could cause life-threatening damage on a wide scale if it were used correctly, which proves the need for further investment in preventative measures for companies that use Modbus.

I did encounter several issues during this project. Now that I know how to tackle them and I have the hardware I need for a project like this, I am confident that if I attempted a similar project in the future I should be able to achieve more.

I think that it is a good overall result that the project is open to be taken in any direction now by myself or anyone interested in attacking the Modbus TCP protocol. As cyber-attacks develop over time, so will the capabilities of this tool, allowing users to compare new and innovative attacks to older ones, such as the MITM attack created alongside the tool.

My motivation behind taking on this project was to demonstrate the security vulnerabilities of the Modbus protocol. The attack I created was able to take advantage of the fact there was no built-in security with the protocol that would stop me from attacking the server, and through the Results section I showed that there were solutions that could both detect and prevent my attack from working. I think I have truly emphasized the importance of companies that use ICS' that run using the Modbus protocol applying at the minimum a detection algorithm, or even investing into security, purely because of how deadly the consequences of attacking the Modbus protocol can be.

10 Final Reflection

I think that within this project it was very valuable to do a large amount of background research into solutions against the Modbus protocol. It allowed me to look at building the attack in a different way to how I would have had I sat down and gone into it straight away. I can now appreciate how helpful it is to look at the background of a topic before developing in that area.

It has also helped me appreciate that from the position of an attacker, the system will rarely be as expected, meaning that I can simulate my attack against any kind of client/server pair with any amount of defence, but every system is created differently, with different settings and protection, so nothing would ever fully prepare me for creating an attack to be used in the real world as it is impossible to simulate every scenario.

I entered this project with a foundational understanding of cyber-security, with my specialism being in malicious software. This project has opened my eyes to the other areas of offensive attacks, like man in the middle, along with the other attacks I learned about during the research stages of the project. As somebody who is keen to learn more about the offensive side of cyber, it has definitely helped me to broaden my understanding of the tools available to an attacker. It was surprising to me that my normal method of disruption, which was writing malicious code, could be transferred to an attack that before this project I wouldn't have thought could be done in the same way.

The skills I have built upon will be transferable to future projects I create. I know now how important it is to prepare as much as possible for a project but expect to fall at hurdles along the way, as you can't prepare for everything during development and certain modules or implementations may not work as expected.

I also now appreciate defensive solutions more than ever. Creating an attack like this has shown me just how dangerous cyber-attacks can be, especially the kind like this one that could target critical systems and create life threatening situations through modifying a handful of values. By deepening my understanding of the danger of attacks, I know now that I will be able to build attacks and tools in the future with defence in mind, essentially building attacks only to try and find cost-effective solutions. Through this project, I also am now aware of the possible solutions to preventing man in the middle attacks, which can also be applied to other attacks, like packet injections. This knowledge can easily be passed on, which was one of the main focuses of this project.

I am content knowing that this project has helped me understand and appreciate cyber security better, and it can hopefully be used to show just how vulnerable the technology we depend on is and be a driver for change.

References

[1] prosoft-technology.com. 2005. *INTRODUCTION TO MODBUS TCP/IP*. [online] Available at: <https://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf> [Accessed 26 February 2022].

[2] Benbenishti, L., 2017. *SCADA MODBUS Protocol Vulnerabilities - Cyberbit*. [online] Available at: <https://www.cyberbit.com/blog/ot-security/scada-modbus-protocol-vulnerabilities/> [Accessed 6 February 2022].

[3] Zou, C., n.d. *Online Digital Forensics Courses and Labs*. [online] Cyberforensic.net. Available at: <http://cyberforensic.net/labs/modbus-attack.html> [Accessed 4 February 2022].

[4] Petrak, H., 2022. *PyModbus - A Python Modbus Stack — PyModbus 2.5.0 documentation*. [online] Available at: <https://pymodbus.readthedocs.io/en/latest/readme.html> [Accessed 26 February 2022].

[5] Yadav, G. and Paul, K., 2022 *Architecture and security of SCADA systems: A Review* [online] Available at: <https://www.sciencedirect.com/science/article/abs/pii/S1874548221000251.> [Accessed 20 February 2022].

[6] Parian, C., Guldimann, T. and Bhatia, S., 2020. *Fooling the Master: Exploiting Weakness in the Modbus Protocol*. [online] Available at: <https://www.sciencedirect.com/science/article/pii/S1877050920312576> [Accessed 20 February 2022].

[7] Huitsing, P., Chandia, R., Papa, M. and Shenoi, S., 2008. *Attack taxonomies for the Modbus protocols*. [online] Available at: <https://www.sciencedirect.com/science/article/pii/S187454820800005X> [Accessed 20 February 2022].

[8] Lamshöft, K. and Dittman, J., 2022. *Assessment of Hidden Channel Attacks: Targetting Modbus/TCP*. [online] Available at: <https://www.sciencedirect.com/science/article/pii/S240589632030536X> [Accessed 21 February 2022].

[9] Bhatia, S., Kush, N., Djamaludin, C., Akande, J. and Foo, E., 2014. *Practical modbus flooding attack and detection*. [online] Available at: <https://dl.acm.org/doi/10.5555/2667510.2667517> [Accessed 22 February 2022].

[10] Modbus.org. 2018. *MODBUS/TCP Security Protocol Specification*. [online] Available at: <https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf> [Accessed 13 March 2022].

[11] Porter, T. and Gough, M., 2007. *How to Cheat at VoIP Security*. [online] Available at: <https://www.sciencedirect.com/science/article/pii/B9781597491693500049?via%3Dihub> [Accessed 13 March 2022].

[12] Howard, S., 2022. *Using Modbus PLC's? Here's How To Protect Them | Tofino Industrial Security Solution*. [online] Available at: <https://www.tofinosecurity.com/blog/using-modbus-plcs-heres-how-protect-them> [Accessed 13 March 2022].

[13] Floyd, W., 2022. *Security Decisions in Modbus Industrial Systems*. [online] Control.com. Available at: <https://control.com/technical-articles/security-decisions-in-modbus-systems/> [Accessed 23 March 2022].

[14] Mazurczyk, W., Wendzel, S., and Cabaj, K., 2018. Towards deriving insights into data hiding methods using pattern-based approach. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 10. ACM [Accessed 29 March 2022].

[15] Jackson Higgins, K., 2022. *Ransomware Trained on Manufacturing Firms Led Cyberattacks in Industrial Sector*. [online] Dark Reading. Available at: <https://www.darkreading.com/attacks-breaches/ransomware-trained-on-manufacturing-firms-led-cyberattacks-in-industrial-sector> [Accessed 29 March 2022].

[16] Kali Linux. 2022. *hydra | Kali Linux Tools*. [online] Available at: <https://www.kali.org/tools/hydra/> [Accessed 5 April 2022].

[17] Snort. 2022. *Modbus Preprocessor | README.modbus*. [online] Available at: <https://www.snort.org/faq/133> [Accessed 6 April 2022].

[18] docs.microsoft.com. 2022. *Netsh commands for interface portproxy*. [online] Available at: <https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-interface-portproxy#delete-v4tov4> [Accessed 8 April 2022].

[19] Imperva. 2022. *What is ARP Spoofing | ARP Cache Poisoning Attack Explained | Imperva*. [online] Available at: <https://www.imperva.com/learn/application-security/arp-spoofing/> [Accessed 9 April 2022].

[20] Biondi, P., 2022. [online] Scapy. Available at: <https://scapy.net/> [Accessed 11 April 2022].

[21] Yoo, G., 2021. *Council Post: The Importance Of Time And Speed In Cybersecurity*. [online] Forbes. Available at: <https://www.forbes.com/sites/forbestechcouncil/2021/01/22/the-importance-of-time-and-speed-in-cybersecurity/> [Accessed 2 May 2022].

[22] Modbus.org. 2022. *Modbus Device Directory*. [online] Available at: <https://modbus.org/devices.php> [Accessed 2 May 2022].

[23] Accuenergy. 2022. *Protecting your Modbus TCP/IP Meter from Security Risks*. [online] Available at: <https://www.accuenergy.com/support/acuvim-ii-user-guide/protecting-your-modbus-tcp-ip-meter-from-security-risks/> [Accessed 4 May 2022].

[24] Security.org. 2022. *How Much Does a VPN Cost? | 2022 Average VPN Pricing*. [online] Available at: <https://www.security.org/vpn/cost/> [Accessed 4 May 2022].

[25] Owl Cyber Defense. 2022. *Modbus Transfer Service (MBTS) | Owl Cyber Defense*. [online] Available at: <https://owlcyberdefense.com/resource/modbus-transfer-service-mbts/> [Accessed 4 May 2022].

[26] Solarwinds.com. 2022. *NetFlow Analyzer - Analyze Remote Network Bandwidth Traffic | SolarWinds*. [online] Available at: <https://www.solarwinds.com/netflow-traffic-analyzer> [Accessed 4 May 2022].

[27] Combest, T., 2022. *Municipal Water Treatment Processes*. [online] Cosatx.us. Available at: <https://www.cosatx.us/home/showpublisheddocument/1010/635380914489870000> [Accessed 5 May 2022].

[28] Aluminummanufacturers.org. 2022. *Aluminium Sulphate* [online] Available at: <https://www.aluminummanufacturers.org/aluminum-sulfate/> [Accessed 5 May 2022].

[29] Federation Of American Scientists. 2022. *Centrifuges and Nuclear Weapon Proliferation*. [online] Available at: <https://fas.org/issues/nonproliferation-counterproliferation/nuclear-fuel-cycle/uranium-enrichment-gas-centrifuge-technology/centrifuges-nuclear-weapon-proliferation/> [Accessed 5 May 2022].

[30] Nyangaresi, V., Ogara, S. and Abeka, S., 2017. *TCP IP Header Attack Vectors and Countermeasures*. [online] Article.sciencepublishinggroup.com. Available at: <https://article.sciencepublishinggroup.com/html/10.11648.j.ajset.20170201.17.html> [Accessed 6 May 2022].

[31] KeyCDN. 2018. *TCP Flags - KeyCDN Support*. [online] Available at: <https://www.keycdn.com/support/tcp-flags> [Accessed 6 May 2022].

[32] APT, Inc. 2022. *Modbus Protocol | Security Vulnerabilities - APT, Inc.*. [online] Available at: <https://www.apt4power.com/2021/05/20/modbus-protocol-security-vulnerabilities/> [Accessed 6 May 2022].

[33] Veracode. 2022. *Man in the Middle Attack: Tutorial & Examples | Veracode*. [online] Available at: <https://www.veracode.com/security/man-middle-attack> [Accessed 6 May 2022].

[34] Rapid7. 2022. *Man in the Middle (MITM) Attacks | Types, Techniques, and Prevention*. [online] Available at: <https://www.rapid7.com/fundamentals/man-in-the-middle-attacks/> [Accessed 6 May 2022].

[35] Hoover, L., 2021. *What Is Qualitative vs. Quantitative Study?*. [online] GCU. Available at: <https://www.gcu.edu/blog/doctoral-journey/what-qualitative-vs-quantitative-study> [Accessed 6 May 2022].