13 Appendix

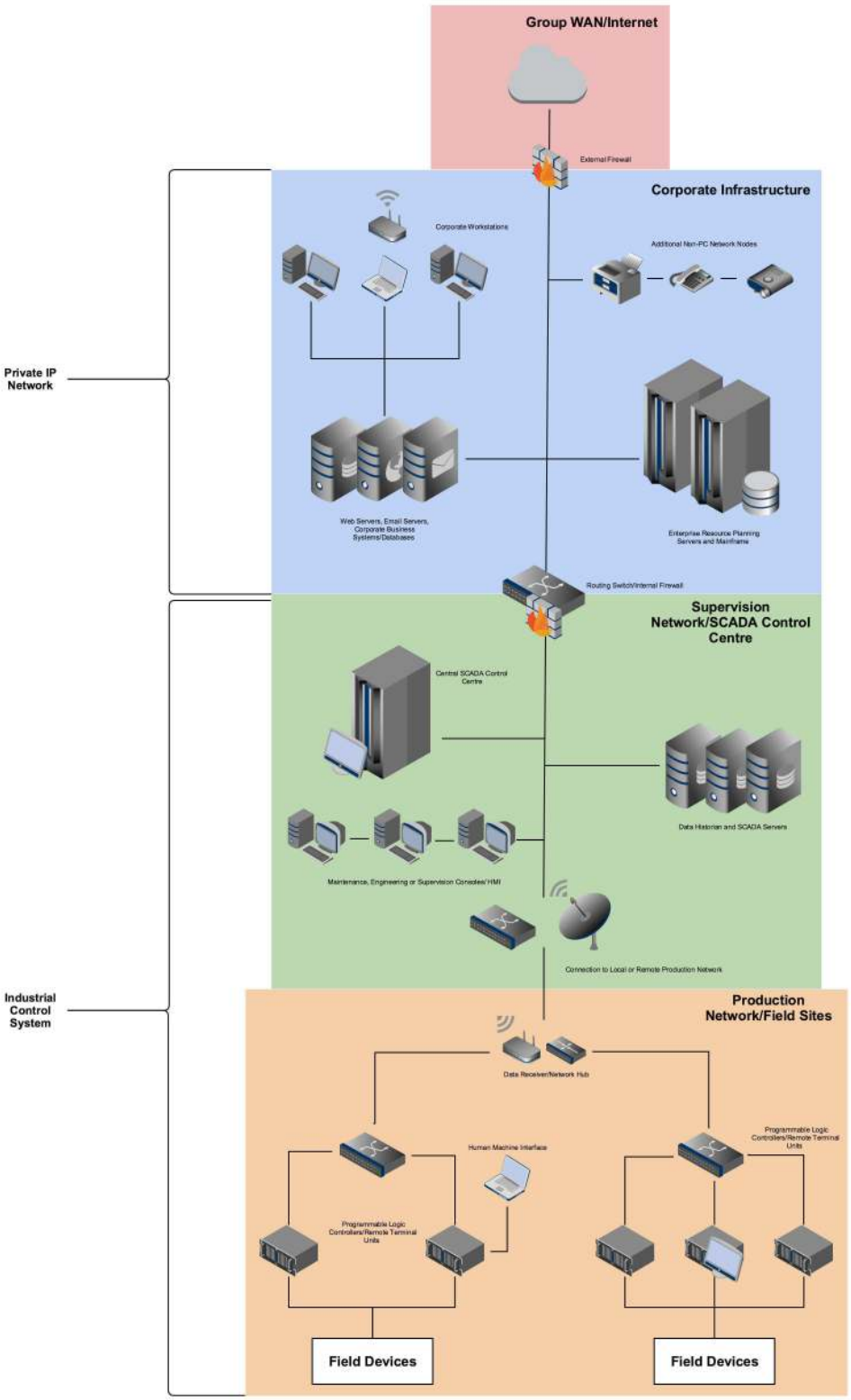
Appendix A Passive and Active Scanning Tools Table

Table 3: A summary of the existing active and passive network scanning tools.

Tool	Summary
NMap + ZMap	• Open Source, Active • Uses a combination of ping sweeping, SYN scanning and TCP connecting to determine which hosts reside on a network and which services they are operating. • Version detection or full TCP connection could cause legacy systems to misbehave. • NMap Scripting Engine has allowed for bespoke modules to be created for SCADA protocols such as Modbus. • Could potentially threaten the operation of a ICS/SCADA system. • ZMap has an almost identical capability but can scan Large Area Networks.
Nessus	• Commercial, Active • Working on a "policy" framework, Nessus allows users to conduct host discovery and vulnerability analysis in a similar way to NMap, again using ICMP, TCP and ARP scanning. • Unlike NMap, Nessus has the capability actively probe each service to report on potential vulnerabilities, which could cause accidental DoS on SCADA systems.
Passive Vulnerability Scanner	• Commercial, Passive • Uses interface packet sniffing to dissect and analyse the data being sent over the network in order to gain information about the assets and services being deployed. • Although it does not require any form of direct probing with nodes, PVS must be continuously ran in order to gain a better understanding of the network it is monitoring. • It is not intrusive, but the time it takes to analyse traffic is significantly higher than the active alternatives.
Shodan	• Open Source/Membership Based, Active • Uses similar techniques to NMap, ZMap and Nessus to find the services that are running on internet-facing devices. • All results are then stored in a database for users of the Shodan search engine to query against. • As this tool uses the same technology as other active scanners, it too poses the risk of affecting ICS/SCADA systems, especially as it has the capability to scan globally, meaning any CNI running legacy software could be at a significant risk. • Shodan has the potential to bring unwanted malicious attention to ICS/SCADA networks through the storing and reporting of information about ICS infrastructures.

Appendix B SCADA Network Diagram

Figure 1: A network diagram showing the link between corporate and SCADA networks.



Appendix C SCADA Network and Protocols Report

This document discusses the different types of protocols used on both SCADA and IP networks and provides a breakdown of each of the relevant technologies to evaluate how network scanners may behave when executed against the different types of network.

C.1 Introduction

Understanding the fundamental differences between non-IP and IP protocols is essential when analysing the application of network scanners on SCADA systems. Dissecting a range of non-IP protocols will give an insight into not only how SCADA devices communicate with each other, it will also help develop an understanding as to how the devices interpret the data once it has received each message, and how this could lead to potential issues when faced with unfamiliar packets supplied by scanners. Drawing internal comparisons between non-IP protocols as well as IP protocols will assist in identifying which communications systems are more unstable when being sniffed or scanned. This will also help identify possible weak points in common SCADA networks and what measures should be taken in order to not disrupt the devices communicating with that specific protocol.

As there are a large number of devices that could be used by a SCADA or ICS system, the most commonly used protocols will be identified and dissected, as to give a broad representation of the common attributes of non-IP communications mechanisms. Using the information provided throughout the Literature Review, the following protocols seem to be the most prevalent within ICS/SCADA environments:

- Ethernet
- Modbus
- Distributed Network Protocol version 3 (DNP3)

Each of these protocols is used by a range of PLC's and RTU's, the most significant devices in question, in ICS and SCADA systems around the globe. Once a breakdown and review of each of these protocols has been given, their structure and function will then be compared against each other to identify the differences between them, and how this may be impacted by network scanning tools.

C.2 Ethernet

It is essential to identify the two significant methods of data transfer between devices on SCADA systems: Ethernet and Serial connections. These two connection types reside on the Data Link Layer of the OSI Model. The Data Link Layer is an interface between the Physical and Network layers, segmenting the data provided by the upper layers into frames which can then be handled by the communications hardware of the lower layer. Frames have several purposes, including providing synchronisation, parity and data bits which indicates to the receiver the beginning and end of packets, as well as parity-checking and sequencing. An example of an Ethernet frame is as follows:

Preamble + SFD	Dest MAC	Src MAC	Eth Type	Data	CRC
8 Bytes	6 Bytes	6 Bytes	2 Bytes	46 - 1500 Bytes	4 Bytes

Table 4: A table showing the distinct frames of a single Ethernet packet.

C.2.1 Preamble + Start Frame Delimiter

The preamble and delimiter frame consists of 8 bytes at the beginning of each Ethernet packet. The preamble always precedes the start frame delimiter as it is used as both a "ready-up" notification to alert the interface that data is arriving, it also acts as a clock-synchronisation device (Mackenzie, n.d.). A valid preamble message is 7 bytes in length, containing alternating binary values. This allows for the receiver's clock to match the pace of the data being transmitted. The choice to alternate the binary values across 7 bytes was to give adequate time for the receiving device to set its clock and synchronise. An example of the binary data within a preamble is as follows:

10101010 10101010 10101010 10101010 10101010 10101010 10101010

Here, the interchanging 1's and 0's are easy to detect, especially if the receiver's clock is old or slow. An alternate preamble value such as *11111111* or *111101110* could be misinterpreted (IEEE Std 802.3, 2012).

The start frame delimiter is a single byte of data which acts as a flag to symbolise where the synchronisation section ends and where the Ethernet header begins and thus the beginning of the Ethernet frame itself. The pattern of bits is used to break the pattern present within the preamble, this is often represented as the decimal value 171 (*0101011*). This then completes the 8-byte frame at the beginning of each packet (IEEE Std 802.3, 2012).

C.2.2 Destination MAC Address

The destination MAC address (Media Access Control address) makes up one half of the Ethernet header. A MAC address is a unique identifier given to the physical Network Interface Card (NIC) present within each device on a network. The addresses are often associated with IEEE 802 technologies, Ethernet and Wi-Fi. Each MAC address is 6 bytes in length, on which the address is split into 3 byte segments which give small amounts of detail about the NIC itself:

3 Bytes	3 Bytes
Identifier of the Adapter's Manufacturer: Organisationally Unique Identifier (OUI)	Unique Network Interface Card Identifier

Table 5: A table explaining the data present within the MAC address fields.

The first 3 bytes represent the Manufacturer ID, or the Vendor ID, who created the adapter/interface card. For example, hex value 0x3C-D9-2B represents NETGEAR devices. 0xCC-46-D6 represents Cisco Systems devices. The latter 3 bytes are a unique ID given to that specific device. This completes the physical address format

used within the Data Link Layer (Ethernet (IEEE 802.3), n.d.).

The destination MAC address specifies the address of the receiver's NIC which the data is to be sent to. As these physical addresses cannot be used to communicate with networks outside of a LAN setup, these addresses work alongside the IP system to create the Address Resolution Protocol (ARP). This protocol allows for the physical MAC addresses to be associated to an IP address which is then used as an addressing system which can span across the internet. This protocol is the fundamental difference between IP and non-IP networks.

C.2.3 Source MAC Address

With the identical specifications of the destination MAC address, the source MAC completes the Ethernet header. Sending the source address allows for the recipient of the data to update their ARP cache, a list of MAC addresses with their resolved IP addresses, so that each machine can remember which IP is associated with each MAC address without having to resolve it each time a packet is received (Ethernet (IEEE 802.3), n.d.).

C.2.4 EtherType

The EtherType field serves two purposes. The first is a single byte value which defines which protocol is being encapsulated within the "data" frame of the packet. For example, 0x0800 is the value of IPv4, 0x8903 represents Cisco's Data Circuit-Terminating Equipment. It can also be used to represent the size of the "data" frame in bytes (Mackenzie, n.d.). This difference is dictated by whether the Ethernet type being used is Ethernet II or 802.3.

C.2.5 Data

This represents the user's data which is being carried by the Ethernet packet and what is being transmitted across the network. This data normally consists of protocol information from the OSI layer above. This could consist of a IP and TCP packet. The data section can be a minimum of 46 bytes and maximum of 1500 bytes, which will be dictated by the protocols and data being used by the higher layers of the OSI model (IEEE Std 802.3, 2012).

C.2.6 Cyclic Redundancy Check

A cyclic redundancy check is used in an Ethernet packet as an error-detection mechanism to ensure that no changes have been made to the raw data whilst in-transit. Ethernet uses a CRC-32 polynomial algorithm which calculates a hexadecimal value based on a polynomial division of all the bits within the data frame (Mackenzie, n.d.). The remainder from the division algorithm is then used to represent a checksum of the data. If the checksum changes between the sender and the receiver, an error has occurred within the data whilst in transit.

C.2.7 Conclusion

It can be recognised that with the use of MAC addressing to support ARP and the non-persistent size of the data frame, Ethernet communications suit the technologies which communicate over a vast number of networks. With the EtherType frame detailing information about embedded protocols, the focus seems to be on addressing, ensuring point to point communication is achieved correctly and efficiently.

C.3 Serial

Serial communication is a method of transmitting data between a computing device and a peripheral device in a master/slave command and response setup. Data is sent one byte at a time in a chronological sequence using a singular communication line to a receiver. As opposed to parallel communications, which use multiple wires to send and receive 8 bytes at a time, serial only requires one wire for data transmission. This could seem inefficient or slow when comparing serial to the parallel alternative (Serial Communication, n.d.). However, the choice to use serial communication is to facilitate ease of configuration and supports data transfer over a longer physical distance (something that is essential when implementing field devices on a SCADA system).

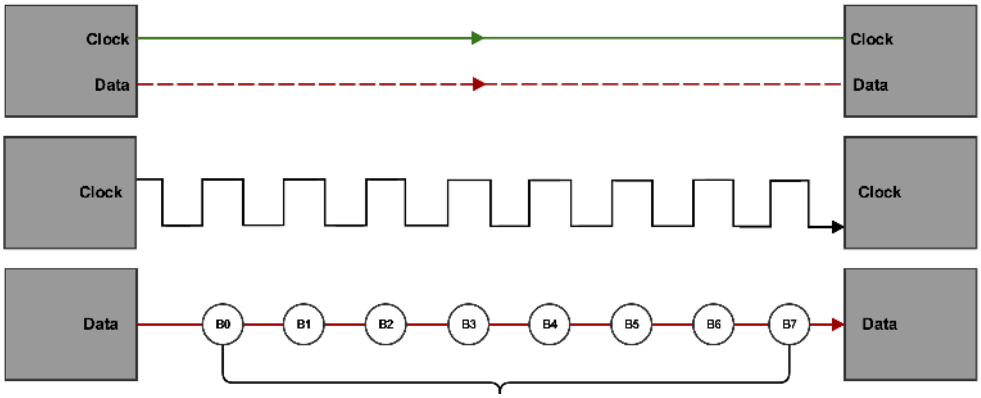


Figure 2: A figure showing 8 bits being transmitted with every pulse of a system clock.

Here is an example of a serial interface where 1 bit is sent with every pulse of the system clock, allowing the receiving device to begin parsing and using each bite of data as it come through. Serial communications can be sent via two different methods: Synchronous or Asynchronous. The methods discussed below focus on synchronous communication.

C.3.1 Synchronous

A synchronous communication channel relies on both the sending and receiving devices having their internal clocks synchronised so that the signal being transmitted can be sent and received as a long stream of data (Serial Communication, n.d.). There are no gaps between each segment of data, so the recipient processes each byte being sent in real-time. This type of communication is often used within master/slave network configurations. As there are no gaps between the bits being sent, the rate of transfer is often faster as no additional data is needed for sequencing or framing. This type of network model both reliable and efficient when sending large

sums of data (Serial Communication, n.d.). As the traffic is sent as a continuous stream, the only two physical mediums needed for this type of connection are a wire for the data and a separate wire for the clock signals, meaning more resources are needed compared to the asynchronous alternatives.

C.3.2 Asynchronous

The main difference between asynchronous and synchronous communication is the use of framing and the dependency of a clock signal across the network (Embedded Systems Course- Module7: SERIAL COMMUNICATION, 2016). In order to send data from one machine to another, the two devices must agree on a transfer speed before any official payloads are sent. Once a transfer speed has been negotiated, the data is grouped into 1 byte frames which are encased within two extra bits, a start bit and a start bit. These act as delimiters and parity bits which tell the receiver how to interpret the data, and clearly signifies where one section of data begins and ends (Serial Communication, n.d.). As it does not require a synchronised clock, there only needs to be a single line of communication. Having additional bits and using breaks in transmission to indicate breaks in data, this method is slower than synchronous communications in some cases, however this means that asynchronous is cheaper and easier to implement. All protocols discussed in this document (Modbus and DNP3) are all examples of asynchronous transmission methods. This section aims to break-down each of the different serial protocols used on non-IP networks in order to gain an understanding how the different technologies compliment and oppose each other, and to highlight the main significance of using Serial as opposed to Ethernet.

C.4 Modbus

Modbus, specifically Modbus RTU, is an 8-bit asynchronous serial communication protocol which supports the transmission of data through either buses or a network using a master/slave model. The most significant difference between the Modbus RTU protocol and more modern Internet-based protocols is the use of an Ethernet frame on the physical layer of the OSI network model (Simply Modbus, 2015). Modbus is a Layer 7 protocol as it sends the raw data straight to the recipient to then be processed as soon as it is received. Although Modbus does not used framing to help dictate the control and flow of data, it uses delimiters to specify the beginning of a new message. The RTU variant of Modbus uses time gaps of silence between communications in order to group data (Modicon Modbus Protocol Reference Guide, 1996). Each time a gap is detected by the receiving device, the receive buffer is cleared in order to prepare for the new message. The stack of a Modbus message is as follows:

Start	Address	Function	Data	CRC	End
3.5 char time gap	8 Bits	8 Bits	N * 8 Bits	16 Bits	3.5 char time gap

Table 6: A table showing a breakdown of a single Modbus packet.

C.4.1 Start + End 3.5 Char Time Gap

As mentioned earlier, the Modbus RTU protocol uses a break in time in order to delimit each individual packet of data. The 3.5t directly relates to the baud rate (transmission speed) that has been agreed between the sending and receiving devices. 3.5t is the time it takes to receive 3.5 characters from across the network (Modicon Modbus Protocol Reference Guide, 1996). This means that if there is a message which has a time break longer or shorter than 3.5 chars, the message should be dropped or ignored by the receiver. The main reason for using

this is to encapsulate each packet so that it has an easily identifiable start and end point (MODBUS over Serial Line, 2006).

C.4.2 Addresses

The address field is an 8-bit value which must consist of hexadecimal values 0-9, A-F (Simply Modbus, 2015). If a machine on the network receives a packet, it will look at the address field in order to distinguish whether the message for that particular machine. If the addresses match, the machine continues to decode the rest of the message, if not, the message is dropped. When the receiver/slave device responds to a message, it places its own address in this field to let the sender/master know which slave is responding. Address 0 is reserved for the broadcast address, which all slave devices recognise (Modicon Modbus Protocol Reference Guide, 1996).

C.4.3 Function

This field is used in order to tell the slave/receiving device what type of action it needs to perform. Function codes can span from 1-255, although some addresses have been reserved for future use or adaptation of the protocol. The following table shows all the possible function codes which can be sent:

Code	Function
01	Read Coil Status
02	Read Digital Input Status
03	Read Holding Registers
04	Read Input Registers
05	Force Single Coil
06	Preset Single Register
08	Reset Slave
15	Force Multiple Coils
16	Preset Multiple Registers
17	Report Slave ID

Table 7: A table showing the Modbus function codes and their corresponding function.

If the slave machine receives a valid message containing a correct function code, it will send a response message to the master device (Modicon Modbus Protocol Reference Guide, 1996). The slave uses the function code segment of the response message to tell the master device whether an error has occurred or whether the command has been accepted with no issues. The typical return value for an error is 0x83.

C.4.4 Data

This message field holds additional information which supports the function sent in the previous frame. This data provides more information in regards to the action the master requires the slave to fulfil. When using

RTU, the data field contains multiple 2-digit hex numbers which form such values as register addresses, count values and quantity of items to be handled. Some messages may have a data field of size zero as some functions do not require data. The slave follows the same error checking/reporting process used in the function code segment (Simply Modbus, Exception Codes, 2015). If no errors occur the response data is sent back to the master, if not, an exception code is sent within the data field, some of these exceptions are detailed below. The byte-count of a Modbus frame allows for a maximum of 252 bytes.

Code	Name	Explanation
01	Illegal Function	The function code sent to the slave in the previous message is not allowable/cannot be executed by the slave device.
02	Illegal Data Address	The data address received by the slave is not accepted by the slave device.
03	Illegal Data Value	A value which is present within the data field of the message is not compatible with the slave device.
04	Slave Device Failure	An error occurred when the slaved tried to execute the action requested by the master device and could not be resolved.
05	Slave Acknowledge	The slave has accepted the request, but the time it will take to process the request will take a significant amount of time. (Sent to prevent timeouts).
06	Slave Device Busy	The slave is processing a resource-intensive command.
07	Slave Negative Acknowledge	The slave failed to perform the process requested by the master machine. A diagnosis error will be sent back to the master.
08	Parity Error	The slave machine found an error in the parity when trying to read memory.
0A	Gateway Path Unavailable	The gateway on the network could not establish a connection. The gateway could possibly be misconfigured or overloaded.
0B	Gateway Device Unresponsive	The target device cannot be found on the network/is not present.

Table 8: A table showing the possible exceptions codes which can be sent from a Slave to a Master. Data within the table supported by (Simply Modbus, Exception Codes, 2015).

C.4.5 Cyclic Redundancy Check

A Cyclic Redundancy Check (CRC) is Modbus RTU's way of error checking each message in order to ensure that no data has been lost during its transit over the network. The CRC is calculated using the entire contents of the message and is sent with each packet of data (Modbus TCP/IP Unplugged An introduction to Modbus TCP/IP, n.d.). The CRC is then calculated by the slaves on receiving. If the CRC matches the once received, processing continues as normal. If the two values do not match an exception error is sent back within the data field (MODBUS over Serial Line, 2006).

C.4.6 Conclusion

Modbus has been optimised for sending data in a single stream. The use of timing pauses to act as delimiters and the simplicity of the address format and data frames means that it is easy to apply to master/slave systems. The most significant aspect of Modbus is that it is a Layer 7 (application layer) protocol, meaning that this method of communication is not concerned with complex routing, encryption or packet sequencing; The goal is to allow data to be sent to a slave, processed and then a responded to in a reliable manor. This is where the most evident differences between traditional IP protocols are shown. The peer-to-peer nature of Modbus means that there is a significant lack of data compared to the IP alternatives.

C.5 DNP3

Like Modbus, DNP3 is a communications protocol which functions using the master/slave methodology. The master device sends a read request to the slave devices which then respond if, and only if, the data being requested is available. The master sends a request, the slave responds with an acknowledgement and then the data being requested. This is then followed by another acknowledgement from the master to signify it has received the data. The following tables show the data contained within DNP3 packets, and the process behind gaining information from remote devices:

Header	Data
10 Bytes	0 - 282 Bytes

Table 9: The two main sections of a DNP3 packet.

Start/Sync	Length	Control	Dest Address	Src Address	CRC
2 Bytes	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes

Table 10: A table showing a breakdown of the DNP3 header.

C.5.1 Start/Sync

This section of the data packet acts a starting delimiter which specifies that a new message is being transmitted to the slave device. The delimiter value for DNP3 conversations is 0x564. This opposes Modbus as it uses data to symbolise the breaks between individual messages being sent across the network (DNP 3.0 Remote Communication Protocol for REC 523, 2001).

C.5.2 Length

The length field specifies the number of bytes not only in the user/data frame, but also the combined size of source address, destination address and control byte.

C.5.3 Control

This frame defines the direction of the traffic, the type of frame being sent and the function code which tells the remote device what actions to perform (DNP 3.0 Remote Communication Protocol for REC 523, 2001), (DNP3 Quick Reference, 2002). Each of the 8 bits within the control frame has a specific purpose:

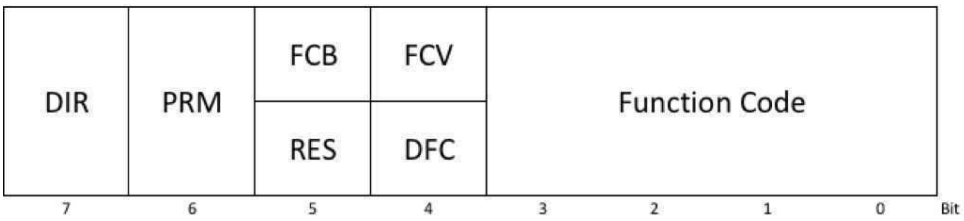


Figure 3: This figure shows the parameters which reside within the control section of the DNP3 frame.

DIR (Direction): This represents the physical direction of the transmission, 1 = Master → Slave, 0 = Slave → Master.

PRM (Primary Message): This bit defines whether the message has been sent from the initiating machine or the responding machine. 1 = Frame from initiating, 0 = frame from responding machine (primary/secondary).

FCB (Frame Count Bit): This bit is used in order to check that there has been no losses of data or duplicate packets. The bit is toggled each time a SEND-CONFIRM service is successful.

FCV (Frame Count Bit Valid): This frame validates the FCB function. If the bit is set to 0, the FCB can be ignored.

RES (Reserved): This frame is reserved for future use. It is always set to 0.

DFC (Data Flow Control Bit): This prevents buffer-overflows from occurring on the receiving (secondary) machine. The secondary machine will return a 1 if the SEND message has caused the data link buffers to overflow.

Function Code: This code identifies the type of data/actions being sent from the master (primary). The following values are valid DNP3 function codes:

Code (Hex)	Function
0	Reset Link States
2	Test Link States
3	Confirmed User Data
4	Unconfirmed User Data
9	Request Link Status

Table 11: DNP3 function codes when PRM = 1

Code (Hex)	Function
0	Acknowledge
1	Negative Acknowledge
B	Link Status
F	Not Supported

Table 12: DNP3 function codes when PRM = 0

C.5.4 Destination Address

The address of the slave device on which the message is being sent to. Similar to Modbus, the address is represented as a hex value, however, the hex string within the DNP3 protocol is twice the length of the Modbus addressing system, meaning that a larger number of secondary devices can reside on the same network.

C.5.5 Source Address

The address of the master device on which the message is being sent from. This is one of the most significant differences between Modbus and DNP3, the acknowledgement of the unique ID of the Master device. This means that DNP3 configured networks can host communications with multiple Master devices. This makes DNP3 a more versatile protocol when trying to remotely control or monitor field devices from multiple sites or vast networks.

C.5.6 Cyclic Redundancy Check

The CRC appended to the end of the header is calculated using all the previous frames discussed so far. Unlike the addressing system used by Modbus, DNP3 has multiple parameters such as the DIR, PRM and function code which must be valid in order for the DNP3 devices to interpret the data. This CRC specifically focusses with irregularities or data loss within the header. This means that the receiving device does not have to receive the whole message before determining whether data has been changed/lost on transit.

C.5.7 Data Field

This frame contains the user data which dictates the action a slave device must carry out. If the master device has requested data from the slave, the data frame of the response packet will contain this information (DNP 3.0

Remote Communication Protocol for REC 523, 2001). Unlike Modbus, the data of a DNP3 packet is separated into data blocks, which can range between 1 and 16 bytes depending on the type of packet being transmitted, demonstrating a method of fragmentation. After each 16-byte segment, a data CRC is then calculated and appended to the end of each block. This is how DNP3 calculates whether any data has been lost or duplicated during its transit across the network.

C.5.8 Conclusion

DNP3 has a very similar mode of operation as Modbus, as it is another asynchronous serial protocol which sends a continuous stream of data to slave devices in order to achieve remote control and monitoring. Like Ethernet, DNP3 is classified as a Data Link Layer protocol as it incorporates addressing services, data fragmentation and block-based error checking. It also provides an Application Layer interface which can be used by SCADA devices such as Remote Terminal Units. Again where this protocol draws the most significant parallels from the other protocols is the networks it has been designed to run on. Although DNP3 uses a more advanced form of addressing allowing multiple master devices to query against the slave devices present on the network, it does not support any form of encapsulation, suggesting that, like Modbus, DNP3 has been designed to only span across one network without routing capabilities.

C.6 Overall Conclusion

When comparing each technology against one another, it is apparent that because of the significant differences between how Ethernet and the two serial protocols first synchronise the data, delimit the data and frame the data, that trying to transmit a foreign protocol through any of these mediums could have negative consequences. For example, as Ethernet uses a large 8-byte preamble in order to synchronise the sending and receiving clocks, if this same data was transmitted and received on a serial network supporting Modbus, the incoming data would be too large for the recipient device to process. This in turn could cause the CPU of the recipient device to hang as it attempts to process the Ethernet data. This would therefore limit the amount of CPU resources being allocated to the other functions of the device, causing a halt in operation or change in behaviour.

The second notable outcome from this research is that if a network scanning tool uses Ethernet frames to send out packets, and the SCADA/ICS network is running a serial protocol such as DNP3, the data being transmitted by the scanning tool may not be able to travel through that particular medium (the example in this case is serial RS232 rather than CAT5 Ethernet cable). Although this may not have an impact on the devices themselves, the scan will return with no results. If used incorrectly, executing IP scanners on these networks could either cause a denial of service, connection disruption or false negative scan results.

The information gathered from this research and analysis will help aid the understanding of how IP and SCADA networks behave when subject to a network scan.

Appendix D Testing of Network Scanners Against IP Devices

This document outlines the experiments conducted against a virtual IP network in order to observe and analyse how passive and active network scanners function. This then provides a platform for discussion about the feasibility of using the same tools against a SCADA network.

D.1 Objectives

In order to assess how active and passive network scanners perform on traditional IP-based networks, a virtual IP environment has been constructed which will allow for the examination and analysis of the packets generated by each scanner. The aims of these experiments are to simulate the activity of a small physical network and to record the status of the network once a scan has been initiated. Once all the necessary scans have been conducted, the captured data will be pulled from the virtual network to be analysed in detail.

D.2 Hypotheses

The following hypotheses will be repeatedly tested throughout all the experiments referenced within this document.

Operational Hypothesis: H_1
The use of current active or passive IP network probers and sniffers will have no effect on the normal behaviour of a non-IP network.

Null Hypothesis: H_0
The use of current active or passive IP network probers and sniffers will have a negative effect on the normal behaviour of a non-IP network.

D.3 Variables

Independent Variable: The execution of a network scanner/sniffer on a network.
Dependent Variable: The behaviour of the IP scanners/sniffers.

D.4 Requirements

This section outlines the tools and resources needed in order to successfully conduct each experiment.

D.4.1 Ubuntu 16.04 LTS Virtual Machine

The first stage of constructing the virtual network was choosing the host operating system (OS). In order to facilitate the running requirements of Netkit, the operating system had to have a Linux kernel. As a wide variety of the network scanning tools being tested are fully compatible with Linux operating systems, Ubuntu 16.04 satisfied the requirements of this experiment. The choice to run version 16.04 LTS was simply because of the stability of the OS and the long-term support provided by Ubuntu. The most recent version may have replaced some of the key features which Netkit requires in order to run. Running LTS instead of the latest

version also ensured that no updates which may be required during the course of the experimentation would have any unprecedented effects on the experiment itself.

In order to ensure that each experiment could be easily repeated without having to reconfigure the virtual network or operating system each time a test has been ran, the most effective solution was to use the VMware Fusion hypervisor to run a copy of Ubuntu 16.04 LTS as a virtual machine. Choosing this method ensures the following capabilities:

- It allows for "snapshots" to be taken before any experiment takes place. Once a snapshot has been taken, the virtual machine can revert back to state which was captured within that snapshot. This in turn will ensure that each experiment will be executed with exactly the same conditions, meaning that there should be no external influence on any of the variables.
- Running a virtual machine allows for the internal operating system to be completely isolated from not only the machine it is running on, but also the Local Area Network (LAN) which the host machine resides on. Again this will ensure that no unexpected or external influences affect the results yielded from any of the experiments.
- The final benefit of using a virtual machine is the ability to share the test environment with other machines if desired. The ability to clone an image which contains the configuration of the test environment will be highly beneficial to anyone who wishes to repeat the experiments conducted throughout this project.

Using a virtual machine provides practicality, integrity and reliability when conducting tests or experiments. As the OS is Linux based, it provides the ideal environment for the Netkit setup.

D.4.2 Netkit 2.8

Netkit is a lightweight solution to network simulation, allowing users to create a multitude of vertical machines which can emulate the functionality of routers, switches, servers and computers. Each virtual machine is based off the User-Mode Linux kernel (UML), an open-source project which provides a Linux kernel which is ran as a process within the user space of a system. This means that the functionality of a lightweight Linux system can be ran completely as a process, providing some isolation and minimal interaction with the host system. Each UML virtual machine has virtual resources such as network interface cards and disk storage, however, the configuration and mounting point for each machine is contained within a singular file on the host machine, making it easy to eliminate and create new machines as and when they are needed.

The files required in order to setup Netkit on the host machine can be obtained from `http://wiki.netkit.org/index.php/Download_Official`. The version being ran within this project is Netkit 2.8.

D.4.2.1 Installation Following the installation guide provided at `http://wiki.netkit.org/download/netkit/INSTALL`, the Netkit kernel and configuration files were unpacked into a Netkit directory on the Ubuntu host OS. Once installed and unpacked the "./check_configuration.sh" script was ran in order to ensure that all the correct dependencies had been met. On successful configuration, Netkit was ready to host a virtual network.

D.5 Design and Setup

The design of the Netkit environment aimed to satisfy two criteria: To offer similar services found on a common modern IP network and to provide a platform which will allow network sniffers and scanners to be executed, captured and analyses in order to fully understand how they function. To do this, a network diagram had to be created in order to plot the different subnets of the network and where different machines were going to reside. The following diagram shows the design of the virtual network coupled with an explanation behind the placement of machines and the services being ran.

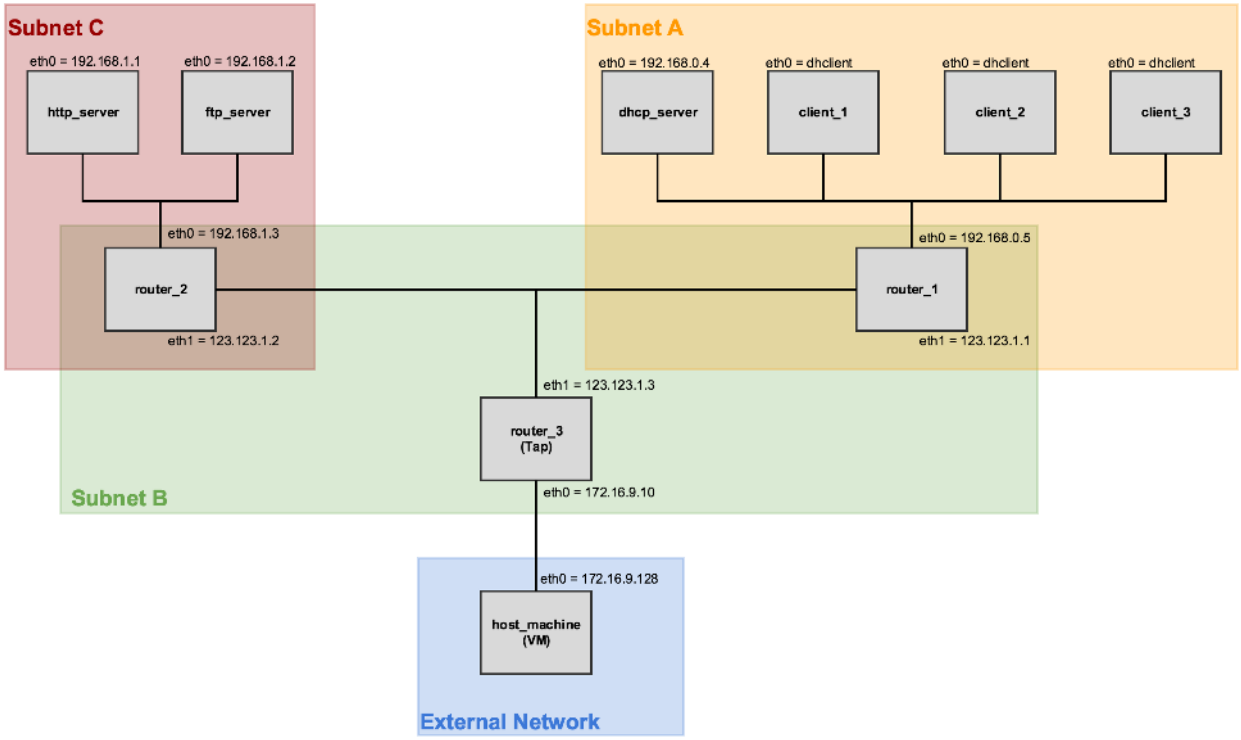


Figure 4: A network topology of the virtual IP environment

D.5.1 Clients

Each client has been setup to mimic a typical user machine on the network. On start-up, each client requests an IP address from the DHCP server present within the same subnet. During the course of the experiments, the clients will be used to not only request IP addresses, but also to fetch webpages from the HTTP server and to transfer files to and from the FTP server. The clients have been given dynamic IP addresses to show the functionality of DHCP being emulated during the experiments as they do not require a permanent static address. A packet capture will be initiated on machine "client_1" in order to represent how network scanners gain information about client machines.

D.5.2 DHCP Server

The Dynamic Host Control Protocol Server simulates the dynamic allocation of IP addresses to the client machines on the virtual network. The original design of the network had the DHCP server located on a separate subnet to the client machines. After running several tests throughout the configuration phase of the

virtual network, it was both impractical and unnecessarily complex to try and facilitate that design, therefore the server was relocated onto the same subnet as the client machines. A packet capture will be initiated on this machine in order to show how a network scan interacts with this type of network device.

D.5.3 HTTP Server

The main purpose of this machine is to distribute simple HTML webpages to a client machine when it sends a HTTP "GET" request on port 80. The webpage does not have to be complex in structure, as long as the process of requesting and distributing web services is adequately emulated. Each client will make a request to the server using a text-based web browser named Links, which comes pre-installed on all the Netkit virtual machines. This machine resides on a different subnet to the clients and DHCP server. This was done intentionally in order to clearly distinguish the configuration of the server machines and also the traffic being sent from this machine, which will become useful within the packet analysis phase. This machine will also perform a packet capture. This is so that the interaction between network scanners and web servers can be monitored and analysed.

D.5.4 FTP Server

The File Transfer Protocol server will be used to demonstrate the transfer of files between the client machines and the FTP server. On conducting the experimentation, each client will login to the FTP server, send a basic .txt file to the server, pull that same file from the server and then issue a "delete" command before terminating the session. The FTP server will be located on the same subnet as the HTTP server for the same reasons as stated before in this document. Similar to the previous server, this machine will also perform a packet capture.

D.5.5 Routers

Two routers will be configured to act as both a gateway for each subnet as well as a routing device. The two routers located between subnet A/B and C/B will also be used to perform the packet captures necessary for the experiments. The reason for this is so that firstly, the packets being captured from the machines themselves can be cross-examined with the packets sent across the routers, and secondly, so the packets being sent and received across the entirety of each subnet can be analysed once it has been interrogated by a network scanner.

D.5.6 Tap Machine

The Tap machine allows for the virtual machines to access and be accessed by devices which are not present on the same virtual network. Configuring a Tap will allow for the host machine (Ubuntu 16.04 virtual machine) to execute network scanning tools against the virtual network and will allow for the instrumentation and acquisition of the subsequent network data. This machine will not perform any form of packet capture as it does not provide a realistic emulation of a device present on a traditional IP network. It could be argued that the tap machine could be configured to serve the same purpose as a Network Address Translation device or Firewall. Although this is achievable, it would be an unnecessary requirement and would not deliver any more benefit to the experiments.

D.5.7 Host Machine

This machine represents the Ubuntu virtual machine which the Netkit environment will be hosted on. Using the Tap machine as a gateway between the Host and the virtual network, each of the network scanning tools will be downloaded and executed from the host machine. On execution of each tool, the host will be disconnected from any other external network that may be present, meaning that the only addresses being targeted by each scan are those held within the virtual network. Having this external machine host and deploy the scanning tools ensures that the tools are executed and configured with the correct dependencies, as well as not causing the status or configuration of the Netkit machines to change during each experiment. Lastly, the choice to perform the scans using an external machine was to ensure that a range of tools could be tested against the virtual network, without the concerns of non-compatibility with Netkit's UML virtual machines.

The host machine required some altering in order for it to detect and communicate with the virtual network. This was achieved by setting the tap machine to route traffic to and from the virtual network. The Tapped address was set to the address of the Ubuntu virtual machine, and the guest address (the address given to the tap machine on the external network) was set to a new address which falls under the same subnet as the host machine. To do this, a routing command was used to alter the IP routing table on the host machine. Appendix F shows a bash script which, when executed, adds the following elements to the host's IP routing tables:

1. The machine 172.16.9.10 is reachable through interface nk_tap_kyle
2. Added a route to 192.168.0.0/24 through gateway 172.16.9.10
3. Added a route to 192.168.1.0/24 through gateway 172.16.9.10

The first element signifies that the external facing NIC of the tap machine can be accessed through the interface nk_tap_kyle (the interface which automatically appears on the host machine when a tap is created). The second section makes reference to the subnet 192.168.0.0 within the virtual network, and that it can be reached through the tap machine. The third command serves the same purpose as the second except with the 192.168.0.1 subnet. Each time the virtual network is launched, the bash script must be executed in order to ensure the host machine can reach each of the virtual machines.

D.6 Obtaining the Results

To enable the Ubuntu virtual machine to be reverted to the controlled state before any scans took place, a snapshot was taken of the virtual machine. Once the snapshot had been taken, it was then loaded in order to test the snapshot had been saved successfully. On loading the snapshot, the virtual network was started using the "lstart" command. As specified the DHCP server loaded before any of the client machines, and the tap machine requested a password to be entered. Once all the machines had started-up correctly, each terminal (used to interact with each virtual machine) was rearranged so that it resembled the network diagram provided earlier in this document (Figure 4). In order to capture the packets being sent by Nmap to a variety of the different machines, the following table shows the virtual machines which performed a "Tcpdump" network capture in order to monitor the packets travelling across the network:

Machine	Address	Address Type
Client_1	192.168.0.10	Dynamic
DHCP_Server	192.168.0.4	Static
Router_1	192.168.0.5	Static
Router_2	192.168.1.3	Static
HTTP_Server	192.168.1.2	Static
FTP_Server	192.168.1.1	Static

Table 13: A table showing the virtual machines and their IP addresses

Each Tcpdump capture was configured to listen on all interfaces, and saved its output as a .pcap file within the /hosthome/pcaps directory. This is a directory which allows each virtual machine to access the host machine's file system.

D.7 Expected Outcomes

Given the configuration of the Netkit environment and the passive and active tools being used, the following assumptions can be made about the possible results being yielded from these experiments:

- Firstly, as both passive and active network scanners will be used within this experiment, it can be anticipated that, when executed on the virtual network, the passive tools will lack the capability to provide details about the specific services running on each machine without having to analyse the packet captures.
- The active tools used within these experiments will attempt to gain information from each host by repeatedly sending packets at every port of every virtual machine until it finds a port which is open and will establish a connection. Once an open port has been identified, data will be sent to this port in order to gain information about the services being sent through it.
- The active tools will be deployed from the host machine which communicates with the virtual network via a tap/gateway. This suggests that the functionality of all the virtual machines within the Netkit environment will remain persistent throughout the duration of all the experiments. Furthermore, as the virtual network emulates an Ethernet/IP network, the tools being executed on the virtual network should have no effect on the configuration or overall behaviour of each endpoint/node.
- As the active scanners use protocols found within layers 3-7 of the OSI network model, the types of information being received by each machine on the network will not be suitable when applied to SCADA devices which run using non-IP protocols, such as ICMP and TCP.
- Using passive network sniffers will not use intrusive techniques in order to gain intelligence about the status of a network and the machines present within it. Instead, the passive tools will capture traffic and output the data into a form which can then be later analysed.

The expected results will be taken into consideration when discussing the results of these experiments.

D.8 Execution of Experiments: Active Scanners

D.8.1 Experiment 1: Nmap 2.8

Nmap has been chosen as a result of the information presented by both researching the relevant literature and a review of the tool's capabilities and features. Nmap is widely renowned in the fields of penetration testing and network administration, and has the ability to conduct a range of different network scans, spanning from ping-sweeps to TCP FIN and fragmentation scanning. The popularity and versatility of this tool were significant factors which lead to the utilisation of some of its capabilities within this experiment. Being open source and cross-platform, it met all the criteria needed in order to run on the host machine. Nmap provides scanning techniques which are not only used within modern corporations, but also can replicate the type of scans used by black-hat hackers, state organisations and activists in order to conduct scans on a wide range of networks around the globe.

D.8.1.1 Nmap Ping Sweep The first scan executed against the virtual network aimed to address the issues with using the ICMP echo request utility "Ping" in order to discover devices at a particular address or a range of devices on a particular network. The first set of commands executed against the virtual network where as follows:

"nmap -sP -r 192.168.0.0-15 -e nk_tap_kyle > nmap1.txt"

"nmap -sP -r 192.168.1.0-6 -e nk_tap_kyle > nmap2.txt"

The "-sP" method of scanning sends ICMP echo requests ("pings")to the range of IP addresses specified within the command. In this case, all the machines on the 192.168.0.0/24 subnet will be scanned. The "-r" flag instructs Nmap to scan the ports consecutively. The "-e" flag dictates which interface on the host device should be used to send each packet through. In this case, the nk_tap_kyle interface represents the tap machine which acts as the gateway between the host machine and the virtual network. Lastly, the ">" operator sends all the output of the Nmap scan into a text file, the name of which represents each unique scan.

On executing this scan there were no failures either with the scans themselves of the machines being interrogated. Both "nmap1.txt" and "nmap2.txt" successfully captured the output of each of the ping scans over the two subnets. On further inspection of these files, the Nmap scan was able to identify that 5 machines resided on the 192.168.0.0/24 subnet, and that 3 devices resided on the 192.168.1.0/24 subnet. The following figure shows the successful detection of each of the virtual machines:

```
Starting Nmap 7.01 ( https://nmap.org ) at 2017-01-26 18:48 GMT
Nmap scan report for 192.168.0.4
Host is up (0.015s latency).
Nmap scan report for 192.168.0.5
Host is up (0.0019s latency).
Nmap scan report for 192.168.0.10
Host is up (0.019s latency).
Nmap scan report for 192.168.0.11
Host is up (0.013s latency).
Nmap scan report for 192.168.0.12
Host is up (0.014s latency).
Nmap done: 16 IP addresses (5 hosts up) scanned in 14.41 seconds
```

(a) Data obtained from "nmap1.txt"

```
Starting Nmap 7.01 ( https://nmap.org ) at 2017-01-26 18:50 GMT
Nmap scan report for 192.168.1.1
Host is up (0.013s latency).
Nmap scan report for 192.168.1.2
Host is up (0.013s latency).
Nmap scan report for 192.168.1.3
Host is up (0.013s latency).
Nmap done: 7 IP addresses (3 hosts up) scanned in 14.21 seconds
```

(b) Data obtained from "nmap1.txt"

Figure 5: The Nmap output from the ping sweep

These output files show that Nmap was able to detect each of the virtual machines which had been configured to run on the virtual network. Once it had been confirmed that the correct machines had been identified, some HTTP and FTP requests were sent across the virtual network to ensure that none of the services had been effected by the previous scan. The servers and clients were able to connect and communicate successfully. No data was provided to prove that the ping scan had caused any damage to the virtual machines.

In order to dissect and analyse how the ping scan achieves its results, the first pcaps taken on the devices specified within Table 13 were opened using Wireshark. The following figures show the results associated with each virtual machine:

- **Client_1**
- **Router_1**
- **HTTP_Server**

Client_1

On inspection of the Tcpdump taken from the client_1 machine, the pcap file does not show any ICMP packets being sent or received by the target machine. Using a display filter in Wireshark to isolate the ICMP packets yielded no results. This does not support the data held within the output file "nmap1.txt", as Nmap was able to find all the hosts which were active on both the 192.168.0.0/24 and 192.168.1.0/24 subnets. This suggests that the ping scan "-sP" does not use ICMP echo messages in order to detect active machines. After consulting the Nmap documentation as well as reviewing the client_1 pcap again, it appears that Nmap uses a form of "TCP ping", which sends either a TCP SYN or TCP ACK to a port on the target machine. In the capture file below, Nmap seems to use port 80 and port 443.

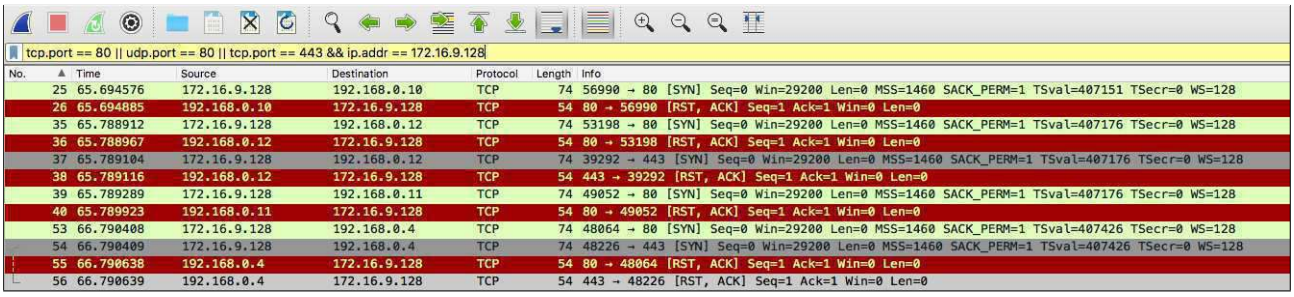


Figure 6: A pcap segment taken from the Client_1 machine

Here, in this section of the pcap taken from the "client_1" machine, it can be observed that the host machine (address 172.16.9.128) is sending TCP SYN packets to both ports 80 (HTTP) and 443 (SSL/TLS) in order to prompt either a SYN/ACK or RST/ACK response. If the target machine responds with either of these packets, Nmap can determine that there is a service behind those ports, and thus a host is active. This scan does not establish the full 3-way-handshake so therefore no data is transferred between both the host and the target machines. The above pcap shows 4 of the 5 addresses present on the 192.168.0.0/24 subnet. The machine which seems to be missing is the address of eth0 on router_1. Analysing the pcaps taken from the DHCP server as well as the gateway router shows no reference to TCP Pings being sent to the router machine, or any response being sent back to the host machine.

Router_1

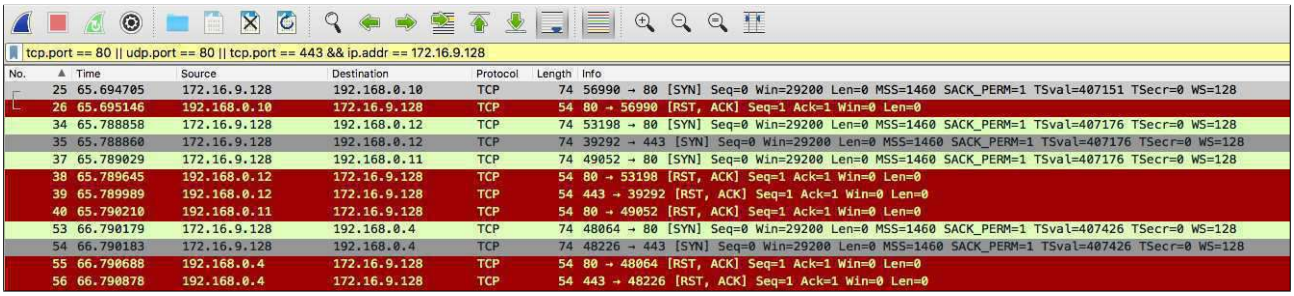


Figure 7: A pcap segment taken from the Router_1 machine

This figure shows the traffic captured from the router machine. This pcap still does not provide any data to show how Nmap was able to find this machine on the network. It does however confirm that Nmap is using TCP packets in order to detect remote hosts, rather than ICMP echo requests.

HTTP_Server

The same Nmap command was executed against the 192.168.1.0/24 subnet. The figure below shows that the same TCP Ping method being used in order to identify the HTTP server and FTP server on the subnet. Again, the two main nodes can be seen in the captured traffic, but there is no data to show how Nmap detected the router.

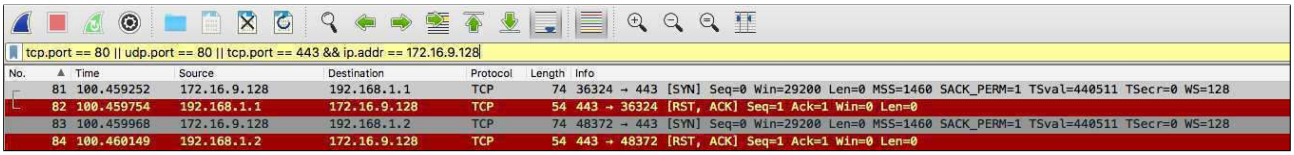


Figure 8: A pcap segment taken from the HTTP_Server machine

D.8.1.2 Nmap Service Detection Scan The purpose of the next set of scans was to replicate the act of detecting which services are being offered by each machine on the different subnets. The Nmap commands entered are as follows:

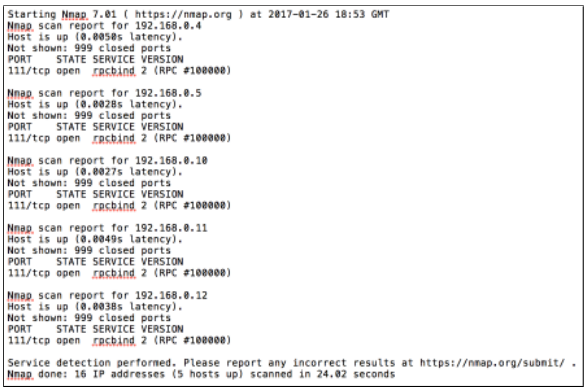
"nmap -sV -r 192.168.0.0-15 -e nk_tap_kyle > nmap3.txt"

"nmap -sV -r 192.168.1.0-6 -e nk_tap_kyle > nmap4.txt"

The purpose of these scans is to replicate the act of detecting which services are being offered by each machine on the different subnets. The first flag, "-sV" enables Nmap's version detection system. This should supply information about which ports are open on each machine, what services are running over that port and what version of that service is running. When used on networks spanning across the internet, this feature provides vital intelligence about the services being distributed across the target network, and whether there are any vulnerabilities associated with that particular service. As per the previous Nmap scan, "-r" is used to instruct Nmap to scan each port on the target machines consecutively, and "-e" directs the packets through the nk_tap_kyle interface. The output will also be stored within two text files which are associated with each unique command.

In order to analyse the traffic being produced by the Nmap scans, the same machines defined in the first experiment will run Tcpdump packet captures and write all the results to pcap files and saved on the host machine. Once the Tcpdump processes had started the scans were deployed.

Both commands executed without any fault, each successfully writing the output data into the text files defined in the commands. Once each scan had finished, the first element to be checked was the output files, this will confirm whether the scans have been successful in finding the correct machines and also the correct information is extracted. The following figures show that both of the commands were successful when executed against the two subnets:



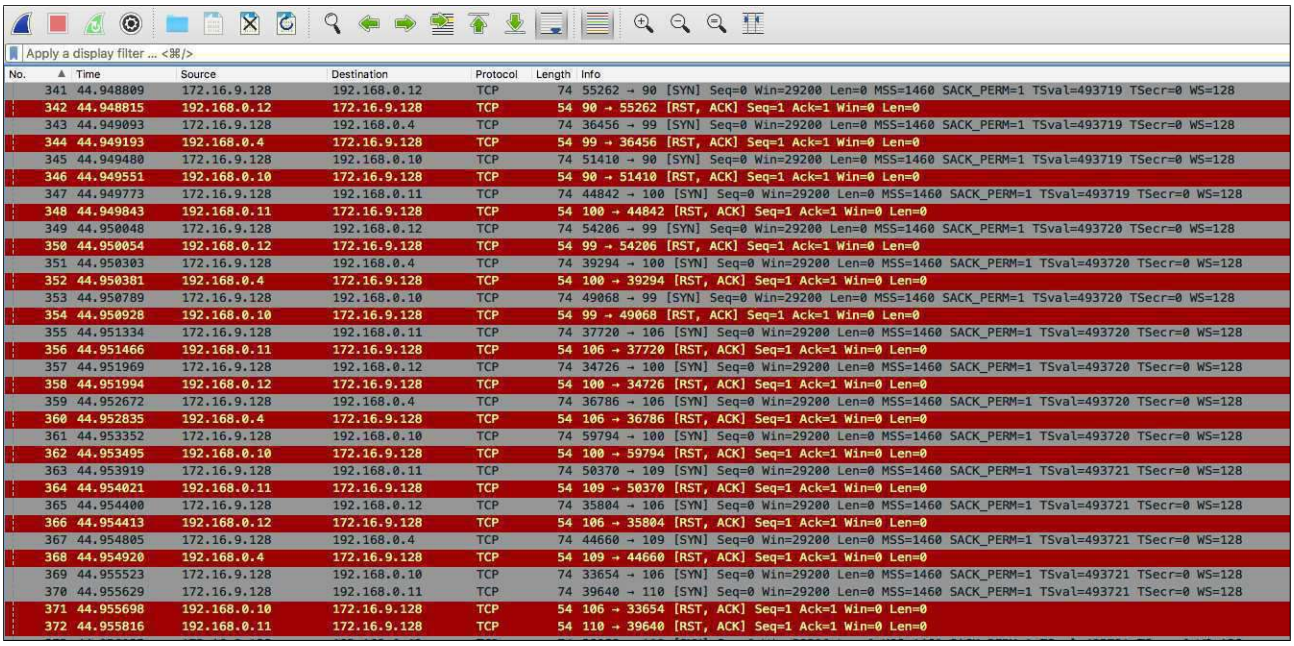
(a) The data obtained from "nmap3.txt"

(b) The data obtained from "nmap4.txt"

Figure 9: The results of the Nmap service detection scan

The two files show that, similar to the Ping scan, Nmap was able to detect each one of the machines connected to each subnet. As well as the host discovery, the service detection was able to identify the services which had been configured to run on certain machines. Both the FTP and the HTTP servers were both identified by the service detection system. However, looking through the output files, there is no data to suggest that the machine residing at address 192.168.0.4 is a DHCP server. As DHCP requires a client to broadcast a DISCOVER packet addressed to 255.255.255.255, the "-sV" flag is not equipped to detect whether a DHCP server exists on the network. As DHCP is not integral when comparing the results with SCADA networks, a lack of data regarding the DHCP server is not impacting on the results. After all the scans had been conducted, the client machines were used in order to perform some standard FTP and HTTP requests to check that the state of the network had not changed as a result of the scanning. The network continued to function as configured.

Once the output files had been checked, the pcap files captured from the virtual network were analysed in order to determine how Nmap extracts the version data from each machine.

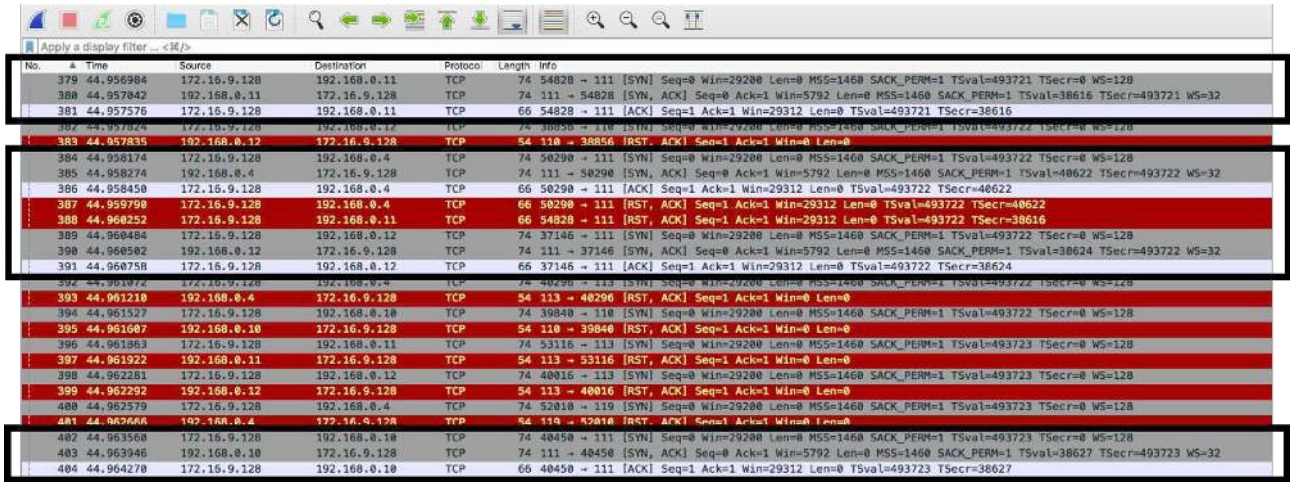


Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol	Length	Info
341	44.948809	172.16.9.128	192.168.0.12	TCP	74	55262 → 90 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493719 TSecr=0 WS=128
342	44.948815	192.168.0.12	172.16.9.128	TCP	54	90 → 55262 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
343	44.949093	172.16.9.128	192.168.0.4	TCP	74	36456 → 99 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493719 TSecr=0 WS=128
344	44.949193	192.168.0.4	172.16.9.128	TCP	54	99 → 36456 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
345	44.949480	172.16.9.128	192.168.0.10	TCP	74	51410 → 90 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493719 TSecr=0 WS=128
346	44.949551	192.168.0.10	172.16.9.128	TCP	54	90 → 51410 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
347	44.949773	172.16.9.128	192.168.0.11	TCP	74	44842 → 100 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493719 TSecr=0 WS=128
348	44.949843	192.168.0.11	172.16.9.128	TCP	54	100 → 44842 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
349	44.950048	172.16.9.128	192.168.0.12	TCP	74	54206 → 99 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493720 TSecr=0 WS=128
350	44.950054	192.168.0.12	172.16.9.128	TCP	54	99 → 54206 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
351	44.950303	172.16.9.128	192.168.0.4	TCP	74	39294 → 100 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493720 TSecr=0 WS=128
352	44.950381	192.168.0.4	172.16.9.128	TCP	54	100 → 39294 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
353	44.950789	172.16.9.128	192.168.0.10	TCP	74	49068 → 99 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493720 TSecr=0 WS=128
354	44.950928	192.168.0.10	172.16.9.128	TCP	54	99 → 49068 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
355	44.951334	172.16.9.128	192.168.0.11	TCP	74	37720 → 106 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493720 TSecr=0 WS=128
356	44.951466	192.168.0.11	172.16.9.128	TCP	54	106 → 37720 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
357	44.951969	172.16.9.128	192.168.0.12	TCP	74	34726 → 100 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493720 TSecr=0 WS=128
358	44.951994	192.168.0.12	172.16.9.128	TCP	54	100 → 34726 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
359	44.952672	172.16.9.128	192.168.0.4	TCP	74	36786 → 106 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493720 TSecr=0 WS=128
360	44.952835	192.168.0.4	172.16.9.128	TCP	54	106 → 36786 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
361	44.953352	172.16.9.128	192.168.0.10	TCP	74	59794 → 100 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493720 TSecr=0 WS=128
362	44.953495	192.168.0.10	172.16.9.128	TCP	54	100 → 59794 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
363	44.953919	172.16.9.128	192.168.0.11	TCP	74	50370 → 109 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493721 TSecr=0 WS=128
364	44.954021	192.168.0.11	172.16.9.128	TCP	54	109 → 50370 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
365	44.954400	172.16.9.128	192.168.0.12	TCP	74	35804 → 106 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493721 TSecr=0 WS=128
366	44.954413	192.168.0.12	172.16.9.128	TCP	54	106 → 35804 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
367	44.954805	172.16.9.128	192.168.0.4	TCP	74	44660 → 109 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493721 TSecr=0 WS=128
368	44.954920	192.168.0.4	172.16.9.128	TCP	54	109 → 44660 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
369	44.955523	172.16.9.128	192.168.0.10	TCP	74	33654 → 106 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493721 TSecr=0 WS=128
370	44.955629	172.16.9.128	192.168.0.11	TCP	74	39640 → 110 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493721 TSecr=0 WS=128
371	44.955698	192.168.0.10	172.16.9.128	TCP	54	106 → 33654 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
372	44.955816	192.168.0.11	172.16.9.128	TCP	54	110 → 39640 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 10: A pcap segment taken from the Client_2 machine

This pcap shows the host machine (172.16.9.128) sending a continuous stream of TCP SYN packets to all the devices present on the 192.168.0.0/24 subnet. This method of scan iterates sequentially through each of the ports on the virtual machines. The figure above demonstrates Nmap scanning each port in ascending order, using a TCP SYN packet to try and establish a connection with each one. The red TCP RST/ACK packages show that there is no service being hosted behind that port. This pcap represents a client machine, so there are no obvious services expected to be running on this machine.



Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol	Length	Info
379	44.956984	172.16.9.128	192.168.0.11	TCP	74	54828 → 111 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493721 TSecr=0 WS=128
380	44.957042	192.168.0.11	172.16.9.128	TCP	74	111 → 54828 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=38616 TSecr=493721 WS=32
381	44.957576	172.16.9.128	192.168.0.11	TCP	66	54828 → 111 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=493721 TSecr=38616
382	44.957824	172.16.9.128	192.168.0.12	TCP	74	38856 → 110 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493722 TSecr=0 WS=128
383	44.957835	192.168.0.12	172.16.9.128	TCP	54	110 → 38856 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
384	44.958174	172.16.9.128	192.168.0.4	TCP	74	50290 → 111 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493722 TSecr=0 WS=128
385	44.958274	192.168.0.4	172.16.9.128	TCP	74	111 → 50290 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=40622 TSecr=493722 WS=32
386	44.958450	172.16.9.128	192.168.0.4	TCP	66	50290 → 111 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=493722 TSecr=40622
387	44.959790	172.16.9.128	192.168.0.4	TCP	66	50290 → 111 [RST, ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=493722 TSecr=40622
388	44.960252	172.16.9.128	192.168.0.11	TCP	66	54828 → 111 [RST, ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=493722 TSecr=38616
389	44.960484	172.16.9.128	192.168.0.12	TCP	74	37146 → 111 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493722 TSecr=0 WS=128
390	44.960502	192.168.0.12	172.16.9.128	TCP	74	111 → 37146 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=38624 TSecr=493722 WS=32
391	44.960758	172.16.9.128	192.168.0.12	TCP	66	37146 → 111 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=493722 TSecr=38624
392	44.961072	172.16.9.128	192.168.0.4	TCP	74	40296 → 113 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493722 TSecr=0 WS=128
393	44.961210	192.168.0.4	172.16.9.128	TCP	54	113 → 40296 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
394	44.961527	172.16.9.128	192.168.0.10	TCP	74	39840 → 110 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493722 TSecr=0 WS=128
395	44.961607	192.168.0.10	172.16.9.128	TCP	54	110 → 39840 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
396	44.961863	172.16.9.128	192.168.0.11	TCP	74	53116 → 113 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493723 TSecr=0 WS=128
397	44.961922	192.168.0.11	172.16.9.128	TCP	54	113 → 53116 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
398	44.962281	172.16.9.128	192.168.0.12	TCP	74	40016 → 113 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493723 TSecr=0 WS=128
399	44.962292	192.168.0.12	172.16.9.128	TCP	54	113 → 40016 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
400	44.962579	172.16.9.128	192.168.0.4	TCP	74	52010 → 119 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493723 TSecr=0 WS=128
401	44.962666	192.168.0.4	172.16.9.128	TCP	54	119 → 52010 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
402	44.963560	172.16.9.128	192.168.0.10	TCP	74	40450 → 111 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=493723 TSecr=0 WS=128
403	44.963946	192.168.0.10	172.16.9.128	TCP	74	111 → 40450 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=38627 TSecr=493723 WS=32
404	44.964270	172.16.9.128	192.168.0.10	TCP	66	40450 → 111 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=493723 TSecr=38627

Figure 11: Successful TCP connections on subnet A

On further analysis of the pcap generated from the client machine, it appears that some of the TCP SYN packets sent from the host machine managed to establish a full TCP connection with ports 111 on all machines on this subnet (highlighted above). Port 111 is reserved for the Remote Procedure Call protocol (RPC), a method of communication which allows a program to access a service located on another computer. When cross-examined with the output file "nmap3.txt", this shows that Nmap was able to identify that there was a service behind this port. The Portmap protocol is what provides Nmap with the information that determines the service behind port 111. The following figure shows an RPC Portmap call to 192.168.0.12:

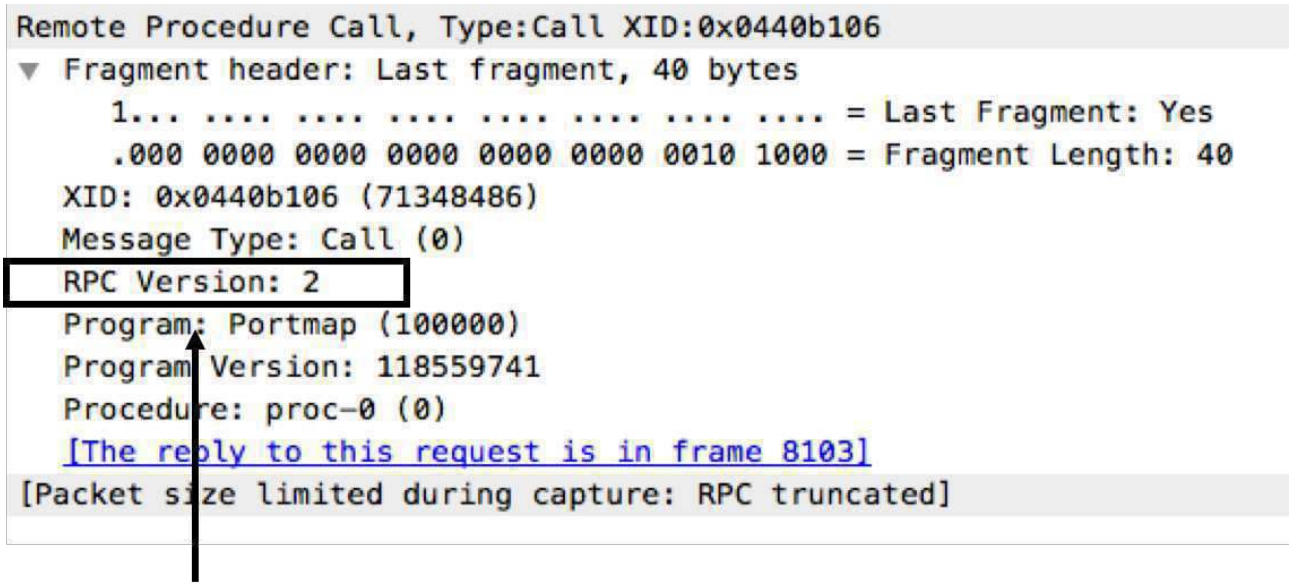


Figure 12: The RPC version shown within the RPC data frame

This RPC call specifies that the client machine (the host) wishes to use RPC version 2 (highlighted) which is then accepted by the server machine (client_2). This is what provides Nmap with its output. This is replicated over all the machines on subnet 192.168.0.0/24.

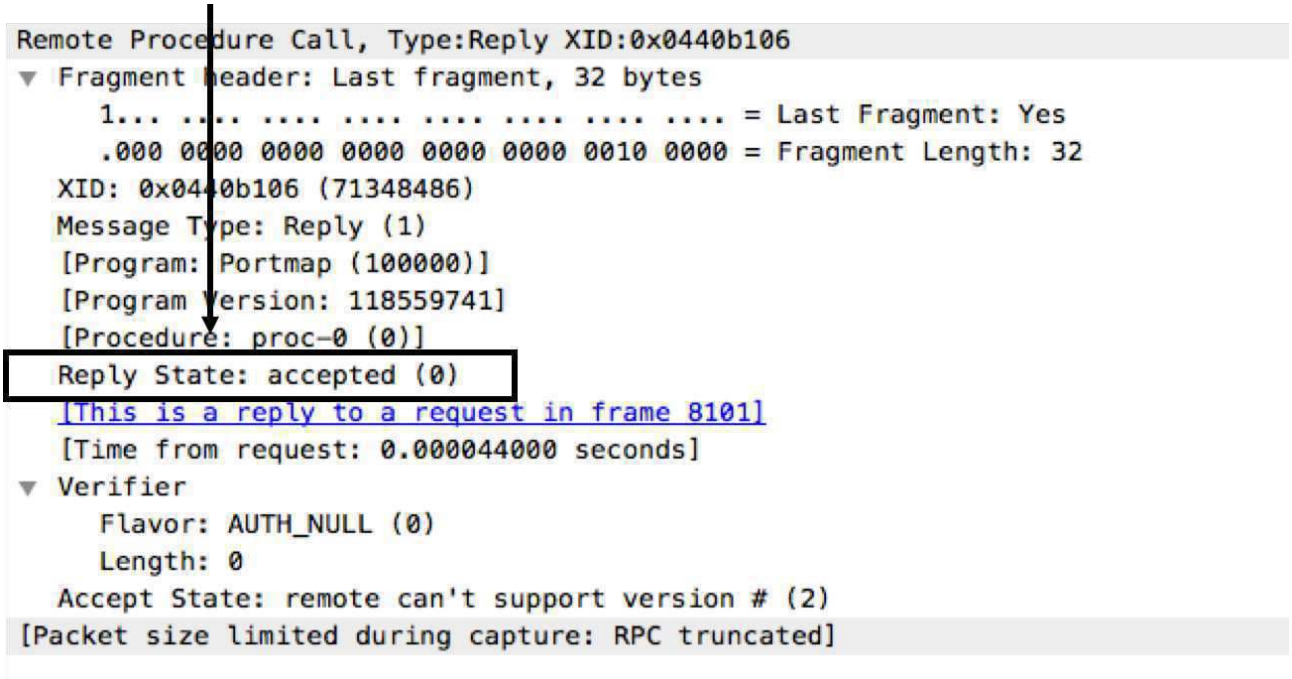


Figure 13: The RPC version being acknowledged between client and server

The same scan was ran on the 192.168.1.0/24 subnet, and the output file "nmap4.txt" shows that Nmap was able to identify the proFTPD running on the FTP server and the Apache httpd service running on the HTTP server. The following figures shows the data proving this:

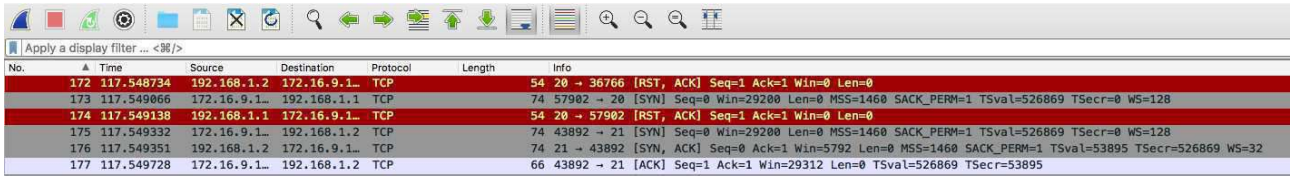


Figure 14: A successful FTP connection

Here, the TCP SYN packet is shown establishing a TCP connection on port 21 of the FTP server. The next packet is the response which follows a successful connection. The FTP server sends an FTP response which details the type of service being offered.

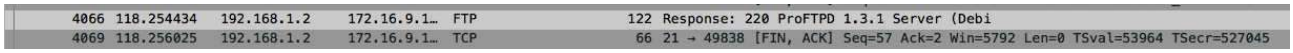


Figure 15: The data sent from the FTP server to the scanning machine

D.8.1.3 Nmap Banner Grab Scan The final stage of the Nmap experiment aimed at replicating the functionality of a wide range of network scanners which effect SCADA devices as well as traditional IP machines, banner grabbing. Banner grabbing is a technique used by a range of active scanners, including SHODAN, in order to gain information about the machines residing on a network. In order to conduct this scan, the pcaps from the previous Nmap experiments were extracted from the host machine and new Tcpdump sessions were activated on the same machines as the previous experiments. Once all the necessary machines began running Tcpdump, the following commands were executed against the two subnets:

"nmap –script=./banner.nse 192.168.0.0-15 -e nk_tap_kyle > nmap5.txt"

"nmap –script=./banner.nse 192.168.1.0-6 -e nk_tap_kyle > nmap6.txt"

These scans differ from the previous as they incorporate the usage of the Nmap Scripting Engine (NSE). The NSE allows for developers to write modules which can then be used by Nmap to extend it's capability. These modules are represented as ".nse" scripts which can be ingested into any Nmap command using the "–script=./" flag. The script chosen for this scan is the "banner.nse", a discovery module available at `https://svn.nmap.org/nmap/scripts/banner.nse`. This script is used to connect to any open port found by Nmap during a scan. If Nmap successfully connects to an open TCP port, the banner.nse script will capture any response that machine sends over that port within a 5 second window. These responses, or "barriers", provide information about the operating systems or services running on a target device, similar to the FTP response witnessed within the previous Nmap experiment.

On executing the commands stated above no errors occurred and the network remained stable throughout the duration of the scan. Output files were generated for each subnet, both containing information which confirms the scan completed successfully. The output for both subnets is as follows:

```
Starting Nmap 7.01 ( https://nmap.org ) at 2017-01-26 19:02 GMT
Nmap scan report for 192.168.0.4
Host is up (0.0043s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE
111/tcp open  rpcbind

Nmap scan report for 192.168.0.5
Host is up (0.0028s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE
111/tcp open  rpcbind

Nmap scan report for 192.168.0.10
Host is up (0.0059s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE
111/tcp open  rpcbind

Nmap scan report for 192.168.0.11
Host is up (0.0036s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE
111/tcp open  rpcbind

Nmap scan report for 192.168.0.12
Host is up (0.0046s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE
111/tcp open  rpcbind

Nmap done: 16 IP addresses (5 hosts up) scanned in 27.00 seconds
```

(a) The data containe withinh "nmap5.txt"

```
Starting Nmap 7.01 ( https://nmap.org ) at 2017-01-26 19:04 GMT
Nmap scan report for 192.168.1.1
Host is up (0.0026s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE
80/tcp  open  http
111/tcp open  rpcbind

Nmap scan report for 192.168.1.2
Host is up (0.0029s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE
21/tcp  open  ftp
|_banner: 220 ProFTPD 1.3.1 Server (Debian) [::ffff:192.168.1.2]
111/tcp open  rpcbind

Nmap scan report for 192.168.1.3
Host is up (0.0021s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE
111/tcp open  rpcbind

Nmap done: 7 IP addresses (3 hosts up) scanned in 25.06 seconds
```

(b) The data containe withinh "nmap6.txt"

Figure 16: The results of the Nmap banner grab scan

The data highlighted above shows that the banner.nse file was able to extract information about the FTP server located at 192.168.1.2. The output is similar to the version detection scan executed in the last experiment. This scan differed to the previous as it was not able to pull the banner from the HTTP server at 192.168.1.1, despite being able to identify that port 80 was open. Reviewing the code contained within the banner.nse script revealed that port 80 had not been declared in the list of ports to be interrogated by the script. The following pcap was taken from the FTP server in order to evaluated how the banner script obtains its data.

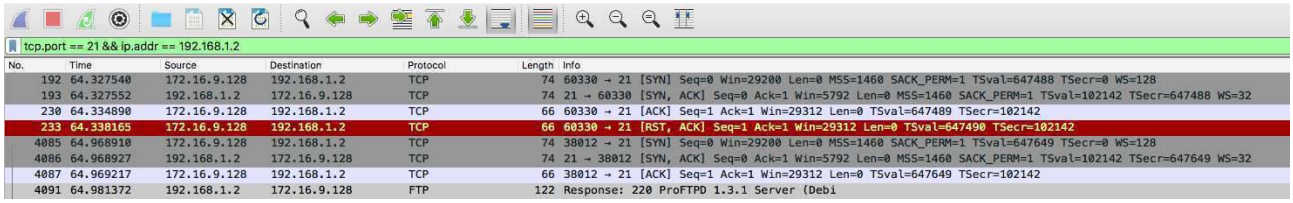


Figure 17: A pcap segment taken from the HTTP_Server

Here, the Nmap scan uses the exact same TCP SYN scan as the previous experiment. The FTP session successfully connects to the host machine before issuing a TCP RST/ACK packet. The reason for the RST/ACK packet is unknown at this point. The most likely explanation is that the port used by the host machine in order to contact the FTP server closed immediately after the TCP connection was established. Because of this, a second TCP connection was sent using a different port on the client side. This TCP connection successfully connects and gains a response from the FTP server behind port 21. It appears that the method of obtaining information about services using the banner script is the same as using a typical TCP SYN scan, the only difference is that the banner script targets specific ports and elaborates on the information provided by the responses gained from a successful TCP connection. The banner script did not provide any additional information over the service detection scan. Further experiments on different ports with a wider range of services would be needed in order to assess the full potential of the banner.nse script and how it differs from TCP scans.

Once these scans had been completed and all the packet captures had been terminated, the Netkit virtual machines were shut down using the "lcrash" command. After all the virtual machines had successfully powered-

off, the host machine was then restored to the snapshot taken before the experiment was conducted. The host machine had been returned to its normal state, as is ready to perform the next set of experiments.

D.8.2 Experiment 2: Zmap 2.1.0

A large issue facing both IP and SCADA based networks is the use of internet-wide scanners in order to gain information about potentially vulnerable systems running on networks across the globe. Zmap is a tool which has been specifically designed to perform Wide Area Network (WAN) scans at a quicker rate than other existing network mappers. To do so, Zmap uses random permutation of the IPv4 address space. This coupled with the fact that Zmap utilises raw sockets by default to send its scan packets means that there is no caching of conversations or any form of data exchange between the host performing the scan and the machines being targeted. This allows for the packets being sent by Zmap to reach the target network interface cards (NIC) as quickly as possible, reducing the potential saturation of either the host or the target networks. The focus on not establishing connections with the target devices and terminating sessions as soon as they are established is the main difference between this tool and Nmap, and is the main reason as to why Zmap is so refined to be used on a larger scale. The aims of this set of experiments is to try and replicate the host detection, service detection and connection capabilities offered by Nmap, but instead applying them using an internet-wide scanner. This will give an insight as to how potential threats to SCADA devices such as Shodan interact with devices within the IP space, and how this would differ on a SCADA-based network

In order to perform these Zmap scans, the virtual network used in the previous experiment was restored to its original state before any experiments had taken place through the use of a virtual machine snapshot. Once the host virtual machine has been restored to its previous state, the virtual network is loaded using the "lstart" command. Each virtual machine loads with the configuration supplied at the beginning of this document. A script is then ran in order to allow the host machine to interact with the virtual network via tap/gateway device. Once this routing information has been set, the environment is ready to conduct the Zmap experiments.

D.8.2.1 Zmap Ping Sweep The first scan, similar to the Nmap experiment, aims to use the ICMP protocol in order to detect which addresses are active on the network, without having to establish a connection with each host it finds. The addresses being scanned are the same as before and are defined in table Table 13. In order to analyse the packets being sent to each of the hosts on the virtual network, Tcpdump has been deployed and will save the data in the form of a pcap file on the host system.

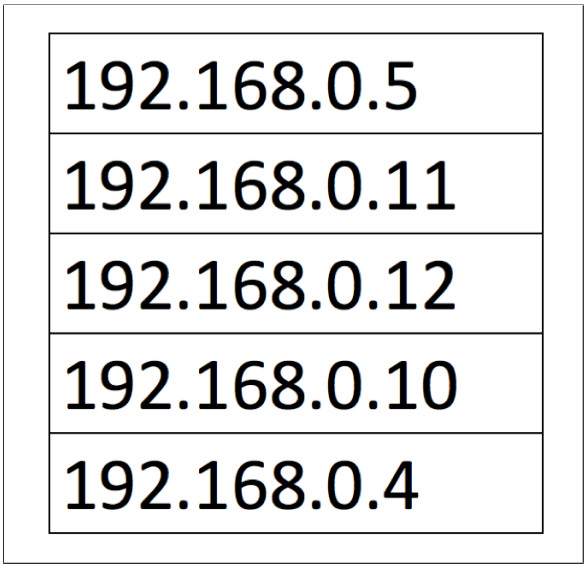
The first set of scans launched using Zmap aimed at performing host discovery using the ICMP echo request as discussed earlier in this document. The commands entered into the host machine are shown below:

"sudo zmap -M icmp_echoscan 192.168.0.0/24 -o zmap1.csv"

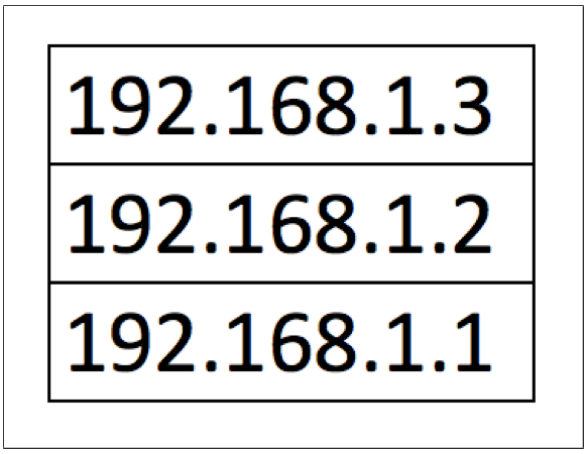
"sudo zmap -M icmp_echoscan 192.168.1.0/24 -o zmap2.csv"

The "-M" flag is used to select the type of probe that Zmap will run against the selected targets. In this case the "icmp_echoscan" has been selected, which sends ICMP echo request packets to each host and logs the addresses

of the devices which respond. Unlike Nmap, the IP addresses are specified as a subnet, rather than range of addresses. Here, the subnets configured have a /24 mask, meaning that Zmap will scan 254 addresses from 192.168.0.0 - 192.168.0.254 and the same on the 192.168.1.0/24 subnet. The "-o" flag utilises Zmap's in-built output system in order to save the information it obtains into a .csv file, which corresponds to each unique scan and is saved on the host machine. On execution of each Zmap command, there were no errors during the scanning process and each .csv file was successfully populated with data. After these initial scans had completed and the Tcpdump captures had been terminated, FTP and HTTP requests were sent from the client machines to the server subnet in order to ensure that the network was still responsive after the scans had taken place. The network was still responsive and correctly configured after the Zmap scans had finished. The following figures show the addresses that Zmap was able to obtain using its ICMP echo scan:



(a) The addresses returned from subnet A



(b) The addresses returned from subnet C

Figure 18: The results of the Zmap ping sweep

Here, Zmap has been able to identify each host configured on both the 192.168.0.0/24 subnet as well as the 192.168.1.0/24 subnet. As the scan consisted of ICMP echo packets, there is no more data to be expected in the output, so only displaying the addresses of the active hosts is sufficient for this type of scan. The pcap extracted from the client_1 machine shows how Zmap used ICMP in order to assess the active hosts on each network:

Client_1

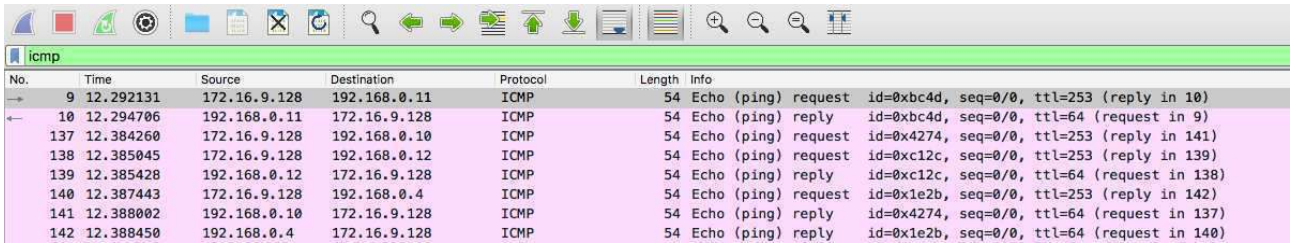


Figure 19: A segment of a pcap taken from the Client_1 machine

Here, a display filter was used in order to extract only the ICMP requests and replies from the previous scan. This was done in order to better analyse the packets, rather than having to filter through a large amount of

ARP and DHCP packets which were relevant to this tool. It can be shown from this packet capture that Zmap is simply sending an ICMP packets to each host on the network and the addresses which form a reply are logged in the output .csv file. Similar to the results from the Nmap experiment, the pcaps captured on the client and server machines are unable to detect the presence of each gateway device, although their eth0 NIC is configured to sit on the same network, and the output files acknowledge their existence. An explanation for this may be that the observation points for these experiments may not be place correctly in order to detect the conversations between the host machine and each gateway device.

FTP_Server

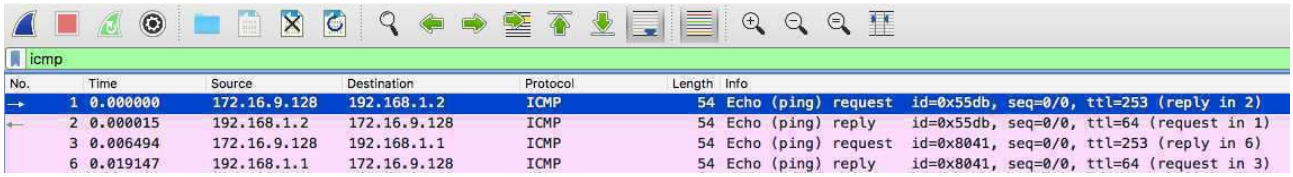


Figure 20: A segment of a pcap taken from the FTP_Server

Here, the pcap taken from the FTP server shows that Zmap follows the exact same procedure on the other subnet. Again, the traffic from each server can be seen, but the gateway device is still not present within this pcap. Once these pcaps had been obtained and analysed, the virtual network was ready for the second phase of Zmap testing.

D.8.2.2 Zmap TCP Scan The second experiment aimed to showcase how Zmap uses TCP packets in order to connect to hosts and retrieve information about the target devices. At the time of these experiments, Zmap does not support the ability to query multiple TCP ports using a singular TCP scan. For this reason, the port chosen for this experiment is one of the most commonly used services, and should provide some data when ran against the virtual network. The commands executed are displayed and explained below:

"sudo zmap -p 80 -M tcp_synscan 192.168.0.0/24 -o zmap3.csv"

"sudo zmap -p 80 -M tcp_synscan 192.168.1.0/24 -o zmap4.csv"

Similar to the commands executed in the previous experiment, "-M" is used again in order to specify the type of probe to be sent out to the target network, in this case a TCP SYN scan. The choice to use a SYN scan was to see if there are any underlying differences between Zmap and Nmap, are there any differences in the use of TCP SYN packets, and is there a difference between the data extracted using Zmap to that gained through Nmap? As referenced above, unlike Nmap, Zmap does not support the probing of multiple ports during a single session, this is the reason for the use of the "-p" flag. This specifies the single port that Zmap should scan against in order to detect whether that port is open on the target device, and whether there is a service running behind it. In order to capture the traffic generated by Zmap, Tcpdump was initiated at the same observation points as in the previous Nmap and Zmap experiments.

Executing these commands on each of the subnets within the virtual network yielded different results to all previous experiments. Launching this scan against the 192.168.0.0/24 subnet failed to disclose details of any of the hosts that resided within it. Despite the ICMP scan confirming that the virtual machines could be reached by the host machine, the TCP scan failed to supply any information about the same devices identified in the previous experiment. In order to determine why this had happened, the pcaps taken from the client_1 machine and DHCP_server were opened and analysed. The traffic can be shown in the following figures:

Client_1

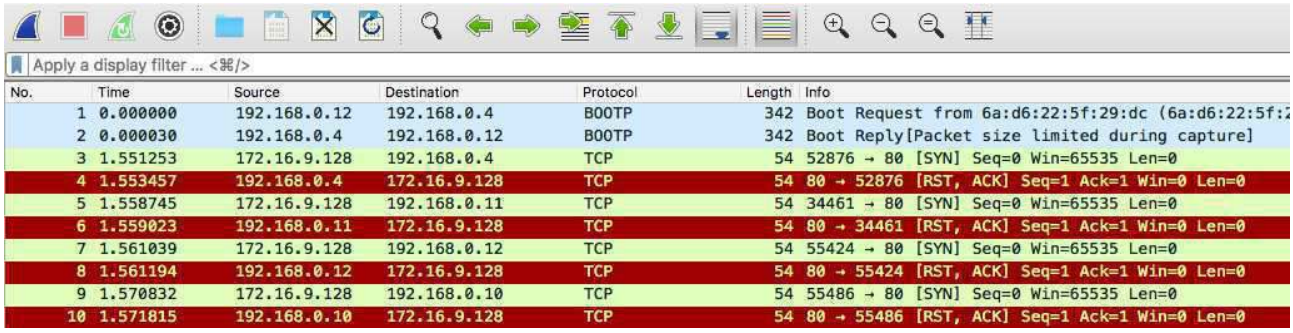


Figure 21: A pcap segment showing TCP packets being received by Client_1

FTP_Server

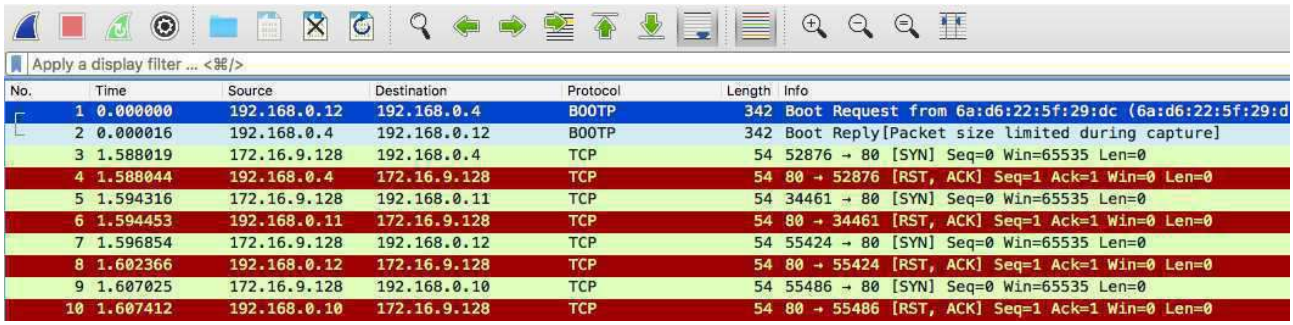


Figure 22: A pcap segment showing TCP packets being received by FTP_Server

The pcap taken from the client_1 machine and DHCP_server shows that Zmap is using a similar technique to the service detection scan performed using Nmap. However, unlike Nmap, the Zmap command is only targeting port 80 as specified within the command shown earlier in this document. It appears that Zmap does not use TCP SYN packets in order to detect whether a host is on a network or to provide details about the services running on each port. Instead, if a port has been specified by the user, Zmap will only give the addresses of machines which have that particular port open. This assumption is then supported by both the output file "zmap4.csv" and the pcap taken from the HTTP server on the 192.168.1.0/24 subnet.

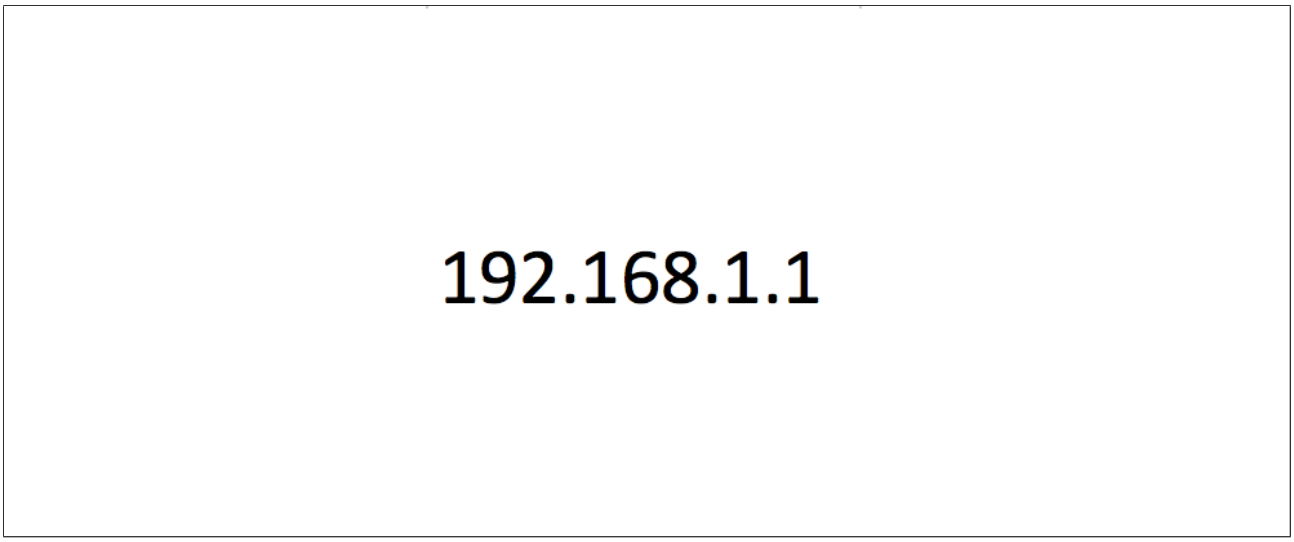


Figure 23: The single address returned from subnet C

HTTP_Server

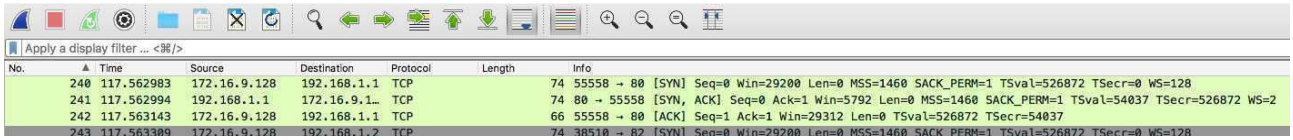


Figure 24: A successful HTTP TCP handshake

Here, as the TCP SYN packet was sent to the web server sitting behind port 80 on the HTTP Server, a 3-way handshake was able to complete its cycle and thus Zmap logged the address of that machine and terminated the connection using a TCP RST packet. This shows that even though Zmap was able to establish a connection with machine 192.168.1.1 on port 80, there was no further translation of data. There was no GET or POST requests to the server, and the server did not send a response message back to the host machine, the TCP stream was simply reset and dropped. This suggests that the Zmap tool is a lot more targeted than Nmap, and that the main focus of the tool is identifying active IP addresses, rather than gathering unique information from each of the machines which have been identified. This supports the statement earlier in this document, confirming that Zmap's functionality has been refined for scanning LAN over WAN or corporate intranets. Once these pcaps had been transferred from the host machine, and all scans and packet captures has been terminated, the network was then ready for the final set of Zmap experiments.

D.8.2.3 Zmap Network Time Protocol Scan The final Zmap tests aim at scanning hosts using the Network Time Protocol (NTP). This protocol is used in order to synchronise the internal clocks of devices over unreliable or variable-latency networks. This protocol relies on the User Datagram Protocol to send timestamps between two machines. Previous scans have not utilised UDP and have not targeted the protocols which control the timing and synchronizing of the physical machines and their NICs. The commands to facilitate these scans are detailed as follows:

"sudo zmap -p 80 -M ntp 192.168.0.0/24 -o zmap5.csv"

"sudo zmap -p 80 -M ntp 192.168.1.0/24 -o zmap6.csv"

The syntax for this command is the same as the previous TCP SYN scan, the difference being that the probing module has been set to “ntp”. The subnets have remained the same, the port number is still aimed at the http port 80, and each of the commands have been instructed to output their data into two individual .csv files.

On execution of these two Zmap commands no errors occurred during the scanning process and the virtual network remained fully usable after the scans had been performed. However, on inspection of the output files, Zmap did not provide any data for either of the two subnets. On opening pcaps from both subnets, the following data was obtained:

Router_1

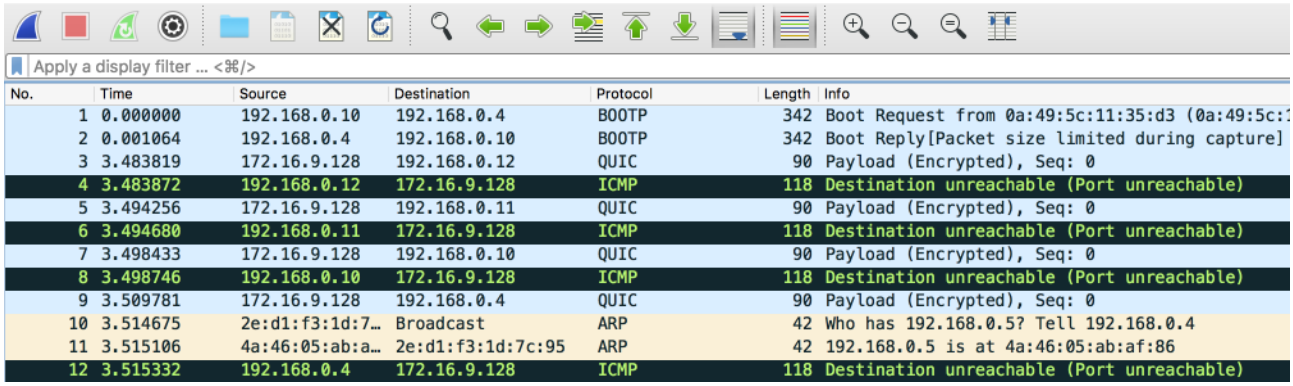


Figure 25: A segment of a pcap taken from the Router_1 machine

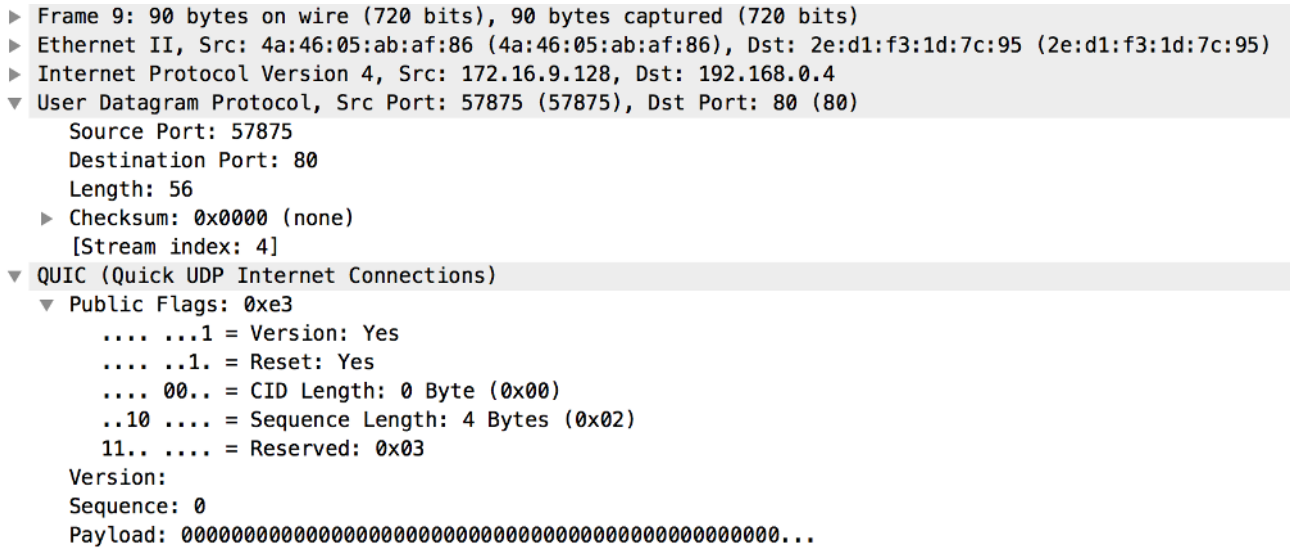


Figure 26: A figure showing the contents of a QUIC UDP packet

Here, through analysing both the sequence of packets being sent/received and the data held in the packets sent using Zmap, it appears that the “Quick UDP Internet Connections (QUIC)” protocol cannot be correctly applied to the port 80. The payload of this type of scan contains all zeros after the UDP frame, something that will not warrant any type of response regardless of the service behind that port or the machine it is running on. This is followed by an ICMP type 3 message from the target machines to the host machine to signify that the destination port is unreachable. The results obtained from the final Zmap scans were reflected on the 192.168.1.0/24 subnet.

With the pcaps from the final Zmap experiment extracted from host machine the virtual network was shut down using the "lcrash" command. Once all virtual machines had successfully powered-off, the host machine was restored back to its pre-experiment state by loading a previously saved snapshot. Once this snapshot had loaded, the virtual network was launched using the "lstart" command. The testing environment was then ready to begin conducting the passive sniffer experimentation.

D.9 Execution of Experiments: Passive Sniffers

D.9.1 Experiment 3: Tshark 2.0.5

D.9.1.1 Tshark ICMP Capture Tshark is a Command Line Interface for the passive network sniffer Wireshark. Tshark uses the libpcap Linux library in order to capture network traffic directly from the NIC of the chosen machines. These packets are then put into Tshark's capture engine called dumpcap which allows the intercepted traffic, along with basic metadata such as timestamps, to files such as .pcap and .pcapng. The dumpcap capture engine also allows for filters to be applied to the captured packets, in order to constrict the types of data being pulled from the NIC and written to the disk. It is a promiscuous-based sniffer, meaning that the NIC running a Tshark capture will be able to receive all network being sent across a network or subnet, as long as the network devices are connected via a hub and not a switch. It is a commonly used tool in both network forensic investigations as well traffic analysis and network administration. For these reasons, Tshark was chosen as one of the tools to asses against IP and non-IP based networks.

In order to conduct experiments with Tshark, the virtual network had to be adapted in order to ensure the best results were obtained. Firstly, the observation points for the passive tests have been moved and replaced with Tshark captures. Tcpdump is no longer needed as it is another form of passive scanner, who's results have already been demonstrated through the capturing of the active scanner traffic in the previous experiments. Secondly, in order to enable the observation points to capture network traffic, each router machine had to be configured with more memory than originally allocated. In order to do this, the following lines were inserted into the "lab.conf" file on the host machine:

"router_3[mem]=512"

"router_2[mem]=512"

"router_1[mem]=512"

This configuration specifies that each machine acting as an observation point will have 512MB of emulated memory (RAM) which is required by Tshark in order to capture and save network traffic. Once these changes had been made, the virtual network was then restarted. Once the virtual network had restarted, the following commands were executed in the observation machines:

"Tshark -ni any -w /hosthome/pcaps/tshark/router1_1.pcap"

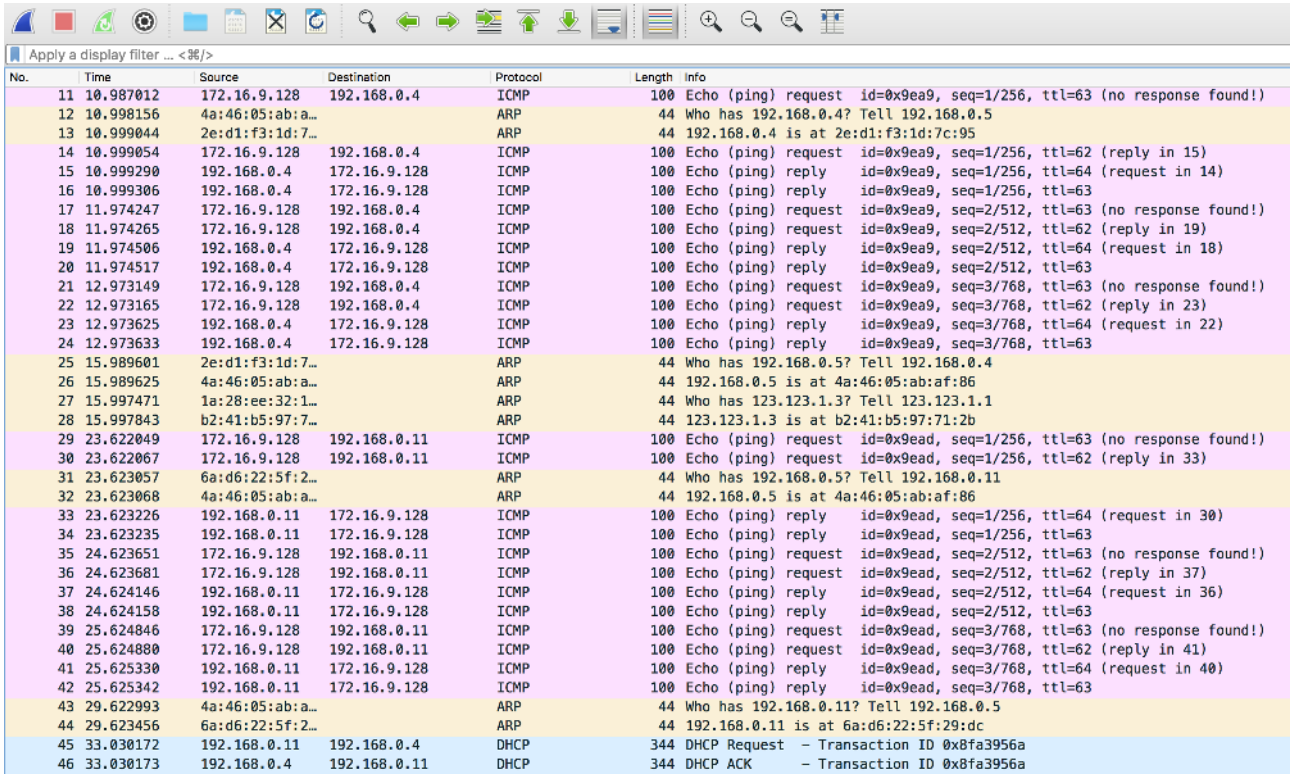
"Tshark -ni any -w /hosthome/pcaps/tshark/router2_1.pcap"

"Tshark -ni any -w /hosthome/pcaps/tshark/router3_1.pcap"

These commands instruct Tshark to capture traffic on any of the NICs configured on that particular device, this specified using the "-ni" flag with the argument "any". All the traffic captured from these interfaces is then written "-w" to a .pcap file saved on the host machine, with a naming convention which references the machine it was captured from and the number of the experiment. On executing these commands Tshark signified that it was running correctly on each machine and that it was capturing traffic on all interfaces. In order to generate traffic for the observation points to capture, an ICMP ping request was sent from the host machine to each one of the addresses residing on the virtual network.

The following pcaps show the flow of each ICMP echo request through the observation point at Router_1.

Router_1


Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol	Length	Info
11	10.987012	172.16.9.128	192.168.0.4	ICMP	100	Echo (ping) request id=0x9ea9, seq=1/256, ttl=63 (no response found!)
12	10.998156	4a:46:05:ab:a…		ARP	44	Who has 192.168.0.4? Tell 192.168.0.5
13	10.999044	2e:d1:f3:1d:7…		ARP	44	192.168.0.4 is at 2e:d1:f3:1d:7c:95
14	10.999054	172.16.9.128	192.168.0.4	ICMP	100	Echo (ping) request id=0x9ea9, seq=1/256, ttl=62 (reply in 15)
15	10.999290	192.168.0.4	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ea9, seq=1/256, ttl=64 (request in 14)
16	10.999306	192.168.0.4	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ea9, seq=1/256, ttl=63
17	11.974247	172.16.9.128	192.168.0.4	ICMP	100	Echo (ping) request id=0x9ea9, seq=2/512, ttl=63 (no response found!)
18	11.974265	172.16.9.128	192.168.0.4	ICMP	100	Echo (ping) request id=0x9ea9, seq=2/512, ttl=62 (reply in 19)
19	11.974506	192.168.0.4	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ea9, seq=2/512, ttl=64 (request in 18)
20	11.974517	192.168.0.4	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ea9, seq=2/512, ttl=63
21	12.973149	172.16.9.128	192.168.0.4	ICMP	100	Echo (ping) request id=0x9ea9, seq=3/768, ttl=63 (no response found!)
22	12.973165	172.16.9.128	192.168.0.4	ICMP	100	Echo (ping) request id=0x9ea9, seq=3/768, ttl=62 (reply in 23)
23	12.973625	192.168.0.4	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ea9, seq=3/768, ttl=64 (request in 22)
24	12.973633	192.168.0.4	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ea9, seq=3/768, ttl=63
25	15.989601	2e:d1:f3:1d:7…		ARP	44	Who has 192.168.0.5? Tell 192.168.0.4
26	15.989625	4a:46:05:ab:a…		ARP	44	192.168.0.5 is at 4a:46:05:ab:af:86
27	15.997471	1a:28:ee:32:1…		ARP	44	Who has 123.123.1.3? Tell 123.123.1.1
28	15.997843	b2:41:b5:97:7…		ARP	44	123.123.1.3 is at b2:41:b5:97:71:2b
29	23.622049	172.16.9.128	192.168.0.11	ICMP	100	Echo (ping) request id=0x9ead, seq=1/256, ttl=63 (no response found!)
30	23.622067	172.16.9.128	192.168.0.11	ICMP	100	Echo (ping) request id=0x9ead, seq=1/256, ttl=62 (reply in 33)
31	23.623057	6a:d6:22:5f:2…		ARP	44	Who has 192.168.0.5? Tell 192.168.0.11
32	23.623068	4a:46:05:ab:a…		ARP	44	192.168.0.5 is at 4a:46:05:ab:af:86
33	23.623226	192.168.0.11	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ead, seq=1/256, ttl=64 (request in 30)
34	23.623235	192.168.0.11	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ead, seq=1/256, ttl=63
35	24.623651	172.16.9.128	192.168.0.11	ICMP	100	Echo (ping) request id=0x9ead, seq=2/512, ttl=63 (no response found!)
36	24.623681	172.16.9.128	192.168.0.11	ICMP	100	Echo (ping) request id=0x9ead, seq=2/512, ttl=62 (reply in 37)
37	24.624146	192.168.0.11	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ead, seq=2/512, ttl=64 (request in 36)
38	24.624158	192.168.0.11	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ead, seq=2/512, ttl=63
39	25.624846	172.16.9.128	192.168.0.11	ICMP	100	Echo (ping) request id=0x9ead, seq=3/768, ttl=63 (no response found!)
40	25.624880	172.16.9.128	192.168.0.11	ICMP	100	Echo (ping) request id=0x9ead, seq=3/768, ttl=62 (reply in 41)
41	25.625330	192.168.0.11	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ead, seq=3/768, ttl=64 (request in 40)
42	25.625342	192.168.0.11	172.16.9.128	ICMP	100	Echo (ping) reply id=0x9ead, seq=3/768, ttl=63
43	29.622993	4a:46:05:ab:a…		ARP	44	Who has 192.168.0.11? Tell 192.168.0.5
44	29.623456	6a:d6:22:5f:2…		ARP	44	192.168.0.11 is at 6a:d6:22:5f:29:dc
45	33.030172	192.168.0.11	192.168.0.4	DHCP	344	DHCP Request - Transaction ID 0x8fa3956a
46	33.030173	192.168.0.4	192.168.0.11	DHCP	344	DHCP ACK - Transaction ID 0x8fa3956a

Figure 27: The packets obtained from Tshark on the Router_1 machine

Here, the data captured from router 1 shows the ICMP requests and responses being sent to each address on the 192.168.0.0/24 subnet. Not only does it show that there are active hosts at these addresses, the packet capture also shows DHCP communications traveling between 192.168.0.11 and 192.168.0.4. It can then be deduced that as 192.168.0.4 is sending a DHCP acknowledge (ACK) packet to the recipient address, this machine must be a DHCP server distributing addresses to that subnet. This packet capture supports and displays all the correct configurations of the virtual network.

In regards to the 192.168.1.0/24 subnet, the packet capture shows which addresses are active hosts, however, as there are no actual services being ran over the network, no further information is given about the active hosts, their purpose or the services they offer.

As Tshark is a passive tool no additional tests need to be performed in order to check the integrity and configuration of the virtual network after the captures have taken place, as it is evident from the pcaps that the network functioned correctly throughout the duration of each capture. Once the pcaps had been saved and extracted from the host machine, the next experiment could be performed.

D.9.1.2 Tshark Baseline Network Capture The next experiment aims to evaluate the success of Tshark when capturing the normal traffic of the virtual network, rather than ICMP echo requests. The syntax for each Tshark capture remains exactly the same as the previous experiment and the observation points also remain in the same position on the network. The only changes made are the names of the pcap files being created so that they represent each individual experiment. The commands are detailed below:

"Tshark -ni any -w /hosthome/pcaps/tshark/router1_2.pcap"

"Tshark -ni any -w /hosthome/pcaps/tshark/router2_2.pcap"

"Tshark -ni any -w /hosthome/pcaps/tshark/router3_2.pcap"

On executing these commands Tshark signified that it was running correctly on each machine and that it was capturing traffic on all interfaces. The traffic being generated aimed to simulate a simple client-server interaction, with each host making an individual request to both the HTTP server and FTP server on the other subnet, as well as any other autonomous traffic produced by other machines on the network. The HTTP interaction will be a simple GET request which will cause the HTTP server to send a HTML webpage to each client (Figure 4). In regards to the FTP session, each host will logon to the server using the username and password "test", which will then be followed by a series of commands which will send, retrieve and delete a file on the server before logging-off. The following pcap was taken from the router_3 machine and shows the traffic being sent between each subnet:

Router_3

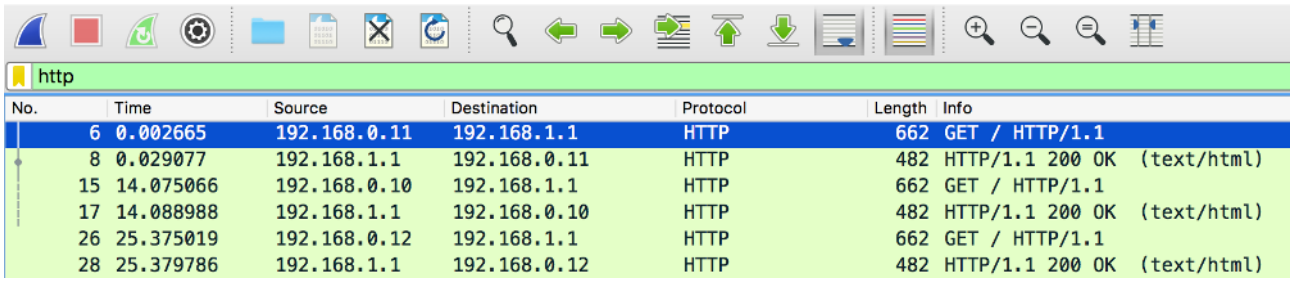
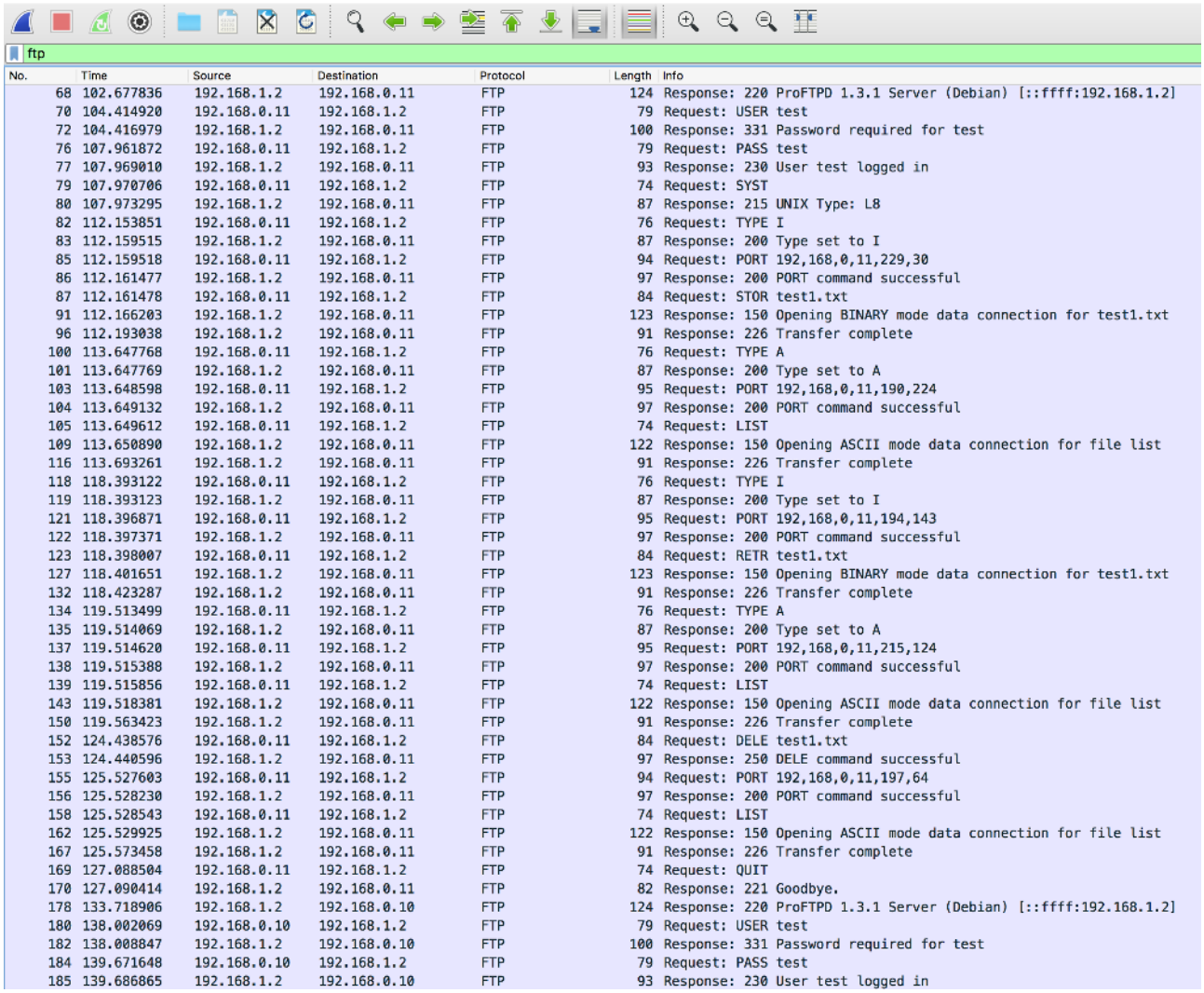


Figure 28: A pcap segment showing a HTTP conversation between the server and Client_1



ftp

No.	Time	Source	Destination	Protocol	Length	Info
68	102.677836	192.168.1.2	192.168.0.11	FTP	124	Response: 220 ProFTPD 1.3.1 Server (Debian) [::ffff:192.168.1.2]
70	104.414920	192.168.0.11	192.168.1.2	FTP	79	Request: USER test
72	104.416979	192.168.1.2	192.168.0.11	FTP	100	Response: 331 Password required for test
76	107.961872	192.168.0.11	192.168.1.2	FTP	79	Request: PASS test
77	107.969010	192.168.1.2	192.168.0.11	FTP	93	Response: 230 User test logged in
79	107.970706	192.168.0.11	192.168.1.2	FTP	74	Request: SYST
80	107.973295	192.168.1.2	192.168.0.11	FTP	87	Response: 215 UNIX Type: L8
82	112.153851	192.168.0.11	192.168.1.2	FTP	76	Request: TYPE I
83	112.159515	192.168.1.2	192.168.0.11	FTP	87	Response: 200 Type set to I
85	112.159518	192.168.0.11	192.168.1.2	FTP	94	Request: PORT 192,168,0,11,229,30
86	112.161477	192.168.1.2	192.168.0.11	FTP	97	Response: 200 PORT command successful
87	112.161478	192.168.0.11	192.168.1.2	FTP	84	Request: STOR test1.txt
91	112.166203	192.168.1.2	192.168.0.11	FTP	123	Response: 150 Opening BINARY mode data connection for test1.txt
96	112.193038	192.168.1.2	192.168.0.11	FTP	91	Response: 226 Transfer complete
100	113.647768	192.168.0.11	192.168.1.2	FTP	76	Request: TYPE A
101	113.647769	192.168.1.2	192.168.0.11	FTP	87	Response: 200 Type set to A
103	113.648598	192.168.0.11	192.168.1.2	FTP	95	Request: PORT 192,168,0,11,190,224
104	113.649132	192.168.1.2	192.168.0.11	FTP	97	Response: 200 PORT command successful
105	113.649612	192.168.0.11	192.168.1.2	FTP	74	Request: LIST
109	113.650890	192.168.1.2	192.168.0.11	FTP	122	Response: 150 Opening ASCII mode data connection for file list
116	113.693261	192.168.1.2	192.168.0.11	FTP	91	Response: 226 Transfer complete
118	118.393122	192.168.0.11	192.168.1.2	FTP	76	Request: TYPE I
119	118.393123	192.168.1.2	192.168.0.11	FTP	87	Response: 200 Type set to I
121	118.396871	192.168.0.11	192.168.1.2	FTP	95	Request: PORT 192,168,0,11,194,143
122	118.397371	192.168.1.2	192.168.0.11	FTP	97	Response: 200 PORT command successful
123	118.398007	192.168.0.11	192.168.1.2	FTP	84	Request: RETR test1.txt
127	118.401651	192.168.1.2	192.168.0.11	FTP	123	Response: 150 Opening BINARY mode data connection for test1.txt
132	118.423287	192.168.1.2	192.168.0.11	FTP	91	Response: 226 Transfer complete
134	119.513499	192.168.0.11	192.168.1.2	FTP	76	Request: TYPE A
135	119.514069	192.168.1.2	192.168.0.11	FTP	87	Response: 200 Type set to A
137	119.514620	192.168.0.11	192.168.1.2	FTP	95	Request: PORT 192,168,0,11,215,124
138	119.515388	192.168.1.2	192.168.0.11	FTP	97	Response: 200 PORT command successful
139	119.515856	192.168.0.11	192.168.1.2	FTP	74	Request: LIST
143	119.518381	192.168.1.2	192.168.0.11	FTP	122	Response: 150 Opening ASCII mode data connection for file list
150	119.563423	192.168.1.2	192.168.0.11	FTP	91	Response: 226 Transfer complete
152	124.438576	192.168.0.11	192.168.1.2	FTP	84	Request: DELE test1.txt
153	124.440596	192.168.1.2	192.168.0.11	FTP	97	Response: 250 DELE command successful
155	125.527603	192.168.0.11	192.168.1.2	FTP	94	Request: PORT 192,168,0,11,197,64
156	125.528230	192.168.1.2	192.168.0.11	FTP	97	Response: 200 PORT command successful
158	125.528543	192.168.0.11	192.168.1.2	FTP	74	Request: LIST
162	125.529925	192.168.1.2	192.168.0.11	FTP	122	Response: 150 Opening ASCII mode data connection for file list
167	125.573458	192.168.1.2	192.168.0.11	FTP	91	Response: 226 Transfer complete
169	127.088504	192.168.0.11	192.168.1.2	FTP	74	Request: QUIT
170	127.090414	192.168.1.2	192.168.0.11	FTP	82	Response: 221 Goodbye.
178	133.718906	192.168.1.2	192.168.0.10	FTP	124	Response: 220 ProFTPD 1.3.1 Server (Debian) [::ffff:192.168.1.2]
180	138.002069	192.168.0.10	192.168.1.2	FTP	79	Request: USER test
182	138.008847	192.168.1.2	192.168.0.10	FTP	100	Response: 331 Password required for test
184	139.671648	192.168.0.10	192.168.1.2	FTP	79	Request: PASS test
185	139.686865	192.168.1.2	192.168.0.10	FTP	93	Response: 230 User test logged in

Figure 29: The full list of FTP conversations originating from subnet A

The first pcap has been filtered to show only the HTTP traffic passing through the router_3 observation point. Here it is shown that all of the hosts from subnet A have managed to establish a connection with the HTTP server at 192.168.1.1 and then sending using GET requests to the machine. The server then responds with a 200 OK response for each individual host and contains the website referenced in Figure 28 in each payload.

Within the second pcap segment, also taken from router_3, the traffic has been filtered to only show the FTP traffic which passed by the observation point. The areas highlighted in black show two separate hosts logging-on to the FTP server with the username and password referenced earlier in this document. The area highlighted in red shows the STORE, LIST, RETREVE and DELETE commands used on the file "test1.txt". The data in this pcap shows that through analysis of the traffic intercepted at 3 different points on the network, the purpose and state of each machine can be identified through the use of passive scanners such as Tshark. However, this experiment has failed to address/identify the machines within acting as gateway devices, placed between subnet B and the other subnets. There has been no traffic aside from ARP broadcasts to prove that these gateway devices exist. Furthermore, an analysts cannot deduce the function of these devices through ARP requests, so there is an issue when trying to identify the potential services these devices hold. A reason for this may have been the placement of the observation points during this experiment, as the active scanners were able to detect

the machines, but the traffic being sent to them from each experiment could not be seen.

Once each Tshark capture had finished, the virtual network was shut down in preparation for the last set of passive experiments.

D.9.2 Experiment 4: Ettercap 0.8.2

D.9.2.1 Ettercap ICMP Interception Ettercap is an open source network tool which allows users to perform man-in-the-middle (MITM) attacks on LANs which can be used for protocol analysis and network auditing. Ettercap uses ARP poisoning in order to send spoofed ARP messages across the LAN in order to imitate an already existing device on the network. By spoofing associating the MITM MAC address with the IP address of another device, any traffic being sent to the existing device gets sent to and intercepted by the MITM machine. In order to create a man-in-the-middle environment, two more machines needed to be added to the virtual network. These two machines will be used in order to spoof the MAC address of a specified machine on each subnet, meaning any packets destined for the victim machine will be sent to the MITM machine. To do this, two more machines were added to the lab.conf file, as well as creating mitm_1/2 directories and mitm_1/2.startup files. These changed are shown in the figure below:

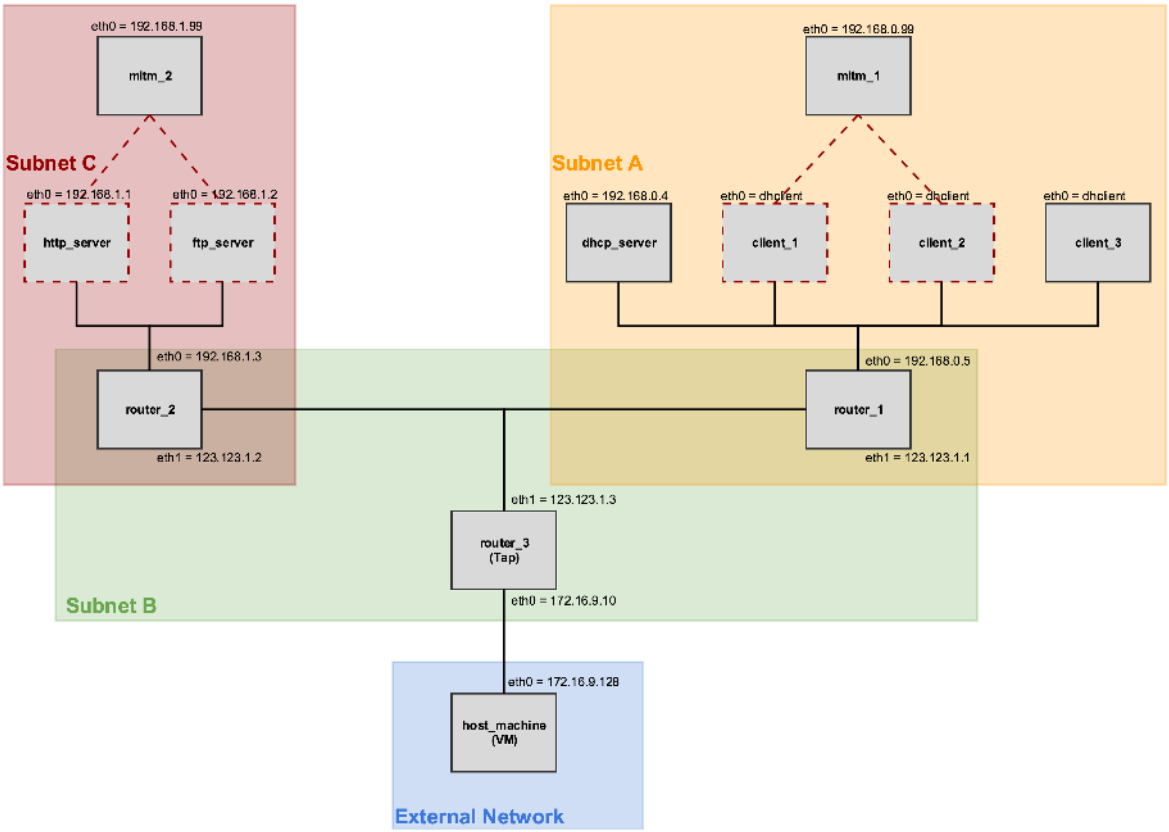


Figure 30: A redesigned topology showing two additional MITM machines and the target machines

The figure above shows the two new machines added to each subnet. The red lines indicate the machines to be spoofed within the Ettercap experiments.

The first Ettercap poisoning was used to spoof the MAC addresses of the client_1 and the http_server machines. When traffic is sent and received by each of these machines, the MITM device will read intercept them before relaying the packets to their true destination. In order to achieve this, the following commands were executed on the MITM devices:

"Ettercap -T -w /hosthome/pcaps/Ettercap/mitm11.pcap -M ARP /192.168.0.10/ // output:"

"Ettercap -T -w /hosthome/pcaps/Ettercap/mitm21.pcap -M ARP /192.168.1.1/ // output:"

Ettercap is executed as a CLI program using the "-T" flag. In order to capture the traffic passing through the MITM machine, the "-w" flag is used in order to write all the traffic to a pcap file on the host machine, similar to all the previous experiments. The "-M ARP" flag and argument tells Ettercap to load its ARP poisoning module, the targets are then enclosed between the two slashes "//". The next two slashes are used to define a particular port to intercept, however leaving this field blank allows for all ports to be intercepted and captured. The final option "output:" shows the data being written to the pcap file onto the screen.

Once each Ettercap command was executed, packets began to show on each one of the MITM machines, meaning that the ARP spoofing had started successfully. Once both MITM machines were running, some ICMP ping packets were sent from the host machine to each of the addresses being spoofed. The following pcaps show the traffic from both the client_1 machine and the mitm_1 machine:

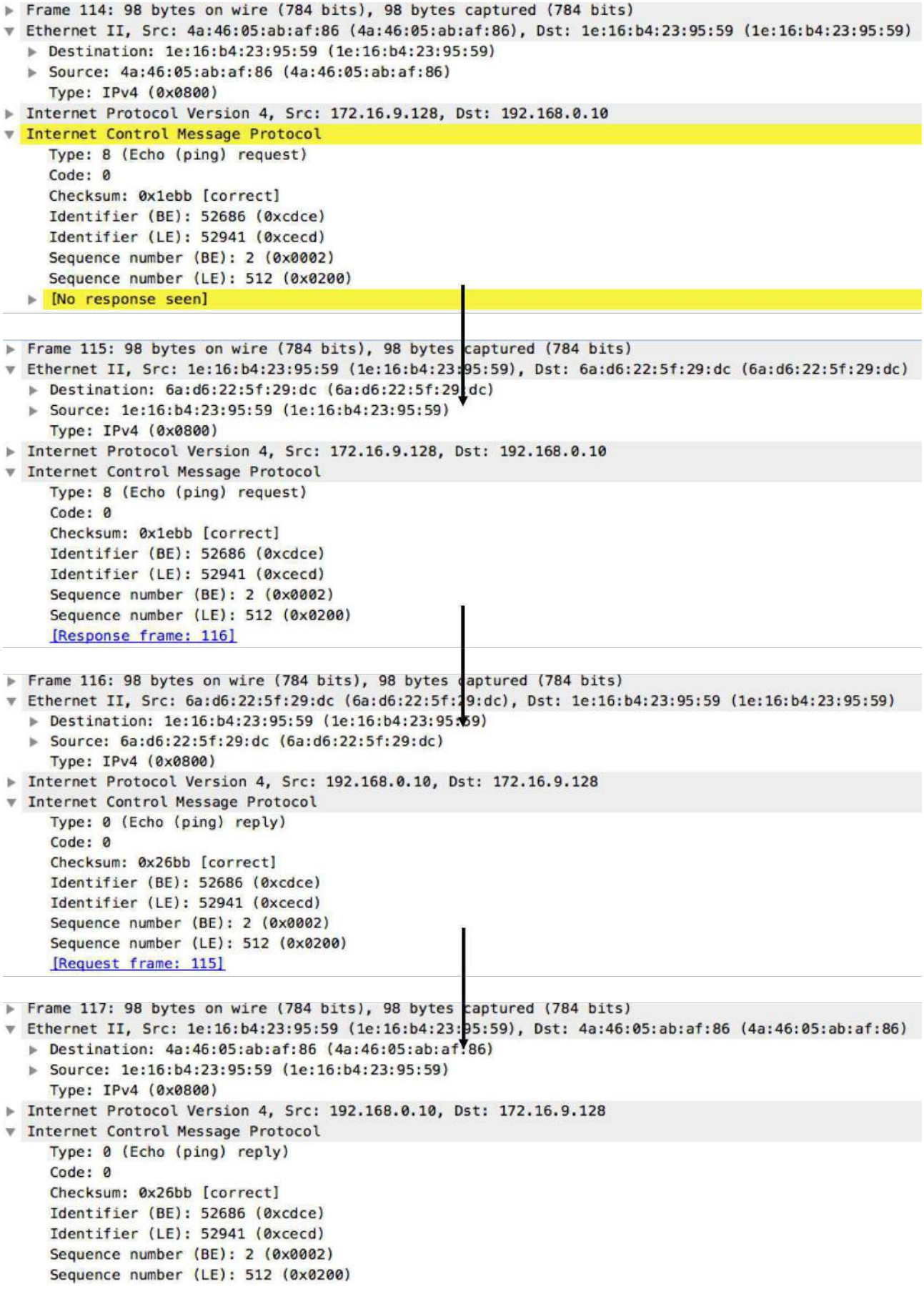


Figure 31: A dissection of three pcaps showing the MITM machine relaying packets to its target

When analysing these ICMP packets, it is important to understand the following:

- 4a:46:05:ab:af:86 = Host Machine
- 6a:d6:22:5f:29:dc = Client_1
- 1e:16:b4:23:95:59 = Mitm_1

The pcap above shows the following process:

1. The host machine sends a ping request packet to the MITM spoofed machine.
2. The spoofed machine then sends a ping request to the original client_1 machine.
3. Client_1 responds to the ping sent from the MITM machine and sends an echo reply.
4. The MITM machine then sends a ping echo reply back to the host machine.

This process of intercepting then forwarding packets between the true source and destination machines allows for every packet to be inspected without having to run a promiscuous sniffing tool on an existing piece of hardware already on the network. The ping packets sent to the spoofed machine on subnet B follows the exact same intercept and forward process as shown in the pcaps above.

To stop the MITM machines spoofing the target machines, the ARP poisoning process was terminated using CRTL+C. Once all spoofing had stopped and the pcaps had been extracted from the host machine the next parts of the experiment could be conducted.

D.9.2.2 Ettercap Baseline Traffic Interception The next Ettercap experiment involved simulating regular internet traffic as opposed to ICMP echo packets. The same MITM machines were used and the commands had the same syntax as in the previous experiment.

"Ettercap -T -w /hosthome/pcaps/Ettercap/mitm12.pcap -M ARP /192.168.0.10/ // output:"

"Ettercap -T -w /hosthome/pcaps/Ettercap/mitm22.pcap -M ARP /192.168.1.1/ // output:"

The only changes made to the commands was the name of the pcap being saved on the host machine, in order to represent each unique experiment. Each command was executed in the MITM machines and output began to appear on each, confirming that the ARP poisoning had been successful once again. In order to generate traffic, a request to both the FTP and HTTP server was made from the original client_1 machine. This means that Ettercap's ability to forward packets correctly and mimic the services of the servers being poisoned would be tested. The following pcaps show the traffic being intercepted from the mitm_2 machine:

No.	Time	Source	Destination	Protocol	Length	Info
34	24.208470	192.168.0.10	192.168.1.1	TCP	74	39668 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=50881 TSecr=0 WS=32
35	24.209167	192.168.0.10	192.168.1.1	TCP	74	[TCP Out-Of-Order] 39668 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=50881 TSecr=0 WS=32
36	24.209571	192.168.1.1	192.168.0.10	TCP	74	80 → 39668 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=53136 TSecr=50881 WS=2
37	24.209632	192.168.1.1	192.168.0.10	TCP	74	[TCP Out-Of-Order] 80 → 39668 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=53136 TSecr=50881 WS=2
38	24.216159	192.168.0.10	192.168.1.1	TCP	66	39668 → 80 [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=50882 TSecr=53136
39	24.216407	192.168.0.10	192.168.1.1	TCP	66	[TCP Dup ACK 38#1] 39668 → 80 [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=50882 TSecr=53136
40	24.219896	192.168.0.10	192.168.1.1	HTTP	660	GET / HTTP/1.1
41	24.220472	192.168.0.10	192.168.1.1	TCP	660	[TCP Retransmission] 39668 → 80 [PSH, ACK] Seq=1 Ack=1 Win=5856 Len=594 TSval=50884 TSecr=53136
42	24.221842	192.168.1.1	192.168.0.10	TCP	66	80 → 39668 [ACK] Seq=1 Ack=595 Win=6900 Len=0 TSval=53136 TSecr=50884
43	24.223175	192.168.1.1	192.168.0.10	TCP	66	[TCP Dup ACK 42#1] 80 → 39668 [ACK] Seq=1 Ack=595 Win=6900 Len=0 TSval=53136 TSecr=50884
48	24.305769	192.168.1.1	192.168.0.10	HTTP	480	HTTP/1.1 200 OK (text/html)
49	24.305881	192.168.1.1	192.168.0.10	TCP	480	[TCP Retransmission] 80 → 39668 [PSH, ACK] Seq=1 Ack=595 Win=6900 Len=414 TSval=53136 TSecr=50884
50	24.307800	192.168.0.10	192.168.1.1	TCP	66	39668 → 80 [ACK] Seq=595 Ack=415 Win=6912 Len=0 TSval=50892 TSecr=53136
51	24.307861	192.168.0.10	192.168.1.1	TCP	66	[TCP Dup ACK 50#1] 39668 → 80 [ACK] Seq=595 Ack=415 Win=6912 Len=0 TSval=50892 TSecr=53136
56	39.210804	192.168.1.1	192.168.0.10	TCP	66	80 → 39668 [FIN, ACK] Seq=415 Ack=595 Win=6900 Len=0 TSval=54637 TSecr=50892
57	39.211015	192.168.1.1	192.168.0.10	TCP	66	[TCP Out-Of-Order] 80 → 39668 [FIN, ACK] Seq=415 Ack=595 Win=6900 Len=0 TSval=54637 TSecr=50892
58	39.243096	192.168.0.10	192.168.1.1	TCP	66	39668 → 80 [ACK] Seq=595 Ack=416 Win=6912 Len=0 TSval=52386 TSecr=54637
59	39.243167	192.168.0.10	192.168.1.1	TCP	66	[TCP Dup ACK 58#1] 39668 → 80 [ACK] Seq=595 Ack=416 Win=6912 Len=0 TSval=52386 TSecr=54637
64	44.312699	192.168.0.10	192.168.1.1	TCP	66	39668 → 80 [FIN, ACK] Seq=595 Ack=416 Win=6912 Len=0 TSval=52894 TSecr=54637
65	44.312777	192.168.0.10	192.168.1.1	TCP	66	[TCP Out-Of-Order] 39668 → 80 [FIN, ACK] Seq=595 Ack=416 Win=6912 Len=0 TSval=52894 TSecr=54637
66	44.312858	192.168.1.1	192.168.0.10	TCP	66	80 → 39668 [ACK] Seq=416 Ack=596 Win=6900 Len=0 TSval=55147 TSecr=52894
67	44.312924	192.168.1.1	192.168.0.10	TCP	66	[TCP Dup ACK 66#1] 80 → 39668 [ACK] Seq=416 Ack=596 Win=6900 Len=0 TSval=55147 TSecr=52894

Figure 32: A pcap segment showing how Ettercap relaying HTTP packets to its target

Here, the data from the pcaps shows that the MITM machine is using the same intercept and forwarding system with a TCP stream. Each green packet represents data being received by the MITM machine. Each black packet represents data being sent from the MITM machine. The pattern above shows that every time the MITM machine receives data, the next packet always shows the MITM machine sending or "relaying" that same data it just received to another recipient, be this the client machine who made the original HTTP request or the HTTP server itself. From analysing the data provided by the MITM capture, not only can the services and active machines be identified, but the MITM machine is exposed to all the data flowing between the client and the server (in this scenario). As the NIC of the MITM machine has been set to promiscuous mode, it is also able to detect other traffic from the same subnet. It appears that Ettercap is also unable to identify the gateway devices and the possible services they provide.

Once this pcap had been analysed the virtual network was then shut-down along with the host virtual machine. This concludes the experiments on the IP network.

Appendix E Testing of Network Scanners Against SCADA Devices

E.1 Objectives

To analyse the effects of using IP network scanners on SCADA devices such as Programmable Logic Controllers (PLCs), a test environment containing a Human Interface Machine (HMI) and a PLC needs to be created in order launch the network scanning tools against the SCADA devices. The aim of this setup is to provide an insight into the possibility of gaining information about SCADA devices through the use of network scanners and sniffers. A host machine will be responsible for performing the scans against two separate types of PLC. This will ensure that all the available communication methods for SCADA devices will be tested against. The interactions between these two devices will be captured using a promiscuous network scanner and will be analysed in detail.

E.2 Hypotheses

The following hypotheses will be repeatedly tested throughout all the experiments referenced within this document.

Operational Hypothesis: H_1
The use of current active or passive IP network probers and sniffers will have a negative effect on the normal behaviour of non-IP specific devices.

Null Hypothesis: H_0
The use of current active or passive IP network probers and sniffers will have no effect on the normal behaviour of non-IP specific devices.

E.3 Variables

Independent Variable: The execution of a network scanner/sniffer on a network.
Dependent Variable: The behaviour of the non-IP devices.

E.4 Requirements

This section outlines the tools and resources needed in order to successfully conduct each experiment.

E.4.1 Siemens SIMATIC S7-1200 PLC

The Siemens S7-1200 is a compact PLC which allows for extendable functionality through the use of additional I/O components. The S7-1200 comes standard with a PROFINET/Ethernet interface, but additional input types can be added to the PLC through the use of external modules. Such modules include serial ports for analogue communications or additional microprocessors or I/O pins. These devices also support access to a HTTP web server using an IP address which can reached by external machines and configured within the TIA Portal. This device will be responsible for the operation of the field devices connected to this small SCADA system.

E.4.2 Human Machine Interface: Siemens SIMATIC KTP400 Basic HMI

The KTP400 is a 4-inch touch screen panel which can be connected to the output pins of the S7-1200 PLC. This small HMI simply allows for basic operator control of other field devices on the SCADA system as well as performing monitoring tasks. It can perform tasks independent of the main control centre and can be reconfigured using the TIA Portal

E.4.3 ASEA Brown Boveri (ABB) PM564 PLC

The ABB PM564 is a compact programmable logic controller which allows for the control and processing a multiple types of field devices through its AC500-eCo CPU. Similar to the Siemens S7-1200 PLC, the PM564 has a modular structure, on which more I/O, storage and processing devices can be attached in order to expand its capability. The PLC comes equip with two network interface cards, an Ethernet port and a serial COM1 port which can support such protocols as Modbus, CAN and PROFIBUS. This shall be used as another type of SCADA device to be scanned against.

E.4.4 Field Device: On-board Modular Motor

The motor being used is a bidirectional motor which is acting as a basic field device for these experiments. The power to the motor is supplied by the 24V power supply connected to the main printed circuit board (PCB). The speed of the motor can be set via a knob located on the PCB. In order to function, the motor must be connected to the PLC with a complete electrical circuit.

E.4.5 Compact Flexible Process Line

This field device consists of several moving components which control a small conveyor-belt system as well as a rotating tool used to mimic an industrial process. This device also comes equip with sensors so that the progress of items travelling across the process line can be monitored. The whole system is connected to the PLC as an external module via ribbon-cable located above the on-board motor referenced above.

E.4.6 IKH Didactic Systems PLC Trainer 1200

These large PCBs allow for external devices to be connected to and controlled by both the Siemens S7-1200 and ABB PM564 PLCs which can be mounted to the metal bar which spans across the right side of the board. This will act as the connection between the PLC and the field devices being used within each experiment.

E.4.7 Windows 7 64bit with Siemens Totally Integrated Automation (TIA)

The initial stages of configuring this environment involved specifying the operating system which would conduct each of the scans. The first option would be to use a Linux-based system which replicated the virtual machines used within the IP experiments. This however would not be suitable as the software suite used to push the S7 logic code onto the Siemens devices (TIA Portal) and the ABB devices (ABB Control Builder Plus) is not supported on that particular OS. As this was not a viable option, there was a Windows 7 machine available which already had the relevant software installed and configured. The choice to use Windows is broken down as follows:

- The TIA Portal is a framework developed by Siemens in order to provide a versatile platform on which users can perform control programming, network configuration, device configuration and system diagnostics all from the same utility. The TIA Portal allows users to switch SCADA components between online and offline states, allowing for new code to be commissioned and downloaded to any particular device present on that network. This software suite will host the logical code which will dictated the activity of the field devices within these experiments, as well as acting as an interface between the host machine and the SCADA hardware. As this software is exclusive to Windows machines, a Windows machine will be setup and configured to perform each test.

- One of the PLCs available for testing is part of Siemens' SIMATIC S7 range, meaning that the code being executed on these devices is programmed by Step7 software. This emphasizes the need for the TIA Portal and the Windows host.

- Windows is an operating system which supports a number of the tools used within the IP experiments conducted on the Netkit network. However, Windows does not yet support the use of Zmap, the WAN scanner used in the previous IP experiments. Although this means that the WAN scanning capability is not able to be tested against the SCADA devices, during the execution of the IP tests, it was noted that both Zmap and Nmap function in very similar ways, despite the use of ICMP to detect hosts on a network differs, meaning that the choice to exclude Zmap from testing will not drastically impact the results of this experiment. Windows also supports Tshark, which will be the tool used to capture the traffic between the host machine and the PLCs.

E.4.8 Windows 7 64bit with ABB Control Builder Plus & CoDeSys 2.2.0

In order to execute the control code of the ABB PLC, the ABB Control Builder needed to be present on the host machine. The control builder allows users to develop logic ladders and control code which can then be compiled, error checked and then downloaded onto a series of ABB automation products. As one of the test PLC being used within these experiments is an ABB PM564 PLC, this software suite will ensure that compatible and stable code can be executed on that particular device. The choice to use a Windows 7 operating system follows on from the requirements set by the Siemens PLC. At the time these experiments were conducted, there only operating system capable of running the Control Builder Plus tool was Windows.

E.4.9 Laptop Running a Linux-Based Operating System (Ubuntu 16.04 LTS)

A Linux laptop is needed for these experiments in order to conduct the Zmap scanning experiments. At the time of these experiments there was no stable version of Zmap available for Windows 7 machines.

E.5 Design and Setup

This section provides details about the design of the SCADA experiments and configuration of the hardware and software.

E.5.1 Siemens S7-1200

The S7-1200 allows for a SIMATIC KTP HMI touch-screen panel to be integrated into the SCADA system. This can be used for basic operator control and monitoring tasks. Both of these Siemens devices are fully configurable through the use of the TIA Portal and can be connected together or setup independently via TIA. The HMI can be used to represent a field device as it can be directly connected to the output pins on the micro PLC (S7-1200). The following figure shows the S7-1200 PLC as well as the board housing the PLC and field devices:

(a) The Siemens S7-1200 PLC with the motor and HMI connection outlined

(b) The S7 PLC and HMI screen

Figure 33: The Siemens S7-1200 PLC and HMI setup

Here, the S7-1200 has been connected to a large PCB which provides a connection to multiple field devices. The highlighted areas show the field devices which will be monitored throughout these experiments. This includes the bidirectional motor and the connection to the KTP HMI. These figures also show the HMI attached to the PLC via the PCB shown above. The HMI is displaying a system monitoring UI which has been pushed to the device via TIA Portal.

With the PLC attached to the PCB, there needs to be a connection between the PLC and the Windows machine running both the TIA suite as well as the network scanning and sniffing tools. As referenced earlier, the S7-1200 comes with a PROFINET/Ethernet interface as standard, without the need for any expansion or additional modules. With an Ethernet cable connecting the PLC and the host Windows machine, the following figure shows the entirety of the SCADA setup:

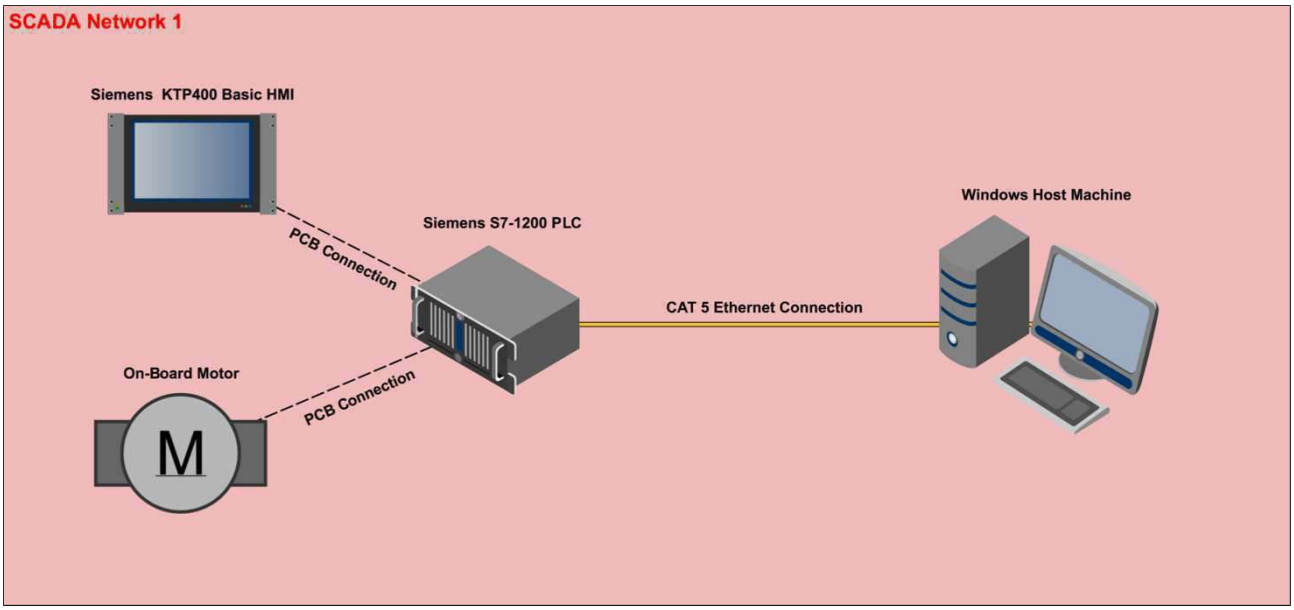


Figure 34: A topology showing the Siemens S7-1200 PLC network setup

On execution of each scan the motor will be running at a medium speed, whilst the HMI is also connected to the PLC, in order to detect any changes to the display or possible loss of connection. The choice to connect both a motor and a HMI device to the PLC was made so that a realistic setup could be tested against. Having both a HMI and a physical moving device gives a true representation of the different types of output devices which may be present on an operational SCADA network. Due to the resources available at the time of the experimentation, the experiment does not facilitate the testing of sensors which may exist within modern SCADA systems.

E.5.2 Running the Step7 Code

In order to ensure that the PLC had operational code running during the course of each scan, a program needed to be created and downloaded to the PLC before any experiments commenced. Firstly, a new Step7 project was created using the TIA Portal. On opening the TIA Portal, the S7-1200 PLC was connected to the host machine using a CAT 5 Ethernet cable. From there, the PLC could be detected using the Portals hardware detection service. In order to gain all the relevant information about the PLC used within these experiments, the interface used to detect the PLC was set to “eth0”. Once the discovery interface had been defined, the host machine began to search for any PLCs or HMIs sitting on the local network. Once the S7-1200 had been found, a graphical interface allowed for the configuration of not only the logic behind the physical I/O pins, but also the details about the devices network address and any associated modules. This graphical interface allowed for a block of control code to be written and assigned to the addresses of the physical pins located on the PLC. The figure below shows the graphical representation of the S7-1200 PLC and the logic assigned to several of the I/O pins:

Figure 35: A figure showing a section of the S7 control code on the TIA Portal

The logic shown in the figure above provides the necessary code which will allow the small field device (motor) to be turned on and off via the PLC. Here, the input pin at address "%I0.0" has been assigned to control the state of the device behind pin "%Q0.3" as well as the device behind "%Q0.1". The combination of having both "%Q0.3" and "%Q0.1" in an ON state in turn causes the motor to spin. This code will be pushed onto the PLC before every scan is executed.

The following figure shows the same pins on the physical device:

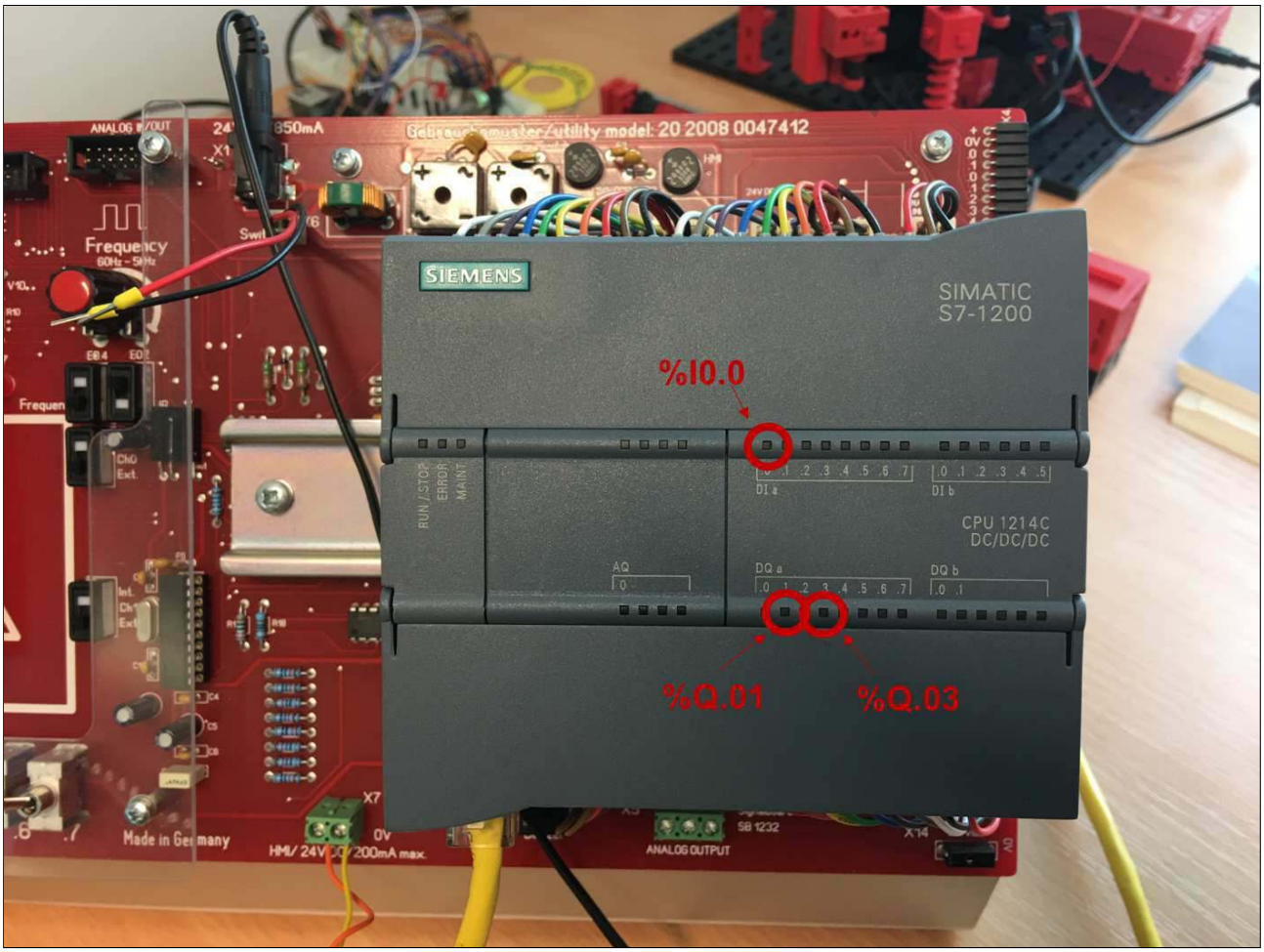


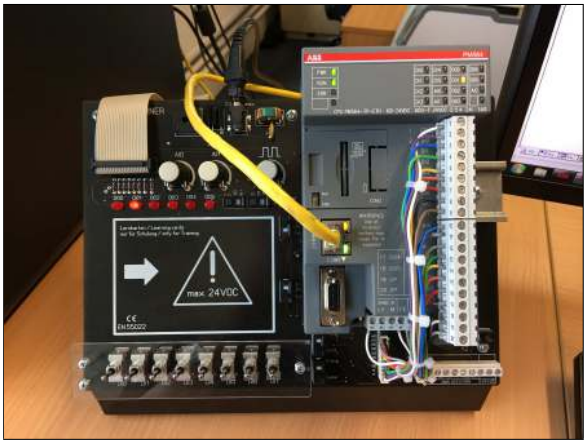
Figure 36: The I/O ports on the Siemens PLC being used to control the motor field device

Once this logic had been created, it was then pushed onto the device. In order to do this, the host machine

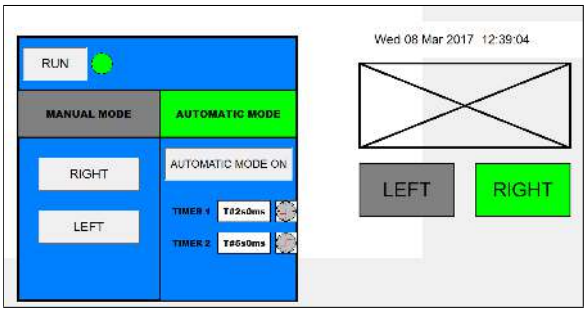
established a connection with the PLC using TCP port 102. Once this connection has been made, the logic is then downloaded locally to the PLC. This means that the functionality of the PLC should not alter until changes are pushed to, and then downloaded to the device. In order to test that the logic code has successfully uploaded, switch zero was activated, which in turn caused the motor to power-up. This accompanied by several green LEDs above each pin confirmed that a signal was being passed through the PLC to the field device (motor).

E.5.3 ABB PM564 PLC

The ABB PM564 has been mounted onto a PCB which allows for different field devices to be connected to the PLC's logical processor. Unlike the Siemens S7 setup, this SCADA system does not have an integrated motor which can act as a field device. However, this device has been configured so that the host machine can act as a fully-functional HMI, allowing the user to control the actions of the PLC, and thus the field devices, through the use of an interface (see Figure 37). The original code was used to control a remote, bidirectional fan. As the fan is not available, the actions of the field devices can be shown by a series of LEDs on the PCB. These will be monitored in order to deduce whether the behaviour of the field devices is being altered during the process of a network scan. The figures below show the physical setup of the ABB PLC system:



(a) The ABB PM564 PLC with an Ethernet connection



(b) The HMI interface configured to run on the host machine

Figure 37: The ABB PLC and HMI setup

The ABB PLC has been connected to the host Windows 7 machine via a CAT5 Ethernet connection, the same method used on the S7 device. This connection allowed for the HMI software to interact with the PLC once the logic code had been downloaded and executed. As seen in the two figures above, the active LED on the PCB corresponds with the "Right" directional button located on the HMI interface. A full representation of this network is shown within the following topology:

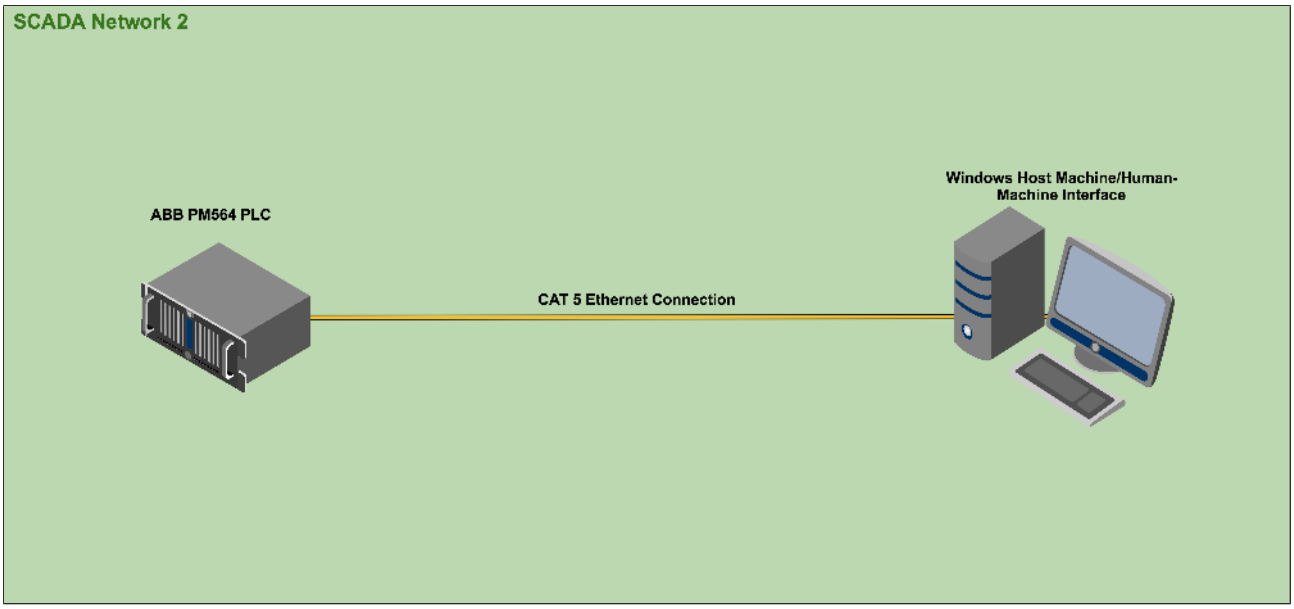


Figure 38: A topology showing the ABB PLC network setup

When each network scan is executed, the HMI will be switched to run in its "automatic mode". Once set, the PLC will alternate the direction of the field devices which will be represented by LEDs "D00" and "D01" (see Figure 37). Due to the resources available at the time of this experiment, no further field devices will be connected to the ABB and therefore will not be exposed to any form of network scanning.

E.5.4 Running the ABB Control Code

The logic code which will be ran on the ABB PLC during each experiment was written in ABB's "CoDeSys" format and compiled within ABB Control Builder Plus. The code (subsubsection E.5.4) allows for the user to control a remote bidirectional fan connection to the PLC via a HMI front-end present on the host machine (Figure 37). The code is pushed to the PLC through the use of a TCP connection established by a host machine running the ABB Control Builder Plus software. Similar to the Siemens S7 setup, the ABB software compiles the code, creates a connection with one or multiple SCADA devices and then pushes the code onto the PLCs CPU. Once downloaded, the logic can then be ran either via the switches located on the PCB or through the HMI screen on the host device.

Variable	Mapping	Channel	Address	Type	Unit	Description
Digital + analog inputs						
		Inputs 0-7	%IB4000	BYTE		Digital inputs 0-7
I00_RUN		Input 0	%IX4000.0	BOOL		Input 0
I01_LEFT		Input 1	%IX4000.1	BOOL		Input 1
I02_RIGHT		Input 2	%IX4000.2	BOOL		Input 2
I03		Input 3	%IX4000.3	BOOL		Input 3
I04		Input 4	%IX4000.4	BOOL		Input 4
I05		Input 5	%IX4000.5	BOOL		Input 5
		Input AI0	%IX4000.6	BOOL		Digital input AI0
		Input AI1	%IX4000.7	BOOL		Digital input AI1
		Interrupt	%IB4001	BYTE		Interrupt
		Analog input 0	%IW2001	INT		Analog input 0
		Analog input 1	%IW2002	INT		Analog input 1
Digital + analog outputs						
		Outputs 0-5	%QB4000	BYTE		Digital outputs 0-5
		Analog output 0	%QW2001	INT		Analog output 0
PWM						
		State byte PWM 2	%IB4040	BYTE		PWM 2 - State byte
		State byte PWM 3	%IB4041	BYTE		PWM 3 - State byte
		Control byte PWM 2	%QB4040	BYTE		PWM 2 - Control byte
		PWM 2, frequency / cycle time	%QW2021	WORD		PWM 2 - Frequency / Cycle time
		PWM 2, duty cycle / duty time	%QW2022	WORD		PWM 2 - Duty cycle / Duty time
		Control byte PWM 3	%QB4046	BYTE		PWM 3 - Control byte
		PWM 3, frequency / cycle time	%QW2024	WORD		PWM 3 - Frequency / Cycle time
		PWM 3, duty cycle / duty time	%QW2025	WORD		PWM 3 - Duty cycle / Duty time
Fast counter						
		Actual value 1	%ID1015	DWORD		Actual value 1
		Actual value 2	%ID1016	DWORD		Actual value 2
		State byte 1	%IB4068	BYTE		State byte 1
		State byte 2	%IB4069	BYTE		State byte 2
		Start value 1	%QD1015	DWORD		Start value 1
		End value 1	%QD1016	DWORD		End value 1
		Start value 2	%QD1017	DWORD		Start value 2
		End value 2	%QD1018	DWORD		End value 2
		Control byte 1	%QB4076	BYTE		Control byte 1
		Control byte 2	%QB4077	BYTE		Control byte 2

Figure 39: A section of the ABB control code shown in ABB Control Builder Plus

When power is sent through output pin “D00” the field device spins anticlockwise and the left LED turns on. When power is sent through the “D01” pin, the field device spins clockwise and is accompanied by the right LED. The status of these LEDs will be used to monitor the behaviour of the possible field devices when each scan is being executed.

E.6 Obtaining the Results

Before each experiment is conducted, a connection between the TIA Portal/Control Builder Plus and the PLC will be established and the logic code referenced above will be downloaded to the device. The device will also be powered on and off before each experiment. Following these two prerequisites ensured that the code being executed during each scan would be the same throughout each experiment. Restarting the machine reverts any of the changes made by the execution of a network scanner. Once the PLC has been restarted and the logic downloaded, the field devices will be activated and the HMI will be connected to the PCB. Any changes made to either the PLC or any of the connected devices signifies the effects of the network scanners/sniffers. As well as observing the behaviour of the SCADA system, the passive packet capture tool Tshark will also be ran on the host machine in order to capture the traffic being sent from the host to the PLC. This has been done in order to provide technical explanations to any of the results yielded from the experiments.

E.7 Expected Outcomes

Using the information obtained from both the network technologies research as well as the results yielded from the IP scanning experiments, the following assumptions can be made about the results of the SCADA tests:

- As the scanners previously tested are aimed at gathering information about devices using IP protocols, executing these tools against PLCs which communicate using Ethernet should present similar data as the machines present on the virtual IP network. Enabling a web server on the S7-1200 or the ABB PM564 should return similar results to the HTTP server configured in the previous set of IP experiments.
- If the Ethernet connected devices do not communicate through the use of the TCP or UDP protocols, the packets being transmitted by the host machine will not be supported by the ports open on the target devices. This could result in either the attempted parsing of this traffic, which could alter the CPU usage of the PLC, meaning that the CPU may allocate more resources towards processing the data being received through the network over the existing control code being ran. The other possible outcome is that the PLC does not recognise the incoming TCP/IP data as a valid input, therefore the packets could be dropped by the target devices.
- The devices communicating on a serial network will be using a different set of layer 1 (physical layer) protocols such as Modbus, S7comm or DNP3. This means that the low-level network framing used by the scanners to send data will not be recognised through a serial/analogue connection. This could cause the same results as previously discussed: overloading the CPU or dropping the IP packets.
- Passive network sniffers do not require data to be sent to the target devices. Because of this, as long as the host machine and the target PLC are on the same local network, communicating via a peer-to-peer connection or a hub, all packets travelling across the network should be captured.
- Dependent on the passive sniffer being used, there may not be a capture engine or packet dissector which supports either a serial interface or the ability to capture the data travelling across it.

These assumptions will be taken into consideration when discussing the results drawn from all of the SCADA experiments.

E.8 Execution of Experiments: Siemens S7-1200

This section details the experiments conducted against the Siemens S7-1200 PLC.

E.8.1 Experiment 1: Nmap 2.8

Nmap was chosen for these experiments due to its reoccurring presence within multiple sources referenced within the literature review section of this project, as well as having a large set of capabilities for scanning different types of networks. The results obtained from the previous IP experiments provided an insight into how Nmap used TCP packets in order to gain information about the machines it targeted, and how these may not be suitable for use against a SCADA network.

E.8.1.1 Nmap Ping Sweep The first scan conducted against the SCADA system was a SYN scan, which aimed to identify active hosts on the local network. The same scan was conducted on the virtual IP network used in the previous set of experiments, on which it successfully identified the active IP addresses of each subnet.

In order to execute this scan, the logic code to control the field devices needed pushing onto the PLC. To do this, the TIA Portal needed to establish a connection with the PLC, otherwise known as the "online state". To do this, the PLC was selected from the device list within the TIA Portal. Once the device had been selected, the "go online" button located within the menu-bar became visible. Once pressed, the host machine connects to the remote device through the use of a TCP connection on remote port 102. Once the TCP connection has been established, the code can then be directly downloaded to the PLC and then ran once all the code has been received. The CPU within the PLC is then put into run mode and the code will begin to function. In order to test that the code had been successfully downloaded onto the PLC, the switch at address "%I0.0" was put into the on position. The motor began to spin, signifying the code was working.

Once the field device was active, the first Nmap scan was ran against the PLC. The syntax of the Nmap command was as follows:

"nmap -sP -r 192.168.0.* -e eth0 > nmap1.txt"

The "-sP" method of scanning sends TCP SYN packets to the range of IP addresses specified within the command. In this case, all the machines on the 192.168.0.0/24 subnet will be scanned. The "-r" flag instructs Nmap to scan the ports consecutively. The "-e" flag dictates which interface on the host device should be used to send each packet through. In this case, the eth0 interface represents the Ethernet interface connecting the host machine and the PLC by a single Ethernet cable. Lastly, the ">" operator sends all the output of the Nmap scan into a text file, the name of which represents each unique scan.

On execution of this scan, both the motor and the HMI remained stable. In addition to this, Nmap was able to complete its scan without any interruption or errors, providing a text file which contained the IP address of the PLC. The contents of this file are shown below:

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-02-18 11:48 GMT Standard Time
Nmap scan report for 192.168.0.54
Host is up (0.00s latency).
MAC Address: 00:1C:06:10:A8:5E (Siemens Numerical Control, Nanjing)
Nmap done: 256 IP addresses (1 host up) scanned in 7.07 seconds
```

Figure 40: The data contained within "nmap1.txt"

Here, the output file shows that Nmap was able to locate the PLC at address 192.168.0.54, which matches the configuration of the Ethernet interface which can be viewed on the TIA Portal. As well as supplying the IP address, Nmap is also able to identify the vendor of the device (Siemens Numerical Control, Nanjing, which is a Siemens factory) alongside the MAC address. Despite supplying this information, there is no overt method of deducing that the device is a PLC without external research or prior knowledge. The most significant element of this output file is the fact that Nmap was able to successfully scan the device without effecting the normal

operation of either the PLC or the field devices connected to it. To understand how Nmap was able to scan the PLC without altering its operation, the pcap from the host machine was opened in Wireshark in order to analyse the traffic being sent to the target device.

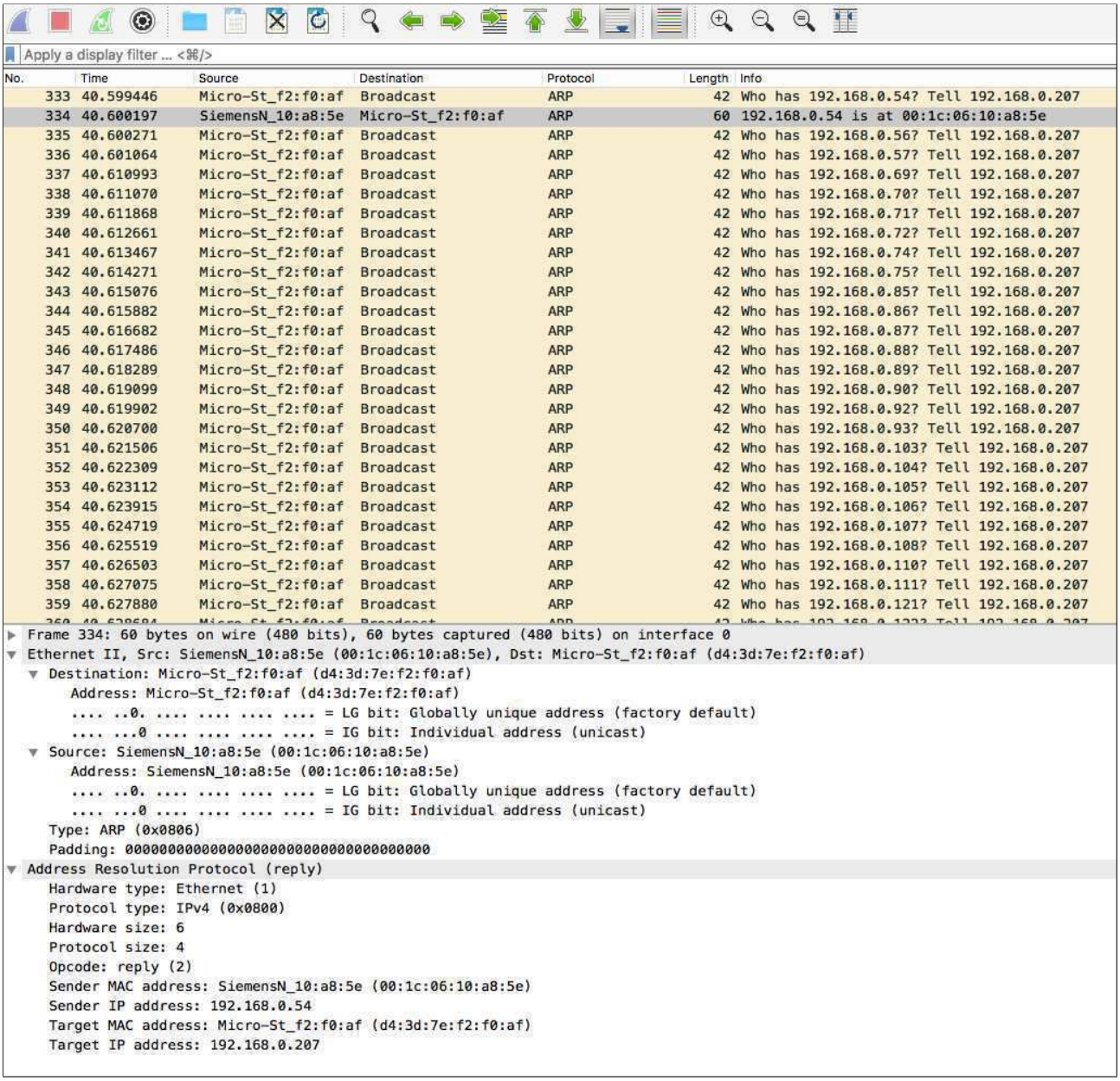


Figure 41: A pcap segment showing Nmap utilising the ARP protocol

Unlike the -sP scan performed on the virtual IP network in the previous experiment, Nmap utilised Address Resolution Protocol (ARP) broadcasting in order to detect hosts on the network. The ARP protocol runs on the top of Ethernet frames using the MAC addresses to associate IP addresses with each individual machine on the network. In this scan, the host machine sent multiple ARP broadcasts, where each packet would make a request for each unique IP address in the 192.168.0.0/24 network. The figure above shows the host machine requesting the MAC address associated with every possible IP address on that network. Once a request has been sent for 192.168.0.54, the PLC then responds with its own MAC address. Using this response, Nmap then uses a set of dissectors in order to distinguish the manufacturer of the device behind 192.168.0.54. This information is then presented within the output .txt file. From both the physical observation of the SCADA equipment as well as

the data presented by the Nmap output, there was no evidence to suggest that the scanning had a negative or damaging effect on the SCADA system.

As no changes had been made to either the network or the field devices, the logic code did not need re-downloading onto the PLC before the next set of experiments.

E.8.1.2 Nmap Service Detection Scan The next Nmap experiment aimed to detect the different services running on the PLC. On an IP network, this method of scanning sweeps all of the ports available on the target machine and tries to establish a TCP connection with as many as possible. Once a TCP connection has been established, the target machine often responds with a banner or data packet in order to share the configuration information with the host machine. This involves a greater amount of interaction with the target devices than the previous scan, meaning the PLC will need to receive and supply a larger amount of data. The Nmap command executed was as follows:

"nmap -sV -r 192.168.0.* -e eth0 > nmap2.txt"

This scan was conducted in order to gain as much information about the services being offered by the PLC. However, as the PLC only offers a single IP-based service, a HTTP server, the scan should not recognise any other services. This also suggests that any other ports found using Nmap will not be correctly configured to interpret or parse the incoming packets. The "-sV" flag instructs Nmap to begin probing any open ports in order to gain information about the services they are running. The "-r" flag instructs Nmap to scan each port on the target device consecutively. The address space being targeted within this experiment covers every device within a 182.168.0.0/24 subnet. The -e flag has been used to specify the interface on which the scanning traffic will be sent down, in this case the Ethernet interface "eth0" has been used. Finally, the ">" operator has been used to capture all of Nmap's output into a text file.

On execution of the scan, neither of the field devices being monitored changed their behaviour and continued normal operation. Nmap was able to complete the service detection scan without any issues or errors, this was confirmed by the creation of an output file which contained data relating to the SCADA network. This again shows that despite not having a wide range of TCP services, Nmap is still able to detect the presence of the PLC. The output file was then opened in order to analyse the information obtained from the scan, the content of the file was as follows:

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-02-18 11:48 GMT Standard Time
Nmap scan report for 192.168.0.54
Host is up (0.00s latency).
All 1000 scanned ports on 192.168.0.54 are filtered
MAC Address: 00:1C:06:10:A8:5E (Siemens Numerical Control, Nanjing)

Service detection performed. Please report any incorrect results at https://nmap.org/
submit/ .
Nmap done: 256 IP addresses (1 host up) scanned in 35.58 seconds
```

Figure 42: The data contained within "nmap2.txt" with filtered ports highlighted

The contents of this file show that although Nmap was able to detect that the PLC was present on the network, the scan was unable to disclose any information about any open ports or the services that they may run. Although a lack of information was expected when running a service detection scan, Nmap failed to discover a TCP port which is known to be present on the PLC, port 102 used to transport control code from the TIA Portal onto the PLC. No changes had been made to either the PLC or the host machine which would make this port unavailable, therefore Nmap's inability to detect that port was unexpected. Another notable aspect of these results is the output data highlighted in the figure above. Nmap declared that there were 1000 possible ports available on the PLC, however, all these ports have been filtered. To gain a better insight as to why these ports were filtered, the pcap taken from the host machine was opened in Wireshark. The figure below shows the packet exchange between the host and the PLC:

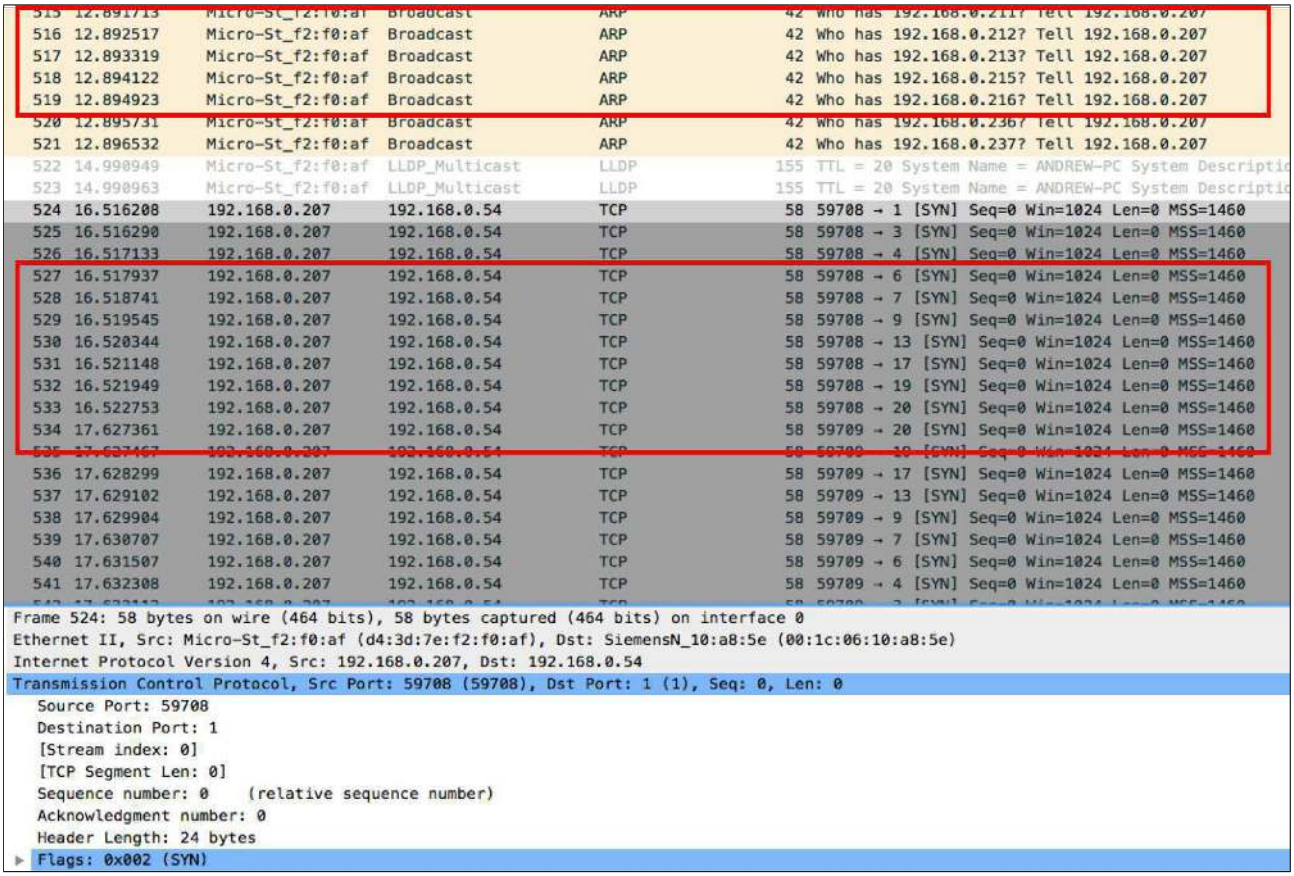


Figure 43: A segment of a pcap showing host detection and then a TCP-SYN scan

This figure shows that like the previous host-discovery scan, Nmap continued to use ARP in order to detect which IP addresses were active on the network. Using this method, Nmap was able to detect that the PLC at address 192.168.0.54 was an active device. This then allowed Nmap to start sending a TCP SYN packet to all the ports it deemed present on that device. Although Nmap was instructed to scan each port on the target device consecutively, some port numbers were not probed, including port 102. The reason for skipping these ports is unknown, but this could have been a significant factor when discussing the inability to detect any services on the PLC. The entirety of the pcap file contained TCP SYN packets sent from the host to the PLC without any form of response, which in-turn does not provide any data to explain how and why the ports on the PLC were filtered. Again this method of network scanning failed to disrupt any of the equipment from observation and the network continued to operate correctly once the scan had completed.

Again, similar to the previous experiment, as none of the equipment or data had been changed through the duration of this testing, the logic code did not need to be re-downloaded to the PLC.

E.8.1.3 Nmap UDP Scan As the previous two experiments relied on using TCP packets in order to gather information about the target devices, the next experiment utilized the User Datagram Protocol (UDP). Unlike TCP, UDP is a connectionless method of communication, meaning that data is sent to a port without having to perform a 3-way handshake to establish a connection. A UDP packet is simpler than a TCP packet, containing less data within the header and does not use sequencing in when sending and receiving data. A UDP packet can be represented as shown:

Source Port	Destination Port	Length	Checksum	Data
2 Bytes	2 Bytes	2 Bytes	2 Bytes	65,535 Bytes

Table 14: A breakdown of a UDP packet

This means that unlike a TCP connection, UDP will send data to the specified port regardless of the service running behind it, as there is no flow control or connection verification. The destination machine will be unable to slow down the amount of data being received if a UDP port is left open. As there are a large number of supposedly filtered ports available on the target PLC (as discovered from the previous experiment), sending UDP packets at a fast rate could cause the CPU within the PLC to allocate more time to processing the incoming UDP traffic rather than carrying-out the downloaded control code. In order to get Nmap to facilitate a UDP scan, the following command was executed:

"nmap -sU -sC 192.168.0.* -e eth0 > nmap_udp.txt"

The "-sU" flag instructs Nmap to use UDP in order to detect any open ports on the target devices. The "-sC" flag activates Nmap's script scan feature, on which Nmap will use a suite of Nmap Scripting Engine (NSE) scripts in order to extend the capability of Nmap and gain more information about the target devices. The default "-sC" flag used in this command executes a series of scripts which have been marked with the "default" tag. A list of the default scripts can be found at: `https://nmap.org/nsedoc/categories/default.html`. The reason for choosing this method of scanning was to add another element of intrusiveness to the scan, meaning more data about a large set of foreign protocols (to the PLC) would possibly be parsed by the device. The address range covers every device that could be configured on a 192.168.0.0/24 subnet and the "-e" flag directs the scan traffic through the "eth0" Ethernet interface. The output as always is piped to a text file using the ">" operator.

The scan was able to execute successfully and all of the arguments were valid. Unlike the previous SCADA and IP experiments, the UDP scan took a longer amount of time to complete. Throughout the extended duration of the UDP scan, none of the field devices changed in behaviour from an observational point of view. The scan was able to complete successfully without any errors and provided a populated output file. The figure below shows the result of the UDP scan and possible data to support why the PLC continued to operate normally.

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-02-22 15:25 GMT Standard Time
Nmap scan report for 192.168.0.54
Host is up (0.00s latency).
Not shown: 999 open|filtered ports
PORT    STATE SERVICE
161/udp open  snmp
| snmp-sysdescr: Siemens, SIMATIC S7, CPU-1200, 6ES7 214-1AG31-0XB0, HW: 1, FW: V.3.0.2,
SZVCDYH0027696
|_  System uptime: 3h29m46.80s (1258680 timeticks)
MAC Address: 00:1C:06:10:A8:5E (Siemens Numerical Control, Nanjing)

Nmap done: 256 IP addresses (1 host up) scanned in 76.14 seconds
```

Figure 44: The data contained within "nmap_udp.txt"

The information shown within this output file appears to be more informative than the information provided by both of the TCP scans used in the previous experiments. Not only did the UDP scan give detail such as the MAC Address and the name of the device's vendor, the UDP scan also revealed the make and model of the PLC, as well as the type of processor within the device and the current firmware it was running at the time of execution. This data was obtained through the probing of UDP port 161 which facilitates the Simple Network Message Protocol. This protocol is used to provide information about devices on an IP network, their current state as well communications channel on which a network administrator can send data to modify or assign a new value to the recipient device. In order to analyse how this method of data acquisition is executed, the pcap taken from the host machine was opened and analysed. In order to find the specific conversation, a filter was applied in Wireshark which only showed conversations using port 61 and port 161 (the ports that support SNMP). The filter entered is shown below:

"udp.port == 161 —— udp.port == 61"

This filter displayed the following packets:

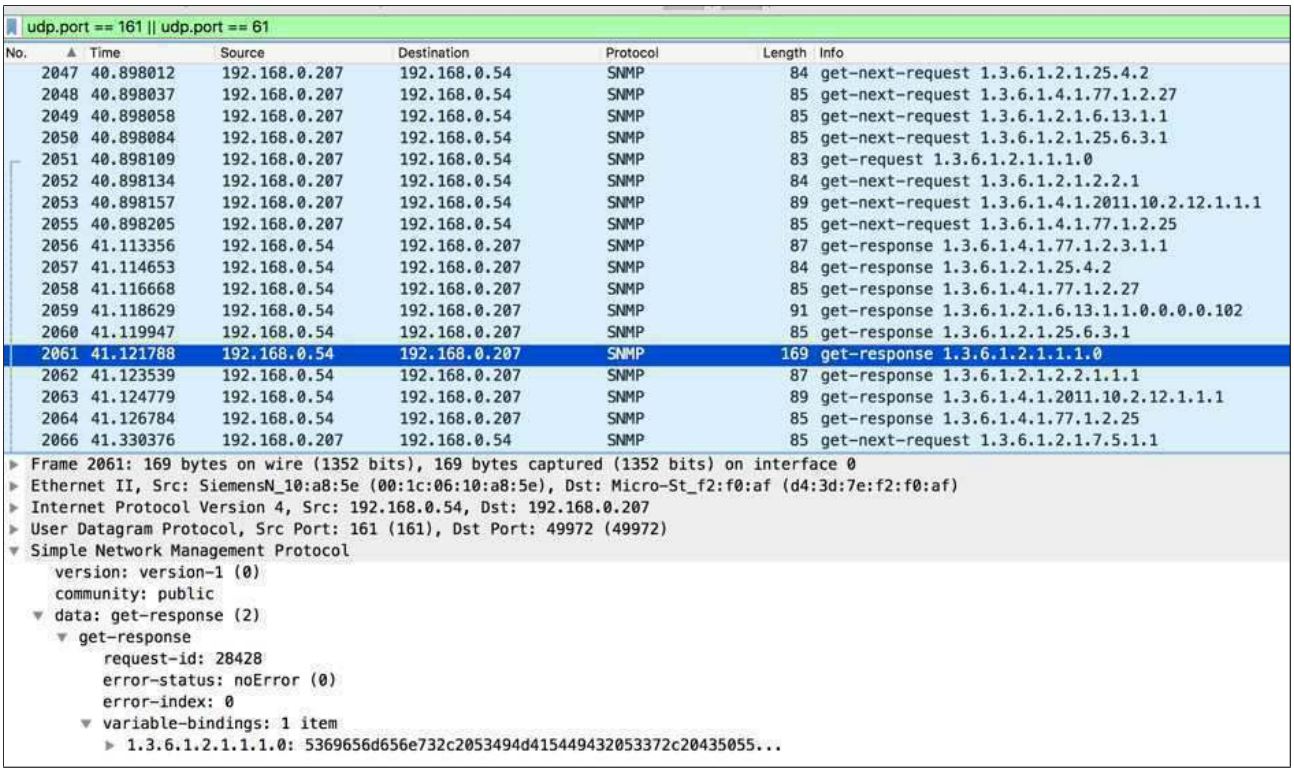


Figure 45: A segment of a pcap showing only SNMP packets

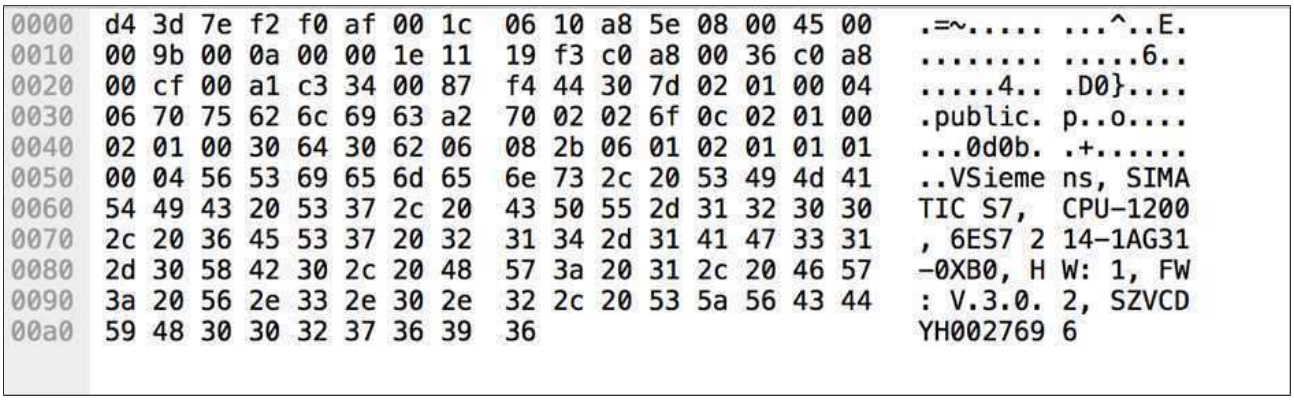


Figure 46: The raw data shown within a SNMP packet

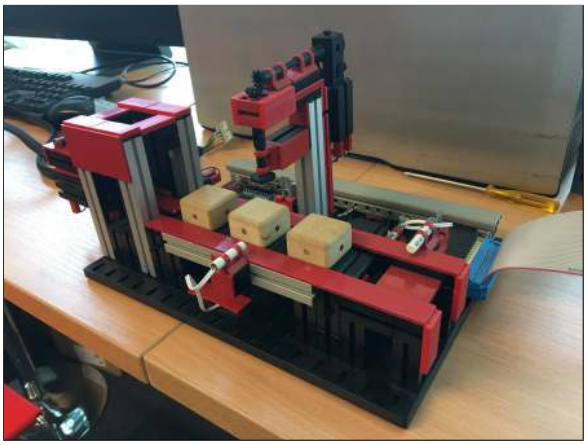
Here, the most notable traffic can be seen frame numbers 2051 and 2061. Here, the host machine makes an SNMP request to the PLC (2051) to which the PLC then replies with a string of data represented in hexadecimal form (2061). The line of hex can be seen above, under the "variable-bindings" data field. The figure below shows that if this hex string is converted into ASCII values, you are able to view the information sent from the PLC.

Figure 46 shows the in-built hex viewer displaying the ASCII decoded values of the data sent from the PLC to the host machine. The data supplied here cross-references with the information supplied in the output file.

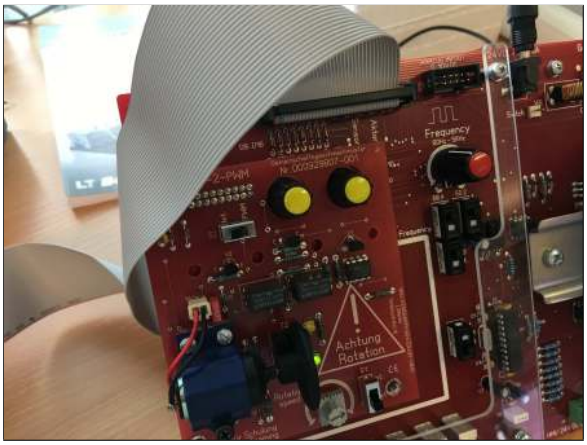
The choice to change the scan from TCP to UDP was unsuccessful in effecting the operation of the SCADA setup. Although a larger amount of data was sent to the PLC, the field devices connected did not alter their behaviour. This suggests that it is not the data being sent to the PC which could cause the issue, but instead,

the CPU of the PLC is not being overloaded with the both the network activity and the code being repeatedly executed. As PLCs run their control code within a real-time environment, any form of disruption to this code has real-time effects.

E.8.1.4 Nmap Conveyer-Belt Scanning As a result of the UDP scan proving to be ineffective at disturbing the operation of the SCADA network, a new hypothesis was formed (see below). Does the increase in field devices coupled with the execution of a network scanner impact the behaviour of the SCADA equipment? This hypothesis was formed due to the absence of data which suggested that network scanners have the capability to harm SCADA devices or whole SCADA networks. The current SCADA setup consists of a basic ladder-logic program which simply controls the state of a single motor. This type of program requires very little CPU usage, whilst the time-critical nature of this type of field device (small motor) is not as significant as other possible field devices such as sensors, actuators or centrifuges. In order to test this hypothesis, a new, more complex SCADA system must be setup and scanned. In order to elaborate on the existing setup, the following components were added to the PCB:



(a) The conveyer belt component



(b) The ribbon cable responsible for connecting the conveyer system

Figure 47: The conveyer-belt field device

This setup aims to investigate a new hypothesis: ***Does adding more devices to the PLC and thus executing more complex code have a significant impact in the systems behaviour when being targeted by a network scanner?***

This setup includes a larger amount of field devices which depend on time-critical code execution. The conveyor-belt system shown above provides an additional motor system responsible for placing boxes onto the conveyor, two sensors which monitor the progress of the boxes across the whole system, as well as a mock-drilling motor which spins for a set amount of time. Having all of these additional field devices, as well as the existing motor and HMI, requires more complex control code. This will result in the CPU having to allocate a greater amount of resources to processes the field device logic, rather than responding to network traffic. This conveyor-belt system was connected to the PCB via the bus located above the existing motor. Once attached, the pre-written logic for this setup was pushed to the PLC using the same method as the previous code, through the of the TIA Portal. Once this code had been downloaded and executed onto the PLC, all the motors began to run and the HMI displayed a monitoring screen. This represented the normal operation of the network and would be

observed during the course of each scan.

The first scan executed against the newly configured system was identical to the Nmap ping-sweep scan performed on the first SCADA setup. The command executed was as follows:

"nmap -sP -r 192.168.0.* -e eth0 > nmap_convyr_1.txt"

On execution of this Nmap scan, there were no changes to the operation of the new SCADA system. The scan was able to complete successfully without any errors and both a network capture and output file were created. On examining both the packet capture and output file, Nmap used the same TCP SYN scanning method as before, as well as supplying identical information to the previous ping-sweep experiment. This can be seen within the figure below:

47	4.216666	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.6? Tell 192.168.0.207
48	4.217467	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.7? Tell 192.168.0.207
49	4.218267	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.8? Tell 192.168.0.207
50	4.219074	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.9? Tell 192.168.0.207
51	4.219877	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.10? Tell 192.168.0.207
52	4.220679	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.11? Tell 192.168.0.207
53	4.221483	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.12? Tell 192.168.0.207
54	4.402887	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.1? Tell 192.168.0.207
55	4.418549	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.4? Tell 192.168.0.207
56	4.418642	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.5? Tell 192.168.0.207
57	4.419476	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.6? Tell 192.168.0.207
58	4.420279	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.7? Tell 192.168.0.207
59	4.421083	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.8? Tell 192.168.0.207
60	4.421889	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.9? Tell 192.168.0.207
61	4.422694	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.10? Tell 192.168.0.207
62	4.423499	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.11? Tell 192.168.0.207
63	4.424307	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.12? Tell 192.168.0.207

Figure 48: A segment of a pcap showing Nmap using the ARP protocol for asset detection

No further analysis was conducted as a result of this. As no changes had been made to the PLC, HMI or field devices, the code did not need to be re-downloaded onto the PLC.

The second scan executed against the conveyor-belt system replicated the service detection scan conducted within subsubection C.8.1.2. With a more complex set of logic being executed within the PLC and having attached more field devices, this scan aimed to investigate whether scanning a SCADA device controlling a larger amount of field devices causes the devices to become unresponsive or behave unexpectedly. The Nmap command executed was as follows:

"nmap -sV -r 192.168.0.* -e eth0 > nmap_convyr_2.txt"

As also specified within subsubsection C.8.1.2 of this document, the "-sV" flag instructs Nmap to perform a service detection scan using TCP packets to extract data from each of the ports present on any device located on the network. The "-r" flag ensures that each port on the target device is scanned consecutively, rather than randomly. Again the interface being used is the Ethernet card at eth0, which is specified using the "-e" flag. Finally, all output from this scan is written to a text file "nmap_convyr_2.txt".

On execution of this Nmap command, the scan completed successfully without any errors or malfunctions. From observation, the conveyor-belt system remained responsive throughout the full duration of the scan and neither the PLC or field devices appeared to be effected by the scan. On opening the output file, the data obtained

from this scan was identical to the data obtained from the previous service detection experiment. This can be shown within the following segment of the captured network traffic, as well as the text file generated from the above command:

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-03-07 10:10 GMT Standard Time
Nmap scan report for 192.168.0.54
Host is up (0.00s latency).
All 1000 scanned ports on 192.168.0.54 are filtered
MAC Address: 00:1C:06:10:A8:5E (Siemens Numerical Control, Nanjing)

Service detection performed. Please report any incorrect results at https://nmap.org/
submit/ .
Nmap done: 256 IP addresses (1 host up) scanned in 34.04 seconds
```

Figure 49: The data contained within "nmap_convyr_2.txt"

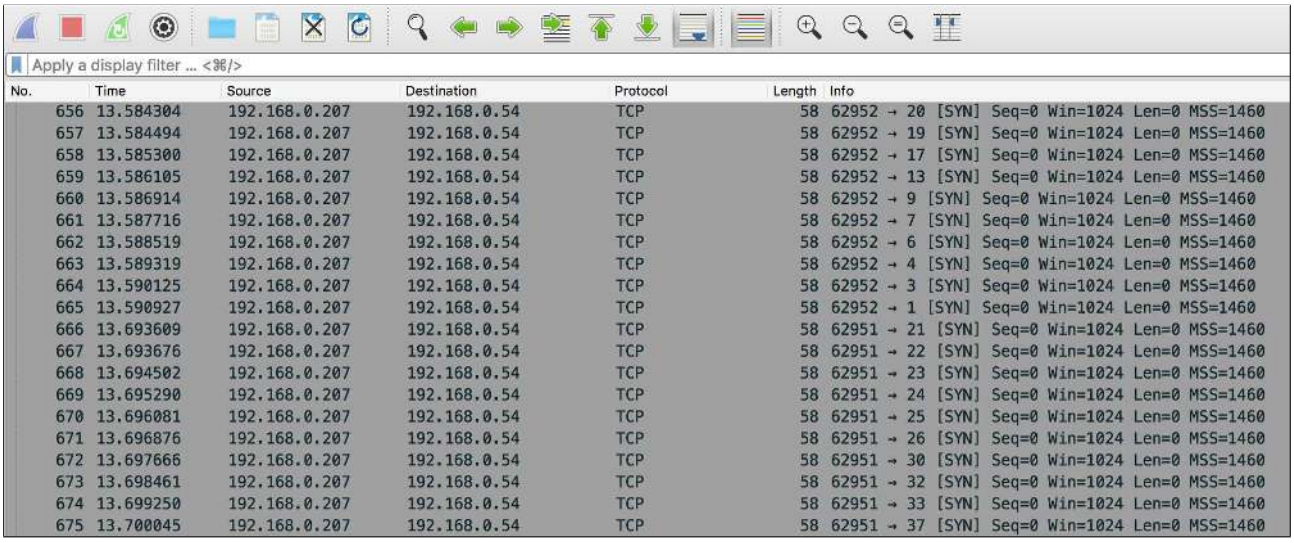


Figure 50: A pcap showing a TCP-SYN scan

The output file shows that Nmap was able to identify the Siemens S7 PLC at address 198.162.0.54 but, similar to the previous Nmap experiments, was unable to detect any services running on the network. The pcap file shows the same method of TCP scanning used within the previous experiments. Conducting a scan on a larger SCADA system did not show any changes in behaviour or alterations from normal operation.

The final Nmap experiment conducted against the conveyor-belt system focussed on scanning all the UDP ports present on the PLC. Similar to previous experiments, this scan should test whether sending streams of UDP data to each port on the target device could have negative effects on a SCADA system. The UDP scan command is shown below:

"nmap -sU -sC 192.168.0.* -e eth0 > nmap_convyr_3.txt"

As stated within subsubsection C.8.1.3 of this document, in order to initialise the UDP scanning functionality, the "-sU" flag has been set. Following this, in order to utilise a more intrusive form of scan, the "-sC" flag was used. This flag executes the default set of Nmap nse scripts found within the Nmap default directory. The Ethernet interface eth0 was selected using the "-e" flag. Finally, the output of this Nmap scan will be written into an output file through the use of the ">" operator.

On entering the command, the scan was able to run to completion without any errors and an output file was created and populated successfully. Through observation of all of the field devices and the PLC, the SCADA network did not appear to be affected by the UDP scan as normal operation persisted through the entire duration of the scan. The following output file shows that the Nmap UDP scan was able to identify the Siemens PLC without disrupting the operation of the system. The following figure shows the data Nmap was able to obtain:

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-02-22 15:25 GMT Standard Time
Nmap scan report for 192.168.0.54
Host is up (0.00s latency).
Not shown: 999 open|filtered ports
PORT    STATE SERVICE
161/udp open  snmp
| snmp-sysdescr: Siemens, SIMATIC S7, CPU-1200, 6ES7 214-1AG31-0XB0, HW: 1, FW: V.3.0.2,
SZVCDYH0027696
|_  System uptime: 3h29m46.80s (1258680 timeticks)
MAC Address: 00:1C:06:10:A8:5E (Siemens Numerical Control, Nanjing)

Nmap done: 256 IP addresses (1 host up) scanned in 76.14 seconds
```

Figure 51: The data contained within "nmap_convyr_3.txt"

The results obtained from this experiment match those yielded from the previous UDP experiment shown within subsubsection C.8.1.3. Unlike the series of TCP scans performed against the SCADA system, the UDP scan provides more specific details about the PLC present on the network. The figure above shows that coupling the UDP scan with the execution of Nmap's default scripts exposes significant data such as make, model, firmware version and CPU without causing the system to crash or act unexpectedly. Conducting a UDP scan on a larger, more resource intensive network appears to have no impact on the operation of the SCADA system. As no changes had been made to the network from the previous experiment, the control code did not need to be re-downloaded onto the PLC and the field devices could remain active. The network was ready for the next set of experiments.

E.8.2 Experiment 2: Zmap 2.1.0

Due to its IPv4 asset detection capability and its similarity with the technology behind the Internet of Things (IoT) scanner Shodan, Zmap provides an insight into how mass-internet scanners could possibly affect the devices which control SCADA/ICS systems. The results obtained from the previous IP experiments revealed that unlike Nmap, Zmap utilises the ICMP protocol in order to perform its asset detection scans. As well as this, Zmap has been configured and optimised to function on WAN IP networks, therefore these experiments aim to address how this technology interacts on a SCADA system.

The first Zmap scan aims to test the PLC against a mass ICMP asset detection scan. These types of scans could be used by either sys admins or possible malicious actors in order to assess which addresses represent an active host on a larger scale than a LAN or corporate network. This method of scanning does not require the host and target devices to establish a connection nor does it require large streams of data to be sent across the network. The first Zmap scan executed against the SCADA setup was as follows:

"sudo zmap -M icmp_echoscan 192.168.0.0/24 -o scada_zmap1.csv"

In order to execute this scan, a Linux laptop had to be connected to the SCADA network as there is not currently supported version for Windows computers. Once the Linux system had been connected to the network via a CAT5 Ethernet cable straight to the PLC, the command could then be executed. In order to run the Zmap scan, the tool had to be ran with elevated privileges. This was achieved by prepending "sudo" to the beginning of the command. On constructing the Zmap command, the "-M" flag specifies which probe module Zmap should run on execution. The probe chosen for this experiment is the "icmp_echoscan". From the data obtained by the IP experiments (see subsection D.8), this method of scanning sends ICMP "echo" packets to each one of the machines within the address-range specified within the Zmap command. Once each individual echo packet has been sent, Zmap then listens for any ICMP echo responses. These responses represent the active hosts. The address-space being scanned within this experiment is the 192.168.0.0/24 subnet. Finally, the "-o" flag instructs Zmap to write its output the a .csv file, which has been named to correspond with each unique scan.

On execution of this command, Zmap appeared to run correctly as no errors were reported on the terminal screen and an output file was successfully created. On inspection of the output file, Zmap had been unable to locate the PLC residing on the small network. The pcap file captured from the host machine was then opened and analysed in Wireshark in order to deduce why the scan had failed. The following figure shows the traffic being sent across the network:

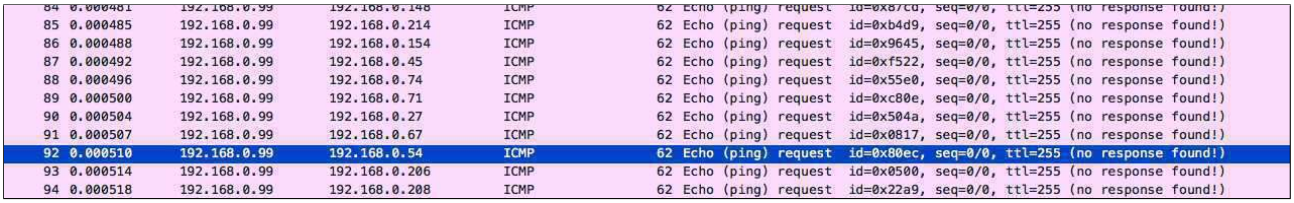


Figure 52: A section of the pcap file showing the transmission of ICMP packets

Here, as expected, Zmap sent ICMP packets to every possible host within the 192.168.0.0/24 subnet in order to deduce which addresses were active. The highlighted packet shows that Zmap was able to send an ICMP packet to the PLCs IP address successfully. However, there is no data within this pcap which shows that the PLC responded to this ICMP request, or that any other method of communication was used in order to respond to the scan.

E.8.3 Experiment 3: Custom UDP Scanner

The final test conducted against the Siemens PLC aimed at expanding on the use of UDP scanners, not to gain information about the target device, but to try and disrupt or disable the PLC through a denial of service attack against the SCADA network. To do this, a custom Python script was constructed. The aim of this script would be to simply connect to a user-specified IP address and continuously send large UDP packets to every possible port on the target device until instructed to stop. A copy of this script, as well as documentation as to how it functions can be found within Appendix G. In order to execute this script, the following command was entered into the host machine:

"python udp_dos.py"

This command allows python to compile and run the code contained within the "udp_dos.py" script (see Appendix G).

This tool was executed and ran continuously against the PLC for 5 minutes. During that time, there appeared to be no change in behaviour of both the field devices or the PLC. In order to ensure that the UDP scanner was functioning correctly and was able to identify the target machine, the pcap taken from the host machine was opened and analysed within Wireshark. The following pcap shows the traffic generated from the python script:

8	5.765925	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=51da) [Reassembled in #12]
9	5.765926	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=51da) [Reassembled in #12]
10	5.765927	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=2960, ID=51da) [Reassembled in #12]
11	5.765929	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=4440, ID=51da) [Reassembled in #12]
12	5.765930	192.168.0.207	192.168.0.54	UDP	672 55489 → 10 Len=6550
13	5.766938	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=51db) [Reassembled in #17]
14	5.766939	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=51db) [Reassembled in #17]
15	5.766940	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=2960, ID=51db) [Reassembled in #17]
16	5.766941	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=4440, ID=51db) [Reassembled in #17]
17	5.766942	192.168.0.207	192.168.0.54	UDP	672 55489 → 11 Len=6550
18	5.767000	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=51dc) [Reassembled in #22]
19	5.767001	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=51dc) [Reassembled in #22]
20	5.767002	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=2960, ID=51dc) [Reassembled in #22]
21	5.767002	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=4440, ID=51dc) [Reassembled in #22]
22	5.767003	192.168.0.207	192.168.0.54	UDP	672 55489 → 12 Len=6550
23	5.767525	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=51dd) [Reassembled in #27]
24	5.767525	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=51dd) [Reassembled in #27]
25	5.767526	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=2960, ID=51dd) [Reassembled in #27]
26	5.767527	192.168.0.207	192.168.0.54	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=4440, ID=51dd) [Reassembled in #27]

Figure 53: A segment of a pcap file showing the large Python UDP packets

This figure show that the UDP packets were being sent to the correct address as well as being sent to every port on the PLC. However, there is no data here to show that the PLC had responded to the UDP traffic or that the UDP traffic had caused the PLC to fail or operate incorrectly.

This concludes the experiments performed on the Siemens S7-1200 PLC.

E.9 Execution of Experiments: ABB PM564

After conducting experiments on the Siemens S7-1200 PLC, the SCADA systems was then reconfigured to operate the ABB PM564 PLC and the corresponding field devices (Figure 37). The aim is to replicate the exact same network scans conducted against the previous PLC in order to determine whether the results of the last experiment were unique to that specific device. In order to begin these experiments, the ABB PLC was connected to the host machine via a CAT5 Ethernet cable. Once this had been implemented, the ABB Controller Builder Plus software was opened on the host machine. Here, the control code (see subsubsection E.5.4) was compiled and downloaded onto the PLC. In order to test that the code was operating correctly, the run button on the HMI interface was pressed (Figure 37). Once pressed, the two LEDs began to turn on and off intermittently. This showed that the code was working as desired and that the system was ready for the experiments to take place.

E.9.1 Experiment 1: Nmap 2.8

E.9.1.1 Nmap Ping Sweep The first experiment conducted on the ABB PLC uses Nmap in order to facilitate a TCP SYN sweep of the SCADA network. This type of scan is used as a mechanism for discovering active hosts within a given address space. This method of network scanning does not require the host to establish a connection with the target devices, nor does it consist of sending large streams of data to the target devices. The Nmap command executed against the ABB PLC was as follows:

"nmap -sP -r 192.168.0.* -e eth0 > abb_nmap1.txt"

The command entered here follows the same syntax as the command used within the previous Nmap experiments on the Siemens S7 PLC. This Nmap command has been created in order to conduct asset detection on the 192.168.0.0/24 subnet and all results of this scan are to be written to the "abb_nmap1.txt" file. Once this command had been executed, the scan was able to complete without any errors and an output file was created successfully. On observation of both the PCB and the HMI present on the host machine, the network scan did not appear to have any effect on the operation of the SCADA system. In order to confirm that the scan had functioned correctly, the output file was opened and the information inside was cross-referenced with the configuration of the SCADA system.

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-03-08 12:40 GMT Standard Time
Nmap scan report for 192.168.0.10
Host is up (0.00s latency).
MAC Address: 00:24:59:0A:38:45 (ABB Automation products GmbH)
Nmap done: 256 IP addresses (1 host up) scanned in 3.38 seconds
```

Figure 54: The data contained within "abb_nmap1.txt"

The data presented within Figure 54 shows that Nmap was able to discover the ABB PLC on the network without disrupting the behaviour of the PLC, the HMI or the field devices being controlled and monitored. Like the previous scan performed on the Siemens PLC, the "-sP" scan does not provide a large amount of information about the PLC other than the IP address, MAC address and the manufacturer associated with the second portion of the MAC address (see subsubsection C.2.3).

96	3.049520	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.1? Tell 192.168.0.207
97	3.057748	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.4? Tell 192.168.0.207
98	3.057802	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.5? Tell 192.168.0.207
99	3.058682	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.6? Tell 192.168.0.207
100	3.059486	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.7? Tell 192.168.0.207
101	3.060289	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.8? Tell 192.168.0.207
102	3.061096	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.9? Tell 192.168.0.207
103	3.061899	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.10? Tell 192.168.0.207
104	3.062703	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.11? Tell 192.168.0.207
105	3.063414	AbbAutom_0a:38:45	Micro-St_f2:f0:af	ARP	64 192.168.0.10 is at 00:24:59:0a:38:45 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
106	3.063540	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.12? Tell 192.168.0.207
107	3.073376	Micro-St_f2:f0:af	Broadcast	ARP	42 Who has 192.168.0.15? Tell 192.168.0.207

Figure 55: A section of the pcap file showing Nmap utilising the ARP protocol

The figure above shows that Nmap is using the ARP protocol in order to discover hosts on the LAN, rather than using unnecessary TCP/IP frames. The above figure shows that once an ARP request has been made to 192.168.0.10, the PLC then replies, supplying its MAC address and subsequently its manufacturer.

E.9.1.2 Nmap Service Detection Scan The next Nmap scan executed against the network aimed at discovering the services being hosted on the target PLC. This type of scan is a more intrusive method of information gathering, this means that in order to probe the target devices for data, the host machine must establish a connection with each target in order for data to be transmitted and received. The Nmap command used to achieve this is shown below:

"nmap -sV -r 192.168.0.* -e eth0 > abb_nmap2.txt"

This command uses the same flags and options as the Nmap experiment in subsubsection C.8.1.2. This command instructs Netkit to perform its service detection scan on the 192.168.0.0/24 subnet. All output from this scan should be captured within the "abb_nmap2.txt" file.

Once the Nmap command had been executed against the network, the scan was able to run until completion without encountering any errors or dropped packets. An output was also generated and populated at the end of the scan. From observing the activity of the PCB components, as well as the status of the HMI interface on the host machine, the Nmap scan did not appear to alter the behaviour of the network. All devices and components present on the network remained consistent through the entire duration of the scan. In order to evaluate the success of the scan, the output file was opened and the information within it was cross-referenced with the configuration of the ABB SCADA system.

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-03-08 12:40 GMT Standard Time
Nmap scan report for 192.168.0.10
Host is up (0.0046s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE       VERSION
80/tcp   open  http          3S_WebServer
1201/tcp open  nucleus-sand?
1 service unrecognized despite returning data. If you know the service/version, please submit t
SF-Port80-TCP:V=7.40%I=7%D=3/8%Time=58BFFBE4%P=i686-pc-windows-windows%r(G
SF:etRequest,13D,"HTTP/1\.1\x20400\x20Page\x20not\x20found\r\nServer:\x203
SF:S_WebServer\r\nDate:\x20Wed\x20Dec\x2031\x2020:49:33\x201969\r\nPragma:
SF:\x20no-cache\r\nCache-Control:\x20no-cache\r\nContent-Type:\x20text/htm
SF:l\r\n\r\n<html><head><title>Document\x20Error:\x20Page\x20not\x20found<
SF:/title></head>\r\n\t\t<body><h2>Access\x20Error:\x20Page\x20not\x20foun
SF:d</h2>\r\n\t\t<p>Cannot\x20stat\x20page\x20for\x20URL</p></body></html>
SF:\r\n\r\n")%r(HTTPOptions,135,"HTTP/1\.1\x20400\x20Page\x20not\x20found\
SF:r\nServer:\x203S_WebServer\r\nDate:\x20Wed\x20Dec\x2031\x2020:49:33\x20
SF:1969\r\nPragma:\x20no-cache\r\nCache-Control:\x20no-cache\r\nContent-Ty
SF:pe:\x20text/html\r\n\r\n<html><head><title>Document\x20Error:\x20Page\x
SF:20not\x20found</title></head>\r\n\t\t<body><h2>Access\x20Error:\x20Page
SF:\x20not\x20found</h2>\r\n\t\t<p>Bad\x20request\x20type</p></body></html
SF:>\r\n\r\n")%r(RTSPRequest,135,"HTTP/1\.1\x20400\x20Page\x20not\x20found
SF:\r\nServer:\x203S_WebServer\r\nDate:\x20Wed\x20Dec\x2031\x2020:49:33\x2
SF:01969\r\nPragma:\x20no-cache\r\nCache-Control:\x20no-cache\r\nContent-T
SF:ype:\x20text/html\r\n\r\n<html><head><title>Document\x20Error:\x20Page\
SF:x20not\x20found</title></head>\r\n\t\t<body><h2>Access\x20Error:\x20Pag
SF:e\x20not\x20found</h2>\r\n\t\t<p>Bad\x20request\x20type</p></body></htm
SF:l>\r\n\r\n")%r(X11Probe,135,"HTTP/1\.1\x20400\x20Page\x20not\x20found\r
SF:\nServer:\x203S_WebServer\r\nDate:\x20Wed\x20Dec\x2031\x2020:49:33\x201
SF:969\r\nPragma:\x20no-cache\r\nCache-Control:\x20no-cache\r\nContent-Typ
SF:e:\x20text/html\r\n\r\n<html><head><title>Document\x20Error:\x20Page\x2
SF:0not\x20found</title></head>\r\n\t\t<body><h2>Access\x20Error:\x20Page\
SF:x20not\x20found</h2>\r\n\t\t<p>Bad\x20HTTP\x20request</p></body></html>
SF:\r\n\r\n")%r(FourOhFourRequest,13D,"HTTP/1\.1\x20400\x20Page\x20not\x20
SF:found\r\nServer:\x203S_WebServer\r\nDate:\x20Wed\x20Dec\x2031\x2020:49:
SF:33\x201969\r\nPragma:\x20no-cache\r\nCache-Control:\x20no-cache\r\nCont
SF:ent-Type:\x20text/html\r\n\r\n<html><head><title>Document\x20Error:\x20
SF:Page\x20not\x20found</title></head>\r\n\t\t<body><h2>Access\x20Error:\x
SF:20Page\x20not\x20found</h2>\r\n\t\t<p>Cannot\x20stat\x20page\x20for\x20
SF:URL</p></body></html>\r\n\r\n");
MAC Address: 00:24:59:0A:38:45 (ABB Automation products GmbH)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 256 IP addresses (1 host up) scanned in 18.35 seconds
```

Figure 56: The data contained within "abb_nmap2.txt", showing a large amount of raw data

The information drawn from the service detection scan against the ABB PLC yielded some unforeseen results. Firstly, as shown in Figure 56, on running a service detection scan, Nmap was able to highlight the presence of a HTTP 3S web-server (port 80) on the PLC which had not previously been configured or referenced. The service detection scan was also able to detect that port 1201 was open on the PLC. Alongside the port number, Nmap ran fingerprint diagnostics across the data it was able to obtain from that port. The Nmap database classes port 1201 as a "nucleus-sand" (Nucleus Sand) service. Nucleus Sand is a database management system which only runs on Windows operating systems and a variety of Linux distributions. Nmap was also able to obtain some data from the "-sV" scan which could not be identified. With tags such as "<head>", "<title>"

and “URL”, the data appears to be a form of HTML. In order to understand how that data was obtained and which service is running on port 1201, the pcap file taken from the host machine was opened and analysed in Wireshark. The following data was obtained:

1	0.000000	192.168.0.207	192.168.0.10	TCP	62 1093 → 1201 [PSH, ACK] Seq=1 Ack=1 Win=65464 Len=8
2	0.001920	192.168.0.10	192.168.0.207	TCP	88 1201 → 1093 [PSH, ACK] Seq=1 Ack=9 Win=5976 Len=34
3	0.156040	192.168.0.207	192.168.0.10	TCP	62 1093 → 1201 [PSH, ACK] Seq=9 Ack=35 Win=65430 Len=8
4	0.158220	192.168.0.10	192.168.0.207	TCP	388 1201 → 1093 [PSH, ACK] Seq=35 Ack=17 Win=5968 Len=334
5	0.158911	192.168.0.207	192.168.0.10	TCP	62 1093 → 1201 [PSH, ACK] Seq=17 Ack=369 Win=65096 Len=8
6	0.161644	192.168.0.10	192.168.0.207	TCP	125 1201 → 1093 [PSH, ACK] Seq=369 Ack=25 Win=5960 Len=71
7	0.202819	192.168.0.207	192.168.0.10	TCP	62 1093 → 1201 [PSH, ACK] Seq=25 Ack=440 Win=65025 Len=8
8	0.204967	192.168.0.10	192.168.0.207	TCP	88 1201 → 1093 [PSH, ACK] Seq=440 Ack=33 Win=5952 Len=34
9	0.358886	192.168.0.207	192.168.0.10	TCP	62 1093 → 1201 [PSH, ACK] Seq=33 Ack=474 Win=64991 Len=8
10	0.361812	192.168.0.10	192.168.0.207	TCP	388 1201 → 1093 [PSH, ACK] Seq=474 Ack=41 Win=5944 Len=334
11	0.362603	192.168.0.207	192.168.0.10	TCP	62 1093 → 1201 [PSH, ACK] Seq=41 Ack=808 Win=64657 Len=8
12	0.365109	192.168.0.10	192.168.0.207	TCP	125 1201 → 1093 [PSH, ACK] Seq=808 Ack=49 Win=5936 Len=71
13	0.405626	192.168.0.207	192.168.0.10	TCP	62 1093 → 1201 [PSH, ACK] Seq=49 Ack=879 Win=64586 Len=8
14	0.408010	192.168.0.10	192.168.0.207	TCP	88 1201 → 1093 [PSH, ACK] Seq=879 Ack=57 Win=5928 Len=34
15	0.561768	192.168.0.207	192.168.0.10	TCP	62 1093 → 1201 [PSH, ACK] Seq=57 Ack=913 Win=64552 Len=8

Figure 57: A segment of a pcap showing data being sent from the host machine to the PLC via TCP

From analysing the network traffic captured from the host machine, port 1201 could represent the communication channel between the Controller Builder Plus client and the PLC in order to allow the HMI interface to display the activity of the field device.

```
00000000  bb bb 00 00 00 02 3b ff                                    ......;.
    00000000  bb bb 00 00 00 1c 00 00  00 01 00 09 fd 58 00 00   ........ .....X..
    00000010  00 01 00 00 00 01 00 00  00 01 00 09 fd 58 00 00   ........ .....X..
    00000020  00 01                                              ..
00000008  bb bb 00 00 00 02 50 05                                    ......P.
    00000022  bb bb 00 00 01 48 00 00  00 05 00 00 00 00 19 a0   .....H.. ........
    00000032  01 00 01 01 01 01 01 01  01 01 00 00 01 00 00 07   ........ ........
    00000042  d0 00 00 13 88 01 01 01  00 00 00 00 00 00 00 00   ........ ........
    00000052  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000062  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000072  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000082  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000092  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    000000A2  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    000000B2  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    000000C2  00 00 00 01 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    000000D2  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    000000E2  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    000000F2  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000102  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000112  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000122  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000132  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000142  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000152  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000162  00 00 00 00 00 00 00 00  00 00 00 00 00 00        ........ ......
00000010  bb bb 00 00 00 02 51 10                                    ......Q.
    00000170  bb bb 00 00 00 41 00 00  00 10 00 00 00 00 00 00   .....A.. ........
    00000180  00 00 00 00 cd cd cd cd  cd cd cd cd cd cd cd cd   ........ ........
    00000190  cd cd cd cd cd cd cd cd  cd cd cd cd cd cd cd cd   ........ ........
    000001A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    000001B0  04 cd 00 00 00 00 00                               .......
00000018  bb bb 00 00 00 02 3b ff                                    ......;.
    000001B7  bb bb 00 00 00 1c 00 00  00 01 00 09 fd 6c 00 00   ........ .....l..
    000001C7  00 01 00 00 00 01 00 00  00 01 00 09 fd 6c 00 00   ........ .....l..
    000001D7  00 01                                              ..
00000020  bb bb 00 00 00 02 50 05                                    ......P.
```

Figure 58: The hex values associated with the TCP stream on port 1201

On analysing the raw hexadecimal data being sent and received from port 1201 (see Figure 58), the client (host machine) sends an 8-byte block of data to the server (PLC), on which the server responds with a larger set of data. This pattern is repeated throughout the entire duration of the network capture. Although there seems to be a data-transfer pattern between the host machine and the PLC on port 1201, it cannot be confirmed

that the TCP stream is specifically the traffic for HMI monitoring and control. However, after analysing all the previous pcap files taken from the ABB interaction, the same communication is present in each one.

1	0.000000	192.168.0.207	192.168.0.10	TCP	62	1093 → 1201 [PSH, ACK] Seq=1 Ack=1 Win=64552 Len=8
2	0.002426	192.168.0.10	192.168.0.207	TCP	388	1201 → 1093 [PSH, ACK] Seq=1 Ack=9 Win=4960 Len=334
3	0.002863	192.168.0.207	192.168.0.10	TCP	62	1093 → 1201 [PSH, ACK] Seq=9 Ack=335 Win=64218 Len=8
4	0.005303	192.168.0.10	192.168.0.207	TCP	125	1201 → 1093 [PSH, ACK] Seq=335 Ack=17 Win=4952 Len=71
5	0.046943	192.168.0.207	192.168.0.10	TCP	62	1093 → 1201 [PSH, ACK] Seq=17 Ack=406 Win=64147 Len=8
6	0.049854	192.168.0.10	192.168.0.207	TCP	88	1201 → 1093 [PSH, ACK] Seq=406 Ack=25 Win=4944 Len=34
7	0.204173	192.168.0.207	192.168.0.10	TCP	62	1093 → 1201 [PSH, ACK] Seq=25 Ack=440 Win=64113 Len=8
8	0.206436	192.168.0.10	192.168.0.207	TCP	388	1201 → 1093 [PSH, ACK] Seq=440 Ack=33 Win=4936 Len=334
9	0.206955	192.168.0.207	192.168.0.10	TCP	62	1093 → 1201 [PSH, ACK] Seq=33 Ack=774 Win=65535 Len=8
10	0.210043	192.168.0.10	192.168.0.207	TCP	125	1201 → 1093 [PSH, ACK] Seq=774 Ack=41 Win=4928 Len=71
11	0.249605	192.168.0.207	192.168.0.10	TCP	62	1093 → 1201 [PSH, ACK] Seq=41 Ack=845 Win=65464 Len=8
12	0.251247	192.168.0.10	192.168.0.207	TCP	88	1201 → 1093 [PSH, ACK] Seq=845 Ack=49 Win=4920 Len=34
13	0.405743	192.168.0.207	192.168.0.10	TCP	62	1093 → 1201 [PSH, ACK] Seq=49 Ack=879 Win=65430 Len=8
14	0.407531	192.168.0.10	192.168.0.207	TCP	388	1201 → 1093 [PSH, ACK] Seq=879 Ack=57 Win=4912 Len=334
15	0.407969	192.168.0.207	192.168.0.10	TCP	62	1093 → 1201 [PSH, ACK] Seq=57 Ack=1213 Win=65096 Len=8

Figure 59: Another figure showing the TCP conversation between PLC and host machine

In reference to the unidentifiable data Nmap extracted from the PLC, the pcap shown in Figure 59 also revealed a successful HTTP connection. This is also supported by the presence of port 80 within the output file shown in Figure 56. Once this HTTP interaction was located within the pcap file, the TCP stream was then examined using Wireshark's "Follow TCP Stream" feature. This revealed that the unidentified block of data was failed HTTP requests and responses sent and received between the host machine and the PLC.

http

No.	Time	Source	Destination	Protocol	Length	Info
2897	12.557906	192.168.0.207	192.168.0.10	HTTP	72	GET / HTTP/1.0
2901	12.576958	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
2909	12.585460	192.168.0.207	192.168.0.10	HTTP	76	OPTIONS / HTTP/1.0
2911	12.603964	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
2918	12.613279	192.168.0.207	192.168.0.10	HTTP	76	OPTIONS / RTSP/1.0
2920	12.633214	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
2937	12.866830	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
2944	12.870769	192.168.0.207	192.168.0.10	HTTP	107	GET /nice%20ports%2C/Tri%6Eity.txt%2ebak HTTP/1.0
2946	12.891964	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
2955	12.920891	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
2975	13.147038	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
2992	13.376915	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3010	13.607434	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3025	13.837058	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3041	14.067065	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3057	14.297165	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3073	14.527291	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3089	14.757295	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3109	14.987286	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3129	15.217347	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3145	15.447672	192.168.0.10	192.168.0.207	HTTP	60	HTTP/1.1 400 Page not found (text/html)
3152	15.451440	192.168.0.207	192.168.0.10	HTTP	277	OPTIONS sip:nm SIP/2.0

Figure 60: A segment of a pcap showing HTTP traffic being sent between the host machine and the PLC

Nmap was able to obtain this information without disrupting the normal operation of the SCADA system.

E.9.1.3 Nmap UDP Scan The final Nmap scan conducted against the SCADA system utilised UDP packets in order to gain information about the ABB PLC. The previous experiments have shown that UDP scans have revealed more in-depth information about the PLC system tested in subsubsection C.8.1.3 than the previous TCP-based scans. UDP is also a connectionless protocol, meaning that data can be sent to any UDP port without any prior synchronisation. The Nmap command used to execute the UDP scan was as follows:

"nmap -sU -sC 192.168.0.* -e eth0 > abb_nmap3.txt"

This Nmap scan has been configured to replicate the UDP scan used within subsubsection C.8.1.3 of this document. As well as conducting a UDP scan, the "-sC" flag instructs Nmap to run a series of default NSE scripts against the target network.

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-03-08 12:42 GMT Standard Time
Nmap scan report for 192.168.0.10
Host is up (0.0052s latency).
Not shown: 999 closed ports
PORT      STATE         SERVICE
49153/udp open|filtered unknown
MAC Address: 00:24:59:0A:38:45 (ABB Automation products GmbH)

Nmap done: 1 IP address (1 host up) scanned in 2.62 seconds
```

Figure 61: The data contained within "abb_nmap3.txt"

This scan was able to run until completion without encountering any errors or connection failures. Once the scan had finished, an output file was generated and populated with network data (see Figure 61). From observing the same PCB components and the HMI interface on the host machine, no changes had been made to the SCADA as a result of the network scanning. Figure 61 shows that Nmap was able to obtain data about the PLC without disrupting the behaviour of the field devices, the HMI or the PLC itself.

This concluded the Nmap experiments conducted on the ABB PLC.

E.9.2 Experiment 2: Zmap 2.1.0

The next experiment to be conducted on the ABB PLC focussed on using ICMP packets in order to ascertain active hosts on the network. This is different to the ARP scanning method used by Nmap in subsubsections C.8.1.3 and C.9.1.1. Zmap is able to facilitate large scale ICMP echo sweeps against small and large networks, which makes it the optimal tool for testing the stability of SCADA equipment when being scanned by ICMP packets. The Zmap command used to conduct this experiment is shown below:

"sudo zmap -M icmp_echoscan 192.168.0.0/24 -o abb_zmap1.csv"

This command has the same configuration seen within the last Zmap experiment in subsection C.8.2. For this experiment, the Linux laptop had to be inserted into the network via a switch. This was so that the Zmap scan could be executed at the same time both the HMI and the PLC were running their assigned code.

Similar to the results of the experiment within subsection C.8.2, on executing the Zmap command, no errors were reported on the terminal screen and an output file was successfully created. However, on inspection of the output file, Zmap was unable to locate the PLC residing on the network. As well as failing to obtain any results from the scan itself, the behaviour of the SCADA system did not change at any point during the experiment. Similar to the results yielded within subsection C.8.2, the pcap file taken from the host machine revealed that ICMP packets had been sent to the correct subnet. However, there was no data in this pcap that showed any of the addresses responding to the ICMP requests (see Figure 62).

121 0.000626	192.168.0.99	192.168.0.30	ICMP	62	Echo (ping) request id=0xc7f2, seq=0/0, ttl=255 (no response found!)
122 0.000630	192.168.0.99	192.168.0.111	ICMP	62	Echo (ping) request id=0xedee, seq=0/0, ttl=255 (no response found!)
123 0.000633	192.168.0.99	192.168.0.14	ICMP	62	Echo (ping) request id=0x40d8, seq=0/0, ttl=255 (no response found!)
124 0.000637	192.168.0.99	192.168.0.219	ICMP	62	Echo (ping) request id=0x8ecd, seq=0/0, ttl=255 (no response found!)
125 0.000640	192.168.0.99	192.168.0.56	ICMP	62	Echo (ping) request id=0xc368, seq=0/0, ttl=255 (no response found!)
126 0.000643	192.168.0.99	192.168.0.64	ICMP	62	Echo (ping) request id=0xb8cf, seq=0/0, ttl=255 (no response found!)
127 0.000646	192.168.0.99	192.168.0.10	ICMP	62	Echo (ping) request id=0xa8fc, seq=0/0, ttl=255 (no response found!)
128 0.000650	192.168.0.99	192.168.0.246	ICMP	62	Echo (ping) request id=0x0556, seq=0/0, ttl=255 (no response found!)

Figure 62: A segment of a pcap showing ICMP packets being transmitted across the network

As this tool was unable to yield any results from both the S7 and the ABB PLCs, this concluded the experiments conducted with Zmap.

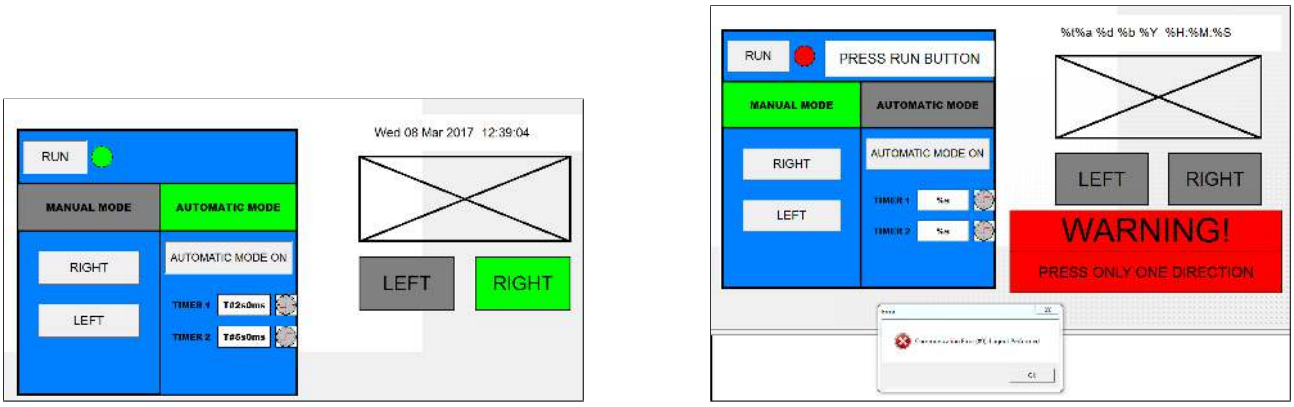
E.9.3 Experiment 3: Custom UDP Scanner

The final experiment conducted against the ABB PLC replicated the UDP DoS attack simulated within subsection C.8.3 of this document. Here, a Python script (see Appendix G) had been created in order to test whether PLCs could withstand a large amount of UDP traffic being sent to every port on the device. This scanning mechanism focussed less on obtaining information as it does not have the data parsing capabilities of the previously used tools. Instead, this scan aimed to monitor the effects of sending large UDP packets to a SCADA device and whether it causes the target device to crash or behave unexpectedly. In order to run this script, the following command was entered into the host machine:

"python udp_dos.py"

This command allowed python to compile and execute the "udp_dos.py" script which can be found alongside its documentation within Appendix G.

The Python script was ran against the PLC for 5 minutes and then manually terminated by the user. Once executed, the UDP scanner began to send large amounts of data to every port present on the PLC. After 1 minute of scanning, the HMI on the host machine abruptly stopped and then last connection to the PLC. Figure 63 shows the HMI interface before and after the UDP scan took place.

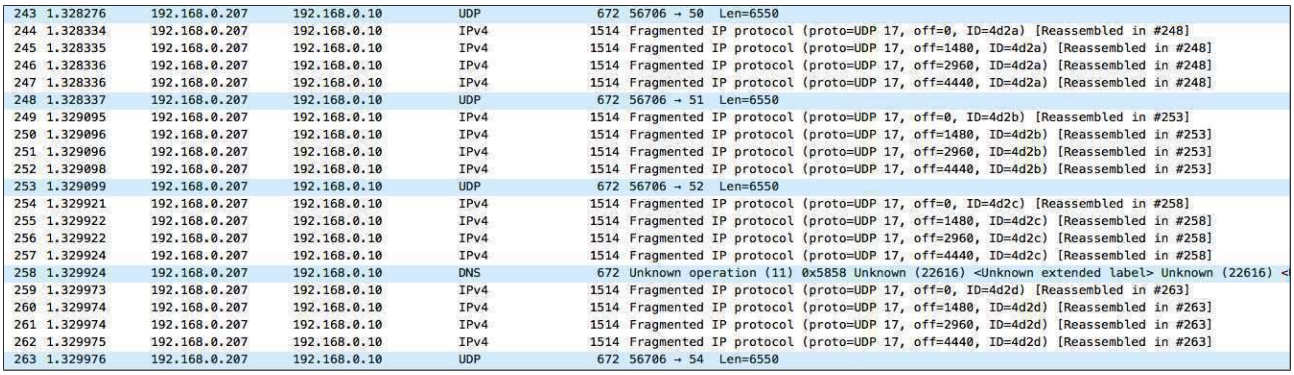


(a) The HMI before being scanned by the Python script

(b) The HMI after being scanned by the Python script

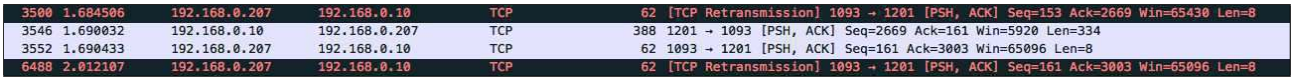
Figure 63: Two figures showing the change in behaviour of the ABB HMI

In order to deduce why the HMI lost connection with the PLC, the pcap taken on the host machine was opened and analysed in Wireshark.



243	1.328276	192.168.0.207	192.168.0.10	UDP	672	56706 → 50 Len=6550
244	1.328334	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=4d2a) [Reassembled in #248]
245	1.328335	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=4d2a) [Reassembled in #248]
246	1.328336	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=4d2a) [Reassembled in #248]
247	1.328336	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=4d2a) [Reassembled in #248]
248	1.328337	192.168.0.207	192.168.0.10	UDP	672	56706 → 51 Len=6550
249	1.329095	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=4d2b) [Reassembled in #253]
250	1.329096	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=4d2b) [Reassembled in #253]
251	1.329096	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=4d2b) [Reassembled in #253]
252	1.329098	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=4d2b) [Reassembled in #253]
253	1.329099	192.168.0.207	192.168.0.10	UDP	672	56706 → 52 Len=6550
254	1.329921	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=4d2c) [Reassembled in #258]
255	1.329922	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=4d2c) [Reassembled in #258]
256	1.329922	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=4d2c) [Reassembled in #258]
257	1.329924	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=4d2c) [Reassembled in #258]
258	1.329924	192.168.0.207	192.168.0.10	DNS	672	Unknown operation (11) 0x5858 Unknown (22616) <Unknown extended label> Unknown (22616) <
259	1.329973	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=4d2d) [Reassembled in #263]
260	1.329974	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=4d2d) [Reassembled in #263]
261	1.329974	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=4d2d) [Reassembled in #263]
262	1.329975	192.168.0.207	192.168.0.10	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=4d2d) [Reassembled in #263]
263	1.329976	192.168.0.207	192.168.0.10	UDP	672	56706 → 54 Len=6550

Figure 64: A section of a pcap showing the large UDP packets sent by the Python script



3500	1.684506	192.168.0.207	192.168.0.10	TCP	62	[TCP Retransmission] 1093 → 1201 [PSH, ACK] Seq=153 Ack=2669 Win=65430 Len=8
3546	1.690032	192.168.0.10	192.168.0.207	TCP	388	1201 → 1093 [PSH, ACK] Seq=2669 Ack=161 Win=5920 Len=334
3552	1.690433	192.168.0.207	192.168.0.10	TCP	62	1093 → 1201 [PSH, ACK] Seq=161 Ack=3003 Win=65096 Len=8
6488	2.012107	192.168.0.207	192.168.0.10	TCP	62	[TCP Retransmission] 1093 → 1201 [PSH, ACK] Seq=161 Ack=3003 Win=65096 Len=8

Figure 65: A segment of a pcap showing the connection loss on port 1201

On opening the pcap (see Figure 64), aside from the large amount of UDP packets being sent to and from the PLC, there was no specific data present which indicated why the HMI lost connection to the PLC. The TCP connection on port 1201 failed to establish a connection between packets 3500 and 6488 (see Figure 65). This delay in the TCP response could be a reason as to why the connection between the HMI and PLC was lost. However, from the data yielded from both this experiment and the test conducted within subsection C.8.3, there is insufficient data to suggest that port 1201 is the port responsible for the communication between the PLC and the HMI, and it was this delay in packet response which cause the error.

Once all the data had been obtained from the host machine, the SCADA network was then powered-off and disconnected from the network.

This concluded the experiments on the SCADA equipment.