

## IoT Device Security

### Exercise 1: Securing IoT Device Communication using TLS/SSL

*Encrypted communication over TLS/SSL is the key to securing IoT Device Communication.*

#### Lab Scenario

Network defenders must encrypt IoT device communication to prevent unauthorized users from viewing sensitive information

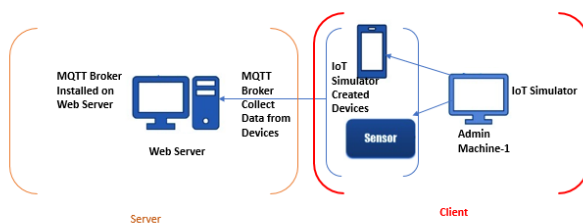
#### Lab Objectives

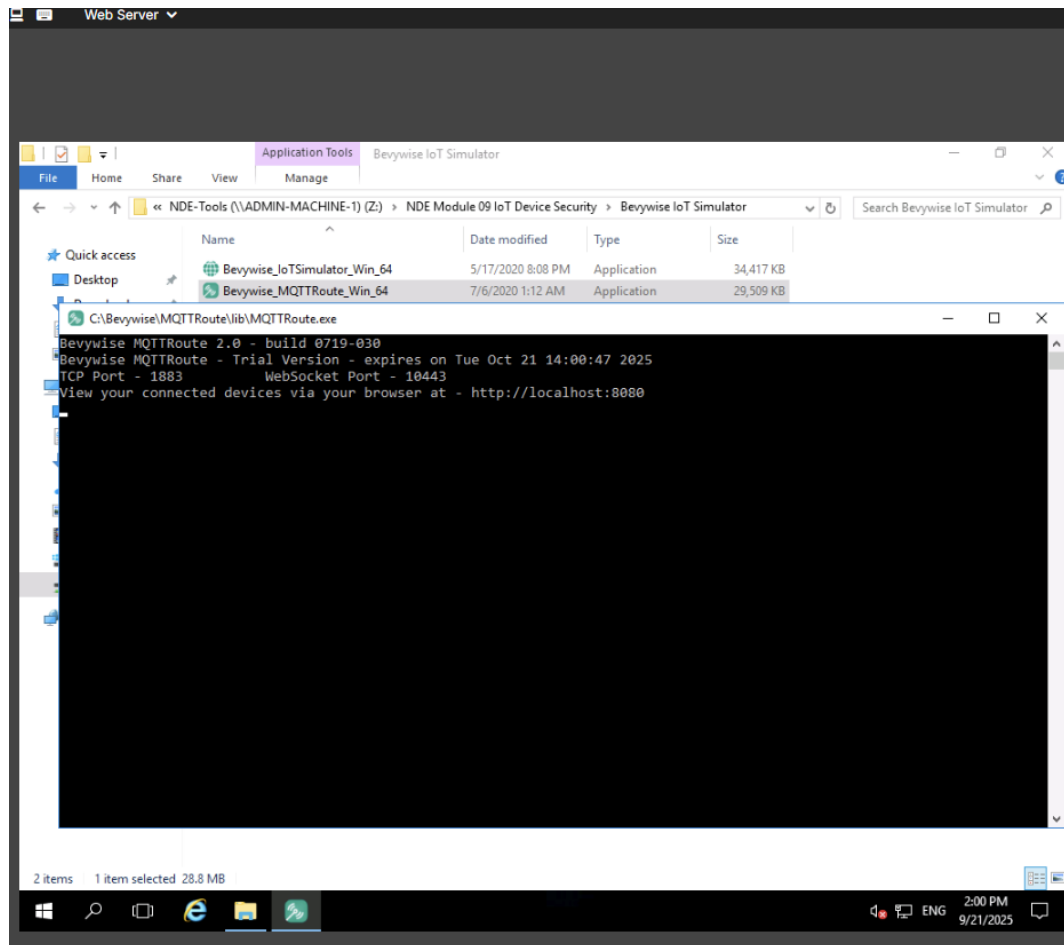
The objective of this lab is to demonstrate how to secure Internet of things (IoT) device communication using the Bevywise message queuing telemetry transport (MQTT) Broker and Simulator. This tool demonstrates the use of IoT devices over the virtual network. In this lab, you will learn to:

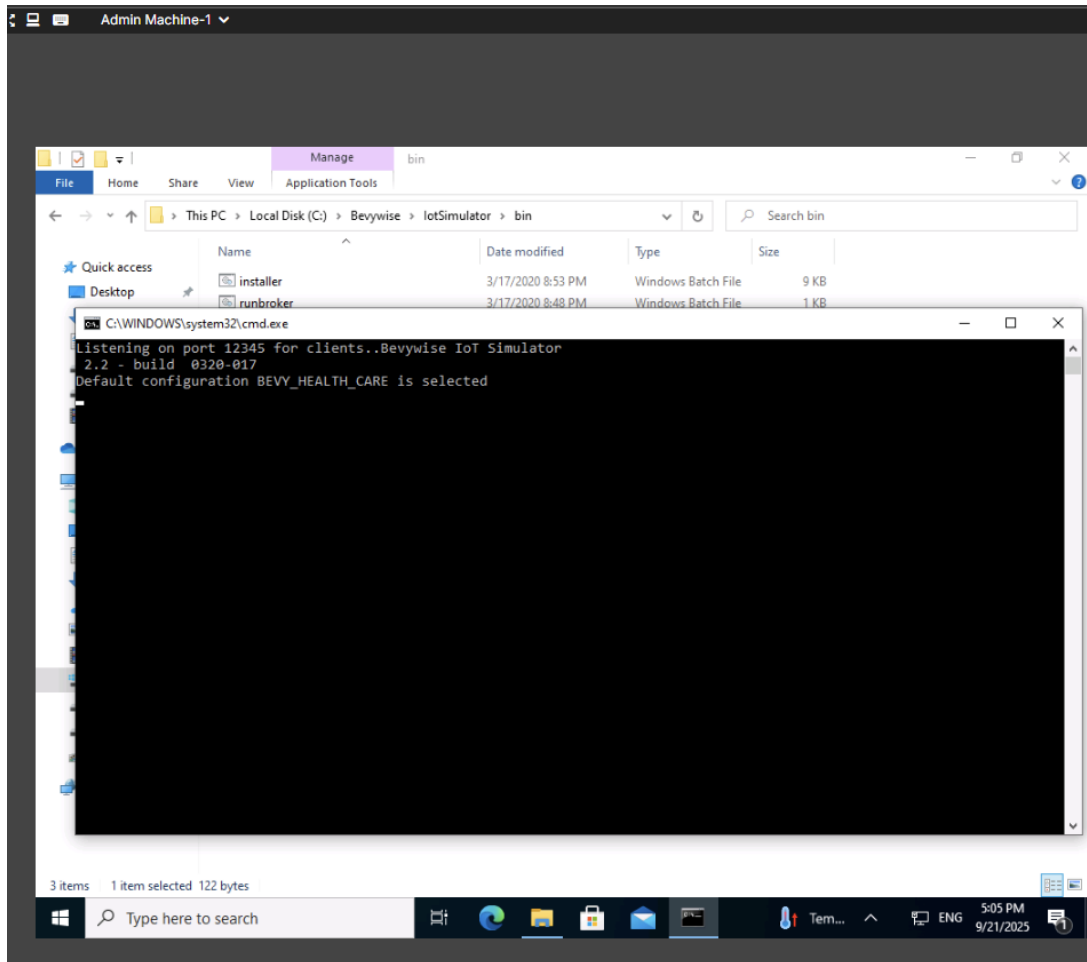
- Install and configure the Bevywise MQTT Broker.
- Implement transport layer security (TLS)/secure sockets layer (SSL) to secure IoT communication.

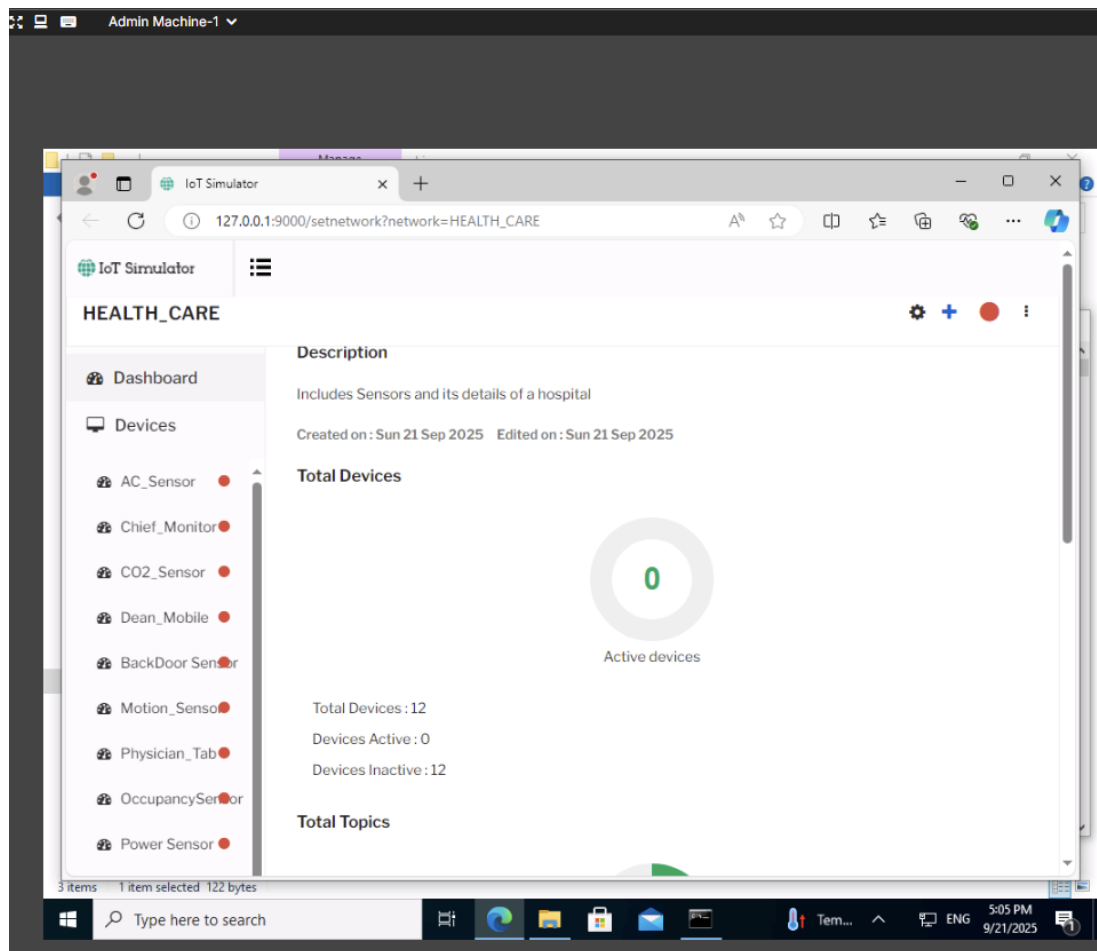
#### Overview of the Bevywise IoT simulator

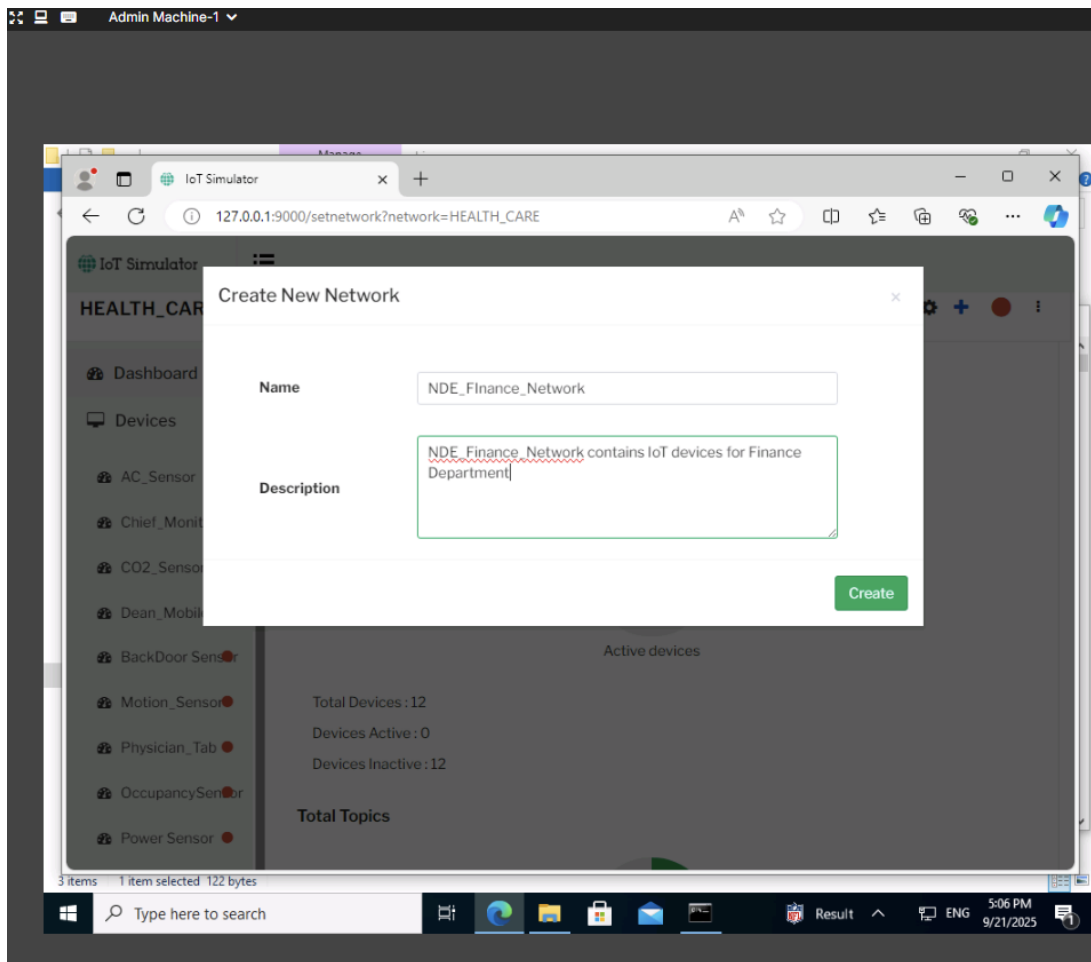
MQTT is a lightweight messaging protocol that uses a publish/subscribe communication pattern. Since the protocol is meant for devices with a low-bandwidth, it is considered ideal for machine-to-machine (M2M) communication or IoT applications. We can create virtual IoT devices over the virtual network using the Bevywise IoT simulator on the client side and communicate these devices to the server using the MQTT Broker web interface. This interface collects data and displays the status and messages of connected devices over the network.

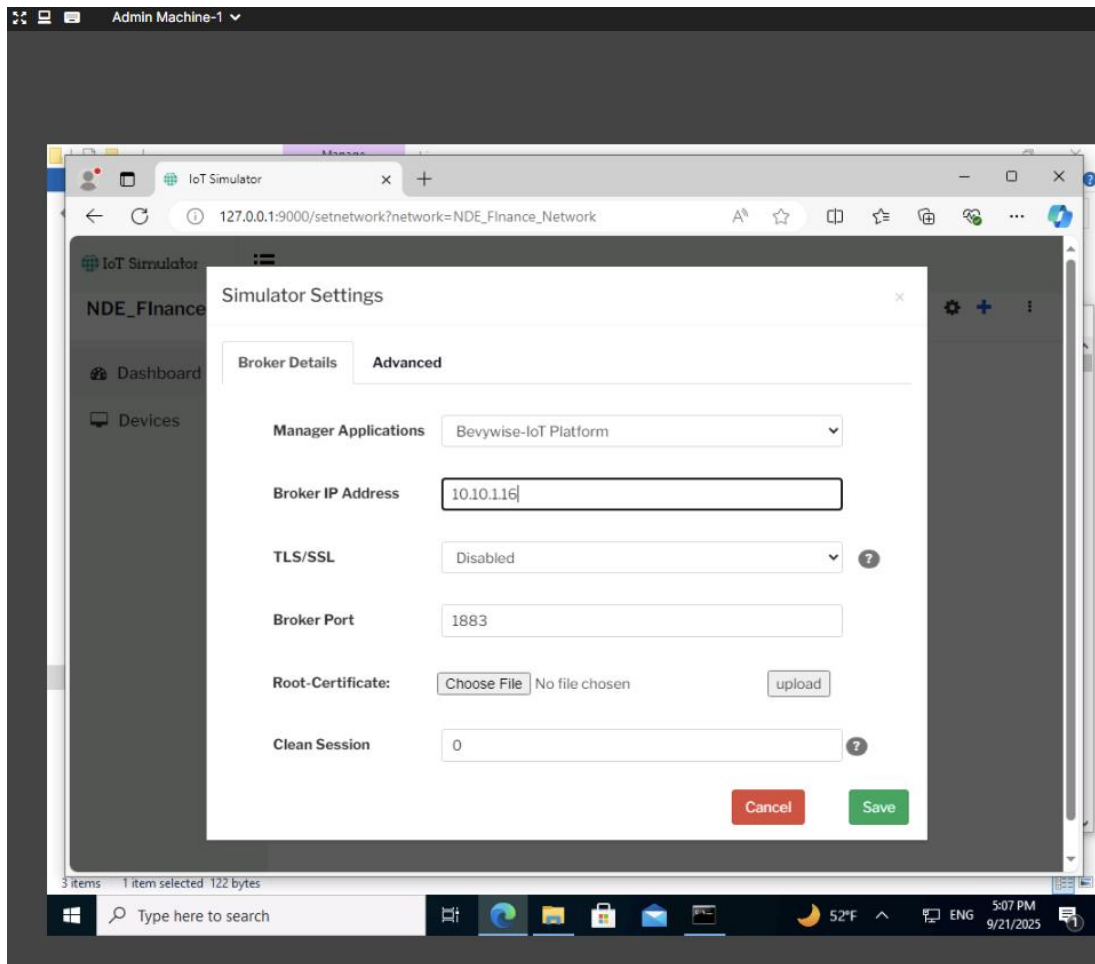


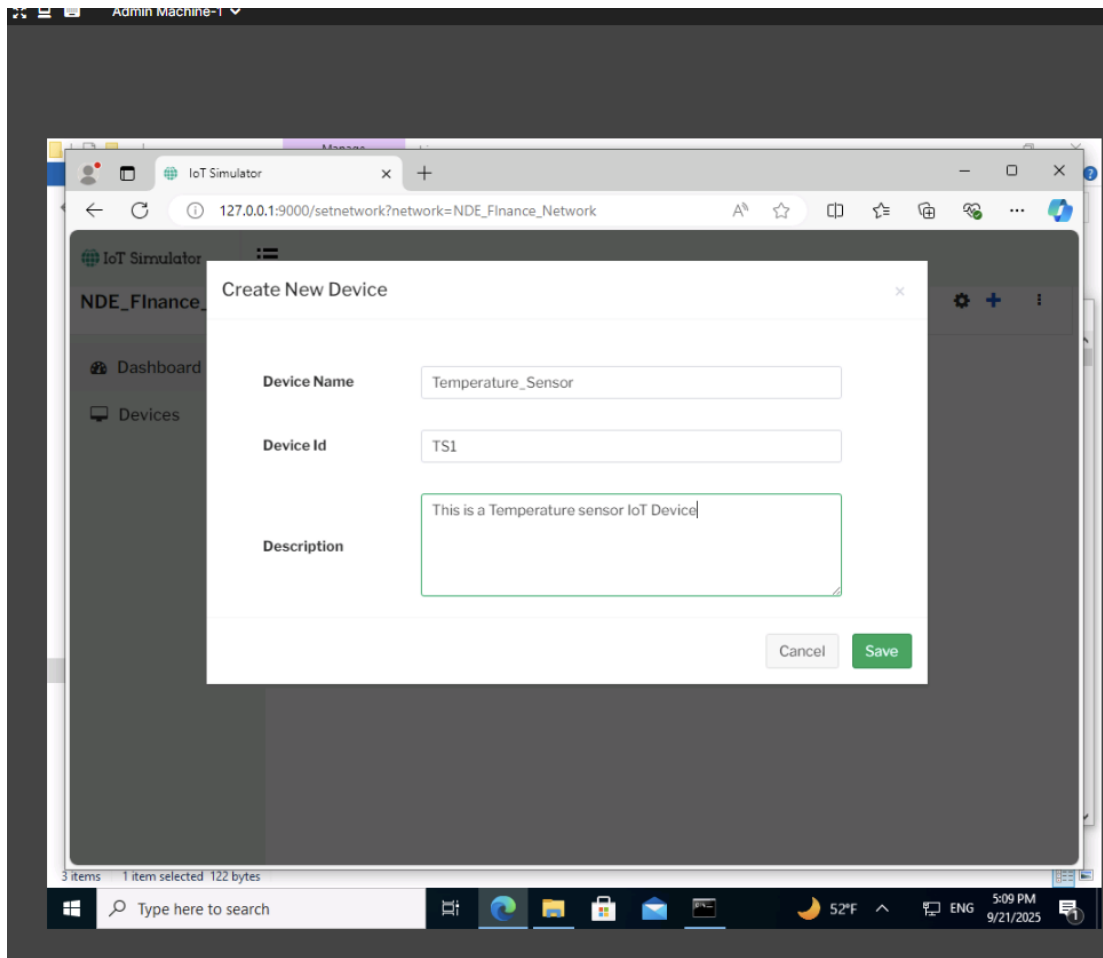


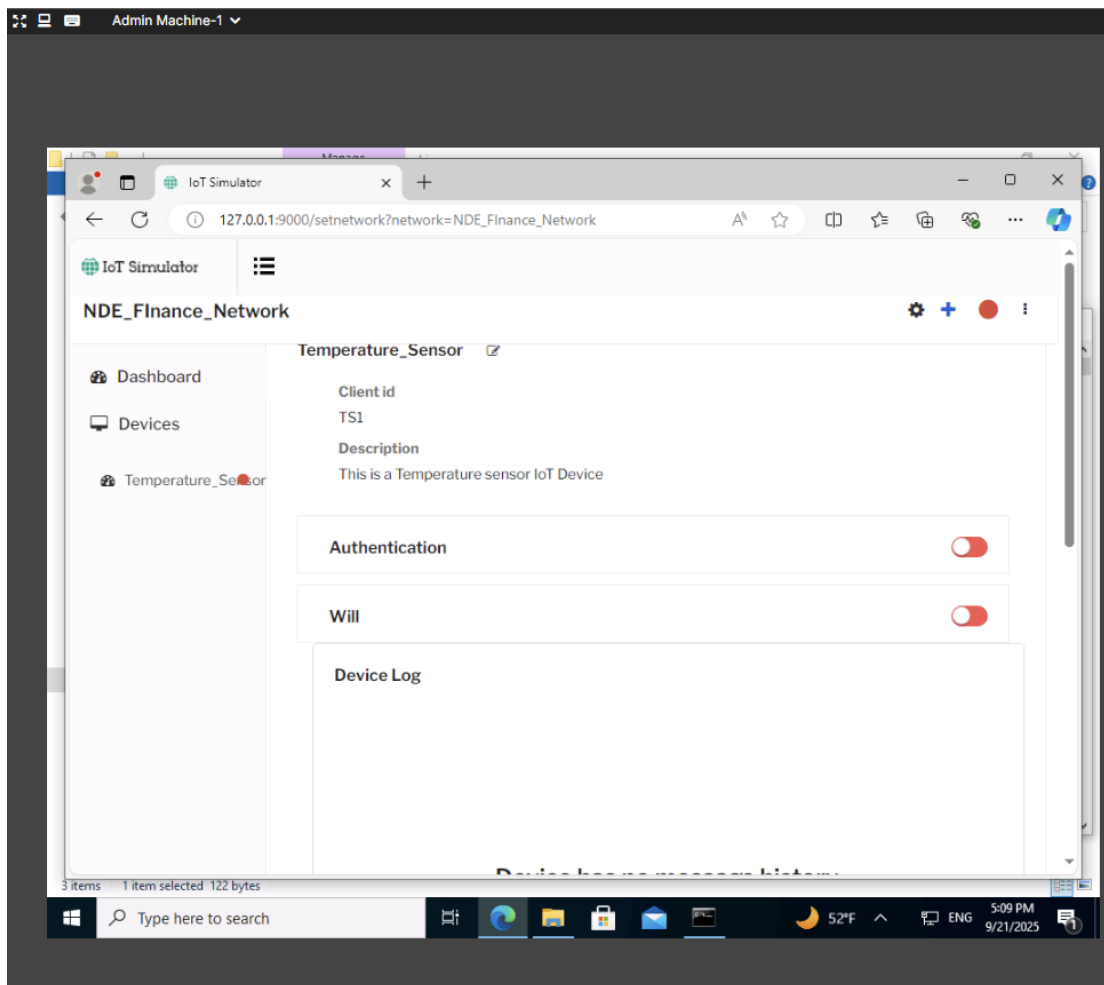




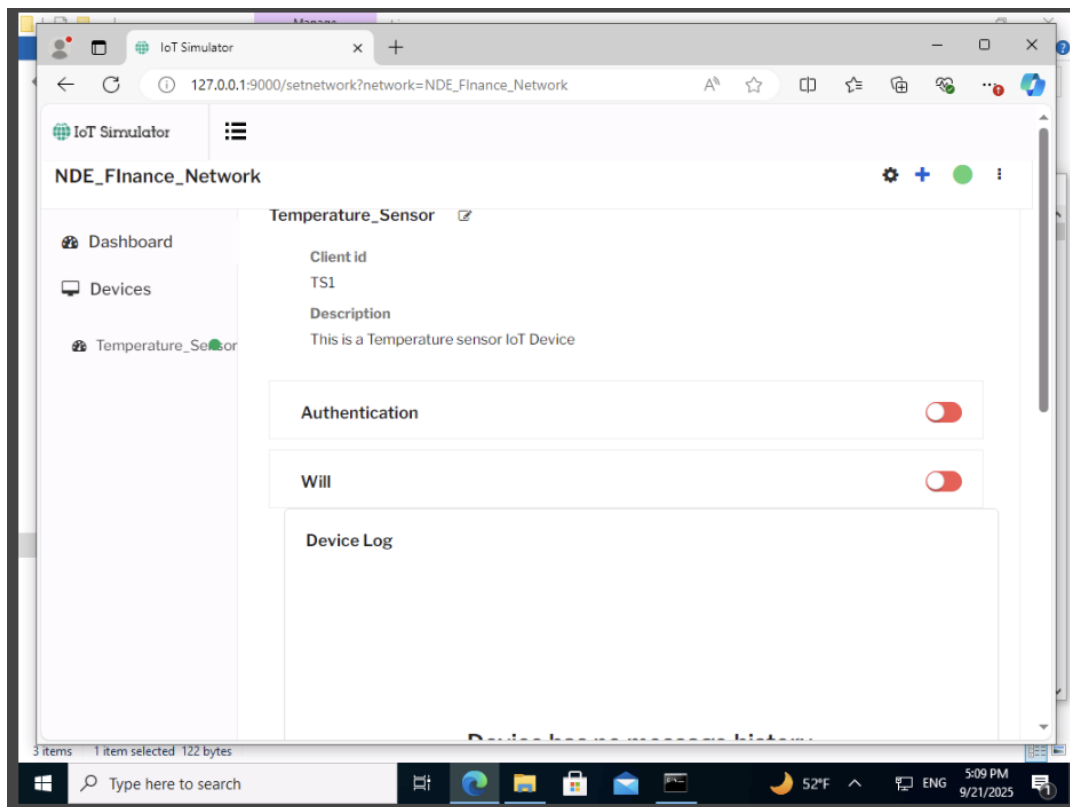


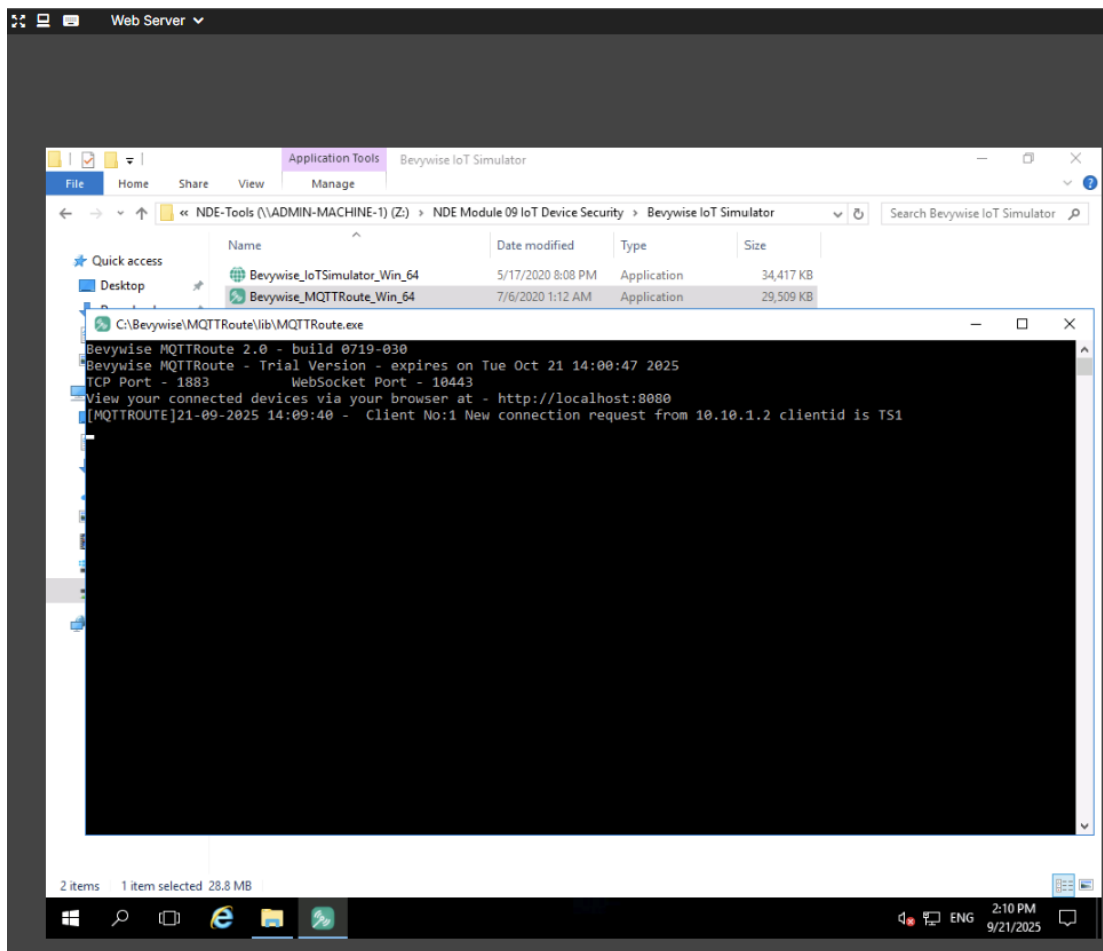


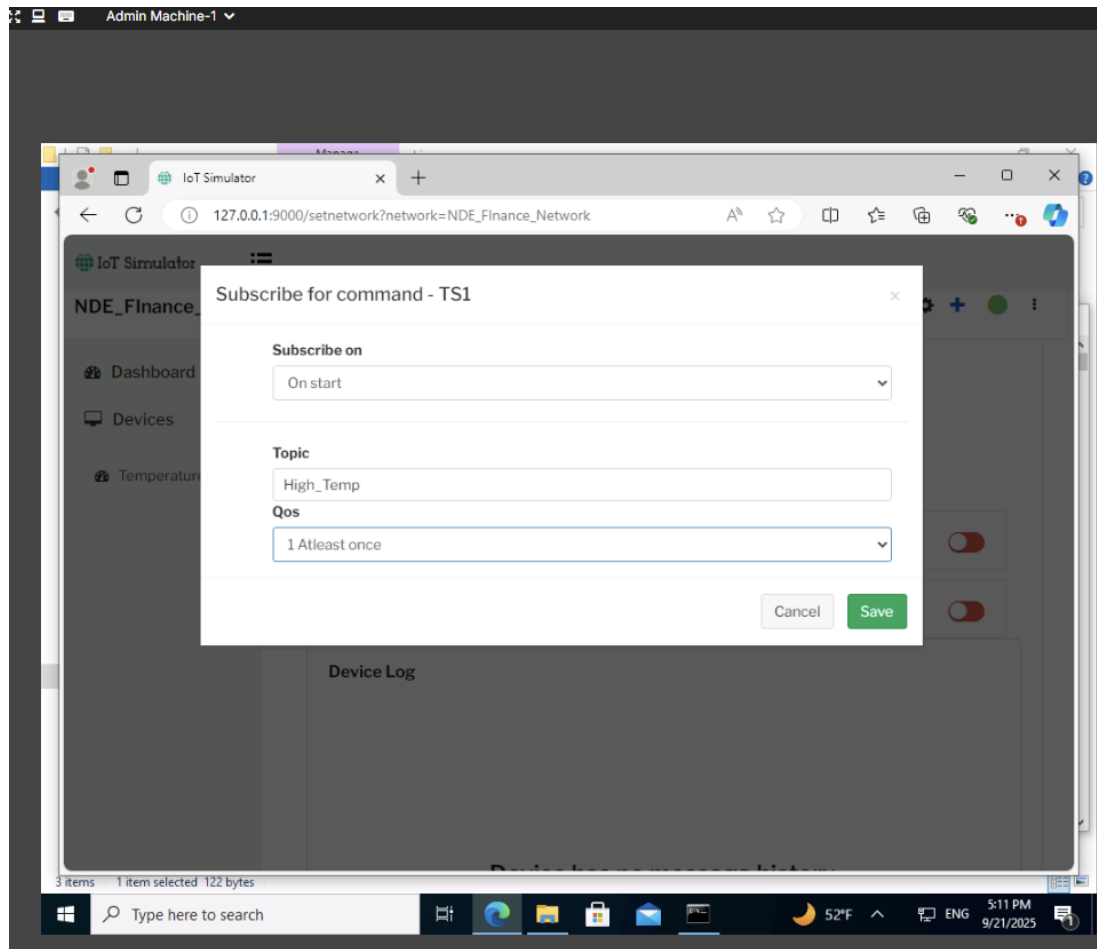




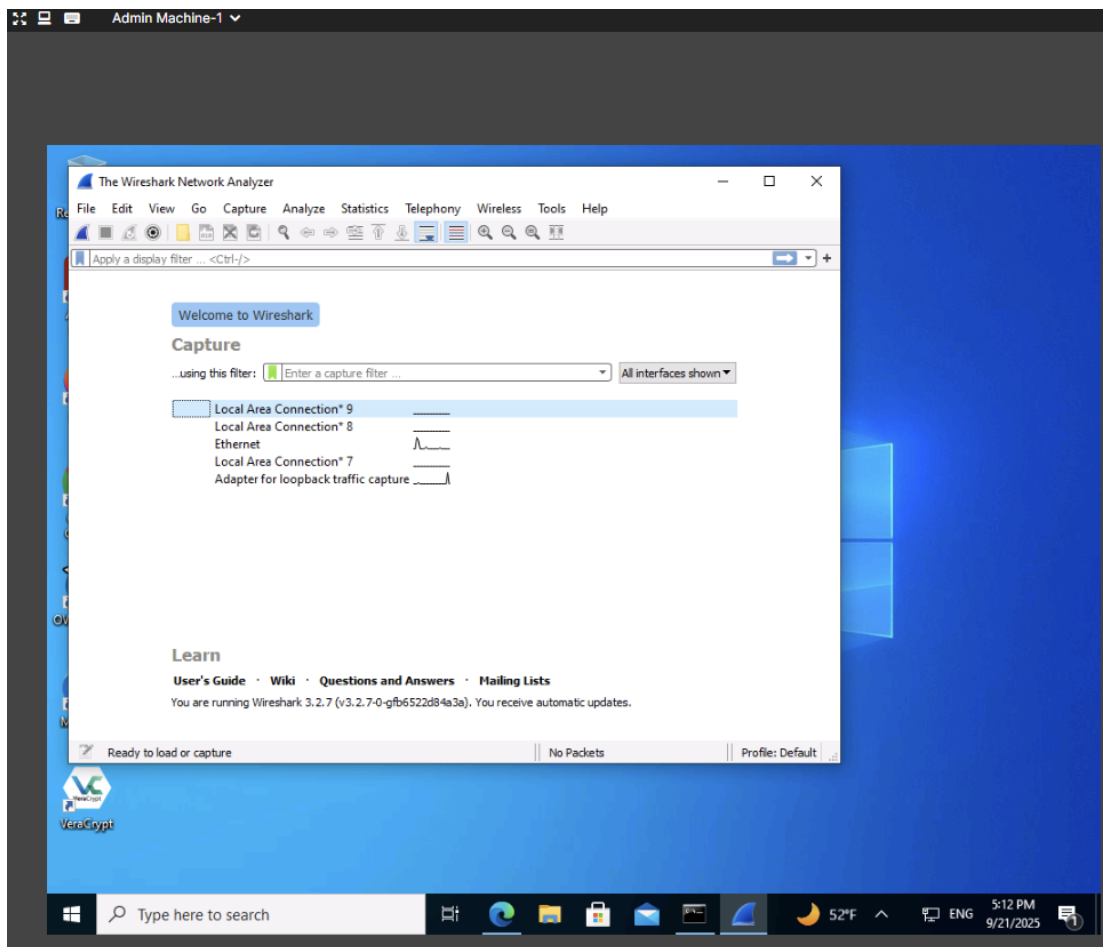


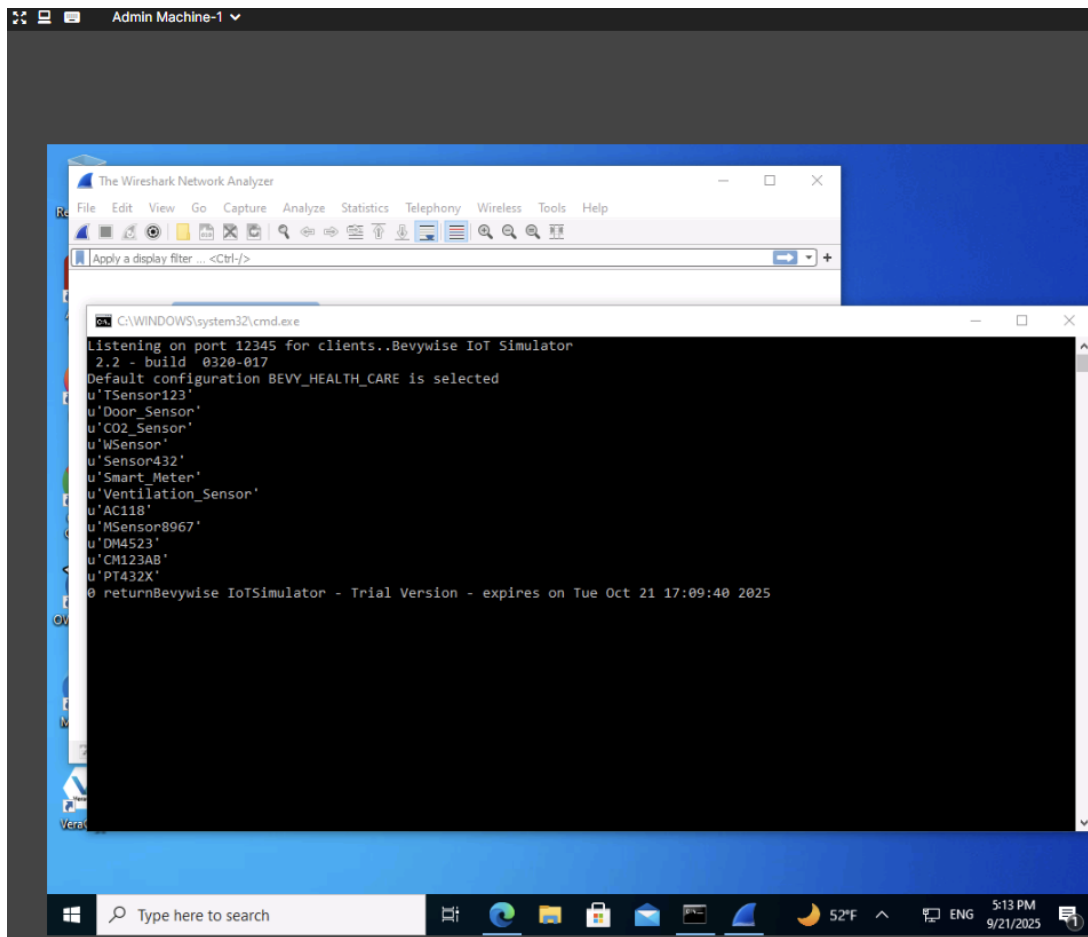


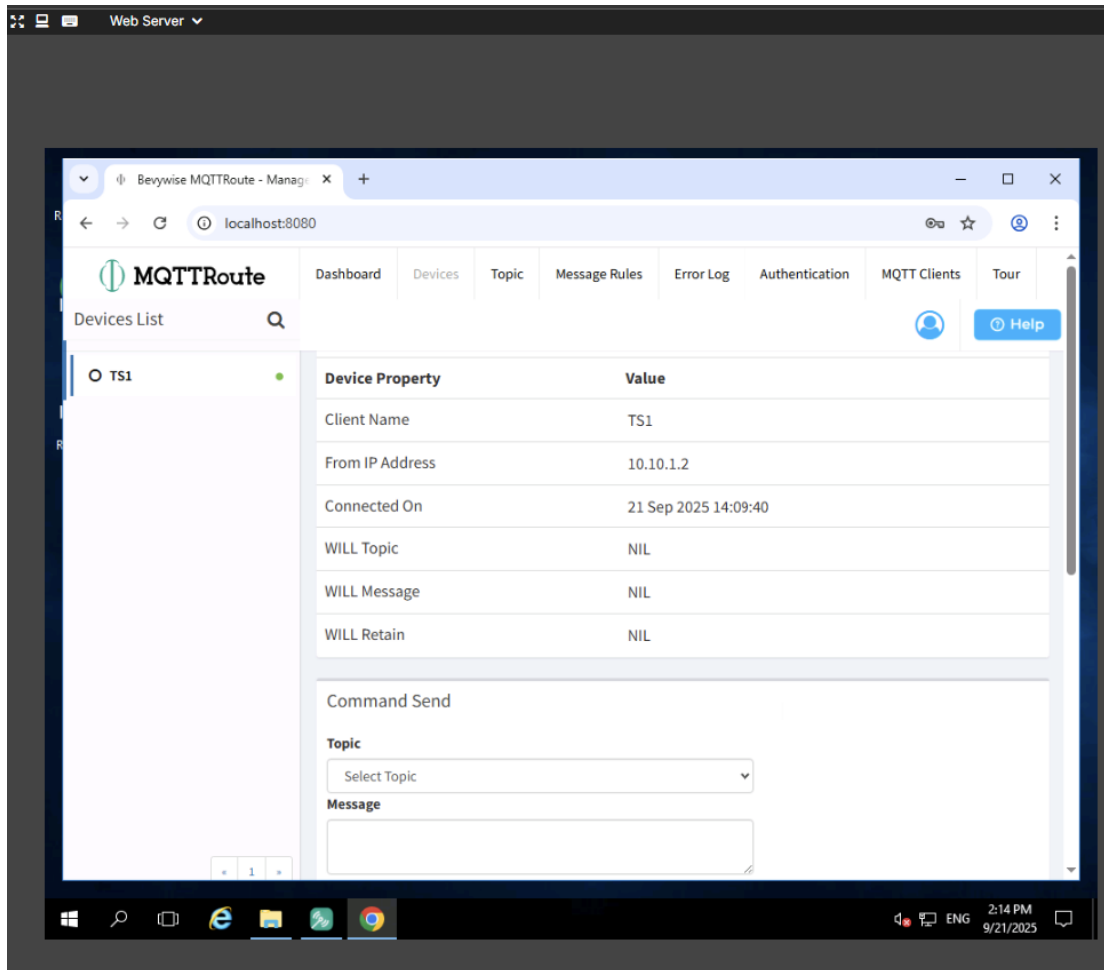


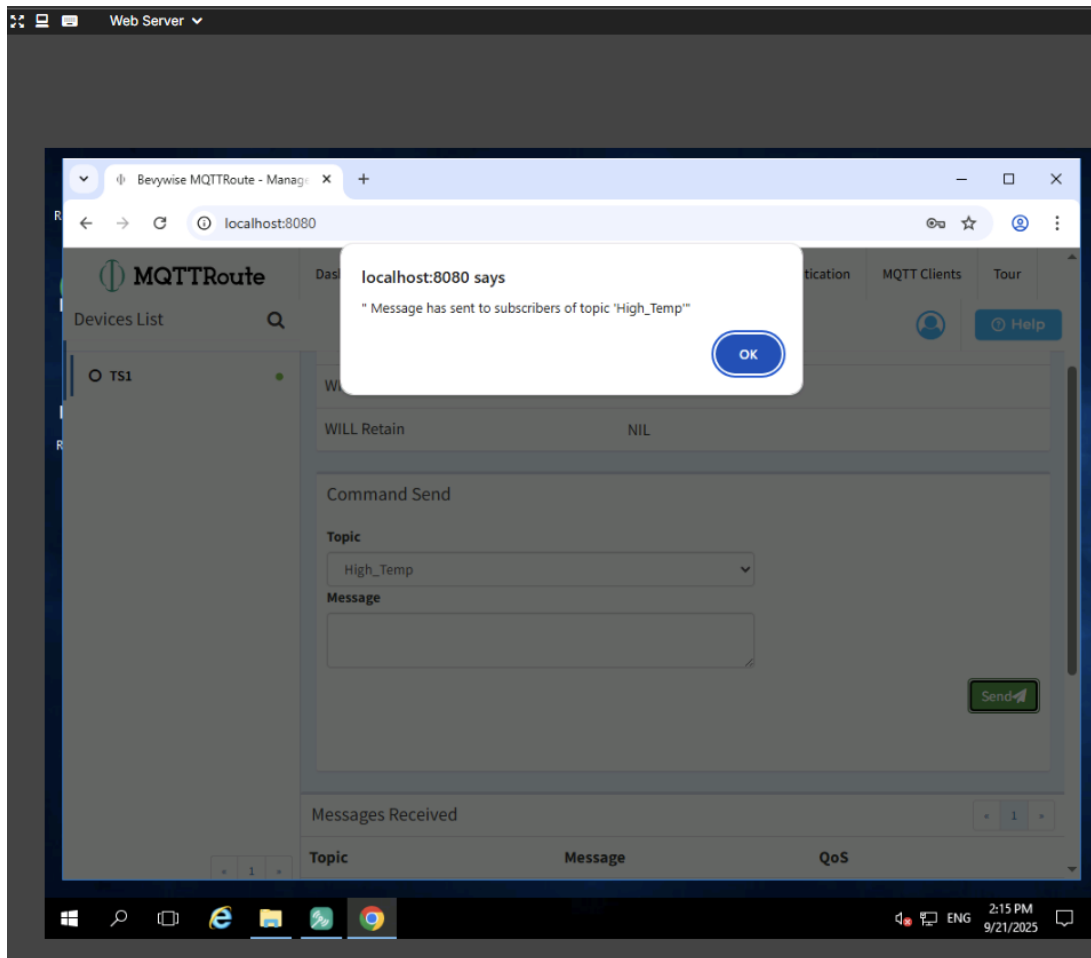


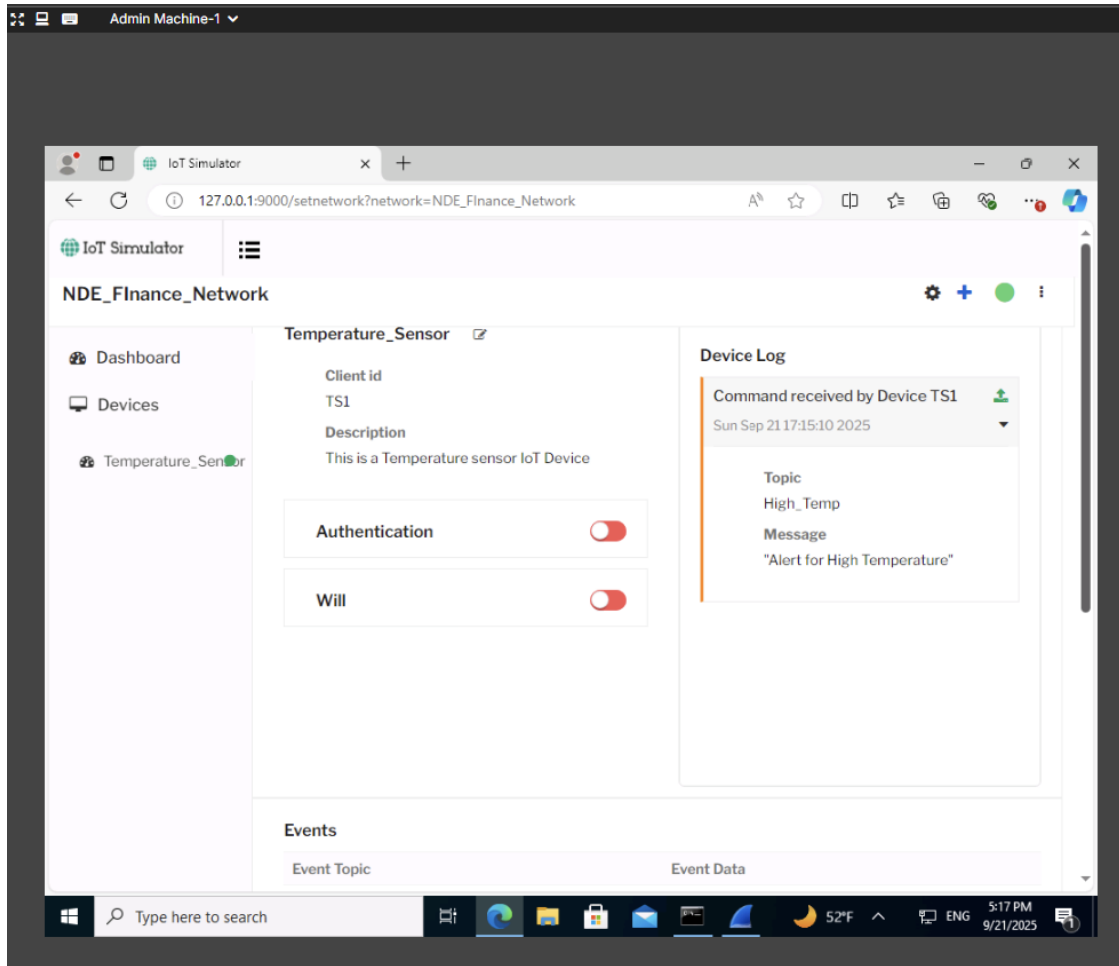
Next, we will capture the traffic between the **virtual IoT network** and the **MQTT Broker** to monitor the secure communication.



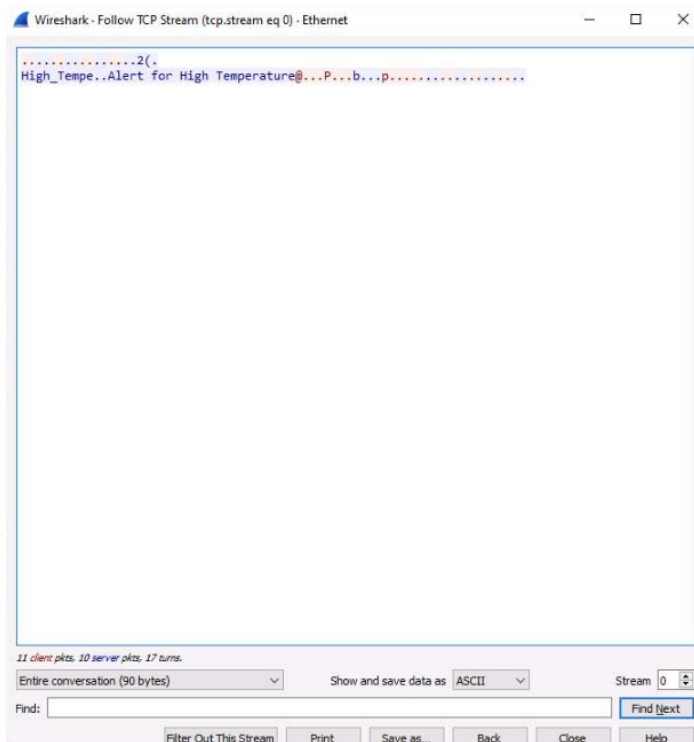
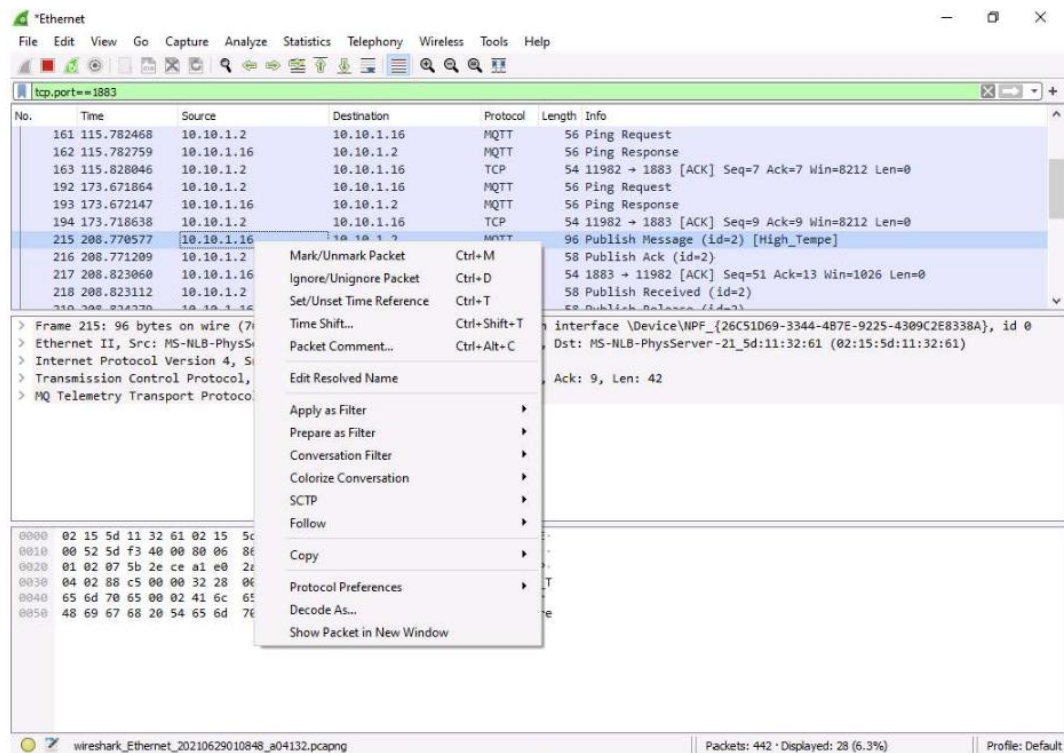




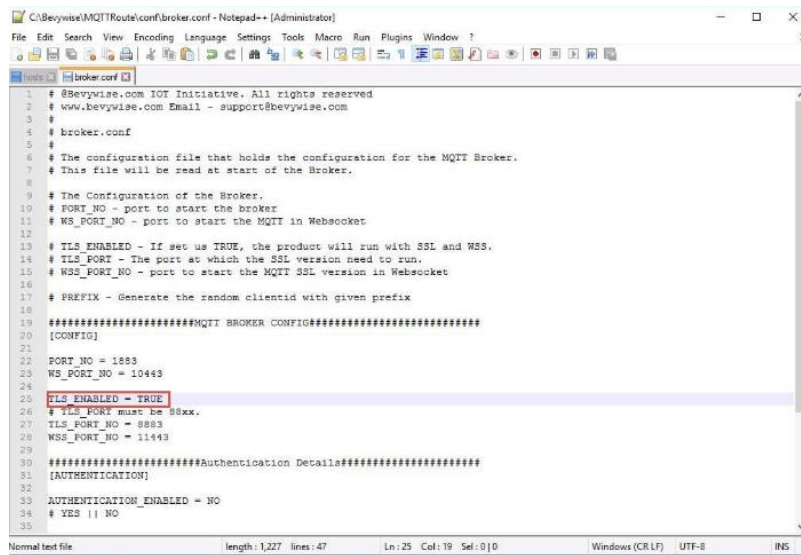




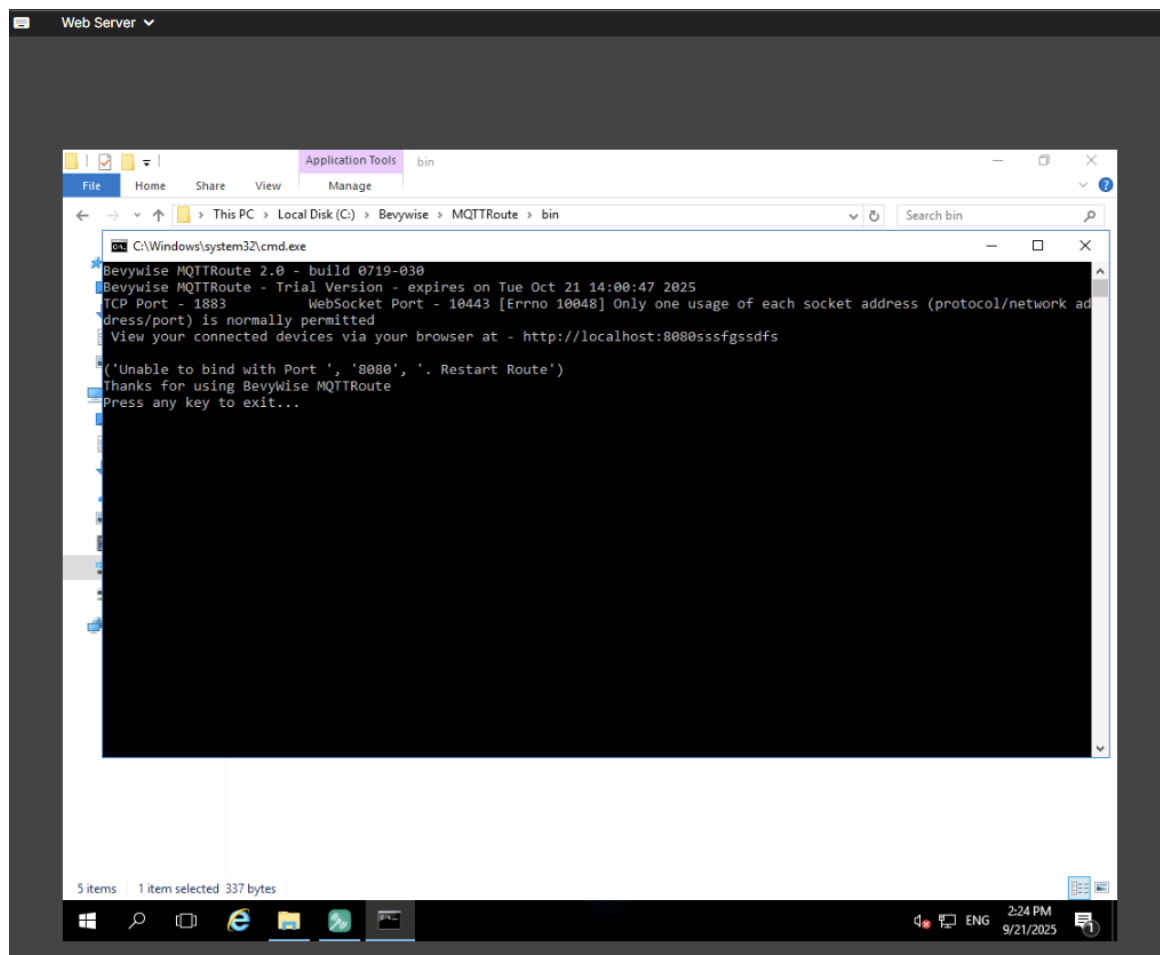




Next, using self-signed certificates, we will implement **TLS/SSL** on the virtual network to ensure a **secure communication** between the device and the server.



```
1 # @Bevywise.com IOT Initiative. All rights reserved
2 # www.bevywise.com Email - support@bevywise.com
3 #
4 # broker.conf
5 #
6 # The configuration file that holds the configuration for the MQTT Broker.
7 # This file will be read at start of the Broker.
8 #
9 # The Configuration of the Broker.
10 # PORT_NO - port to start the broker
11 # WS_PORT_NO - port to start the MQTT in Websocket
12 #
13 # TLS_ENABLED - If set as TRUE, the product will run with SSL and NSS.
14 # TLS_PORT - The port at which the SSL version need to run.
15 # WS_PORT_NO - port to start the MQTT SSL version in Websocket
16 #
17 # PREFIX - Generate the random clientid with given prefix
18 #
19 #####MQTT BROKER CONFIG#####
20 [CONFIG]
21
22 PORT_NO = 1883
23 WS_PORT_NO = 10443
24
25 TLS_ENABLED = TRUE
26 # TLS_PORT must be 88xx.
27 TLS_PORT = 8883
28 WS_PORT_NO = 11443
29
30 #####Authentication Details#####
31 [AUTHENTICATION]
32
33 AUTHENTICATION_ENABLED = NO
34 # YES || NO
35
```



## Lab Summary: Securing IoT Device Communication using TLS/SSL

### Scenario

This lab demonstrated how to secure IoT device communications by using the Bevywise

MQTT Broker and IoT Simulator to create a virtual IoT environment. The goal was to first observe unencrypted communications between virtual devices and the broker, and then implement TLS/SSL to encrypt traffic and protect against interception.

## **Objectives**

The lab was designed to:

- Install and configure the Bevywise MQTT Broker on a Windows server.
- Deploy the Bevywise IoT Simulator on a client machine.
- Create a virtual IoT network and connect simulated devices to the broker.
- Capture traffic in Wireshark to demonstrate unencrypted communication.
- Implement TLS/SSL on the broker and simulator using provided certificates.
- Re-run communication to show encrypted traffic in Wireshark.

## **Outcome**

- The broker and simulator were installed successfully, a network was created, and a simulated IoT device (Temperature\_Sensor) was connected.
- Commands were sent from the broker to the device, and the device logs displayed the alert messages as expected.
- When attempting to capture traffic in Wireshark, the expected “Published Message” in clear text did not appear. Because this step failed, the demonstration of plaintext traffic could not be completed.
- Since the unencrypted capture was missing, the follow-on steps (TLS/SSL implementation and encrypted verification) were rendered inoperable.

## **Reflection**

The lab illustrated the value of TLS/SSL for securing IoT device communications, but its execution failed at the Wireshark verification stage. One limitation of this environment is the lack of real-time error feedback, as a beginner, it’s difficult to identify where mistakes occurred until the final outcome fails. Despite this, the configuration steps made it clear how MQTT brokers and simulators work together, and how TLS/SSL is applied at the broker level to enforce secure communication.