CYT 160 - Security for Cloud and Internet of Things

Final Project – Raspberry Pi Temperature Sensor

Instructor

Professor **Kamyar Ghaderi**

Groups Members

Khang Le 119039253

Md Abid Al Mohaimin 100986249

Introduction

We built a complete IoT temperature monitoring system. A Raspberry Pi with an MCP9808 sensor reads the temperature every 5 seconds and sends the data securely to AWS IoT Core. To protect the system, we added real-time attack detection using Suricata + Kibana on an EC2 instance and we also analyzed network traffic with VPC Flow Logs stored in S3. We simulated two common IoT attacks (DDoS flood and malformed payloads) and proved that our monitoring tools can catch them immediately.

(Part 1) Raspberry Pi & Sensor Setup with AWS IoT Core

In this part, we will explain how we prepared the Raspberry Pi, connected the MCP9808 temperature sensor and linked the device to AWS IoT Core so it can send data securely.

Step 1: Installing Raspberry Pi OS and Basic Configuration:

We started by installing the operating system on the Raspberry Pi:

- We flashed a Raspbian OS image onto a microSD card using Raspberry Pi Imager.
- We inserted the SD card into the Raspberry Pi 3B and powered it on.
- On the first boot, we completed the basic setup: language, keyboard layout, Wi-Fi network, and a new password.

Step 2: Hardware Setup: Raspberry Pi and MCP9808 Sensor

Next, we assembled the hardware. In Raspberry Pi 3B, we installed the microSD card, the MCP9808 temperature sensor, a 4-pin cable, and jumper wires.

We wired the sensor to the Pi's GPIO header as recommended:

- VIN on the sensor to 3.3 V on the Pi
- GND on the sensor to GND on the Pi
- SDA on the sensor to SDA pin on the Pi
- SCL on the sensor to SCL pin on the Pi

At this stage, the physical IoT device was ready.

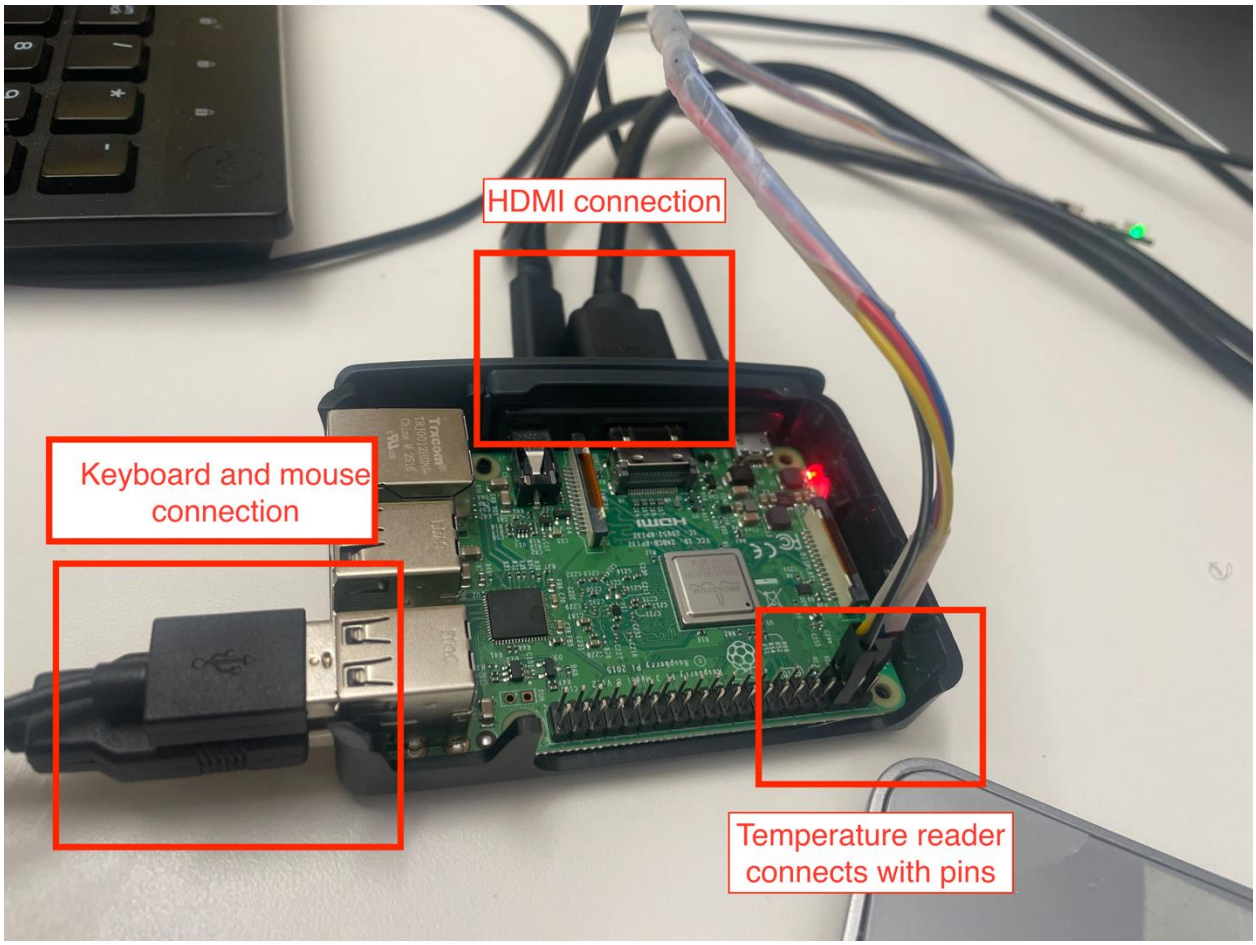


Figure: Photo of the Raspberry Pi and MCP9808 sensor connected on the desk.

Step 3: Testing the Temperature Sensor

We then followed the lab steps to set up Python, install the MCP9808 library, and create a small test script.

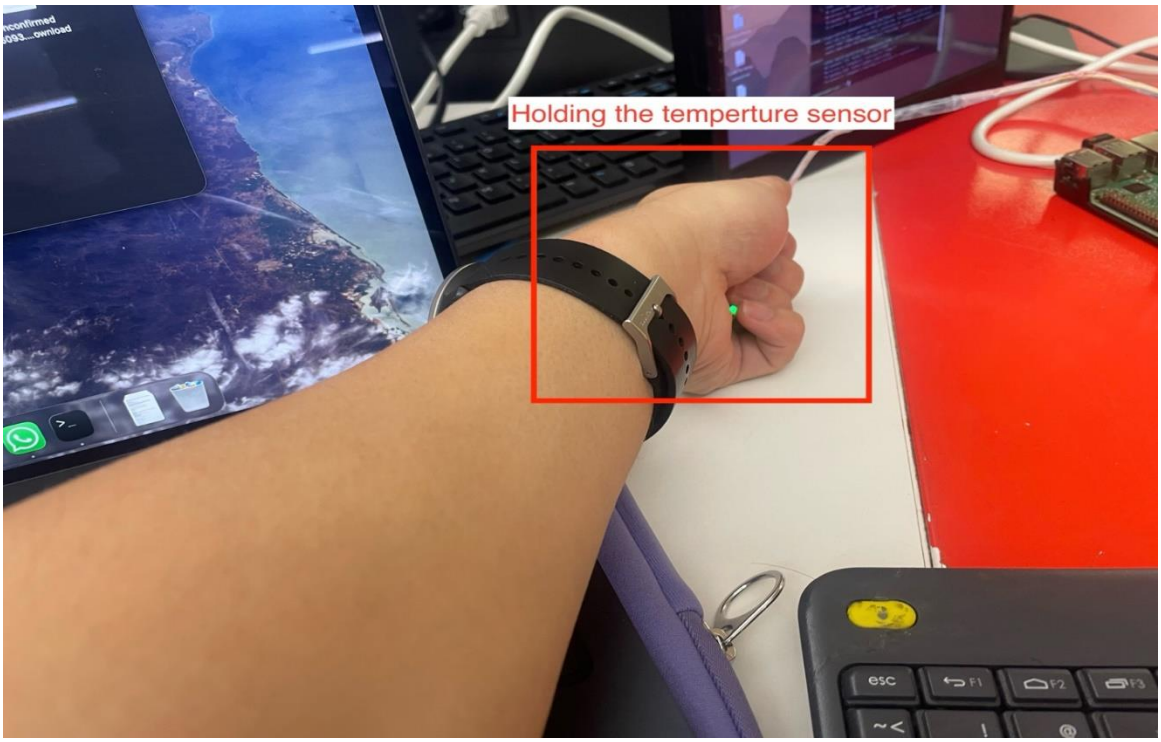
The script:

- Initializes the I2C bus
- Creates a MCP9808 sensor object
- Prints the temperature in Celsius when it runs

When we ran the test, the sensor showed a value in a normal room range (around 20–21 °C). When we touched the sensor, the reported temperature increased; when we let it cool, the value went down. This confirmed that:

- The Raspbian image and Raspberry Pi OS were installed correctly

- I2C was working
- The MCP9808 sensor was wired and responding as expected




```
khanghacker — dammy@raspberrypi: ~ — ssh dammy@172.20.10.7 — 8...

Last login: Fri Nov 28 17:52:04 2025 from 192.168.137.1
[dammy@raspberrypi:~ $ source iot_project_env/bin/activate
[(iot_project_env) dammy@raspberrypi:~ $ ls
Desktop     Downloads        Music     Public          Templates
Documents   iot_project_env  Pictures  sensor_test.py  Videos
[(iot_project_env) dammy@raspberrypi:~ $ python3 sensor_test.py
 Temperature: 21.875 °C
[(iot_project_env) dammy@raspberrypi:~ $ python3 sensor_test.py
 Temperature: 23.6875 °C
[(iot_project_env) dammy@raspberrypi:~ $ python3 sensor_test.py
 Temperature: 24.0 °C
[(iot_project_env) dammy@raspberrypi:~ $ python3 sensor_test.py
 Temperature: 24.5 °C
[(iot_project_env) dammy@raspberrypi:~ $ python3 sensor_test.py
 Temperature: 24.9375 °C
[(iot_project_env) dammy@raspberrypi:~ $ python3 sensor_test.py
 Temperature: 25.1875 °C
[(iot_project_env) dammy@raspberrypi:~ $ python3 sensor_test.py
 Temperature: 25.5625 °C
[(iot_project_env) dammy@raspberrypi:~ $ python3 sensor_test.py
 Temperature: 25.75 °C
[(iot_project_env) dammy@raspberrypi:~ $ python3 sensor_test.py
 Temperature: 25.625 °C
 (iot_project_env) dammy@raspberrypi:~ $
```

Screenshot: Terminal output showing the increasing temperature reading from the sensor as the reading is being hold.

Step 4: Registering the Raspberry Pi as an AWS IoT Thing

After the local sensor test was working, we registered the Raspberry Pi in AWS IoT Core so it could act as a managed IoT device in the cloud.

In AWS Academy Learner Lab, we opened the AWS console, went to AWS IoT Core, and used the “Connect one device” wizard with:

- Platform: Linux
- Language/SDK: Python

We created a new IoT Thing for the Raspberry Pi. During this process, AWS IoT Core generated a connection kit that contained:

- the Amazon Root CA
- a device certificate
- a private key
- a helper script (start.sh) to test the connection

We downloaded this kit and copied the files to the Raspberry Pi. These certificates and keys let the Pi prove its identity and connect securely to AWS IoT Core over TLS.

Step 5: Verifying Connectivity with start.sh and MQTT Test Client

After registering the device, we used the start.sh script from the connection kit to confirm that the Raspberry Pi could connect to AWS IoT Core. On the Pi, we ran the script, which used the endpoint, certificate, and key from the kit to open a secure MQTT connection.

In the AWS IoT Core console, we opened the MQTT test client, subscribed to the topic used by the script, and saw the test messages from the Pi appear in real time.

This quick test confirmed that:

- the Raspberry Pi could reach AWS IoT Core over the internet
- the certificates and private key were valid and matched the Thing
- MQTT over TLS was working end to end for our device

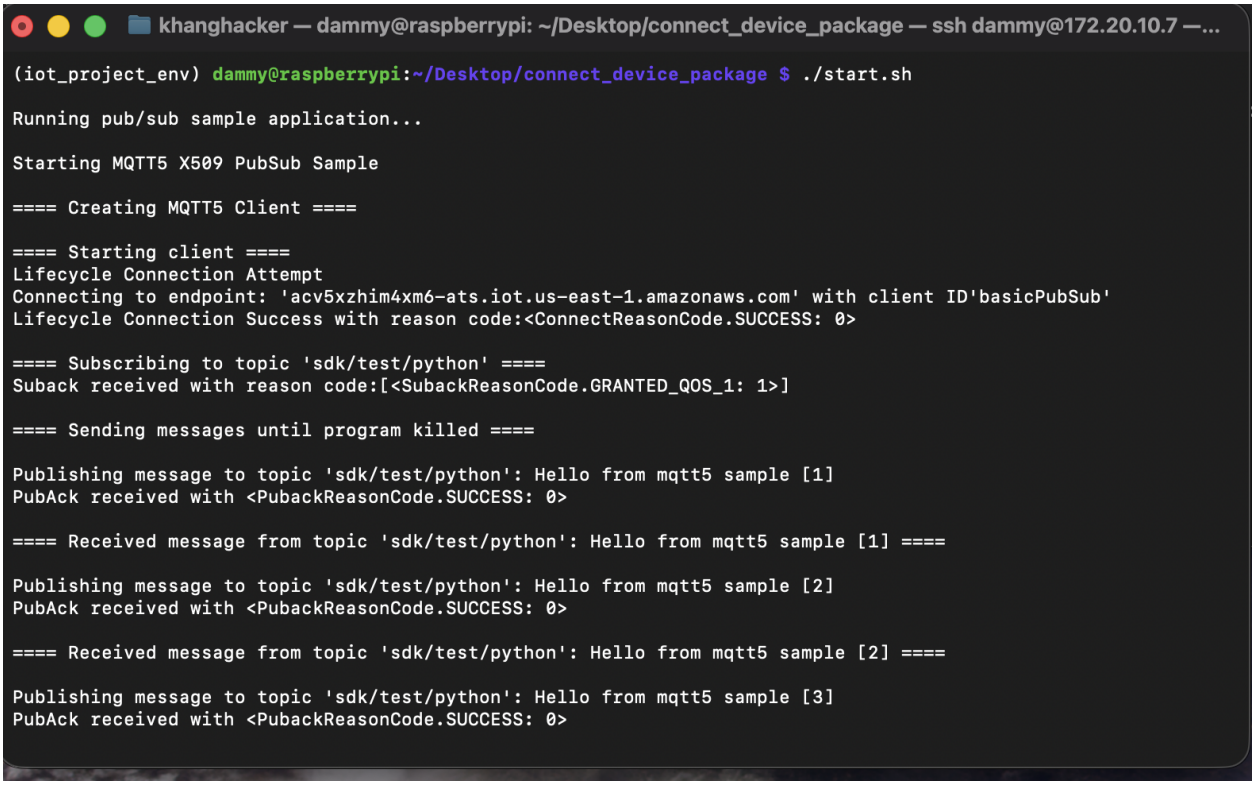


Figure: Terminal on the Raspberry Pi showing the start.sh script running and indicating a successful connection

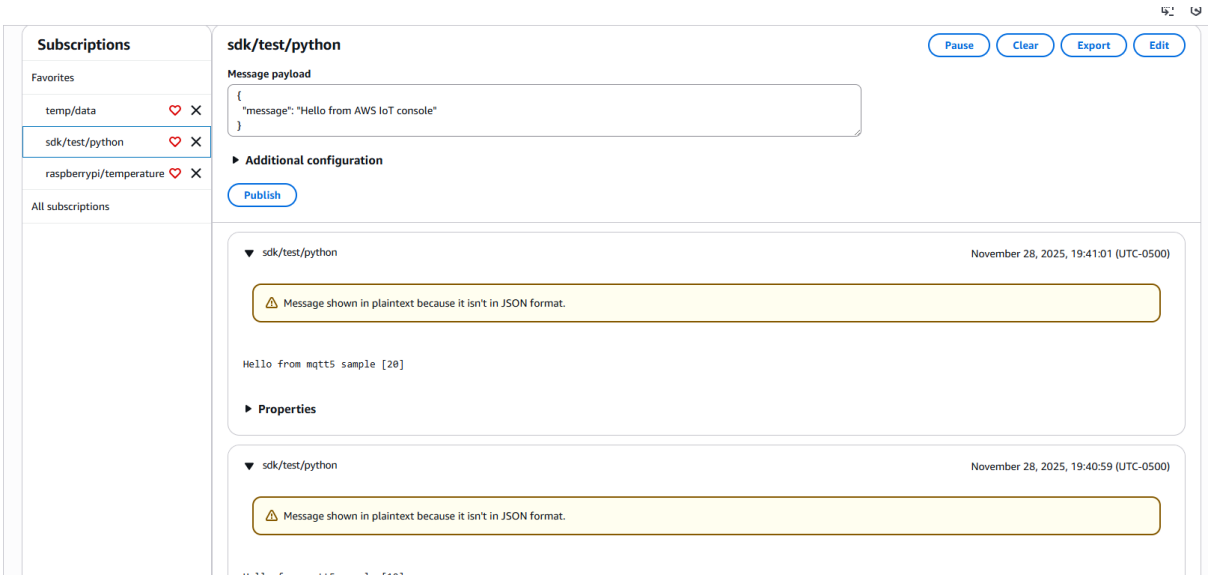


Figure: AWS IoT Core MQTT test client view with incoming messages from the Raspberry Pi.

(Part 2) – Continuous Temperature Publishing to AWS IoT Core

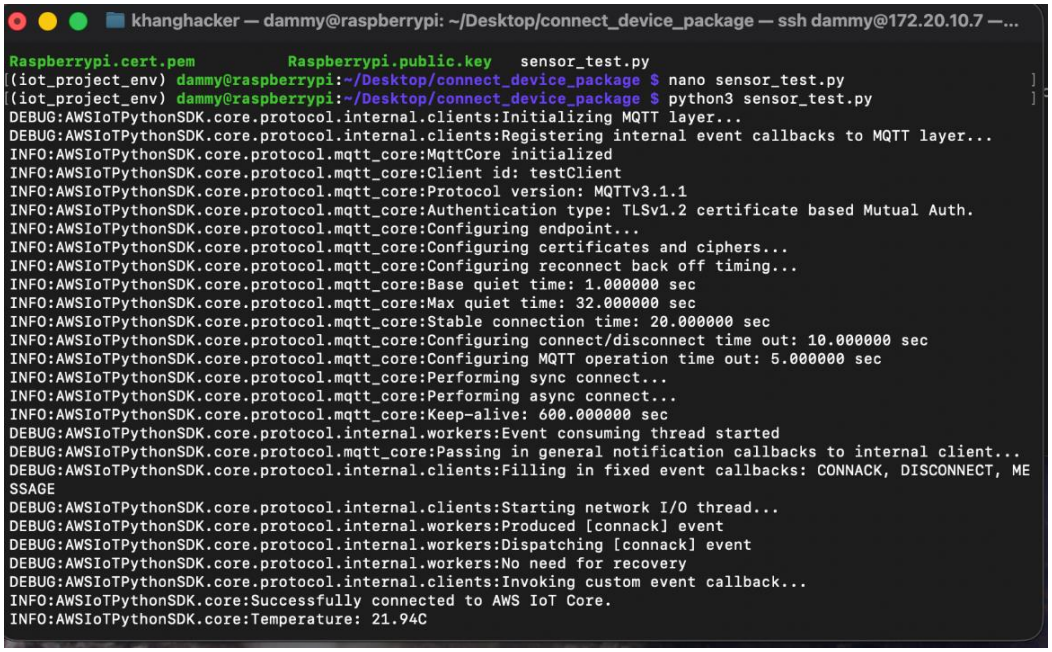
In this step, we moved from a one-time sensor test to a real IoT workload. We updated our Python script so the Raspberry Pi reads the MCP9808 sensor in a loop and sends temperature data to AWS IoT Core every few seconds. We also adjusted the IoT policy and used the MQTT test client to confirm that the data stream was continuous and stable.

Step 1: Updating the Python Script for Continuous Publishing

We took the original sensor_test.py script, which printed one temperature reading, and turned it into a continuous IoT client for AWS IoT Core. We:

- kept the I2C and MCP9808 sensor-reading code
- added logging to show each reading and publish event
- imported and configured the AWS IoT Python SDK with our endpoint, root CA, device certificate, and private key
- changed the script to connect once, then loop to:
    - read the temperature in Celsius
    - log the value
    - build a JSON payload
    - publish to the raspberrypi/temperature topic with QoS 1
    - wait about five seconds before the next reading
- added basic error handling so temporary network issues are logged instead of stopping the script

This turned the one-time test script into a continuous publisher that sends live temperature data from the Raspberry Pi to AWS IoT Core.



Screenshot: Terminal on the Raspberry Pi showing log lines such as "Temperature: x C" and "Message published to AWS IoT Core."

Step 2: Customizing Endpoint, Certificates, and Thing Details

In this step, we customized the Python script so that we can use our own AWS IoT settings instead of example values. Based on the guideline, we:

- replaced the example AWS IoT endpoint with our actual AWS IoT Core endpoint
- updated rootCAPath with the correct path to the **Amazon Root CA** file on the Raspberry Pi
- updated certificatePath with the correct path to our **device certificate** file
- updated privateKeyPath with the correct path to our **device private key** file
- kept the MQTT topic in the code as raspberrypi/temperature

We also kept the MQTT client configuration, including the endpoint port (8883) and the reconnect and timeout settings provided in the sample code.

Step 3: Subscription in AWS IoT MQTT Test Client

To confirm that the continuous temperature data was reaching AWS IoT Core, we used the MQTT Test Client as described. Following the project guide, we:

- opened the AWS IoT Console
- selected Test → MQTT test client from the menu
- entered raspberrypi/temperature as the subscription topic
- set the QoS level in the test client to 1, matching the QoS used in the Python script

- clicked Subscribe to start listening on the topic
- kept the Python script running on the Raspberry Pi so it continued to publish temperature messages
- observed the messages appearing in the MQTT Test Client while the script was sending data

(Part 3) – IoT Security Monitoring on EC2 (Suricata, Filebeat, Elasticsearch, Kibana)

In this part, we built a monitoring server on AWS EC2. The EC2 instance runs Mosquitto to receive mirrored MQTT traffic from the Raspberry Pi, Suricata to inspect that traffic, Filebeat to send Suricata logs to Elasticsearch, and Kibana to visualize alerts.

Step 1: EC2 Monitoring Instance and Mosquitto Setup

We first created an EC2 instance to host all security tools. We:

- logged into AWS Academy Learner Lab and opened the AWS Management Console
- launched an EC2 instance with:
  - Ubuntu Server 22.04 LTS
  - t2.medium (2 vCPUs, 4 GiB RAM)
  - an SSH key pair
  - default VPC and subnet with a public IP
- created a security group that allowed:
  - SSH on port 22 (from our IP)
  - MQTT on port 1883 (from the Raspberry Pi or from anywhere for the lab)
- kept the default 8 GiB root volume
- allocated an Elastic IP and attached it to the instance
- connected over SSH with the .pem key and updated packages

Then we installed and set up Mosquitto so the EC2 instance could act as an MQTT broker:

- installed Mosquitto and its client tools
- configured Mosquitto to listen on port 1883 and allow anonymous access for the lab
- enabled and started the Mosquitto service so it runs on boot

Step 2: Suricata Installation and Rule Configuration

Next, we installed Suricata on the EC2 instance so it could monitor MQTT traffic. We:

- installed Suricata on Ubuntu and updated its rules
- checked the EC2 network interface with ip a and noted the correct name (for example, enX0)

- edited the Suricata config to:
  - use af-packet on that interface
  - enable eve-log and write JSON logs to /var/log/suricata/eve.json
- added custom rules to:
  - detect high MQTT traffic to port 1883 (DDoS-style bursts)
  - detect unusually large MQTT packets on port 1883
- reloaded Suricata to apply the new rules
- enabled and started the Suricata service so it continuously monitors the interface

Step 3: Filebeat Setup to Send Suricata Logs to Elasticsearch

To send Suricata alerts into Elasticsearch, we installed Filebeat on the same EC2 instance. We:

- downloaded and installed the Filebeat 8.12.0 .deb package
- edited the Filebeat config to:
  - enable a log input
  - read from /var/log/suricata/eve.json
  - enable JSON parsing so fields from eve.json are mapped correctly
- set the output to Elasticsearch for easier filtering in Kibana
- enabled and started the Filebeat service so it runs on boot and continuously ships new Suricata events to Elasticsearch

Step 4: Elasticsearch Installation and Heap Size Tuning

We installed Elasticsearch on the EC2 instance to index Suricata logs from Filebeat. We:

- added the Elastic APT repository and GPG key, then installed Elasticsearch with the package manager
- edited elasticsearch.yml to:
  - bind http.host / network.host to 0.0.0.0 so Kibana on the same host can reach it
  - enable security (xpack.security.enabled: true)
  - disable HTTP SSL for the lab (xpack.security.http.ssl.enabled: false)
  - set cluster.initial_master_nodes using the EC2 private hostname
- enabled and started the Elasticsearch service

After the first run, we set the password for the elastic user with elasticsearch-reset-password and used NewPass123! later in Filebeat and Kibana.To keep Elasticsearch running smoothly on a t2.medium instance, we also tuned the heap size. We:

- created /etc/elasticsearch/jvm.options.d/heap.options with:
  - -Xms512m
  - -Xmx512m
- fixed ownership and permissions on this file and directory

- commented out the default -Xms2g and -Xmx2g lines in jvm.options to avoid conflicts
- restarted Elasticsearch and checked the service status.

Step 5: Kibana Configuration and Credentials

We installed and configured Kibana so we could visualize Suricata alerts in Elasticsearch. We:

- installed Kibana on the same EC2 instance
- edited kibana.yml to:
  - set server.host to 0.0.0.0
  - keep the port as 5601
  - set elasticsearch.hosts to ["http://localhost:9200"]
- set the Elasticsearch username in kibana.yml to the internal Kibana user (for example, kibana_system) with the correct password
- restarted Kibana and checked that the service was running
- opened a browser at http://<Elastic-IP>:5601 to access the Kibana web interface

Step 6: Verifying All Monitoring Services on the EC2 Instance

At the end of this part, we verified that all monitoring services were running correctly on the EC2 instance. We:

- checked the status of Mosquitto, Suricata, Filebeat, Elasticsearch, and Kibana with systemctl status and confirmed that each service was active.

This showed that the EC2 monitoring stack was ready to receive mirrored MQTT traffic from the Raspberry Pi, generate Suricata alerts, send them to Elasticsearch, and visualize them in Kibana. In the next part of the project, we use this setup to run simulated attacks and analyze the alerts

(Part 4) – VPC Flow Logs and S3 Setup

In this part, we focused on network-level forensics using AWS VPC Flow Logs. The goal was to capture metadata about all traffic in the VPC where our monitoring EC2 instance is running and store it in S3. Then we manually downloaded the log files so we could later look for signs of our simulated IoT attacks, such as high-frequency connections and heavy MQTT traffic to port 1883.

Step 1: Setting Up S3 Storage for VPC Flow Logs

We first created an S3 bucket to store the VPC Flow Log files.We:

- opened the AWS Management Console and went to S3

- created a new bucket (project3-vpc-logs) in the same region as our EC2 monitoring instance
- kept default encryption enabled
- unchecked “Block all public access” as instructed in the guide
- created the bucket and confirmed it appeared in the S3 bucket list

Step 2: Generating Normal and Attack Traffic

To make the VPC Flow Logs meaningful, we needed real traffic that includes both normal and malicious patterns. We:

- ran our existing Raspberry Pi script again so it sent normal temperature data to AWS IoT Core and mirrored MQTT traffic to the EC2 Mosquitto broker
- triggered our simulated attack traffic from the Pi, including:
    - bursts of high-frequency MQTT messages (DDoS-style)
    - malformed payloads towards the EC2 Mosquitto broker
- generated some normal comparison traffic in the VPC, such as:
    - simple ping or curl traffic, as suggested in the overall project description

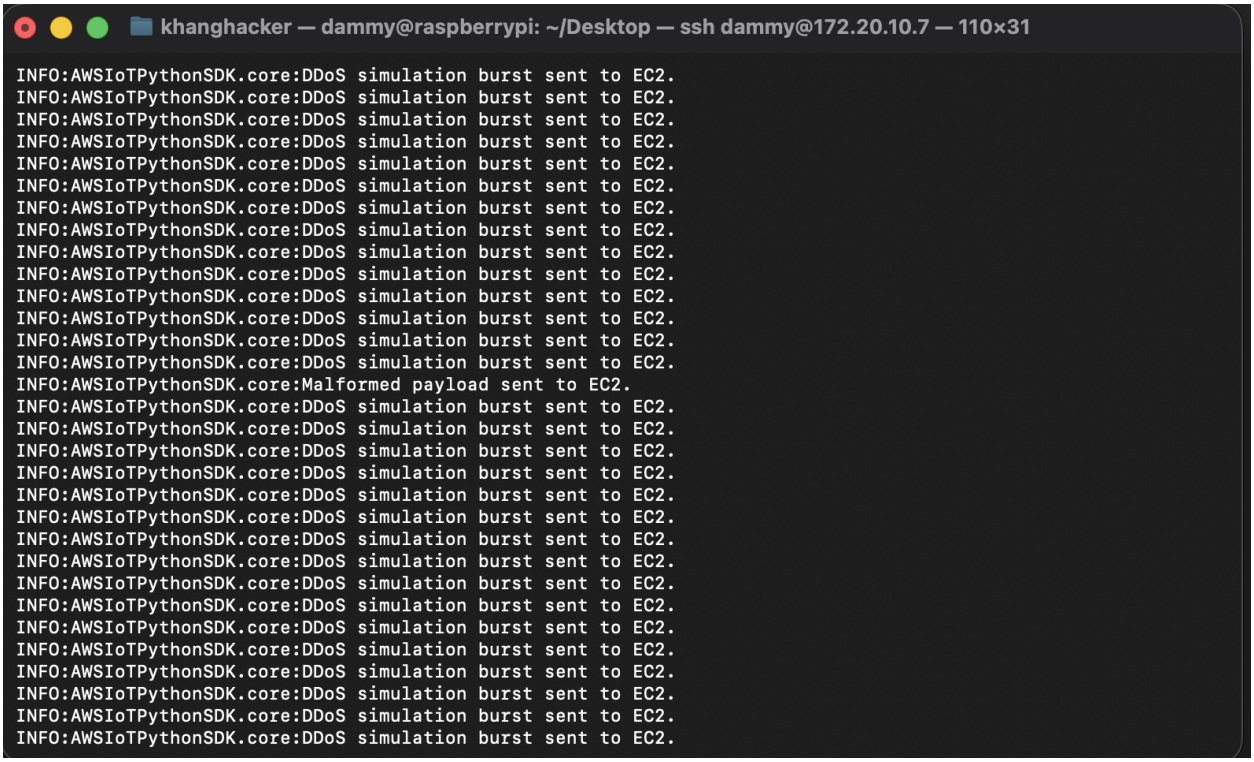


Figure: Terminal on the Raspberry Pi showing the attack script running (normal + DDoS + malformed payloads)

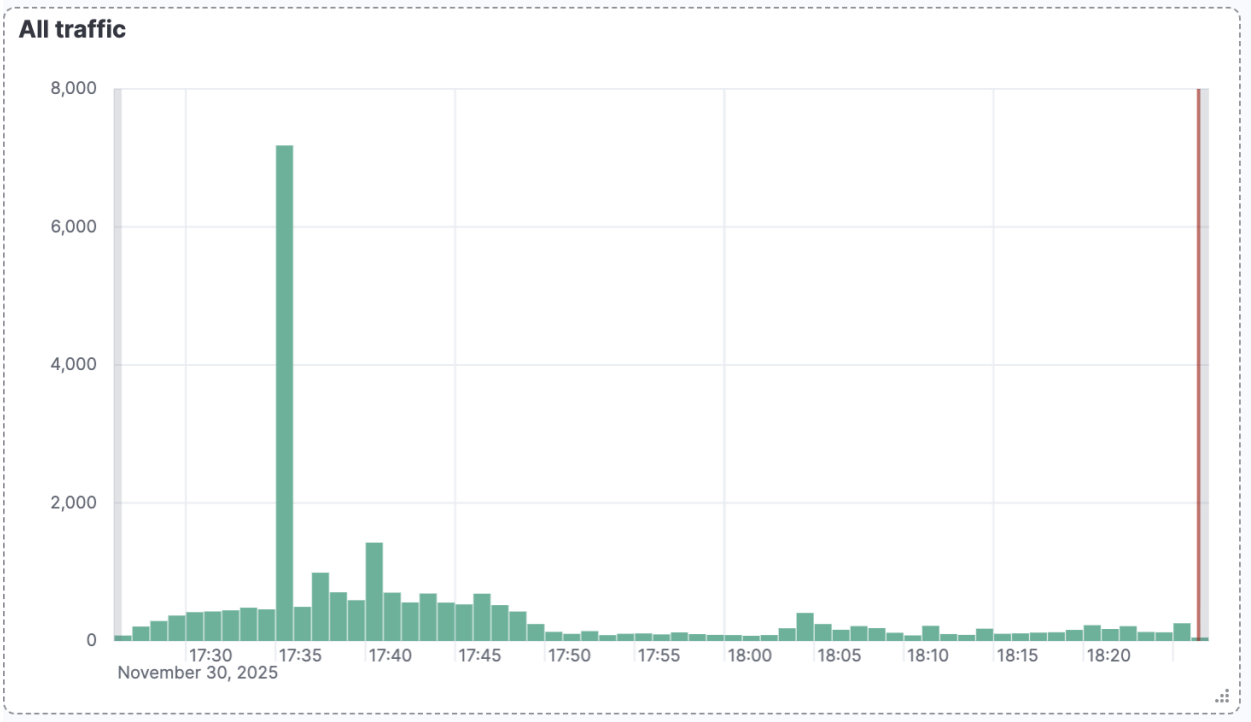


Figure: Screenshot of basic ping or other normal traffic commands used for comparison.

Figure: Screenshot shows in above time, there is large number of activities at 17:35, with over 5000 DDoS Attacks and 800 Malform packages were capture.

(Part 5) – VPC Flow Log Analysis and Security Findings

In this section, we present the results of analyzing the VPC Flow Logs. We downloaded the log files from S3, opened them in Excel, and used filtering and sorting to focus on MQTT traffic (port 1883), high packet and byte counts, and REJECT flows.

VPC Flow Log Analysis (S3 Logs)

Each row shows one network flow with source IP, destination IP, ports, packet count, byte count, timestamps, and an action (ACCEPT or REJECT). In total, the file has **1,291** flows:

- **ACCEPT:** 980 flows
- **REJECT:** 310 flows
- 1 flow with a different status

Most of this traffic is related to our EC2 monitoring instance with private IP 172.31.30.62, which is expected because it runs Mosquitto, Suricata, Elasticsearch, Kibana, and receives mirrored MQTT traffic.

Suspicious IPs and Their Behavior

Following the project instructions, we looked for many requests from one source address and high packet/byte counts

- When we filtered for flows using port 1883 (MQTT), we found 28 flows. These flows are almost all between:
  - 172.31.30.62 (EC2 monitoring instance) and
  - 69.158.246.215 (attack source)

	version	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
2	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	991	51534	1764383411	1764383450	ACCEPT	OK
5	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	961	49986	1764383463	1764383519	ACCEPT	OK
7	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	954	49610	1764383646	1764383695	ACCEPT	OK
9	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	954	49610	1764383522	1764383579	ACCEPT	OK
10	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	954	49622	1764383589	1764383639	ACCEPT	OK
13	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	948	49298	1764383062	1764383099	ACCEPT	OK
14	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	935	48646	1764383339	1764383393	ACCEPT	OK
16	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	928	48258	1764383901	1764383931	ACCEPT	OK
19	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	901	46902	1764383783	1764383819	ACCEPT	OK
21	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	899	46770	1764383719	1764383756	ACCEPT	OK
23	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	874	45450	1764383840	1764383878	ACCEPT	OK
51	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	1883	46526	6	85	33451	1764383646	1764383695	ACCEPT	OK
52	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	1883	46526	6	85	33451	1764383646	1764383695	ACCEPT	OK
995	2	905418058769	eni-0aadd41a6546b32ft	162.216.149.9	172.31.30.62	51883	13020	6	1	44	1764289732	1764289776	REJECT	OK
1203	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	185.73.23.133	1883	47199	6	1	52	1764383522	1764383579	ACCEPT	OK

	version	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
3	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	990	745701	1764383411	1764383450	ACCEPT	OK
4	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	962	741782	1764383463	1764383519	ACCEPT	OK
6	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	954	743840	1764383646	1764383695	ACCEPT	OK
8	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	954	742565	1764383522	1764383579	ACCEPT	OK
11	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	954	741294	1764383589	1764383639	ACCEPT	OK
12	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	949	742197	1764383062	1764383099	ACCEPT	OK
15	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	934	745412	1764383339	1764383393	ACCEPT	OK
17	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	928	741241	1764383901	1764383931	ACCEPT	OK
18	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	906	751370	1764383783	1764383819	ACCEPT	OK
20	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	900	733658	1764383719	1764383756	ACCEPT	OK
22	2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	882	738841	1764383840	1764383878	ACCEPT	OK
47	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	46526	1883	6	90	4716	1764383646	1764383695	ACCEPT	OK
48	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	46526	1883	6	89	4722	1764383646	1764383695	ACCEPT	OK
507	2	905418058769	eni-0aadd41a6546b32ft	185.73.23.133	172.31.30.62	47199	1883	6	2	98	1764383522	1764383579	ACCEPT	OK

- For these MQTT flows, the packet and byte counts are much higher than normal:
    - Many flows on port 1883 have around 900–990 packets.
    - Several flows carry tens or hundreds of thousands of bytes (up to around 750,000 bytes in a single flow).

version	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	991	51534	1764383411	1764383450	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	990	745701	1764383411	1764383450	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	962	741782	1764383463	1764383519	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	961	49986	1764383463	1764383519	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	954	743840	1764383646	1764383695	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	954	49610	1764383646	1764383695	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	954	742565	1764383522	1764383579	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	954	49610	1764383522	1764383579	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	954	49622	1764383589	1764383639	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	954	741294	1764383589	1764383639	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	949	742197	1764383062	1764383099	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	1883	33610	6	948	49298	1764383062	1764383099	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	69.158.246.215	1883	33610	6	935	48646	1764383339	1764383393	ACCEPT	OK
2	905418058769	eni-0aadd41a6546b32ft	69.158.246.215	172.31.30.62	33610	1883	6	934	745412	1764383339	1764383393	ACCEPT	OK

This shows that 69.158.246.215 acts as a “heavy talker” towards our EC2 instance on the MQTT port. The repeated, very large flows between this IP and 172.31.30.62 match our simulated attack traffic, where we sent bursts of MQTT messages to the Mosquitto broker.

Ports Targeted

From the log, the main ports are:

- **Port 1883 (MQTT):** This is the key port for our IoT security test. It is clearly the target of high-volume traffic between 69.158.246.215 and 172.31.30.62. All flows on this port are marked ACCEPT, which makes sense because we allowed MQTT traffic to reach Mosquitto.

- **Port 22 (SSH):** Used for managing the EC2 instance from our own IP. Packet and byte counts are moderate and match normal admin sessions.

- **Port 5601 (Kibana) and 9200 (Elasticsearch):** These flows show our browser and services connecting to the monitoring stack. They are expected and do not show the same “burst” pattern as the MQTT attack flows.

1	version	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
368	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	142.204.17.59	5601	36844	6	7	1160	1764383380	1764383427	ACCEPT	OK
462	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	142.204.17.59	5601	17218	6	4	2954	1764383646	1764383695	ACCEPT	OK
470	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	142.204.17.59	5601	17218	6	4	1884	1764383783	1764383819	ACCEPT	OK
491	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	142.204.17.59	5601	17218	6	3	988	1764383719	1764383756	ACCEPT	OK
505	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	142.204.17.59	5601	36844	6	2	948	1764383440	1764383485	ACCEPT	OK
511	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	142.204.17.59	5601	17218	6	2	92	1764383901	1764383931	ACCEPT	OK
1191	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	142.204.17.59	5601	36844	6	1	52	1764383501	1764383544	ACCEPT	OK
1218	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	142.204.17.59	5601	36844	6	1	40	1764383559	1764383610	ACCEPT	OK
1274	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	142.204.17.59	5601	17218	6	1	52	1764383840	1764383878	ACCEPT	OK

1	version	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
369	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	36844	5601	6	7	3748	1764383380	1764383427	ACCEPT	OK
424	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	17218	5601	6	5	2475	1764383646	1764383695	ACCEPT	OK
469	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	17218	5601	6	4	2005	1764383719	1764383756	ACCEPT	OK
471	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	17218	5601	6	4	1532	1764383783	1764383819	ACCEPT	OK
474	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	61373	5601	6	4	208	1764386373	1764386399	ACCEPT	OK
492	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	17218	5601	6	3	138	1764383901	1764383931	ACCEPT	OK
504	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	36844	5601	6	2	92	1764383440	1764383485	ACCEPT	OK
508	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	36844	5601	6	2	92	1764383559	1764383610	ACCEPT	OK
1200	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	36844	5601	6	1	46	1764383501	1764383544	ACCEPT	OK
1268	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	17218	5601	6	1	46	1764383840	1764383878	ACCEPT	OK
1287	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	30388	5601	6	1	52	1764386373	1764386399	ACCEPT	OK
1289	2	905418058769	eni-0aadd41a6546b32ft	142.204.17.59	172.31.30.62	54312	5601	6	1	52	1764386373	1764386399	ACCEPT	OK

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	version	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
49	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	59300	6	87	4978	1764383801	1764383848	ACCEPT	OK
50	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	59300	6	87	4978	1764383801	1764383848	ACCEPT	OK
107	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	34768	6	22	1520	1764383801	1764383848	ACCEPT	OK
108	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	34768	6	22	1520	1764383801	1764383848	ACCEPT	OK
191	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	46854	6	14	973	1764383859	1764383909	ACCEPT	OK
192	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	46854	6	14	973	1764383859	1764383909	ACCEPT	OK
205	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	58414	6	13	919	1764383783	1764383819	ACCEPT	OK
206	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	58414	6	13	919	1764383783	1764383819	ACCEPT	OK
213	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	44066	6	13	919	1764383859	1764383909	ACCEPT	OK
214	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	44066	6	13	919	1764383859	1764383909	ACCEPT	OK
232	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	33882	6	12	866	1764383463	1764383519	ACCEPT	OK
233	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	33882	6	12	866	1764383463	1764383519	ACCEPT	OK
243	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	33078	6	12	866	1764383719	1764383756	ACCEPT	OK
244	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	33078	6	12	866	1764383719	1764383756	ACCEPT	OK
261	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	34222	6	11	815	1764383589	1764383639	ACCEPT	OK
263	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	34222	6	11	815	1764383589	1764383639	ACCEPT	OK
274	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	45144	6	11	812	1764383740	1764383783	ACCEPT	OK
275	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	45144	6	11	812	1764383740	1764383783	ACCEPT	OK
276	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	38652	6	11	815	1764383801	1764383848	ACCEPT	OK
277	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	38652	6	11	815	1764383801	1764383848	ACCEPT	OK
280	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	48926	6	11	813	1764383840	1764383878	ACCEPT	OK
281	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	48926	6	11	813	1764383840	1764383878	ACCEPT	OK
282	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	41204	6	11	815	1764383859	1764383909	ACCEPT	OK
285	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	41204	6	11	815	1764383859	1764383909	ACCEPT	OK
286	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	44068	6	11	815	1764383941	1764383970	ACCEPT	OK
287	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	44068	6	11	815	1764383941	1764383970	ACCEPT	OK
295	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	56336	6	10	761	1764383646	1764383695	ACCEPT	OK
296	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	57274	6	10	760	1764383646	1764383695	ACCEPT	OK
297	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	9200	56336	6	10	761	1764383646	1764383695	ACCEPT	OK
298	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	9200	57274	6	10	760	1764383646	1764383695	ACCEPT	OK

1	version	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
34	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	59300	9200	6	130	182736	1764383801	1764383848	ACCEPT	OK
35	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	59300	9200	6	130	182736	1764383801	1764383848	ACCEPT	OK
58	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	34768	9200	6	68	92723	1764383801	1764383848	ACCEPT	OK
59	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	34768	9200	6	68	92723	1764383801	1764383848	ACCEPT	OK
153	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	45076	9200	6	16	15210	1764383589	1764383639	ACCEPT	OK
154	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	45076	9200	6	16	15210	1764383589	1764383639	ACCEPT	OK
155	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	46854	9200	6	16	15798	1764383859	1764383909	ACCEPT	OK
156	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	46854	9200	6	16	15798	1764383859	1764383909	ACCEPT	OK
173	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	58414	9200	6	15	13615	1764383783	1764383819	ACCEPT	OK
174	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	58414	9200	6	15	13615	1764383783	1764383819	ACCEPT	OK
175	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	44066	9200	6	15	14565	1764383859	1764383909	ACCEPT	OK
176	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	44066	9200	6	15	14565	1764383859	1764383909	ACCEPT	OK
186	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	33882	9200	6	14	12481	1764383463	1764383519	ACCEPT	OK
187	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	33882	9200	6	14	12481	1764383463	1764383519	ACCEPT	OK
188	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	33078	9200	6	14	12942	1764383719	1764383756	ACCEPT	OK
189	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	33078	9200	6	14	12942	1764383719	1764383756	ACCEPT	OK
201	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	34222	9200	6	13	11718	1764383589	1764383639	ACCEPT	OK
202	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	34222	9200	6	13	11718	1764383589	1764383639	ACCEPT	OK
203	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	45144	9200	6	13	10775	1764383740	1764383783	ACCEPT	OK
204	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	45144	9200	6	13	10775	1764383740	1764383783	ACCEPT	OK
207	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	38652	9200	6	13	10726	1764383801	1764383848	ACCEPT	OK
208	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	38652	9200	6	13	10726	1764383801	1764383848	ACCEPT	OK
209	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	48926	9200	6	13	11994	1764383840	1764383878	ACCEPT	OK
210	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	48926	9200	6	13	11994	1764383840	1764383878	ACCEPT	OK
211	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	41204	9200	6	13	11258	1764383859	1764383909	ACCEPT	OK
212	2	905418058769	eni-0aadd41a6546b32ft	172.31.30.62	13.223.19.237	41204	9200	6	13	11258	1764383859	1764383909	ACCEPT	OK
215	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	40140	9200	6	13	11710	1764383941	1764383970	ACCEPT	OK
216	2	905418058769	eni-0aadd41a6546b32ft	13.223.19.237	172.31.30.62	44068	9200	6	13	11849	1764383941	1764383970	ACCEPT	OK

- **Random higher ports with REJECT:**
  The REJECT flows often target many different ports such as 23 (Telnet), 3389 (RDP), 5060 (SIP), 6379 (Redis), and other high-numbered ports. Each of these flows usually has only 1 packet and about 40–60 bytes. This pattern is typical of Internet scanning, where external hosts probe a large range of ports.

1	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
450	905418058769	eni-0aadd41a6546b32fb	79.124.60.6	172.31.30.62	50161	46266	6	4	160	1764289715	1764289745	REJECT	OK
480	905418058769	eni-0aadd41a6546b32fb	79.124.40.118	172.31.30.62	53663	60245	6	3	120	1764289415	1764289444	REJECT	OK
482	905418058769	eni-0aadd41a6546b32fb	79.124.40.118	172.31.30.62	53663	65204	6	3	120	1764289569	1764289603	REJECT	OK
493	905418058769	eni-0aadd41a6546b32fb	78.128.114.110	172.31.30.62	49576	2193	6	3	120	1764383941	1764383970	REJECT	OK
496	905418058769	eni-0aadd41a6546b32fb	122.100.96.175	172.31.30.62	49709	2323	6	2	120	1764288936	1764288970	REJECT	OK
498	905418058769	eni-0aadd41a6546b32fb	79.124.56.230	172.31.30.62	43334	1091	6	2	80	1764289142	1764289183	REJECT	OK
499	905418058769	eni-0aadd41a6546b32fb	79.124.56.230	172.31.30.62	43334	1091	6	2	80	1764289175	1764289212	REJECT	OK
500	905418058769	eni-0aadd41a6546b32fb	175.142.171.187	172.31.30.62	36796	23	6	2	120	1764289175	1764289212	REJECT	OK
506	905418058769	eni-0aadd41a6546b32fb	79.124.56.6	172.31.30.62	52110	57716	6	2	80	1764383522	1764383579	REJECT	OK
509	905418058769	eni-0aadd41a6546b32fb	79.124.56.6	172.31.30.62	52110	57716	6	2	80	1764383559	1764383610	REJECT	OK
512	905418058769	eni-0aadd41a6546b32fb	91.196.152.178	172.31.30.62	17942	62618	6	1	60	1764288718	1764288764	REJECT	OK
514	905418058769	eni-0aadd41a6546b32fb	147.185.133.9	172.31.30.62	52924	9612	6	1	44	1764288718	1764288764	REJECT	OK
520	905418058769	eni-0aadd41a6546b32fb	54.208.185.38	172.31.30.62	51934	4000	6	1	52	1764288718	1764288764	REJECT	OK
521	905418058769	eni-0aadd41a6546b32fb	31.22.104.44	172.31.30.62	6040	12172	6	1	52	1764288718	1764288764	REJECT	OK
522	905418058769	eni-0aadd41a6546b32fb	170.80.48.16	172.31.30.62	5955	5060	17	1	440	1764288718	1764288764	REJECT	OK
533	905418058769	eni-0aadd41a6546b32fb	195.184.76.3	172.31.30.62	47884	554	6	1	60	1764288718	1764288764	REJECT	OK
535	905418058769	eni-0aadd41a6546b32fb	147.185.133.154	172.31.30.62	54763	12096	6	1	44	1764288718	1764288764	REJECT	OK
545	905418058769	eni-0aadd41a6546b32fb	147.185.133.247	172.31.30.62	54533	48391	6	1	44	1764288755	1764288793	REJECT	OK
570	905418058769	eni-0aadd41a6546b32fb	162.216.149.124	172.31.30.62	49940	2137	17	1	78	1764288755	1764288793	REJECT	OK
582	905418058769	eni-0aadd41a6546b32fb	162.216.150.66	172.31.30.62	55232	8074	6	1	44	1764288755	1764288793	REJECT	OK
586	905418058769	eni-0aadd41a6546b32fb	65.108.128.42	172.31.30.62	22	50798	6	1	44	1764288779	1764288821	REJECT	OK
587	905418058769	eni-0aadd41a6546b32fb	198.235.24.161	172.31.30.62	50186	1434	17	1	29	1764288779	1764288821	REJECT	OK
588	905418058769	eni-0aadd41a6546b32fb	35.203.210.159	172.31.30.62	50430	47830	6	1	44	1764288779	1764288821	REJECT	OK
590	905418058769	eni-0aadd41a6546b32fb	109.205.213.43	172.31.30.62	52035	4332	6	1	40	1764288779	1764288821	REJECT	OK
592	905418058769	eni-0aadd41a6546b32fb	198.235.24.28	172.31.30.62	54197	20257	6	1	44	1764288779	1764288821	REJECT	OK
595	905418058769	eni-0aadd41a6546b32fb	147.185.132.163	172.31.30.62	56024	10092	6	1	44	1764288779	1764288821	REJECT	OK
598	905418058769	eni-0aadd41a6546b32fb	162.216.149.208	172.31.30.62	50668	8306	6	1	44	1764288816	1764288855	REJECT	OK
599	905418058769	eni-0aadd41a6546b32fb	124.198.132.121	172.31.30.62	46255	81	6	1	40	1764288816	1764288855	REJECT	OK
601	905418058769	eni-0aadd41a6546b32fb	35.203.210.58	172.31.30.62	53737	9692	6	1	44	1764288816	1764288855	REJECT	OK

Overall, the only port that clearly shows strong, focused high-volume behaviour is **1883**, which is exactly where we aimed our simulated MQTT attack.

Indicators of DDoS, Scanning, or Brute-Force Attempts

- **DDoS-style or flood behaviour**
  - The flows between 69.158.246.215 → 172.31.30.62 on port 1883 have:
    very high packet counts (around 900–990 packets per flow)
    very high byte counts (up to ~750 KB per flow)
  - These flows occur repeatedly in a short time window.
  - This matches a DDoS-style or flood attack against the MQTT broker, which is what we tried to simulate from the client side.

1	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
3	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	990	745701	1764383411	1764383450	ACCEPT	OK
4	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	962	741782	1764383463	1764383519	ACCEPT	OK
6	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	954	743840	1764383646	1764383695	ACCEPT	OK
8	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	954	742565	1764383522	1764383579	ACCEPT	OK
11	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	954	741294	1764383589	1764383639	ACCEPT	OK
12	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	949	742197	1764383062	1764383099	ACCEPT	OK
15	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	934	745412	1764383339	1764383393	ACCEPT	OK
17	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	928	741241	1764383901	1764383931	ACCEPT	OK
18	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	906	751370	1764383783	1764383819	ACCEPT	OK
20	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	900	733658	1764383719	1764383756	ACCEPT	OK
22	905418058769	eni-0aadd41a6546b32fb	69.158.246.215	172.31.30.62	33610	1883	6	882	738841	1764383840	1764383878	ACCEPT	OK

- **Scanning behaviour**
    - The **REJECT** flows come from many different external IPs, each with only one or a few packets.
    - They target many different ports, including Telnet (23), RDP (3389), SIP (5060), Redis (6379), and various high ports.
    - This looks like normal background **port scanning** from the Internet. The traffic is low volume and is rejected by the network, so it does not impact the MQTT service directly.

1	account-id	interface-id	srcaddr	dstaddr	srcport	dstport	protocol	packets	bytes	start	end	action	log-status
500	905418058769	eni-0aadd41a6546b32fb	175.142.171.187	172.31.30.62	36796	23	6	2	120	1764289175	1764289212	REJECT	OK
522	905418058769	eni-0aadd41a6546b32fb	170.80.48.16	172.31.30.62	5955	5060	17	1	440	1764288718	1764288764	REJECT	OK
642	905418058769	eni-0aadd41a6546b32fb	141.98.11.172	172.31.30.62	34740	23	6	1	40	1764288875	1764288912	REJECT	OK
667	905418058769	eni-0aadd41a6546b32fb	71.6.199.65	172.31.30.62	54332	6379	6	1	52	1764288936	1764288970	REJECT	OK
719	905418058769	eni-0aadd41a6546b32fb	60.209.38.156	172.31.30.62	60502	6379	6	1	60	1764289087	1764289123	REJECT	OK
1114	905418058769	eni-0aadd41a6546b32fb	125.82.243.118	172.31.30.62	33624	3389	6	1	44	1764355385	1764355437	REJECT	OK
1179	905418058769	eni-0aadd41a6546b32fb	23.95.132.51	172.31.30.62	53361	3389	6	1	40	1764383463	1764383519	REJECT	OK
1193	905418058769	eni-0aadd41a6546b32fb	91.232.238.112	172.31.30.62	5072	5060	17	1	434	1764383501	1764383544	REJECT	OK

Security Controls and Their Effectiveness

Across the project, several security controls worked together:

- **AWS IoT Core with certificates** protected the main sensor data path. The Raspberry Pi had to use the correct certificate and private key to send MQTT over TLS, which stopped unknown devices from sending data as our Thing.
- **Suricata on EC2** detected the simulated MQTT attacks. The custom rules for high traffic and large packets on port 1883 triggered alerts when we ran the DDoS-style and malformed payload traffic.
- **Filebeat, Elasticsearch, and Kibana** made these alerts easy to search and visualize. Instead of reading raw log files, we could quickly filter by source IP, port, and alert type.
- **VPC Flow Logs to S3** added a second view at the network level. Even without seeing packet contents, we could still spot the attack by looking at flows with very high packets and bytes on port 1883, and separate them from normal and rejected traffic.

Taken together, these controls showed that the environment can catch and explain unusual MQTT traffic, not only at the application layer but also at the VPC/network layer.

Challenges and Limitations

While the system worked, we faced some practical challenges:

- **Resource limits on EC2:** Running Elasticsearch, Kibana, Suricata, Mosquitto, and Filebeat on a t2.medium instance required heap tuning and careful checks so the instance did not run out of memory.
- **Configuration complexity:** Getting all configs aligned (Suricata interface and rules, Filebeat input and output, Elasticsearch security settings, Kibana credentials) took time and small mistakes could stop the pipeline until fixed.
- **Log volume and analysis:** The VPC Flow Logs produced a lot of rows. We had to rely on filtering and sorting in Excel to find the important flows, which works for a lab but would be hard to scale in a large production network.
- **Scope of detection:** Our Suricata rules focused mainly on high-rate and large MQTT packets. Other types of attack would need more rules and tuning.

Conclusion

This project combined an IoT device, cloud services, and security monitoring into one end-to-end setup. The MCP9808 sensor and Raspberry Pi sent continuous temperature data to AWS IoT Core, while mirrored MQTT traffic went to an EC2 monitoring server with Mosquitto, Suricata, Filebeat, Elasticsearch, and Kibana. VPC Flow Logs added an extra layer of network visibility through S3.

When we launched the simulated attacks, all layers reflected the same story:

- Suricata and Kibana showed alerts for abnormal MQTT traffic to the EC2 broker.
- VPC Flow Logs showed high-volume flows on port 1883 between the EC2 instance and the attack source, clearly different from normal SSH, Kibana, and Elasticsearch traffic.

Overall, the project demonstrates how an IoT system can be monitored and investigated using a mix of device-level security, IDS alerts, log forwarding, dashboards, and VPC Flow Logs, and how simulated attacks can be traced across these different data sources.