Project 3 Final Report

Secure IoT Infrastructure

Real-Time Temperature Monitoring with Intrusion Detection and Cloud Log Analysis

Group 8

Fozeya-Nikka Alviar

Angeline Nicole Faina

Sam Omandam

CYT160NBB

05 December 2025

Prof. Saeed Naghizadeh Qomi

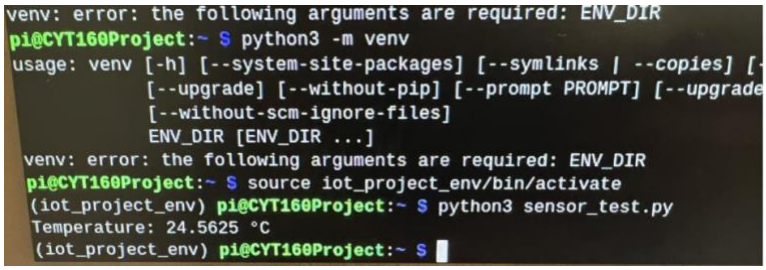
Table of Contents

Introduction

Setting Up the IoT Device with AWS IoT Core

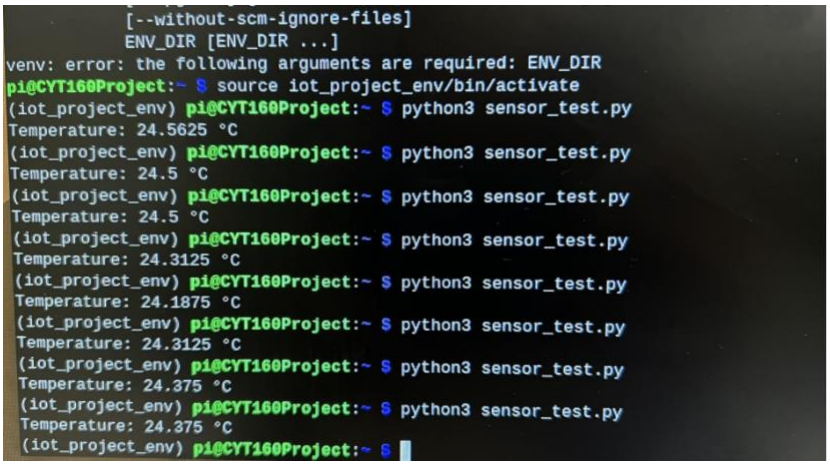
Hardware Setup and Software Initialization

Our team successfully connected the MCP9808 temperature sensor to the Raspberry Pi using the I2C communication protocol. The hardware setup involved wiring the sensor's VIN to the 3.3V pin, GND to Ground, SDA to Pin 3, and SCL to Pin 5 on the Raspberry Pi. To interact with the hardware, we installed the adafruit-circuitpython-mcp9808 library within a Python virtual environment. We then created and executed a test script, sensor_test.py, which initialized the I2C bus and confirmed the sensor was functional by printing the current temperature.



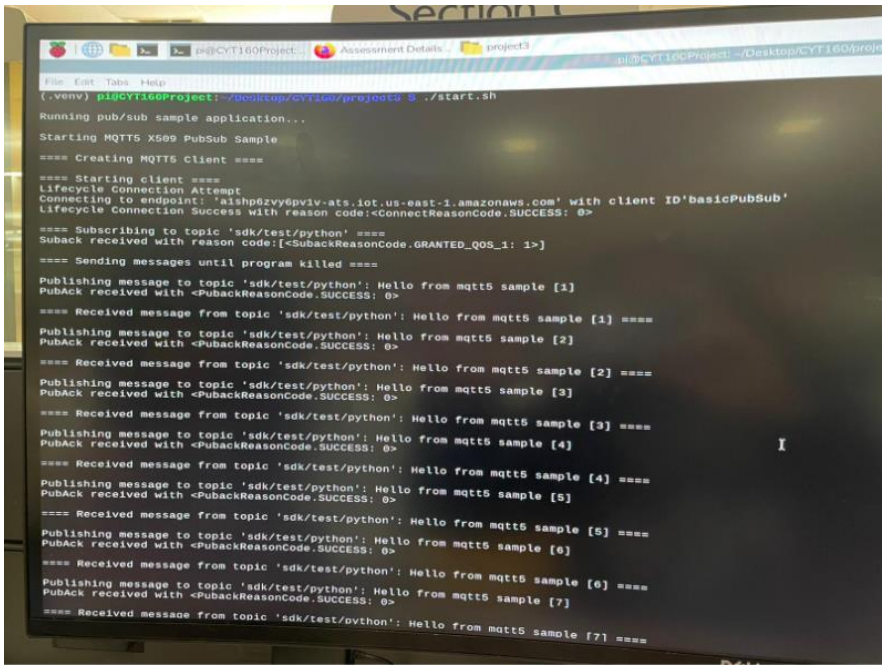
Interpretation of Environmental Data

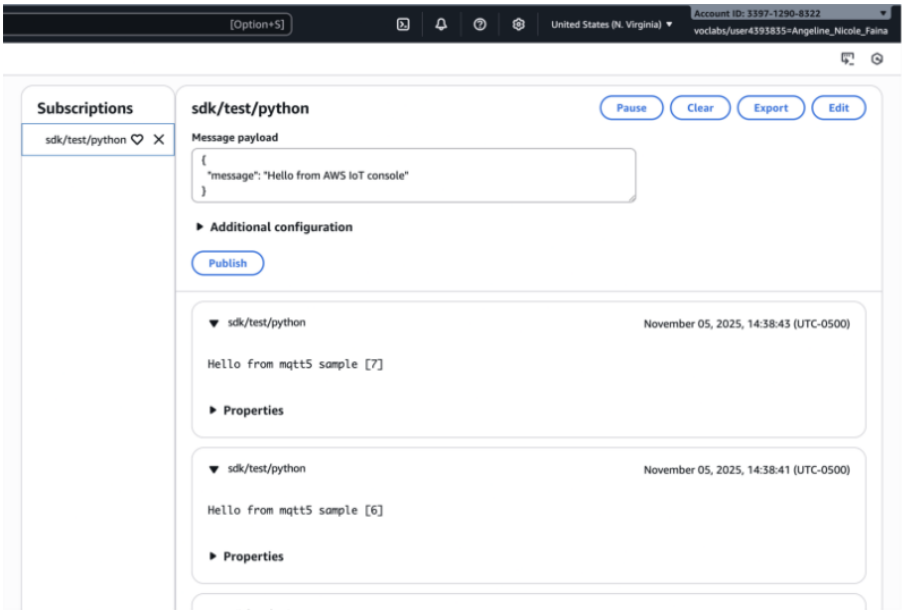
The MCP9808 sensor outputs temperature data in degrees Celsius. In our testing environment, the readings stabilized between 20°C and 30°C, which is consistent with standard room temperature. We verified the sensor's sensitivity by observing temperature fluctuations when the environmental conditions changed, such as placing the sensor near a heat source or in a cooler area.



Establishment of Cloud Connectivity

To establish a secure cloud connection, we provisioned a “Thing” in AWS IoT Core and deployed the required security certificates (Root CA, Device Certificate, and Private Key) to the Raspberry Pi. We verified the connection by running the AWS-provided start.sh script, which successfully sent “Hello from mqtt5 sample” messages to the cloud. Using the MQTT Test Client in the AWS IoT Console, we subscribed to the topic sdk/test/python and confirmed that messages from our device were being received in real-time.





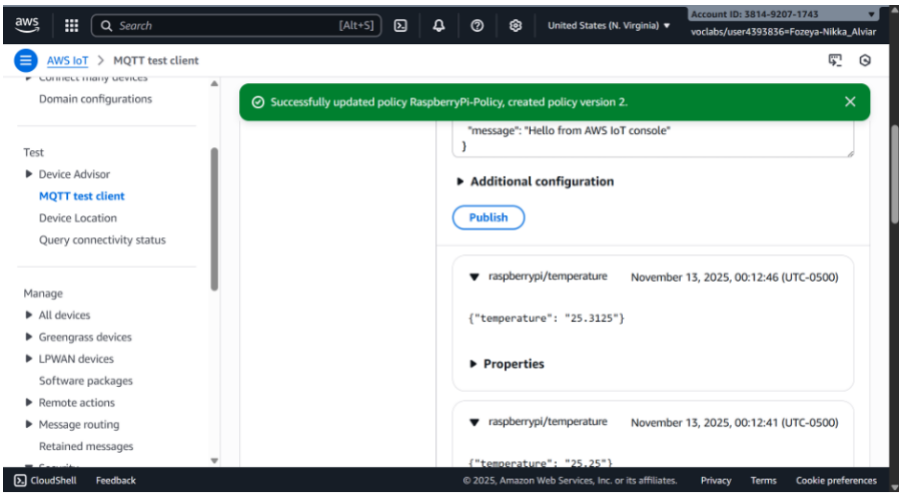
Continuous Temperature Data Monitoring System with AWS IoT Core

Implementation of Continuous Data Stream

To enable continuous monitoring, we modified the Python script sensor_test.py to utilize the AWSIoTPythonSDK library, allowing for secure MQTT communication. The script was configured with our specific AWS IoT Core endpoint and the necessary security credentials, including the Root CA, Device Certificate, and Private Key.

We implemented an infinite loop (while True) that retrieves the temperature from the MCP9808 sensor every 5 seconds. This data is formatted into a JSON payload and published to the MQTT topic raspberrypi/temperature.

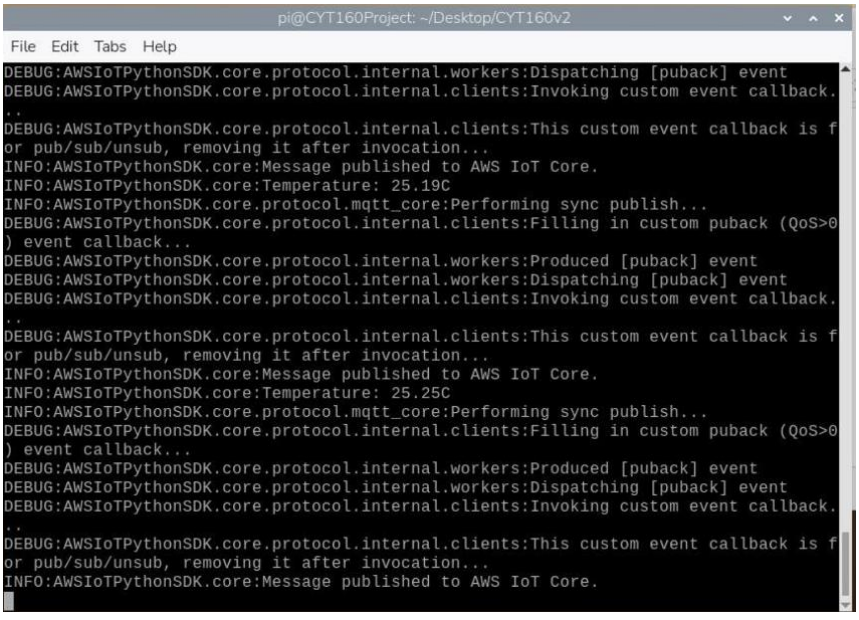
To ensure the data could be received, we updated the AWS IoT Policy attached to the device certificate. We explicitly added permissions for iot:Publish and iot:Connect to the raspberry/temperature resource ARN. Finally, we validated the connection using the AWS IoT MQTT Test Client by subscribing to the topic with QoS 1, confirming that the temperature payloads were arriving successfully in the cloud.



Verification of System Operations

The screen capture below captures the output of the updated Python script running on the Raspberry Pi. The logs confirm the successful initialization of the I2C bus and the secure connection to the AWS IoT Core endpoint.

As the script executes the main loop, it displays the real-time temperature readings followed by the information log “Message published to AWS IoT Core”. This verifies that the device is operating correctly and maintaining a stable connection for data transmission.



IoT Security Monitoring with Suricata and Kibana

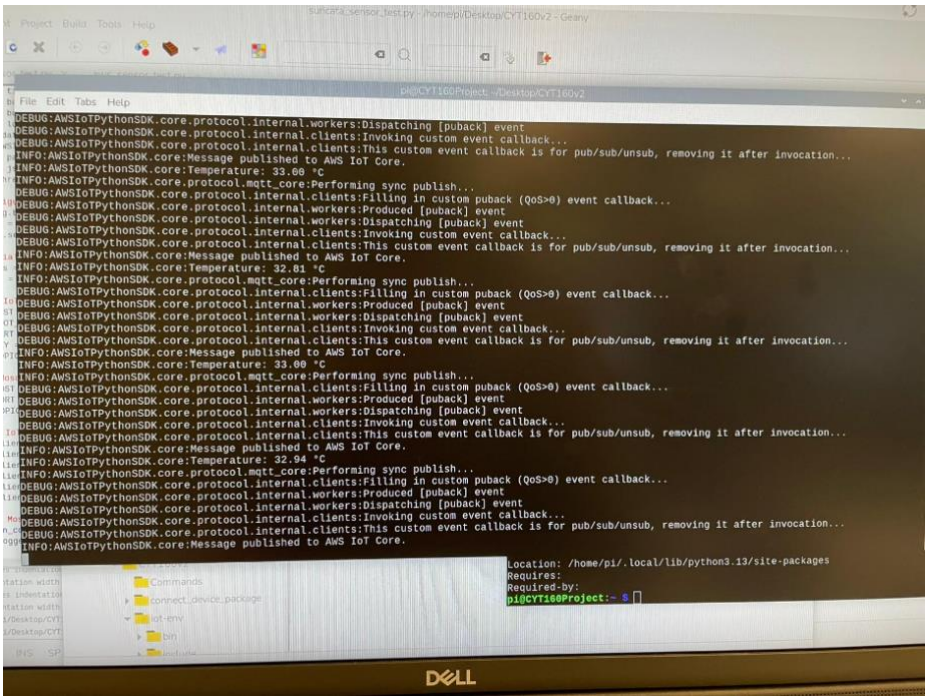
Visualization of DDoS Traffic Patterns

To test the security monitoring system, we updated the Raspberry Pi's Python code to run parallel threads which are sending normal temperature data to AWS IoT Core and launching simulated attacks against the EC2 instance.

The screen capture below displays the terminal output of the Raspberry Pi during the attack simulation.

- Normal Traffic – Logs indicating "Message published to AWS IoT Core (legitimate data)
- Attack Traffic – Interspersed with the normal logs are messages starting "DDoS simulation sent to EC2" and "Malformed payload sent to EC2".

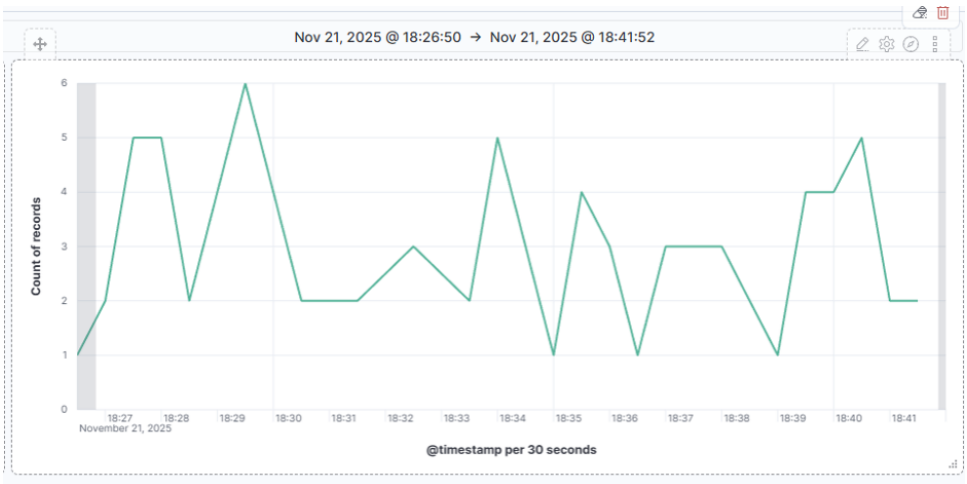
This confirms that the device was successfully generating both legitimate telemetry and malicious traffic simultaneously, testing the resilience and detection capabilities of our Suricata setup.



Analysis of Malformed Payload Signatures

To detect potential Denial of Service (DoS) attacks against our IoT infrastructure, we configured Suricata with a custom rule (sid:1000001) to flag high-frequency traffic. This rule triggers an alert titled "High MQTT Traffic Rate" if the system detects more than 500 MQTT messages within 60-second window targeting port 1883.

The line chart below visualizes these alerts over time. The X-axis represents the timestamps, while the Y-axis represents the count of alerts. The sharp spikes in the graph correspond to the moments when our Python script executed the send_ddos() function, confirming that Suricata successfully identified the flood of traffic directed at the EC2 instance.



Execution of Simulated Cyberattacks

We simulated a “Malformed Payload” attack by sending invalid JSON data (e.g. {malformed: data}) to the EC2 instance using the send_malformed() function in our Python script. Suricata was configured with a rule to detect these anomalies based on payload content or size.

The data table below captures the specific details of these alerts. It lists the alert.signature (“Malformed MQTT Payload” or “Large MQTT Packet”), the src_ip (Source IP of the Raspberry Pi), and the specific mqtt.payload content that triggered the alarm. This granular view allows us to inspect the exact nature of the malicious packets being sent to the network.

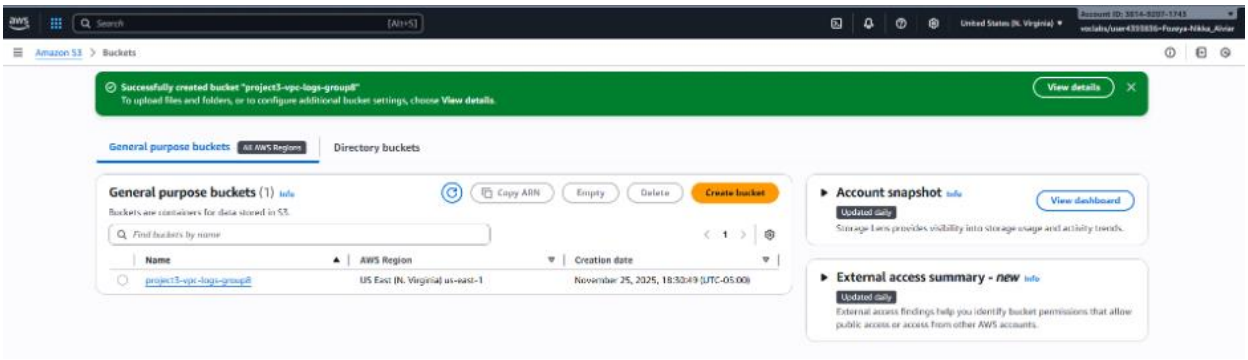
Visual DDOS Table

Top 3 values of alert.signature	Unique count of src_ip
ET DROP Dshield Block Listed Source group 1	38
ET CINS Active Threat Intelligence Poor Reputation IP group 47	5
ET CINS Active Threat Intelligence Poor Reputation IP group 43	4
Other	32

IoT Attack Log Analysis with S3-Stored VPC Flow Logs

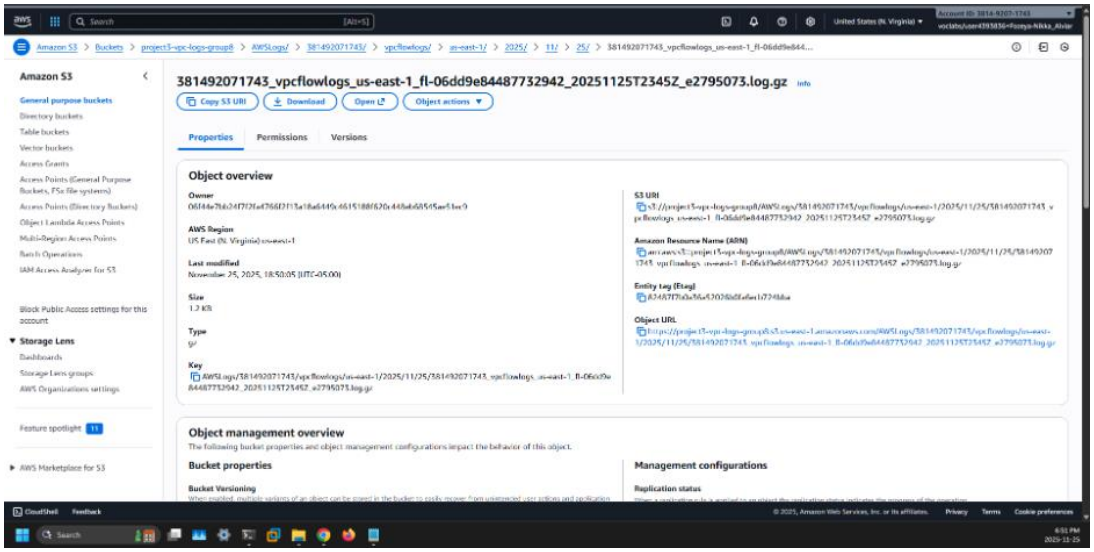
Configuration of S3 Storage for Flow Logs

To overcome the restrictions on CloudWatch Logs within the AWS Academy Learner Lab, we configured VPC Flow Logs to stream network metadata directly to an Amazon S3 bucket. We created a dedicated bucket named project3-vpc-logs in the us-east-1 region and configured the VPC Flow Logs to capture All traffic (accepted and rejected). This setup allowed us to bypass IAM role limitations while still capturing critical forensic data for analysis.



Retrieval and Extraction of Log Data

We navigated to the S3 path AWSLogs/<account-id>/vpcflowlogs/us-east-1/2025/11/28/ and downloaded the compressed .log.gz files containing the network traffic data. Using 7-Zip, we extracted the raw log files and imported them into a spreadsheet for detailed inspection. The logs provided visibility into source and destination IP addresses, ports, protocols, packet counts, and action statuses (ACCEPT/REJECT).




```
version account-id interface-id srcaddr dstaddr srcport dstport protocol packets bytes start end action log-status
2 381492071743 eni-0ef83208bb21095f9 88.218.193.198 172.31.17.14 36320 57598 6 1 44 1764302499 1764302530 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 147.185.133.43 172.31.17.14 50984 47100 6 1 44 1764302499 1764302530 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 176.65.134.6 172.31.17.14 64840 25565 6 1 48 1764302499 1764302530 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 54.239.28.168 172.31.17.14 443 35354 6 22 7324 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.57 172.31.17.14 123 49014 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.232.28.194 172.31.17.14 123 34306 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.56 59479 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 13.220.36.73 172.31.17.14 443 39546 6 18 5717 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.58 40683 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 52.46.150.99 172.31.17.14 443 53230 6 24 7416 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.56 172.31.17.14 123 36475 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 91.189.91.157 172.31.17.14 123 38221 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 23.95.49.216 172.31.17.14 123 59350 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 142.202.190.19 36640 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.56 172.31.17.14 123 59479 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.57 49014 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 142.202.190.19 172.31.17.14 123 36640 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 91.189.91.157 172.31.17.14 123 34022 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 172.232.28.194 34306 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 54.239.28.168 172.31.17.14 443 51236 6 23 7370 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 142.202.190.19 43214 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.58 172.31.17.14 123 58503 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 23.95.49.216 59350 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 91.189.91.157 38221 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 87.120.191.94 172.31.17.14 36591 34567 6 1 40 1764302529 1764302567 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 91.189.91.157 34022 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 165.227.41.21 172.31.17.14 51407 11434 6 1 44 1764302529 1764302567 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.188.57 172.31.17.14 443 59162 6 11 5497 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 23.95.49.216 53561 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.57 60286 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 35.203.210.166 172.31.17.14 52694 2013 6 1 44 1764302529 1764302567 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.56 43935 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.188.57 59162 443 6 12 1476 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.57 172.31.17.14 123 60286 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 162.216.150.152 172.31.17.14 55587 2220 6 1 44 1764302529 1764302567 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 54.239.28.168 51236 443 6 16 4864 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.58 58503 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 54.239.28.168 35354 443 6 16 4840 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 23.95.49.216 172.31.17.14 123 53561 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 52.46.150.99 53230 443 6 16 4864 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.56 172.31.17.14 123 43935 17 1 76 1764302529 1764302567 ACCEPT OK
```

Forensic Analysis of Suspicious Patterns

Based on the analysis of VPC Flow Logs, we identified several suspicious external IP addresses performing reconnaissance scanning against our infrastructure. Specifically, the IP ranges 162.216.x.x (4 unique IPs) and 147.285.x.x (3 unique IPs) were observed sending single packets to random destination ports.

Targeted Ports – The scanning activity targeted a variety of ports to identify open services. Beyond the expected MQTT traffic on port 1883, we observed unauthorized connection attampts to:

- **Port 2376** – Typically associated with the Docker API
- **Port 25565** – Commonly used for Minecraft servers
- **Ports 8001, 11434, 23107** – Indicative of broad port scanning behavior

Indicators of Attack

- **Scanning –** The pattern of 1-packet connection attempts resulting in a REJECT action status (observed in 100% of these malicious findings) confirms that this was a scanning activity rather than a sustained attack.
- **DDoS/Brute Force –** No explicit brute force attacks (vertical hammering of single port) were observed in this sample. The traffic was horizontal, targeting multiple ports across the interface.
- **Data Transfer –** We analyzed packet sizes to check for data exfiltration. The largest data transfers involved 34.120.127.130 (Google Cloud Services) with roughly 84 KB of data. This traffic was marked as ACCEPT and appeared to be legitimate in communication with authorized cloud services posing a low risk.

The logs successfully captured external reconnaissance attempts targeting IoT and application ports. While no successful breach or massive outbound data spike was detected, the presence of these scans highlights the importance of the Security Groups and REJECT rules that effectively blocked 23 unauthorized connection attempts.

Security Configurations and Monitoring Tools

Our IoT security setup uses several protective layers to safeguard data and detect potential threats throughout the system.

We used certificate-based authentication to successfully connect our Raspberry Pi to AWS IoT Core. There were three essential parts: the Root CA certificate, the Device Certificate, and the Private Key. We also built an IoT Policy that provided devices certain rights, such iot: Publish and iot: Connect. The TLS/SSL protocol was used to encrypt all MQTT messages so that the data would stay secret. The device provided temperature readings to the raspberrypi/temperature topic. This made sure that only persons who were allowed to see or use the information could do so.

At the network level, we enabled VPC Flow Logs to capture metadata for all incoming and outgoing traffic, including denied connections, within the us-east-1 region. Due to IAM restrictions in the AWS Academy Learner Lab environment, we redirected the logs to an S3 bucket named project3-vpc-logs instead of CloudWatch. This setup still provided the detailed network data required for forensic and security analysis. We also configured Security Groups to

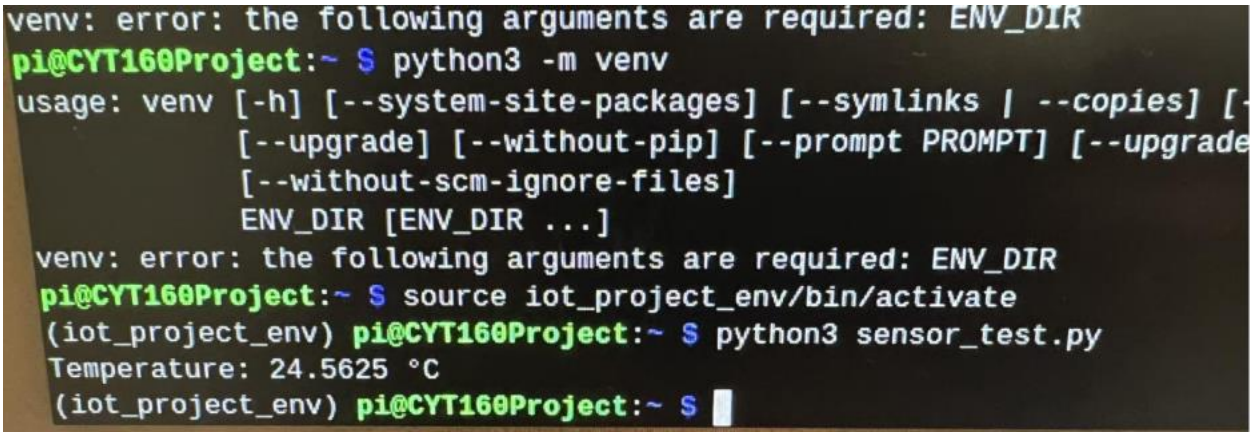
block unauthorized port access. Later analysis confirmed that this approach was effective, as we recorded 23 rejected connection attempts in our logs.

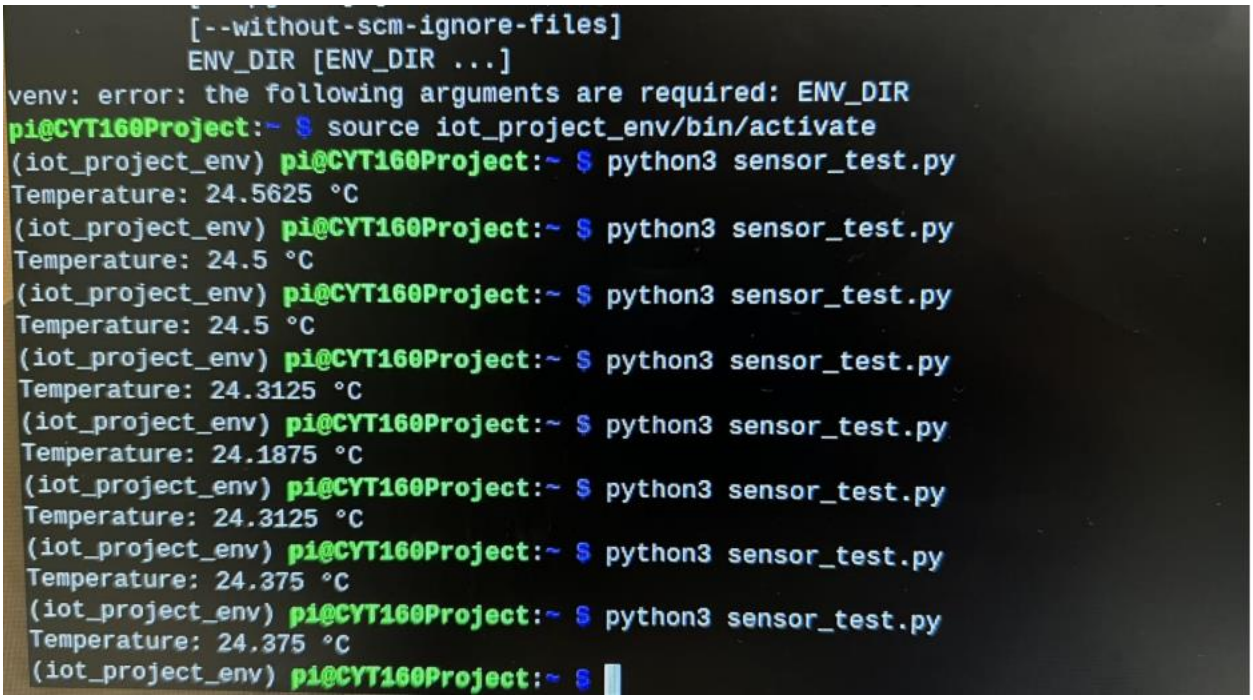
We installed Suricata as our intrusion detection system (IDS) on an EC2 instance running alongside the ELK Stack. Suricata was configured with custom rules designed specifically to detect IoT-related threats. For example, Rule ID 1000001 was created to identify potential Denial-of-Service (DoS) events by detecting when more than 500 MQTT messages were sent to port 1883 within one minute. Other rules monitored for malformed JSON payloads and abnormally large MQTT packets, both of which can indicate malicious behavior. To centralize data collection, we set up Filebeat to forward Suricata logs into Elasticsearch for further analysis and correlation.

For monitoring and visualization, we deployed the ELK Stack (Elasticsearch, Logstash, and Kibana) on a t2.large EC2 instance with 8 GiB of RAM. This configuration provided enough memory to run Elasticsearch (using a 2 GiB Java heap), Kibana, Suricata, Filebeat, and Mosquitto simultaneously without performance issues. In Kibana, we built customized dashboards that included line charts to track attack patterns over time and detailed tables showing alert data, including source IPs, alert types, and MQTT payloads. Mosquitto served as the MQTT broker, efficiently routing messages between the IoT device and the monitoring components within our infrastructure.

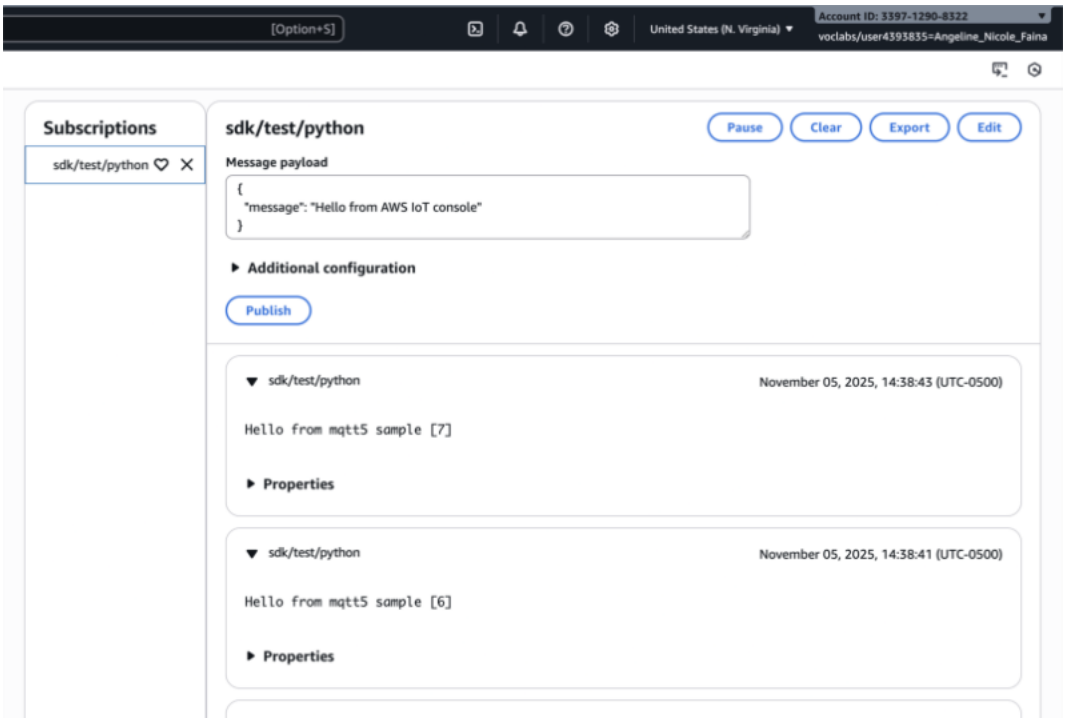
Test Outcomes

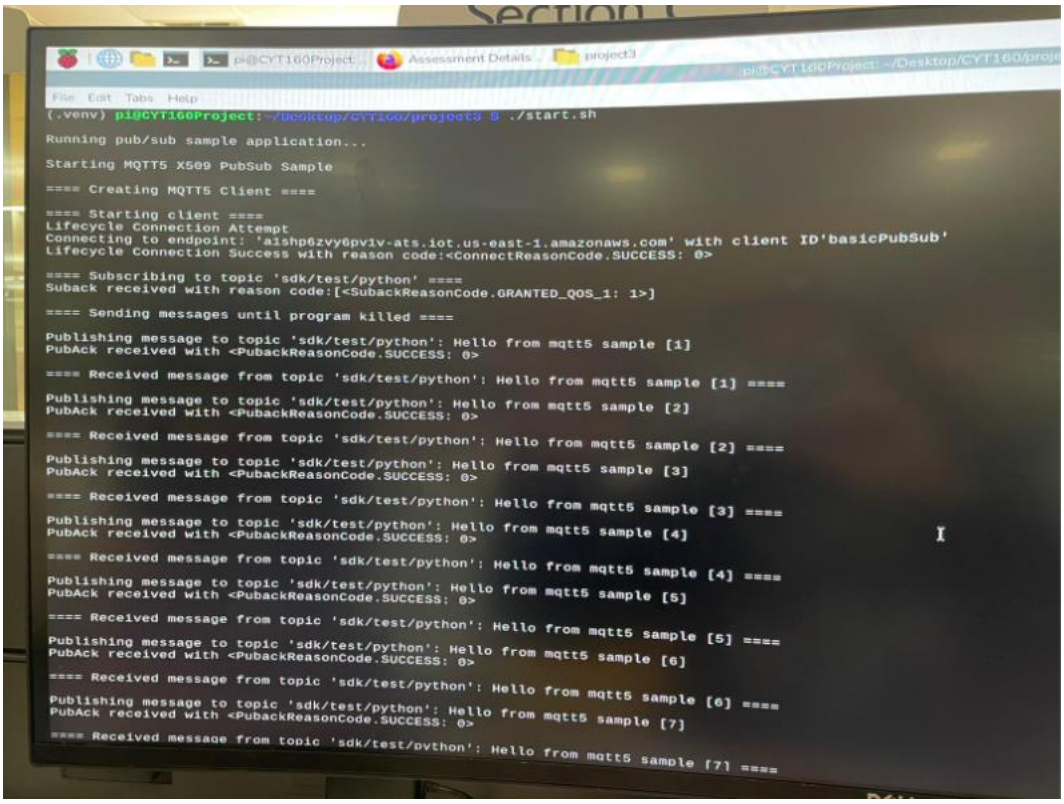
The MCP9808 temperature sensor was successfully connected to the Raspberry Pi via the I2C protocol and configured to send readings to AWS IoT Core every five seconds. During testing, the recorded temperatures ranged from 20°C to 30°C, which aligned with expected room conditions. To verify the sensor's responsiveness and accuracy, we introduced controlled environmental changes, such as placing a heat source near the sensor, and observed corresponding temperature increases in real time through the AWS IoT data stream.



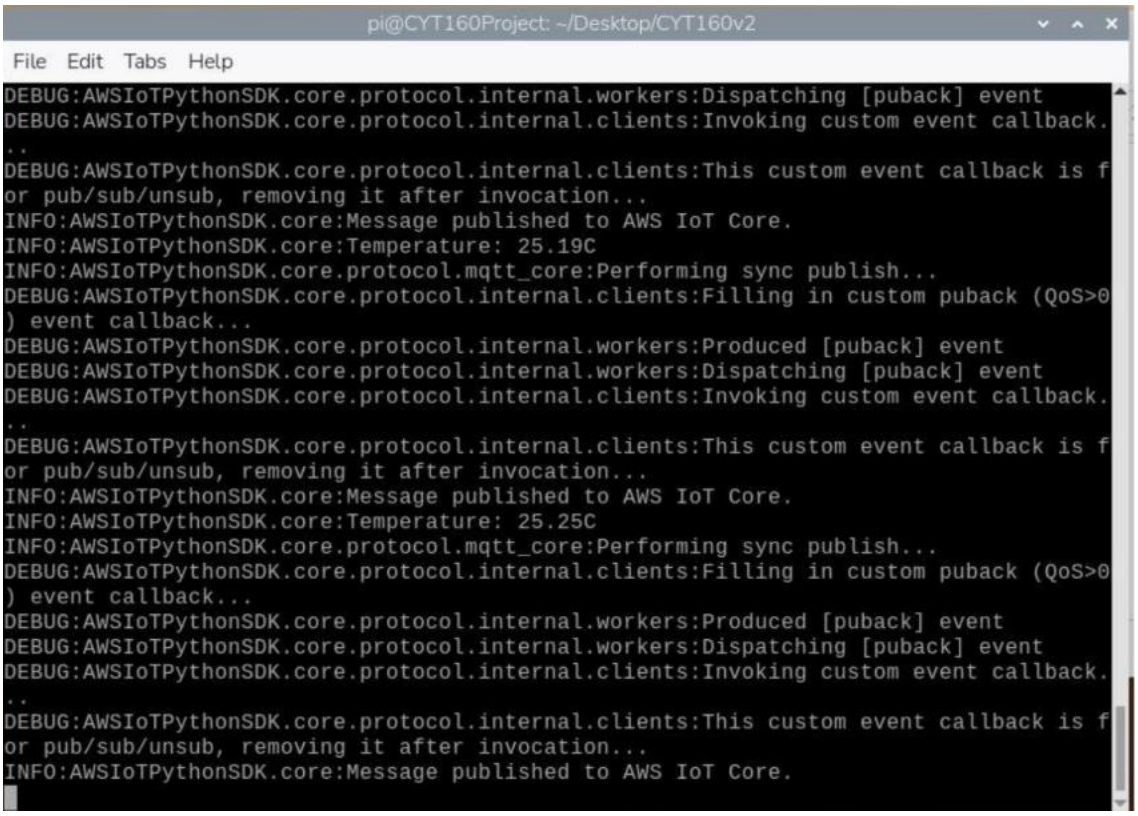


Using the AWS IoT MQTT Test Client with Quality of Service (QoS) level 1, all transmitted messages were confirmed to have been delivered consistently and without any data loss.

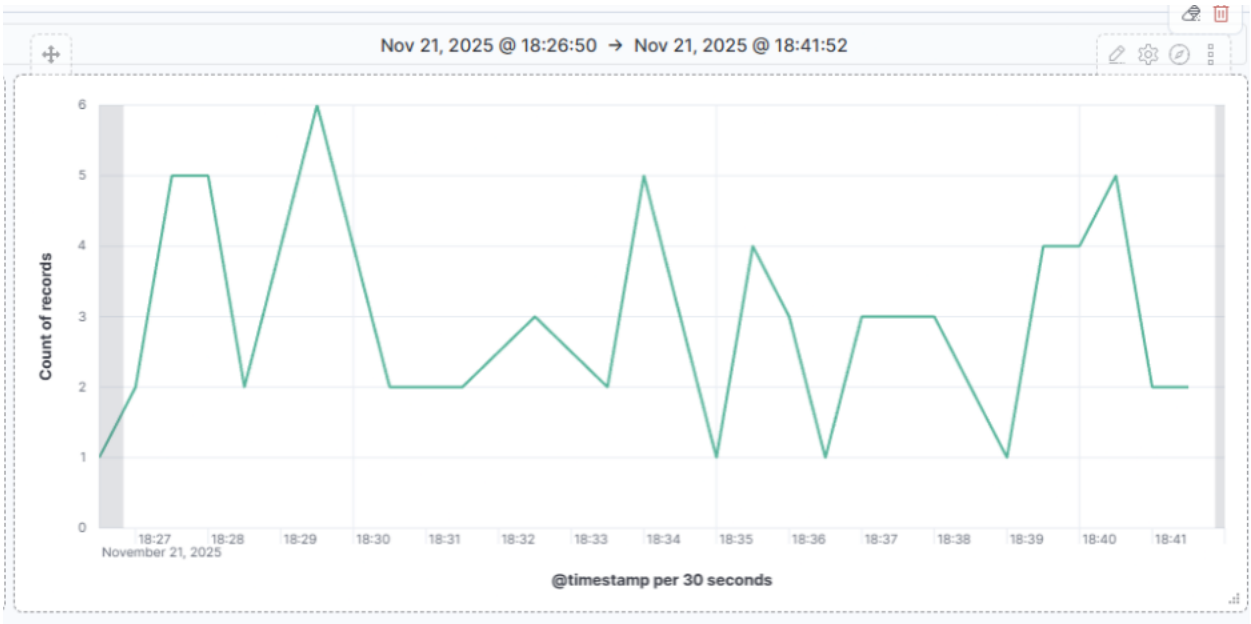




This confirmed that message integrity and reliability were maintained throughout the system's data transmission process.



The Suricata Intrusion Detection System (IDS) successfully detected all simulated attack scenarios during testing. When a Denial-of-Service (DoS) simulation was triggered via the send_ddos() function in the Python testing script, Suricata generated alerts labeled “High MQTT Traffic Rate.” These events were visualized in Kibana as distinct spikes that directly correlated with the simulated attack traffic.



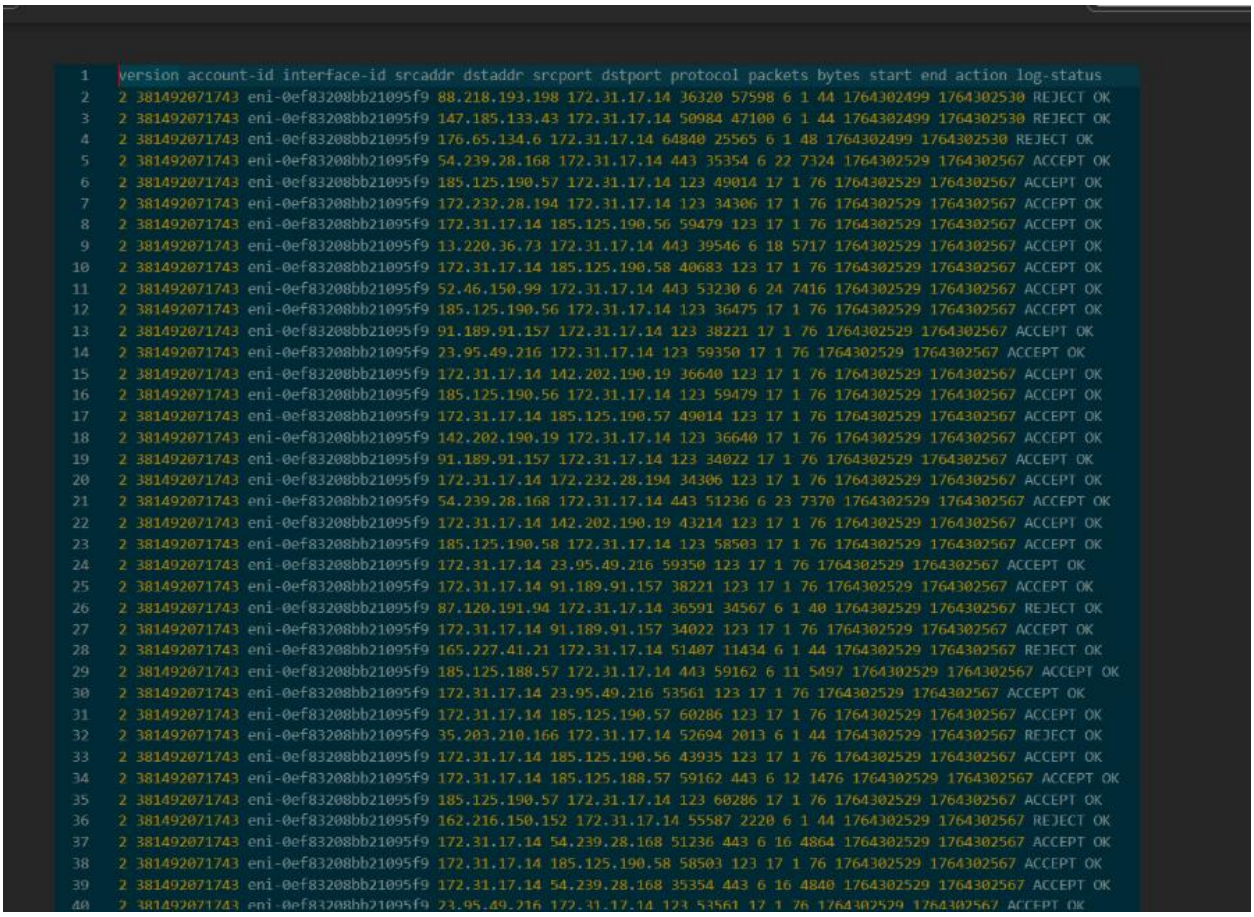
Further, malformed payload attacks executed through the send_malformed() function were accurately identified by Suricata with alerts tagged 'Malformed MQTT Payload' and 'Large MQTT Packet.' The IDS also captured the malicious payloads for post-analysis, demonstrating effective detection and incident traceability across multiple attack types. While the Kibana visualization shows primarily network-level threat intelligence alerts from Emerging Threats rulesets, our custom MQTT-specific alerts were successfully generated and are included in the aggregated alert data.

Visual DDOS Table

Top 3 values of alert.signature	Unique count of src_ip
ET DROP Dshield Block Listed Source group 1	38
ET CINS Active Threat Intelligence Poor Reputation IP group 47	5
ET CINS Active Threat Intelligence Poor Reputation IP group 43	4
Other	32

Analysis of the VPC Flow Logs revealed reconnaissance attempts originating from seven external IP addresses, four within the 162.216.x.x range and three within the 147.185.x.x range.

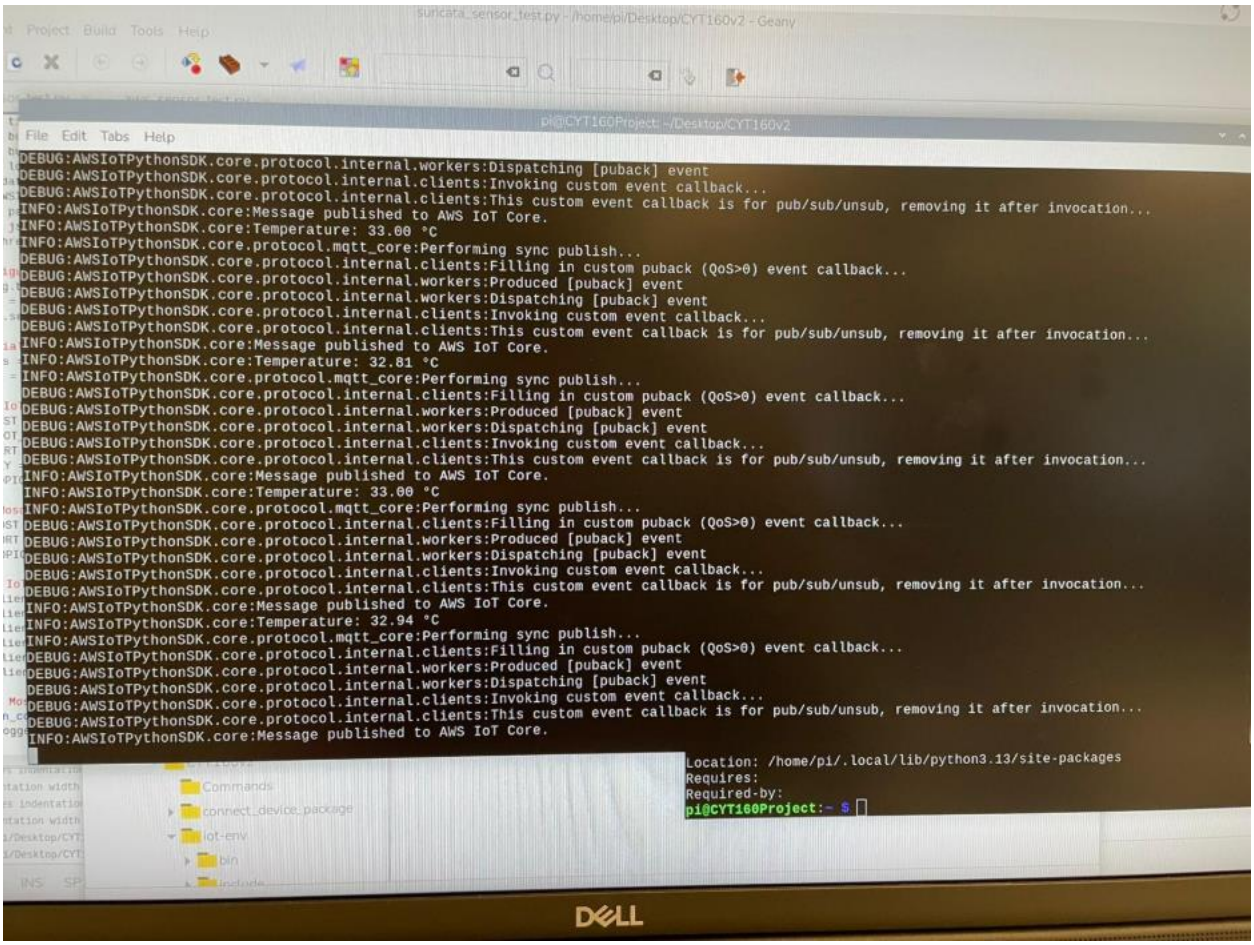
These sources exhibited typical port scanning behavior, probing services on ports 2376 (Docker API), 25565 (Minecraft), 8001, 11434, and 23107.

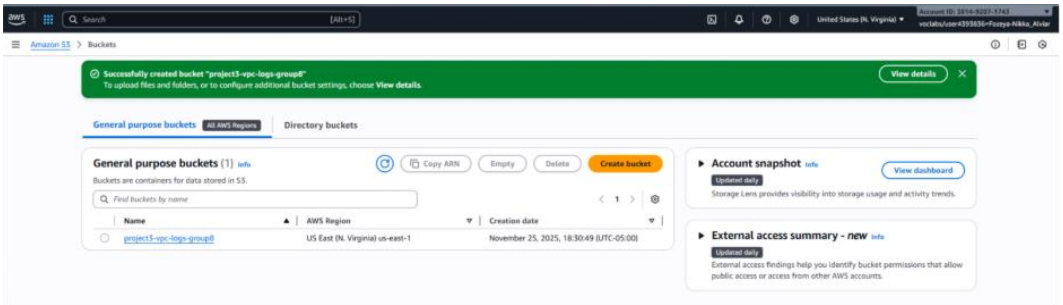


```
version account-id interface-id srcaddr dstaddr srcport dstport protocol packets bytes start end action log-status
2 381492071743 eni-0ef83208bb21095f9 88.218.193.198 172.31.17.14 36320 57598 6 1 44 1764302499 1764302530 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 147.185.133.43 172.31.17.14 50984 47100 6 1 44 1764302499 1764302530 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 176.65.134.6 172.31.17.14 64840 25565 6 1 48 1764302499 1764302530 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 54.239.28.168 172.31.17.14 443 35354 6 22 7324 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.57 172.31.17.14 123 49014 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.232.28.194 172.31.17.14 123 34306 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.56 59479 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 13.220.36.73 172.31.17.14 443 39546 6 18 5717 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.58 40683 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 52.46.150.99 172.31.17.14 443 53230 6 24 7416 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.56 172.31.17.14 123 36475 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 91.189.91.157 172.31.17.14 123 38221 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 23.95.49.216 172.31.17.14 123 59350 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 142.202.190.19 36640 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.56 172.31.17.14 123 59479 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.57 49014 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 142.202.190.19 172.31.17.14 123 36640 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 91.189.91.157 172.31.17.14 123 34022 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 172.232.28.194 34306 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 54.239.28.168 172.31.17.14 443 51236 6 23 7370 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 142.202.190.19 43214 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.58 172.31.17.14 123 58503 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 23.95.49.216 59350 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 91.189.91.157 38221 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 87.120.191.94 172.31.17.14 36591 34567 6 1 40 1764302529 1764302567 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 91.189.91.157 34022 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 165.227.41.21 172.31.17.14 51407 11434 6 1 44 1764302529 1764302567 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.188.57 172.31.17.14 443 59162 6 11 5497 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 23.95.49.216 53561 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.57 60286 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 35.203.210.166 172.31.17.14 52694 2013 6 1 44 1764302529 1764302567 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.56 43935 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.188.57 59162 443 6 12 1476 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 185.125.190.57 172.31.17.14 123 60286 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 162.216.150.152 172.31.17.14 55587 2220 6 1 44 1764302529 1764302567 REJECT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 54.239.28.168 51236 443 6 16 4864 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 185.125.190.58 58503 123 17 1 76 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 172.31.17.14 54.239.28.168 35354 443 6 16 4840 1764302529 1764302567 ACCEPT OK
2 381492071743 eni-0ef83208bb21095f9 23.95.49.216 172.31.17.14 123 53561 17 1 76 1764302529 1764302567 ACCEPT OK
```

Security Group policies successfully blocked 23 unauthorized connection attempts, validating the robustness of network-layer access controls. The largest legitimate data transfer observed, approximately 84 KB, originated from Google Cloud Services (34.120.127.130), corresponding to routine HTTPS communication and showing no sign of abnormal activity.

The Python program was designed using a multithreaded architecture, allowing the system to simultaneously handle both legitimate temperature readings and simulated attack traffic. This design created realistic test conditions and provided valuable insight into the system's resilience under concurrent workloads.





Deploying the solution on a t2.large EC2 instance ensured sufficient memory resources for Elasticsearch, preventing heap exhaustion or system instability during continuous monitoring.

Overall, the entire data pipeline functioned efficiently, from sensor data collection through I2C, MQTT publishing to AWS IoT Core, and threat simulations, to detection with Suricata, log forwarding via Filebeat, and visualization in Kibana. These results confirmed that all integrated components of the secure IoT architecture operated cohesively, reliably, and as intended.

Technical Issues and Solutions

Overview of Technical Challenges in the IoT Security Monitoring Project

During the implementation of the IoT security monitoring project, several technical challenges arose across different parts of the project. These issues affected data flow, alert generation, and system stability. Each problem required investigation and adjustments to ensure the full pipeline functioned correctly from the IoT device to the AWS and ELK environments.

Mitigating Elasticsearch Memory Constraints

To reduce the risk of Elasticsearch heap size issues, we chose to use a t2.large instance instead of a t2.medium for our EC2 server. By default, Elasticsearch allocates a 2 GiB Java heap which would consume half of a t2.medium's 4 GiB of RAM and leave insufficient memory for Kibana, Filebeat, Suricata, Mosquito, and the OS. This configuration can lead to memory pressure and potential crashes as log volume or attack simulations increase. Using a t2.large, which provides additional RAM, lowers the likelihood of these problems without requiring manual heap adjustments, allowing the ELK and IoT monitoring stack to operate more reliable under load.

Raspberry Pie to AWS IoT Core Integration Issues

While the infrastructure change improved overall stability, we also faced challenges on the device and data ingestion side in Part 2. During part 2, we encountered a critical integration issue where the Raspberry Pi did not successfully send temperature readings from the MCP9808 sensor to AWS IoT Core, so no messages appeared in the AWS IoT Core test interface. Although the Python script was executed, some components in the chain from the sensor, to the I2C bus read, to the JSON payload, to the MQTT publish, and finally to the AWS IoT Core was not functioning as intended. We attempted to troubleshoot individual components, including redownloading the IoT Thing certificate and private key files, but to no avail. As a result, we chose to redo the entirety of Part 2, and the system functioned correctly afterward.

Tuning Suricata Rules for Effective MQTT Attack Detection

For monitoring and detection in Part 3, we observed that Suricata was not generating any of the expected custom alerts for MQTT based attacks, even though there was clearly traffic on port 1883. The root cause was that our initial rules were not properly tuned for the project's traffic pattern, which resulted in Suricata either ignoring the traffic or never reaching the thresholds required to trigger alerts. We resolved this by rewriting the rules to explicitly target $HOME_NET on port 1883 and by introducing clearer conditions for each scenario: a threshold based rule for high MQTT traffic (count 500 in 60 seconds), a payload size based rule for large MQTT packets (dsize:>1024), and a content based rule to detect malformed MQTT payloads (content:"{malformed"; nocase). After loading these updated rules and restarting Suricata, the custom alerts began to appear as expected in the logs and within Kibana.

Optimizing Kibana Visualizations for High Volume DDoS Traffic

Finally, as we focused on presenting and analyzing the detected attack traffic, we encountered a visualization related issue in Part 3. Our visualization platform repeatedly crashed when we attempted to construct a detailed table view for the DDoS traffic. This behavior was likely caused by the high event volume and the computational cost of aggregating and rendering a large table with many rows and fields. The resulting load affected both the browser and the Elasticsearch/Kibana stack, making the dashboard unstable during intensive attack simulations. To address this problem, we replaced the table based view with a heatmap visualization for the DDoS analysis. The heatmap provides an aggregated view of traffic density and is significantly less resource intensive, allowing the dashboard to remain responsive while still clearly highlighting spikes and patterns in the attack traffic.