PROJECT REPORT

Bank Server DoS Monitoring System

JADE HOANG
BACHELOR OF COMPUTER SCIENCE – MAJOR: CYBERSECURITY

TABLE OF CONTENTS

1. Project Introduction

1.1. Problem Statement

Financial institutions that rely on online services to provide customers with uninterrupted access to banking functionalities often face threats from Denial of Service (DoS) attacks. These attacks can disrupt critical banking services, leading to significant downtime and compromising customer trust. To mitigate these risks, banks require a monitoring system that can:

- Detect potential DoS attacks in real-time
- Monitor system resource usage effectively
- Automate threat responses to minimise downtime
- Provide clear insights into system performance and security metrics

1.2. System Overview

The **Bank Server DoS Monitoring System** is a Message Queuing Telemetry Transport (MQTT)-based application that monitors bank server resources (CPU, RAM) and network traffic to detect and respond to potential DoS attacks. The system uses a publish-subscribe architecture with four main components:

- Sensors deployed on the bank server (server_sensor.py)
- A monitoring system to detect abnormal behaviour (monitor.py)
- A controller (acting as a load balancer) to dynamically scale up the server or enable rate-limiting during an attack (controller.py)
- A user interface designed to replace Graphical MQTT Client displaying server metrics and alerts, and is capable of sending commands (ui.py).

2. Running Instructions

Prerequisites:

- Python 3
- Pip package manager: for installing Python libraries
- Paho-mqtt library: for MQTT client communication
- Tkinter (comes pre-installed with Python): for building the UI

To run the system, we need to install the required Python package:

pip install paho-mqtt

Then run each component in this sequence:

1. Start the Server Sensor (server_sensor.py) to begin publishing metrics: **python server_sensor.py**
2. Start the Monitoring Application (monitor.py) to begin monitoring the metrics: **python monitor.py**
3. Start the Controller (controller.py) to process control commands: **python controller.py**
4. Launch the UI (ui.py) to visualise the data and alerts: **python ui.py**

3. Architecture Diagram

The diagram below displays the architectural overview of this system:

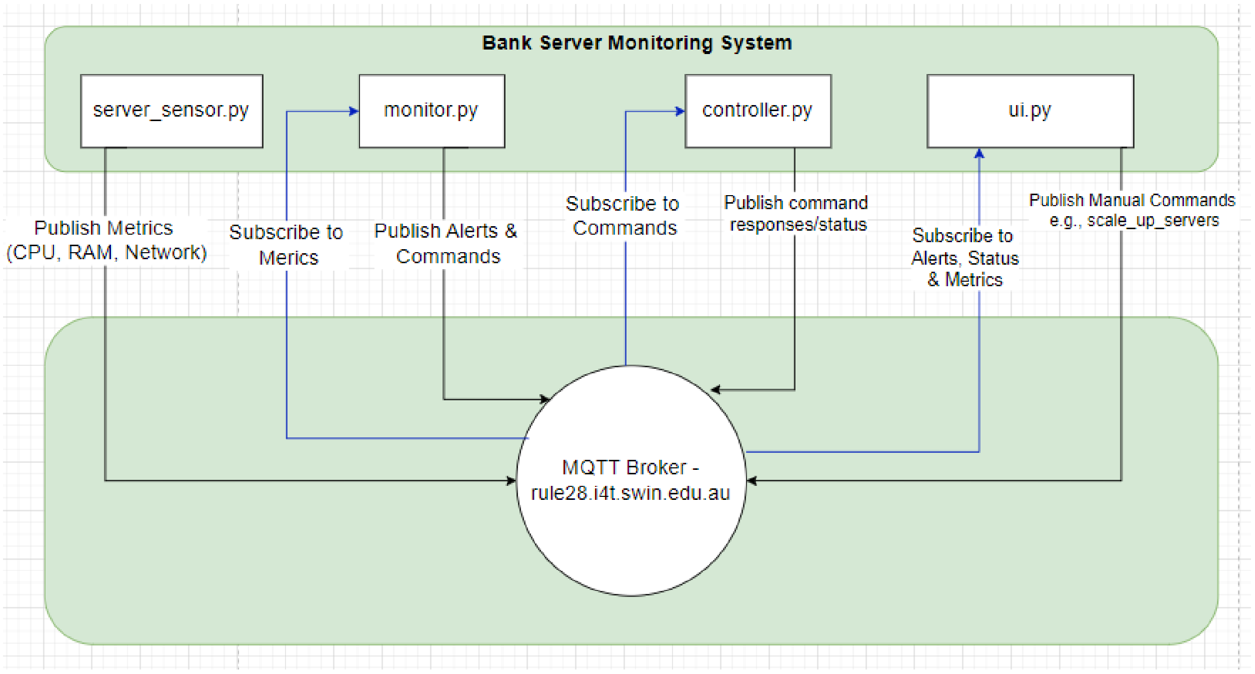

Bank Server Monitoring System
server_sensor.py
monitor.py
controller.py
ui.py
Publish Metrics (CPU, RAM, Network)
Subscribe to Merics
Publish Alerts & Commands
Subscribe to Commands
Publish command responses/status
Subscribe to Alerts, Status & Metrics
Publish Manual Commands e.g., scale_up_servers
MQTT Broker - rule28.i4t.swin.edu.au

Figure 1. Architecture Diagram

4. Workflow Scenario

The following illustrates an example workflow of the **Bank Server DoS Monitoring System**:

1. During normal operations, the **server_sensor.py** collects data on CPU usage, RAM usage, and network traffic at five-second intervals. This data is published to the private MQTT **metrics** topics.
2. Suddenly, the resource monitoring application (**monitor.py**) detects an unusual spike in network traffic—10 times the normal level.
3. Upon receiving the metric data, **monitor.py** identifies that the traffic spike exceeds predefined thresholds indicative of a potential DoS attack (e.g., 700 SYN packets received in a very short time). It generates an alert and publishes it to the **alerts** MQTT topic, notifying system administrators and the security team.
4. The **monitor.py** application also automatically initiates predefined countermeasures, such as enabling rate limiting on incoming traffic to protect server resources. This command is sent to the **control command** topic.
5. The load balancer (**controller.py**) receives the command and implements the necessary action. It also publishes the status of the rate-limiting command (fail or success) to the **rate_limiting status** topic, keeping the system updated on the command's execution status.
6. Meanwhile, the graphical interface (**ui.py**) updates data in real-time, displaying traffic spikes, the generated alert, and the status of the rate-limiting command. Administrators can use this interface to make informed decisions such as sending manual control commands as needed.

5. Communication Topics

This project uses a total of 8 topics: 7 private topics and 1 public topic:

Topic Name	Description
Private Topics	
<103795587>/bank_server/metrics/cpu	CPU usage metrics
<103795587>/bank_server/metrics/ram	RAM usage metrics
<103795587>/bank_server/metrics/network	Network traffic metrics
<103795587>/bank_server/alerts	System alerts (triggered if metrics exceed detection thresholds)
<103795587>/bank_server/control/command	Control commands for the load balancer
<103795587>/bank_server/status/scaling	Status of scaling commands (success or failure)
<103795587>/bank_server/status/rate_limiting	Status of rate-limiting commands (success or failure)
Public Topic	
public/#	Public messages

Table 1. Communication topics summary

6. Component Details

6.1. Server Sensor (server_sensor.py)

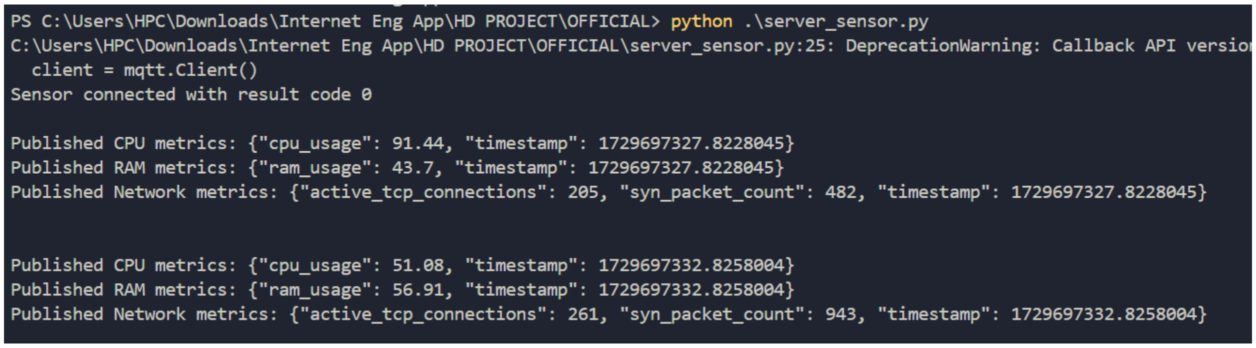
The server sensor is responsible for collecting the following metrics:

- **CPU usage**: Percentage of CPU resources being used.
- **RAM usage**: Percentage of memory in use.
- **Active TCP connections**: The number of active TCP connections on the server.
- **SYN packet count**: The number of incoming SYN packets, which can indicate a SYN flood attack (a type of DoS attack).

Publishing Topics: The sensor then publishes collected metrics to their corresponding topic:

- **<103795587>/bank_server/metrics/cpu**: CPU usage metrics
- **<103795587>/bank_server/metrics/ram**: RAM usage metrics
- **<103795587>/bank_server/metrics/network**: Network metrics

The sensor publishes messages to these topics every 5 seconds.



```
PS C:\Users\HPC\Downloads\Internet Eng App\HD PROJECT\OFFICIAL> python .\server_sensor.py
C:\Users\HPC\Downloads\Internet Eng App\HD PROJECT\OFFICIAL\server_sensor.py:25: DeprecationWarning: Callback API versio
  client = mqtt.Client()
Sensor connected with result code 0

Published CPU metrics: {"cpu_usage": 91.44, "timestamp": 1729697327.8228045}
Published RAM metrics: {"ram_usage": 43.7, "timestamp": 1729697327.8228045}
Published Network metrics: {"active_tcp_connections": 205, "syn_packet_count": 482, "timestamp": 1729697327.8228045}


Published CPU metrics: {"cpu_usage": 51.08, "timestamp": 1729697332.8258004}
Published RAM metrics: {"ram_usage": 56.91, "timestamp": 1729697332.8258004}
Published Network metrics: {"active_tcp_connections": 261, "syn_packet_count": 943, "timestamp": 1729697332.8258004}
```

Figure 2. Running server_sensor.py

6.2. Monitor Application (monitor.py)

The monitor application subscribes to all metrics topics **(<103795587>/bank_server/metrics/#)** along with the public topic, and compares the received data against predefined thresholds:

- CPU Usage: >80%
- RAM Usage: >80%
- SYN Packet Count: >700 packets
- Active TCP Connections: >400 connections

Subscribing Topics:

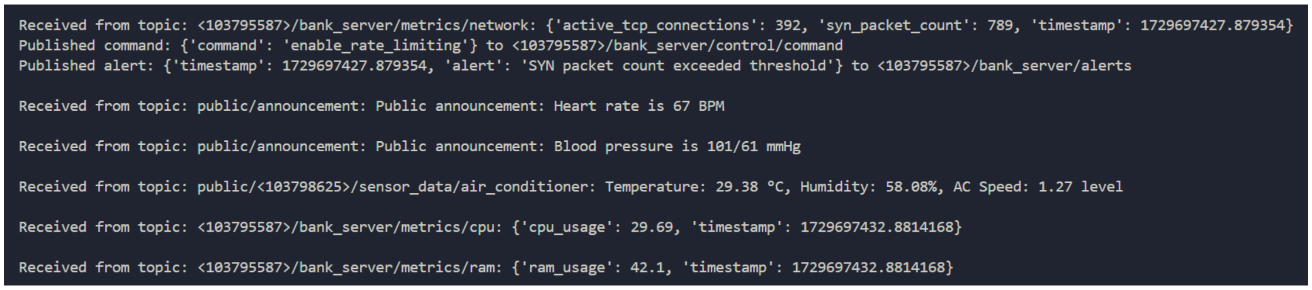
- **<103795587>/bank_server/metrics/#**: All metrics
- **public/#**: Public channel messages

Publishing Topics: When a metric exceeds the threshold, the monitor publishes control commands to **<103795587>/bank_server/control/command** and sends alerts to **<103795587>/bank_server/alerts**.

- **<103795587>/bank_server/control/command**: Control commands
- **<103795587>/bank_server/alerts**: System alerts

Alert and Command types:

1. **High CPU/RAM Usage**: If the CPU usage or RAM usage exceeds 80%
   - **Alert**: "CPU/RAM usage exceeded threshold"
   - **Command**: scale_up_servers
2. **Network Anomalies**: If the SYN packet count exceeds 700 or TCP connections exceed 400:
   - **Alert**: "SYN packet count/TCP connections exceeded threshold"
   - **Command**: enable_rate_limiting



```
Received from topic: <103795587>/bank_server/metrics/network: {'active_tcp_connections': 392, 'syn_packet_count': 789, 'timestamp': 1729697427.879354}
Published command: {'command': 'enable_rate_limiting'} to <103795587>/bank_server/control/command
Published alert: {'timestamp': 1729697427.879354, 'alert': 'SYN packet count exceeded threshold'} to <103795587>/bank_server/alerts

Received from topic: public/announcement: Public announcement: Heart rate is 67 BPM

Received from topic: public/announcement: Public announcement: Blood pressure is 101/61 mmHg

Received from topic: public/<103798625>/sensor_data/air_conditioner: Temperature: 29.38 °C, Humidity: 58.08%, AC Speed: 1.27 level

Received from topic: <103795587>/bank_server/metrics/cpu: {'cpu_usage': 29.69, 'timestamp': 1729697432.8814168}

Received from topic: <103795587>/bank_server/metrics/ram: {'ram_usage': 42.1, 'timestamp': 1729697432.8814168}
```

Figure 3. Running monitor.py

6.3. Controller Application (controller.py)

The controller acts as a load balancer and responds to commands published by the monitor. It subscribes to the control command topic and performs actions based on the command received:

- **Scale Up Servers**: Increases server capacity to handle additional load.
- **Enable Rate Limiting**: Restricts incoming traffic to reduce the load from potential DoS attacks.

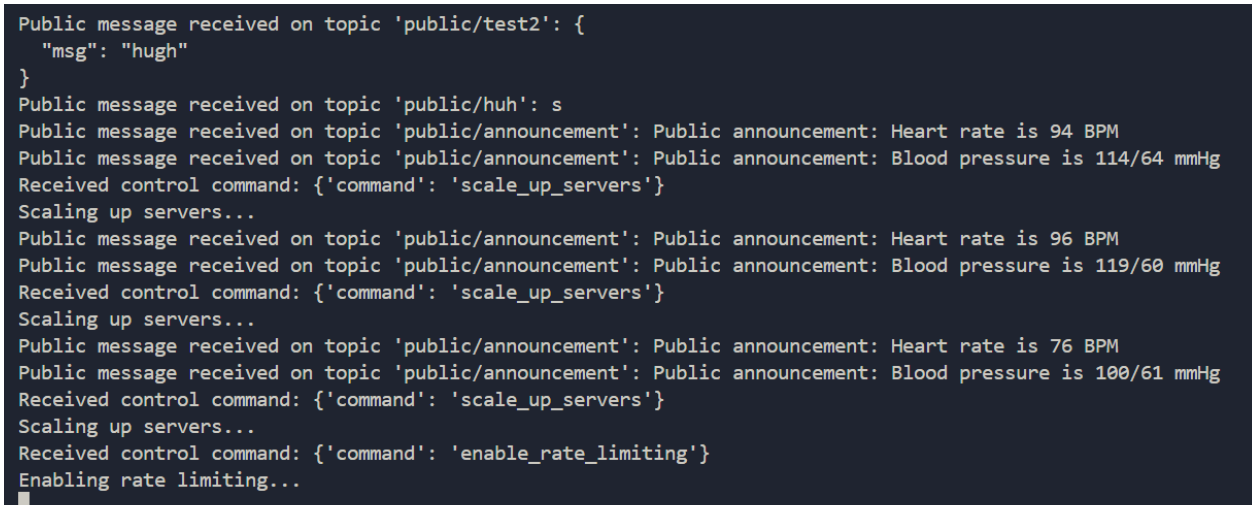
Subscribing Topics:

- **<103795587>/bank_server/control/command**: Control commands

- **public/#:** Public messages

Publishing Topics: The controller also publishes the status of these operations (e.g., success or failure) to:

- **<103795587>/bank_server/status/scaling**: Scaling operation status
- **<103795587>/bank_server/status/rate_limiting**: Rate limiting status



```
Public message received on topic 'public/test2': {
  "msg": "hugh"
}
Public message received on topic 'public/huh': s
Public message received on topic 'public/announcement': Public announcement: Heart rate is 94 BPM
Public message received on topic 'public/announcement': Public announcement: Blood pressure is 114/64 mmHg
Received control command: {'command': 'scale_up_servers'}
Scaling up servers...
Public message received on topic 'public/announcement': Public announcement: Heart rate is 96 BPM
Public message received on topic 'public/announcement': Public announcement: Blood pressure is 119/60 mmHg
Received control command: {'command': 'scale_up_servers'}
Scaling up servers...
Public message received on topic 'public/announcement': Public announcement: Heart rate is 76 BPM
Public message received on topic 'public/announcement': Public announcement: Blood pressure is 100/61 mmHg
Received control command: {'command': 'scale_up_servers'}
Scaling up servers...
Received control command: {'command': 'enable_rate_limiting'}
Enabling rate limiting...
```

Figure 4. Running controller.py

6.4. User Interface (ui.py)

The UI is a graphical interface that provides a real-time display of server metrics and alerts just like a Graphical MQTT Client (e.g., MQTTX). It has multiple tabs to show:

- CPU Metrics
- RAM Metrics
- Network Metrics
- Alerts
- Command Status
- Public Messages

It also has these buttons to publish messages to the **<103795587>/bank_server/control/command** topic:

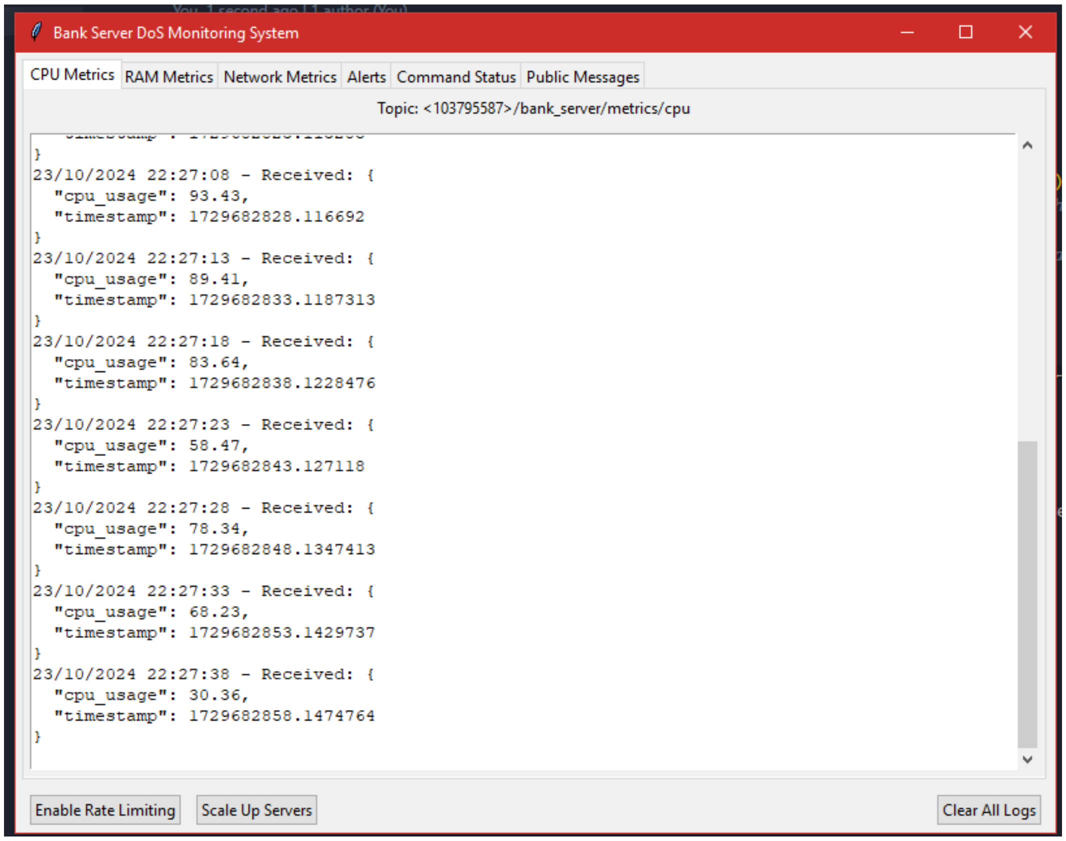
- **Enable Rate Limiting button**: publish command "enable_rate_limiting"
- **Scale Up Servers button**: publish command "scale_up_servers"
- **Clear All Logs button**: clear all messages in 6 tabs.

Subscribing Topics:

- **<103795587>/bank_server/metrics/cpu**: CPU usage metrics
- **<103795587>/bank_server/metrics/ram**: RAM usage metrics
- **<103795587>/bank_server/metrics/network**: Network metrics
- **<103795587>/bank_server/alerts**: System alerts
- **<103795587>/bank_server/status/#**: Status updates for commands (success or failure messages)
- **public/#:** Receive all public messages under the public topic hierarchy.

Publishing Topics:

- **<103795587>/bank_server/control/command**: Sends commands like enable_rate_limiting or scale_up_servers.



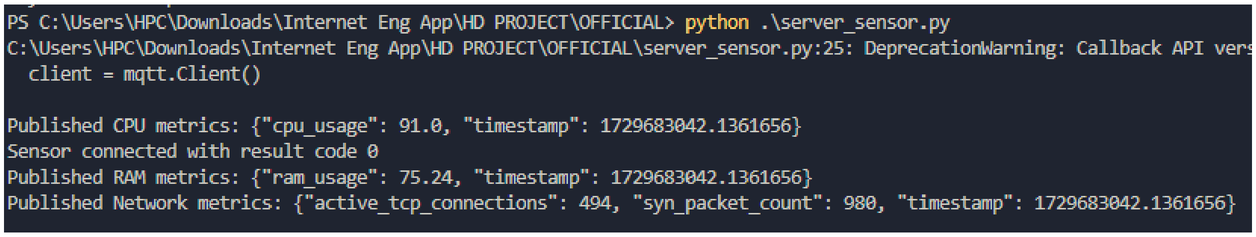
```
}
23/10/2024 22:27:08 - Received: {
  "cpu_usage": 93.43,
  "timestamp": 1729682828.116692
}
23/10/2024 22:27:13 - Received: {
  "cpu_usage": 89.41,
  "timestamp": 1729682833.1187313
}
23/10/2024 22:27:18 - Received: {
  "cpu_usage": 83.64,
  "timestamp": 1729682838.1228476
}
23/10/2024 22:27:23 - Received: {
  "cpu_usage": 58.47,
  "timestamp": 1729682843.127118
}
23/10/2024 22:27:28 - Received: {
  "cpu_usage": 78.34,
  "timestamp": 1729682848.1347413
}
23/10/2024 22:27:33 - Received: {
  "cpu_usage": 68.23,
  "timestamp": 1729682853.1429737
}
23/10/2024 22:27:38 - Received: {
  "cpu_usage": 30.36,
  "timestamp": 1729682858.1474764
}
```

Figure 5. Running ui.py

7. View Messages in Graphical MQTT Client

This section demonstrates received messages from device 1 (server_sensor.py) and generating a message and sending it to device 2 (monitor.py) using a graphical MQTT client (Pass task requirements):

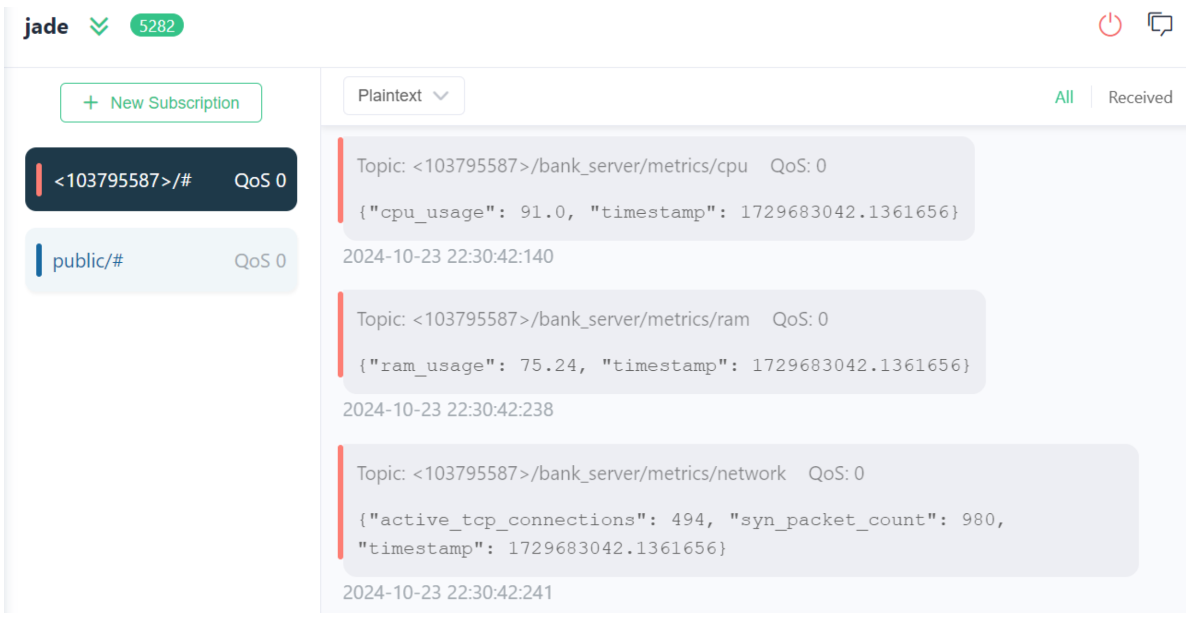
Test publishing data from the sensor and the message shown successfully in MQTTX:



```
PS C:\Users\HPC\Downloads\Internet Eng App\HD PROJECT\OFFICIAL> python .\server_sensor.py
C:\Users\HPC\Downloads\Internet Eng App\HD PROJECT\OFFICIAL\server_sensor.py:25: DeprecationWarning: Callback API vers
  client = mqtt.Client()

Published CPU metrics: {"cpu_usage": 91.0, "timestamp": 1729683042.1361656}
Sensor connected with result code 0
Published RAM metrics: {"ram_usage": 75.24, "timestamp": 1729683042.1361656}
Published Network metrics: {"active_tcp_connections": 494, "syn_packet_count": 980, "timestamp": 1729683042.1361656}
```

Figure 6. Device 1 (server_sensor.py) publishes messages



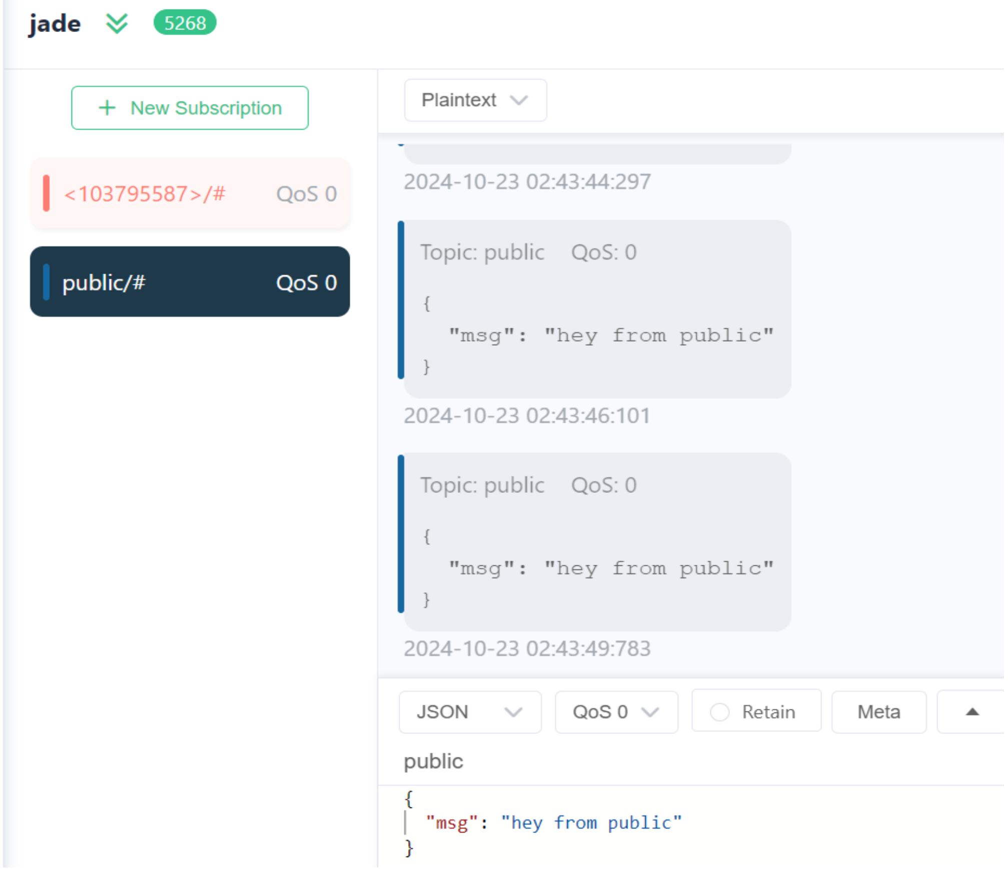
```
Topic: <103795587>/bank_server/metrics/cpu    QoS: 0
{"cpu_usage": 91.0, "timestamp": 1729683042.1361656}
2024-10-23 22:30:42:140

Topic: <103795587>/bank_server/metrics/ram    QoS: 0
{"ram_usage": 75.24, "timestamp": 1729683042.1361656}
2024-10-23 22:30:42:238

Topic: <103795587>/bank_server/metrics/network    QoS: 0
{"active_tcp_connections": 494, "syn_packet_count": 980,
"timestamp": 1729683042.1361656}
2024-10-23 22:30:42:241
```

Figure 7. MQTTX client successfully received messages from the sensor

Test sending a message to the “public” topic from MQTTX and monitor.py successfully receives and prints that message:



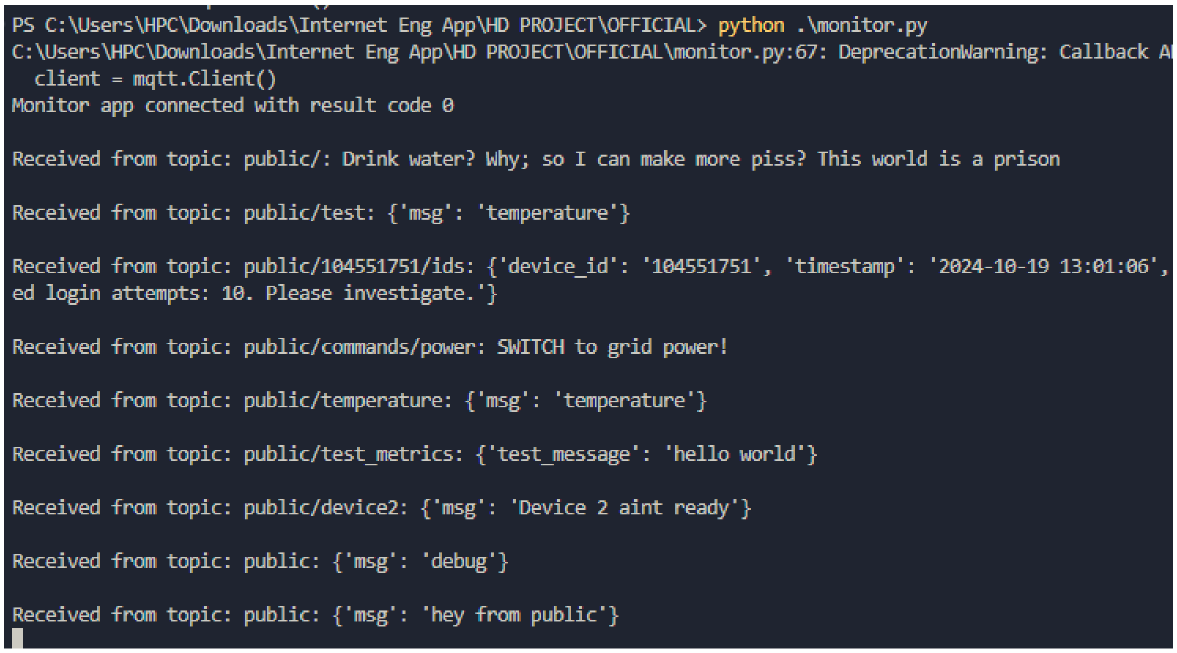
```
2024-10-23 02:43:44:297

Topic: public    QoS: 0
{
  "msg": "hey from public"
}
2024-10-23 02:43:46:101

Topic: public    QoS: 0
{
  "msg": "hey from public"
}
2024-10-23 02:43:49:783
```

```
public
{
  "msg": "hey from public"
}
```

Figure 8. Sending "hey from public" from MQTTX to the “public” topic



```
PS C:\Users\HPC\Downloads\Internet Eng App\HD PROJECT\OFFICIAL> python .\monitor.py
C:\Users\HPC\Downloads\Internet Eng App\HD PROJECT\OFFICIAL\monitor.py:67: DeprecationWarning: Callback A
  client = mqtt.Client()
Monitor app connected with result code 0

Received from topic: public/: Drink water? Why; so I can make more piss? This world is a prison

Received from topic: public/test: {'msg': 'temperature'}

Received from topic: public/104551751/ids: {'device_id': '104551751', 'timestamp': '2024-10-19 13:01:06',
ed login attempts: 10. Please investigate.'}

Received from topic: public/commands/power: SWITCH to grid power!

Received from topic: public/temperature: {'msg': 'temperature'}

Received from topic: public/test_metrics: {'test_message': 'hello world'}

Received from topic: public/device2: {'msg': 'Device 2 aint ready'}

Received from topic: public: {'msg': 'debug'}

Received from topic: public: {'msg': 'hey from public'}
```

Figure 9. Device 2 (monitor.py) subscribing to "public" topic successfully received the message.

8. Cybersecurity Considerations

8.1. Security Issues in On-Site MQTT Broker Deployment (C tasks)

This section details the security vulnerabilities associated with the current internal MQTT broker deployment. The organisation's MQTT broker currently operates within a closed network environment, serving as a site-wide IoT solution. While the internal-only deployment provides some inherent security through network isolation, several critical security concerns have been identified.

Authentication Vulnerabilities

The primary vulnerability stems from the basic authentication system currently in place, which relies solely on simple username/password combinations. This represents a significant security risk as these credentials are transmitted in plaintext across the network, making them susceptible to interception through internal Man-in-the-middle (MitM) attacks such as network sniffing attacks [1]. Additionally, using the student ID as both username and password is not a strong authentication method. Attackers could easily gain access to the broker by guessing or stealing credentials, allowing them to publish/subscribe to sensitive topics and gaining access to the entire monitoring infrastructure.

For a banking system monitoring critical DoS attacks, relying on basic authentication is severely inadequate. If an attacker compromises these credentials, they could access real-time server resource metrics, potentially exposing system vulnerabilities. Additionally, the attacker could publish false monitoring data to conceal actual DoS attacks or submit malicious control commands, such as scaling server resources or enabling rate limiting on the load balancer. This could further allow them to manipulate rate-limiting parameters during ongoing attacks.

To overcome this, it is recommended to replace studentID-based authentication with unique credentials and enforce password complexity requirements (minimum length, special characters and numbers) along with regular password rotation policies (every 30-90 days). It is also suggested that multi-factor authentication (MFA) be deployed for broker access [7].

Authorisation Issues

The current access control mechanism presents a significant security concern. It lacks granular, topic-level controls, as all MQTT clients share the same credential set, limiting differentiation in access permissions. Access control is only applied to top-level topics (e.g., <username> and public), which restricts interactions but fails to enforce fine-grained controls over sub-topics. As a result, there is no way to limit access based on specific roles, such as distinguishing between sensors and controllers. This violates the principle of least privilege, where each device or user should only have the minimum necessary permissions to perform its function. Without implementing role-based access control (RBAC) to define who can publish or subscribe to specific sub-topics, it becomes much harder to properly segregate device permissions and effectively monitor client activities. In an IoT environment, where devices should have tightly restricted access based on their roles, this lack of fine-grained control is particularly problematic [2, 4].

To enhance security, it is recommended to implement RBAC that defines specific roles for each component, such as sensors, monitors, controllers, and user interfaces. Granular permissions should then be established for different topic hierarchies, enforcing strict publish/subscribe rules based on roles. For instance, a sensor should only be allowed to publish messages to “metrics/#” topics and be restricted from accessing other topics, while only the monitor application and graphical user interface should be permitted to read messages from the “alerts” topic. This approach ensures that each component has the minimum necessary access, following the least privilege principle.

Unencrypted Communication

The current deployment of the MQTT broker utilises the default MQTT port 1883 without encryption, resulting in all communication between sensors, the monitoring system, the controller application, and the user interface being transmitted in plaintext. This lack of encryption poses significant risks, exposing sensitive operational data to potential internal threats such as eavesdropping, tampering, and MITM attacks [1, 3]. Sensitive information, including real-time data regarding bank sector systems, is particularly vulnerable under these conditions.

By using the default port without encryption, several critical vulnerabilities arise. Server resource metrics, such as CPU and RAM usage, as well as network traffic data—are transmitted in plaintext, making them easily accessible to any malicious actors on the Internet. Detection data related to DoS attacks could be intercepted and manipulated, allowing attackers to mislead security measures and disrupt operations. Additionally, commands related to load balancing and rate limiting are susceptible to tampering, which could result in ineffective responses to real threats. Attackers could also gather intelligence about server resource utilisation patterns, which may help them understand the bank's DoS detection thresholds and defence mechanisms.

These vulnerabilities present a significant risk, as understanding the operational metrics could enable attackers to craft more sophisticated attacks designed to evade existing detection systems. To mitigate these risks, it is highly recommended to switch to the more secure port 8883 and enable TLS encryption to protect MQTT messages [1].

8.2. Security Issues of Internet-Facing Deployment MQTT Broker (HD tasks)

This section outlines the security challenges associated with deploying the MQTT broker of the IoT infrastructure to the Internet. The expansion of access to the broker significantly increases the attack surface, making it more susceptible to various threats, including Distributed Denial of Service (DDoS) attacks and sophisticated breach attempts [5, 6].

Increased Risk of MITM attacks

When MQTT messages are transmitted over a public network without encryption, sensitive data can be easily intercepted by anyone with network access. This may include critical operational data, such as server resource usage metrics (CPU and RAM), network traffic information, and user commands. The use of the default MQTT port (1883) makes the broker a visible target for global attackers, thereby increasing its vulnerability to port scanning and exploitation attempts [1].

Additionally, attackers can launch brute-force attacks against the existing simple username/password authentication system from any location. Given the weak nature of the current authentication method, it is susceptible to being compromised, leading to unauthorised access. To mitigate this risk, it is strongly recommended to transition to a more secure cryptographic authentication mechanism when deploying the system over the Internet, rather than relying on basic username and password combinations [10].

DoS attacks

With the broker exposed to the Internet, the potential for DoS attacks increases [1, 5]. Attackers can target the broker to overwhelm it with traffic, disrupting services. To mitigate this risk, the organisation should implement throttling methods to prevent the MQTT broker from being flooded with numerous erroneous subscription and publication messages. By controlling the rate of incoming messages, resource consumption can be managed, thus maintaining service availability even during malicious attempts to disrupt operations [5].

Lack of Message Integrity and Replay Protection

The MQTT broker does not verify message integrity or protect against replay attacks [5, 6]. Attackers on the internet can intercept and alter MQTT messages or replay previously sent commands, potentially disrupting the system's operations in various ways. For example, they could modify resource usage metrics to conceal ongoing DoS attacks, replay old "normal" metrics during an active attack to create confusion, inject false scaling or rate-limiting commands that degrade system performance, or manipulate attack alert messages, causing misinterpretation of the threat landscape. These vulnerabilities allow attackers to bypass DoS detection mechanisms or trigger unnecessary defensive actions, resulting in service disruptions due to false positives.

To address these issues, it is recommended to implement a hash-based message authentication code (HMAC), to ensure the integrity of messages and confirm they have not been tampered with. Furthermore, integrating replay attack protection, such as adding timestamps to all messages, will enhance the security posture of the MQTT broker [5].

Multi-site Data Exposure

When it comes to multi-site data exposure, the current architecture of the MQTT broker raises serious issues regarding the handling of banking data over the Internet. The basic username/password system does not effectively isolate access between different banking sites, making it challenging to ensure that only authorised personnel from specific locations can access their respective data. For instance, if multiple bank branches across different cities utilise the same MQTT broker for DoS monitoring, any authorised user from one branch could access sensitive data or send commands related to other branches without proper authorisation.

Additionally, there is currently no mechanism to properly isolate DoS monitoring data and control commands between different banking locations. A breach at one site could potentially expose sensitive information from all connected banking locations. If the system permits all branches to publish and subscribe to shared topics without segmentation, a breach at one branch could compromise data from all branches.

Cross-Site Command Publishing Issues

The current deployment of the MQTT broker lacks adequate safeguards against unauthorised command publishing when exposed to the Internet. Attackers could issue commands related to scaling or rate-limiting that affect multiple bank sites due to the absence of validation mechanisms to confirm the origin of these commands. For instance, the system does not verify whether monitoring data originates from legitimate bank sensors, such as by checking access tokens [8]. The absence of message signing enables attackers to tamper with DoS alert messages during transmission, which could lead to confusion and severe operational consequences. Malicious actors could trigger false DoS responses across multiple bank locations or mask real attacks through manipulated monitoring data, resulting in significant disruptions.

To strengthen security, it is essential to implement site-based access controls. The existing access control and data segregation strategies must be reworked to develop a topic structure that facilitates granular access controls. This includes ensuring that each location has its own credentials to isolate sensitive data from different banking sites and reduce the risk of cross-site exposure. Furthermore, enforcing strict topic ACLs based on site and component role will control which users or systems can publish or subscribe to specific topics [9].

In conclusion, exposing the MQTT broker to the Internet presents multiple security challenges that need to be addressed to protect the organisation's banking infrastructure. By proactively addressing these vulnerabilities, the organisation can significantly enhance the security posture of its MQTT broker and ensure the safe operation of its IoT infrastructure in a multi-site banking environment.

9. Conclusion

This Bank Server DoS Monitoring System provides a solution for detecting and mitigating DoS attacks on bank servers for the banking sector. By leveraging MQTT for real-time communication between sensors, the monitoring system, and the load balancer, the system can dynamically adjust server resources to maintain availability and performance under attacks.

10. References

[1] D. Kant, A. Johannsen, and R. Creutzburg, “Analysis of IoT Security Risks based on the exposure of the MQTT Protocol,” *Electronic Imaging*, vol. 33, no. 3, pp. 96–196–8, Jun. 2021, doi: https://doi.org/10.2352/issn.2470-1173.2021.3.mobmu-096.

[2] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, and A. Panya, “Authorization mechanism for MQTT-based Internet of Things,” *2016 IEEE International Conference on Communications Workshops (ICC)*, May 2016, doi: https://doi.org/10.1109/iccw.2016.7503802.

[3] A. Singh, A. Kumar, and V. Kumar, “A Study on MQTT Protocol and its Cyber Attacks,” *IARJSET International Advanced Research Journal in Science*, vol. 9, no. 1, 2022, doi: https://doi.org/10.17148/IARJSET.2022.9136.

[4] E. Atilgan, I. Ozcelik, and E. N. Yolacan, “MQTT Security at a Glance,” Dec. 2021, doi: https://doi.org/10.1109/iscturkey53027.2021.9654337.

[5] F. Chen, Y. Huo, J. Zhu, and D. Fan, “A Review on the Study on MQTT Security Challenge,” *IEEE Xplore*, Nov. 01, 2020. https://ieeexplore.ieee.org/document/9265962 (accessed Oct. 23, 2024).

[6] A. Hintaw, S. Manickam, S. Karuppayah, M. Abomaali, and M. Aboalmaaly, “A Brief Review on MQTT’s Security Issues within the Internet of Things (IoT) A Brief Review on MQTT’s Security Issues within the Internet of Things (IoT),” *Article in Journal of Communications*, 2019, doi: https://doi.org/10.12720/jcm.14.6.463-469.

[7] M. Saqib and A. H. Moon, “A Novel Lightweight Multi-factor Authentication Scheme for MQTT-based IoT Applications,” *Microprocessors and Microsystems*, vol. 110, pp. 105088–105088, Aug. 2024, doi: https://doi.org/10.1016/j.micpro.2024.105088.

[8] B. Tejaswi, M. Mannan, and A. Youssef, “Security Weaknesses in IoT Management Platforms,” *IEEE Internet of Things Journal*, vol. 11, no. 1, pp. 1572–1588, Jan. 2024, doi: https://doi.org/10.1109/JIOT.2023.3289754.

[9] D. I. Dikii, “Remote Access Control Model for MQTT Protocol,” *IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, Jan. 2020, doi: https://doi.org/10.1109/eiconrus49466.2020.9039122.

[10] S. V, V. A, and S. Pattar, “MQTT based Secure Transport Layer Communication for Mutual Authentication in IoT Network,” Global Transitions Proceedings, Apr. 2022, doi: https://doi.org/10.1016/j.gltp.2022.04.015.