# Simple Authenticator

Name: Md Redwan Ahmed Zawad
Course: *Bachelor of Engineering (Software)*
Institute: *Swinburne University of Technology (Student)*
Location: City, Country
Contact: 103501849@student.swin.edu.au

*Abstract*— **This report explains the mechanism behind the implementation of a simple authenticator using mqtt protocol.**

## I. Introduction (*Heading 1*)

The advent of the Internet of Things (IoT) has revolutionized the way devices and applications communicate with each other. MQTT (Message Queuing Telemetry Transport), a lightweight and efficient messaging protocol, has become a cornerstone for IoT communication due to its simplicity and scalability. In this era of interconnected devices, security is paramount, especially when dealing with sensitive data or critical operations.

This report delves into the implementation of a secure MQTT-based authentication system designed to verify the identity of clients connecting to a broker. The script provided in this report serves as a practical example, showcasing how MQTT can be used to establish a rudimentary yet illustrative authentication mechanism. It explores the basics of setting up an MQTT client, subscribing to topics, and responding to incoming messages, all within the context of an authentication workflow.

The script utilizes the Python programming language and the Paho MQTT client library to interact with an MQTT broker. While the provided code is simple in its approach, it serves as a foundation for understanding the fundamental components involved in secure authentication over MQTT.

The report begins by presenting the code, explaining its key components, and providing a step-by-step breakdown of the authentication process. We will discuss the MQTT configuration, user input options, and the client's connection to the broker. Furthermore, we will explore how the script handles incoming messages, identifying messages beginning with the keyword "ForYou," and responding with the acknowledgment "Authenticated."

By the end of this report, readers will have gained insights into the principles of MQTT-based authentication and will be equipped with foundational knowledge that can be extended for more robust and production-ready security solutions. Additionally, potential use cases, security considerations, and room for improvement in the code will be highlighted, allowing for a comprehensive understanding of the practical application of secure MQTT-based authentication in the world of IoT.

## II. Methodology

1. MQTT Setup:

Configuration Parameters: The MQTT setup is fundamental to the script. The script initializes the MQTT client with essential configuration parameters:
Broker Address: The address of the MQTT broker to which the client will connect.
Port: The specific port on which the broker is listening for incoming MQTT connections.
Client ID: A unique identifier for the client, often generated using random values.
Username and Password: Credentials used to authenticate with the MQTT broker.
Establishing Connection: The script establishes a connection with the MQTT broker using the Paho MQTT client library. This step includes configuring the client to use the provided username and password for authentication.

2. User Input:

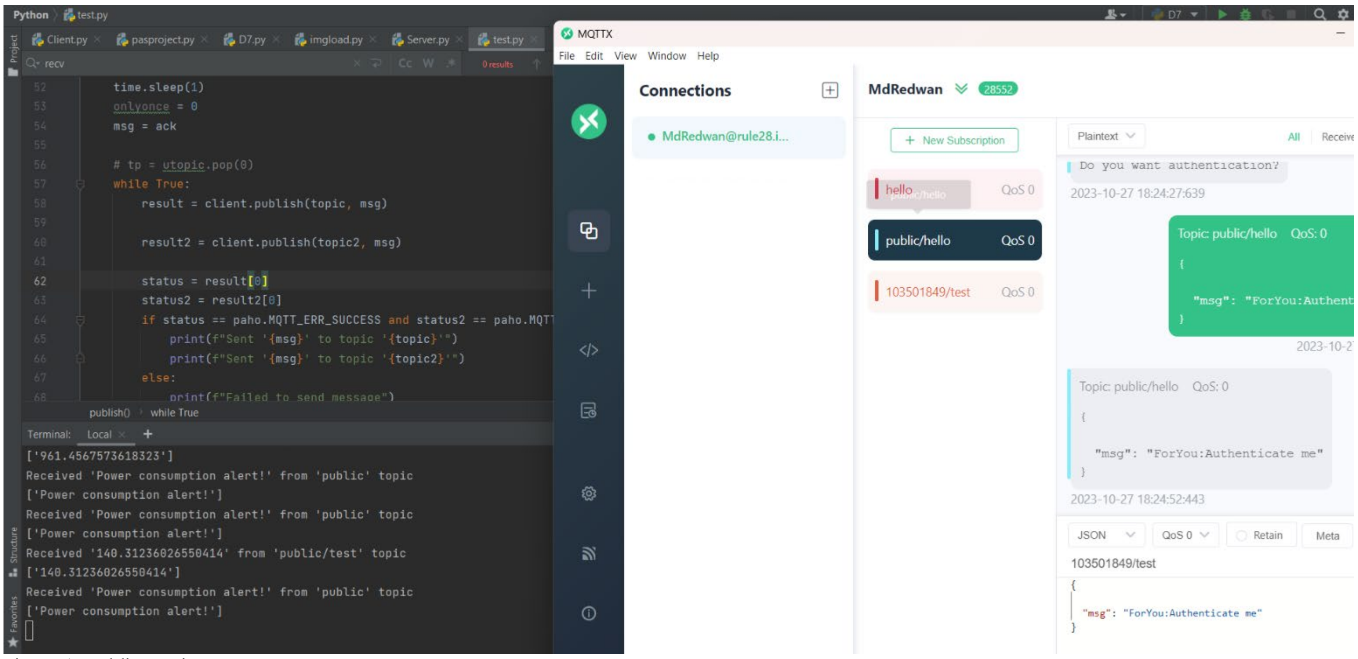
User Input Functions: The script includes two user input functions, namely get_user_input and get_topic. These functions allow for the dynamic input of topics. However, it's important to note that the actual usage of these functions within the script is limited.

3. MQTT Functions:

connect_mqtt Function:
The connect_mqtt function initializes the MQTT client with the provided configuration parameters.
It sets up an on_connect callback to handle the connection status. If the connection is successful, a message indicating the connection status is printed.
publish Function:
The publish function is responsible for sending MQTT messages.
It publishes a message (in this case, "Do you want me to authenticate you?") to a predefined topic ("103501849/test").
The function handles the status of the message publication, printing a success or failure message.
subscribe Function:
The subscribe function is central to the MQTT script. It handles subscription to MQTT topics and message handling.
It subscribes to two topics: "public/#" and "103501849/+".
The function sets up an on_message callback to process incoming messages.
When a message is received, it checks if the message content begins with "ForYou." If this condition is met, the function calls the publish function to send an "Authenticated" message in response.
4. Message Handling:

on_message Callback: Within the subscribe function, the on_message callback is defined to process incoming MQTT messages.
When a message is received, the script prints the payload and the topic from which it was received.
It extracts and analyzes the message content to determine if it starts with "ForYou."
If "ForYou" is detected, the script publishes an "Authenticated" message using the publish function.

Output:



```python
        time.sleep(1)
        onlyonce = 0
        msg = ack

        # tp = utopic.pop(0)
        while True:
            result = client.publish(topic, msg)

            result2 = client.publish(topic2, msg)

            status = result[0]
            status2 = result2[0]
            if status == paho.MQTT_ERR_SUCCESS and status2 == paho.MQTT
                print(f"Sent '{msg}' to topic '{topic}'")
                print(f"Sent '{msg}' to topic '{topic2}'")
            else:
                print(f"Failed to send message")
```

```
['961.4567573618323']
Received 'Power consumption alert!' from 'public' topic
['Power consumption alert!']
Received 'Power consumption alert!' from 'public' topic
['Power consumption alert!']
Received '140.31236026550414' from 'public/test' topic
['140.31236026550414']
Received 'Power consumption alert!' from 'public' topic
['Power consumption alert!']
```

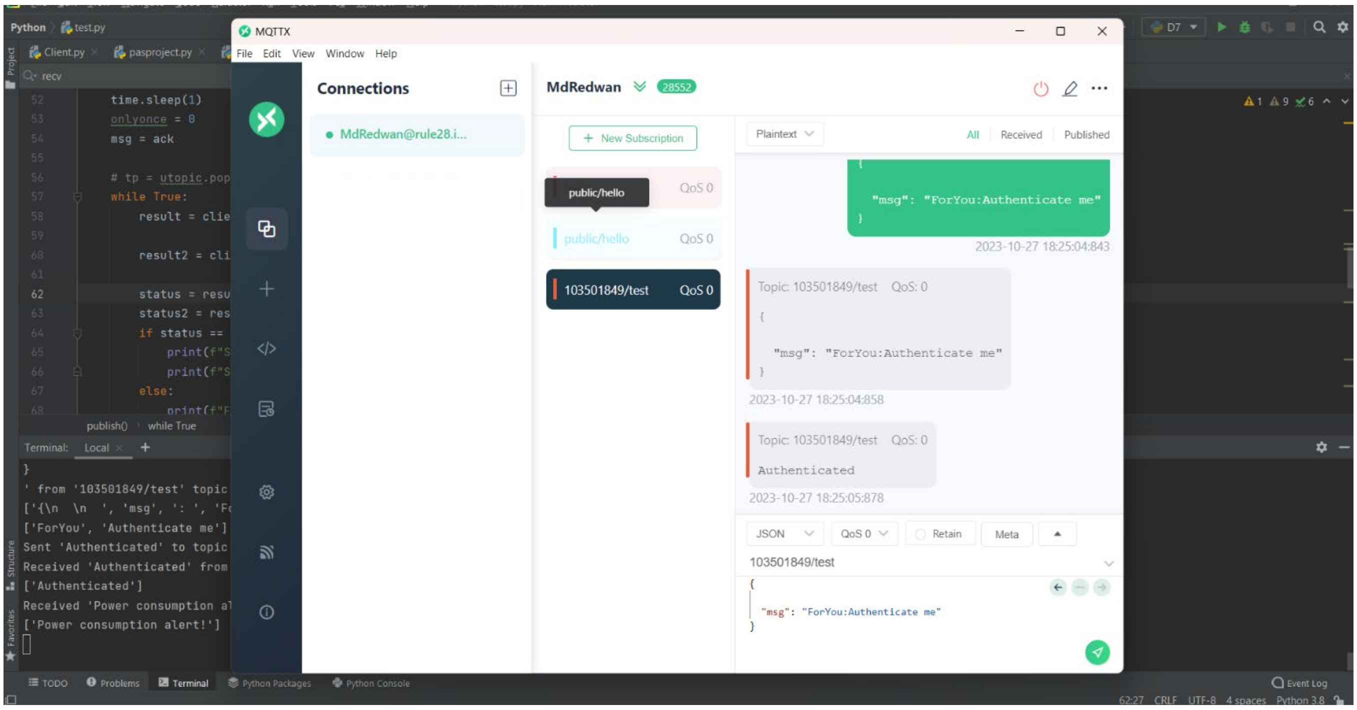
```
Do you want authentication?
2023-10-27 18:24:27:639

Topic: public/hello   QoS: 0
{
  "msg": "ForYou:Authent
}
2023-10-2

Topic: public/hello   QoS: 0
{
  "msg": "ForYou:Authenticate me"
}
2023-10-27 18:24:52:443

103501849/test
{
  "msg": "ForYou:Authenticate me"
}
```

Figure 1: Public Topic Output



```
        time.sleep(1)
        onlyonce = 0
        msg = ack

        # tp = utopic.pop
        while True:
            result = clie

            result2 = cli

            status = resu
            status2 = res
            if status ==
                print(f"S
                print(f"S
            else:
                print(f"F
```

```
}
' from '103501849/test' topic
['{\n  \n  ', 'msg', ': ', 'Fo
['ForYou', 'Authenticate me']
Sent 'Authenticated' to topic
Received 'Authenticated' from
['Authenticated']
Received 'Power consumption al
['Power consumption alert!']
```

```
{
  "msg": "ForYou:Authenticate me"
}
2023-10-27 18:25:04:843

Topic: 103501849/test   QoS: 0
{
  "msg": "ForYou:Authenticate me"
}
2023-10-27 18:25:04:858

Topic: 103501849/test   QoS: 0
Authenticated
2023-10-27 18:25:05:878
```

Figure 2: Authentication message

## III. Discussion

The discussion section of this report explores the MQTT-based authentication script, shedding light on its strengths, limitations, and potential applications. It also delves into the security considerations and areas for improvement, presenting a comprehensive view of the script's practical implications.

1. Script Functionality:

The script demonstrates a basic implementation of an MQTT-based authentication mechanism. It connects to an MQTT broker, subscribes to specific topics, and listens for incoming messages. When a message starting with "ForYou" is received, it responds with an "Authenticated" message. This functionality showcases the core principles of MQTT communication and message handling.

2. Practical Use Cases:

While the script's primary purpose is authentication, its potential use cases extend beyond this. The simplicity and efficiency of MQTT make it suitable for various IoT applications. For instance, it can be used for remote control of devices, real-time monitoring of sensor data, and data exchange between IoT nodes. The authentication aspect ensures that only authorized clients can participate in these operations.

3. Security Considerations:

Security is a paramount concern, especially in IoT applications where sensitive data may be involved. The script introduces a basic level of security by requiring the "ForYou" keyword in the incoming message to trigger authentication. However, it lacks encryption and more robust authentication methods. For enhanced security, it is advisable to implement techniques like TLS/SSL for encryption and mutual authentication.

4. User Input Functionality:

The script includes functions for user input of topics. While this feature has potential, it is not fully integrated into the script's functionality. Further development could involve a more interactive and dynamic way for users to input topics, making it more adaptable to different scenarios.

5. Scalability and Production Use:

The script, as presented, is a simplified example meant for educational and illustrative purposes. In a production environment, scalability, reliability, and error handling are critical considerations. Production-grade MQTT applications often require robust quality of service (QoS) levels and support for multiple clients, which the script currently lacks.

6. Extensibility:

The script serves as a foundation that can be extended to create more complex IoT applications. With additional features and logic, it could be used for advanced use cases, such as home automation, industrial control, and telemetry.

7. Optimization:

The script is a simple demonstration, and there is room for optimization in terms of code structure and efficiency. Cleaning up the code and handling exceptions can make it more reliable and maintainable.

8. Security Best Practices:

For real-world applications, it is crucial to follow MQTT and IoT security best practices. This includes proper access control, strong authentication methods, and regular security audits.

## IV. Cybersecurity concerns

When considering the implementation of MQTT-based authentication in an IoT environment, there are several cybersecurity concerns that must be addressed to ensure the security and integrity of the system. Some of the key cybersecurity concerns for this report include:

Message Encryption:
Without proper message encryption, MQTT communications can be intercepted and read by unauthorized parties. It is essential to implement Transport Layer Security (TLS) or Secure Sockets Layer (SSL) to encrypt data in transit, protecting it from eavesdropping.

Authentication Security:
The use of a basic keyword ("ForYou") as a form of authentication is rudimentary and insecure. In a real-world scenario, strong and multi-factor authentication mechanisms should be implemented to ensure that only authorized clients can access the MQTT broker.

Access Control:
The script should enforce proper access control mechanisms, ensuring that only authenticated clients can subscribe to or publish messages on specific topics. MQTT brokers often provide access control lists (ACLs) that can be configured to restrict client actions.

Authorization and Role-Based Access:
In a production environment, it is crucial to implement role-based access control. Different clients should have specific roles and permissions to access and perform actions on MQTT topics. This helps prevent unauthorized access to critical resources.

Data Integrity:
MQTT does not inherently guarantee data integrity. Messages can be tampered with in transit. Implementing integrity checks, such as message signing or message digests, can help ensure the data's integrity and authenticity.

Message Replay Attacks:
Insecure MQTT implementations may be susceptible to message replay attacks, where an attacker captures and resends MQTT messages to achieve malicious objectives. Implementing message freshness checks or nonces can help mitigate this risk.

Secure Credentials Handling:
The script uses hardcoded credentials for MQTT connection. In a production environment, credentials should be stored securely, and best practices for credential management should be followed to prevent unauthorized access to the broker.

Client Identity Verification:
The script lacks a comprehensive way to verify the identity of MQTT clients. In a real-world scenario, clients should use client certificates or other methods to prove their identity to the broker.

Monitoring and Intrusion Detection:
Implementing continuous monitoring and intrusion detection mechanisms can help identify and respond to suspicious or unauthorized activities on the MQTT broker.

Secure Broker Configuration:
MQTT brokers should be securely configured to minimize attack surfaces. This includes disabling unnecessary features, keeping software up to date, and following security best practices.

Denial of Service (DoS) Mitigation:
MQTT brokers can be vulnerable to DoS attacks. Implementing rate limiting, access controls, and other protective measures can help mitigate the impact of such attacks.

Security Patching and Updates:
Regularly updating and patching MQTT broker software and libraries is essential to address known vulnerabilities and maintain the security of the system.

Security Training and Awareness:
Users and administrators should receive training on MQTT security best practices and be aware of potential threats and risks associated with IoT and MQTT communication.
Addressing these cybersecurity concerns is crucial to ensure that MQTT-based authentication is robust and secure in real-world IoT applications, especially in scenarios where sensitive data or critical operations are involved.

## V. Conclusion

In the realm of the Internet of Things (IoT), where interconnected devices communicate seamlessly, security is of paramount importance. This report has explored the implementation of a basic MQTT-based authentication script, shedding light on its functionality, practicality, and its potential as a steppingstone for understanding MQTT and IoT security principles.

The script, while rudimentary, serves as a starting point for grasping the fundamentals of MQTT communication and message handling. It demonstrates the importance of secure authentication in ensuring that only authorized clients can access and participate in IoT operations.

However, it is essential to acknowledge that the script, as presented, is primarily an educational and illustrative tool. It lacks some of the critical security features and considerations necessary for real-world IoT applications. Security concerns such as message encryption, robust authentication, access control, and data integrity need to be thoroughly addressed to ensure the confidentiality, integrity, and availability of IoT data and operations.

This report has also highlighted the significance of proper user input, scalability, and optimization for MQTT-based applications. While the script allows for user input of topics, its potential is not fully harnessed, and there is room for further development in this aspect. Furthermore, real-world IoT applications demand a more robust and scalable approach, with error handling and support for multiple clients.

Looking forward, the script's extensibility presents opportunities for more advanced use cases, such as home automation, industrial control, and telemetry. By building upon the foundational concepts introduced in the script, developers can create secure and feature rich IoT solutions that meet the demands of modern IoT ecosystems.

In summary, the MQTT-based authentication script is an entry point into the world of secure IoT communication. It serves as an educational tool, providing insights into MQTT principles and the importance of authentication in IoT security. To harness the full potential of MQTT and ensure the security of IoT applications, developers must build upon this foundation by implementing robust security practices, scalability, and advanced features. The script, while basic, represents a critical first step in the journey towards secure and efficient IoT solutions.

## References

1. Ltd, E.T.C. and Ltd, E.T.C. (n.d.). How to use MQTT in Python (Paho). [online] www.emqx.com. Available at: https://www.emqx.com/en/blog/how-to-use-mqtt-in-python.