Paper Title* (use style: paper title)

line 1: 2nd Given Name Surname
line 2: *dept. name of organization*
(of Affiliation)
line 3: *name of organization*
(of Affiliation)
line 4: City, Country
line 5: email address or ORCID

Abstract— **This report presents the development and implementation of a network monitoring system integrating Snort, a network intrusion detection system (NIDS), with an MQTT (Message Queuing Telemetry Transport) broker. The system comprises two key components: a publishing client and a subscribing client. The publishing client is designed to generate and send a predefined message to a private MQTT topic whenever Snort detects the HTTP protocol in network traffic. This process, however, is simulated for the scope of this project. The subscribing client listens to messages on public MQTT topics and displays them, showcasing the flow of information within the MQTT framework. Developed as a model based on lab 9c, this project aims to demonstrate the basic functionalities of MQTT in network event monitoring and reporting, focusing on the integration of external network monitoring tools with MQTT messaging systems. The report details the implementation, testing, and potential future enhancements of this system, emphasizing its role in educational and demonstrative settings.**

Keywords—MQTT, Snort, HTTP

I. Introduction (*Heading 1*)

In the evolving landscape of network security and monitoring, the integration of various tools and protocols plays a crucial role in efficiently managing and responding to network events. This report delves into a project that exemplifies this integration by combining Snort, a renowned network intrusion detection system (NIDS), with the Message Queuing Telemetry Transport (MQTT) protocol, a lightweight messaging system widely used in IoT and network applications.

MQTT facilitates efficient and reliable message transmission, making it an ideal choice for real-time network monitoring and alert systems. Snort, on the other hand, is a powerful tool for network traffic analysis, capable of detecting a wide range of security threats, including protocol-specific events. The project's objective was to create a system where Snort, upon detecting an HTTP protocol in the network traffic, triggers an MQTT client to publish a message to a specific topic. This setup serves as a basic yet effective model for integrating network monitoring tools with message brokers for real-time alerting and data dissemination.

The implementation comprises two MQTT clients developed using Python and the Paho MQTT library. The first client acts as a publisher, sending messages to a private MQTT topic when an HTTP protocol detection event is simulated. The second client functions as a subscriber, listening to public MQTT topics and displaying received messages. This arrangement demonstrates the fundamental aspects of MQTT communication, including topic-based message filtering and client-to-broker interactions.

This report outlines the design, implementation, and testing of the system. It provides a comprehensive view of how MQTT can be leveraged in network security contexts, particularly in educational and demonstration scenarios. Through this project, we aim to showcase the practical application of MQTT in network monitoring and the potential of such integrations in enhancing real-time response capabilities in network security.

II. Methodology

The project methodology is centered around the development and functioning of two MQTT clients: the publisher and the subscriber. Both are implemented using Python and the Paho MQTT library. Here, I detail the approach taken for each.

MQTT Publisher Client

Initial Setup:

The publisher client begins by establishing a connection with the MQTT broker. This is achieved through the connect_mqtt function, which sets up the client ID, username, password, and broker address.
The on_connect callback function is used to confirm the successful connection to the MQTT broker.

Message Publishing Loop:

The core functionality of the publisher client is encapsulated in the publish function. Here, a loop is initiated to continuously generate and send messages.
In this implementation, the message is a static string indicating the detection of the HTTP protocol. This mimics the response from Snort in detecting a specific network protocol.
The messages are sent to a predefined private MQTT topic.

Transmission Confirmation:

Each attempt to publish a message is followed by a status check. The client confirms whether the message was successfully transmitted to the MQTT broker and logs the outcome.

MQTT Subscriber Client
Connection and Subscription:

Similar to the publisher client, the subscriber also initiates a connection to the MQTT broker using the connect_mqtt function.
Once connected, the client subscribes to a specified topic, in this case, a wildcard topic that listens to all public messages.
Message Reception and Display:

The subscribe function includes an on_message callback, which is triggered whenever a message is received on the subscribed topic.
This callback function decodes and prints the received message along with the topic information, effectively displaying all public messages.

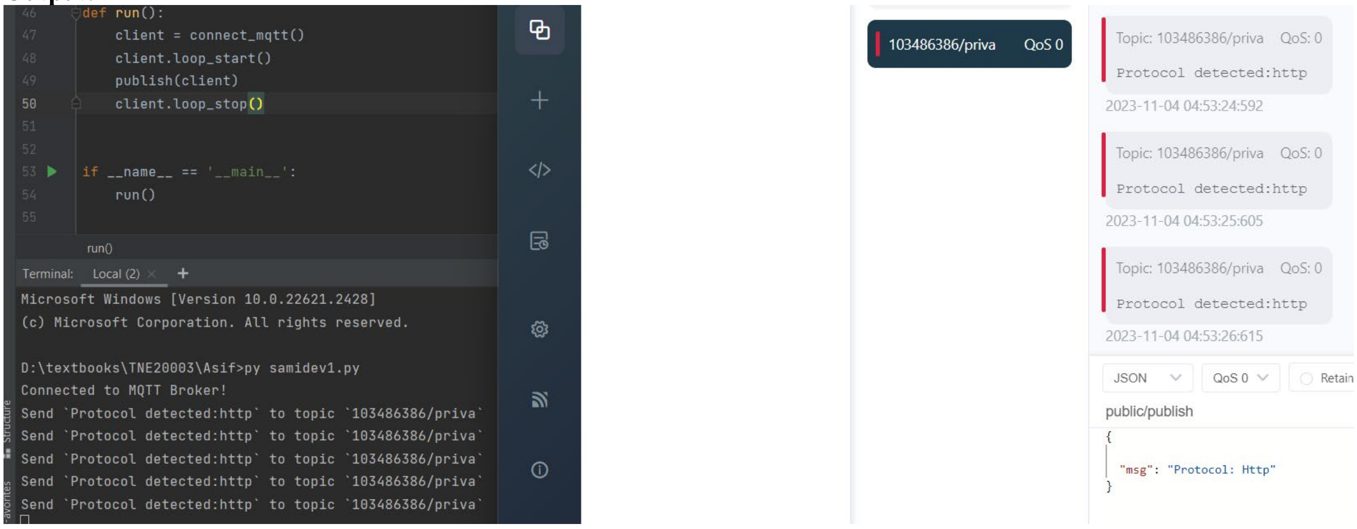
Integration and Testing
Integration of Publisher and Subscriber:

Although both clients operate independently, their integration is demonstrated through the MQTT broker. The publisher sends messages to a topic, and the subscriber listens and responds to those messages if they are on public topics.
For the purposes of this project, integration is limited as the publisher sends messages to a private topic, which the subscriber does not listen to.

Testing Methodology:

Testing involved running both clients simultaneously to ensure that the publisher could successfully send messages and that the subscriber could receive and display messages from public topics.
Additional tests included verifying the connection stability and the responsiveness of the message transmission and reception.

Output:



```
def run():
    client = connect_mqtt()
    client.loop_start()
    publish(client)
    client.loop_stop()


if __name__ == '__main__':
    run()
```

```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

D:\textbooks\TNE20003\Asif>py samidev1.py
Connected to MQTT Broker!
Send `Protocol detected:http` to topic `103486386/priva`
Send `Protocol detected:http` to topic `103486386/priva`
Send `Protocol detected:http` to topic `103486386/priva`
Send `Protocol detected:http` to topic `103486386/priva`
Send `Protocol detected:http` to topic `103486386/priva`
```

103486386/priva QoS 0

Topic: 103486386/priva QoS: 0
Protocol detected:http
2023-11-04 04:53:24:592

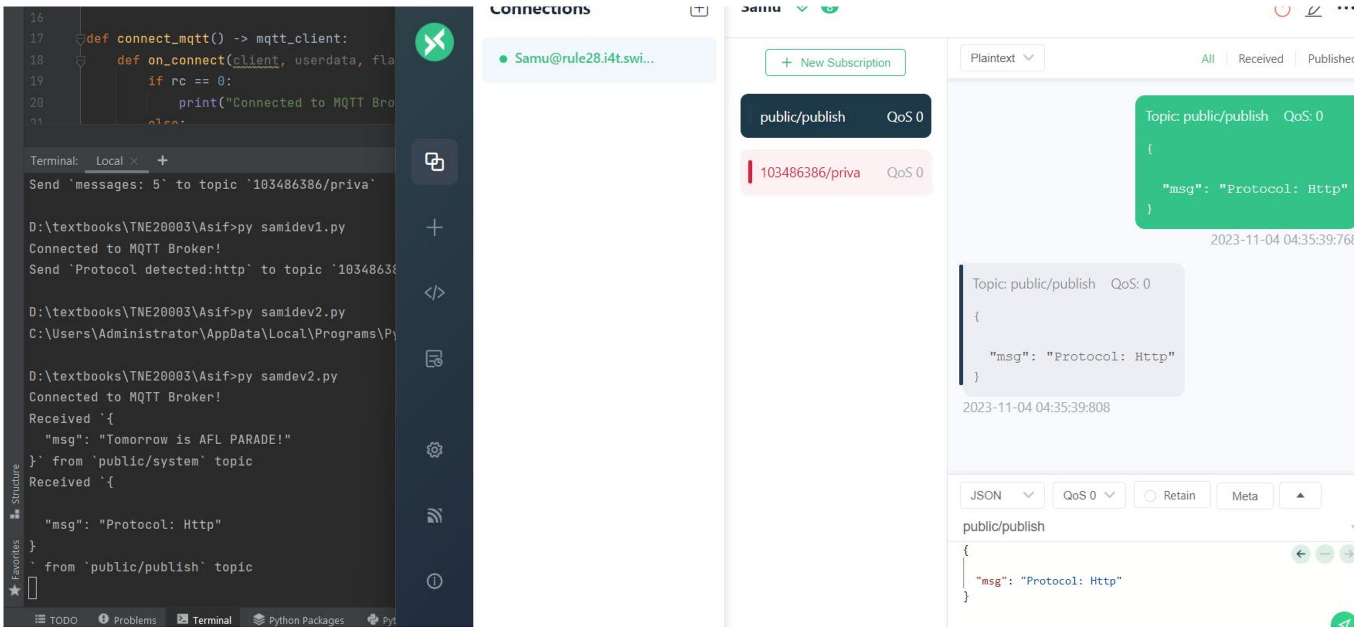
Topic: 103486386/priva QoS: 0
Protocol detected:http
2023-11-04 04:53:25:605

Topic: 103486386/priva QoS: 0
Protocol detected:http
2023-11-04 04:53:26:615

JSON QoS 0 Retain
public/publish

```
{
  "msg": "Protocol: Http"
}
```

Figure 1: Publish Output


```
def connect_mqtt() -> mqtt_client:
    def on_connect(client, userdata, fla
        if rc == 0:
            print("Connected to MQTT Bro
```

```
Send `messages: 5` to topic `103486386/priva`

D:\textbooks\TNE20003\Asif>py samidev1.py
Connected to MQTT Broker!
Send `Protocol detected:http` to topic `10348638

D:\textbooks\TNE20003\Asif>py samidev2.py
C:\Users\Administrator\AppData\Local\Programs\Py

D:\textbooks\TNE20003\Asif>py samdev2.py
Connected to MQTT Broker!
Received `{
  "msg": "Tomorrow is AFL PARADE!"
}` from `public/system` topic
Received `{

  "msg": "Protocol: Http"
}
` from `public/publish` topic
```

Samu@rule28.i4t.swi...

public/publish QoS 0

103486386/priva QoS 0

```
Topic: public/publish    QoS: 0
{

  "msg": "Protocol: Http"
}
2023-11-04 04:35:39:768

Topic: public/publish    QoS: 0
{

  "msg": "Protocol: Http"
}
2023-11-04 04:35:39:808
```

JSON QoS 0 Retain

public/publish

```
{
  "msg": "Protocol: Http"
}
```

Figure 2: Subscription Output

III. Discussion

This project successfully demonstrated the fundamental principles of MQTT in a network monitoring context. The MQTT publisher and subscriber clients, developed using Python and the Paho MQTT library, operated effectively within the defined parameters. The publisher client was able to simulate the role of a network intrusion detection system, like Snort, by sending a predefined message upon the detection of the HTTP protocol. The subscriber client, on the other hand, showcased its ability to listen to and display messages from public topics.

Insights from the MQTT Implementation

Topic-based Messaging Efficiency:
The project highlighted the efficiency of MQTT's topic-based messaging system. The clear distinction between private and public topics illustrated how MQTT can be tailored for specific communication needs in network monitoring.

Scalability and Flexibility:
MQTT's lightweight nature was evident, suggesting its scalability and flexibility in various network environments, from small-scale IoT setups to larger, more complex networks.

Limitations in Current Setup:
The current implementation, while effective for demonstration purposes, highlighted certain limitations. Notably, the publisher client was restricted to sending messages to a private topic, which the subscriber client did not access. This aspect limited the direct interaction between the two clients in the current configuration.

Integration with Snort
The simulated integration with Snort provided a conceptual understanding of how MQTT can enhance network intrusion detection systems. However, actual integration with Snort would offer more practical insights and a realistic demonstration of MQTT's role in real-time network monitoring and alerting.

Challenges and Learning Experiences
Challenge in Simulating Real-time Detection: Simulating the detection of the HTTP protocol instead of integrating with a live Snort setup posed a challenge in showcasing real-time network monitoring.

Solution and Learning: Despite this, the simulation approach served as an effective learning experience in understanding MQTT's capabilities and limitations. It also paved the way for considering future integrations with real-time detection systems.

Future Work and Potential Improvements
Real Snort Integration: Integrating with an actual Snort setup would provide a more realistic demonstration of MQTT in network monitoring, allowing for real-time detection and messaging.

Dynamic Message Generation: Enhancing the publisher to generate dynamic messages based on actual network events could significantly improve the system's applicability.

Expanding Subscriber Access: Modifying the subscriber client to listen to both private and public topics would enable a complete demonstration of the message flow within the MQTT system.

IV. Cybersecurity Concerns

Addressing cybersecurity concerns is crucial for a project involving MQTT and network monitoring tools like Snort. Here are some key cybersecurity concerns and considerations for this project:

1. Secure MQTT Communication
Encryption: Unencrypted MQTT messages are vulnerable to interception and eavesdropping. Implementing Transport Layer Security (TLS) encryption ensures that the data transmitted between MQTT clients, and the broker is secure.
Authentication and Authorization: Using strong authentication mechanisms (like TLS client certificates) and implementing proper authorization for MQTT topics prevents unauthorized access and ensures that clients only publish or subscribe to topics they are permitted to.

2. Secure Broker Configuration
Broker Vulnerabilities: An improperly configured MQTT broker can be a target for cyber-attacks. It's essential to secure the broker, keep it updated with the latest patches, and configure it to minimize vulnerabilities.
Access Control: Implementing robust access control mechanisms on the MQTT broker ensures that only authorized devices can connect and interact with the MQTT network.

3. Network Monitoring and Intrusion Detection
Snort Configuration: The effectiveness of Snort in detecting malicious activity depends on its configuration. Regularly updating Snort rulesets and tailoring them to the specific network environment are critical for effective detection.
False Positives and Negatives: Misconfigured Snort rules can lead to false positives or negatives, impacting the reliability of the system. Continuous tuning and analysis of Snort's performance are necessary.

4. Data Privacy and Integrity
Message Integrity: Ensuring the integrity of the messages sent between MQTT clients is crucial. Implementing message signing or checksums can help verify that messages have not been tampered with during transit.
Data Privacy: If the system handles sensitive information, it's important to consider data privacy laws and regulations. Ensuring that the system complies with relevant regulations (like GDPR or HIPAA) is essential.

5. Device and Client Security
Endpoint Security: The security of the devices running the MQTT clients is vital. These devices should be protected against malware and unauthorized access.
Software Security: Regularly updating and patching the software (including the operating system, MQTT client libraries, and other dependencies) minimizes vulnerabilities.

6. Denial of Service (DoS) Attacks
DoS Resilience: MQTT brokers and clients could be targets for DoS attacks. Implementing rate limiting, monitoring unusual traffic patterns, and having redundancy in place can help mitigate these risks.

7. Logging and Monitoring
Audit Trails: Keeping detailed logs of MQTT connections, disconnections, and message transmissions helps in auditing and forensic analysis in the event of a security incident.
Anomaly Detection: Implementing monitoring tools to detect anomalies in network traffic or MQTT usage patterns can help identify potential security breaches early.

V. Conclusion

This project has successfully demonstrated the integration of MQTT with a simulated network monitoring tool, providing valuable insights into the application of MQTT in the realm of network security and monitoring. Through the development of the MQTT publisher and subscriber clients, the project showcased the protocol's effectiveness in facilitating real-time message transmission and reception in a network monitoring context.

The use of Python and the Paho MQTT library proved to be robust and efficient for implementing MQTT clients. The project highlighted MQTT's lightweight nature, topic-based message filtering, and its potential scalability in various network environments. The simulation of Snort's integration, although limited, offered a conceptual understanding of how real-time network monitoring systems can be enhanced with MQTT for effective communication and alerting.

Despite facing challenges in simulating real-time network traffic monitoring and managing MQTT topic subscriptions, the project achieved its primary goals. It provided a foundational understanding of MQTT in network security applications and opened avenues for further exploration, particularly in integrating real network monitoring tools like Snort.

Future enhancements could involve actual integration with network intrusion detection systems, dynamic message generation based on real-time network events, and expanding the subscriber client to encompass a broader range of topics. These improvements would not only enhance the system's practicality but also provide deeper insights into the real-world application of MQTT in cybersecurity.

In conclusion, this project has laid the groundwork for further exploration and development in the intersection of MQTT and network security, demonstrating the protocol's versatility and its potential role in enhancing network monitoring and response capabilities.

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.