Design and Hardening of an MQTT Broker Server on AWS EC2 for Secure IoT Communication

Deo Triyanuar Putra[1], Muhammad Syirajuddin Suja'i[2*], Ahmad Wilda Yulianto[3]

1,3 Digital Telecommunication Network Study Program, Department of Electrical Engineering, State Polytechnic of Malang, 65141, Indonesia.
2 Telecommunication Engineering Study Program, Department of Electrical Engineering, State Polytechnic of Malang, 65141, Indonesia

[1] deoyanuar20@gmail.com, [2]syirajuddin@polinema.ac.id, [3]ahmadwildan@polinema.ac.id

Abstract— The rapid growth of Internet of Things (IoT) applications has increased the reliance on lightweight communication protocols such as Message Queuing Telemetry Transport (MQTT), while simultaneously raising security risks due to the exposure of broker servers to public networks. This study presents the design and security hardening of an MQTT Broker Server using EMQX deployed on Amazon Web Services (AWS) EC2 as a communication infrastructure for IoT devices. The research methodology includes system design, implementation of the MQTT Broker and Node-RED on AWS EC2, integration of IoT devices as MQTT clients, and the application of server hardening techniques. The hardening methods applied consist of operating system updates, root account restriction, firewall configuration using iptables and UFW, Secure Socket Layer (SSL/TLS) implementation through an Nginx reverse proxy, port access limitation, and system log monitoring. Security evaluation is conducted through penetration testing, including Information Gathering, Vulnerability Scanning, and simulated cyberattacks such as Denial of Service (DoS), Distributed Denial of Service (DDoS), Brute Force, and Remote Code Execution (Reverse Shell). The results show that before hardening, the server was highly vulnerable and could be taken down easily, while after hardening, all attack scenarios were successfully mitigated and system availability increased from 0% to 100%. These results demonstrate that server hardening significantly enhances the security and reliability of MQTT-based IoT communication on cloud infrastructure.

Keyword— AWS EC2, IoT Communication, MQTT Broker, Network Security, Server Hardening

I. INTRODUCTION

The rapid development of the Internet of Things (IoT) has led to a significant increase in the number of connected devices that rely on the internet for data exchange, monitoring, and remote control. IoT systems are widely applied in various sectors, including industry, smart cities, healthcare, and environmental monitoring, where reliable and efficient communication is essential [1], [2]. To support lightweight communication between constrained devices, the Message Queuing Telemetry Transport (MQTT) protocol has become one of the most commonly used solutions due to its low bandwidth consumption, publish–subscribe model, and suitability for resource-limited environments [3], [4].

Despite its advantages, MQTT introduces security challenges, particularly when broker servers are deployed on public cloud infrastructure and exposed to the internet. Several studies have reported that MQTT brokers are vulnerable to various cyber threats, including Denial of Service (DoS), Distributed Denial of Service (DDoS), brute-force authentication attacks, and remote code execution exploits [5], [6]. Attacks targeting MQTT availability can severely disrupt IoT services, as the broker functions as a central communication hub for all connected devices [7]. Furthermore, misconfigured brokers, lack of encryption, open ports, and weak access control mechanisms significantly increase the risk of unauthorized access and service disruption [8], [9].

Cloud platforms such as Amazon Web Services (AWS) EC2 are widely adopted for deploying MQTT brokers due to their scalability, flexibility, and global accessibility. However, cloud-based deployments do not inherently guarantee security, and improper configuration may expose critical services to public attacks [10], [11]. Previous research has shown that many MQTT implementations prioritize functionality and performance, while security aspects such as firewall configuration, secure authentication, encrypted communication, and attack mitigation mechanisms are often insufficiently addressed [12].

To mitigate these risks, server hardening is a crucial approach that aims to reduce system vulnerabilities by applying security best practices, including operating system updates, access control restriction, firewall configuration, secure communication using SSL/TLS, port limitation, and continuous log monitoring [13], [14]. Several studies have demonstrated that hardening techniques can significantly improve system resilience against common attack vectors; however, comprehensive experimental evaluations focusing on MQTT brokers deployed on cloud infrastructure remain limited [15].

Therefore, this research focuses on the design and security hardening of an MQTT Broker Server using EMQX deployed on AWS EC2 as an IoT communication infrastructure. The contribution of this study lies in the systematic implementation of multiple hardening techniques and their evaluation through penetration testing, including DoS, DDoS, brute-force, and reverse shell attacks. The results are expected to provide practical insights into securing cloud-based MQTT

*Corresponding author

communication systems and enhancing the reliability and availability of IoT services.

II. METHOD

This research was conducted through several systematic stages to evaluate the security and resilience of an MQTT Broker Server deployed on cloud infrastructure. The initial stage involved a literature review of scientific journals and technical references related to IoT communication, MQTT broker architecture, server hardening techniques, and common cyberattack scenarios. Based on this review, the system design phase was carried out by developing flowcharts and block diagrams that served as implementation guidelines. The implementation stage included configuring the EMQX MQTT Broker and Node-RED on Amazon Web Services (AWS) EC2, integrating IoT devices using NodeMCU with Tasmota firmware, and deploying Nginx and Cloudflare as reverse proxy services to enhance the security layer. Finally, penetration testing was performed to evaluate system resilience against various cyberattacks before and after the application of hardening methods.

A. System Description

The proposed system utilizes the MQTT protocol for communication between IoT microcontroller devices and the MQTT Broker, while the HTTP protocol is used for interaction between the broker and Node-RED. The MQTT Broker is deployed on an AWS EC2 instance and assigned a public IP address, enabling global accessibility. However, the exposure of the broker to public networks increases the risk of cyberattacks. Therefore, server hardening techniques are applied to mitigate security vulnerabilities. To evaluate the effectiveness of the implemented hardening methods, penetration testing is conducted using attack simulations such as Denial of Service (DoS), Distributed Denial of Service (DDoS), Brute Force attacks, and Reverse Shell exploitation.

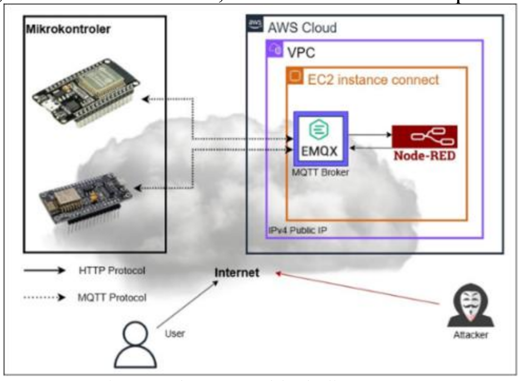

Figure 1 the system block diagram,

The system architecture is designed specifically to assess server resilience through security hardening. In this architecture, the microcontroller functions as an IoT node that transmits data via the MQTT protocol. AWS cloud services are used to build the infrastructure, employing a Virtual Private Cloud (VPC) to deploy EC2 instances as servers. The MQTT Broker operates as the backend system for device authentication and message management, while Node-RED acts as a centralized platform for monitoring and managing IoT communication. The entire system can be accessed remotely through a public IP address.

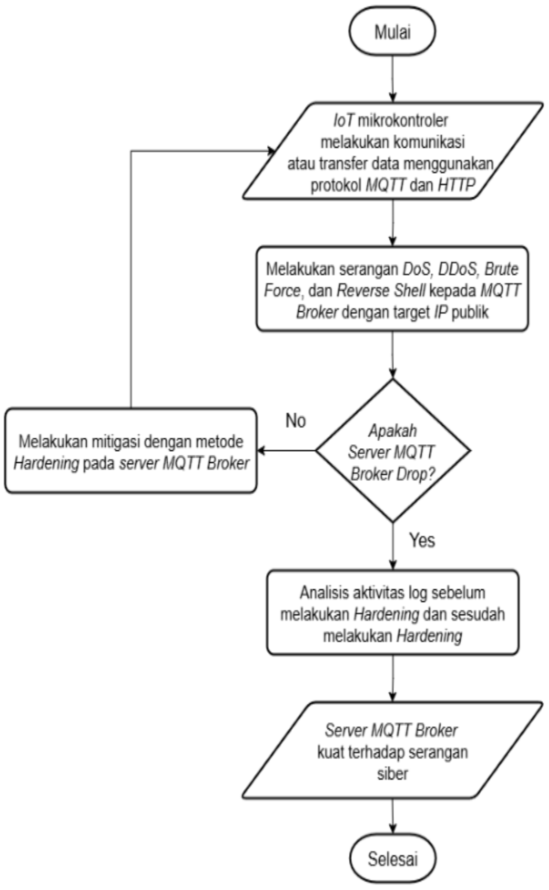
Mulai
IoT mikrokontroler melakukan komunikasi atau transfer data menggunakan protokol MQTT dan HTTP
Melakukan serangan DoS, DDoS, Brute Force, dan Reverse Shell kepada MQTT Broker dengan target IP publik
Apakah Server MQTT Broker Drop?
No
Melakukan mitigasi dengan metode Hardening pada server MQTT Broker
Yes
Analisis aktivitas log sebelum melakukan Hardening dan sesudah melakukan Hardening
Server MQTT Broker kuat terhadap serangan siber
Selesai

Figure 2 Research Process Flowchart

The system architecture design used for research purposes, specifically as a means of testing server resilience through the implementation of hardening methods. In this design, the microcontroller module serves as an IoT node responsible for transmitting data over the internet using the MQTT protocol. AWS cloud services are utilized to build the infrastructure, leveraging a Virtual Private Cloud (VPC) to deploy EC2 instances as servers. This server is also equipped with an MQTT Broker as the backend system for managing communication and authentication of connected devices. Additionally, Node-RED is used as a centralized control platform enabling efficient monitoring and management of IoT communication. The entire system can be accessed from various locations via a public IP address, allowing users to access it from anywhere with an internet connection.

B. Implementation of EC2 Instances on AWS

The EC2 instance deployment process begins with assigning an instance name as an identity label, followed by selecting Ubuntu Server 24.04 (64-bit) as the operating system. The t2 instance type is chosen, providing 1 CPU core and 1 GB of

RAM, which is suitable for lightweight IoT communication services. Secure Shell (SSH) access is configured using a key pair with a .ppk extension. Subsequently, the Virtual Private Cloud (VPC), storage allocation, and Security Group rules are configured. The Security Group functions as a basic firewall that regulates inbound and outbound traffic. Before launching the instance, all configurations are verified to minimize operational and security risks. The EC2 instance is then launched once all parameters are confirmed.

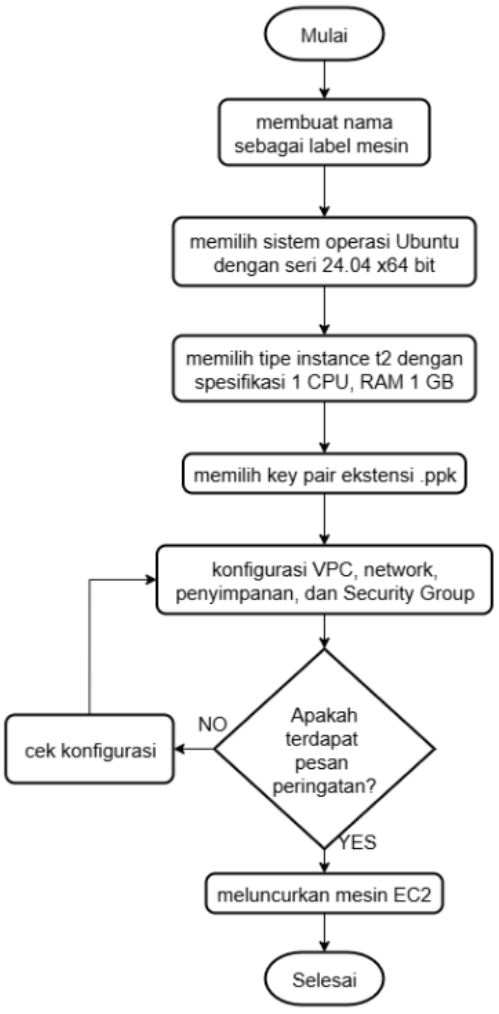
Mulai
membuat nama sebagai label mesin
memilih sistem operasi Ubuntu dengan seri 24.04 x64 bit
memilih tipe instance t2 dengan spesifikasi 1 CPU, RAM 1 GB
memilih key pair ekstensi .ppk
konfigurasi VPC, network, penyimpanan, dan Security Group
Apakah terdapat pesan peringatan?
NO
cek konfigurasi
YES
meluncurkan mesin EC2
Selesai

Figure 3 Launch Instance EC2 Flowchart

C. *Implementing an MQTT Broker on AWS EC2*

The design and implementation of the MQTT server are visualized using a flowchart to describe the backend authentication and communication process. Docker containers are used to provide an isolated runtime environment. After updating the Ubuntu system packages, the official EMQX repository is added, and the MQTT Broker is installed using the package manager. Once installation is complete, the EMQX service is activated and configured to accept connections on the default MQTT ports, enabling it to function as the central communication hub for IoT devices.

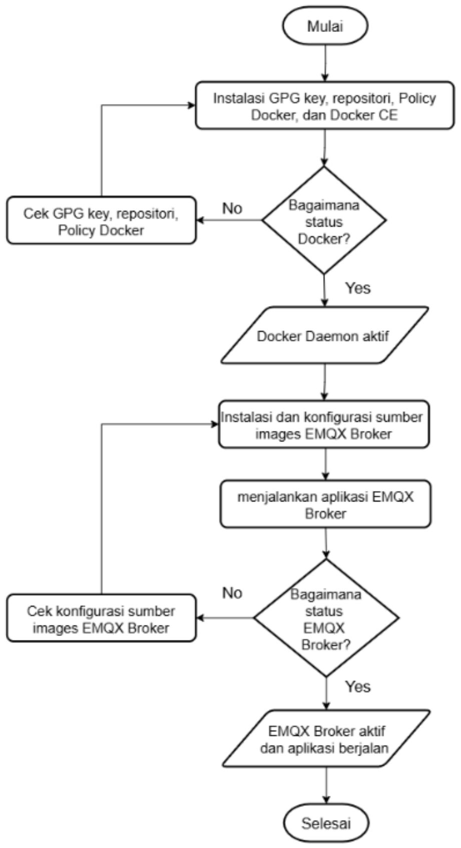
Mulai
Instalasi GPG key, repositori, Policy Docker, dan Docker CE
Bagaimana status Docker?
No
Cek GPG key, repositori, Policy Docker
Yes
Docker Daemon aktif
Instalasi dan konfigurasi sumber images EMQX Broker
menjalankan aplikasi EMQX Broker
Bagaimana status EMQX Broker?
No
Cek konfigurasi sumber images EMQX Broker
Yes
EMQX Broker aktif dan aplikasi berjalan
Selesai

Figure 4 EMQX Installation Flowchart

D. *Implementing an Node-RED on AWS EC2*

Node-RED is implemented as a centralized application platform using the MQTT protocol. The installation process is illustrated in a flowchart. Node-RED is deployed using Docker to simplify dependency management and ensure environment consistency. Prior to installation, Node.js and Node Package Manager (npm) are installed as core dependencies. After successful configuration, Node-RED is deployed using the official Docker image and accessed via a web browser through the default port 1880.

E. *Integrating Node-RED as an IoT MQTT Client*

The integration between Node-RED and EMQX Broker is done by utilizing the MQTT In and MQTT Out nodes in Node-RED, where the MQTT In node functions as a subscriber to receive messages from the broker, while the MQTT Out node acts as a publisher that sends messages to the broker. The process begins with configuring the MQTT In node, which involves entering the EMQX Broker address or URL, setting up MQTT client authentication such as username and password if required, and specifying the topics to subscribe to. After the subscriber configuration is complete, the MQTT Out node is configured for data publishing, including entering the broker address, authentication, and specifying the topics to send. Once all configurations are complete, the deploy flow process is run to activate the integration. If the deploy fails, the configuration is rechecked until the system runs normally. If the deploy is successful and communication between Node-RED and the EMQX Broker runs smoothly, the integration process is deemed successful.

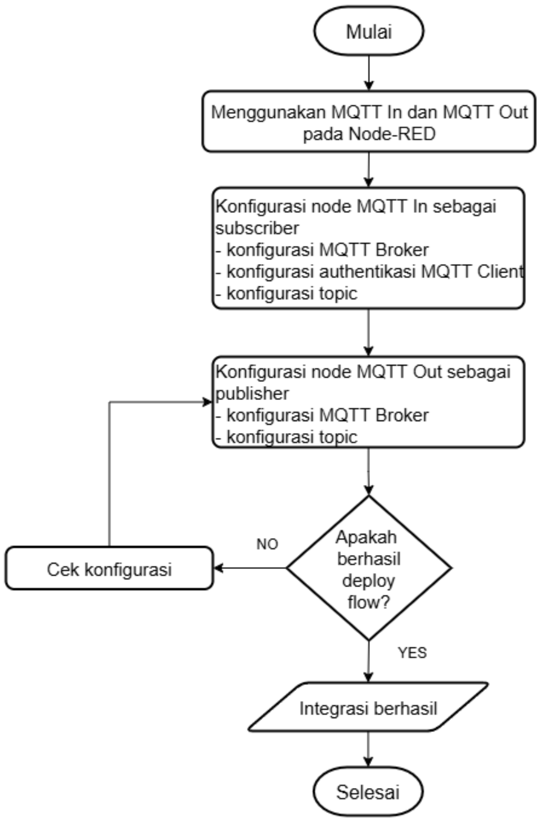
Mulai
Menggunakan MQTT In dan MQTT Out pada Node-RED
Konfigurasi node MQTT In sebagai subscriber
- konfigurasi MQTT Broker
- konfigurasi authentikasi MQTT Client
- konfigurasi topic
Konfigurasi node MQTT Out sebagai publisher
- konfigurasi MQTT Broker
- konfigurasi topic
Apakah berhasil deploy flow?
NO
Cek konfigurasi
YES
Integrasi berhasil
Selesai

Figure 5 Node-RED Integration Flowchart

F. *Integrating Tasmota into IoT Microcontrollers as an MQTT Client*

The Tasmota firmware installation process is carried out through the official web interface available at https://tasmota.github.io/install/, which allows direct installation via a browser without the need for additional software. Users simply follow the provided instructions to automatically flash the firmware to the microcontroller device. Before installation, it is important to ensure that the USB to Serial Converter driver is properly installed on the computer or laptop being used to ensure a smooth flashing process. After the firmware has been successfully flashed, the device will enter the initial configuration mode, where users need to set the SSID and Wi-Fi password so that the device can connect to the local network. Next, hardware configuration is performed by selecting the appropriate microcontroller module type and setting the connection to the MQTT Broker, including entering the server address, port, username, and password if needed, so that the device is ready to communicate within the designed IoT system.

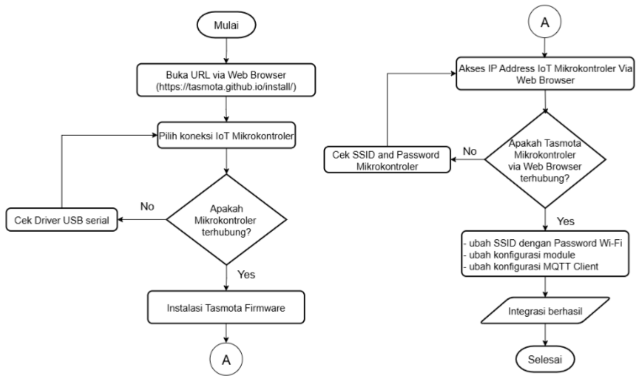
Mulai
Buka URL via Web Browser (https://tasmota.github.io/install/)
Pilih koneksi IoT Mikrokontroler
Apakah Mikrokontroler terhubung?
No
Cek Driver USB serial
Yes
Instalasi Tasmota Firmware
A
A
Akses IP Address IoT Mikrokontroler Via Web Browser
Apakah Tasmota Mikrokontroler via Web Browser terhubung?
No
Cek SSID and Password Mikrokontroler
Yes
- ubah SSID dengan Password Wi-Fi
- ubah konfigurasi module
- ubah konfigurasi MQTT Client
Integrasi berhasil
Selesai

Figure 6 Node Microcontroller Integration Flowchart

G. *Designing Nginx and Cloudflare as a Reverse Proxy Web Server*

To support web-based application access and integration between services, Nginx was installed and configured as a reverse proxy on the server. The process began with a system update to ensure that all dependencies were up to date, followed by the installation of the Nginx package through the operating system's built-in package manager. Next, a dedicated directory was created for the application to be accessed via Nginx, followed by the creation of a virtual host configuration file pointing to the domain being used and specifying the application directory path. To ensure the configuration is recognized by Nginx, a symbolic link was created from the configuration directory to the sites-enabled directory. After the configuration is complete, syntax validation is performed using the nginx -t command to ensure there are no errors, then the Nginx service is restarted so that the new configuration is applied and the reverse proxy is active. The next step is to point the domain using the Cloudflare service, starting with creating an account and adding the domain, then changing the domain's nameservers to Cloudflare's nameservers according to the provided instructions. After the propagation process is complete, subdomains are created as needed and pointed to the public IP of the server running the Nginx service. The DNS configuration is saved to apply the changes, and with this integration, service access can be performed more securely and optimally, as Cloudflare also provides additional features such as caching, DDoS protection, and SSL/TLS management.

H. *Implementation of Server Hardening Methods*

Table 1
Hardening Method

Method	Description
Keep system Up-to-Date	The operating system performs regular updates.
Mengunci akun	Disables access to root accounts.
Konfigurasi *UFW* dan *Iptables*	Firewall configuration.
Konfigurasi *SSL Nginx Reverse Proxy*	HTTP to HTTPS migration.
Port blocking	Port access restrictions.
Monitoring log	Log activity analysis.

Regular operating system updates (keep system up-to-date) ensure that servers are always protected from various security vulnerabilities that can be exploited. Locking root accounts also reduces the risk of unauthorized access by limiting full control to authorized parties only. Meanwhile, firewall configuration through UFW and iptables allows incoming and outgoing traffic to be restricted as needed, and the implementation of SSL through Nginx Reverse Proxy provides communication encryption to protect the integrity of the data being transmitted. Port restrictions also successfully minimize entry points for unauthorized parties, and the implementation of a log monitoring system enables early detection and analysis of suspicious activity. Overall, the implementation of these hardening methods has proven effective in strengthening system defenses against various forms of cyber threats, making the IoT communication infrastructure more reliable, secure, and ready for use in production scale.

During the testing phase, various methods were used to evaluate the level of security before and after the implementation of hardening. This testing includes Information Gathering (using OSINT Whois, DNS Enumeration, and WhatWeb), Vulnerability Scanning (with SmokeWAF, WafW00f, and Nmap), and simulations of various cyber attacks (DoS, DDoS, Brute Force, and Reverse Shell). The test results are analyzed to measure the effectiveness of the hardening methods applied, including IP restrictions, firewall configuration with iptables, SSL implementation with Nginx, and security log monitoring.

This series of methods enables research to evaluate the resilience and effectiveness of hardening MQTT Broker servers, in order to ensure more reliable and secure communication for IoT devices against various forms of cyber attacks.

III. RESULTS AND DISCUSSION

The researchers explained the results of security testing on the MQTT Broker Server before the hardening method was implemented, using various penetration testing techniques such as Information Gathering, Vulnerability Scanning, and cyber attack simulations including DoS, DDoS, Brute Force Attack, and Reverse Shell Attack. Based on the testing data, it was found that a significant amount of sensitive information from the server was exposed during the Information Gathering process. From the domain projectenginesecurity.my.id, the researchers successfully identified several vulnerabilities, including hosting information detected through OSINT Whois tools and domain registration data. Additionally, DNS Enumeration and WhatWeb tools (for banner grabbing) revealed exposed public IP addresses along with application details, operating systems, and web servers complete with their versions, which could serve as entry points for attacks. Validation of this information was conducted through scanning using tools such as SmokeWAF, WafW00f, and NMAP, which indicated that the server was not protected by a firewall, making it highly vulnerable to external attacks. The SmokeWAF and WafW00f tools did not detect any WAF protection on the domain, while the NMAP scan results showed a number of open ports such as port 22 (SSH), port 80 (HTTP), and port 1883 (MQTT). In particular, the open port 80 without encryption is highly vulnerable to attacks because it is not protected by default, making this a strong basis for researchers to proceed to the next stage of testing cyber attacks on the MQTT server.

Table 2
Attack Validation

Penestration Testing	Validation of attack success	Description
DoS *Tools Slowloris*	✓	-
DoS Tools *AnonymousDoser*	✓	-
DoS Tools *SlowHTTPTest*	✓	-
DDoS Tools HOIC	✓	-
Brute-Force Tools Hydra	x	The server uses public-key by default.
Reverse Shell Tools RCE	✓	-

The results of exploiting the MQTT Broker Server before hardening revealed a number of important findings that indicated the system's weak defense against various types of cyber attacks. DoS (Denial of Service) attack testing using Slowloris, AnonymousDoser, and SlowHTTPTest tools showed consistent results, namely that the MQTT service became unresponsive due to slow and continuous HTTP connection saturation. DDoS (Distributed Denial of Service) attacks using the HOIC tool were also successful, where traffic pressure from multiple hosts caused the service to become unavailable, even causing the server to freeze and temporarily shut down. Meanwhile, brute force testing using Hydra tools failed to penetrate SSH authentication, indicating that public-key-based authentication configurations are effective in preventing illegal login attempts. However, in Reverse Shell testing exploiting RCE (Remote Code Execution) vulnerabilities, the attack was successful and the attacker was able to gain remote shell access to the target system. This success indicates weak system configuration, lack of access rights separation, and the absence of filtering and protection against malicious command injection. Overall, these findings show that before hardening, the MQTT server had numerous security vulnerabilities, open ports without adequate control, and minimal defense systems, making it highly susceptible to exploitation.

Security testing conducted before and after implementing hardening methods on the MQTT Broker Server on AWS EC2 showed significant differences in the system's resilience to various forms of cyber attacks. The testing consisted of simulations of DoS, DDoS, Brute Force, and Reverse Shell attacks, with detailed testing parameters as measures of attack success or failure.

Table 3
Slowloris DoS Attack Test Parameters

Hardening Method	Parameters	Conditions Before Hardening	Effects After Hardening
Firewall Iptables	CPU	>8%	< 2%
	Network in	466k bytes	< 100k bytes
	Network out	< 1.69 M bytes	< 260k bytes
	Packet in	>5.07k	< 1k
	Packet out	>4.62k	< 700
SSL Nginx Reverse Proxy	CPU	>8%	< 2.5%
	Network in	466k bytes	< 370k bytes
	Network out	< 1.69 M bytes	<420k bytes
	Packet in	>5.07k	< 3.5k
	Packet out	>4.62k	< 2.5k
Firewall Iptables and SSL Nginx Reverse Proxy	CPU	>8%	< 2%
	Network in	466k bytes	< 100k bytes
	Network out	< 1.69 M bytes	< 100k bytes
	Packet in	>5.07k	>5.07
	Packet out	>4.62k	>4.62

Table 4
AnonymousDoser DoS Attack Test Parameters

Hardening Method	Parameters	Conditions Before Hardening	Effects After Hardening
Firewall Iptables	CPU	>7%	< 10.9%
	Network in	>223k bytes	< 1.52M bytes
	Network out	> 432k bytes	< 1M bytes
	Packet in	> 2.44k	< 26k
	Packet out	>2.38k	< 500
SSL Nginx Reverse Proxy	CPU	>7%	< 2.3%
	Network in	>223k bytes	< 200k bytes
	Network out	> 432k bytes	<567k bytes
	Packet in	> 2.44k	< 3k
	Packet out	>2.38k	< 2.6k
Firewall Iptables and SSL Nginx Reverse Proxy	CPU	>7%	< 2.3%
	Network in	>223k bytes	< 71k bytes
	Network out	> 432k bytes	<213k bytes
	Packet in	> 2.44k	< 250
	Packet out	>2.38k	< 250

Table 5
SlowHTTPTest DoS Attack Test Parameters

Hardening Method	Parameters	Conditions Before Hardening	Effects After Hardening
Firewall Iptables	CPU	>8%	< 4%
	Network in	>926k bytes	< 3.79M bytes
	Network out	> 1.11M bytes	< 1.42M bytes
	Packet in	< 7.49k	< 65k
	Packet out	<7.39k	< 584
SSL Nginx Reverse Proxy	CPU	>8%	< 6%
	Network in	>926k bytes	< 8.88M bytes
	Network out	> 1.11M bytes	<5.12M bytes
	Packet in	< 7.49k	< 28.9k
	Packet out	<7.39k	< 26.2k
Firewall Iptables and SSL Nginx Reverse Proxy	CPU	>8%	< 3%
	Network in	>926k bytes	< 3M bytes
	Network out	> 1.11M bytes	< 2M bytes
	Packet in	< 7.49k	>10.5k
	Packet out	<7.39k	>9.54

Table 6
HOIC DDoS Attack Test Parameters

Hardening Method	Parameters	Conditions Before Hardening	Effects After Hardening
Firewall Iptables	CPU	>19.5%	< 2%
	Network in	>10.1M bytes	< 100k bytes
	Network out	> 7.57M bytes	< 260k bytes
	Packet in	> 83.6k	< 1k
	Packet out	> 75.4k	< 700
SSL Nginx Reverse Proxy	CPU	>19.5%	< 2.5%
	Network in	>10.1M bytes	< 370k bytes
	Network out	> 7.57M bytes	<420k bytes
	Packet in	> 83.6k	< 3.5k
	Packet out	> 75.4k	< 2.5k
Firewall Iptables and SSL Nginx Reverse Proxy	CPU	>19.5%	< 2%
	Network in	>10.1M bytes	< 100k bytes
	Network out	> 7.57M bytes	< 100k bytes
	Packet in	> 83.6k	>5.07
	Packet out	> 75.4k	>4.62

Before the hardening method was implemented, the MQTT Broker Server was vulnerable and could be easily taken down by various attack patterns. DoS testing with Slowloris, AnonymousDoser, and SlowHTTPTest showed that the server could not handle high numbers of connections and slow connection patterns, resulting in the server going down and connections not being served (Service Availability: 0%). DDoS attacks using HOIC also successfully made the server unreachable, while Brute Force methods using Hydra and Reverse Shell with RCE exposed critical entry points, enabling attackers to gain full control over the system.

After implementing hardening methods, including the deployment of a firewall with iptables, SSL Nginx Reverse Proxy configuration, port restrictions, and the implementation of AWS Security Group rules, the system was able to withstand various attack patterns. The test parameters used included the number of active connections, the number of failed connections, Service Availability status, and the number of successful requests.

These results confirm that the implemented hardening methods can close various security gaps that could initially be exploited by attackers. The iptables firewall successfully blocked suspicious connection patterns from various IP addresses, while the implementation of SSL Nginx Reverse Proxy successfully reduced the risk of unauthorized connections. Overall, the implementation of hardening methods can increase the system's availability and reliability from 0% to 100% during simulations of various cyber attack patterns, proving the effectiveness of the security measures implemented.

IV. CONCLUSION

This study successfully designed, implemented, and tested hardening methods on MQTT Broker servers hosted on AWS EC2 for IoT device communication. The test results showed that the implementation of hardening methods, including firewall configuration with iptables, SSL Nginx Reverse Proxy implementation, port restrictions, and log monitoring, can significantly improve service security and availability. Cyberattacks such as DoS (Slowloris, AnonymousDoser, and SlowHTTPTest), DDoS (HOIC), Brute Force (Hydra), and Reverse Shell (RCE), which previously could bring down the system, were successfully mitigated after the hardening implementation, achieving a success rate of up to 100%.

This study proves that the application of appropriate hardening methods can make IoT communication infrastructure more resilient and reliable in facing various cyber attack patterns. With this application, IoT device communication can be more secure, stable, and protected from risks that can cause downtime or system takeover by unauthorized parties. The results of this study can serve as a reference for system developers and cybersecurity practitioners in building and securing IoT communication systems that utilize cloud-based MQTT Broker servers.

ACKNOWLEDGEMENTS

The author would like to express his deepest gratitude to all parties who have contributed to this research, especially the supervising lecturers and teaching staff at Politeknik Negeri Malang who have provided valuable guidance and input. Thanks also go to Amazon Web Services (AWS) for providing the cloud infrastructure for this research, as well as all colleagues and stakeholders who supported the data collection process, system implementation, and security testing. The support and collaboration from various parties have made this research possible and are expected to contribute positively to the development of safer and more reliable IoT communication technology.

REFERENCES

[1] Calabretta, M., Pecori, R., Vecchio, M., and Veltri, L. (2018). MQTT Auth: A token based solution to endow MQTT with authentication and authorization capabilities. Journal of Communications Software and Systems, 14(4). https://doi.org/10.24138/jcomss.v14i4.604 jcoms.fesb.unist.hr

[2] Lohachab, A. (2019). ECC based inter device authentication and authorization scheme using MQTT for IoT networks. Journal of Information Security and Applications, 46, pp. 1 to 12. https://doi.org/10.1016/j.jisa.2019.02.005 ScienceDirect+1

[3] Vaccari, I., Aiello, M., and Mongelli, M. (2020). SlowITe, a novel denial of service attack affecting MQTT. Sensors, 20(10), 2932. https://doi.org/10.3390/s20102932 ScienceDirect+1

[4] Elemam, E., Bahaa Eldin, A. M., Shaker, N. H., and Sobh, M. (2020). Formal verification for a PMQTT protocol. Egyptian Informatics Journal, 21(3), pp. 169 to 182. https://doi.org/10.1016/j.eij.2020.01.001 DOAJ

[5] Seoane, V., Garcia Rubio, C., Almenares, F., and Campo, C. (2021). Performance evaluation of CoAP and MQTT with security support for IoT environments. Computer Networks, 197, 108338. https://doi.org/10.1016/j.comnet.2021.108338 ScienceDirect

[6] Silva, D., Carvalho, L., Soares, J., et al. (2021). A performance analysis of Internet of Things networking protocols: Evaluating MQTT, CoAP, OPC UA. Applied Sciences, 11(11), 4879. https://doi.org/10.3390/app11114879 MDPI+1

[7] Chen, Z., Wang, Y., Ning, H., and Dai, H. N. (2021). A survey on security in IoT cloud computing. ACM Computing Surveys, 54(8). https://doi.org/10.1145/3447625 SciSpace

[8] Roldán Gómez, J., Carrillo Mondéjar, J., Castelo Gómez, J. M., and Ruiz Villafranca, S. (2022). Security analysis of the MQTT SN protocol for the Internet of Things. Applied Sciences, 12(21), 10991. https://doi.org/10.3390/app122110991 MDPI+1

[9] Dewanta, F., and colleagues. (2022). A study of secure communication scheme in MQTT: TLS vs AES in IoT networks. Jurnal Infotel, 14(4). Jurnal Ittelkom PWT

[10] Shilpa, V., Vidya, A., and Pattar, S. (2022). MQTT based secure transport layer communication for mutual authentication in IoT network. Global Transitions Proceedings, 3(12). https://doi.org/10.1016/j.gltp.2022.04.015 ResearchGate+1

[11] Surianarayanan, C. (2023). Integration of the Internet of Things and cloud: Security challenges and solutions. International Journal of Cloud Applications and Computing, 13(1), pp. 1 to 30. https://doi.org/10.4018/IJCAC.325624 IGI Global+1

[12] Kosaka, K., Noda, Y., Yokotani, T., and Ishibashi, K. (2023). Implementation and evaluation of the control mechanism among distributed MQTT brokers. IEEE Access, 11, pp. 134211 to 134216. https://doi.org/10.1109/ACCESS.2023.3335273 UMSIDA Preprints

[13] Kashyap, M., Dev, A. K., and Sharma, V. (2024). Implementation and analysis of EMQX broker for MQTT protocol in the Internet of Things. e Prime: Advances in Electrical Engineering, Electronics and Energy, 10, 100846. https://doi.org/10.1016/j.prime.2024.100846 ScienceDirect+1
[14] Saha, N., Paul, P., Ji, K., and Harik, R. (2024). Performance evaluation framework of MQTT client libraries for IoT applications in manufacturing. Manufacturing Letters, 41, pp. 1237 to 1245. https://doi.org/10.1016/j.mfglet.2024.09.150 ScienceDirect
[15] Bangare, P. S., and Patil, K. P. (2024). Enhancing MQTT security for internet of things: Lightweight two way authorization and authentication with advanced security measures. Measurement: Sensors, 101212. https://doi.org/10.1016/j.measen.2024.101212