IOT-Based Monitoring System with Security Testing

MARTIN VELANGINI VASANTHA
Dept. Of Computer Science & Systems Engineering Andhra University Visakhapatnam, India

***Abstract*—The rapid growth of the Internet of Things (IoT) has brought significant benefits but also introduced serious security challenges. Since many IoT devices are resource-constrained and yet process sensitive information, they are highly vulnerable to cyberattacks. This project focuses on designing a secure IoT framework using NodeMCU ESP and Raspberry Pi, where sensor data is collected and stored in a MariaDB database. Communication between devices is handled through the lightweight MQTT protocol, ensuring efficient data transfer. To evaluate the robustness of the system, penetration testing was carried out using Kali Linux, where simulated cyberattacks such as Cross-Site Request Forgery (CSRF), Command Injection attack and Denial of Service were performed. Where fake data are sent to make the device perform unwanted actions. These tests helped us identify areas where the system was susceptible. Based on what we found, we suggested improvements such as better input filtering, more secure firmware management and strong protection for data being sent over the network.**

***Index Terms*—NodeMCU, Soil Moisture, Raspberry Pi, MQTT Protocol.**

1. INTRODUCTION

The Internet of Things (IoT) is transforming the way physical objects interact with the digital world. It enables devices such as sensors, household appliances, and industrial machines to collect and exchange information without human intervention. IoT applications are growing rapidly in areas like remote healthcare monitoring, smart traffic management, home automation, and industrial efficiency. In recent years, the development of advanced technologies has made sensors and microcontrollers more compact, affordable, and powerful, which has accelerated the widespread adoption of IoT solutions. Abdullah et al. [1], designed a real-time energy monitoring system using NodeMCU ESP and MySQL, showcasing the reliability of ESP-based data transmission and logging. Bakry et al. [2], investigated how Raspberry Pi and Kali Linux can be used to simulate real-world security attacks on IoT devices. Their worked on how easily attackers can exploit unprotected networks. In a similar effort, Gunawan et al. [3], outlined the process of setting up a security audit lab using Kali Linux tools such as Nmap and Wireshark, enabling in-depth vulnerability scanning of IoT infrastructures. Khan et al. [4], developed a hands-on testing approach customized for IoT systems, enabling early detection of security flaws within network setups. Their model supports proactive defense by identifying and addressing vulnerabilities before they can be misused. To protect IoT environments from external threats, focusing on backend issues Donta et al. [5], provide a detailed survey of recent developments in IoT application layer protocols such as MQTT, CoAP, AMQP, and HTTP. They compare these protocols based on factors like reliability, data size, latency, and power consumption.Lazzaro et al. [6], tackled the issue of replay attacks with a testing tool that automates vulnerability detection by replaying captured traffic to observe system behavior under attack. On the defense side, Straussenburg et al. [7], looked at how offices with shared desks can use IoT sensors to track which desks are actually being used. By collecting real-time data, companies can better understand how their workspaces are used and make smarter decisions about space and resources. The study also mentions some challenges like protecting people's privacy and making sure the sensors work accurately. Wang et al. [8], delivered an efficient intrusion detection method for IoT networks that combines deep learning with dynamic quantization. Their approach focuses on reducing the computational load, making it suitable for resource-limited IoT devices.

The OWASP Top 10 is a list of the most serious security issues that affect modern web applications. One of the top risks is Broken Access Control**:** Occurs when users are able to view or perform actions beyond their intended permissions, such as accessing admin panels or confidential records. Cryptographic

Failures**:** Sensitive information is at risk if proper encryption is not applied or if weak algorithms are used. Injection Attacks**:** Malicious inputs, often through forms or queries, can trick an application into executing harmful commands. Insecure Design**:** Flaws introduced during the planning and architecture stage can create long-term vulnerabilities in the system. Security Misconfiguration**:** Mistakes in server or application setup, like default passwords or unnecessary features, leave systems exposed. Vulnerable and Outdated Components**:** Using unpatched or obsolete libraries and software modules can open the door to attackers. Identification and Authentication Failures**:** Weak or improperly implemented login mechanisms allow unauthorized access. Software and Data Integrity Failures**:** Without validating updates or external inputs, attackers can insert malicious data or code. Security Logging and Monitoring Failures**:** A lack of proper logging and alerting means breaches may remain undetected for long periods. Server-Side Request Forgery (SSRF)**:** Attackers manipulate servers into sending requests to internal systems, potentially exposing sensitive resources.

1.1 Security Threats in IoT

IoT systems face a variety of security challenges that can put both users and organizations at risk. Researchers such as Podder et al. [9], have highlighted several common threats: Malware: Harmful programs such as viruses, worms, or ransomware can infect IoT devices, steal sensitive data, or corrupt system files. Phishing: Attackers send deceptive emails or messages that trick users into revealing confidential information like usernames or passwords.DoS/DDoS Attacks: By overwhelming a device or server with excessive requests, attackers can slow down operations or make the service unavailable. Weak Passwords: Using simple, default, or repeated passwords makes it easier for unauthorized users to gain access. Outdated Software: Running older software or firmware with known flaws gives cybercriminals an opportunity to exploit those weaknesses. Insider Threats: Authorized individuals such as staff members or contractors may abuse their privileges, whether by mistake or with malicious intent. Man-in-the-Middle (MitM) Attacks: Cybercriminals intercept communication between connected devices and can manipulate or steal the transmitted information. SQL Injection: By inserting malicious commands into input fields, attackers can manipulate databases to view, modify, or delete sensitive records. Zero-Day Exploits: Newly discovered software vulnerabilities can be exploited by hackers before patches or fixes are released. Security Misconfiguration: Using default settings, weak configurations, or leaving unnecessary services enabled exposes IoT systems to potential attacks.

1.2 Focused Threat Analysis

This project focuses on uncovering security flaws in IoT devices such as a soil moisture sensor by simulating real world cyber-attacks. A Raspberry Pi is configured as the target IoT devices, while Kali Linux serves as the attack platform. Once the Raspberry Pi is connected to the internet, it becomes susceptible to potential threats. In this setup, two specific attacks are performed: Cross-Site Request Forgery (CSRF): In a CSRF attack, a malicious actor tricks an authenticated user into unknowingly sending unwanted requests to a server. When the user is logged in, an attacker can inject a crafted link or script that performs unauthorized actions like changing settings or submitting data on their behalf. In IoT systems, this could allow attackers to manipulate sensor readings or alter configurations without the user's consent. Command Injection Attack: Command injection occurs when an application improperly handles input, allowing attackers to insert system commands through insecure fields or parameters. If the IoT device executes these commands, the attacker can gain control of the device, access sensitive data, or disrupt its normal operation. In this project, such an attack simulates how weak input validation in IoT systems can be exploited to compromise device security. Denial of service attack: A DoS (Denial of Service) attack is like someone blocking the entrance to a shop so real customers can't get in. The goal of this attack is to shut down a website or service by sending it too much traffic or too many requests all at once. To protect against DoS threats, security measures like authentication**,** TLS encryption**,** rate limiting**,** and traffic monitoring are used. These defenses help ensure only trusted clients can connect**,** detect unusual activity, and keep IoT systems running reliably.

The structure of this paper is as follows: Section 2 explains the methodology used in the study. Section 3 presents the conclusion**.**

2. METHODOLOGY

This section presents the execution and analysis of two different cyber-attacks performed on the Raspberry Pi is: a) Command Injection Attack b) Cross Site Request Forgery c) Denial of Services

a) Command Injection Attack:
Command Injection attack was also conducted to highlight how an attacker can run system-level commands by exploiting a vulnerable input field. This type of attack typically occurs when the server does not properly validate or sanitize user input before processing it. In this scenario, the Raspberry Pi was running a service on port 8000 which accepted user input through an API or a web form. The sequential process followed to simulate Command Injection Attack during testing. In this setup, Raspberry Pi served as the target system, while Kali Linux was employed to launch the attacks. It illustrated how Command Injection works in the Raspberry Pi accepted input through a web API, but because the input was not properly checked, an attacker could potentially inject harmful commands. In this scenario, the injected input was limited to a harmless ping to DNS server (8.8.8.8), which confirmed that the system could receive instructions and execute them. The experiment highlighted input validation and sanitization are critical when designing IoT services, especially those that accept external commands via MQTT or web APIs by adding proper input validation**,** avoiding unsafe system command calls, and enforcing authentication and encryption**,** the Raspberry Pi service can be secured against Command Injection attacks. These measures ensure that even if an attacker tries to inject malicious input, it will either be rejected or neutralized before it can harm the system.
i) Mosquitto_pub: In this setup, the Raspberry Pi used the mosquitto_pub command to publish MQTT messages to the broker. These messages were sent to the topic commands/update and carried a payload containing a URL, which was used to simulate a firmware update process. Following this, a curl reques**t** was made to a Flask API running on IP: 5000.This API accepted the IP address as input and executed a ping command, serving as part of the command injection test. The Raspberry Pi terminal showed multiple commands being executed to validate communication and simulate the injection scenario through MQTT. The response confirmed successful pings to 8.8.8.8 with low latency and no packet loss, demonstrating that the network was active and reliable, and that the system could correctly interpret and execute MQTT-based instructions such as network ping operations.

b) Cross Site Request Forgery:
In the initial stage, the IoT system was configured without any authentication mechanism. The NodeMCU transmitted soil moisture data to the Raspberry Pi server using simple HTTP requests, and the Flask server accepted all incoming requests on the /soil endpoint without verifying their source. This insecure design made the system highly vulnerable to Cross-Site Request Forgery (CSRF). Using Kali Linux, an attacker could easily craft a malicious HTML page that automatically submitted forged requests to the server. When the victim unknowingly accessed this page, false values such as moisture=9999 were successfully inserted into the Mariadb database. System lacked protection against unauthorized data manipulation. To address this issue, an additional security layer was introduced through API key–based authentication. In the improved version, every request from the NodeMCU included a secret header (X-API-KEY: vasu@123). On the server side, Flask was modified to verify this key before processing data. If the key was incorrect or missing, the request was immediately rejected with a 403 unauthorized response.As a result, only trusted IoT devices could communicate with the server, and any malicious CSRF attempts from attackers were blocked. This enhancement ensured that the database stored only valid sensor readings.

c) DoS Attack:
Denial of Service (DoS): A Denial of Service (DoS) attack targets a device, such as a sensor or an IoT system, by overwhelming it with excessive traffic or repeated requests. This overload can cause the device to slow down, stop responding, or even crash. Unlike other cyber-attacks, the goal of a DoS attack is not to steal or modify data, but rather to make the service unavailable by disrupting normal operations. In this scenario, the attack was simulated using Kali Linux and penetration testing tools like hping3, Slowloris, curl, and SYN flood**.** The IoT setup acted as the target system, while Kali Linux functioned as the attacking

environment. By launching different intrusion attempts, the resilience of the IoT devices against DoS threats was assessed. The initial phase of the penetration testing setup operates a NodeMCU ESP board connected to a soil moisture sensor, All components are linked to the same network via a mobile hotspot. In this configuration, Kali Linux functions as the attacking system, while the Raspberry Pi running Raspberry Pi OS acts as the target IoT server. The Raspberry Pi receives sensor readings from the NodeMCU and stores the data in a MariaDB database, where it is structured and displayed in a tabular format.This attack is display using three different tools: 1.SYN Flood using hping3: This method floods the target system with a large number of TCP/SYN packets. These packets start connections but never complete them, main to numerous half-open connections. A connection of the target system's resources overcome, eventually making it indifferent.2. Slowloris**:** It targets a server by creating multiple slow, unfinished connections. The server waits for each one to complete, eventually becoming overloaded and unable to respond to real users.3**.**MQTT Flood**:** An MQTT Flood attack involves sending a large number of MQTT publish or subscribe messages is sent to an MQTT broker. It consumes the broker's memory and processing power, leading to system overload. This attack is executed using custom Python scripts developed with the Paho MQTT library.

A Denial-of-Service (DoS) attack was carried out using a tool called hping3**.** In this attack, the attacker targeted the MQTT service running on the Raspberry Pi, which listens on port 1883**.** The command used was hping3 –S –flood –p 1883 that means –S sends SYN packets is used to start a connection**.** --flood sends these packets as fast as possible -p 1883**:** targets port 1883, where the MQTT service is running **is** the IP address of the Raspberry Pi. The aim of this attack is to flood the MQTT broker with a huge number of fake connection requests. Because of this flood, the broker gets overloaded and can't respond to real devices. This causes the system to slow down or crash completely. To strengthen the IoT system against Denial of Service (DoS) threats, several MQTT security enhancements were implemented. The broker was configured with authentication and authorization**,** ensuring that only trusted clients could connect and exchange messages. All communication was secured with TLS/SSL encryption**,** which protected data in transit from tampering or malicious injection. In addition, topic-based access control was enforced so that only verified clients could publish or subscribe to sensitive topics such as commands/update. To prevent flooding, rate limiting and session management were applied, disconnecting idle or suspicious clients automatically. Finally, logging and monitoring were enabled to track unusual traffic patterns and quickly identify potential intrusion attempts. Together, these measures improved the reliability of the IoT setup and reduced the impact of DoS-style attacks on the MQTT communication channel.

3. CONCLUSION

This study clearly demonstrates that unsecured IoT systems are highly susceptible to attacks such as CSRF, DoS, and Command Injection. By implementing robust safeguards—including input validation, traffic monitoring, and activity logging—the Raspberry Pi server effectively resisted repeated attacks, maintaining uninterrupted operation and system integrity. The results underscore that rigorous security practices are not optional but essential for real-time IoT deployments. The integrating machine learning–based intrusion detection, a centralized security dashboard, and secure communication protocols like MQTT over TLS will further fortify the system, ensuring maximum privacy, data integrity, and resilience. This work establishes that proactive and comprehensive security design is critical for the safe and reliable operation of IoT systems in an increasingly connected world.

REFERENCES

[1] Al Neyadi, E., Al Shehhi, S., Al Shehhi, A., Al Hashimi, N., Qbea'H, M., & Alrabaee, S. (2020, April). Discovering public wi-fi vulnerabilities using raspberry pi and kali linux. In 2020 12th Annual Undergraduate Research Conference on Applied Computing (URC) (pp. 1-4). IEEE.

[2] Bakry, B. B. M., Adenan, A. R. B., & Yussoff, Y. B. M. (2022, November). Security attack on IoT related devices using Raspberry Pi and Kali Linux. In 2022 International Conference on Computer and Drone Applications (IConDA) (pp. 40-45). IEEE.

[3] Gunawan, T. S., Lim, M. K., Zulkurnain, N. F., & Kartiwi, M. (2018). On the review and setup of security audit using Kali Linux. Indonesian Journal of Electrical Engineering and Computer Science, 11(1), 51-59.

[4] Islam, M. S., Islam, M. S., Rabbi, J. H., Biddut, A. B. A., & Hossen, M. S. (2025). VARIOUS APPLICATION OF IOT-BASED WEATHER MONITORING SYSTEMS IN THE AGRICULTURE SECTOR FOR BANGLADESHI FARMERS. Scientific and practical cyber security journal.

[5] Donta, P. K., Srirama, S. N., Amgoth, T., & Annavarapu, C. S. R. (2022). Survey on recent advances in IoT application layer protocols and machine learning scope for research directions. Digital Communications and Networks, 8(5), 727-744.

[6] Lazzaro, S., De Angelis, V., Mandalari, A. M., & Buccafurri, F. (2024, March). Is your kettle smarter than a hacker? a scalable tool for assessing replay attack vulnerabilities on consumer iot devices. In 2024 IEEE international conference on pervasive computing and communications (PerCom) (pp. 114-124). IEEE.

[7] Arz von Straussenburg, A. F., Blazevic, M., & Riehle, D. M. (2023, May). Measuring the actual office workspace utilization in a desk sharing environment based on IoT sensors. In International Conference on Design Science Research in Information Systems and Technology (pp. 69-83). Cham: Springer Nature Switzerland.

[8] Wang, Z., Chen, H., Yang, S., Luo, X., Li, D., & Wang, J. (2023). A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization. PeerJ Computer Science, 9, e1569.

[9] Podder, P., Mondal, M. R. M. R., Bharati, S., & Paul, P. K. (2021). Review on the security threats of internet of things. arXiv preprint arXiv:2101.05614.