International Journal of Innovative Computing, Information and Control
Volume **20**, Number **2**, April **2024**

ICIC International ©2024 ISSN 1349-4198
pp. 617–634

IMMUNITY-BASED DETECTION OF CYBERATTACKS ON MQTT BROKERS

Takeshi Okamoto

Department of Information Network and Communication
Kanagawa Institute of Technology
1030 Shimo-ogino, Atsugi, Kanagawa 243-0292, Japan
take4@nw.kanagawa-it.ac.jp

Received July 2023; revised November 2023

Abstract. *In smart cities, public services face the risk of cyberattacks, with the most significant threat being denial-of-service attacks targeting unknown vulnerabilities. To bolster defence mechanisms, we previously introduced a method known as "immunity-based attack detection". This approach dynamically develops immunity against both known and unknown cyberattacks without the need for prior training on attack data. This paper focuses on a disaster prevention service utilizing Message Queuing Telemetry Transport (MQTT) brokers. We propose implementing immunity-based Attack Detection (ibAD) specifically for the Mosquitto broker, to enhance the resilience of the MQTT broker. In performance evaluations, the ibAD method successfully detected and prevented attack messages with an accuracy of 99.72% against actual vulnerabilities. Furthermore, our assessments revealed that ibAD imposed minimal overhead, especially when the MQTT message size remained below approximately 4,000 bytes within our experimental environment.*

Keywords: Intrusion detection, DoS attack, Message Queuing Telemetry Transport (MQTT), Vulnerability, Machine learning, Immune system

1. **Introduction.** As we move towards realizing smart cities, the smartification of social infrastructures and public services is gaining prominence. Smart cities are anticipated to enhance the efficiency of traditional social infrastructures and public services, offering improved quality of life while prioritizing environmental sustainability. One crucial aspect of smart cities is disaster prevention services, which rely on collecting sensor data, such as water levels and tidal information, to detect and predict disasters like floods and storm surges. This data-driven approach minimizes the impact of disasters, ensuring the safety and well-being of residents. However, the disaster prevention services of smart cities are vulnerable to cyberattacks. If these services experience disruptions in data collection and control due to cyberattacks, detecting and predicting disasters becomes challenging. This could potentially exacerbate the damage caused by disasters, posing a risk to the lives and property of residents. Therefore, it is crucial for the disaster prevention services in smart cities to enhance their resilience against cyberattacks.

To bolster the resilience of Internet services, we previously introduced a method known as "immunity-based Attack Detection" (ibAD). Inspired by the strategies of biological immune systems, encompassing both innate and adaptive immunity, ibAD adaptively acquires immunity to known and unknown cyberattacks [1]. While ibAD may not prevent an initial specific attack without halting the service, it effectively does so for subsequent

DOI: 10.24507/ijicic.20.02.617

similar attacks by learning from the initial one. One advantage of ibAD is its independence from prior training data. It autonomously collects and learns attack data each time the service faces an attack and is halted. This stands in contrast to conventional machine learning, which often encounters practical barriers due to the need for diverse prior training data. Another advantage is the autonomous recovery of services halted by attacks. These features contribute to a server application's high resilience against cyberattacks. In an experimental case applying ibAD to the Internet Systems Consortium Berkeley Internet Name Domain (ISC BIND), the most popular Domain Name System (DNS) implementation, we verified ibAD's ability to detect attacks on actual vulnerabilities with 99.94% accuracy and only 2.7% overhead [1]. However, applying ibAD to BIND is heavily reliant on the DNS protocol and the specific implementation of BIND, rendering it unsuitable for use in disaster prevention services for smart cities.

To bolster the resilience of disaster prevention services, it is imperative to redesign and implement ibAD according to the protocols and implementations specific to disaster prevention services. In this context, the measurement and collection of observational data in the natural environment – such as water and tide levels, humidity, and temperature, crucial for predicting natural disasters – play a pivotal role. The Message Queuing Telemetry Transport (MQTT) [2] protocol emerges as a fitting choice for collecting such observational data. MQTT operates as a publish/subscribe messaging protocol, facilitating lightweight and efficient communication among Internet of Things (IoT) devices. Compared to prominent protocols in IoT services like Hyper Text Transfer Protocol (HTTP) and Constrained Application Protocol (CoAP) [3], MQTT boasts advantages in high-speed communication performance at low bandwidth, reliable message delivery capabilities, flexible routing, and security features through authentication and encryption. However, the weak point of disaster prevention services adopting MQTT lies in the MQTT brokers responsible for relaying observation data. Some vulnerabilities, such as CVE-2018-12543[1] and CVE-2019-11779, have been identified in MQTT brokers. The exploitation of these vulnerabilities can disrupt the service by blocking the relay of observation data, thereby impeding the prediction of disasters based on this data. As of the author's knowledge, methods to enhance the resilience of MQTT brokers against attacks exploiting vulnerabilities have not been thoroughly investigated.

In this paper, we presume a disaster prevention service utilizing MQTT brokers and propose ibAD for the Mosquitto broker – an implementation of the MQTT broker – aiming to fortify the resilience of MQTT brokers. We designed ibAD specifically for the Mosquitto broker and implemented a prototype. To elucidate the performance of ibAD on the Mosquitto broker, we initially determined the optimal parameters through simulation evaluations. Subsequently, we assessed the detection accuracy against attacks on real vulnerabilities. Additionally, we gauged the overhead of ibAD on actual hardware.

This paper contributes the following.

- This paper introduces a feature called *innate immune function*, which detects and identifies attacks using signals within the Mosquitto broker. Attack detection relies on compiler built-in security features. Notably, this function is entirely distinct and unique compared to ibAD for BIND.
- The paper proposes the implementation of an *adaptive immune function* for the Mosquitto broker. This function can learn and classify normal and attack messages through supervised machine learning. While this function aligns with ibAD for BIND, the paper outlines specific modifications to the source code for Mosquitto.

[1]CVE (Common Vulnerabilities and Exposures) identifiers are unique common identifiers for publicly known information-security vulnerabilities in publicly released software packages.

- The paper identifies optimal parameters for ibAD through simulation evaluations. Demonstrating robust performance, ibAD achieved a high accuracy of 99.72% in detecting attacks on actual vulnerabilities. The true negative rate was 99.77%, and the true positive rate reached 99.67%.
- In assessing the overhead of ibAD, it was found to be negligible when the size of the MQTT message was limited to 4,000 bytes or less. In practical terms, this implies that ibAD for MQTT can effectively establish disaster prevention services with high resilience against cyberattacks, provided the observation data is within the 4,000-byte limit.

Section 2 of this paper describes related work. Section 3 outlines the MQTT protocol and its implementation. Section 4 presents a design and implementation of ibAD for the Mosquitto broker. Section 5 shows performance evaluations of the prototype for accuracy of detection and overhead. Section 6 discusses disaster prevention services applicable to ibAD and cyberattacks specific to ibAD. Section 7 is the conclusion.

2. **Related Work.** Machine learning methods can identify attack data within network traffic through centralized traffic monitoring [4]. The use of machine learning for attack detection enables highly accurate and rapid classification [5, 6]. IDS-MA, a system designed to detect attacks on the MQTT protocol using both centralized and federated learning models, has been proposed and demonstrated high detection accuracy [7]. However, it is important to note that these detection accuracies, while high, may not reach 100%. Consequently, zero-day attacks may go undetected, potentially leading to DoS incidents. Even if the service is restarted after an attack, machine learning alone may not prevent DoS because machine learning has no means to adaptively learn attack data. In contrast, ibAD possesses the unique capability to automatically recover the service from DoS incidents. It achieves this by detecting and preventing subsequent attacks similar to the initial one that caused the DoS, learning from the first attack. Moreover, traditional machine learning methods often face challenges due to the requirement for extensive training datasets comprising both normal and attack data. The collection of such diverse datasets can be challenging, especially for zero-day attacks. In contrast, ibAD eliminates the need for prior training data, as it acquires training data every time it detects DoS incidents.

Anomaly detection methods require no prior attack data. Anomaly detection using autoencoders has been a subject of active research. The application of variational autoencoders for anomaly detection has been proposed, and it has been demonstrated that variational autoencoders excel in detecting unknown attacks compared to traditional autoencoders or one-class support vector machines [8]. Additionally, anomaly detection employing a probabilistic artificial immune algorithm, inspired by the immune system, has shown highly accurate detection capabilities at a low computational cost [9]. Even when large quantities of normal data are collected, ensuring its true normalcy can be challenging, especially when sourced from diverse users and devices. In contrast, ibAD ensures the legitimacy of each message as either normal or an attack by verifying its impact on service operation, particularly its potential to cause DoS [7].

Adaptive machine learning methods, aimed at detecting unknown attacks through recursive learning [10, 11], may introduce false positives, leading to a cascade of additional false positives. To mitigate this, such methods necessitate mechanisms guaranteeing the legitimacy of each detected attack. As highlighted earlier, ibAD eliminates erroneous learning by ensuring the authenticity of each message.

Behavior-based attack prevention methods can heuristically detect attacks by monitoring the behavior of an application process on the host, e.g., system call sequences [12, 13],

and control flows [14, 15, 16, 17, 18]. However, these techniques terminate the application process when they detect attacks, resulting in DoS. ibAD employs behavior-based attack prevention methods but distinguishes itself by its ability to recover the service from DoS incidents.

Intrusion-tolerant systems strengthen resistance to zero-day attacks by implementing fault tolerance based on diversity. These systems run multiple server applications with different implementations in parallel [19, 20, 21, 22]. For instance, in an intrusion-tolerant system designed for a DNS server, BIND, unbound, and PowerDNS would run in parallel, employing round-robin processing for requests across the three server applications. If one of these server applications becomes unavailable due to an implementation vulnerability, the remaining server applications continue to provide service. The effectiveness of the intrusion-tolerance system relies on the absence of common vulnerabilities across all implementations, specifically vulnerabilities in their specifications [23]. However, implementing intrusion-tolerance systems involves substantial resource requirements (CPU, memory, disk, etc.) to run multiple server applications. The initial setup of each server application and ongoing maintenance costs, including security updates, can be high. In contrast, ibAD requires no additional diverse broker applications, eliminating additional costs associated with their setup and maintenance. In essence, ibAD can fortify its resistance to attacks with just a single implementation. This capability stems from its ability to promptly identify attack data, restart the service when disrupted by an attack exploiting a vulnerability, and subsequently learn from the attack data to prevent similar attacks in the future. Moreover, ibAD circumvents the need for "data preparation", including manual labelling and verification of normal and attack data, as well as "system maintenance", such as updating the learning model. Unlike traditional methods, ibAD dynamically collects and labels data during service operation, learning from these data each time the server application is attacked.

The distinction between conventional methods and ibAD lies in the service's resilience against attacks. Conventional methods, capable only of detecting and preventing cyberattacks, result in DoS when an attack goes undetected or cannot be prevented. Conventional methods lack service resilience. In contrast, ibAD acknowledges the inherent limitation of conventional methods in preventing all attacks and boasts the resilience to swiftly recover the service in the face of disruptions caused by an attack.

To enhance this resilience for MQTT brokers, this paper proposes a novel function aimed at identifying MQTT messages employed in attacks. This function complements the behavior-based attack detection discussed earlier. By pinpointing the attack messages, ibAD becomes self-sufficient in the supervised data necessary for machine learning. Consequently, ibAD eliminates the need for advance data preparation of attack data. This function is unique to MQTT and was newly introduced in ibAD.

3. MQTT and the Mosquitto Broker.

The MQTT protocol, a lightweight publish/subscribe data distribution protocol, stands as one of the most popular choices for IoT-based services. It comes in two versions: version 3.1.1, released in 2013, and version 5, released in 2019. As of the current writing, 23 out of the total 27 MQTT implementations, encompassing both brokers and clients, support version 3.1.1. However, only 18 implementations currently support version 5.0. Given that IoT devices often operate for extended periods without frequent updates, we anticipate a continued prevalence of version 3.1.1 implementations. Consequently, the initial prototype in this study centers on version 3.1.1 implementations. For version 5.0 vulnerabilities, mitigation can be achieved by blocking all MQTT messages with version 5.0 in firewall settings.

A standard MQTT communication flow is depicted in Figure 1. The publisher initiates the sequence by sending a `CONNECT` message to the broker, followed by a `PUBLISH` message containing observation data. The communication concludes with the publisher terminating the connection through a `DISCONNECT` message. On the other hand, a subscriber sends a `CONNECT` message to the broker, then a `SUBSCRIBE` message, and waits to receive data on a specified topic.

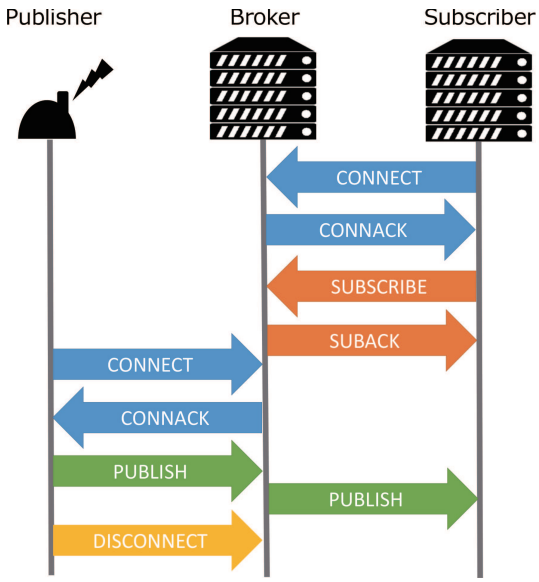

FIGURE 1. Typical communication flow of MQTT

MQTT version 3 incorporates features such as will messages, delivery assurance, and user/device authentication, though these features are not within the scope of this paper.

At the time of writing, there were 17 implementations of MQTT brokers [24]. Among open-source implementations, Eclipse Mosquitto, implemented in the C language, was chosen. This choice was made, in part, by the discovery of vulnerabilities in the Mosquitto broker, including CVE-2018-12543 and CVE-2019-11779, which can lead to DoS incidents.

4. Immunity-Based Detection of Cyberattacks on the Mosquitto Broker.

ibAD encompasses an innate immune function and an adaptive immune function. The innate immune function mirrors the innate immunity of biological systems. It detects both known and unknown attacks using behavior-based attack prevention methods outlined in Section 2. Additionally, the innate immune function identifies the data used in the attack. The adaptive immune function then learns this attack data, preventing subsequent similar attacks.

The adaptive immune function mirrors the adaptive immunity of biological systems. It cannot detect the first attack but learns from attacks detected by the innate immune function, enabling the detection of second and subsequent similar attacks. This function is realized through supervised machine learning techniques. The innate immune function supplies supervised data, eliminating the need to learn attack data before starting the service.

ibAD scrutinizes incoming messages of any type, even those not defined in the MQTT specification. The flow of how ibAD acquires immunity to attacks by receiving messages is as follows (Figure 2).

1) When the broker starts up, the adaptive immune function learns supervised data. Note that the supervised data does not contain any attack data at the first start.

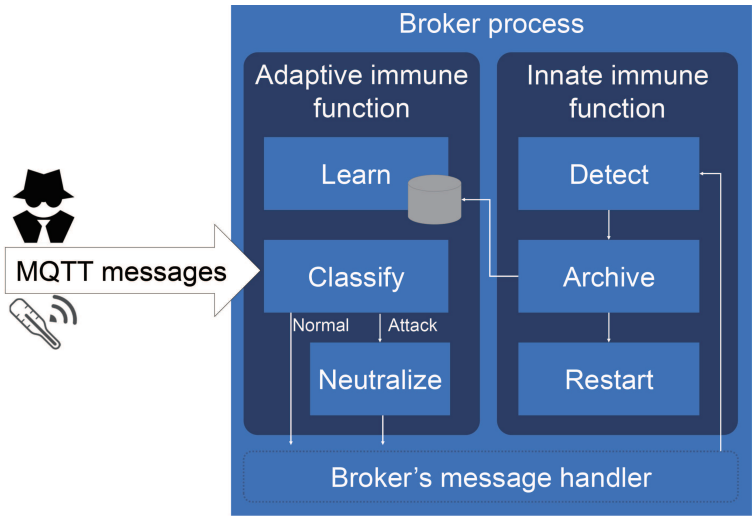

FIGURE 2. Overview of the ibAD method

2) When the broker receives an MQTT message, the adaptive immune function uses supervised machine learning to classify it.
3) If the machine learning classifies the message as an attack, the message is neutralized, and the broker continues to process the neutralized message. If the machine learning classifies the message as normal, the broker continues to process the message. Any formatting errors in the MQTT message are detected by the broker's standard parser, and the broker takes appropriate action.
4) If the innate immune function detects an attack during message processing by the broker, the innate immune function archives the previously received MQTT messages as supervised data and restarts the broker.
5) Return to Step 1).

Both functions are incorporated in the Mosquitto broker by modifying its source code, which is version 1.6.1, in order to reproduce the DoS caused by attacks on CVE-2019-11779.

4.1. **Innate immune function.** The innate immune function not only detects cyberattacks but also saves training data – specifically, MQTT messages received by the Mosquitto broker – for the adaptive immune function to learn from. After saving the training data, the innate immune function initiates a restart of the process. This section elaborates on the mechanism of detecting attacks, preserving training data, and restarting the process.

4.1.1. *Cyberattack detection.* The primary role of the innate immune function is the detection of attacks targeting vulnerabilities that lead to DoS or Remote Code Execution (RCE). The latter not only triggers detection but also includes prevention measures. It is important to note that attacks consuming network bandwidth or internal OS resources, as well as attacks against lower layers such as TLS, TCP, and IP vulnerabilities, are excluded from consideration.

DoS vulnerabilities manifest in two types: memory access violations and assertion failures. A memory access violation, when attacked, prompts the process to receive a segmentation violation signal (SIGSEGV). On the other hand, an assertion failure vulnerability, when attacked, leads to the process receiving an abort signal (SIGABRT). The reception of these signals serves as an indicator of attack detection.

Given that the Mosquitto broker lacks the inherent capability to detect attacks against RCE vulnerabilities, we have augmented the Mosquitto broker with additional security features using GNU C compiler and linker options.

- Stack smash attack protection: `-fstack-protector-strong`
- Stack crash attack protection: `-fstack-clash-protection`
- Control flow protection: `-fcf-protection=full`
- Signed-integer overflow detection: `-ftrapv`
- Replacement with safe functions: `-D_FORTIFY_SOURCE=2`
- Position independent executable generation: `-fPIE` and `-pie`
- Read-only memory segments: `-z relro`
- No code execution on stack: `-z noexecstack`

For instance, to prevent RCE attacks on a buffer overflow vulnerability, Stack Smash attack Protection (SSP) is employed [25]. SSP detects buffer overflows by inserting a special value, known as a "canary", between a buffer and control data on the stack and then checking for corruption of that special value.

These security features empower a process to receive signals such as SIGABRT when under attack, facilitating the detection and prevention of RCE vulnerabilities. To manage each signal, the innate immune function registers a signal handler for each signal during process startup. These signal handlers treat the most recent message handled by the broker process just before a signal is generated as the message used in an attack. This is attributed to the single-threaded nature of the Mosquitto message handler, which processes messages sequentially. If no signals are received, the innate immune function considers the most recent message as normal.

4.1.2. *Saving training data and restarting the process.* Messages for training data are stored in queues by class, the normal queue, Q_{c_0}, and the attack queue, Q_{c_1}, where $c_0 = \mathit{normal}$ and $c_1 = \mathit{attack}$. When a queue reaches its maximum size, the oldest messages are discarded in order. The length of a queue, L, is a parameter of ibAD determined by the simulation evaluation described in Section 5.3, as it depends on detection accuracy.

Upon receiving a signal, the signal handler of the innate immune function saves all the messages in the two queues to a file, terminates the process, and then immediately restarts the Mosquitto broker process using Systemd.

4.2. **Adaptive immune function.** The adaptive immune function not only learns and detects attack messages using supervised machine learning but also neutralizes attack messages. The procedure for learning and classification is illustrated in Figure 3.

For classification, each time an MQTT message is received, it is converted into an N-dimensional vector of features. This N-dimensional vector is then input into the learning model to obtain classification results. The input vectors are stored in the normal or attack queue for training. All vectors in the normal and attack queues are trained by a learning algorithm, generating a learning model. Figure 3 illustrates an example using decision tree learning. This section further details the features fed into a learning model and each operation of the adaptive immune function.

4.2.1. *Features for a learning model.* The features for a learning model consist of a sequence of bytes from the first to the N-th byte of the MQTT message received from the client, mapped to an N-dimensional byte array. Specifically, the MQTT message "m" is defined by $m = (a_1, \ldots, a_i, \ldots, a_N)$, where a_i represents the value of the i-th byte of the MQTT message, i.e., $0 \leq a_i \leq 255$, and $1 \leq i \leq N$. This N-dimensional byte array only includes MQTT data, excluding IP and TCP headers. If the MQTT message is less

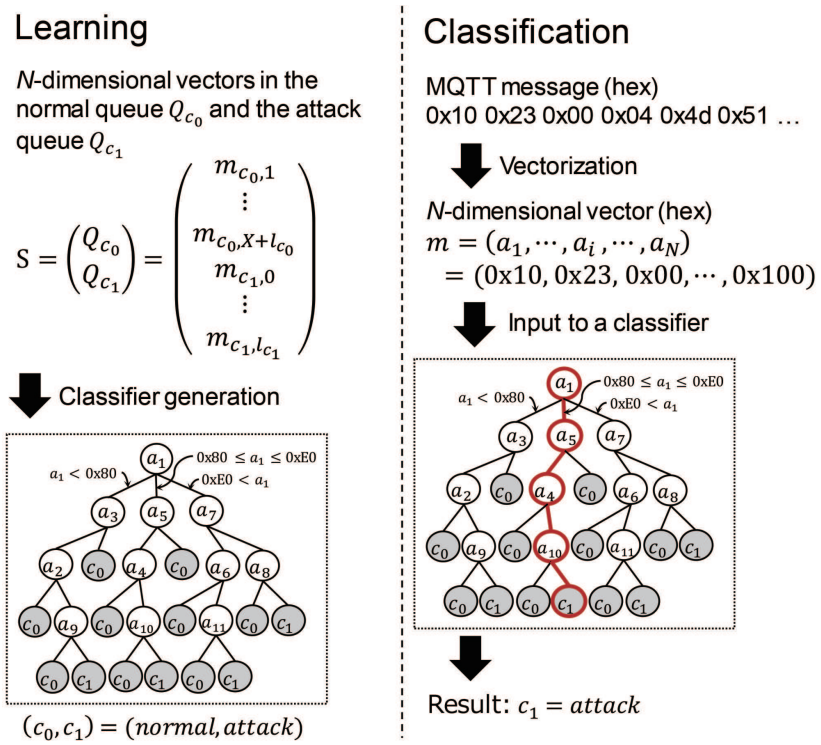

FIGURE 3. Procedure for classification and learning

than N bytes, the adaptive immune function fills the remaining features with 256 values, indicating no data at that position.

To clarify, data after the N-th byte of the message is not subject to analysis by machine learning. Consequently, the adaptive immune function cannot detect an attack message with embedded attack data occurring after the N-th byte of the message. As a preventive measure, we configure the Mosquitto parameter, `message_size_limit`, to N, compelling the Mosquitto broker to discard messages exceeding N bytes.

The classification speed of machine learning diminishes with an increase in the number of feature dimensions. For instance, when N is set to 256 MB, the maximum message size of the MQTT protocol, the processing time surpasses 10 seconds per message on a server equipped with Ryzen Threadripper 2950X. Therefore, the number of feature dimensions significantly influences classification speed, making it a critical parameter for ibAD. This parameter is determined through the performance evaluation outlined in Section 5.5.

4.2.2. *Generating a learning model.* The adaptive immune function generates a learning model by supervised machine learning just after the Mosquitto broker process is created, and then the MQTT broker service is started. The algorithm of supervised machine learning is a parameter of ibAD and is determined by the performance evaluation described in Section 5.2. It affects detection accuracy and classification speed.

Supervised data encompasses both static and dynamic components, exclusively comprising normal messages collected before the initiation of the service. This data remains constant after the service commences, and its size, denoted as X, is a parameter determined through the performance evaluation detailed in Section 5.4. Conversely, dynamic supervised data is generated whenever the innate immune function identifies an attack. This allows the adaptive immune function to recognize subsequent similar attacks. Dynamic supervised data includes messages stored in both the normal and attack queues by the innate immune function just before the process terminates. The normal queue,

Q_{c_0}, combines static and dynamic supervised data: $Q_{c_0} = \left(m_{c_0,1}, \ldots, m_{c_0,i}, \ldots, m_{c_0,X+l_{c_0}}\right)$, where $m_{c_0,i}$ represents the i-th normal message, $1 \leq i \leq X + l_{c_0}$, l_{c_0} denotes the length of the normal queue, and $0 \leq l_{c_0} \leq L$. In contrast, the attack queue, Q_{c_1}, exclusively contains dynamic supervised data: $Q_{c_1} = (m_{c_1,1}, \ldots, m_{c_1,i}, \ldots, m_{c_1,l_{c_1}})$, where $m_{c_1,i}$ is the i-th attack message, $0 \leq i \leq l_{c_1}$, l_{c_1} represents the length of the attack queue, and $0 \leq l_{c_1} \leq L$. Each $m_{c_0,i}$ and $m_{c_1,i}$ is the N-dimensional vector defined in Section 4.2.1.

Learning of messages is based on the supervised machine learning algorithm. For the sake of clarity, the following is the algorithm for learning messages using Iterative Dichotomiser 3 (ID3), a supervised machine learning algorithm also known as decision tree learning.

- Initialization: Set the entire dataset as a set S for the root node, where $S = (Q_{c_0}, Q_{c_1})$.
- Iteration: Repeat the following steps.
  - Attribute Selection: From the attributes, corresponding to a "feature" in Section 4.2.1, that have not been used yet, select an attribute that can split S so that the entropy $H(S)$ is minimized. $H(S)$ is defined by the formula:

$$H(S) = -\sum_{x \in (c_0, c_1)} p(x) \log_2 p(x)$$

    where S is the current dataset for which entropy is being calculated, and $p(x)$ is the ratio of elements belonging to class x in S. The smaller the entropy, the more biased the class distribution of S.
  - Division: Divide S into subsets by the selected attribute. For each subset, create a new node and branch it from the parent node. The label of the branch is the name of the attribute. For example, as shown in Figure 3, the root node, a_1, can be divided into child nodes according to subsets of MQTT messages. These subsets are determined by the first attribute value being less than 0x80, between 0x80 and 0xE0, and greater than 0xE0.
  - Termination Judgment: For each subset, determine whether any of the following conditions are met:
    * Uniformity: If all elements in the subset belong to the same class, label the node as a leaf node and label it with the class.
    * Attribute Exhaustion: If there are no more usable attributes, label the node as a leaf node and label it with the most common class in the subset.
    * Element Exhaustion: If the subset becomes empty, designate the node as a leaf node and label it with the most common class in the parent node's subset.
  - Recursion: Apply the iteration steps recursively to nodes that did not become leaf nodes in the termination judgment. In this case, use the subset as a new S and exclude the attributes already utilized.
- End: When all nodes become leaf nodes, conclude the algorithm.

Finally, this algorithm outputs a decision tree consisting of internal and leaf nodes. The internal nodes represent the attributes into which the subset is split, and the leaf nodes represent the class of normal or attack.

4.2.3. *Detection and neutralization of attacks.* The adaptive immune function uses a learning model, i.e., a classifier, to categorize messages into normal or attack classes when the Mosquitto broker receives a message from a client. The program code for classification is added to the `handle__packet` function defined in `read_handle.c`, one of the Mosquitto source files.

The procedure for classifying messages using ID3 is described in detail below: First, when the Mosquitto broker receives an MQTT message, depicted on the right side of Figure 3, consisting of a sequence of N bytes (e.g., 0x10, 0x23, 0x00, 0x04, 0x4D, 0x51, ...), the adaptive immune function represents the MQTT message as N-dimensional input data m. Subsequently, the adaptive immune function inputs this data into ID3 to obtain the classification result.

ID3 initiates the classification process from the root node of the decision tree, traversing the tree based on each attribute value of the input data. Upon reaching a leaf node, it assigns the label of that node as the class of the input data. For instance, the root node in Figure 3 branches into three nodes corresponding to the value of a_1. Given that a_1 satisfies the inequality $0\text{x}10 \leq a_1 < 0\text{xE}0$, the traversal moves from the root node to the node with a_5. Repeating this process, upon reaching the leaf node assigned the class of c_1, the classifier classifies the message as an attack class, as c_1 represents an attack message.

If the classifier classifies a message as an attack, the adaptive immune function neutralizes the attack message by overwriting with zeros following the "remaining length" at the second byte from the beginning of the message. Once neutralized, the adaptive immune function continues the execution of the `handle__packet` function.

If the classifier classifies the message as normal, it continues the execution of the `handle__packet` function. The timing of adding a message to the normal queue is not when it is classified as normal, but when the `handle__packet` function successfully completes handling the message. This is because the classifier may incorrectly classify an attack message as normal. In other words, if no SIGSEGV and SIGABRT signals are received before the function successfully completes, the innate immune function regards the message as normal and adds it to the normal queue.

5. **Performance Evaluation.**

We evaluated the performance of the Mosquitto broker equipped with ibAD. First, we determined suitable parameters of ibAD through simulation evaluations. In these evaluations, we attacked two actual vulnerabilities: CVE-2018-12543 and CVE-2019-11779. We also implemented the ibAD prototype described in the previous section to evaluate the overhead of ibAD.

5.1. **Scenario for performance evaluations.**

We assumed that various publishers and subscribers repeatedly send messages to the Mosquitto broker. Each client sends messages in the order: `CONNECT` → `PUBLISH` / `SUBSCRIBE` → `DISCONNECT`. The MQTT message data sent by normal publishers and subscribers are as follows:

- Client ID: A random string of 10 to 100 characters.
- Message ID: A random number between 0 and 65535.
- Topic: The topic level is determined by a random number from 1 to 4, and the topic name for each level is a random string of 1 to 8 characters. The topic name of the lowest level is one of three strings: "temperature", "humidity", or "waterlevel". The subscriber has a 10% chance of assigning a wildcard ("`#`" or "`+`") to the topic name.
- Message: A random number between −100 and 100, assuming temperature, humidity, and water level.

To assess the worst-case scenario, we assumed that the broker had vulnerabilities CVE-2018-12543 and CVE-2019-11779 simultaneously. There might be few instances where multiple vulnerabilities coexist and are attacked simultaneously in a server application.

The CVE-2018-12543 vulnerability causes a DoS due to an assertion failure when the first character of the topic is a dollar sign ("`$`"), excluding topics that start with "`$SYS`" [26]. An attacker sends a `PUBLISH` or `SUBSCRIBE` message with the topic starting with a dollar sign.

The CVE-2019-11779 vulnerability causes a DoS due to stack overflow when a topic contains more than about 65,400 slash symbols ("/") [27]. An attacker sends a `SUBSCRIBE` message with a topic containing 65,400 to 65,535 slash symbols.

The scenarios for performance evaluations was as follows.

1) The broker starts its own service, and the publisher or subscriber sends a total of X normal messages to the broker. The broker successfully handles them, where X represents the number of pre-training data. Normal messages include `CONNECT`, `PUBLISH`, `DISCONNECT`, and `SUBSCRIBE` messages.
2) The vulnerability CVE-2018-12543 or CVE-2019-11779 is exploited, causing the broker to learn X normal messages and 1 attack message and then restart itself.
3) The broker receives 1,500 normal messages and 1,500 attack messages in random order.

Performance evaluations were performed starting from the third step described above. Since detection accuracy depends on the message contents and their transmission order, 50 trials were conducted for each performance evaluation, with the contents and transmission order randomly changed for each trial. The performance indices, such as detection accuracy, were averaged over the 50 trials.

5.2. **Comparative analysis of classifiers.** We compared and evaluated the detection accuracy and classification time of various classifiers using the Python scikit-learn module. Detection accuracy was calculated from the confusion matrix, and classification time represents the average time required to classify a message. The classifiers considered were all the classifiers in the scikit-learn module. However, we utilized LightGBM [28] due to its superior implementation of gradient boosting compared to that in the scikit-learn module. Default parameters were used for all classifiers except LightGBM; the parameters for LightGBM were the same as the default parameters for gradient boosting in the scikit-learn module. The parameters of ibAD are as follows:

- Number of pre-training data: $X = 100$
- Queue length: unlimited
- Number of feature dimensions: $N = 300$

Figure 4 illustrates the results of detection accuracy and classification time; classifiers $< 85\%$ detection accuracy and classifiers > 35 µs classification speed were excluded. The

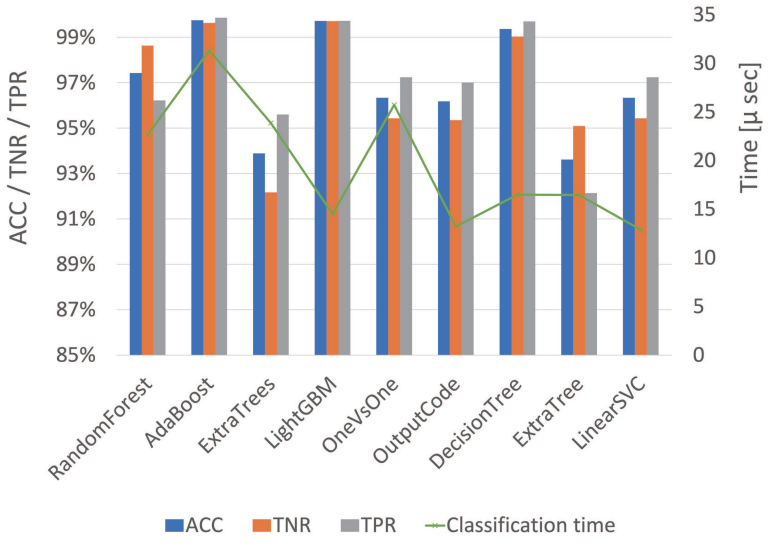

FIGURE 4. Comparison of classifiers

left y-axis measures detection accuracy (ACC), the True Negative Rate (TNR), and the True Positive Rate (TPR); the right axis measures classification speed, presented as a line plot in the graph.

Table 1 presents the top three classifiers for detection accuracy. While AdaBoost achieved the highest accuracy, LightGBM, with nearly the same accuracy as AdaBoost, was 2.16-fold faster. Consequently, we selected LightGBM as the machine learning algorithm for the prototype.

TABLE 1. Statistics of accuracy and classification time

Classifier	Accuracy [%]	Time [μs]
AdaBoost	99.75 ± 0.31	31.31 ± 11.98
LightGBM	99.72 ± 0.31	14.49 ± 0.16
DecisionTree	99.36 ± 1.14	16.48 ± 0.21

5.3. **Evaluation of queue length.** We assessed the detection accuracy for different queue lengths, varying from 20 to 200, to determine the most suitable length. The other parameters of ibAD were set as follows:

- Machine learning algorithm: LightGBM
- Number of pre-training data: $X = 100$
- Number of feature dimensions: $N = 300$

Figure 5 illustrates the accuracy of the queue length. The queue length converges at around 7, reaching approximately 99.69%. This suggests that the number of attack data required to detect attacks on two vulnerabilities is approximately 7. Since the convergence value may vary based on the number and type of vulnerabilities, it is advisable not to pursue a lower bound for the queue length.

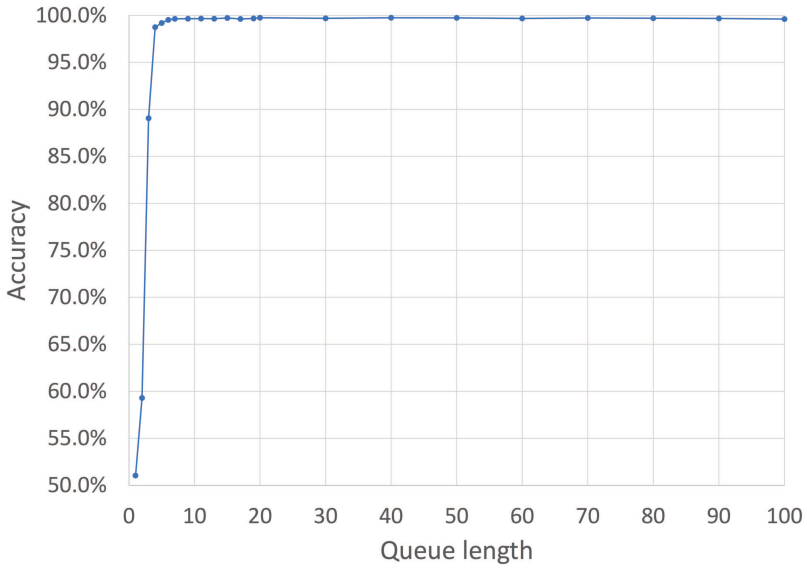

FIGURE 5. Average accuracy as a function of the length of two queues

As the adaptive immune function learns when the broker is started, learning time may affect startup time. We examined learning time for each queue length. It took less than 1 second to learn two queues within 200 messages, having little effect on startup time. Therefore, we set the length of a queue, L, to 100.

5.4. **Evaluation of the number of pre-training data.** We evaluated the detection accuracy for different numbers of pre-training data, ranging from 10 to 200, to determine a suitable number of pre-training data. The other parameters of ibAD were the same as those in Section 5.3, except for the queue length, which was set to 100.

Figure 6 shows detection accuracy as a function of the number of pre-training data. Detection accuracy converged to around 99.60% when the number of pre-training data exceeded approximately 100. Accordingly, we set the number of pre-training data, X, to 100.

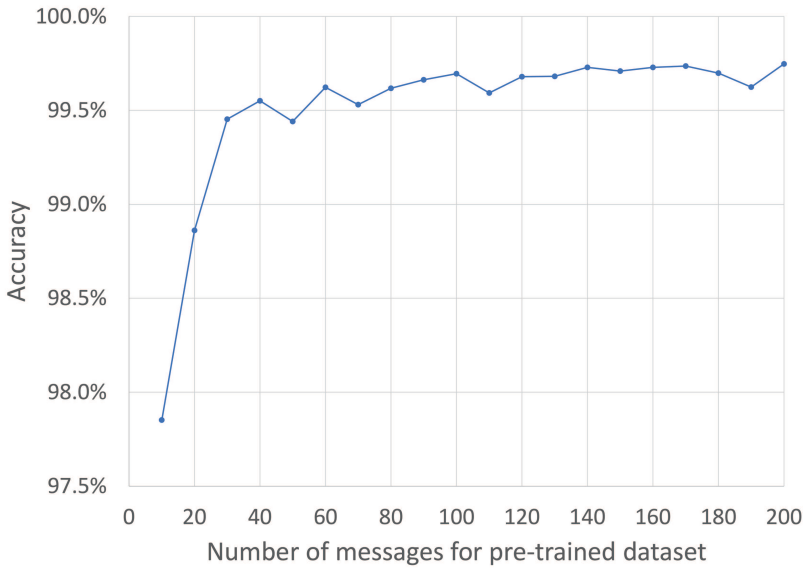

FIGURE 6. Average accuracy as a function of the number of messages for pre-training dataset

5.5. **Evaluation of the number of feature dimensions.** We evaluated the throughput of MQTT messages for each number of feature dimensions by varying the number of feature dimensions from 300 to 10,000 to determine a suitable number. The other parameters of ibAD were the same as those in Section 5.3, except for the queue length, which was set to 100.

The throughput was evaluated on an actual broker connected to a client via a Layer 2 Gigabit Ethernet switch. The machine and software specifications of the broker and client are as follows:

- Broker: Mosquitto 1.6.1
  - CPU: Ryzen Threadripper 2950X
  - Memory: 128 GB
  - OS: Linux (Kernel 5.9.0)
- Client: MQTTLoader 0.7.3
  - CPU: Celeron G3900 @ 2.8GHz
  - Memory: 8 GB
  - OS: Linux (Kernel 5.6.0)

We used Mosquitto 1.6.1, which has the CVE-2019-11779 vulnerability, to verify the detection of attacks on the vulnerability. The CPU of the broker machine does not support Intel CET [15]; therefore, the control flow protection described in Section 4.1.1 would not be effective. MQTTLoader is benchmark software for MQTT [29]. We launched only one publisher in MQTTLoader, which published 2.5 million messages to the broker.

Figure 7 displays a box-and-whisker plot of the throughput. The dotted line represents the median throughput of Mosquitto without ibAD. If the number of feature dimensions was below approximately 4,000, the throughput of Mosquitto with ibAD was nearly the same as that without ibAD. This result remained consistent even when the number of publishers in MQTTLoader was increased. These findings indicate that the ibAD overhead has little impact on throughput if the number of feature dimensions is less than 4,000 bytes in our experimental environment. Since the IoT service in this study was designed to collect observation data such as temperature and water level, message sizes are several hundred bytes at most. For convenience of speeding up the simulation evaluation, we set the number of dimensions, N, to 300.

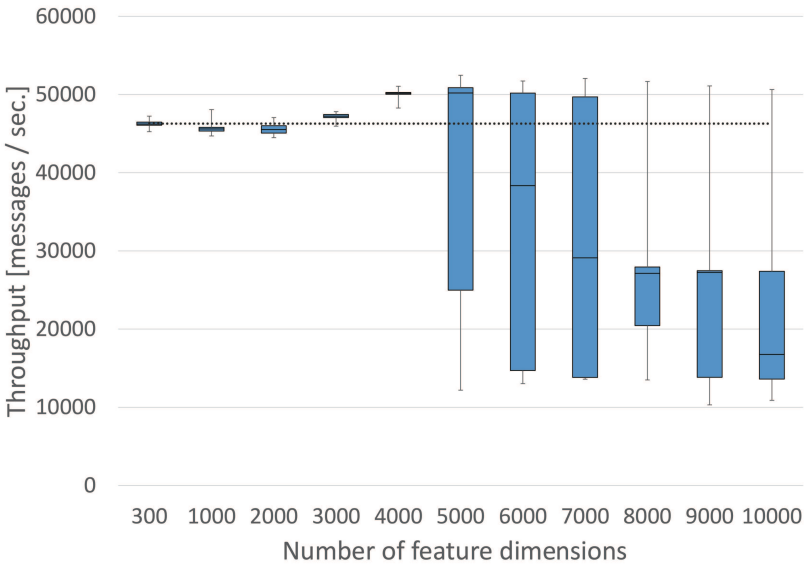

FIGURE 7. Throughput as a function of the number of feature dimensions

5.6. **Analysis of changes in detection accuracy, TNR, and TPR.** We investigated change in detection accuracy during a simulation evaluation:

- Machine learning algorithm: LightGBM
- Number of pre-training data: $X = 100$
- Queue length: $L = 100$
- Number of feature dimensions: $N = 300$

Figure 8 shows changes in detection accuracy, TNR, and TPR from the first to 35th received message; plots at 0% and 100% along the x-axis represent normal and attack class of the message, respectively.

The first two attack messages were missed, but the third attack was detected, then TPR gradually improved, repeatedly detecting and missing attack messages. In contrast, normal messages were never falsely flagged. By the time 3,000 messages had been received, detection accuracy was 99.87%, the true negative rate was 100.0%, and the true positive rate was 99.73%.

In Figure 8, the detection accuracy decreased for the 2nd, 4th, 9th, and 17th attack messages. Each decrease indicates that the adaptive immune function missed the attack message. In these cases, the innate immune function detected the attack, and subsequently, the adaptive immune function learned about the attack. The detection accuracy did not drop after the 18th message, implying that only four learning sessions resulted in the correct classification of all messages from the 18th to the 3,000th message. Moreover, the

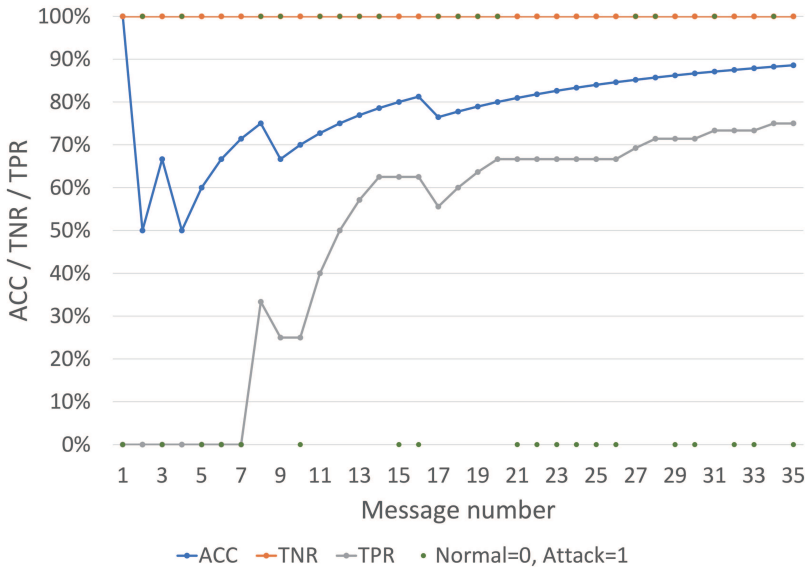

FIGURE 8. Change in performance indices

average time per learning was 54.10 ms, making the total learning time extremely short, even for iterative learning.

The last learning occurred on the 17th message, suggesting that very little supervised data is needed to achieve high detection accuracy. Therefore, ibAD achieved high detection accuracy with very few learning iterations and minimal supervised data. This result indicates that ibAD has achieved high resilience for the Mosquitto broker. This resilience allows the Mosquitto broker to sustain its disaster prevention services without human intervention. In addition to disaster prevention services, this high resilience could play a crucial role in services where human intervention is challenging, such as satellite services in outer space.

5.7. **Analysis of detection time.** In ibAD, when the adaptive immune function incorrectly classifies an attack as normal, the innate immune function detects the attack. Therefore, the time T required for ibAD to detect an unknown attack is composed of the classification time T_{aif} by the adaptive immune function and the detection time T_{aif} by the innate immune function. In the experimental environment described in Section 5.5, T_{aif} averages 9.94 (μs). T_{iif} depends on the vulnerability. For example, T_{iif} was 54.10 ms for CVE-2018-12543 and 53.57 ms for CVE-2019-11779.

Thus, the time required for ibAD to detect an unknown attack for each vulnerability is $T_{CVE\text{-}2018\text{-}12543} = T_{aif} + T_{iif} = 54.11$ ms and $T_{CVE\text{-}2019\text{-}11779} = 53.58$ ms, respectively. The time required for ibAD to detect a learned attack is T_{aif} since the adaptive immune function detects it, i.e., the innate immune function does not need to detect it.

Analysis of detection time confirms that ibAD is capable of high real-time detection. This high real-time performance allows for high throughput.

6. Discussion.

6.1. **Disaster prevention services applicable to ibAD.** As illustrated in Figure 7 of Section 5.5, the throughput of the Mosquitto broker with ibAD dropped when the number of feature dimensions – i.e., the upper limit of the message size – was increased beyond 4,000 bytes. In this context, we discuss the applicable disaster warning services with reference to the throughput results in Figure 7. Table 2 showcases the relationship between

TABLE 2. Relationship between disaster warning services and their key observation data

Warning service	Key observation data
Flood	Water level, rainfall amount, soil moisture, image
Earthquake	Earthquake magnitude, earthquake epicenter, occurrence time
Tsunami	Earthquake magnitude, earthquake epicenter, tsunami height, image
Volcano	Volcanic activity, volcanic earthquakes, infrasound, image
Typhoon	Typhoon name, direction, strength, atmospheric pressure
Landslide	Rainfall amount, soil moisture, slope information, image
High tide	Tide level, wind speed, wind direction, image
Fire	Temperature, humidity, image

disaster warning services and their corresponding key observation data. Observation data can be categorized into numerical data, such as rainfall and water levels; string data, including typhoon names and volcano activity; and media data, such as river images and infrasound waveforms.

The throughput is anticipated to remain largely unaffected by ibAD, given that numerical and string observation data typically fall below the 4,000-byte threshold. This implies that disaster warning services relying on such observation data are likely to experience minimal impact from ibAD. Furthermore, ibAD has broad applicability, as numerous disaster warning services primarily utilize either numerical or string-based key observation data.

Conversely, media data, such as images and audio, frequently surpass 100K bytes, leading to a reduction in the Mosquitto broker's throughput due to ibAD, as illustrated in Figure 7. To alleviate the decrease in throughput, one approach is to extend the interval between message transmissions. For instance, by adjusting the transmission interval of observation data from every second to every minute, the number of messages processed by the Mosquitto broker can be curtailed by approximately a factor of 60. Consequently, the impact of ibAD on throughput would be considerably mitigated.

6.2. **Cyberattacks on the adaptive immune function and workarounds.** Adaptive machine learning introduces potential security risks related to integrity, availability, and privacy [30]. [30] identifies two primary types of cyberattacks on immunity-based detection: Misclassification as normal message. In this scenario, an attacker sends various normal messages resembling an attack message before delivering the actual attack message. Despite the adaptive immune function misclassifying the attack message as normal initially, the innate immune function can detect it. This detection triggers the adaptive immune function to learn from the attack message, enabling the subsequent identification of similar attacks.

Misclassification as attack. This attack involves sending an attack message highly resembling a normal message before the adaptive immune function has a chance to learn the normal message. As a result, normal messages resembling the attack message may be denied by the system.

To mitigate the second type of attack, it is suggested to include all possible messages in the pre-training data. For instance, real IoT services often involve MQTT message topics that are proper nouns representing locations, building names, or other specific identifiers. By adding such normal messages to the pre-training data, the system can enhance its ability to correctly classify and handle messages related to real-world IoT scenarios.

7. **Conclusion.** We proposed a prototype immunity-based attack detection method for the Mosquitto broker specifically focusing on MQTT version 3.1.1. Through simulation evaluations, we identified suitable parameters, and performance evaluations demonstrated high detection accuracy, with a true negative rate of 99.77% and a true positive rate of 99.67%. The method exhibited robust performance, particularly against real vulnerabilities of the Mosquitto broker: CVE-2018-12543 and CVE-2019-11779. In the implementation of the prototype, the evaluation of message throughput by the Mosquitto broker revealed minimal overhead, especially when the number of feature dimensions was below approximately 4,000 in our experimental environment.

As part of our future work, we aim to extend the capabilities of the proposed method to address vulnerabilities specific to MQTT version 5, such as CVE-2019-11778. This ongoing research seeks to further enhance the resilience of MQTT brokers, making them more secure and reliable in the face of potential cyber threats.

Acknowledgment. This work was supported by JSPS Grant-in-Aid for Scientific Research JP20K05012. We thank David Price for proofreading.

REFERENCES

[1] T. Okamoto, An immunity-enhancing security module for cloud servers, *International Journal of Innovative Computing, Information and Control*, vol.16, no.1, pp.137-151, 2020.
[2] OASIS Standard, *MQTT Version 3.1.1*, http://docs.oasis-open.org/mqtt/mqtt/v3, 2014.
[3] Z. Shelby, K. Hartke and C. Bormann, *RFC 7252: The Constrained Application Protocol (CoAP)*, Internet Engineering Task Force, 2014.
[4] A. L. Buczak and E. Guven, A survey of data mining and machine learning methods for cyber security intrusion detection, *IEEE Commun. Surv. Tutor.*, vol.18, no.2, pp.1153-1176, 2016.
[5] R. A. R. Ashfaq, X. Z. Wang, J. Z. Huang, H. Abbas and Y. L. He, Fuzziness based semi-supervised learning approach for intrusion detection system, *Information Sciences*, vol.378, pp.484-497, 2017.
[6] M. Lopez-Martin, B. Carro and A. Sanchez-Esguevillas, Application of deep reinforcement learning to intrusion detection for supervised problems, *Expert Systems with Applications*, vol.141, 112963, 2020.
[7] A. Omotosho, Y. Qendah and C. Hammer, IDS-MA: Intrusion detection system for IoT MQTT attacks using centralized and federated learning, *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp.678-688, 2023.
[8] S. Zavrak and M. Iskefiyeli,, Anomaly-based intrusion detection from network flow features using variational autoencoder, *IEEE Access*, vol.8, pp.108346-108358, 2020.
[9] M. Mohammadi, A. Akbari, B. Raahemi, B. Nassersharif and H. Asgharian, A fast anomaly detection system using probabilistic artificial immune algorithm capable of learning new attacks, *Evolutionary Intelligence*, vol.6, pp.135-156, 2014.
[10] Y. Al-Nashif, A. A. Kumar, S. Hariri, Y. Luo, F. Szidarovsky and G. Qu, Multi-level intrusion detection system (ML-IDS), *2008 International Conference on Autonomic Computing*, pp.131-140, 2008.
[11] M. Danforth, WCIS: A prototype for detecting zero-day attacks in web server requests, *Proc. of USENIX LISA E1*, 2011.
[12] S. A. Hofmeyr et al., Intrusion detection using sequences of system calls, *Journal of Computer Security*, vol.6, no.3, pp.151-180, 1998.
[13] W. Haider et al., Detecting anomalous behavior in cloud servers by nested arc hidden SEMI-Markov model with state summarization, *IEEE Transactions on Big Data*, vol.5, no.3, pp.305-316, 2017.
[14] M. Abadi et al., Control-flow integrity principles, implementations, and applications, *ACM Transactions on Information and System Security*, vol.13, no.1, pp.1-40, 2009.
[15] Intel Corp., *Control-Flow Enforcement Technology Specification*, Revision 3.0, Intel Corp., 2019.
[16] M. Rahman et al., Origin-sensitive control flow integrity, *Proc. of USENIX Security 19*, pp.195-211, 2019.
[17] S. Canakci, L. Delshadtehrani, B. Zhou, A. M. Joshi and M. Egele, Efficient context-sensitive CFI enforcement through a hardware monitor, *Detection of Intrusions and Malware, and Vulnerability Assessment*, vol.12223, pp.259-279, 2020.

[18] E. D. Dileesh and A. P. Shanthi, An application specific dynamic behavior model using function-call sequence and memory access-graph for execution integrity verification, *Comput. Secur.*, vol.107, 102299, 2021.
[19] A. Saidane et al., The design of a generic intrusion-tolerant architecture for web servers, *IEEE Transactions on Dependable and Secure Computing*, vol.6, no.1, pp.45-58, 2009.
[20] I. Winarno, Y. Ishida and T. Okamoto, A performance evaluation of resilient server with a self-repair network model, *Mobile Networks and Applications*, vol.24, pp.1095-1103, 2018.
[21] J. Zheng et al., Security evaluation of a VM-based intrusion-tolerant system with pull-type patch management, *Proc. of 2019 IEEE HASE 19*, pp.156-163, 2019.
[22] T. Okamoto, F. Sano, I. Winarno, Y. Hata and Y. Ishida, Implementation and evaluation of an intrusion-resilient system for a DNS service, *International Journal of Innovative Computing, Information and Control*, vol.13, no.5, pp.1735-1750, 2017.
[23] M. Garcia et al., OS diversity for intrusion tolerance: Myth or reality?, *Proc. of DSN 2011*, pp.383-394, 2011.
[24] Wikipedia Contributors: Comparison of MQTT Implementations, *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Comparison_of_MQTT_implementations, 2022.
[25] P. Wagle et al., Stackguard: Simple stack smash protection for GCC, *Proc. of the GCC Developers Summit*, pp.243-255, 2003.
[26] NIST, *CVE-2018-12543*, https://nvd.nist.gov/vuln/detail/CVE-2018-12543, 2018.
[27] NIST, *CVE-2019-11779*, https://nvd.nist.gov/vuln/detail/CVE-2019-11779, 2019.
[28] G. Ke et al., LightGBM: A highly efficient gradient boosting decision tree, *Advances in Neural Information Processing Systems*, pp.3146-3154, 2017.
[29] R. Banno, K. Ohsawa, Y. Kitagawa, T. Takada and T. Yoshizawa, Measuring performance of MQTT v5.0 brokers with MQTTLoader, *Proc. of 2021 IEEE CCNC*, 2021.
[30] L. Huang et al., Adversarial machine learning, *Proc. of the 4th ACM Workshop on Security and Artificial Intelligence*, pp.43-58, 2011.

Author Biography

Takeshi Okamoto received his Ph.D. degree in Engineering from Toyohashi University of Technology, Japan, in 2002. Prof. Okamoto is currently a full-time professor at the Department of Information Network and Communication, Faculty of Information Technology, Kanagawa Institute of Technology. His main research interests include machine learning applications to security and malware analysis.