Token-Based Authentication Monitoring System

Pattharadanai Rujichaikul and Ittipon Rassameeroj*

Faculty of Information and Communication Technology, Mahidol University, Thailand
E-mail: pattharadanai.ruj@alumni.mahidol.ac.th; ittipon.ras@mahidol.ac.th
**Corresponding Author*

Received 29 November 2024; Accepted 19 July 2025

Abstract

In modern web applications, token-based authentication has become a crucial mechanism for securing access to protected resources. JSON Web Tokens (JWTs), in particular, are widely adopted due to their stateless and scalable nature. However, this reliance makes tokens a prime target for attackers, with incidents of token theft and misuse via techniques such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and token hijacking on the rise. Existing security solutions like IDS and application firewalls are not designed to effectively detect token-specific attack patterns, leaving a critical security gap in modern authentication systems. To address this problem, we propose a Token-based Authentication Monitoring System capable of detecting, tracking, analyzing, and investigating suspicious token behaviors in real time. Our research focuses on JWT-based access tokens and the refresh token technique in OAuth 2.0 environments. A core contribution of this work is the design of 25 specialized detection rules based on patterns. We validated the proposed system through 70 comprehensive test cases covering both normal and attack scenarios. The system achieved an overall detection accuracy of 81.4%, demonstrating its capability to effectively detect token-related attacks overlooked by conventional defenses. Additionally, we evaluated the system's

Journal of Cyber Security and Mobility, Vol. 14_4, 777–798.
doi: 10.13052/jcsm2245-1439.1441

performance, measuring detection latency and operational overhead in a real-world integration scenario. The results confirmed that the monitoring system delivers real-time detection with minimal impact on application responsiveness and system resources. This research offers a practical, adaptable framework that enhances the security of any system employing token-based authentication, reducing the risk of unauthorized access while maintaining system performance.

Keywords: JSON web token, token-based authentication, rule-based detection, OAuth 2.0, monitoring system.

1 Introduction

Token-based authentication is widely used in many web applications [1–3]. It offers a stateless, scalable, and decentralized approach to securing access to protected resources, replacing traditional session-based mechanisms. Among the available token formats, JSON Web Token (JWT) is one of the most popular options for implementing access tokens due to its self-contained and compact design [4]. Additionally, the OAuth 2.0 framework often incorporates JWTs alongside refresh tokens to enable secure token renewal without requiring users to re-authenticate frequently [5]. However, this widespread adoption has made tokens a primary target for attackers. Techniques such as Cross-Site Scripting (XSS) can be used to steal tokens from client-side storage, while Cross-Site Request Forgery (CSRF) exploits automatically included credentials like cookies to perform unauthorized actions on behalf of users [6]. Additionally, sophisticated attacks such as token hijacking, unauthorized token reuse, and malicious token generation pose serious security threats in token-based authentication systems. Critically, because token-based authentication is inherently stateless, once an access token has been compromised, it typically cannot be revoked from the server side and remains valid until it expires. This means attackers can misuse stolen tokens without detection. Moreover, in most current implementations, there is no mechanism to track token usage or inspect whether a token has been compromised.

Intrusion Detection Systems (IDS) and Web Application Firewalls (WAF) are widely deployed security solutions that help protect web applications by preventing a broad range of attacks. However, they are not specifically designed to detect token-specific attacks [7–10]. Their detection models generally focus on network-layer or HTTP request anomalies rather than the contextual behavior of tokens within authenticated sessions. Furthermore,

while some existing solutions monitor token expiration or lifetime rules, they do not adequately analyze token usage patterns across multiple requests and endpoints or correlate context-specific information such as IP address and User-Agent consistency. Given the increasing importance of token-based authentication in modern web applications and the growing prevalence of token-targeted attacks, there is currently no comprehensive, real-time monitoring solution specifically designed to detect, track, and investigate suspicious behaviors associated with token usage. Existing security tools like IDS and WAF lack the ability to correlate contextual token information, leaving systems vulnerable to token hijacking, misuse, and unauthorized access, as discussed in Section 2.6. Although prior research and practical implementations have explored token storage security and client-side protections, these approaches are insufficient once a token is compromised. Moreover, no existing, widely-adopted solutions systematically monitor token behaviors at the infrastructure level using a rule-based detection mechanism tailored for token-based authentication environments. This gap leaves modern web applications exposed to sophisticated token-based attack scenarios that conventional security tools overlook. Additionally, little attention has been paid to evaluating the system performance overhead and detection latency of token monitoring solutions in real-time environments, which is a crucial consideration for deployment in production systems.

To address this issue, we propose a Token-based Authentication Monitoring System designed to detect, monitor, track, and analyze token attack behaviors in real time. Our research specifically focuses on JWT access tokens [4] and OAuth 2.0 refresh tokens [5]. The system applies an algorithm based on 25 detection rules that identify anomalies such as IP address and User-Agent changes, simultaneous token use from multiple locations, and other suspicious token behaviors. Risk levels for each detection event are classified into four categories: low, moderate, high, and critical to support appropriate incident response decisions. Additionally, we designed the system to be lightweight and efficient, ensuring minimal performance overhead and low detection latency, making it suitable for integration with high-traffic, real-world applications.

The remainder of this paper is organized as follows: Section 2 provides the background, laying the foundation for understanding the core concepts and context of this research. Section 3 reviews related work, highlighting existing studies and identifying the research gaps this work addresses. Section 4 outlines the methodology, detailing the approaches and processes employed in this study. Section 5 presents the implementation and results,

demonstrating the practical application of the methodology and discussing the findings. Section 6 concludes the paper with a summary of the contributions and potential future directions. Finally, Section 7 acknowledges the organization that supported this work.

2 Background

This section presents the theoretical background related to our research. It reviews key concepts, frameworks, and existing technologies in token-based authentication, exiting solutions like IDS and WAF and risk assessment methodologies, providing the foundation for the design and development of our proposed monitoring system.

2.1 Token-based Authentication

In this approach, tokens such as JSON Web Tokens (JWT) are stored on the client side and are not maintained on the server after issuance. As a result, when a user logs out or if a token becomes compromised, the server lacks the ability to revoke that token directly. This limitation introduces a significant security challenge, as malicious actors who obtain a valid token can continue to access protected resources until the token naturally expires.

2.2 JSON Web Token (JWT)

A central focus of our study is the monitoring of JWTs [4], which are widely used as access tokens in token-based authentication systems. A JWT is composed of three parts: the header, payload, and signature, each separated by a dot (.). The header specifies the token type, and the algorithm used to generate the signature. The payload contains the authorization data and any other claims, typically representing the user's identity and access permissions. Both the header and payload are structured as JSON objects and encoded using Base64Url. The signature is generated by combining the encoded header and payload with a secret key known only to the communicating parties, using the algorithm declared in the header. This ensures the token's integrity and authenticity.

2.3 JSON Web Token Security Challenges

In this research, our objective is to study, design, and implement detection rules aimed at identifying suspicious behaviors that attempt to exploit tokens,

particularly JWTs. Various attack techniques and scenarios targeting JWTs have emerged in recent years, posing significant risks to token-based authentication systems. It is crucial for web developers and security practitioners to be aware of these threats and to implement JWT securely and appropriately to minimize the risk of token compromise and unauthorized use. Our proposed detection rules are intended to fill this critical security gap by identifying abnormal token usage patterns and potential attack behaviors in real time.

(1) Misconfiguration

A standard JWT consists of three components: the header, payload, and signature. However, JWTs can be misconfigured to exclude the signature entirely by setting the algorithm specified in the header to "none". In this case, the signature part is omitted, and the server does not perform any signature verification. This dangerous configuration allows an attacker to modify the token's payload freely, as the server will continue to accept the tampered token without validating its integrity.

(2) Library Vulnerabilities

Some JWT libraries introduce vulnerabilities by allowing multiple verification algorithms or skipping signature validation. A common attack involves changing the algorithm from "RS256" to "HS256", then signing a forged token using the RSA public key often exposed in client-side code. This tricks the server into accepting a malicious token. Such issues occur when libraries rely on the token's header to determine the verification algorithm, which is unsafe since headers are user-controlled. Additionally, libraries that accept unsigned tokens enable attackers to alter payload data for privilege escalation or impersonation.

(3) Using a weak secret key

A weak JWT secret key is vulnerable to dictionary and brute-force attacks. Since HMAC uses the same key for signing and verification, if attackers obtain it, they can forge, modify, and sign tokens, compromising authentication and authorization.

(4) Reusing the same key across services

Reusing the same secret key across multiple services poses a security risk, as a valid token for one service could be accepted by others, leading to unauthorized access. To mitigate this, unique keys should be used per service.

If shared keys are unavoidable, developers must implement and validate the audience (aud) claim to restrict token usage to its intended service.

(5) The token lifetime

JWT is inherently stateless, which means the server cannot track or revoke tokens once they are issued. If compromised, a token remains valid until it expires. Short lifetimes improve security but require frequent logins, while long lifetimes increase exposure risk.

(6) Verifying the token on the client side

Client-side verification exposes the secret key in symmetric algorithms, enabling attackers to forge valid tokens and compromise authentication.

(7) Misuse

Sensitive data should not be stored in the JWT payload, as both the header and payload are Base64Url-encoded and easily decoded, exposing any embedded information.

(8) Irrevocability of tokens

JWTs are inherently stateless. Once issued, they cannot be revoked by the server. If stolen, a token remains usable by an attacker until it expires, posing a continuous security risk.

(9) Cross-Site Scripting (XSS) attacks

XSS vulnerabilities enable attackers to inject malicious scripts into web pages, allowing them to steal JWTs stored in web storage or cookies from users who access the compromised page.

(10) Cross-Site Request Forgery (CSRF)

CSRF vulnerabilities allow attackers to perform unauthorized actions on behalf of authenticated users by exploiting their JWTs. Tokens stored in cookies are susceptible, as cookies are sent automatically with each request, while JWTs stored in web storage are not exposed to this risk.

(11) Man-in-the-Middle (MITM) Attack

If JWTs are transmitted over insecure channels, attackers can intercept or tamper with them. To mitigate this risk, all requests carrying sensitive data should be sent exclusively over encrypted connections, such as HTTPS, to ensure confidentiality and integrity.

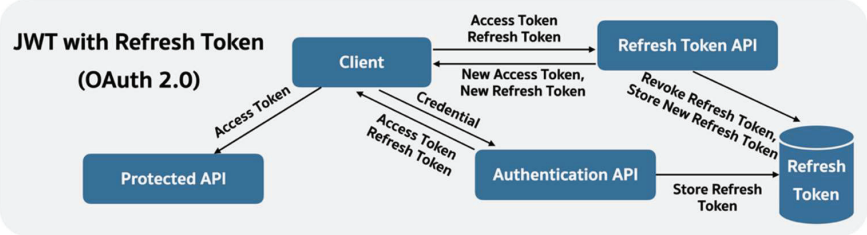

Figure 1 Workflow of JWT authentication with refresh token.

2.4 JSON Web Token with Refresh Token

As many token-based authentication systems implement JWTs alongside refresh tokens, this study also examines their use and management. Refresh tokens (OAuth 2.0) [5] address key challenges such as short access token lifetimes.

Figure 1 illustrates the workflow of JWT authentication combined with a refresh token mechanism. Initially, the client submits credentials to the authentication API, which responds with an access token (JWT) and a refresh token (random string). The access token is used to access protected APIs until it expires. Once expired, the client sends both the expired access token and the refresh token to the refresh token API to obtain new tokens.

The refresh token API then generates a new access token and a new refresh token, storing the new refresh token in a dedicated database while revoking the previous one by updating its status. The new tokens are returned to the client, who continues to access protected APIs with the new access token. This token renewal cycle continues until the client becomes inactive. If the refresh token expires, the user must re-authenticate to obtain new tokens.

Typically, access tokens are stored in web storage (local storage or session storage), while refresh tokens are securely stored in cookies to mitigate security risks.

2.5 OWASP Risk Rating Methodology

The OWASP Risk Rating Methodology [11] is a standardized framework for assessing potential security risks in software applications. It provides a structured approach for evaluating and prioritizing risks. In this research, we adopt this methodology to assess the potential risks associated with token-based attacks and to assign appropriate severity levels to the detection rules accordingly.

2.6 Exiting Solutions

An Intrusion Detection System (IDS) monitors system and network activities to detect unauthorized access or malicious behavior. Common IDS techniques include (1) Signature-based detection, which triggers alerts by matching predefined attack patterns (2) Anomaly-based detection, which identifies deviations from normal system behavior and (3) Hybrid detection, combining both to enhance detection accuracy.

However, common IDSs primarily operate at the network and transport layers, analyzing packet headers and payloads transmitted over networks. These systems typically lack visibility into application-layer authentication mechanisms, such as JWTs, especially when tokens are stored in HTTP-only cookies or transmitted over encrypted HTTPS connections. As a result, conventional IDS solutions cannot inspect or analyze JWTs to detect token-specific attacks. Similarly, Web Application Firewalls (WAFs) operate at the application layer and can detect certain injection or protocol violation patterns. While WAFs can enforce access controls and filter common web-based attacks, they are generally not designed to monitor or track token usage behaviors over time. Most WAFs assess each request independently without maintaining a historical context of token usage across sessions or IP addresses, which limits their effectiveness in detecting token hijacking, misuse, or anomaly patterns related to token behaviors.

To address these limitations, we propose a token-based authentication monitoring system operating at the application layer within the authentication system itself, where JWTs are generated, validated, and utilized. This positioning allows full visibility into token contents and usage patterns, even for tokens transmitted over secure channels or stored in HTTP-only cookies. Moreover, we adopted a signature-based IDS approach because token-based attacks often follow recognizable patterns (e.g., tokens reused across multiple IPs), enabling effective detection through predefined rules without the need for extensive training. This approach lowers operational overhead compared to anomaly-based systems, which require continuous profiling and adaptive baselining. Additionally, by targeting explicit attack signatures, it significantly reduces false positives in dynamic web environments, focusing detection on high-confidence, verifiable threats. In addition, we developed 25 signature-based detection rules specifically designed to identify malicious request patterns and abnormal token usage behaviors. These rules focus on key indicators such as IP address inconsistencies, User-Agent anomalies, unusual token lifetimes, refresh token misuse that common IDS and WAF solutions cannot reliably detect.

Table 1 Impact of our work compared to common IDSs and WAFs

Features	Common IDSs	WAFs	Our Monitoring System
Operates at application layer (JWT-aware)	×	✓ (partial)	✓
Detects token hijacking via IP/User-Agent anomalies	×	×	✓
Detects misuse of refresh tokens	×	×	✓
Handles token-specific attack patterns (e.g., forged secret key)	×	×	✓
Requires profiling or learning	Varies (for anomaly-based)	×	×
Rule-based, low overhead detection	✓ (signature-based only)	✓ (basic rules)	✓

Table 1 illustrates that while common IDSs and WAFs provide valuable protections, they are insufficient for addressing the unique security challenges of token-based authentication. By operating at the application layer and incorporating targeted, signature-based rules, our system effectively detects token-specific attack behaviors overlooked by existing solutions.

3 Related Work

At present, JWT is essential for the authentication process in several web applications and mobile applications. It is widely utilized across various industries and organizations to ensure the security of their systems against unauthorized access. For example, [1] they utilized OAuth 2.0 and JWT in healthcare services to enhance security and performance. Moreover, they evaluated the performance by comparing the security and efficiency of the existing scheme (OAuth 2.0) with the proposed scheme (OAuth 2.0 and JWT). The result of their experiment indicates that the proposed scheme (OAuth 2.0 and JWT) is effective in enhancing security and performance in healthcare services. In conclusion, their experiment indicates that JWT can be utilized to enhance security and performance. [2] They utilized JWT in the SIKASIR RESTful web service by replacing server-based authentication with token-based authentication using JWT, enabling their web service to be accessed on multiple platforms. As a result, the SIKASIR RESTful web service can be accessed through various platforms, such as mobile devices

and web applications. Their research indicates that JWT can be utilized not only for the authentication process but also to support multiple platforms. [3] They utilized JWT in their RESTful web service on the Unklab Information System to ensure that only authorized users can access specific data. Consequently, they can provide their service exclusively to authorized users by using JWT for the authentication process. Their research indicates that JWT can be utilized to secure data from unauthorized access.

In addition, JWT is utilized not only for authentication in web applications and services but also in other systems, such as applications in Software-Defined Networking (SDN). [12] they utilized JWT for applications in SDN and proposed an approach to enhance JWT security by using a dynamic secret key instead of a static secret key. The dynamic key is determined based on the user's password hash value, ensuring that whenever the user changes their password, all active tokens become invalid. Even if a single token is compromised, the user can reset their password, which invalidates all valid tokens and generates new ones based on the updated password. Their research proposed this approach to address the issue of compromised tokens and enhance JWT security. Although JWT is widely utilized for authentication, there are other security challenges associated with it that must be addressed. In particular, the JWT security challenges discussed in Section 2.3 require special attention. There are some research studies that identify solutions. For example, [6] they performed penetration testing to exploit the JWT stored in cookie storage by using a CSRF attack on a target system's vulnerability. The CSRF technique employed in this research successfully utilized JWT tokens stored in cookies to send forged requests. As a result, the victim's account was compromised, and the resource was taken over. In conclusion, this vulnerability can lead to account takeover. Furthermore, their research illustrates how to address a specific example of JWT security challenges.

The selection of signing algorithms for JWT is one of the factors contributing to the system's security, as demonstrated in Section 2.3. However, the selection of signing algorithms for JWT also impacts performance, as discussed in [13]. Their research evaluated the signing algorithms among three options (HMAC, RSA, and ECDSA) by comparing their performance based on parameters such as token generation speed, token size, and data transfer speed. The experimental results showed that the HMAC algorithm performed excellently. This implies that when selecting signing algorithms for JWT, it is essential to consider both security and performance aspects.

Although JWT is widely utilized across various industries and organizations, it faces security challenges, as demonstrated in Section 2.3. This

is why it is essential to monitor the token to ensure it is used by legitimate users and has not been compromised. To address this, an intrusion detection system (IDS) is considered, as it is a system designed to monitor intruders. Several research studies have developed IDS solutions for web applications. For example, [7] they proposed an application intrusion detection system capable of detecting and preventing web application attacks in real time. The system includes a mechanism to address input validation attacks and can also mitigate other types of attacks, such as invalidated redirects and forwards. [10] They used a deep learning approach for their intrusion detection system. Different deep learning techniques were applied to determine which algorithm would provide the highest accuracy. The algorithms they used included Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Multilayer Perceptron (MLP). Their experimental results showed that the Multilayer Perceptron (MLP) achieved better performance in terms of accuracy, reaching approximately 90%. However, IDS can employ various techniques to identify suspicious behavior. [8] They conducted research on user behavior-based intrusion detection using statistical techniques to determine whether user behavior on host-based GUI systems is normal or abnormal. They analyzed various user behavior parameters, such as resource access and usage, as well as the frequency of input device usage, including keyboard and mouse access. Moreover, they applied simple aggregation measures and logistic regression methods to user behavior logs. Their experimental results showed that the statistical mean method performs well in detecting unauthorized users, whereas logistic regression is more effective in identifying authorized users. [9] They proposed a signature-based intrusion detection system based on user behavior using a pattern-matching technique. In their study, they used the same dataset as that used in [8]. The detection rules were derived from expert knowledge. Their experimental results demonstrated that the pattern-based intrusion detection model achieved 75% accuracy. Furthermore, the pattern-based technique achieved 100% accuracy in identifying "unauthorized" users and a 5:4 ratio in identifying "authorized" users. Based on this dataset, the signature-based intrusion detection method outperformed the anomaly-based intrusion detection method in terms of accuracy. For our research, we aim to apply a signature-based intrusion detection method to identify suspicious behaviors attempting to exploit tokens. This approach is motivated by our understanding of patterns associated with token attack behaviors. Consequently, we have developed algorithms and rules capable of detecting these token attack behaviors. Furthermore, we strive to design

algorithms and rules that address the challenges in JWT security, as discussed in Section 2.3.

4 Methodology

This section outlines the methodologies employed in this research. It details the system architecture, the request logs, and the design of detection rules, which collectively support the implementation and evaluation of the proposed token-based monitoring system.

4.1 System Architecture

This research proposes a monitoring system that applies predefined detection rules to identify, track, and analyze suspicious token usage behaviors in real time. The system determines whether tokens are being used legitimately and displays active refresh tokens, allowing administrators to promptly revoke tokens associated with compromised accounts. When suspicious activity is detected, the system generates alerts, enabling administrators to take immediate action and conduct further investigation.

Figure 2 illustrates the architecture of the proposed monitoring system integrated with a web API using JWT authentication. The monitoring system

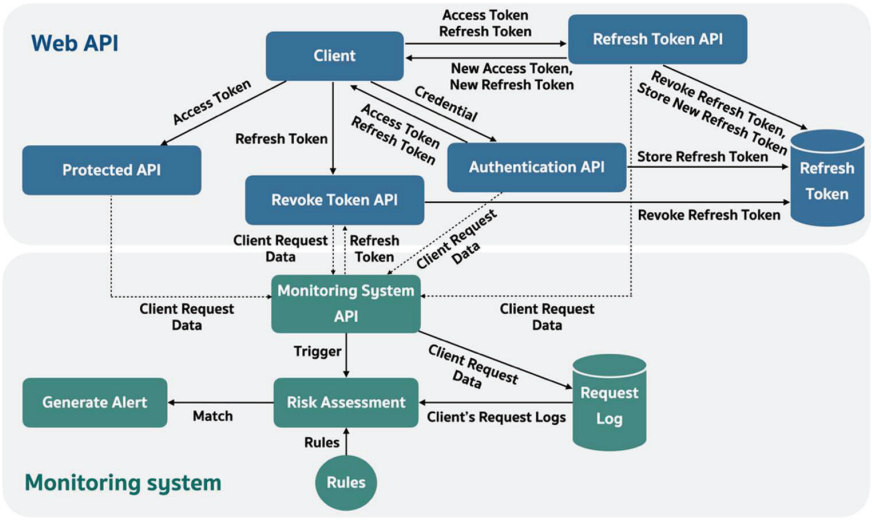

Figure 2 The architecture of the proposed monitoring system integrated with a web API using JWT authentication.

is designed to interface with any web API implementing JWT-based authentication. When a client requests access to a protected API, the process begins with user authentication via the authentication API. Upon successful login, the authentication API issues an access token (JWT) containing user information and a refresh token (random string), which is stored in a refresh token database. The client stores the access token in web storage (local/session storage) and the refresh token in cookies.

Subsequent client requests to protected APIs include the access token, while the associated request data is simultaneously sent to the monitoring system API. When the access token expires, the client submits both the expired token and refresh token to the refresh token API, which generates new tokens, revokes the old refresh token by updating its status in the database, and forwards the updated request data to the monitoring system.

If a user logs out, the refresh token is explicitly sent to the revoke token API, which invalidates it by updating its status to "inactive". Throughout these interactions, the monitoring system collects request data, storing it in a request log database and invoking a risk assessment function. This function evaluates incoming request logs against the predefined detection rules. If a match is found, an alert is triggered, notifying administrators to investigate and, if necessary, revoke tokens through the monitoring system, which logs this action for auditing purposes.

This architecture enables real-time token activity tracking, proactive detection of suspicious behavior, and immediate incident response capabilities.

4.2 Request Log

When a client sends requests to the web API, request data such as IP address, User-Agent, and other attributes are forwarded to the monitoring system. These data points are stored in a request log database and analyzed to detect suspicious or malicious behavior targeting the application and its APIs. In this research, predefined detection rules are applied to the request logs. If a pattern matches any rule, an alert is triggered, allowing the system to identify token-related attack behaviors in real time.

4.3 Rules Design

A key outcome of this research is the development of an algorithm and 25 detection rules for identifying token attack behaviors in JWT-based authentication systems. These rules, integrated into our monitoring system,

classify risks into 4 severity levels following the OWASP Risk Rating Methodology [11]. The design was guided by a comprehensive analysis of JWT security challenges, documented attack scenarios, OWASP guidelines, public advisories, and real-world case studies. We conducted in-depth reviews of known JWT vulnerabilities (as detailed in Section 2.3), analyzed public security reports, and simulated attack patterns in a controlled environment to observe behavioral indicators within request data. From this, recurrent patterns such as token theft, replay, misuse across services, brute-force attempts, and unauthorized token generation were identified and formalized into detection rules. Most rules utilize client metadata (IP address, User-Agent) and token activity patterns to detect anomalies at various API endpoints, while Rule 25 uniquely verifies token issuance integrity to identify forged tokens. Finally, the 25 detection rules were organized into 8 groups based on API interactions, client metadata, and behavioral patterns.

Group 1: Rules 1–3 detect inconsistencies in client metadata (IP address, user-agent) between the authentication/refresh token APIs (endpoint 1) and the protected API (endpoint 2) to identify potential token misuse or unauthorized access.

- Rule 1: If the IP address that accessed the protected API is different from the authentication API or the refresh token API, the alert will be generated. The risk level is Moderate.
- Rule 2: If the user-agent that accessed the protected API is different from the authentication API or the refresh token API, the alert will be generated. The risk level is High.
- Rule 3: If the IP address and user-agent that accessed the protected API are different from the authentication API or the refresh token API, the alert will be generated. The risk level is Critical.

Group 2: Rules 4–6 detect inconsistencies in client metadata (IP address, user-agent) between the Authentication API (endpoint 1) and the refresh token API (endpoint 2), indicating potential misuse or unauthorized activity during token refresh operations.

- Rule 4: If the IP address that accessed the refresh token API is different from the authentication API, the alert will be generated. The risk level is Low.
- Rule 5: If the user-agent that accessed the refresh token API is different from the authentication API, the alert will be generated. The risk level for this rule is High.

- Rule 6: If the IP address and user-agent that accessed the refresh token API are different from the authentication API, the alert will be generated. The risk level for this rule is Critical.

Group 3: Rules 7–12 detect concurrent or sequential use of the same access token from different IP addresses and/or user-agents at the protected API (endpoint 1), identifying potential token theft or session hijacking.

- Rule 7: If the same access token is used by different IP addresses to access the protected resource API at the same time, the alert will be generated. The risk level is Critical.
- Rule 8: If the same access token is used by different IP addresses to access the protected resource API at different times, the alert will be generated. The risk level is Low.
- Rule 9: If the same access token is used by different user-agents to access the protected resource API at the same time, the alert will be generated. The risk level is High.
- Rule 10: If the same access token is used by different user-agents to access the protected resource API at different times, the alert will be generated. The risk level is Moderate.
- Rule 11: If the same access token is used by different IP addresses and user-agents to access the protected resource API at the same time, the alert will be generated. The risk level is Critical.
- Rule 12: If the same access token is used by different IP addresses and user-agents to access the protected resource API at different times, the alert will be generated. The risk level is High.

Group 4: Rules 13–15 detect multiple active refresh tokens associated with the same user account but accessed from different IP addresses and/or user-agents at the authentication API (endpoint 1), indicating possible account compromise.

- Rule 13: If there is more than one refresh token that is active at the same time and authenticated by different IP addresses, the alert will be generated. The risk level is Moderate.
- Rule 14: If there is more than one refresh token that is active at the same time and authenticated by different user-agents, the alert will be generated. The risk level is Low.
- Rule 15: If there is more than one refresh token that is active at the same time and authenticated by different IP addresses and user-agents, the alert will be generated. The risk level is High.

Group 5: Rules 16–18 detect simultaneous usage of multiple active access tokens for the same user account at the protected API (endpoint 1) from different IP addresses and/or user-agents, suggesting abnormal or unauthorized activity.

- Rule 16: If there is more than one active access token that is used to access the protected API at the same time by different IP addresses, the alert will be generated. The risk level is High.
- Rule 17: If there is more than one active access token that is used to access the protected API at the same time by different user-agents, the alert will be generated. The risk level is Low.
- Rule 18: If there is more than one active access token that is used to access the protected API at the same time by different IP addresses and user-agents, the alert will be generated. The risk level is Critical.

Group 6: Rules 19–21 detect multiple active refresh tokens being used concurrently at the refresh token API (endpoint 1) from different IP addresses and/or user-agents, indicating suspicious token refresh behavior.

- Rule 19: If there is more than one active refresh token that is used to access the refresh token API at the same time by different IP addresses, the alert will be generated. The risk level is High.
- Rule 20: If there is more than one active refresh token that is used to access the refresh token API at the same time by different user-agents, the alert will be generated. The risk level is Low.
- Rule 21: If there is more than one active refresh token that is used to access the refresh token API at the same time by different IP addresses and user-agents, the alert will be generated. The risk level is Critical.

Group 7: Rules 22–24 detect changes in client metadata (IP address, user-agent) between successive requests to the refresh token API (endpoint 1), signaling possible token misuse or session hijacking.

- Rule 22: If the IP address that accessed the refresh token API is different from the last time, the alert will be generated. The risk level is Low.
- Rule 23: If the user-agent that accessed the refresh token API is different from the last time, the alert will be generated. The risk level is High.
- Rule 24: If the IP address and user-agent that accessed the refresh token API are different from the last time, the alert will be generated. The risk level is Critical.

Group 8: Rule 25 detects the use of an access token at the protected API (endpoint 1) that was not issued by the system's authentication or refresh token APIs, indicating unauthorized token generation or forgery.

- Rule 25: If the access token is not generated from the authentication API or the refresh token API, and it is used to access the protected API, the alert will be generated. The risk level is Critical.

Table 2 presents comparison of the 25 detection rules' features, categorized into 8 groups based on API endpoints, client metadata (IP address, user-agent), token usage timing, the number of active tokens, and token generation sources.

5 Implementation and Result

This section describes the implementation of the monitoring system, its integration with a token-based authentication environment represented by the student diary system, and the experimental evaluation conducted to assess detection accuracy and system performance. The results from validating the proposed detection rules and operational efficiency are subsequently presented.

5.1 Student Diary System

To support this research, a token-based authentication system was implemented through the development of a student diary application, integrated with the monitoring system for evaluation purposes. A dedicated database was deployed to manage refresh tokens, diary entries, and user information. The student diary system replicates the architecture of a typical token-based authentication environment, providing a practical platform for testing and validating the proposed detection rules and algorithms.

5.2 Monitoring System

The monitoring system applies the proposed detection algorithms and rules, with a dedicated database for storing request logs, alerts, configurations, rule definitions, and user data. It facilitates real-time risk assessment, enabling web administrators to monitor, respond to, and mitigate security incidents. The system interface allows administrators to review alerts, active sessions, and request logs, manage refresh token revocation, and configure system parameters, such as refresh token lifetimes and private IP exclusions.

Upon receiving a client request, the monitoring system API records the request data and invokes the risk assessment function, which analyzes historical request logs against predefined detection rules. If a match is

Table 2 Comparison of rules' features

Group	Rule	First API	Second API	IP	User-Agent	Remark
1	1	Authentication, Refresh token	Protected	✓		–
	2	Authentication, Refresh token	Protected		✓	–
	3	Authentication, Refresh token	Protected	✓	✓	–
2	4	Authentication	Refresh token	✓		–
	5	Authentication	Refresh token		✓	–
	6	Authentication	Refresh token	✓	✓	–
3	7	Protected	–	✓		same token, same time
	8	Protected	–	✓		same token, different time
	9	Protected	–		✓	same token, same time
	10	Protected	–		✓	same token, different time
	11	Protected	–	✓	✓	same token, same time
	12	Protected	–	✓	✓	same token, different time
4	13	Authentication	–	✓		Concurrent active refresh tokens.
	14	Authentication	–		✓	
	15	Authentication	–	✓	✓	
5	16	Protected	–	✓		Concurrent use of active access tokens.
	17	Protected	–		✓	
	18	Protected	–	✓	✓	
6	19	Refresh token	–	✓		Concurrent use of multiple active refresh tokens.
	20	Refresh token	–		✓	
	21	Refresh token	–	✓	✓	
7	22	Refresh token	Refresh token	✓		–
	23	Refresh token	Refresh token		✓	–
	24	Refresh token	Refresh token	✓	✓	–
8	25	Protected	–	–	–	Access token not issued by the system.

found, the alert generation function is triggered, producing an alert to notify administrators of suspicious activity for further action.

5.3 Evaluation Result

We developed 25 detection rules to identify token-related attack behaviors. For evaluation, the monitoring system was integrated with the student diary system, where simulated token attacks were executed to verify the accuracy and functionality of each rule. We conducted 70 test cases covering all of our rules as well as additional situations that could cause false positives and false negatives, such as the use of VPN connections, IP address changes, or IP spoofing, including various scenarios commonly employed by attackers to exploit tokens, as discussed in Section 2.3. Consequently, the evaluation results indicate that the 25 rules were successfully performed as expected, as shown in Table 3.

We designed a total of 70 test cases, consisting of 50 positive cases, which should be detected as attack patterns, and 20 negative cases, which are normal requests. The evaluation demonstrates a recall of 92%, indicating a strong capability in detecting actual positive cases. Although the false positive rate reached 45%, this trade-off resulted in a significantly lower false negative rate of 8%, which aligns with standard expectations for signature-based intrusion detection systems that typically prioritize minimizing undetected attacks. While the precision was 83.6%, our monitoring system achieved an overall accuracy of 81.4%. These results indicate that our 25 detection rules performed as intended.

In addition to detection accuracy, we evaluated the system's performance in terms of detection latency and operational overhead to assess its suitability for real-time environments. The monitoring system was deployed alongside the student diary system under simulated production conditions, processing both normal and attack scenarios in real time. The results indicated that the system achieved an average detection latency of 9.7 milliseconds per request,

Table 3 Evaluation results

Metric	Value
Accuracy	0.814
Precision	0.836
Recall	0.92
False Positive Rate	0.45
False Negative Rate	0.08

ensuring immediate detection feedback without introducing noticeable delays to client requests. Furthermore, we measured the system's resource utilization, which showed an average CPU overhead of 2.3% and memory consumption increase of 18 MB under peak testing loads. These findings confirm that the proposed monitoring system delivers effective, real-time detection with minimal impact on application responsiveness and system resources, making it suitable for integration into high-traffic, production-grade environments.

6 Conclusion

This research proposed a lightweight, signature-based token monitoring system for detecting suspicious behaviors associated with token hijacking and misuse in JWT-based authentication. A set of 25 detection rules was developed, grounded in a comprehensive analysis of JWT vulnerabilities, OWASP guidelines, and real-world attack scenarios. Integrated into the monitoring system, these rules generate alerts when client request patterns match known attack behaviors, enabling timely risk mitigation. The system was validated through 70 test cases on a web application, achieving an overall detection accuracy of 81.4%, a recall of 92%, and a false negative rate of 8%, demonstrating its strong capability in identifying token-related attacks. In performance evaluation, the system maintained an average detection latency of 9.7 milliseconds per request, with a 2.3% CPU overhead and an additional 18 MB of memory under peak conditions. These results confirm its suitability for real-time, high-traffic environments.

Our approach addresses a critical limitation in existing IDS and WAF solutions by enabling token tracking and contextual token metadata correlation, allowing proactive identification and revocation of compromised tokens. While the system currently faces challenges in handling dynamic network conditions such as VPN use, IP changes, and spoofing, an exclusion feature for private IP addresses offers administrators control over detection sensitivity. For future work, the system can be enhanced through the integration of anomaly-based detection using machine learning, enabling the identification of previously unseen or zero-day token attacks. This hybrid model aims to improve detection accuracy, adaptability, and responsiveness in diverse deployment scenarios.

This research closes a practical security gap by introducing the first dedicated, deployable, signature-based token monitoring framework tailored for JWT environments. Its novel combination of real-time detection, low operational overhead, and modular, extensible rule design provides a foundation

for both immediate application and future research. The system's architecture enables adaptation to emerging token standards and distributed microservices ecosystems and API-driven architectures that rely on stateless authentication, offering a valuable platform for advancing token security monitoring practices. Future researchers can build on this by incorporating machine learning models for adaptive thresholding, extending our detection framework to other token standards (e.g., PASETO), or integrating token validation with SIEM and SOAR platforms for automated incident response.

Acknowledgment

We would like to express our sincere gratitude to the Asia Pacific Network Information Centre (APNIC) Foundation for generously funding our research through their project called "Switch!". Their invaluable support has been instrumental in facilitating this research. We deeply appreciate APNIC's dedication to fostering innovation and advancing knowledge in the field.

References

[1] Prajakta Solapurkar, 'Building secure healthcare services using OAuth 2.0 and JSON web token in IOT cloud scenario', 2017.

[2] Muhamad Haekal, Eliyani, 'Token-based authentication using JSON Web Token on SIKASIR RESTful Web Service', 2017.

[3] Stenly Ibrahim Adam, Jimmy H Moedjahedy, Jeremiah Maramis, 'RESTful Web Service Implementation on Unklab Information System Using JSON Web Token (JWT)', 2021.

[4] M. Jones, J. Bradley, N. Sakimura, 'JSON Web Token (JWT)', ISSN: 2070-1721, RFC 7519, 2015.

[5] D. Hardt, Ed., 'The OAuth 2.0 Authorization Framework', ISSN: 2070-1721, RFC 6749, 2012.

[6] Irfan Darmawan, Aditya Pratama Abdul Karim, Alam Rahmatulloh, Rohmat Gunawan, Dita Pramesti, 'JSON Web Token Penetration Testing on Cookie Storage with CSRF Techniques', 2022.

[7] Kanika Sharma, Naresh Kumar, 'SWART: Secure Web Application Response Tool', 2013.

[8] Zakiyabanu S. Malek, Bhushan Trivedi, Axita Shah, 'User Behavior-Based Intrusion Detection Using Statistical Techniques', 2018.

[9] Zakiyabanu S. Malek, Bhushan Trivedi, Axita Shah, 'User behavior Pattern -Signature based Intrusion Detection', 2020.

[10] S Sasipriya, L R Madhan Kumar, R Raghuram Krishnan, K Naveen Kumar, 'Intrusion Detection System in Web Applications (IDSWA)', 2021.
[11] 'OWASP Risk Rating Methodology', https://owasp.org/www-community/OWASP_Risk_Rating_Methodology.
[12] P. Varalakshmi, Guhan B, Vignesh Siva P, Dhanush T, Saktheeswaran K, 'Improvising JSON Web Token Authentication in SDN', 2022.
[13] A Rahmatulloh, R Gunawan, F M S Nursuwars, 'Performance comparison of signed algorithms on JSON Web Token', 2019.

Biographies

Pattharadanai Rujichaikul received a bachelor's degree in computer science from Kasetsart University in 2014 and a master's degree in cyber security and information assurance from Mahidol University in 2023.

Ittipon Rassameeroj received bachelor's degree and master's degree in computer science from Mahidol University, and the philosophy of doctoral degree in computer science from University of California, Davis. He is currently working as a faculty member at the Faculty of Information and Communication Technology, Mahidol University, Thailand. His research areas include computer and networked systems, cybersecurity, operating/distributed/parallel systems, and HPC.