Published online on the page: https://jurnal.ilcagam.org/index.php/jsti

J S T I
Jurnal Serumpun Teknik Informatika
| ISSN (*Online*) 3123-965x | DOI: https://doi.org/10.66485/jsti.v1i2.20

Comparative Analysis of Performance and Security Static and Dynamic JSON Web Token (JWT)

Rizky Parlika[1,], Muhammad Romi Nasution[2], Dino Rosalino Yuswanto[3]*

[1,3]Universitas Pembangunan Nasional "Veteran"Jawa Timur, Surabaya, Indonesia
[2]Universitas Pasir Pengaraian, Rokan Hulu, Indonesia

Article Information

Article History:
Submit: 24 Maret 2026
Accepted: 25 April 2026
Published: 30 April 2026

Keywords

JSON Web Token; Peformance; REST API

Correspondence

E-mail: rizkyparlika.if@upnjatim.ac.id*

ABSTRACT

The rapidly evolving technology era demands a secure and efficient authentication mechanism when exchanging information between users and servers. One of the most common authentication methods used in REST APIs is JSON Web Token (JWT) due to its stateless and lightweight nature. However, the implementation of static JWT still has a weakness because pre-existing tokens can be used in other contexts such as other devices or other IP addresses. This can result in token misuse, resulting in data leakage. This study was conducted by comparing the performance and security aspects of static JWT and dynamic JWT in REST APIs using the PHP Laravel framework. Testing results show that the implementation of static and dynamic JWT does not have a significant difference in performance. However, dynamic JWT excels in security aspects because it is able to detect unauthorized access attempts due to context mismatch.

1. Introduction

Technological advancements have become one of the main drivers of transformation in various sectors of human life, including communication, business, education, and government services (Rizki, 2022). The rapid development of digital technology has enabled the creation of new systems and infrastructures that support faster information exchange and more efficient data management processes (Arianto, Witanti, & Ashaury, 2025). In the modern digital era, information technology is not only used to improve productivity but also plays a strategic role in maintaining the confidentiality, integrity, and availability of information (Nurjaman, Utomo, & Hermanto, 2024). As organizations increasingly rely on interconnected systems and digital platforms, the importance of robust information security mechanisms continues to grow to protect sensitive data from unauthorized access, data breaches, and cyber threats (Riadi, Umar, & Busthomi, 2020).

One area significantly influenced by technological progress is the field of information security, particularly in web-based and distributed applications (Software Engineering Institute [SEI], 2024). Modern software development practices increasingly adopt service-oriented architectures and microservices, where applications interact with one another through standardized interfaces known as Application Programming Interfaces (APIs) (Lodder, 2023). Among the various API architectures available, Representational State Transfer (REST) has become one of the most widely adopted approaches due to its simplicity, scalability, and compatibility with HTTP protocols (Dalimunthe, Putra, & Ridha, 2023). RESTful APIs enable efficient communication between clients and servers by using

stateless requests and standardized data formats such as JSON, allowing systems to exchange information seamlessly across different platforms and environments (Corradini, Ceccato, & Ghafari, 2025).

In REST-based systems, security mechanisms such as authentication and authorization are essential to ensure that only legitimate users or systems can access protected resources (Arcuri, Zhang, & Galeotti, 2023). Authentication is the process of verifying the identity of a user or application, while authorization determines whether the authenticated entity has permission to access specific resources or perform certain actions (Salt Security, 2023). Without proper implementation of these mechanisms, APIs may become vulnerable to various security threats, including unauthorized access, data leakage, and malicious attacks. Therefore, implementing robust authentication and authorization methods is a critical aspect of secure API design and deployment (SEI, 2024).

One of the most widely used authentication methods in REST APIs is the JSON Web Token (JWT) mechanism (Dalimunthe et al., 2022). JWT is a compact and self-contained token format that allows secure transmission of authentication information between parties in JSON form (Naik & Jenkins, 2022). This approach enables stateless authentication because the server does not need to store session information; instead, all necessary user identity and authorization data are embedded directly within the token. As a result, JWT provides advantages such as scalability, reduced server-side storage requirements, and improved performance in distributed systems where multiple services must verify user identity independently (Dalimunthe et al., 2023).

Despite its advantages, conventional JWT implementation still has several security limitations. A generated token typically remains valid until its expiration time is reached, regardless of changes in the user's context or environment. This means that if a token is stolen or intercepted by an attacker, it can potentially be reused until it expires, posing a significant security risk (Arcuri et al., 2023). To mitigate this vulnerability, it is necessary to implement a more adaptive authentication approach, such as dynamic JWT validation. This method enhances token verification by embedding additional contextual attributes—such as IP address, device information, browser fingerprint, or user location—within the token or validation process. By comparing these contextual attributes during each request, the system can detect inconsistencies that may indicate suspicious activity. If a mismatch occurs, the token can be immediately rejected, thereby strengthening the overall security of the authentication process and reducing the risk of token misuse (Nikolaou, 2025).

2. Research Methods

This study employs an experimental method. This research method involves conducting an experiment and falls under quantitative research methods. This method is used to determine the effect of independent variables on dependent variables under controlled conditions[30]. Using this method, the author tested and compared two authentication system variables—static JWT and dynamic JWT—by directly examining the differences in performance and security levels of the two mechanisms based on the test results.

2.1. Data Collection Methods

The test data was obtained through direct testing of the implementation of the REST API system, namely:

1. REST API using static JWT authentication.

2. REST API using context-based dynamic JWT authentication.

The testing was conducted locally on a personal computer with the following specifications:

Table 1. Black Box Testing Results

Specifications	Expected Result
Operating System	Windows 11
Framework	Laravel 12
Database	MySQL
Web Server	Laragon
Tools	Postman, Apache JMeter

Data collection was conducted in two ways:

1. Performance Testing : Measuring average response time, CPU usage, memory usage, and throughput using Postman.
2. Security Testing : Testing the system's accuracy in validating invalid tokens, expired tokens, mismatched contexts, and token reuse.

Each test was performed once to ensure consistent results across tests, and the average value was taken as the final result.

2.1. Data Collection Methods

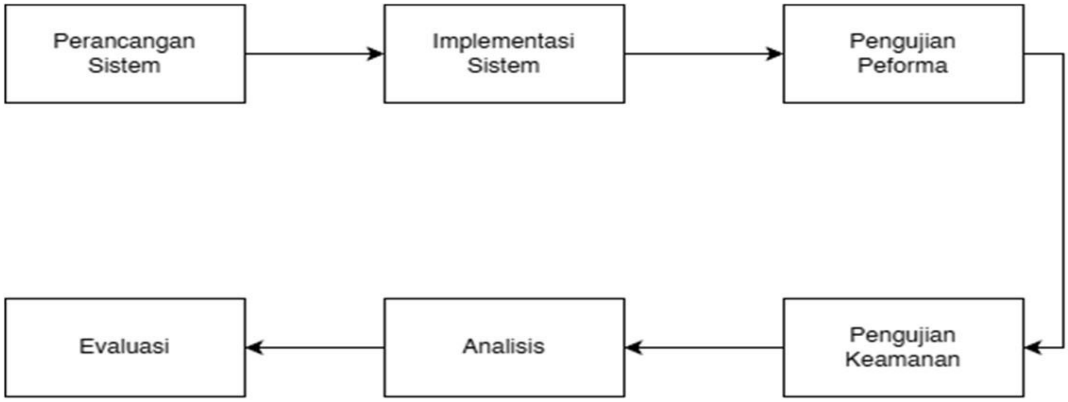

Figure 1. Procedure Flowchart

The research process to obtain the results of the comparative analysis between static and dynamic JWTs was carried out in the following stages:

1. System Design
   The system design stage involved designing a REST API architecture using two authentication modes: static JWT and dynamic JWT. The dynamic JWT was created by adding the IP address and User Agent attributes to the token payload.
2. System Implementation
   The implementation phase began by developing two REST API applications using the same endpoint, along with configuring middleware to perform both static and dynamic validation.
3. Performance Testing
   The performance testing phase involved conducting load testing using Postman by sending 100–1,000 requests, with test results recorded for response time, throughput, and memory usage.
4. Security Testing
   The security testing phase involves checking the system against several conditions such as invalid tokens, context mismatches, and token reuse; from these tests, the system's success rate in rejecting these scenarios will be calculated.

5. Analysis
   Processing the test data by converting it into tables and comparison charts, then conducting an analysis to determine the extent of the differences between static and dynamic JWTs.
6. Evaluation
   Summarizing the results of the security and performance testing for each method.

3. Results and Discussion

Performance testing was conducted to determine the extent of the performance difference between REST APIs with static JWT and dynamic JWT. The testing process was carried out using Postman, utilizing the Collection Runner feature to send a large number of repeated requests. The number of requests tested was 100, 500, and 1000. This round of testing was conducted on several endpoints that require authentication.

This performance test examined several parameters used as benchmarks to describe system performance, namely:

a. Response Time (ms)
   Response Time is the time required by the server to process a request and send a response back to the client. This parameter is a key indicator in evaluating API performance. The lower the Response Time value, the better the system's performance. This test records the average Response Time for each request volume scenario.
b. Throughput
   Throughput refers to the number of requests a server can process within a specific timeframe, typically measured in requests per second (req/s). The throughput figure indicates how well the REST API can handle a high volume of requests simultaneously. This comparison helps determine whether static or dynamic JWT is more stable in maintaining performance as load increases.
c. Memory Usage Memory
   Usage is used to determine how much memory is consumed during API execution. This measurement is important to identify whether a specific authentication method places a heavier load on the system. Dynamic JWT may consume more memory due to the token validation and expiration management processes involved. Meanwhile, static JWT is generally lighter.

Table 2. Comparison of Static and Dynamic JWT Response Times

No	Total Requests	Static JWT (ms)	Dynamic JWT (ms)	Difference
1	100	357	354	0.84%
2	500	342	341	0.29%
3	1000	339	348	2.65%

The test results show that, in some scenarios, dynamic JWTs have faster response times than static JWTs. However, dynamic JWTs exhibit slightly longer response times when the number of requests reaches the thousands; nevertheless, these tests indicate that the addition of security mechanisms does not have a significant impact on system performance, as the maximum difference remains below 3%.

Table 3. Comparison of Static and Dynamic JWT Response Times

No	Total Requests	Static JWT (req/s)	Dynamic JWT (req/s)
1	100	2.26	2.25

2	500	2.38	2.38
3	1000	2.40	2.34

In the test scenarios with 100 and 500 requests, both methods produced nearly identical throughput values, averaging 2.26 req/s and 2.38 req/s, respectively. A slight difference began to emerge when the load was increased to 1,000 requests, where Static JWT recorded 2.40 req/s while Dynamic JWT stood at 2.34 req/s.

Although there was a slight decrease in performance for Dynamic JWT under the highest load scenario, the performance difference was very small (less than 2.5%). This indicates that the additional security mechanisms implemented in the Dynamic model do not significantly burden the server's processing resources, allowing the system to maintain a throughput level comparable to that of the Static model.

Table 4. Comparison of Static and Dynamic JWT Response Times

No	Total Requests	Static JWT (MB)	Dynamic JWT (MB)
1	100	22	22
2	500	22	22
3	1000	22	22

In memory usage testing, there was no difference in performance; both static JWT and dynamic JWT used 22 MB of memory during repeated requests.

After conducting performance testing, security testing was also performed to compare how resilient both authentication methods are against unauthorized access attempts. The purpose of this test is to ensure that the Dynamic JWT mechanism can address security vulnerabilities typically found in the Static JWT method. Security testing was conducted by simulating three attack scenarios or token usage errors, namely:

a. Invalid token
   Sending a request with a token whose format or signature has been modified.
b. Context Mismatch
   Using a token that is correct in format but used in the wrong context. For example, user A's token is used to access user B's data.
c. Token Reuse
   Attempting to reuse a token that has already been used or a token that is no longer active.

Below is a table of the security test results for static JWT and dynamic JWT:

Table 5. Comparison of Static and Dynamic JWT Response Times

No	Scenario	Static JWT	Dynamic JWT
1	Token Invalid	401 Unauthorized (Aman)	401 Unauthorized (Aman)
2	Context Mismatch	200 OK (Tidak Aman)	401 Unauthorized (Aman)
3	Reuse Token	401 Unauthorized (Aman)	401 Unauthorized (Aman)

As shown in the table above, the test results indicate that the primary advantage of dynamic JWTs lies in their ability to handle context mismatches. Dynamic JWTs can prevent the use of stolen tokens if the device or IP address differs, whereas standard JWTs cannot address this issue.

4. Conclusion

Based on the results of the testing and comparative analysis conducted, it can be concluded that the use of Dynamic JSON Web Tokens (JWT) does not have a significant impact on system performance when compared to Static JWT. This is evident from the minor differences in response time and throughput capacity between the two methods—less than 3%—as well as identical memory usage of 22 MB. Although the performance impact is minimal, Dynamic JWT is superior in terms of security because it can detect and prevent unauthorized access in Context Mismatch scenarios or when devices change—a security vulnerability that Static JWT cannot address.

.

References

Arianto, I. G., Witanti, W., & Ashaury, H. (2025). Sistem keamanan otentikasi pengguna pada modul single sign on menggunakan OAuth 2.0 dan one time password. Jurnal Ilmu Komputer dan Teknologi, 6(1), 25-31. https://doi.org/10.35960/ikomti.v6i1.1768

Arcuri, A., Zhang, M., & Galeotti, J. P. (2023). Advanced white-box heuristics for search-based fuzzing of REST APIs. arXiv Preprint, arXiv:2309.08360. https://doi.org/10.1145/3652157

Corradini, D., Ceccato, M., & Ghafari, M. (2025). Automated testing of broken authentication vulnerabilities in web APIs with AuthREST. arXiv Preprint.

Dalimunthe, S., et al. (2022). Model for storing tokens in cookies using JSON Web Token (JWT) with HMAC in e-learning systems. Journal of Applied Engineering and Technological Science.

Dalimunthe, S., et al. (2023). Utilization of JSON Web Token for authentication and verification in digital information systems. Journal of Computer Engineering and Informatics.

Dalimunthe, S., Putra, E. H., & Ridha, M. A. F. (2023). RESTful API security using JSON Web Token (JWT) with HMAC-SHA512 algorithm in session management. IT Journal Research and Development, 8(1), 81-94. https://doi.org/10.25299/itjrd.2023.12029

Lodder, M. (2023). Token-based authentication and authorization with Zero Trust architecture (Master's thesis). Dakota State University.

Naik, N., & Jenkins, P. (2022). Securing RESTful APIs using token-based authentication mechanisms. In Proceedings of the International Conference on Cybersecurity and Digital Forensics.

Nikolaou, I. (2025). REST API access control: An OPTIONS-based authorization enforcement approach. In Proceedings of the ACM International Conference on Web Engineering. https://doi.org/10.1145/3701716.3718327

Nurjaman, I., Utomo, F. S., & Hermanto, N. (2024). Penerapan REST API Laravel sebagai fondasi back-end aplikasi G-MOOC 4D. Journal of Informatics Interaction Technology, 1(1), 9-18. https://doi.org/10.63547/jiite.v1i1.4

Riadi, I., Umar, R., & Busthomi, I. (2020). Optimasi keamanan autentikasi dari man in the middle attack (MiTM) menggunakan teknologi blockchain. Journal of Information Engineering and Educational Technology, 4(1), 15-19. https://doi.org/10.26740/jieet.v4n1.p15-19

Rizki, M. (2022). Perkembangan sistem pertahanan/keamanan siber Indonesia dalam menghadapi tantangan perkembangan teknologi dan informasi. Politeia: Jurnal Ilmu Politik, 14(1), 54-62. https://doi.org/10.32734/politeia.v14i1.6351

Salt Security. (2023). The state of API security report Q1 2023. Salt Labs Research Report.

Software Engineering Institute (SEI). (2024). API vulnerabilities and risks. Carnegie Mellon University.

Soni, N. (2024). Impact of performance on security: JWT token implementation for microservices authentication (Master's thesis). California State University.