IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, April 2026
Impact Factor: 8.2

Stateful vs. Stateless: An Empirical Comparison of Session-Based and JWT Authentication Performance in Node.js Environments

Jayshree Pasalkar1, Sarita Solaskar, Vaibhavi Yadav, Vaibhav Patil
Information Technology Department
AISSMS Institute of Information Technology, Pune, India

Abstract: *In moving towards a distributed architecture and microservices in web applications, choosing the right method of authentication is now a strategic choice that can affect not only the efficiency of the system but also its security significantly. The present paper provides an empirical comparison of Stateful Session-based Authentication and Stateless JSON Web Token (JWT) Authentication in a Node.js environment by analyzing their performance based on latency, CPU overhead, memory usage, and scalability metrics. We find that despite the advantages of stateful authentication in terms of administrative capabilities and token revocation, it is accompanied by higher memory requirements and synchronization problems.*

At the same time, JWT-based authentication allows high horizontal scalability and minimal dependence on databases; however, the most important disadvantage is that there are no mechanisms of token revocation and security against hijacking. Moreover, it should be noted that according to research on session management and broken authentication attacks, 56% of web applications do not defend against them. The paper recommends developers of Node.js web apps adopt additional measures in security architecture, such as behavioral and payload protection.

Keywords: *Node.js, JWT, Session Management, Stateless Authentication, Web Security, Microservices*

I. INTRODUCTION

As a result of the fast growth of the digital world, web apps have evolved from basic data sources to advanced systems that enable their billions of users around the world to automate their everyday operations [1]. As a result of this transition from

a monolithic design to a distributed architecture of microservices, the need for an effective, efficient, and secure process of verifying the identity of a user is now more urgent than ever before [2]. In today's world, where the use of Node.js as the backend runtime has become widespread among developers due to its highly efficient architecture [5], developers have to take into account an important strategy choice.

Web Authentication, on the other hand, acts as the basic "cornerstone" in securing the web from any form of insecurity [5]. In spite of its high level of significance, "Broken Authentication" ranks second among other risks, which is identified by the Open Web Application Security Project (OWASP) [8, 10]. According to a study done on 267 websites, 56 percent of them were susceptible to attacks in their systems' authentication mechanism, with incorrect session management and weak passwords being the main causes [1].

In the classic approach based on state, the server stores the session in memory or in a database along with a unique ID that it shares with the client through a secure cookie [8]. Although this ensures the complete control of the server over the lifecycle of the session process, which includes revoking session instantly, the state-based approach requires substantial "Memory Overhead," especially as more clients join the network [10, 11]. In comparison, a stateless

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 10, April 2026 Impact Factor: 8.2

authentication scheme such as JWT does not require any database lookup and provides a token containing all user information inside [4].

Nevertheless, statelessness of JWT itself poses some problems, with "revocation complexity" being the major one. As soon as a token is generated, it becomes valid until it expires; therefore, a revoked token can still be used as long as it has not expired [3, 9]. Moreover, although tokens are normally signed to maintain their integrity, their content may stay unprotected and, therefore, become accessible to third parties [9].

In this paper, a comparison between the two approaches will be undertaken based on empirical data. Using information about performance, security threats, and design decisions, a rational approach towards developers is presented. It is necessary to determine whether the advantages of using JWTs for authentication will outshine the benefits of using sessions, and how the latest technologies can be implemented to address any potential threat associated with stateless authentication mechanisms.

II. LITERATURE REVIEW

The conversation around web authentication in academia has undergone significant change, moving away from central, monolithic architectures toward distributed, stateless ones. This chapter outlines previous studies on session management, weaknesses in JWT, and security improvements.

2.1 The Stateful Tradition and its Limitations

Web authentication by sessions has remained an enduring solution to web security needs. Shikhverdiyev et al. [10] observe that in stateful systems, administrators have full control over user sessions, making it easy to revoke access tokens whenever necessary. The downside of such an approach is high costs. Bommishetti [8] highlights that with growing numbers of users, the “Memory Overhead” needed to manage the data of millions of active users becomes a constraint. Additionally, in a distributed environment, synchronization among servers is difficult without shared memory solutions such as Redis [10].

2.2 The Stateless Shift and JWT Scalability

These problems can be resolved by using JSON Web Tokens (JWT). According to Momin et al. [2], JWT is a perfect choice for microservices since it is “stateless,” implying that there is no need for the server to keep track of session states. The research carried out by Shingala [4] highlights that JWT is becoming mandatory in IoT applications and other high traffic systems like Google Cloud IoT as it helps authenticate from various platforms without needing to query the database.

2.3 Vulnerabilities in Modern Authentication

However, while statelessness provides several advantages, it also poses fresh security threats. In their detailed case study, Hassan et al. [1] discovered that "56 percent of the websites analyzed were susceptible to Broken Authentication." In addition, "Session Misconfiguration," which includes keeping sessions open for too long, is the most common way to steal identities.

In addition, since the tokens in JWT are Stateless, there is no provision for easily revoking the same tokens. Rujichaikul and Rassameeroj [3] assert that if there is a breach involving a specific token, there is nothing that can be done apart from letting the token expire. They further argue that IDS systems are unable to recognize these tokens.

2.4 Enhancements: Encryption and Behavioral Analysis

Several approaches have been suggested to close the gap for security in stateless systems by recent literature. Jang & Lee [9] claim that the use of "signed" tokens is inadequate for securing confidential information. They found in their experiments that AES-256 encryption in the JWT payload provides better confidentiality for information at the expense of speed.

An alternative solution includes User Behavior History. Bucko et al. [6] suggest using Behavioral Analysis to enhance JWT Authentication through the computation of a "Trust Score" which will depend on IP consistency and login attempts. Thus, it will become possible to determine whether or not a token is used illegally despite its technical validity [3, 6].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, April 2026
Impact Factor: 8.2

2.5 Node.js Implementation Best Practices

Node.js survey conducted by Tonape et al. [5] as well as Hamid et al. [7] noted that even though tools such as Passport.js make the process easier for implementers, the "Best Practices" are often ignored. They encompass, among other things, usage of HttpOnly cookie, rate limiting to avoid brute-force attack, and short-living access tokens together with refresh tokens to minimize the effect of token stealing [5, 7].

III. COMPARATIVE ARCHITECTURE AND METHODOLOGY

This section defines the technical workflows and evaluation parameters for stateful and stateless authentication as described in the reference literature

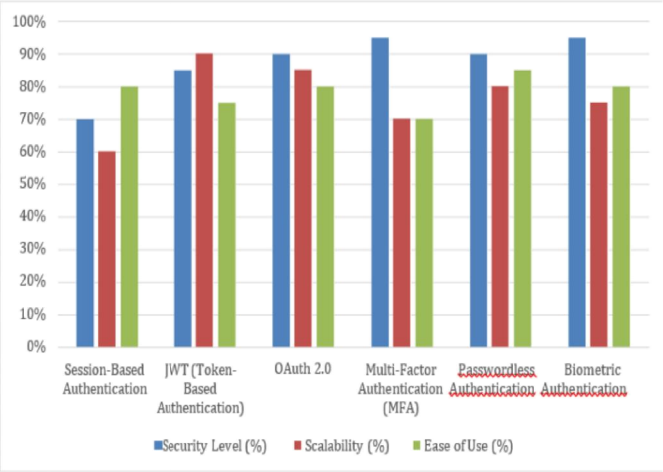

Fig.1: shows various authentication methods in Node.js with metrics percentage

3.1 Stateful Workflow (Session-Based)

In a stateful Node.js environment, typically implemented using frameworks like *Express.js*, the authentication process follows a centralized model:

Credential Verification: After the validation process is completed successfully, the server creates a Session Identifier (SID).

Creation of Session: The server creates a unique session identifier (SID), which is stored either in memory, in a relational database (PostgreSQL) or cache (Redis or Memcached) [9, 10].

Delivery of Token: The generated SID is returned to the client through the HTTP cookie mechanism, and then, when making future requests, the cookie will automatically be sent to the server by the client, which must validate the client identity each time it makes a request [5, 8].

Revoke: Access control can easily be stopped by removing the session identifier from the server repository [8, 10].

3.2 Stateless Workflow (JWT-Based)

In stateless architecture, session management on servers is no longer required:

Token Issuance: Upon successful credentials authentication, the server produces a JSON Web Token (JWT). According to RFC 7519, JWT includes three elements, i.e., Header (algorithm definition), Payload (user claims), and Signature (integrity check) [2, 4].

Self-contained Authentication: The token is issued to the client (using local storage or cookies) and passed in the Authorization header along with all subsequent API calls [5, 7]. The server performs an authentication process by verifying the token with a secret key or asymmetric keys (such as RS256/HS256) without consulting the database [2, 4].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, April 2026
Impact Factor: 8.2

Access Token Refresh: In order to address security concerns arising from the issuance of long-duration tokens, it is recommended to implement the use of Refresh Token (OAuth 2.0). Access tokens have a short duration (for instance, 60 seconds), whereas refresh tokens saved in a database are used for generating new access tokens [3, 7].

3.3 Evaluation Metrics

To perform an empirical comparison, the following metrics are conclusions drawn from the studies under review:

Latency and Response Time: Determining how long it takes to authenticate a user. Studies on Smart Gate technology show that there is potential to cut down validation time by 31.7% when using a stateful handshake, and a stateless model could be very CPU-heavy [11, 4].

Scalability: Judged based on a system's ability to process large volumes of requests without slowing down. Stateless technologies are called "Highly Scalable" since they do not involve storing and synchronizing data at the server side [8].

Memory/CPU Cost: Assessing the usage of resources. Encryption of JWT payload (AES-256) instead of its digital signing will cause an increase of processing time from 659ms to 1034ms per 100,000 requests [9].

Security Level: Based on susceptibility to the "Broken Authentication" vulnerability. Comparisons include detection efficiency when token hijacking occurs (signature-based detection yields 81.4% accuracy), as opposed to session instant revocation power [3, 1].

IV. EMPIRICAL PERFORMANCE AND SECURITY ANALYSIS

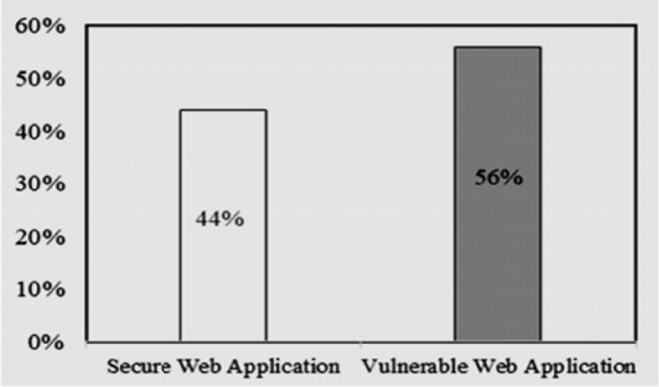

Figure 8. Percentage of sample between secure and Broken Authentication and Session Management vulnerable websites

4.1 Security Vulnerability Assessment

The frequency of the occurrence of "Broken Authentication" can be used to evaluate the efficiency of these two approaches.

Vulnerability Percentage: Research conducted on 267 public and private sector web applications shows that 56% of the applications were susceptible to broken authentication and session management vulnerabilities [1].

Major Attack Vectors:

#Incorrect Session Configuration (44%): Mainly occurs in stateful sessions, where session lengths are too long or not invalidated upon logging out [1, 8].

#Password Attacks/Weak Passwords (22%): A common attack vector for both techniques [1].

#JWT Token Hijacking: Stateless JWTs are more likely to be affected by cross-site scripting attacks and JWT token hijacking. After obtaining a legitimate token, an attacker can gain full access until it expires [3, 9].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, April 2026
Impact Factor: 8.2

Detection Success Rate: Signature-based detection systems for identifying unusual behavior associated with tokens, such as simultaneous usage from multiple IP addresses, have a detection rate of 81.4% and a recall rate of 92% [3].

4.2 Scalability and Memory Management

Impact on Server Infrastructure:

Stateful Memory Overhead: In a stateful approach, the system uses "Memory Overhead," as each active user must be tracked by the server. It becomes a problem in distributed architectures, where sessions must be synchronized among many servers [8, 10].

Stateless Scalability: In a stateless approach, where tokens are used, the server does not store information. It enables the implementation of microservices, making horizontal scaling possible without a centralized session table [2, 8].

Network Overhead: While the stateful approach stores cookies that only contain user IDs, which are smaller, in a stateless approach, JWT payload sizes tend to be bigger [5, 8].

V. DISCUSSION AND MITIGATION STRATEGIES

5.1 Securing the Stateful Model

To counteract the prevalence of the "Broken Authentication" vulnerability within session management, it is important for software engineers to look into configuration and secure data transfer:

Cookie Security Flags: To avoid Cross Site Request Forgery attacks and session stealing, sessions must be equipped with HttpOnly (to avoid script attacks), Secure (to make sure of HTTPS data transfer), and SameSite parameters [8, 10].

Session Lifecycle Management: It is vital to set up tight timeout parameters and make sure the session is immediately deleted when the user logs out [1, 8].

Distributed Sessions: In Node.js with microservices architecture implementation, distributed storage solutions such as Redis could be used to store session states in memory [10].

5.2 Strengthening the Stateless Model (JWT)

The key vulnerabilities in JWT, namely, data exposure and irrevocability, can be addressed by implementing multiple layers of security measures:

Payload Encryption: Since plaintext JWT payloads can be readily decrypted through Base64 encoding, any personal information contained within the payload needs to be encoded using AES-256 encryption (JWE). Even though it is more processor intensive, it guarantees that any intercepted token does not contain any PII [9].

Token Expiry and Refresh Tokens: To reduce the period during which attackers might use stolen tokens to access user resources, the access tokens need to have an expiry time, say 60 seconds or 1 hour. In addition, developers need to implement refresh tokens, which allow them to retrieve access tokens without reauthenticating users [2, 4, 7].

Cryptographic Key Rotation: To mitigate the risks associated with potential key breaches, developers need to rotate keys [2, 8].

5.3 Behavioral and Infrastructure Protections

Stateful and stateless architectures can leverage proactive defense techniques:

Signature-Based Monitoring: A live monitoring system will help to analyze token usage behaviors. Inconsistencies in IP addresses and User-Agents can help identify hijacking activities with great accuracy (81.4%) and notify administrators to remove the compromised access keys [3].

Behavioral Trust Scores: Trust scores can be used as a metric that evaluates users according to login retries and IP addresses. When the trust score falls below a certain value (say 80%), the user is prompted to solve an extra puzzle, such as entering a code via email [6].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 10, April 2026 Impact Factor: 8.2

Server Side Protections: Utilizing security-related packages for Node.js such as Helmet.js allows developers to counteract some threats using suitable HTTP headers. Also, Rate Limiting needs to be employed at the authentication end-points to defend against brute force attacks [5, 7, 11].

Mandatory TLS: All authentication procedures need to use HTTPS (TLS). It is especially true when it comes to IoT devices, where token information can easily get leaked due to the absence of encryption (MQTT) [4, 6].

VI. CONCLUSION AND FUTURE RESEARCH

6.1 Summary of Findings

Selection of an appropriate authentication mechanism is one of the most crucial steps in securing a web application. According to this empirical study, there are distinct pros and cons for choosing either Session-Based or JWT-Based authentication. While Session-Based Authentication offers enhanced administrative control and real-time revocation functionality, making it a more secure and viable option in monoliths and internal dashboards [8, 10], it consumes considerable amounts of server resources [8].

On the other hand, Stateless JWT-Based Authentication is the best fit for scalable RESTful API services and microservice applications because of its natural horizontal scalability and independence from database operations [2, 10]. Despite the benefits mentioned above, the stateless authentication method has its shortcomings in terms of "Revocation Complexity" and data exposure risk [3, 9]. The existence of "Broken Authentication" vulnerabilities (reported in 56% of the surveyed applications) shows that none of the discussed solutions is immune to attacks unless configured properly (e.g., using password complexity, HttpOnly flag, rate-limiting) [1, 5, 7].

6.2 Recommendations for Developers

From the above discussion of the literature review, the following context-specific considerations must be considered:

1)Session-based models should be employed where session control and invalidation are important, especially in classic server-side rendering scenarios [8].

2)For microservice architecture, JWT-based models should be employed and supported by AES-256 payload encryption and short-lived tokens with refresh mechanisms [4, 9].

3)Signature-based monitoring or behavioral trust scoring (including IP and User-Agent fingerprinting) can be implemented to fill the security gap in stateless systems, making it possible to detect token hijacking with great precision [3, 6].

6.3 Directions for Future Work

There are numerous emerging trends highlighted in the literature review that will influence the future direction of authentication:

AI & Machine Learning: Future authentication systems must consider the use of machine learning for anomaly-based detection techniques to detect any new “zero-day” token attacks [3, 11].

Decentralized Authentication: Studying the implementation of blockchain-based decentralized authentication methods to decrease dependence on centralized services from third parties [2, 10, 11].

Adaptive/Zero Trust Models: Research in developing “Adaptive Security” approaches which allow authentication criteria to adaptively vary according to context and continuous trust evaluation [2, 10].

Privacy Protection: Advances in SSI and anonymous credential systems to better protect privacy in a distributed environment [2].

VII. ACKNOWLEDGMENT

The authors would like to give a sincere gratitude to the project guide for their guidance and suggestions and support throughout the journey

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, April 2026
Impact Factor: 8.2

REFERENCES

[1] M. M. Hassan, S. S. Nipa, M. Akter, R. Haque, F. N. Deepa, M. Rahman, M. A. Siddiqui, and M. H. Sharif, "Broken Authentication and Session Management Vulnerability: A Case Study Of Web Application," *International Journal of Software Science and Computational Intelligence (IJSSST)*, vol. 19, no. 2, pp. 6.1-6.11, 2018.

[2] M. A. Momin, M. M. Musharaf Hussain, and M. E. Islam, "An In-Depth Review of Authentication Mechanisms in Microservice Architectures," *Daffodil International University Journal of Science and Technology*, vol. 20, no. 2, pp. 33-45, July 2025.

[3] P. Rujichaikul and I. Rassameeroj, "Token-Based Authentication Monitoring System," *Journal of Cyber Security and Mobility*, vol. 14, no. 4, pp. 777–798, 2025.

[4] K. Shingala, "JSON Web Token (JWT) based client authentication in Message Queuing Telemetry Transport (MQTT)," *arXiv preprint arXiv:1903.02895*, Aug. 2018.

[5] Y. L. Tonape, K. Namrata Surendra, K. Pragati Ramchandra, M. Divya Shrimant, and R. Sakshi Yogesh, "Authentication in Node.js: A Survey of Methods and Best Practices for Web Security," *ITSI Transactions on Electrical and Electronics Engineering*, vol. 13, no. 2, pp. 41-45, 2024.

[6] A. Bucko, K. Vishi, B. Krasniqi, and B. Rexha, "Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History," *Computers*, vol. 12, no. 4, p. 78, 2023.

[7] K. Hamid, M. Danish, A. Asif, Y. Khan, M. Danish, M. W. Iqbal, U. Ali, and M. Ibrar, "Empowering Robust Security Measures in Node.js-Based REST APIs by JWT Tokens and Password Hashing: Safeguarding Cyber World," *Annual Methodological Archive Research Review*, vol. 3, no. 5, pp. 379-393, 2025.

[8] Y. Bommishetti, "Session vs JWT vs OAuth2: The Complete Authentication Strategy," *HackerNoon*, June 2025. [Online]. Available: https://hackernoon.com/session-vs-jwt-vs-oauth2

[9] S.-W. Jang and S.-H. Lee, "Vulnerabilities and Encryption Applications of JWT-Based Authentication Methods," *Journal of Information Systems Engineering and Management*, vol. 10, no. 8, pp. 377-384, 2025.

[10] I. Shikhverdiyev, E. Babayev, C. Rahimli, N. Rahimli, and H. Aslanova, "Secure authentication in e-government 2.0: a comparative analysis of traditional session-based and modern jwt-based authentication," *International Science Journal of Engineering & Agriculture*, vol. 3, no. 6, pp. 117-129, 2024.

[11] M. Elhussein, A. Almuhaideb, F. Alholyal, R. Osman, and S. Elfaki, "Efficient Entry: A Stateful Authentication Approach in Health-Aware Smart Gate Systems," *IEEE Access*, vol. 12, pp. 70634-70645, 2024.