HSR (LONDON),
HOUGHTON STREET REVIEW JOURNAL
ISSN: 2058-5519|Impact factor-15.9
http://houghtonstreetreview.com/index.php/hsr

UDC: 004.056.5:004.77

SECURITY AND SCALABILITY TRADE-OFFS IN WEB AUTHENTICATION: A COMPARATIVE BENCHMARKING OF JWT AND SESSION-BASED MECHANISMS

Etibor Luqmanova

Millat Umidi University

Doi: https://doi.org/10.5281/zenodo.21929106

Abstract: In modern web development, choosing the right way to log in and identify users (authentication) is one of the most important decisions for software developers. This paper compares two popular authentication methods: traditional **Session-based authentication** (which uses cookies and stores user data on the server) and modern **JSON Web Token (JWT) authentication** (which stores user data inside a secure token on the client side). The paper analyzes both methods in terms of server memory usage, speed, system scaling, and resistance to common security threats like **Cross-Site Scripting (XSS)** and **Cross-Site Request Forgery (CSRF)**. The goal of this research is to give web developers clear and practical guidelines on when to use each authentication approach.

Keywords: Web Authentication, JWT, Session Management, Web Security, Microservices, Scalability, XSS, CSRF, Web Development.

Key terminology

- **Session-based Authentication:** A traditional login system where the web server saves active user information in its own database or memory.
- **JWT (JSON Web Token) Authentication:** A modern login system where user information is stored directly inside a digitally signed token sent to the client.
- **XSS (Cross-Site Scripting):** A web attack where malicious code is injected into a website to steal stored tokens or cookies.
- **CSRF (Cross-Site Request Forgery):** An attack that tricks an authenticated browser into sending unwanted commands to a web server.
- **Scalability:** The ability of a web system to handle growing amounts of users and traffic without losing performance.

1. INTRODUCTION

When building modern websites and web applications, verifying who the user is (authentication) and controlling what they can do (authorization) is the first line of defense. Today, web applications are moving away from monolithic designs (where everything runs on a single server) toward modern Single-Page Applications (SPAs) and microservice systems that run across multiple cloud servers.

For many years, **Session-based authentication** was the default standard for websites. When a user logged in, the server created a session record in its memory and saved a small cookie on the user's browser. Every time the user clicked a link or submitted a form, the server checked this session record. While this method is very secure and allows easy management of user access, it creates challenges when a web platform grows and needs dozens of servers working together.

HSR (LONDON),
HOUGHTON STREET REVIEW JOURNAL
ISSN: 2058-5519|Impact factor-15.9
http://houghtonstreetreview.com/index.php/hsr

To solve this problem, developers started using **JSON Web Tokens (JWT)**. Instead of saving data on the server, the server creates a encrypted token containing user details and gives it to the browser. The browser sends this token with every request, and the server simply checks the signature without needing to search a database. However, this convenience brings new challenges, especially regarding how to log out a compromised user immediately.

2. LITERATURE REVIEW

Security experts and software engineers have widely studied the pros and cons of both authentication methods:

1. **Nygren & Bäckström (2019)** studied where to store tokens in web browsers. They found that saving JWTs in browser localStorage makes them easy targets for XSS attacks and recommnded using secure cookies instead.

2. **Jovanovic et al. (2020)** measured system speed in cloud applications. They showed that checking database sessions adds small delays, whereas JWTs can be verified locally much faster.

3. **Fermín et al. (2021)** analyzed microservice architectures and demonstrated that stateless JWTs reduce server infrastructure costs because servers do not need to constantly sync user session data.

4. **OWASP Security Guides (2023)** highlighted that while JWTs make scaling easier, revoking or deleting a token before its natural expiration time requires extra architecture work.

5. **Acar et al. (2020)** explored security habits among developers and emphasized that understanding how tokens work is crucial to avoiding basic security mistakes.

Despite these studies, web developers often struggle to decide which method fits their project best. This paper provides a clear comparison based on simple performance metrics and security needs.

3. RESEARCH METHODOLOGY

3.1. How Both Systems Work

1. Session-Based Workflow:

1. The user enters a login name and password on the website.
2. The server checks the credentials and creates a unique **Session ID**.
3. The server saves this Session ID in its database/RAM and sends a cookie to the user's browser.
4. On every new page visit, the browser sends the cookie back, and the server checks its database to confirm the user is logged in.

2. JWT Workflow:

1. The user enters a login name and password on the website.
2. The server checks the credentials and creates a digital **JWT token** signed with a secret key.

HSR (LONDON),
HOUGHTON STREET REVIEW JOURNAL
ISSN: 2058-5519|Impact factor-15.9
http://houghtonstreetreview.com/index.php/hsr

3. The server sends the token to the browser, but saves **nothing** in its own database

4. On every new request, the browser sends the JWT token in the header.

5. The server checks the cryptographic signature using its secret key. If valid, the user is authenticated instantly.

3.2. Evaluation Factors

To evaluate both systems clearly, the following four main factors were tested:

- **Server Memory Usage:** How much RAM the server needs to keep users logged in.
- **Authentication Speed:** How fast the server validates a request.
- **System Scaling:** How easy it is to add more servers as traffic grows.
- **Security & Vulnerabilities:** How well each system protects against XSS and CSRF attacks.

4. RESULTS AND DISCUSSION

The comparison between Session-based and JWT authentication is summarized in the table below:

Feature / Metric	Session-Based Authentication	JWT Authentication	Simple Advantage
Server Memory (RAM)	Requires RAM for every user	Uses almost zero server RAM	**JWT saves server memory**
Validation Speed	Needs a database/cache lookup	Verified instantly via secret key	**JWT is faster**
Scaling across multiple servers	Needs shared session storage	Works automatically on all servers	**JWT is easier to scale**
Immediate User Revocation (Log out)	Very easy (Delete session record)	Harder (Requires token blacklist)	**Session is more controllable**
CSRF Attack Risk	Higher risk (Cookies auto-sent)	Lower risk (Headers used)	**JWT is safer by default**

HSR (LONDON),
HOUGHTON STREET REVIEW JOURNAL
ISSN: 2058-5519|Impact factor-15.9
http://houghtonstreetreview.com/index.php/hsr

Feature / Metric	Session-Based Authentication	JWT Authentication	Simple Advantage
XSS Attack Risk	Lower risk (Protected by HttpOnly)	Higher risk if saved in LocalStorage	**Session is safer for storage**

Key Takeaways from the Comparison:

1. **Performance and Speed:** JWT is noticeably faster because the server does not need to query a database or cache every time a user opens a new page. The server simply verifies the digital signature locally.

2. **Growth and Scaling:** If your website suddenly receives thousands of new users, JWT makes it simple to add new servers. Any server can verify the user's token independently. With Session-based systems, all servers must share a central session database (like Redis).

3. **Control and Revocation:** The biggest downside of JWT is revoking access. If a user's account is hacked, or if an admin wants to block a user immediately, doing so with a stateless JWT is tricky because the token remains valid until it expires. Session-based systems allow admins to delete a user's session instantly.

5. CONCLUSION AND RECOMMENDATIONS

Both authentication mechanisms have clear strengths and weaknesses depending on the project type:

1. **Use JWT Authentication if:**
   - You are building a large microservices system or mobile API.
   - You plan to scale your platform across multiple servers.
   - Speed and saving server memory are your top priorities.
2. **Use Session-Based Authentication if:**
   - You are building a traditional website or administrative portal.
   - You need strict security control, such as instant user logout or forced password reset.
   - Your system runs on a single server or a simple architecture
3. **Recommended Best Practice:**

HSR (LONDON),
HOUGHTON STREET REVIEW JOURNAL
ISSN: 2058-5519|Impact factor-15.9
http://houghtonstreetreview.com/index.php/hsr

When using JWT in modern web applications, developers should store the token in an **HttpOnly cookie** instead of browser localStorage. This hybrid method combines the high speed and scalability of JWT with the XSS protection of traditional cookies.

REFERENCES

1. Nygren, M., & Bäckström, A. (2019). *Evaluating Token-Based Authentication Mechanisms in Modern Single-Page Web Applications*. KTH Royal Institute of Technology.
2. Jovanovic, M., Stojkov, M., & Vidakovic, M. (2020). Performance evaluation of stateful vs stateless authentication mechanisms in distributed web architectures. *Journal of Web Engineering*, 19(3), 345-368.
3. Fermín, A., Sánchez, J., & Torres, R. (2021). Microservices security patterns: A comparative performance benchmark of OAuth2 and JWT implementations. *IEEE Access*, 9, 112450-112462.
4. OWASP Foundation. (2023). *JSON Web Token Cheat Sheet for Web Developers*. OWASP Cheat Sheet Series.
5. Acar, Y., Stransky, C., Wermke, D., & Mazurek, M. L. (2020). Developers and security: An empirical study of cryptographic API usage in RESTful APIs. *IEEE Symposium on Security and Privacy (S&P)*, 412-429.
6. Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)*. RFC 7519, Internet Engineering Task Force (IETF).
7. Fielding, R. T., & Reschke, J. (2014). *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. RFC 7235, Internet Engineering Task Force (IETF).
8. Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446, Internet Engineering Task Force (IETF).
9. Howard, P., & Wright, G. (2022). Benchmarking memory and throughput overheads of token validation in serverless architectures. *Software: Practice and Experience*, 52(8), 1789-1805.
10. Smith, D., & Brown, L. (2021). Mitigating Cross-Site Scripting (XSS) and Session Hijacking in Modern Web Frameworks. *ACM Computing Surveys*, 54(4), 1-36.