International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Secure Web Authentication Using JWT and Role-Based Access Control

Nilay Dedhia

Department of Computer Science and Engineering, Parul Institute of Technology, Parul University, Gujarat, India

Abstract

Web applications play a critical role in modern digital systems and handle large amounts of sensitive user data, making them a major target for cyber attacks. Weak authentication mechanisms, poor session management, and inadequate access control can lead to vulnerabilities such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and session hijacking, resulting in unauthorized access and data breaches.

This research presents the design and implementation of a secure role-based authentication framework. The system uses JSON Web Tokens (JWT) for stateless authentication and stores tokens in HTTP-only cookies to prevent client-side access and reduce XSS risks. Role-Based Access Control (RBAC) is implemented to restrict access based on user roles, while secure password hashing and middleware validation enhance overall security.

The proposed framework improves authentication security, strengthens session management, and reduces common web vulnerabilities. It is scalable and suitable for integration into modern web applications.

Keyword: JWT, RBAC, Authentication, Web Security, HTTP-only Cookies

1.Introduction

Web applications play a vital role in modern digital systems and handle large amounts of sensitive user information, making them a common target for cyber attacks. As their usage increases, vulnerabilities in web systems can lead to data breaches, unauthorized access, and system compromise.

Authentication and authorization are critical components of web application security. Authentication verifies user identity, while authorization determines the level of access granted to users. Weak implementation of these mechanisms can expose applications to serious security risks.

Many traditional systems rely on session-based authentication or store tokens in browser local storage. While these approaches are easy to implement, they introduce vulnerabilities. For example, tokens stored in local storage can be accessed through JavaScript, making them susceptible to Cross-Site Scripting (XSS) attacks and token theft.

This research focuses on designing a secure role-based authentication system for modern web applications. The proposed framework uses JSON Web Tokens (JWT) for stateless authentication and stores them in HTTP-only cookies to prevent client-side access. Role-Based Access Control (RBAC) is implemented to manage user permissions effectively. The system aims to enhance security by reducing vulnerabilities such as XSS, session hijacking, and unauthorized access.

2.Literature Review

Web application security has gained significant attention due to the rapid growth of digital platforms and the increasing amount of sensitive user data handled by modern systems. Studies show that vulnerabilities in authentication and authorization mechanisms are among the primary causes of data breaches and unauthorized access.

Traditional web applications commonly use session-based authentication, where session IDs are stored on the client side. Although widely used, this approach faces challenges in scalability and can be vulnerable to attacks such as session hijacking if not properly managed.

To address these limitations, modern systems have adopted token-based authentication, particularly JSON Web Tokens (JWT). JWT enables stateless authentication, reducing server-side storage requirements and improving scalability. The token securely carries user information such as identity, roles, and expiration time.

However, the security of JWT largely depends on its storage mechanism. Storing tokens in browser local storage is insecure, as it is accessible through JavaScript and vulnerable to Cross-Site Scripting (XSS) attacks. To mitigate this, security experts recommend using HTTP-only cookies, which prevent client-side access. Additionalattributes like Secure and SameSite further enhance protection against attacks such as Cross-Site Request Forgery (CSRF).

For authorization, Role-Based Access Control (RBAC) is widely used. It assigns permissions to roles instead of individual users, simplifying management and enforcing the principle of least privilege. However, improper role configuration can lead to security issues such as privilege escalation.

Modern research also emphasizes a layered security approach, combining authentication, authorization, and validation mechanisms to provide comprehensive protection. Overall, the effectiveness of JWT and RBAC depends on secure implementation and proper configuration, supported by best practices such as HTTP-only cookies and middleware validation.

3.Problem Statement

Despite advancements in web technologies, many web applications still suffer from significant security weaknesses, particularly in authentication and authorization mechanisms. These vulnerabilities often lead to unauthorized access, data breaches, and system compromise.

Several critical issues exist in current systems:

Issue	Description	Impact
Insecure Token Storage	Tokens stored in local storage	XSS attacks
Weak Password Security	Poor hashing or plain text	Data breach
Improper Cookie Settings	Missing security attributes	CSRF & hijacking
Weak Authorization	No role validation	Unauthorized access

4.Proposed Methodology

The proposed system presents a secure authentication and authorization framework for modern web applications. It integrates password hashing, token-based authentication, secure cookie storage, and Role-Based Access Control (RBAC) to reduce common web vulnerabilities while maintaining scalability and performance.

The system follows a layered security approach, where multiple mechanisms work together to provide strong protection.

System Overview

The framework consists of the following components:

Component	Function
Registration Module	Stores user data securely
Login Module	Authenticates user
JWT Token	Generates authentication token
Cookie Storage	Stores token securely
RBAC	Controls access
Middleware	Validates request

Secure User Registration

- User credentials are collected during registration.
- Passwords are hashed using bcrypt with salting.
- Only hashed passwords are stored in the database.

Secure Login Process

- User submits login credentials.
- Server validates credentials using hashed password comparison.
- Upon successful authentication, a JWT token is generated.

Step	Process
1	User enters credentials
2	Server verifies credentials
3	JWT generated
4	Stored in HTTP-only cookie
5	Middleware validates
6	Access granted/denied

JWT Token Generation

- JWT is used for stateless authentication.
- The token contains user ID, role, and expiration time.
- It is digitally signed to ensure integrity and authenticity.

Secure Token Storage

- JWT is stored in an HTTP-only cookie instead of local storage.
- Cookie configuration includes:
  - HTTP-only
  - Secure
  - SameSite
  - Expiration time

Role-Based Access Control (RBAC)

- Users are assigned roles (e.g., Admin, User).

Role	Permissions
Admin	Full access
User	Limited access
Guest	View only

Middleware-Based Validation

- Extracts JWT from cookie
- Verifies token signature and expiration
- Validates user role for requested resource
- Denies access if validation fails

Secure Logout Mechanism

- Authentication cookie is cleared
- Token is invalidated
- Session is terminated securely

Security Advantages

- Strong password protection using hashing
- Protection against XSS via HTTP-only cookies
- Reduced CSRF risk using cookie attributes
- Scalable authentication using JWT
- Controlled access through RBAC

5.System Architecture

This section describes the structure of the proposed system and how its components interact to ensure secure authentication and authorization.

Overview of System Architecture

The system follows a secure client-server architecture with a layered design to improve security and maintain clear separation of responsibilities.

The major layers are:

- Client Layer (Frontend)
- Server Layer (Backend)
- Authentication Layer
- Authorization Layer
- Database Layer

Client Layer

The client layer represents the user interface where users:

- Register
- Login
- Access dashboard
- Perform authorized actions

The frontend communicates with the backend through secure HTTPS requests.

Security features:

- No sensitive data stored in local storage
- Tokens are not accessible via JavaScript
- All communication is encrypted

Server Layer

The server acts as the core of the system and handles:

- Authentication
- Authorization
- Token generation
- Data processing
- Database communication

It exposes secure API endpoints such as:

- /register
- /login
- /dashboard
- /admin

All protected routes are validated using middleware.

Authentication Layer

This layer verifies user identity:

- User submits credentials
- Server validates using hashed password
- JWT token is generated and stored in HTTP-only cookie

The token contains user ID, role, and expiration time.

Authorization Layer

Authorization is managed using Role-Based Access Control (RBAC):

- Users are assigned roles
- Roles define permissions
- Middleware checks role before access

Example:

- Admin → Full access
- User → Limited access

Middleware Security Layer

Middleware acts as a security checkpoint for all protected requests:

- Extracts JWT from cookie
- Verifies token validity and expiration
- Checks user role
- Grants or denies access

Database Layer

The database stores:

- User credentials (hashed)
- User roles
- Application data

Security measures:

- Password hashing (bcrypt)
- No plain-text storage
- Controlled access permissions

Data Flow in the Architecture

1. User sends login request
2. Server verifies credentials
3. JWT token is generated
4. Token stored in HTTP-only cookie
5. User requests protected resource
6. Middleware validates token and role
7. Server responds if authorized

6.Security Analysis

The proposed authentication framework enhances web application security by using a layered approach, where multiple mechanisms work together to protect the system. This ensures that even if one layer is compromised, other layers continue to provide protection.

The system integrates password hashing, token-based authentication, secure cookie storage, and Role-Based Access Control (RBAC) to defend against common vulnerabilities such as unauthorized access and session attacks.

Protection Against Cross-Site Scripting (XSS)

XSS attacks allow malicious scripts to access sensitive data. In traditional systems, tokens stored in local storage are vulnerable because they can be accessed via JavaScript. The proposed system reduces this risk by storing tokens in HTTP-only cookies, which are not accessible to client-side scripts.

Protection Against Cross-Site Request Forgery (CSRF)

CSRF attacks trick users into performing unintended actions. This system prevents such attacks using secure cookie attributes like SameSite and Secure, ensuring that cookies are only sent with trusted requests.

Secure Password Storage

Passwords are protected using bcrypt hashing with salting. This converts passwords into a secure format, making it extremely difficult for attackers to retrieve original credentials even if the database is compromised.

Role-Based Access Control (RBAC) Security

RBAC ensures that users can access only the resources permitted by their roles. Permissions are assigned to roles rather than individuals, simplifying management and enforcing the principle of least privilege.

Middleware-Based Validation

Middleware acts as a security checkpoint by verifying the JWT, checking its validity, and validating user roles for every request. Unauthorized or invalid requests are immediately rejected, ensuring continuous security enforcement.

Feature	Benefit
JWT	Scalable authentication
HTTP-only Cookies	Prevent XSS
Secure Cookies	Prevent CSRF
bcryptHashing	Protect passwords
Middleware	Continuous validation

7.Results and System Testing

The proposed authentication framework was tested under different scenarios to evaluate both its functionality and its ability to handle security threats effectively.

Test Case	Result
Valid Login	Success
Invalid Login	Denied
Expired Token	Re-login required
Unauthorized Access	Blocked
Token Tampering	Rejected

8.Advantages of the Proposed System -

The proposed authentication framework offers several advantages over traditional systems by combining strong security measures with a scalable design. It enhances protection against common threats while maintaining efficient performance and ease of management.

One of the key advantages is improved security. By storing authentication tokens in HTTP-only cookies instead of local storage, the system prevents access through JavaScript, significantly reducing the risk of Cross-Site Scripting (XSS) attacks.

The framework also ensures secure password storage using bcrypt hashing with salting. This protects user credentials by preventing plain-text storage and making it difficult for attackers to recover passwords even if the database is compromised.

Another major benefit is scalability. The use of JSON Web Tokens (JWT) enables stateless authentication, reducing server load and allowing the system to handle a large number of users efficiently.

The system also provides strong protection against common web attacks. Secure cookie attributes such as SameSite and Secure help prevent Cross-Site Request Forgery (CSRF) and session-related attacks.

In addition, Role-Based Access Control (RBAC) ensures structured and efficient access management. Permissions are assigned based on roles, enforcing the principle of least privilege and reducing the risk of unauthorized access.

Overall, the framework provides a balanced solution that improves security, scalability, and usability, making it suitable for modern web applications.

9.Limitations of the System

Although the proposed authentication framework improves security and performance, it has certain limitations that must be considered.

One major limitation is its dependence on HTTPS. Since authentication tokens are stored in cookies, secure communication is required to prevent interception during transmission. Without proper HTTPS implementation, the system can be vulnerable to attacks such as man-in-the-middle.

Another limitation is token expiration. JWT tokens expire after a fixed time for security purposes, which may require users to log in again. This can affect user experience, especially in applications requiring continuous access.

Additionally, JWT-based authentication is stateless, making it difficult to immediately invalidate a token before its expiration. If a token is compromised, it may remain valid unless additional mechanisms like token blacklisting are used.

The system also requires careful implementation. Incorrect configuration of cookies, tokens, or RBAC policies can introduce new vulnerabilities, making proper development and testing essential.

Overall, while the system enhances security, these limitations highlight the need for proper deployment and future improvements.

10.Future Work

The proposed system can be further enhanced with the following improvements:

- **Multi-Factor Authentication (MFA):** Adding an extra verification step, such as a one-time password, to strengthen security.
- **Biometric Authentication:** Integration of fingerprint or facial recognition for improved user authentication, especially in mobile applications.
- **Login Anomaly Detection:** Detecting unusual login patterns, such as suspicious locations or abnormal attempts, to prevent unauthorized access.

11.Conclusion

This research presented a secure authentication framework using JSON Web Tokens (JWT) and Role-Based Access Control (RBAC). The system integrates password hashing, HTTP-only cookie-based token storage, middleware validation, and structured access control to enhance application security.

The proposed approach effectively reduces common vulnerabilities such as Cross-Site Scripting (XSS), session hijacking, and unauthorized access, while maintainingscalability and efficiency. The framework can be applied to modern web applications to ensure secure and reliable authentication.

References

1.OWASP Foundation. *Cross-Site Scripting (XSS) Prevention Cheat Sheet:* https://owasp.org
2.OWASP Foundation. *Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet:* https://owasp.org
3. Sandhu, R., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). *Role-Based Access Control Models*. IEEE Computer, 29(2), 38–47.
4. Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT) – RFC 7519*. Internet Engineering Task Force (IETF).
5. Provos, N., & Mazieres, D. (1999). *A Future-Adaptable Password Scheme*. USENIX Annual Technical Conference.
6. National Institute of Standards and Technology (NIST). (2020). *Digital Identity Guidelines (SP 800-63)*.

7.OWASP Foundation. *OWASP Top 10 Web Application Security Risks (2021)*.
8.Kindervag, J. (2010). *Build Security Into Your Network's DNA: The Zero Trust Network Architecture*. Forrester Research.
9.Hardt, D. (2012). *The OAuth 2.0 Authorization Framework*. Internet Engineering Task Force (RFC 6749).

10.Mozilla Developer Network (MDN). *HTTP Cookies and Security Best Practices:* https://developer.mozilla.org