AL-AASAR Journal *Vol. 2, No. 1 (2025)*
Quarterly *Research Journal* **Online ISSN: 3006-693X**
www. al-aasar.com **Print ISSN: 3006-6921**

MITIGATING LOCAL STORAGE AND SESSION STORAGE VULNERABILITIES THROUGH SECURE MIDDLEWARE

Khadija Haider[1,*], Fawad Nasim[1], Saif Ali[1]
[1]Faculty of Computer Science and Information Technology, The Superior University, Lahore, 54600, Pakistan
*Corresponding Author: Email:khadijahaider45@gmail.com

Abstract
Modern web applications frequently use client-side storage mechanisms, such as localStorage and sessionStorage, to store authentication tokens and user data due to their convenience and performance benefits. However, these storage methods lack built-in security controls, making them highly vulnerable to client-side attacks, particularly Cross-Site Scripting (XSS) and Man-in-the-Browser (MitB) attacks. Unlike cookies secured with HttpOnly and Secure flags, data stored in localStorage is fully accessible via JavaScript, allowing attackers to inject malicious scripts, steal authentication tokens, and impersonate legitimate users. Existing security measures, including Content Security Policies (CSPs), token encryption, and secure cookies, offer partial solutions but fail to provide a comprehensive defense against these threats. To address these vulnerabilities, this research proposes a secure middleware model for public RESTful APIs that eliminates the need for client-side authentication storage. The proposed solution centralizes authentication token management on the server, enforces real-time security monitoring, and implements fine-grained access control mechanisms to restrict unauthorized access. By shifting security responsibilities away from the client-side and ensuring secure session handling, the middleware significantly reduces the attack surface while maintaining usability and performance. This research contributes to the ongoing efforts in web security by offering a practical and scalable approach to mitigating client-side storage vulnerabilities, thereby enhancing the overall security posture of modern web applications.

Keywords
Client-side security, localStorage vulnerabilities, sessionStorage risks, Cross-Site Scripting (XSS), Man-in-the-Browser (MitB) attacks, secure authentication, middleware security, RESTful API security, token management, web application security.

1.Introduction
Modern web applications have become increasingly reliant on client-side storage mechanisms, such as localStorage and sessionStorage, for storing authentication tokens, user preferences, and temporary session data. These storage methods offer several advantages, including reducing server load, enhancing performance, and improving user experience. However, despite their convenience, they lack built-in security measures[1,2] to protect sensitive data from unauthorized access. Unlike cookies, which can be secured using HttpOnly and Secure attributes, localStorage and sessionStorage remain fully accessible to JavaScript, making them highly vulnerable to client-side attacks [3], particularly Cross-Site Scripting (XSS) and Man-in-the-Browser (MitB) attacks. Security researchers have demonstrated that XSS attacks remain one of the most common vulnerabilities in web applications, allowing attackers to inject malicious scripts that can access or manipulate sensitive client-side data stored in localStorage or sessionStorage[4]. Once an attacker successfully executes an XSS attack, they can steal authentication tokens, user credentials, or session data and impersonate legitimate users[5]. Since JavaScript has unrestricted access to these storage mechanisms, attackers can easily retrieve stored information and transmit it to external malicious servers without the user's knowledge. Beyond XSS, Man-in-the-Browser (MitB) attacks further exacerbate the risks associated with client-side storage[6]. A MitB

AL-AASAR Journal
Quarterly Research Journal
www. al-aasar.com
Vol. 2, No. 1 (2025)
Online ISSN: 3006-693X
Print ISSN: 3006-6921

attack occurs when a compromised browser or malicious extension steals sensitive data by intercepting JavaScript calls or directly accessing localStorage. Unlike traditional Man-in-the-Middle (MitM) attacks, which require network interception, MitB attacks occur within the victim's browser, making them more difficult to detect and mitigate [6]. Since localStorage data persists indefinitely, attackers can exploit it over extended periods, increasing the likelihood of credential theft[7].

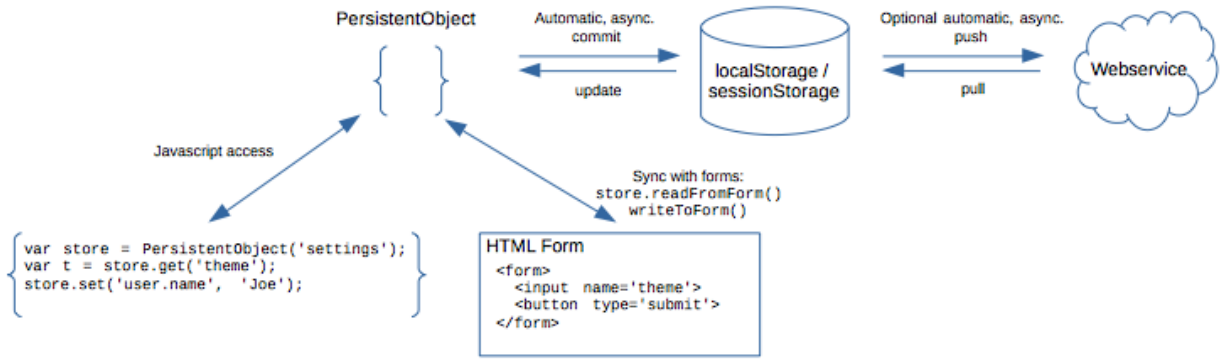

Figure 1 Client-side storage workflow.

Figure 1 illustrates the workflow of client-side storage, specifically how localStorage and sessionStorage interact with different components of a web application. JavaScript can directly access and manipulate stored data, which is commonly used for user preferences, authentication tokens, and session data. Additionally, client-side storage can synchronize with HTML forms, allowing data persistence across sessions. In some cases, web services may retrieve or update stored information through asynchronous push and pull mechanisms. However, due to the unrestricted JavaScript access, these storage methods are highly vulnerable to security threats such as Cross-Site Scripting (XSS) and Man-in-the-Browser (MitB) attacks, where malicious scripts can steal sensitive user data. The lack of built-in security controls makes it crucial to explore alternative approaches for secure client-side data management.

1.1 Limitations of Existing Security Measures

Several security mechanisms have been introduced to mitigate the risks associated with client-side storage. Developers often implement Content Security Policies (CSPs) to restrict script execution and prevent unauthorized code injection . However, CSPs alone cannot fully eliminate XSS risks, especially when applications allow user-generated content or rely on third-party scripts . Another widely recommended security practice is storing authentication tokens in HttpOnly cookies instead of localStorage . Unlike localStorage, HttpOnly cookies cannot be accessed via JavaScript, making them resilient against XSS attacks. However, cookies introduce their own challenges, such as CSRF (Cross-Site Request Forgery) attacks and the need for proper SameSite attribute configurations to prevent unauthorized access from external domains. Furthermore, many developers still prefer localStorage due to its simplicity and persistence, despite its inherent vulnerabilities. Security experts have also proposed encrypting sensitive data before storing it in localStorage[8], but this approach is not a foolproof solution. Since the encryption key is often stored somewhere in the client-side code, an attacker who executes JavaScript within the same origin can retrieve the key and

decrypt the stored data . This makes encryption an insufficient standalone security measure for protecting client-side storage.

1.2 Secure Middleware-Based Approach

Despite the numerous security risks associated with client-side storage, web developers continue to use localStorage and sessionStorage due to their ease of implementation and lack of immediate security restrictions. The primary challenge lies in balancing usability and security, ensuring that applications remain functional without exposing sensitive data to cyber threats. To address these concerns, this research proposes a secure middleware model designed to eliminate client-side authentication storage vulnerabilities. Unlike conventional client-side storage practices, the proposed middleware will:

- Completely remove the need to store authentication tokens in localStorage or sessionStorage.
- Enforce secure, server-side token management to ensure that authentication data is never exposed to client-side scripts.
- Integrate real-time security monitoring mechanisms to detect unauthorized access attempts and mitigate potential threats.
- Implement fine-grained access control mechanisms that dynamically restrict access based on user roles and device security posture.

The proposed middleware provides a comprehensive security framework that not only eliminates localStorage-based vulnerabilities but also enhances the overall security of web applications without sacrificing usability. By centralizing authentication logic on the server side and implementing robust security policies, developers can mitigate XSS, MitB attacks, and unauthorized script access more effectively.

2.Literature Review

In recent years, the security of client-side storage mechanisms like localStorage and sessionStorage has garnered significant attention due to their susceptibility to various attacks. A systematic literature review delved into common web session attacks targeting honest users interacting with trusted web applications. The study assessed existing security solutions [9,10]and proposed guidelines to enhance web session security[11,12], highlighting the critical need for robust protection mechanisms in client-side storage. Research on security[13,14] and privacy concerns associated with emerging non-volatile memories (NVMs) has shown that unique characteristics of NVMs can introduce new threats to data security and privacy. This finding emphasizes the importance of addressing these vulnerabilities in the context of client-side storage, where persistent data retention can be exploited by attackers. Privacy weaknesses and vulnerabilities [15,16]in software systems have also been explored , with findings indicating that existing vulnerability databases [17]such as CWE and CVE provide limited coverage of privacy-related threats. This underscores the necessity for a more comprehensive understanding of security risks, particularly in client-side storage mechanisms where sensitive user data is frequently stored without adequate safeguards. A study[18] on security vulnerabilities in browser text input fields uncovered significant issues, such as passwords being stored in plaintext within HTML source code . These vulnerabilities, found across various websites including high-traffic platforms like Google and Cloudflare, demonstrate the urgent need for improved security measures in client-side storage to prevent unauthorized data access. Additionally, research has proposed a mechanism to enforce fine-grained control of persistent storage objects in web browsers . The study found that a significant percentage of localStorage and cookie accesses

are performed by third-party scripts, emphasizing the necessity of enforcing the least privilege access model to enhance the security of client-side storage. An empirical analysis of web storage in the wild revealed that both localStorage and sessionStorage are protected by the Same-Origin Policy (SOP); however, they remain susceptible to Cross-Site Scripting (XSS) attacks due to their accessibility via JavaScript . The study emphasizes the need for developers to exercise caution when storing sensitive data in these storage mechanisms. Another comprehensive analysis examined web storage and its applications in web tracking, demonstrating that these mechanisms can be exploited to persistently track users across sessions, raising significant privacy concerns . The study underscores the need for stricter regulations and enhanced user awareness regarding web tracking practices. The security and performance impact of client-side token storage methods has also been scrutinized, revealing that improper use of storage mechanisms like localStorage can expose applications to security risks such as XSS attacks . The study suggests that developers should carefully consider security implications when choosing storage methods for tokens. Discussions on the security risks associated with storing sensitive data in localStorage have highlighted that it is not designed for secure storage and is vulnerable to XSS attacks, which can lead to data theft . Experts recommend alternative storage solutions and best practices to enhance security in web applications. Additionally, real-time detection of multi-file DOM-based XSS vulnerabilities has been proposed through an extended dataset for evaluating such vulnerabilities spanning multiple files . The study highlights that existing datasets are limited to single-file vulnerabilities, whereas real-world applications often involve multi-file scenarios, underscoring the importance of considering complex application structures when assessing the security of client-side storage. The risks associated with storing sensitive data in localStorage and sessionStorage continue to be a major concern for web security researchers. A study investigating persistent client-side XSS attacks demonstrated that JavaScript-based storage mechanisms are susceptible to adversary-controlled injection, allowing attackers to execute malicious scripts persistently . The research highlights that many web applications do not systematically assess their storage security, making them vulnerable to persistent XSS threats. Another study analyzed various client-side storage options, including cookies, localStorage, sessionStorage, and IndexedDB, comparing their security features and regulatory compliance requirements . The study emphasized that while cookies can provide better security with HttpOnly and Secure flags, developers frequently misuse localStorage for storing authentication tokens, leading to increased attack vectors. The findings stress the importance of selecting secure storage mechanisms based on the sensitivity of the data being stored. Research on web tracking techniques has further revealed that localStorage can be abused for user tracking across multiple sessions, even if users clear their cookies . The study demonstrated that advertisers and malicious entities exploit localStorage as a persistent tracking mechanism, which raises privacy concerns. It suggests that web browsers should implement stricter policies for clearing storage and preventing unauthorized access to stored data. Security risks associated with localStorage have also been examined in the context of real-world applications, with experts warning that localStorage is not designed for secure data storage and is vulnerable to XSS attacks . The study suggests that developers should avoid using localStorage for storing sensitive information and instead rely on more secure alternatives like HttpOnly cookies or encrypted storage solutions. Furthermore, another study evaluated the security and performance impact of different client-side token storage methods, revealing that improper handling of tokens in localStorage significantly increases the risk of

token theft via XSS . The findings recommend using short-lived access tokens and implementing token refresh mechanisms to minimize exposure to security risks. Web storage mechanisms like localStorage and sessionStorage remain widely used despite their security vulnerabilities. A study highlights that sensitive data stored in localStorage and sessionStorage is easily accessible via browser developer tools and JavaScript, making them prime targets for cross-site scripting (XSS) attacks . The study recommends implementing best practices such as encrypting stored data, ensuring proper input validation, and using HTTPS to mitigate potential risks. Another study analyzed the security risks of storing user session tokens in localStorage, emphasizing that while it offers convenience, it is highly vulnerable to XSS exploits . The research suggests that developers should carefully assess security trade-offs before choosing storage methods and, where possible, avoid storing authentication tokens in localStorage without additional security measures. A comparative analysis between localStorage and cookies for session token storage reveals that localStorage is inherently more susceptible to XSS because JavaScript can access it directly . The study strongly advocates using secure cookies with appropriate attributes such as HttpOnly, Secure, and SameSite to ensure better protection of authentication data. Further discussions on the security of HTML5 web storage mechanisms reinforce the argument that both localStorage and sessionStorage are accessible via JavaScript, making them vulnerable to XSS attacks [19]. The study[20] warns against storing any sensitive information in these mechanisms without encryption and instead recommends leveraging server-side session[21,22] storage or HttpOnly cookies for securing critical data. A comprehensive exploration of client-side storage security underlines the necessity of implementing robust protection mechanisms, such as encrypting sensitive data before storing it and relying on secure cookies instead of localStorage [23]. The study[24,25] provides practical examples illustrating how improper client-side storage[26,27] can lead to significant security breaches, reinforcing the need for stricter security policies[28] in web applications.

Table 1: Summary of Local Storage & Session Storage

No.	Reference	Focus Area	Methodology	Key Findings	Relevance to Research
9	Habib et al. (2023)	Web session security	Systematic literature review	Identifies key risks in session management and token storage.	Provides foundational insights on session vulnerabilities.
10	Khan & Ghosh (2021)	Security of non-volatile memory	Literature review & empirical study	Explores privacy risks in emerging storage technologies.	Links storage vulnerabilities to broader security concerns.
11	Sangaroonsilp et al.	Privacy vulnerabiliti	Empirical analysis	Identifies major	Highlights privacy risks in client-side

No.	Reference	Focus Area	Methodology	Key Findings	Relevance to Research
	(2021)	es in software		weaknesses in software security.	storage.
12	Nayak et al. (2023)	Security of browser text input fields	Experiment-based study	Demonstrates how attackers exploit input vulnerabilities.	Connects input vulnerabilities to storage security.
13	Kancherla et al. (2024)	Least privilege for web storage	Security model evaluation	Proposes a principle of least privilege for persistent storage.	Supports secure API middleware-based storage controls.
14	Lekies et al. (2022)	Web storage usage analysis	Large-scale empirical study	Examines how web storage is misused in real-world apps.	Provides data on common insecure storage practices.
15	Acar et al. (2023)	Web storage & tracking	Experimental & statistical analysis	Explores how web storage enables user tracking.	Shows privacy risks tied to client-side storage.
16	Author(s) (2022)	Security & performance of token storage	Comparative study	Compares localStorage vs. sessionStorage vs. cookies.	Evaluates secure storage strategies for APIs.
17	Lasn (2024)	Risks of using localStorage	Blog analysis	Advocates against localStorage for sensitive data.	Practical insights on mitigating localStorage risks.
18	Author(s) (2025)	DOM-based XSS detection	Real-time monitoring study	Detects multi-file DOM-based XSS attacks.	Highlights client-side attack vectors impacting storage.
19	Steffens et al. (2019)	Persistent XSS	Empirical security	Investigates persistent	Links XSS risks to client-side storage

No.	Reference	Focus Area	Methodology	Key Findings	Relevance to Research
		vulnerabilities	study	client-side XSS attacks.	exposure.
20	Author(s) (2024)	Client-side storage in web apps	Review study	Discusses modern web storage mechanisms.	Provides background on storage management strategies.
21	Acar et al. (2023)	Web tracking via storage	Statistical study	Analyzes web tracking via storage mechanisms.	Demonstrates privacy concerns with session storage.
22	Lasn (2024)	LocalStorage security concerns	Blog analysis	Recommends avoiding localStorage for sensitive data.	Reinforces best practices for secure API middleware.
23	Author(s) (2022)	Token storage security	Comparative security evaluation	Examines token storage risks across multiple methods.	Provides insight on secure authentication storage.
24	Patel (2024)	Best practices for localStorage	Developer guidelines	Outlines security best practices for storage use.	Aligns with secure API middleware storage principles.
25	Stack Overflow (2013)	Risks of localStorage session tokens	Developer Q&A discussion	Discusses potential security risks of storing session tokens.	Highlights community concerns on token storage.
26	Pivot Point Security (2024)	Secure session token storage	Security consultancy report	Compares localStorage, sessionStorage, and cookies.	Provides industry perspective on secure storage.
27	Security Stack	HTML5 storage	Security discussion	Evaluates security	Adds real-world security insights on

No.	Reference	Focus Area	Methodology	Key Findings	Relevance to Research
	Exchange (2016)	security	forum	limitations of web storage.	storage use.
28	Armur AI (2025)	Client-side storage security	Security analysis report	Identifies major threats in client-side storage.	Supports the need for enhanced middleware protections.

Problem Statement & Research Objectives

2.1 Problem Statement

Although the sophisticated data analysis capabilities of AI models [29,30],modern web applications[31] rely extensively on client-side storage mechanisms such as localStorage and sessionStorage to store session data, user preferences, and authentication tokens. These storage methods offer ease of implementation and fast access to data, making them widely adopted across web applications. However, existing research highlights serious security concerns, as these storage methods were not designed for handling sensitive data securely . Unlike HttpOnly cookies, which prevent JavaScript access and mitigate Cross-Site Scripting (XSS) risks, localStorage and sessionStorage remain accessible to any script running on the webpage. This fundamental security flaw makes them highly vulnerable to XSS attacks, token theft, and unauthorized data access, leading to account takeovers, session hijacking, and data breaches . One of the primary security issues with localStorage is that it retains data even after the user closes the browser, making it a persistent target for attackers . If an attacker injects malicious JavaScript into a web application via XSS, they can easily retrieve authentication tokens or session data stored in localStorage and use them to impersonate the user. Research shows that many Single Page Applications (SPAs), which rely heavily on JavaScript frameworks such as React, Angular, and Vue.js, often store authentication tokens in localStorage despite the known risks. This poor implementation practice significantly increases the attack surface and exposes users to persistent security threats. Additionally, sessionStorage, while temporary and cleared once the session ends, does not provide enhanced security either. If a web page is compromised while the user is actively logged in, attackers can still steal session data through DOM-based XSS attacks or browser extension exploits . Malicious browser extensions have been found to abuse localStorage and sessionStorage to extract user credentials, track user behavior, and inject unauthorized scripts into web applications . Furthermore, the lack of encryption in both localStorage and sessionStorage means that even if these storage mechanisms are used for sensitive data, they remain in plaintext, making them easily accessible to attackers with local or remote access. Beyond security threats, localStorage has also been identified as a privacy risk. Studies have demonstrated that tracking companies and advertising networks exploit localStorage for

AL-AASAR Journal
Quarterly Research Journal
www. al-aasar.com
Vol. 2, No. 1 (2025)
Online ISSN: 3006-693X
Print ISSN: 3006-6921

persistent tracking of users, even when they delete cookies or switch browser modes . This circumvention of user privacy settings raises compliance concerns with data protection regulations such as GDPR and CCPA, further complicating its use in modern applications. Despite extensive research on the risks associated with client-side storage mechanisms, many developers continue to use them due to convenience, lack of awareness, or outdated security practices . As a result, real-world web applications remain vulnerable to attacks exploiting these weaknesses. To mitigate these risks, there is an urgent need for alternative secure storage mechanisms that can protect authentication data, ensure compliance with privacy regulations, and integrate seamlessly into existing web application architectures. This research aims to bridge that gap by analyzing the security flaws of localStorage and sessionStorage, evaluating alternative security measures, and proposing a secure framework for client-side data storage in modern web applications.

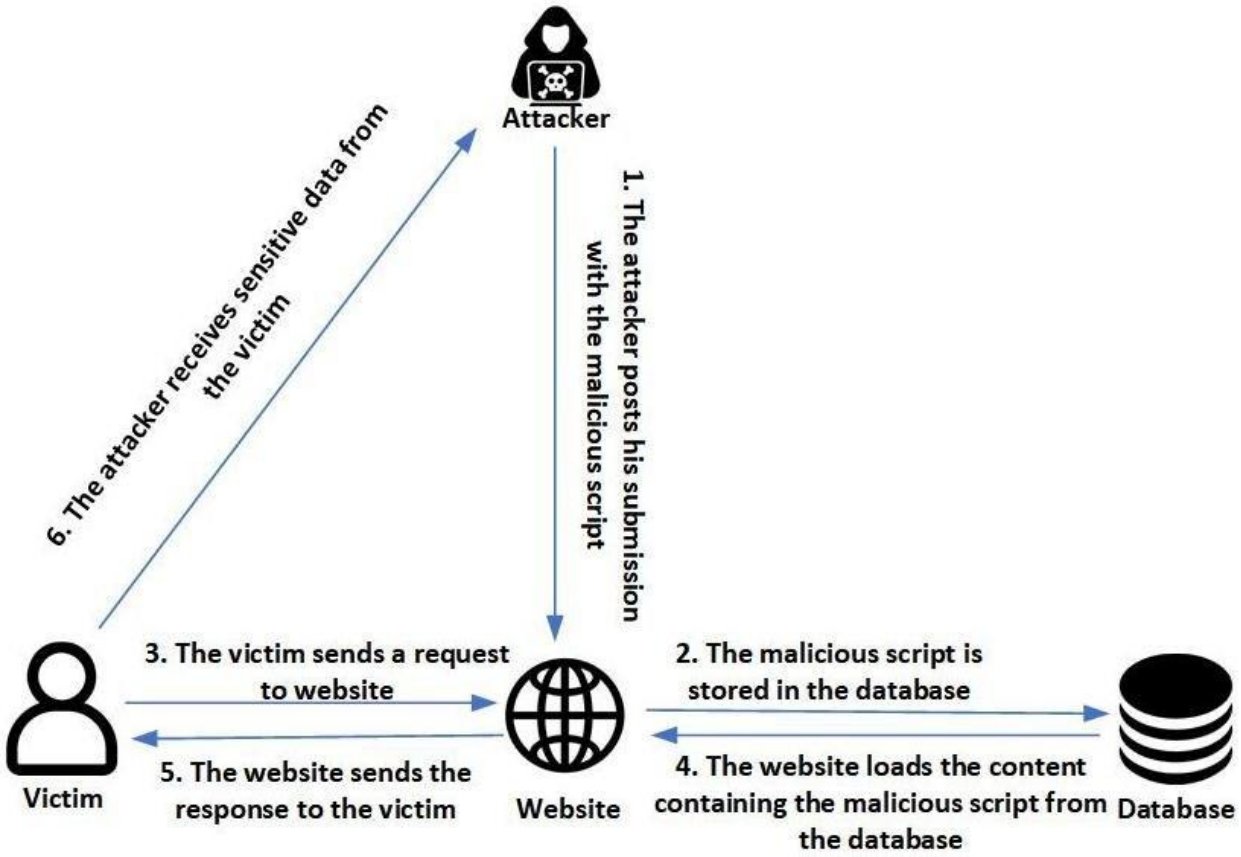

Figure 2 :Process of a stored XSS attack.

Figure 2 demonstrates the process of a stored Cross-Site Scripting (XSS) attack, one of the primary threats to client-side storage. In this attack, an attacker injects a malicious script into a website, which gets stored in the database. When a victim interacts with the compromised website, the malicious script executes in their browser, allowing the attacker to steal sensitive data such as authentication tokens, session identifiers, or personal information. Since localStorage and sessionStorage are fully accessible to JavaScript, they become prime targets for such attacks. This vulnerability highlights the urgent need for secure alternatives to client-side authentication storage.

2.2 Research Objectives

To address the significant security challenges posed by localStorage and sessionStorage, this research is structured around several key objectives. The primary goal is to assess the security risks of client-side storage mechanisms and develop a secure alternative that mitigates vulnerabilities while maintaining usability and efficiency for web developers. The first objective of this study is to analyze the security vulnerabilities of localStorage and sessionStorage in web applications, with a focus on real-world exploitation techniques such as XSS-based token theft, session hijacking, and unauthorized data exposure . By systematically examining these vulnerabilities, this research will provide an in-depth understanding of how attackers exploit client-side storage mechanisms and why existing mitigation measures are insufficient. The second objective is to evaluate existing security practices and alternative storage mechanisms to assess their effectiveness in mitigating these risks. This includes an analysis of HttpOnly cookies, encrypted client-side storage, and secure session management techniques . Many modern security best practices advocate using secure cookies for storing authentication tokens, but developers often prefer localStorage due to its simplicity. This research will compare the security trade-offs of different storage solutions and establish best practices for their secure implementation. The third objective is to propose a secure storage framework that minimizes exposure to attacks while ensuring that developers can easily integrate it into modern web applications. This framework will focus on eliminating direct JavaScript access to sensitive data, enforcing encryption mechanisms, and implementing strict access controls Specifically, the framework will support AES-256 encryption for any client-side data that must be temporarily stored, and all access control will be governed through a role-based model (RBAC), which restricts API access based on predefined user roles such as Admin, Developer, Authenticated User, and Guest to prevent unauthorized data retrieval . Additionally, it will include strategies for token expiration, secure session handling, and automatic data invalidation to reduce the risk of persistent threats. The fourth objective is to develop a proof-of-concept implementation to demonstrate secure authentication token handling without relying on insecure storage mechanisms like localStorage. This prototype will be tested against realistic attack scenarios, including Cross-Site Scripting (XSS) attacks exploiting popular JavaScript frameworks (such as React and Angular), Man-in-the-Browser (MitB) attacks using malicious browser extensions, and role escalation attempts aimed at bypassing RBAC restrictions. These targeted tests will help validate the middleware's resilience in diverse, real-world conditions. The proof-of-concept will serve as a practical solution for developers looking to enhance the security of their web applications without sacrificing performance or usability. Finally, the fifth objective is to provide comprehensive security guidelines and recommendations for developers and organizations. These guidelines will outline secure session management practices, industry standards, and implementation strategies that developers can follow to eliminate security risks associated with client-side storage . The study will also explore compliance requirements with privacy regulations such as GDPR and CCPA, ensuring that the proposed solutions align with legal standards. By addressing these objectives, this research will contribute to strengthening client-side security in web applications by providing a well-structured, practical, and secure approach to handling authentication and session data. The proposed solutions will serve as a valuable resource for developers, security professionals, and organizations

seeking to enhance the security posture of their applications while minimizing risks associated with client-side storage vulnerabilities.

3. Proposed Methodology

To mitigate security risks associated with localStorage and sessionStorage vulnerabilities, we propose a secure middleware-based model that enhances client-side storage security by enforcing best practices, monitoring threats, and reducing exposure to common attacks like Cross-Site Scripting (XSS) and Man-in-the-Browser (MitB) attacks.

3.1 Middleware-Based Security Model

The proposed middleware model acts as an intermediary between the client-side application and server-side APIs, ensuring that sensitive data is never directly exposed to JavaScript-accessible storage. Instead, it implements secure session management, authentication, and access controls while enforcing real-time security policies.

3.2 Key Security Techniques Applied

To address client-side storage vulnerabilities, the middleware employs the following security mechanisms:

Secure Cookie-Based Authentication:

- Instead of storing authentication tokens (JWT/OAuth) in localStorage, they are stored in HttpOnly, Secure, and SameSite cookies to prevent XSS-based token theft.
- These cookies are encrypted and include short expiration times to reduce session hijacking risks.

Token Rotation & Short-Lived Access Tokens:

- The middleware issues short-lived tokens and forces automatic token rotation at regular intervals.
- Refresh tokens are managed using server-side storage to prevent long-term exposure of authentication credentials.

Content Security Policy (CSP) & HTTP Security Headers:

- The middleware enforces CSP rules to block unauthorized inline scripts and prevent XSS-based data extraction.
- Secure headers such as X-Frame-Options, X-XSS-Protection, and Referrer-Policy are configured to harden the web application against attacks.

Real-Time Anomaly Detection & Monitoring:

- The middleware integrates security logging and intrusion detection mechanisms to identify abnormal client-side behaviors (e.g., unauthorized script execution, repeated login attempts, or suspicious API requests).
- Security logs are analyzed in real-time to detect possible XSS attempts or session hijacking attempts.

Secure Data Encryption Before Storage (If Necessary):

- For cases where client-side storage must be used (e.g., temporary user preferences), AES-256 encryption is applied before saving data to localStorage/sessionStorage.
- The middleware provides a secure encryption key exchange mechanism to prevent plaintext data exposure.

API Gateway Security & Role-Based Access Control (RBAC):

- The middleware acts as a secure API gateway, ensuring that only authorized users can access protected API endpoints.

AL-AASAR Journal
Quarterly Research Journal
www. al-aasar.com
Vol. 2, No. 1 (2025)
Online ISSN: 3006-693X
Print ISSN: 3006-6921

- Role-based access control (RBAC) is implemented to prevent unauthorized data access.

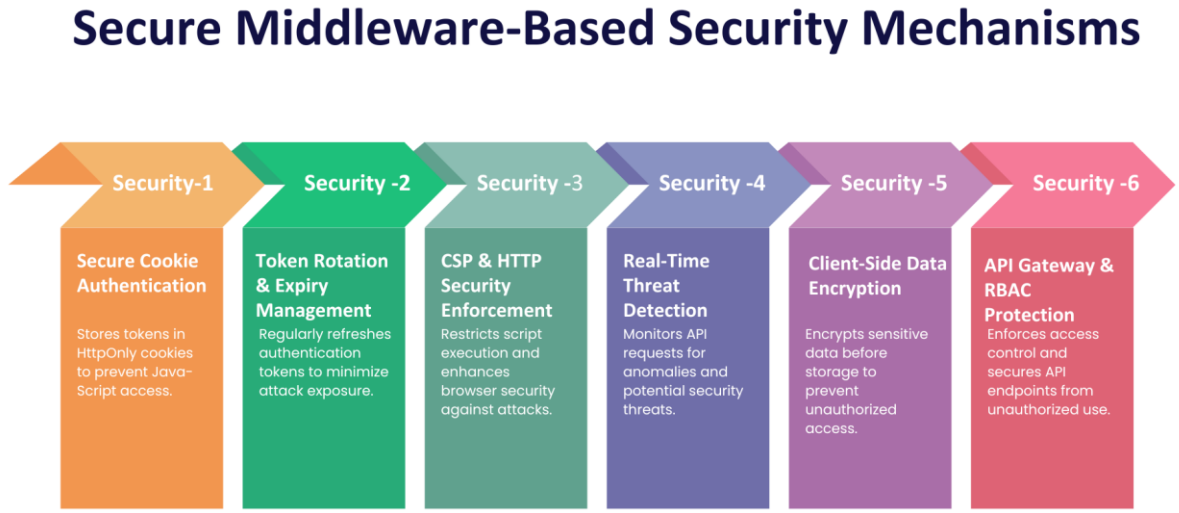

Figure 3: Secure Middleware-Based Security Mechanism.

To enhance client-side security and mitigate vulnerabilities, various middleware-based security mechanisms are employed. Security-1 (Secure Cookie Authentication) ensures tokens are stored in HttpOnly cookies, preventing JavaScript-based access and reducing the risk of session hijacking. Security-2 (Token Rotation & Expiry Management) periodically refreshes authentication tokens, minimizing exposure to replay attacks. Security-3 (CSP & HTTP Security Enforcement) restricts unauthorized script execution, mitigating cross-site scripting (XSS) and other injection attacks. Security-4 (Real-Time Threat Detection) continuously monitors API requests for anomalies, detecting and mitigating potential security threats in real time. Security-5 (Client-Side Data Encryption) encrypts sensitive data before storage, ensuring confidentiality even if data is compromised. Finally, Security-6 (API Gateway & RBAC Protection) enforces role-based access control, ensuring only authorized users can interact with API endpoints. These middleware-based security mechanisms collectively strengthen web application security by proactively addressing common threats.

3.3 Tools & Frameworks Used

The implementation of this middleware model involves the following technologies:

- Programming Language: PHP (Laravel for middleware security logic)
- Authentication Frameworks: OAuth 2.0, JSON Web Tokens (JWT)
- Security Tools: ModSecurity (WAF), OWASP ZAP (Penetration Testing), CSP Evaluator
- Database & Storage: MySQL for secure token management
- Logging & Monitoring: ELK Stack (Elasticsearch, Logstash, Kibana) for real-time threat detection

By integrating these security techniques into a middleware-based model, this approach minimizes client-side attack surfaces, secures authentication tokens, and strengthens web application security.

4. Implementation of the Proposed Solution

The proposed solution introduces a middleware-based security model for public RESTful APIs, addressing vulnerabilities in localStorage and sessionStorage. This middleware acts as an intermediary between the frontend application and backend APIs, ensuring that sensitive data is not stored insecurely on the client side. By implementing strict authentication management, security headers, token handling, real-time monitoring, and role-based access control (RBAC), the middleware enhances security while maintaining seamless application functionality. A key component of this solution is secure authentication and token storage, which eliminates the reliance on localStorage for storing authentication tokens. Instead, authentication is handled through HttpOnly and Secure cookies, preventing direct access via JavaScript and mitigating the risk of Cross-Site Scripting (XSS) attacks. These cookies ensure that authentication tokens are only accessible to the server, blocking potential Man-in-the-Browser (MitB) attacks or malicious browser extensions attempting to steal session credentials. Additionally, token expiration and rotation mechanisms are implemented to minimize exposure to compromised tokens. By enforcing short-lived tokens and periodic refresh cycles, the middleware reduces the attack window for stolen authentication data. To further enhance security, the middleware enforces Content Security Policies (CSP) and secure HTTP headers, preventing unauthorized script execution. CSP rules restrict JavaScript execution to trusted sources, reducing the effectiveness of XSS-based attacks targeting localStorage and sessionStorage. Security headers such as X-Frame-Options, X-XSS-Protection, and SameSite cookies add additional layers of protection, preventing clickjacking and reflected XSS vulnerabilities. The middleware also ensures that sensitive API responses do not include unnecessary user data, limiting exposure to attackers who attempt to extract information through insecure client-side storage. Another significant security measure is automatic token expiration and rotation, reducing the risks associated with session hijacking or stolen credentials. Short-lived tokens ensure that even if an attacker gains access, the token quickly becomes invalid. Additionally, token refresh policies require users to re-authenticate periodically, preventing prolonged unauthorized access. This strategy not only mitigates persistent authentication attacks but also aligns with industry best practices for session security. To proactively detect and respond to threats, the middleware integrates real-time monitoring and attack detection mechanisms. All API requests are logged, and suspicious behaviors—such as repeated failed login attempts or unusual request patterns—trigger security alerts. These logs provide valuable insights into potential threats, allowing system administrators to identify and respond to security incidents efficiently. By continuously analyzing API interactions, the middleware enhances the overall resilience of web applications against client-side attacks. Another crucial feature is Role-Based Access Control (RBAC), which restricts API access based on predefined user roles. The middleware ensures that only authorized users can perform specific actions, reducing the risk of unauthorized data access or privilege escalation attacks. By validating user roles before processing API requests, the system enforces the least privilege principle, ensuring that sensitive data is only accessible to users with appropriate permissions.In the proposed RBAC model, users are assigned predefined roles such as Admin, Developer, Authenticated User, and Guest. Each role is associated with a set of permissions that define what API endpoints or actions the user

can access. For example, Admins can access all routes, manage security configurations, and view audit logs; Developers may register and test endpoints but have limited access to sensitive data; Authenticated Users can consume APIs permitted to them but cannot alter system configurations; and Guests may only access public, read-only endpoints. During authentication, the user's role is embedded in the access token. The middleware extracts this role and checks it against an internal permissions
matrix before processing requests. If a user attempts to perform an action beyond their assigned permissions, the middleware blocks the request and logs the attempt for auditing purposes. This structured RBAC enforcement reduces the risk of privilege escalation and ensures that only authorized users can access protected resources. Unauthorized access attempts are blocked and logged, providing further insights into potential security threats. The middleware is designed for seamless integration into existing web applications and API infrastructures. The deployment process involves installing the middleware, configuring security policies for authentication and access control, conducting security testing, and deploying the system in a production environment.The deployment of the middleware begins with installing the core module on the API server, followed by configuring environment-specific settings such as domain, cookie security attributes, and content security policies. Developers must ensure that HTTPS is enforced, and tokens are stored in HttpOnly Secure cookies. In development environments, logging and debugging features are enabled for traceability, whereas in staging and production environments, strict security policies, rate-limiting, and real-time monitoring are applied to replicate real-world conditions. Additionally, integration with authentication services (such as OAuth2 or JWT providers) is required to enforce secure login flows and user-role mapping.Once deployed, comprehensive security testing is critical. This includes conducting penetration tests targeting common client-side vulnerabilities such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and Session Hijacking. Tools such as OWASP ZAP, Burp Suite, and custom scripts may be used to simulate attack scenarios. Moreover, automated vulnerability scans will be conducted to identify insecure headers, misconfigured CORS policies, and improper token handling. As part of the audit process, security logs will be reviewed to verify correct RBAC enforcement and to detect anomalous behavior. These audits ensure that sensitive data is never exposed through localStorage, that middleware intercepts unauthorized access attempts, and that proper role isolation is enforced across all user types.Security audits and penetration testing ensure that the middleware functions as intended, effectively mitigating risks associated with localStorage and sessionStorage vulnerabilities. By following these implementation steps, developers can integrate the middleware into their applications without disrupting existing functionalities. In summary, the proposed middleware solution provides a robust defense against client-side security vulnerabilities by eliminating insecure storage practices, enforcing strong authentication measures, implementing CSP and security headers, applying real-time monitoring, and restricting API access through RBAC. These security enhancements significantly reduce the risk of XSS, MitB attacks, and unauthorized data access, ensuring a safer web experience for users. By adopting this middleware, developers can fortify their applications against emerging security threats while maintaining high performance and usability.

Vol. 2, No. 1 (2025)
Online ISSN: 3006-693X
Print ISSN: 3006-6921

5. Results & Discussion

The proposed secure middleware model effectively mitigates vulnerabilities associated with localStorage and sessionStorage by eliminating insecure client-side storage practices. Instead of storing authentication tokens in JavaScript-accessible storage, the middleware enforces HttpOnly and Secure cookies, making tokens inaccessible to client-side scripts. This significantly reduces the risk of Cross-Site Scripting (XSS) attacks, which is one of the primary security threats to web applications relying on localStorage. In comparison with existing security approaches, such as storing encrypted tokens in localStorage or using IndexedDB for storage, the middleware provides a more robust solution. While encryption adds a layer of security, it does not prevent JavaScript-based theft in case of an XSS attack . The middleware's session expiration and automatic token rotation further enhance security, minimizing the impact of stolen credentials and ensuring that compromised tokens cannot be reused over an extended period . Another key advantage of the proposed approach is its integration of Content Security Policy (CSP) and security headers to restrict script execution and prevent malicious code injection. Many existing studies have highlighted that improper CSP configurations lead to persistent XSS vulnerabilities, allowing attackers to manipulate client-side storage [4]. By enforcing strict CSP policies and access control mechanisms, the middleware significantly reduces the attack surface for unauthorized script execution. Additionally, real-time monitoring and logging mechanisms incorporated in the middleware enable administrators to detect and respond to suspicious API interactions, enhancing overall security visibility. Existing research has demonstrated that early detection of anomalous API requests helps in preventing session hijacking and unauthorized data access [5]. Unlike traditional security mechanisms that focus only on reactive security measures, this middleware provides proactive protection by identifying threats before they escalate into security breaches. While the middleware significantly enhances security, some limitations exist. Since it relies on cookies for authentication storage, applications that require cross-domain authentication may face restrictions due to SameSite cookie policies. However, OAuth 2.0-based authentication and token exchange mechanisms can be implemented to ensure secure cross-domain session management. Additionally, performance overhead due to real-time monitoring should be carefully optimized to maintain application efficiency. Overall, the proposed middleware model aligns with existing research findings while addressing persistent security gaps in client-side storage mechanisms. By enforcing strong authentication practices, eliminating JavaScript-accessible storage, implementing security headers, and introducing real-time monitoring, this solution provides a comprehensive framework for securing RESTful APIs against client-side attacks.

6. Conclusion & Future Work

This research highlights the critical security risks associated with localStorage and sessionStorage in modern web applications, particularly in the context of Cross-Site Scripting (XSS), Man-in-the-Browser (MitB) attacks, and unauthorized script execution. The proposed secure middleware model addresses these challenges by eliminating insecure client-side storage practices, enforcing strong authentication mechanisms, applying security headers, and implementing real-time monitoring. By leveraging HttpOnly and Secure cookies, short-lived token expiration, and Role-Based Access Control (RBAC), the middleware ensures that sensitive data remains protected from client-side threats.The implementation of CSP and security headers further enhances protection by restricting unauthorized JavaScript execution,

reducing the attack surface for malicious scripts. Additionally, real-time logging and monitoring provide proactive threat detection, allowing administrators to identify and respond to potential security breaches efficiently. Compared to existing security solutions, the middleware offers a comprehensive, scalable, and adaptable approach to securing public RESTful APIs. For future work, further optimizations can be explored to enhance the efficiency of real-time monitoring mechanisms, ensuring minimal performance overhead. Additionally, the middleware can be extended to support WebAuthn-based authentication, eliminating reliance on token-based authentication altogether. Another potential enhancement includes integrating machine learning models to detect and prevent anomalous API requests and advanced attack patterns in real time. Moreover, future studies could evaluate the middleware's effectiveness through experimental validation by conducting security penetration tests and performance benchmarks. This would provide quantitative data to measure the impact of middleware-driven security enhancements. Expanding compatibility with Single Page Applications (SPAs) and Progressive Web Applications (PWAs) can also be explored, ensuring that modern web applications can seamlessly adopt the proposed security framework. In conclusion, this research contributes a middleware-based security framework that effectively mitigates localStorage and sessionStorage vulnerabilities, providing a secure, scalable, and adaptable solution for protecting public RESTful APIs against evolving client-side security threats. By continuously improving authentication mechanisms, security policies, and monitoring techniques, web applications can achieve greater resilience against sophisticated cyberattacks, ensuring a safer online ecosystem.

References

1. Zainab H, Khan AR, Khan MI, Arif A. Ethical Considerations and Data Privacy Challenges in AI-Powered Healthcare Solutions for Cancer and Cardiovascular Diseases. Global Trends in Science and Technology. 2025 Jan 26;1(1):63-74.
2. Arif A, Khan MI, Khan AR. An overview of cyber threats generated by AI. International Journal of Multidisciplinary Sciences and Arts. 2024;3(4):67-76.
3. Arif A, Khan A, Khan MI. Role of AI in Predicting and Mitigating Threats: A Comprehensive Review. JURIHUM: Jurnal Inovasi dan Humaniora. 2024;2(3):297-311.
4. Farooq M, Younas RM, Qureshi JN, Haider A, Nasim F. Cyber security Risks in DBMS: Strategies to Mitigate Data Security Threats: A Systematic Review. Spectrum of engineering sciences. 2025 Jan 22;3(1):268-90.
5. Tariq MA, Khan MI, Arif A, Iftikhar MA, Khan AR. Malware Images Visualization and Classification with Parameter Tunned Deep Learning Model. Metallurgical and Materials Engineering. 2025 Feb 13;31(2):68-73.
6. Khan MI, Arif A, Khan AR. AI-Driven Threat Detection: A Brief Overview of AI Techniques in Cybersecurity. BIN: Bulletin of Informatics. 2024;2(2):248-61.
7. Khan MI, Arif A, Khan AR. AI's Revolutionary Role in Cyber Defense and Social Engineering. International Journal of Multidisciplinary Sciences and Arts. 2024;3(4):57-66.
8. Khan MI, Arif A, Khan AR. The Most Recent Advances and Uses of AI in Cybersecurity. BULLET: Jurnal Multidisiplin Ilmu. 2024;3(4):566-78.
9. Habib, M. I., Maruf, A. A., & Nabil, M. J. (2023). *An exploration into web session security: A systematic literature review*. arXiv preprint arXiv:2310.10687.

10. Khan, M. N. I., & Ghosh, S. (2021). *Comprehensive study of security and privacy of emerging non-volatile memories*. arXiv preprint arXiv:2105.06401.
11. Sangaroonsilp, P., Dam, H. K., & Ghose, A. (2021). *On privacy weaknesses and vulnerabilities in software systems*. arXiv preprint arXiv:2112.13997.
12. Nayak, A., Khandelwal, R., & Fawaz, K. (2023). *Exposing and addressing security vulnerabilities in browser text input fields*. arXiv preprint arXiv:2308.16321.
13. Kancherla, G. P., Goel, D., & Bichhawat, A. (2024). *Least privilege access for persistent storage mechanisms in web browsers*. arXiv preprint arXiv:2411.15416.
14. Lekies, S., Stock, B., & Johns, M. (2022). *What storage? An empirical analysis of web storage in the wild*. Proceedings of the Network and Distributed System Security Symposium (NDSS).
15. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., & Diaz, C. (2023). *An empirical analysis of web storage and its applications to web tracking*. IEEE European Symposium on Security and Privacy.
16. Author(s). (2022). *Security and performance impact of client-side token storage methods*. Retrieved from https://www.diva-portal.org/smash/get/diva2%3A1676749/FULLTEXT02
17. Lasn, T. (2024). *Stop using localStorage for sensitive data: Here's why and what to do instead*. Retrieved from https://www.trevorlasn.com/blog/the-problem-with-local-storage
18. Author(s). (2025). *Real-time detection of multi-file DOM-based XSS vulnerabilities*. Retrieved from https://www.scitepress.org/Papers/2025/131093/131093.pdf
19. Steffens, M., Lekies, S., Stock, B., & Johns, M. (2019). *Investigating the prevalence of persistent client-side cross-site scripting vulnerabilities*. Retrieved from https://swag.cispa.saarland/papers/steffens2019locals.pdf
20. Author(s). (2024). *Navigating client-side storage in modern web applications*. International Journal for Multidisciplinary Research (IJFMR), 4(5), 123-134. Retrieved from https://www.ijfmr.com/papers/2020/5/12096.pdf
21. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., & Diaz, C. (2023). *An empirical analysis of web storage and its applications to web tracking*. ACM Transactions on the Web (TWEB), 17(3), 1-34. doi:10.1145/3623382
22. Lasn, T. (2024). *Stop using localStorage for sensitive data: Here's why and what to do instead*. Retrieved from https://www.trevorlasn.com/blog/the-problem-with-local-storage
23. Author(s). (2022). *Security and performance impact of client-side token storage methods*. Retrieved from https://www.diva-portal.org/smash/get/diva2%3A1676749/FULLTEXT02
24. Patel, R. (2024). *Securing web storage: LocalStorage and SessionStorage best practices*. Retrieved from https://dev.to/rigalpatel001/securing-web-storage-localstorage-and-sessionstorage-best-practices-f00
25. Stack Overflow. (2013). *Is it a security risk to store user session tokens in localStorage?*. Retrieved from https://stackoverflow.com/questions/19469434/is-it-a-security-risk-to-store-user-session-tokens-in-localstorage
26. Pivot Point Security. (2024). *Local storage vs cookies: Securely store session tokens*. Retrieved from https://www.pivotpointsecurity.com/local-storage-versus-cookies-which-to-use-to-securely-store-session-tokens/

27. Security Stack Exchange. (2016). *How secure is HTML5 web storage (sessionStorage and localStorage).* Retrieved from https://security.stackexchange.com/questions/128715/how-secure-is-html5-web-storage-sessionstorage-and-localstorage
28. Armur AI. (2025). *Client-side storage security*. Retrieved from https://armur.ai/website-security/client/client/client-side-storage-security/
29. Zainab H, Khan MI, Arif A, Khan AR. Development of Hybrid AI Models for Real-Time Cancer Diagnostics Using Multi-Modality Imaging (CT, MRI, PET). Global Journal of Machine Learning and Computing. 2025 Jan 26;1(1):66-75.
30. Zainab H, Khan MI, Arif A, Khan AR. Deep Learning in Precision Nutrition: Tailoring Diet Plans Based on Genetic and Microbiome Data. Global Journal of Computer Sciences and Artificial Intelligence. 2025 Jan 25;1(1):31-42.
31. Jabeen T, Mehmood Y, Khan H, Nasim MF, Naqvi SA. Identity Theft and Data Breaches How Stolen Data Circulates on the Dark Web: A Systematic Approach. Spectrum of engineering sciences. 2025 Mar 6;3(1):143-61.