Secure Web Application Development Using ASP.NET Core with JWT Authentication

Bhavesh Katkar[1] Mrs. Vrushali Shinde[2]
[1]Student [2]Guide
[1,2]Master of Computer Applications
[1,2]P. E. S. Modern College of Engineering, Pune, India

Abstract — The rapid adoption of web applications and REST-based services has increased exposure to security threats, including SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), authentication weaknesses, and improper access control mechanisms. This study examines secure web application development using ASP.NET Core together with JSON Web Token (JWT) authentication techniques. ASP.NET Core is one of the leading modern frameworks for developing efficient and secure applications, offering numerous features, including middleware request processing, model validation, dependency injection, and secure database connection via Entity Framework Core. Adding JWT authentication makes ASP.NET Core even more capable of developing safe web apps, thanks to its ability to provide stateless authentication, claims and roles-based identities and authorizations, as well as expiring tokens. Important concepts in the field of secure web development discussed here include safe coding, management of secret keys, encryption of data over HTTPS, refresh token techniques, and OWASP guidelines.

Keywords: ASP.NET Core, JWT Authentication, Web Security, Secure Web Applications, Cybersecurity, RESTful APIs, XSS, CSRF;

I. Introduction

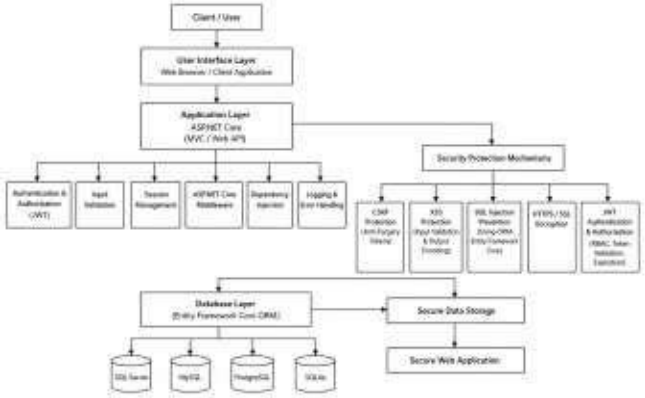
Today, web-based systems are essential components of the digital ecosystem, supporting services such as internet banking, online shopping, healthcare platforms, and social networking applications. Security threats have become a part of everyday life along with cyber attacks on web applications. SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) vulnerabilities may lead to data leakage and breaches of system integrity. The popularity of ASP.NET Core lies in its performance capabilities and security features. Due to its modular design, use of middleware and other features, ASP.NET helps developers reduce possible security vulnerabilities. Features such as input validation, dependency injection, and Entity Framework Core contribute to a more secure development process by minimizing coding errors and reducing potential security flaws. The second component of a secure web application is JWT authentication technology which allows for implementing token-based, stateless authentication and client-server communication as well as role-based authorization and claims-based identity and secure token expiration and refresh process. This research attempts to analyze how web applications based on ASP.NET Core and JWT authentication could be developed.

II. Literature Review

Smith J. and colleagues (2022) proposed a web security framework built on ASP.NET Core that incorporates authentication through security tokens. Their research highlights the use of JWT (JSON Web Token) for stateless authentication in RESTful APIs. This approach improves scalability by eliminating the need for server-side session storage. Security is improved through token validation, claims-based identity, and role-based authorization. The paper also notes how effective ASP.NET Core's middleware and built-in security features are in protecting applications from unauthorized access. Patel R. and Sharma K. (2023) examine common vulnerabilities in modern web applications, including SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). They show how ASP.NET Core counters these threats with model validation, anti-forgery tokens, and secure coding practices. The study explains how Entity Framework Core is used for secure authentication and authorization in ASP.NET Core Web APIs. JWTs encapsulate user identity information and authorization roles, allowing efficient implementation of role-based access management. The authors discuss token expiration, refresh strategies, and secure management of cryptographic keys. They conclude that JWT-based systems outperform traditional session-based authentication in scalability and performance. Anderson P. and Lee M. (2021) focus on secure API development with ASP.NET Core, highlighting the importance of HTTPS enforcement, authentication middleware, and authorization policies. Their work outlines best practices such as secure configuration, input validation, and compliance with OWASP guidelines. The combination of these measures with token-based authentication significantly strengthens web application security. Gupta A. et al. (2023) present a secure web application model that integrates ASP.NET Core with modern security standards. Their research assesses the effectiveness of JWT authentication in preventing session hijacking and unauthorized access. The study also explores the role of logging, monitoring, and exception handling in detecting security threats. Findings indicate that a layered security approach greatly improves application reliability and protection. Kumar S. et al. (2024) investigate the implementation of JWT authentication within ASP.NET Core. Their proposed architecture combines secure coding, token-based authentication, and database security measures. They show that using framework-level security features can substantially reduce application risks.

All rights reserved by www.ijsrd.com

III. SYSTEM ARCHITECTURE



- The Proposed Architecture is a model which provides a service of a secure web application by means of a combined use of ASP.NET Core with a JWT (JSON Web Token) authentication system that utilizes common security features in app building (Secure Programming Practices, Token-Based Authentication, and Secure Database Access). The system design consists of several main components (Modules) as follows;
- User Interface Layer – Handles user interactions (inputs and outputs) through either web pages or client applications. Collects user input, (i.e. login credentials, API requests, etc.). Implements input validation & output encoding techniques and has stored the JWT tokens securely in order to protect the application from client-side attacks.
- Application Layer – Contains all the business logic and implements the model-view-controller (MVC) or web API design principles using ASP.NET core. This layered separation of function and duties improves the overall security and maintainability of the secure web application. Additionally, middleware and dependency injection are also used to enhance overall performance and security.
- Database Layer – Responsible for the secure storage of the application data. ASP.NET Core leverages the Entity Framework Core (ORM) to interact with databases, this in turn helps prevent SQL Injection attacks by utilizing secure data handling techniques (i.e. using parameterized queries). The support database types for this architecture include SQL Server, MySQL and PostgreSQL.
- Security Layer: Various security features come together to form a cohesive development experience:
- Authentication and Authorization: User identification is handled through JWT Authentication. Tokens are sent back to authentic users so they can gain access to restricted resources. RBAC (Role Based Access Control) are used to ensure users only have access to the resources to which they are assigned.
- CSRF Prevention: Anti-forgery tokens prevent unauthorized requests to the system.
- XSS Prevention: Input validation and output encoding help protect against script injection.
- SQL Injection Prevention: Entity Framework Core helps to create safe database queries.
- Secure Communication: Data The transfer of data between the client and the server is done over SSL (HTTPS). This provides a secure communication channel between the client and the server and protects against the unauthorized interception of sensitive information.
- Malicious users may exploit weaknesses in database interactions to manipulate application behavior or gain unauthorized access to information.
- SQL Injection: Attackers take advantage of security vulnerabilities in database queries; they enter artificially created data into a database query with malicious intent. ASP.NET Core will help to prevent this with the use of Entity Framework Core to create parameterized queries. This allows the SQL Server to treat what a user enters into a query as data rather than executable code.Cross-Site Scripting (XSS): Cross-Site Scripting occurs when harmful client-side code is embedded within web content and executed in a user's browser. ASP.NET Core mitigates these risks through input validation and output encoding, with Razor views automatically encoding output for added security.
- Cross-Site Request Forgery (CSFR):
- Cross-Site Request Forgery exploits an authenticated user's session by causing unintended requests to be sent to the application without the user's knowledge.

ASP.NET Core validates the source of a CSRF request through the use of built-in anti-forgery tokens.

A. Authentication Attacks:

By having a weak implementation of authentication, an attacker can gain access to an entire user account. By using JSON Web Tokens (JWT), ASP.NET Core implements secure and stateless (meaning the user's session does not require storage on the server) authentication. This allows ASP.NET Core to support Token-based, Role-based, Token Expiration, and Token Validation as a means of ensuring that only authorized users are permitted to access protected resources.

IV. ADVANTAGES:

- Security Built into the FrameworkASP.NET Core provides integrated security mechanisms that help mitigate threats such as SQL Injection, XSS attacks, and CSRF exploits. Developers do not have to start from zero when creating a secure application because they can utilize the framework's security capabilities.
- High Performance and Speed of Development: ASP.NET Core provides the tools necessary for rapid development and high-performance applications by allowing reuse of components, implementing middleware, and utilizing dependency injection.
- Secure Database Access: Entity Framework Core promotes safer communication with databases by reducing dependence on manually written SQL statements, thereby lowering your risk of SQL Injection attacks, while providing a secure mechanism for communicating with your database.
- Stateless JWT Authentication: JWT allows you to authenticate users securely as a stateless entity; validate

All rights reserved by www.ijsrd.com

tokens issued to users, provide access control (based on roles), and protect API communications.
- Scalability and Flexibility: ASP.NET Core supports integration to different databases, APIs, cloud platforms, and microservices, making it the right solution for enterprises and enterprise-level applications.
- Modern Security Standards: ASP.NET Core's framework promotes the use of secure configuration standards (i.e., HTTPS), implements best practice guidelines provided by the OWASP (Open Web Application Security Project), implements logging (of incidents), monitoring (for anomalies), and exception/error handling capabilities, and thus enhances the security management processes within your organization.

V. LIMITATIONS:

- Misconfigurations in Authentication and Middleware can Lead to Vulnerabilities—and Developers Need to Be Adequately Trained to Avoid These Types of Errors (misconfigurations).
- There Are Significant Risks Associated with Improperly Handling JSON Web Token (JWT) Tokens (token exposure, improper storage, improper expiration, and improper refresh of tokens).
- As JWT are Stateless, There Are Significant Challenges to Immediately Revoking a JWT's Grant of Access (i.e., Immediate Revocation of Tokens Without Using Token Blacklisting Mechanisms).
- Large Systems Have Additional Security Layers (i.e., encryption, API gateway, monitoring) That Increase the Complexity and Time Required to Develop Such Systems.
- Some Security Features May Slightly Affect Performance (i.e., performance overhead) for Systems with High Volume of Concurrent Users Due to Token Validation and/or Encrypting Tokens.
- Protecting the Secret Key(s) Used to Sign and/or Encrypt JWT Is Critical, as Poorly Managed Secret Keys Will Compromise the Security of the JWT.

VI. CONCLUSION:

The findings of this study demonstrate that ASP.NET Core is a reliable platform for developing secure and scalable web applications. The framework provides many included security features that protect applications from several common web application security threats, including SQL Injection attacks, cross-site scripting (XSS), and cross-site request forgery (CSRF). When you combine the framework's built-in security features and the use of secure coding practices, JSON Web Token (JWT)-based authentication, and encryption with the ASP.NET Core development platform, you can develop reliable, scalable, and secure web services. Entity Framework Core contributes to secure database management by encouraging structured and protected data access practices. JWT will provide stateless, secure user authentication based on the role of the authenticated user. The architecture proposed by this paper demonstrates the necessity to use a modern framework with best security practices to establish secure web applications; for example: using an HTTPS connection, implementing appropriate token management practices, and following OWASP standards. Future work will focus on advanced security methods such as use of artificial intelligence for threat detection; automated vulnerability scanning; and improved token management to increase web application security.

REFERENCES:

Research Papers

[1] Feric Z., et al. (2021). A Secure and Reusable Software Architecture for Supporting Online Data Harmonization. IEEE Big Data Conference.
[2] Duisebekova K., et al. (2021). Django as Secure Web Framework in Practice. Bulletin of KazATC.
[3] Steinhauser A., Tuma P. (2019). DjangoChecker: Detection of Context-Sensitive XSS Flaws in Django Applications.
[4] Kumar M., Nandal R. (2024). Python's Role in Accelerating Web Application Development with Django. IRJAEM.
[5] Alhaag A. A. A. (2025). Secure Web Application Design: Django and Laravel Frameworks. Scientific Journal of Science and Technology.
[6] Ruohonen J. (2018). An Empirical Analysis of Vulnerabilities in Python Packages for Web Applications. arXiv preprint arXiv:1810.13310.
[7] Farasat T., et al. (2024). SafePyScript: A Web-Based Solution for Machine Learning-Driven Vulnerability Detection in Python. arXiv preprint arXiv:2411.00636.
[8] Kiashemshaki K., et al. (2025). Secure Coding for Web Applications: Frameworks, Challenges, and the Role of LLMs. arXiv preprint arXiv:2507.22223.
[9] Jain G. (2023). Secrets of Security in a Django Application. Medium Article.

All rights reserved by *www.ijsrd.com*