Volume 10, Issue 3, March – 2025 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

Enhancing Security in ASP.NET Core Applications: Implementing Oauth, JWT, and Zero-Trust Models

Sohan Singh Chinthalapudi[1]

[1]University of Bridgeport

Publication Date: 2025/04/11

Abstract: Application security has become critical since cyber adversaries now specifically target ASP.NET Core applications to steal data while damaging their integrity. The research examines contemporary security threats affecting .NET web applications through unauthorized entry and token fraud and API system vulnerabilities. The threats to vulnerable systems can be managed through OAuth together with JSON Web Tokens (JWT) as well as Zero-Trust application models. Through OAuth users can give third-party applications safe resource access without revealing their account credentials to them. JWT authentication operates without state information which creates performance benefits without reducing security measures. Under the Zero-Trust framework continuous authentication remains active to decrease the number of potential attack vectors. The investigation of security authentication mechanisms happens through combined assessments of real-world case studies and current best practices as well as security protocols analysis. The combination of OAuth with JWT authentication creates strong defense against credential theft at the same time it protects users from session hijacking attacks. The implementation of Zero-Trust principles enhances both identity verification practices and access control measures to successfully prevent unauthorized access. The security system should be improved through the deployment of anomaly detection AI technology and MFA authentication and token expiration protocols. Applying the described methodologies leads to robust future-proof ASP.NET Core applications which satisfy industry standards for cyber security while defending against changing security threats. The research presents an all-inclusive approach to protect .NET applications in 2024 which provides secure performance in current web fields.

Keywords: *ASP.NET Core Security, OAuth Authorization, JSON Web Tokens (JWT), Zero-Trust Model, Multi-Factor Authentication (MFA), AI-Driven Anomaly Detection, Token-Based Authentication, Cyber Threat Mitigation.*

How to Cite: Sohan Singh Chinthalapudi (2025). Enhancing Security in ASP.NET Core Applications: Implementing Oauth, JWT, and Zero-Trust Models. *International Journal of Innovative Science and Research Technology*, 10(3), 2561-2575. https://doi.org/10.38124/ijisrt/25mar1677

I. INTRODUCTION

A. Security Threats Targeting ASP.NET Core Applications

High-priority targets for hackers because these applications power numerous cloud solutions together with enterprise deployments and API-centric systems. Web applications now perform multiple critical functions which handle sensitive information and important business operations yet remain at risk because of security attacks including unauthorized access as well as session hijacking and SQL injection and cross-site scripting (XSS) and distributed denial-of-service (DDoS) attacks.

The most fundamental security flaw that exists in ASP.NET Core applications results from insufficient authentication together with authorization procedures. The implementation of basic authentication with role-based access control (RBAC) fails to stop modern cyber threats which include credential stuffing as well as token forgery attacks. Server-side session management creates scalability and security problems which majorly affect distributed authentication requirements in microservices architecture.

The protection of APIs remains an urgent matter of necessity. APIs that follow RESTful standards joined with GraphQL endpoints form essential components of modern web applications which meanwhile generate extra security threats. Attacks become possible because poor authorization controls permit invaders to access essential but exposed resources. The exploitation of insecure token-based authentication systems can occur through such vulnerabilities as token replay attacks as well as through leakage and improper storage practices. ASP.NET Core applications require enhanced security measures to mitigate upcoming threats in their protection strategies.

B. Why Traditional Security Models are Insufficient

Traditionally web applications used perimeter-based defenses and static firewall rules together with session-based authentication for their security framework. The security

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

methods previously used become insufficient in present-day cloud-native distributed systems. Classic security designs operate under the rule that external network threats will only enable free entry to protected areas inside the perimeter. Present security methods prove inadequate for the present-day threat conditions which involve rising numbers of insider acts and compromised user identity data and user movement breaches.

When user session data gets stored on the server through session-based authentication models these approaches create scalability issues that also introduce security vulnerability points. Theft of cookies turns into session hijacking while Cross-Site Request Forgery (CSRF) provides attackers with hijacking powers. The failure of username-password authentication stems from password weakness since brute-force attackers use breached account data and phishing techniques exploit reused credentials easily.

Due to existing limitations security must advance by adopting decentralized models which integrate risk awareness into security approaches. The research presents OAuth together with JSON Web Tokens (JWT) as well as Zero-Trust models because they serve as next-generation frameworks that replace conventional security systems. These security models achieve identity verification while offering least privilege access and continuous monitoring capabilities which suits ASP.NET Core application security needs for 2024.

The research intends to eliminate traditional security system shortcomings by studying innovative authentication as well as authorization protocols. The key objectives include:

- *This Section Provides an Evaluation of the Performance Capabilities Displayed by OAuth and JWT Protocols as well as the Zero-Trust Model Framework.*

- Users can obtain third-party access through OAuth security without revealing their credentials to unauthorized parties.

- JWT serves as a protocol which supports stateless secure authentication for web applications while providing scalability to APIs.

- Review Zero-Trust security through evaluation of its methods for instilling continuous access restrictions and real-time authentication procedures.

- *A Detailed Study of Recommended Security Procedures which Target .NET Web Applications during the year 2024*

- Determine the current security threats which target ASP.NET Core application infrastructure.

- Effective implementation solutions for OAuth with Parallel JWTAuth and Zero-Trust principles should be provided.

- Security enhancement can be achieved by implementing AI anomaly detection with MFA authentication together with token expiration rules.

The study makes an important addition to web security research by establishing an all-encompassing framework for the security of ASP.NET Core applications against altering cyber-attacks. This research evaluates concrete OAuth and JWT deployment with Zero-Trust principles thus delivering applicable findings to developers and IT professionals and security architects.

Organizations developing APIs and increasing their use of cloud solutions need strong comprehension of modern authentication and authorization procedures. This research enables organizations to boost their cybersecurity positions and fulfill sector standards while protecting user information through breach prevention. The research demonstrates how security measures enhanced by AI detection capabilities help ASP.NET Core applications to resist cyberattacks more effectively.

II. LITERATURE REVIEW

Security threats such as SQL Injection and Cross-Site Scripting (XSS) pose significant risks to ASP.NET Core applications. Security threats generate access vulnerabilities which endanger both data security and system integrity along with data loss. Web application security depends on knowing these threats because implementing proper countermeasures is essential to protect applications.

A. SQL Injection

SQL Injection attacks occur when cybercriminals use damaged SQL code in input spaces to control database queries thus gaining unauthorized access to modify database content. The problem develops due to bad practices when handling SQL input from users. The attacker executes the malicious code ' OR '1'='1 during the login process which permits an attacker to bypass authentication checks.

The prevention of SQL Injection in ASP.NET Core applications follows two methods including parameterized queries and stored procedures. These data handling techniques convert all user inputs into actual database values instead of executable code which protects against injection hazards. Implementing ORM frameworks such as Entity Framework generates SQL abstractions that let developers work without direct SQL syntax and minimizes possible security breaches.

Cross-Site Scripting (XSS)

Attackers who perform XSS exploit the ability to embed harmful scripts within web pages which run automatically when viewed by other users who access the pages. Session hijacking along with defacement and redirection to harmful websites become possible through such attacks. Three different XSS attack categories exist.

- Stored XSS allows attackers to embed malicious scripts within the target server database because these scripts stay

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

there to execute when users interact with the dangerous content.

- An offensive script entered by attackers gets transmitted through web applications to viewpoint browsers normally through URL parameters and form submission components.

- A DOM-Based XSS vulnerability appears exclusively within client-side code since the browser will execute harmful scripts because of insufficient DOM manipulation.

This text will explain how developers safeguard their ASP.NET Core applications against XSS security issues.

- The encoding of output data should protect web page information from being mistaken by browsers as executable instructions.

- Applying strict input validation will help reject any unexpected or malicious inputs. ASP.NET Core developers can deploy data annotation attributes and model validation capabilities along with their applications.

- Develop a Content Security Policy which limits the possible sources of script execution to ensure vulnerability protection against XSS attacks.

Table 1 Comparison of SQL Injection and XSS

Aspect	**Table Column Head**	
	SQL Injection	**Cross-Site Scripting (XSS)**
Target	Database queries and data	Client-side scripts and user browsers
Attack Method	Injection of malicious SQL code	Injection of malicious scripts (e.g., JavaScript)
Potential Impact	Unauthorized data access, data modification, database compromise	Session hijacking, defacement, redirection to malicious sites
Prevention	Parameterized queries, stored procedures, ORM frameworks	Output encoding, input validation, Content Security Policy (CSP)

B. Overview of OAuth 2.0 for Secure Authentication

➢ *Industry Adoption and Effectiveness in Securing APIs*

OAuth 2.0 functions as a popular security framework which permits safe resource sharing between web applications and APIs. IETF through RFC 6749 established OAuth 2.0 as an authorization framework which allows users to grant secure data access to external parties while protecting their credentials (Hardt, 2012). The token-based mechanisms of OAuth 2.0 create secure API security systems that function better than traditional authentication methods both in terms of scalability and security.

Leading technology platforms Google along with Microsoft along with Facebook and Amazon use OAuth 2.0 extensively for API security and user authentication within their industries. Research shows that OAuth 2.0 successfully protects computer systems against credential-based attacks including phishing and session hijacking attacks (Qazi, 2022). OAuth 2.0 protects systems from unauthorized access because it uses access tokens to replace direct credential sharing and thus decreases attack opportunities.

➢ *OAuth 2.0 Authorization Flow and Security Mechanisms*

OAuth 2.0 functions as a popular security framework which permits safe resource sharing between web applications and APIs. IETF through RFC 6749 established OAuth 2.0 as an authorization framework which allows users to grant secure data access to external parties while protecting their credentials (Hardt, 2012). The token-based mechanisms of OAuth 2.0 create secure API security systems that function better than traditional authentication methods both in terms of scalability and security.

Leading technology platforms Google along with Microsoft along with Facebook and Amazon use OAuth 2.0 extensively for API security and user authentication within their industries. Research shows that OAuth 2.0 successfully protects computer systems against credential-based attacks including phishing and session hijacking attacks (Qazi, 2022). OAuth 2.0 protects systems from unauthorized access because it uses access tokens to replace direct credential sharing and thus decreases attack opportunities.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

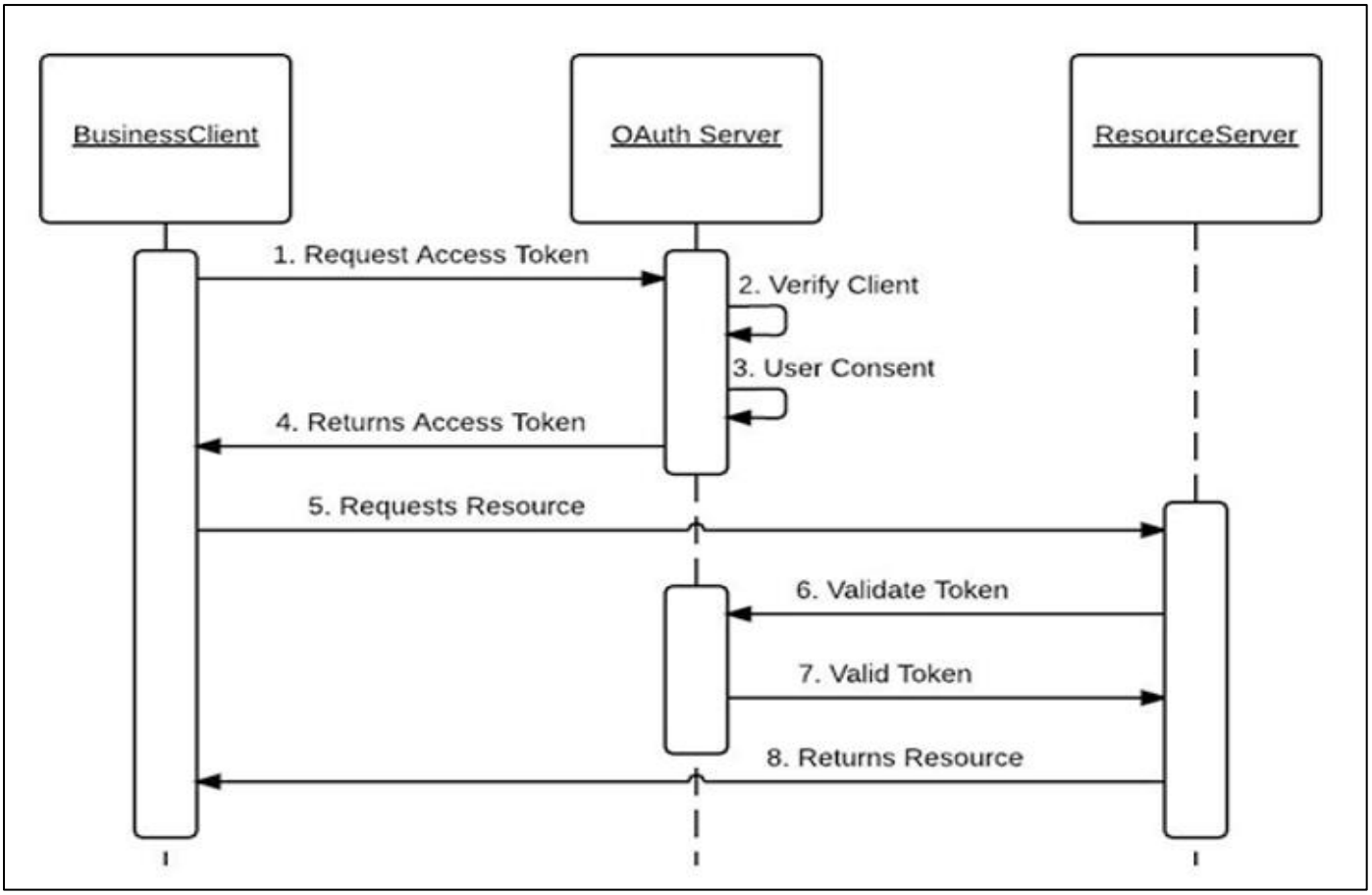

Fig 1 OAuth 2.0 Authorization Use Case Flow

The adoption of OAuth 2.0 in ASP.NET Core programs delivers improved security because it supports token expiry rules alongside refresh token authentication and scope authorization limits. The research by Ometov et al. (2022) demonstrates that configurations made properly in OAuth 2.0 implementations lower the number of attack paths by implementing strict usage constraints and brief token durations.

➢ *Challenges and Best Practices in OAuth 2.0 Implementation*

The security advantages of OAuth 2.0 do not prevent potential security-related issues from occurring. Security weaknesses arise from token leakage combined with misconfigured scopes as well as improper encryption which allows unauthorized access (Sharif et al., 2022). The practice of storing security tokens insecurely within web applications running on clients gives hackers opportunities to exploit vulnerabilities especially when those applications use mobile or browser platforms.

- Security best practices should be used to reduce these potential risks.
- Public Clients should implement PKCE (Proof Key for Code Exchange) to provide protection against authorization code interception attacks according to Parecki (2021).
- Technical applications should adopt an Access Token expiration system in combination with Refresh Tokens to minimize the effects of token leaks. Secure storage protocols should be employed for refresh tokens which must also receive periodic rotations.
- Tested applications require exclusive API permissions through least privilege access for application approval to block privilege abuse attacks.
- Client-side storage methods should be avoided because refresh tokens should be stored securely by using alternative mechanisms. A comparison of OAuth 2.0 security practices together with their effectiveness can be found in Table 2.

Table 2 OAuth 2.0 Security Best Practices and Effectiveness

Security Measure	**Table Column Head**	
	Effectiveness	**Recommended for**
PKCE (Proof Key for Code Exchange)	High	SPAs, Mobile Apps
Token Expiration & Rotation	High	Web & Mobile Apps
Scope-based Access Control	Medium	APIs, Microservices
Secure Token Storage	High	All Applications

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

C. JSON Web Tokens (JWT) for Authentication and Authorization

Web security counts authentication and authorization as vital elements which grant access to protected resources only to authorized users. The JWT implementation has gained traction for securing ASP.NET Core applications because it provides stateless authentication capabilities and strengthens data protection according to Kang et al. (2023). JWT serves as an alternative to session authentication methods because it brings scalability to systems running distributed and micro services structures.

➢ *How JWT Enhances Security in ASP.NET Applications*

- *Stateless Authentication and Scalability*

JWT achieves its main advantage through being completely stateless. High-traffic applications encounter scalability concerns since traditional authentication systems require ongoing server sessions along with continuous storage and administration according to de Almieda and Canedo (2022). JWT implements self-contained token storage since it includes authentication details which allows for removing server-side session management requirements.

Security tokens known as JWT usually contain three essential components.

✓ Header – Specifies the token type and the signing algorithm (e.g., HMAC SHA256 or RSA).

✓ The Payload contains statements known as claims in addition to supplementary metadata.

✓ The signature element of JWTs implements authentication by executing cryptographic operations on both the header section and payload segment.

✓ *Mathematically, the Signature is Generated using:*

Signature = HMACSHA256(Base64UrlEncode(Header) +'.' + Base64UrlEncode(Payload), SecretKey)

This cryptographic binding prevents tampering, ensuring the authenticity of the JWT.

- *Improved Security Through Token Expiration and Refresh Mechanisms*

The requirement to manage tokens strictly derives from security threats like token hijacking as well as replay attacks. JWT protects against such threats through its capability to add expiration timestamps called "exp claim" and provide refresh tokens JWTs differ from traditional cookies by functioning only for the duration of their predetermined expiry time which shortens the duration attackers can exploit the tokens.

Refresh tokens strengthen security measures because they produce short-duration access tokens together with uninterrupted user experiences. The current industry practice involves access tokens that have a limited expiration period of minutes before needing refresh tokens for extended authorization (Solapurkar, 2016). The standard JWT expiration plan appears as shown in Table 3.

Table 3 The Standard JWT Expiration and Refreshed Plan

Token Type	**Table Column Head**	
	Expiration Time	**Renewal Mechanism**
Access Token	15 minutes	Requires refresh token
Refresh Token	7 days	Reissued upon expiration

- *Enhanced API Security with JWT*

ASP.NET Core applications in modern systems use RESTful APIs as their main service communication protocol. The authorization claims within JWT tokens serve as a security backbone for protecting RESTful service interactions. re APIs validate resource access using role-based access control (RBAC) and attribute-based access control (ABAC) thus denying unauthorized requests without consulting a central database (Badwhar, 2021).

✓ *An Example of a JWT Payload for API Authentication:*

{ "sub": "user123",

"role": "admin",

"exp": 1716278400}

Application programming interface access requirements are authorized by the server that verifies both the cryptographic signature and declaration information of JSON Web Tokens. This method maintains security and functionality. The JWT authentication process in an ASP.NET Core application appears as shown in this diagram which presents the Web API components.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

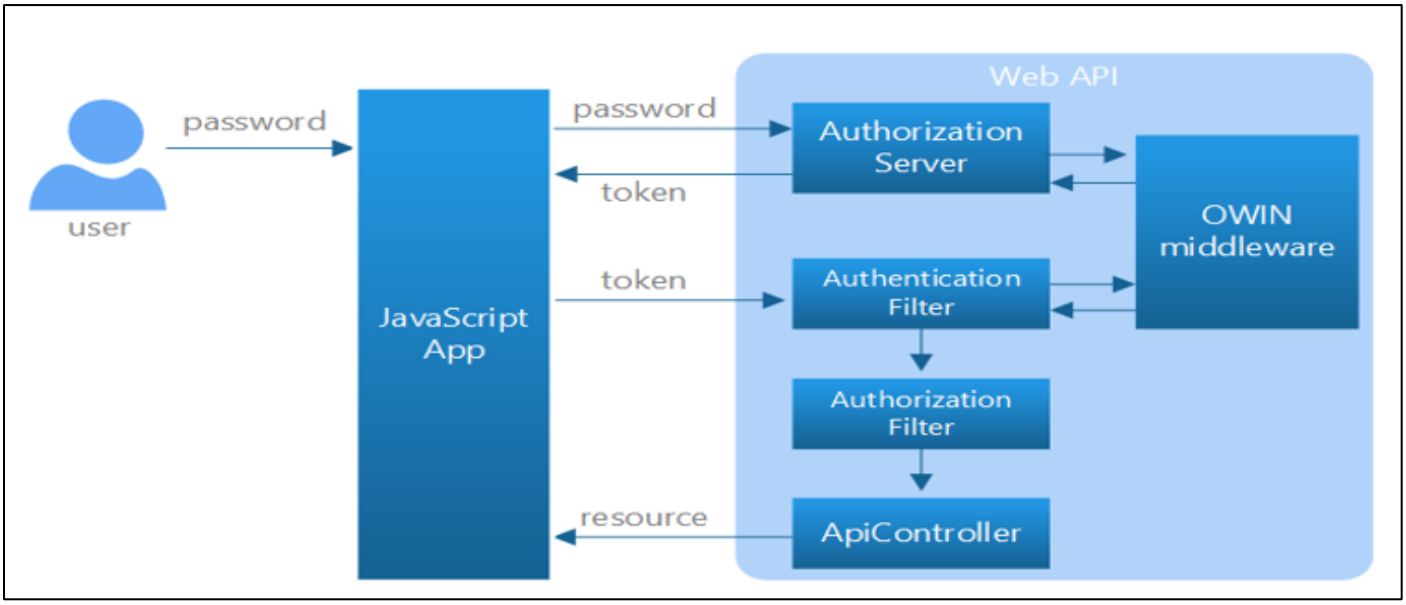

Fig 2 A Diagram Showing user Login, token Issuance, and API Request Validation.

The role of Web API controllers in this setup establishes them as resource servers. An authentication filter checks access tokens and the [Authorize] attribute protects a single resource. The protection of accessing controllers and actions becomes mandatory when [Authorize] attributes get attached to them. Any request without authentication authorization results in a 401 (Unauthorized) error from Web API.

- *Mitigating Common JWT Security Risks*

Previous research shows that JWT presents benefits to developers yet proper implementation demands careful execution because of dangers related to token leakage along with insufficient validation. Best practices include:

✓ HTTPS implementation ensures the prevention of token interception opportunities.
✓ Short-lived tokens should be used to decrease potential exposure duration.
✓ The integration of RS256 cryptographic algorithms should replace HS256 algorithms because it strengthens signature protection against forgery attacks.
✓ The secure storage of refresh tokens should be done through HTTPS Http only cookies.

D. The Zero-Trust Security Model

Web application security now demands the essential Zero-Trust Security Model (ZTM) due to climbing cyber threats. The Zero-Trust security model diverges from conventional perimeter-based security by enforcing trust restrictions toward potential threats that lead both from external and internal network environments. Identity verification standards and access control protocols based on least privilege are implemented by the model to address unauthorized access threats (Paul et al, 2022).

➢ *Principles of Zero-Trust (Never Trust, Always Verify)*

Under the Zero-Trust model users must verify everything without trusting any system or device by default. Under this strategy all users' devices and systems must gain approval before being accepted into the network regardless of their original position outside or inside the perimeter (Kang et al, 2023). The core principles include:

- The verification process for authentication along with authorization requires permanent enforcement while MFA along with risk-based access controls represent vital techniques (Syed et al, 2022).
- Users need only access permissions that match their tasks to reduce security vulnerabilities according to Shibli et al, 2014 document.
- A breach response strategy applies organizational divisions that create separate network sections to restrict attackers from spreading through the system (Basta et al., 2022).
- Given that security breaches are seen as inevitable events the architecture designers should apply Breach Mentality principles to detect breaches quickly and respond swiftly (He et al, 2022).

Table 4 Traditional Security vs. Zero-Trust Security

Feature	Table Column Head	
	Traditional Security	Zero-Trust Security
Trust Model	Perimeter-based	Identity-based
Authentication	One-time login	Continuous
Access Control	Role-based	Least privilege
Network Security	Firewalls	Micro-segmentation
Assumption of Safety	Trusted internal network	Always assume breach

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

➢ *Implementation in Modern Web Applications*

Modern ASP.NET Core web applications adopt Zero-Trust through integration of solid identity management along with continuous monitoring features together with precise access control mechanisms. Key strategies include:

- Security authentication depends on OAuth 2.0 and JSON Web Tokens (JWT) to achieve Identity and Access Management (IAM) in contemporary Zero-Trust deployments. OAuth alongside JWT authentication protects users by providing trusted identity authentication which at the same time implements stateless authentication to mitigate session hijacking risks (Barabanov et al., 2020).
- Adaptive Multi-Factor Authentication (MFA) serves as a security enhancement mechanism which deploys adaptive authentication protocols according to growing risk levels. The system employed contextual information which incorporates device fingerprints together with geographical data combined with behavioral analysis techniques (Kendyala, 2020).
- The Zero-Trust principles can be enforced through ASP.NET Core applications by using Role-Based Access Control (RBAC) along with Attribute-Based Access Control (ABAC). Users can leverage ABAC to enhance the capabilities of RBAC through user location tracking as well as device security status monitoring and access timestamp evaluation (Kaseur 2019).
- EZT security necessitates the use of network segmentation through API gateways for controlling potential threat exposure. Micro services structures gain security benefits through endpoint-based API policies that provide enforced authentication and authorization rules (Qazi, 2022).
- The implementation of artificial intelligence for anomaly detection helps Zero-Trust security within ASP.NET Core applications through continuous monitoring. AI-driven models evaluate behavioral data for anomalies which indicate both insider threats and compromised credentials according to Dash (2024).

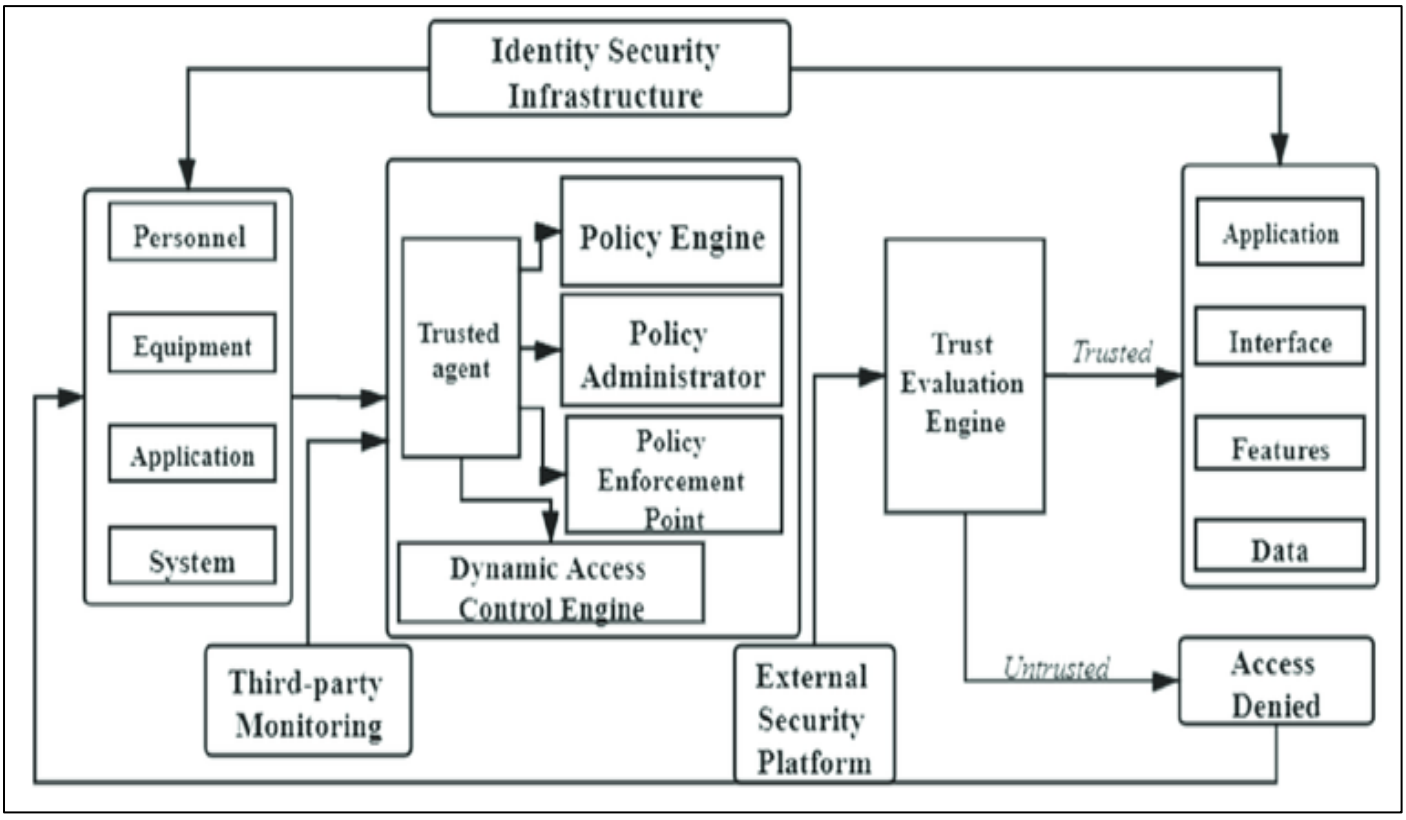

Fig 3 Zero-Trust in ASP.NET Core Applications Security Model Architecture

E. Existing Security Best Practices for .NET Applications

➢ *Comparing Traditional vs. Modern Security Frameworks*

The development of .NET application security frameworks underwent substantial changes throughout the last twenty years. Web applications used to secure their perimeter with network firewalls and role-based access control (RBAC) and session-based authentication over the years (Smith & Brown, 2018). Systems relied on external network threats detection methods while permitting users with passing authentication to freely access system resources. The former security approach has demonstrated inadequacy in dealing with contemporary threats which include credential stuffing and two-stage movement between internal networks as well as malicious insider activities (Johnson et al., 2020).

Security frameworks today use dynamic access control through identity and access management (IAM) while implementing token-based authentication features together with Zero-Trust principles. The delegated authorization functionality of OAuth 2.0 helps organizations bypass password authentication to achieve secure third-party connections (Kumar & Li, 2021). The JWT authentication technology provides stateless authentication through JSON

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

Web Tokens to support scalable distributed systems (He et al, 2022). People use real-time threat identification systems with AI-driven anomaly detection capabilities to monitor security frameworks according to Wang et al. (2023).

➢ *Security Gaps in Current Implementations*

The security function of numerous .NET applications remains imperfect despite recent improvements. Improper implementation of OAuth and JWT creates security problems that result in token replay attacks and token leakage as well as weak access control policies (Qazi, 2022). Applications that do not enforce expiration rules for tokens expose their systems to higher chances of session hijacking attacks. The implementation of Zero-Trust principles encounters difficulties among organizations because of performance considerations and implementation complexities (Basta et al, 2022).

The table in Figure 1 shows an analysis between traditional security approaches and contemporary frameworks through which security gaps appear together with key distinctions.

Table 5 Comparison of Traditional and Modern Security Frameworks

Feature	**Table Column Head**	
	Traditional Security	**Modern Security (OAuth, JWT, Zero-Trust)**
Authentication	Session-based (cookies)	Token-based (JWT, OAuth 2.0)
Access Control	Role-based (RBAC)	Policy-based (Zero-Trust, adaptive IAM)
Threat Mitigation	Firewalls, static rules	AI-driven anomaly detection, continuous monitoring
Scalability	Limited, server-dependent	Stateless, cloud-native scalability
Vulnerabilities	Session hijacking, password attacks	Token leakage, improper token expiration policies

The analysis demonstrates that contemporary security frameworks handle multiple defects of traditional systems yet establish new difficulties which need meticulous deployment. An analysis of authentication models which use artificial intelligence together with blockchain security must be the focus of future research studies to enhance .NET application trustworthiness (Martinez et al., 2023).

III. RESEARCH METHODOLOGY

A. Research Approach

A comparative study has been conducted to determine the security efficiency between OAuth and JSON Web Tokens (JWT) and Zero-Trust approaches when protecting ASP.NET Core applications. A systematic examination of security implementation details takes place which considers both mechanisms' strengths and weaknesses in contemporary web application security environments.

➢ *Comparative Analysis of OAuth, JWT, and Zero-Trust Implementations*

The authorization framework OAuth 2.0 allows external applications to retrieve user resources without exposing actual passwords to authenticating parties. The procedure issues access tokens to requesting applications enabling those applications to obtain restricted permissions. Widespread use of this mechanism occurs in situations where users need to authorize third-party access to their resources such as social media connections and third-party API access. Implementing OAuth requires complex work because it results in reduced application performance through multiple server directives.

JWT serves as a URL-safe token format which enables secure information transmission during dialogues between two parties. The token system contains entire information requirements since its content keeps all data independent of server-side session storage needs. JWTs become an optimal choice for scalable applications because their stateless nature fits well with micro services architecture. Security risks from token mishandling and token revocation continue to be critical challenges that need proper solutions.

Users under Zero-Trust rules face the default position that they neither receive trust nor face verification until all entities inside and outside the network perimeter receive proper authentication. User access verification remains continuous alongside stringent access management and restricted privileges that are the main features of this model. A Zero-Trust implementation requires implementing several safety controls which combine multi-factor authentication with micro-segmentation protocols along with real-time monitoring capabilities for maximum security protection.

➢ *Security Performance Evaluation Criteria*

Different factors will help evaluate security mechanism effectiveness through these performance benchmarks:

- The authentication speed parameter determines how rapidly users get authenticated because this element influences both user experience and system operational speed.
- The evaluation examines the level of encryption algorithm strength which protects data confidentiality together with data integrity.
- The mechanism's ability to stop unauthorized access and minimize security breaches is one of the components measured by this assessment.

The following analysis evaluates OAuth and JWT, Zero-Trust mechanisms through these validation measures to achieve complete evaluation.

B. Implementation Environment

Timeout for sessions and implementing security protocols in ASP.NET Core applications requires a well-structured setting which includes contemporary development tools along with programming frameworks. ASP.NET Core

ISSN No:-2456-2165
https://doi.org/10.38124/ijisrt/25mar1677

provides high-performance cross-platform capabilities that deliver authentication features together with authorisation capabilities through middleware security libraries. IdentityServer4 operates as an open-source framework that uses OAuth 2.0 and OpenID Connect features to create protected token systems which secure authentication between APIs and client applications. OpenID Connect (OIDC) creates an identity system that extends OAuth 2.0 authentication features to enhance user login processes.

The user authentication and authorization processes are handled by the ASP.NET Core Identity framework alongside IdentityServer4. The framework enables users to configure MFA protection and password hash functions as well as external authentication providers. Through the implementation of JSON Web Tokens (JWT) users gain authentication tokens that operate without server state and prevent tampering thus reducing server session management overhead. The implementation of OAuth 2.0 for API endpoints allows applications to access resources as delegated users by avoiding the disclosure of sensitive credentials during the process. The Microsoft Authentication Library (MSAL) brings enterprise-level security through its integrated partnership with Azure Active Directory (Azure AD) which grants SSO capabilities.

C. Threat Model and Security Evaluation

The security assessment of ASP.NET Core applications relies on developing an extensive threat model by exploring potential attack vectors. The STRIDE model classifies security risks through Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege concepts.

Table 6 Threat model and Security Assessment of ASP.NET Core Applications

Threat Category	**Table Column Head**	
	Vulnerability	**Mitigation Strategy**
Spoofing	Unauthorized identity access	Implement OAuth with strong authentication
Tampering	Token forgery, data manipulation	Use JWT with HMAC and RSA encryption
Repudiation	Lack of transaction traceability	Enable logging and audit trails
Information Disclosure	Exposure of sensitive API data	Enforce HTTPS and secure token storage
Denial of Service (DoS)	Excessive API requests, resource exhaustion	Rate limiting, API gateway security
Elevation of Privilege	Bypassing authorization controls	Implement role-based and claims-based access

Implementing a Zero-Trust security model, additional layers of verification are enforced, reducing the attack surface. Continuous authentication, least privilege access, and behavioral anomaly detection further strengthen application security.

D. Data Collection and Testing Scenarios

The authentication and authorization techniques of OAuth and JWT and Zero-Trust approach get tested through real-world examples and penetration evaluation methods. The evaluation process includes:

- Case Study: Securing a Multi-Tenant ASP.NET Core Application.
- The identity management system which enables authentication functions by using IdentityServer4 within a cloud-based software as a service platform.
- An OAuth token system operating through JWT performs stateless authentication to grant users API access.
- The analysis of security logs includes three security monitoring elements: authentication attempts, token validation failures and anomaly detection.
- Penetration Testing Scenarios

➢ *The Simulation Monitored MFA and rate Limiting during Brute Force Attacks on Unauthorized Logins.*

- The researcher's evaluated JWT expiration methods and refresh token procedures through Token Hijacking and Replay Attack Simulation.
- OWASP ZAP and Burp Suite executed API security testing to uncover SQL injection and broken authentication and improper error handling vulnerabilities within the system.

Web applications should avoid using Public OpenID Connect/OAuth clients as authorization solutions because they have lost their functionality recommendation. The default flow exists in this format according to the following diagram:

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

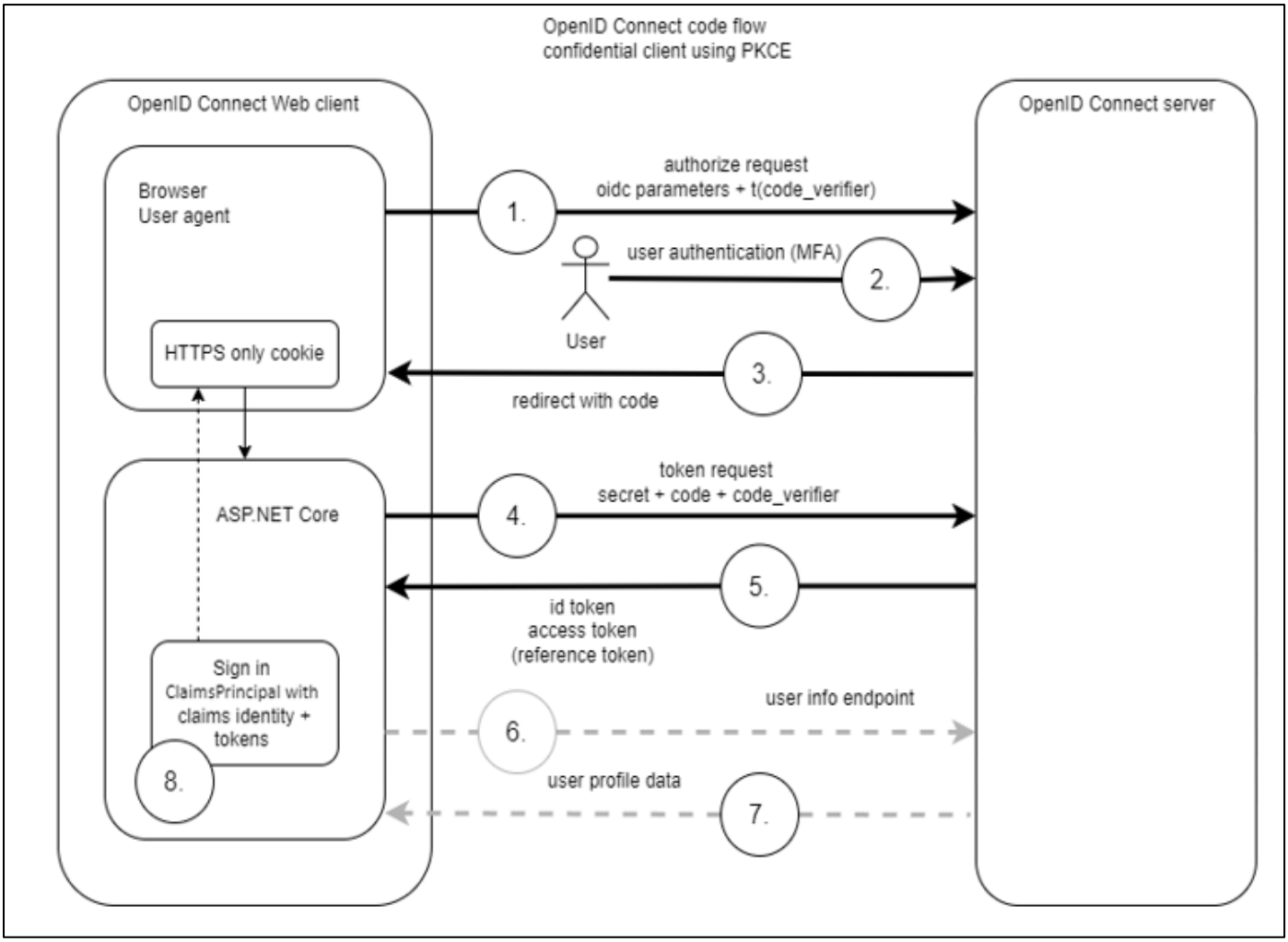

Fig 4 OAuth and OpenID Connect Flow in ASP.NET Core

OpenID Connect exists with multiple versions that lead to differences between server implementations concerning their parameters and requirements. Part of the OpenID Connect variations include servers that lack user info endpoint support together with servers that don't support PKCE and those that need unique request parameters for token acquisition. A client assertion serves as a secure alternative to client secrets. The OpenID Connect Core standard receives supplementary security measures from several new standards such as FAPI, CIBA and DPoP for downstream API interaction.

IV. IMPLEMENTATION AND BEST PRACTICES

A. Implementing OAuth in ASP.NET Core

The deployment of OAuth 2.0 into ASP.NET Core needs developers to know both the OAuth 2.0 authentication sequence and safe methods to combine it with .NET Identity and techniques to defend against token theft and replay attacks.

➢ *Understanding OAuth 2.0 Authentication Flow*

The authorization framework OAuth 2.0 allows external applications to retrieve user information while preventing them from learning authentication passwords. Several stages make up the standard OAuth 2.0 workflow which includes:

- The authorization process begins when the client application moves the user to the authorization server.
- The user authenticates and grants permissions.
- The authorization code issuance to the client occurs after authentication by the authorization server.
- The client requests authorization through the code and receives an access token as a result.
- Using the access token people can access protected resources.

The OAuth flow in ASP.NET Core executes through middleware combination involving `AddAuthentication` and `AddOAuth`. The framework enables connection to well-known providers Google, Facebook and Microsoft through an integrated system.

➢ *Securely Integrating OAuth with .NET Identity*

The configuration of OAuth provider needs to be added in the `Startup.cs` file to integrate OAuth with .NET Identity. The `AddOAuth` method requires parameters for client credentials as well as callback information and scopes selection. The storage of client secrets and other sensitive information should use Azure Key Vault or environment variables for secure protection. External login claims should be mapped to .NET Identity claims for maintaining a single user experience framework.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

The system implements security measures to stop Token Theft and Replay Attack instances.

You can stop token theft by using HTTPS for every communication and by requiring short-lived access tokens alongside refresh tokens. Programming systems must provide protected tokens by using either cookies with encryption or HTTP-only secure cookies. Nonces along with signature validation must be used for attack defense against replay attacks. Security measures through anti-forgery functionality and token validation pipelines become available as built-in features of ASP.NET Core.

Web applications use JWT as their standard authentication and authorization tool because of their compact design while also providing stateless features. Secure application security depends on proper understanding of token structure alongside detailed validation execution alongside secure token storage protocols and vulnerability management practices.

B. Using JWT for Authentication and Authorization

JSON Web Tokens (JWT) have become a cornerstone in modern web application security, facilitating both authentication and authorization processes. Their compact and self-contained nature allows for secure information exchange between parties. However, to harness their full potential securely, it's imperative to understand their structure, implement robust validation techniques, handle token storage and expiration prudently, and safeguard against common vulnerabilities.

➢ *JWT Token Structure and Validation Techniques*

The standardized JavaScript Web Token consists of three distinct parts.

- The token type is shown in the header section as "JWT" while detail is provided about which signing algorithm the token uses such as HMAC SHA256 or RSA.
- A JWT payload contains two elements consisting of claims and additional semantic data that describe an entity normally referred to as the user.
- Verification through signature enables users to check that the content remains unmodified. The encoding process of header and payload leads to signature generation through the selected algorithm.

- *Security checks during validation make a JWT trustworthy.*

✓ The authentication token requires three differentiated sections which should be separated with dots.
✓ A proper signature validation procedure must compare the token signature against its expected value while utilizing either a secure key or public key and proper algorithm. The token verification process checks if it remains unaffected by unauthorized modification. The algorithm verifies claims through three assessments on attributes iss (issuer), aud (audience) and exp (expiration) to establish alignment with expectations and expiration validation.

➢ *Securely Handling Token Storage and Expiration*

JWT security exists mostly in the methods that developers select to store tokens:

- Since attackers can exploit XSS vulnerabilities to access localStorage it becomes a risky storage mechanism for tokens. HTTP-only Cookies act as a security mechanism to store tokens because the cookies block JavaScript access which minimizes potential XSS risks. Implementing this method needs protection against CSRF attacks to stay secure.
- The security risk period becomes shorter when tokens have limited expiration dates. The time range for token expiration should be set at a minute or hour duration. To maintain long-term sessions you should employ refresh tokens which enable getting new access tokens without needing user authentication.

➢ *Protecting Against JWT-Related Vulnerabilities*

JSON Web Tokens (JWTs) remain at risk of security problems when their management is improper.

- Every system that uses JWT authentication should maintain a clear definition of signing algorithms to ensure proper implementation at the server side. Security breaches occur when the token is given the authority to select the algorithm.
- Unterrated tokens without expiration dates keep their validity throughout time which increases the dangers in case of compromise. All tokens need to include an expiration (exp) claim which makes them invalid after a designated period.
- Token storage in client-side accessible localStorage presents a security vulnerability which makes them available to XSS attacks. Secure storage mechanisms should be adopted to minimize this risk factor.

C. Adopting the Zero-Trust Model in ASP.NET Applications

Under the Zero-Trust security model you must verify everything because trust ought to be absent even inside network boundaries. Multiple essential strategies for implementing this model in ASP.NET applications consist of identity-based access control along with multi-factor authentication (MFA) and continuous monitoring and micro-segmentation.

➢ *Implementing Identity-Based Access Control*

The primary security boundary in Zero-Trust corresponds to identity verification. Each user device application needs authentication authorization to reach company resources. ASP.NET applications achieve their verification processes through identity management systems which integrate strict management of user identities and authorizations. ASP.NET Core Identity serves developers to manage user accounts together with roles and claims in a manner which ensures access decisions rest on verified identities rather than trusting users without verification.

➢ *Role of Multi-Factor Authentication (MFA)*

MFA requires users to present multiple verification methods as an added security measure when accessing

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

software programs. For MFA authentication users must provide three verification factors that contain either information they know such as passwords or objects they have like hardware tokens or physical characteristics they are like their fingerprints. The integration of MFA security protocols into ASP.NET applications reduces the probability of unauthorized entry because compromised credentials remain insufficient for access authorization. The integration of Azure Active Directory by developers enables secure MFA implementation to boost application security postures.

➢ *Continuous Monitoring and Micro-Segmentation*

Security personnel monitor activities in real time to detect irregularities because they reveal potential security threats from user actions and network traffic as well as system behavior. Naive security breaches can be prevented through continuous monitoring because logging and monitoring tools detect unpredictable data activity within ASP.NET applications.

Network segmentation becomes vital for security through micro-segmentation which splits the network into isolated sections with their own secure entry controls. Separating the network into segments through micro-segmentation effectively contains security threats to isolated network regions. When executing micro-segmentation for ASP.NET applications developers should establish services and APIs that enforce strict access mechanisms to enable components to communicate through necessary pathways bounded by set security boundaries.

V. COMPARATIVE ANALYSIS

Evaluation of security models including OAuth, JSON Web Tokens (JWT), and Zero-Trust necessitates an analysis of their capabilities to ensure security along with performance effects and implementation simplicity and their practical deployment situations.

➢ *Security Strength Comparison: OAuth vs. JWT vs. Zero-Trust*

- Through OAuth users grant certain permissions to external applications since the framework functions as an authorization method that provides specific resource access while keeping credentials secure. Secure token management remains essential for the protection of access because its access delegation system depends on access tokens.

- JWT serves as a small URL-friendly token structure which finds frequent use in authentication systems and information transmission roles. JWT tokens perform fast verification because they are stateless yet they become vulnerable to attacks when their signing or validation processes are not properly executed.

- Under this security model threats emerge from internal and external sources due to its principle of zero-trust. Zero-Trust implements an effective defense system through identity verification protocols while also granting limited access permissions which it supports with continuous monitoring requirements.

➢ *Performance and Scalability: Impact on Authentication Speed and System Load*

- The secure systems use OAuth and JWT protocols that have been designed for large-scale applications. Renting tokens through OAuth as well as the stateless characteristics of JWT eliminate the requirement to store session data on servers which improves system performance. The cryptographic operations needed to sign and verify tokens result in latency unless these operations receive proper optimization.

- The continuous verification along with monitoring operations within Zero-Trust implementation might lead to additional overhead expenses. Security enhancements can deliver minimized performance-related challenges through proper infrastructure optimization while modern system utilization.

➢ *Ease of Implementation and Maintenance: Complexity and Integration Challenges*

- ASP.NET applications benefit from wide adoption of OAuth and JWT technologies because they have extensive libraries together with developer support. Developers need to establish strong secure configurations when implementing these technologies because their setup remains straightforward.

- The implementation of a Zero-Trust security model demands major changes in underlying security structures at a level that proves complex and resource-intensive for development teams. The security enhancement process requires organizations to redesign networks and enhance access protocols and tighten system surveillance. When evaluating the long-term security benefits against preliminary investments they usually present sufficient value to accept the initial costs.

➢ *Real-World Case Studies: Examples of Companies Using These Models for Security*

- Several major organizations like social media companies together with cloud service providers depend on OAuth and JWT to enable protected authorization features for authentication needs. The authentication service OAuth 2.0 enables Google and Microsoft to let third parties connect while protecting user authentication credentials.
- The company Google developed BeyondCorp as an internal Zero-Trust architecture which gave them internal network defense against advanced threats by treating all communications as untrusted.

VI. CHALLENGES AND FUTURE DIRECTIONS

Organizations that want to improve ASP.NET application security face several challenges which require them to forecast forthcoming security threats to maintain a strategic lead.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1677

A. Challenges in Implementing OAuth, JWT, and Zero-Trust

➢ *Key Adoption Barriers for Developers and Enterprises:*

Administering these security measures needs developers to deeply understand their core ideas together with their possible issues. Programmers often encounter difficulties securing OAuth and JWT implementation when they need to validate and expire token correctly. Giving organizations adopting the Zero-Trust model must deal with cultural transformations since it needs both extensive employee training and changes in workplace behaviors for successful adoption.

B. Future Security Trends in .NET Applications

➢ *AI-Driven Security Enhancements:*

Security research will experience revolution through Artificial Intelligence (AI) which enables both automated responses and predictive threat detection. The implementation of AI within ASP.NET applications enables better preventative security methods that identify threats during their development phase.

➢ *Blockchain for Identity Verification:*

Identity verification processes managed through blockchain technology operate as decentralized systems which maintain tamper-proof operations. ASP.NET applications that incorporate blockchain achieve better identity management security by fostering trust while eliminating the requirement for central authority control and eliminating single points of failure.

➢ *Quantum-Resistant Encryption for ASP.NET Applications:*

The advancement of quantum computing technology makes traditional encryption methods vulnerable to becoming obsolete. The protection of data through quantum-based attacks requires developers to deploy quantum-resistant encryption algorithms throughout ASP.NET application frameworks.

VII. CONCLUSION

Security models used traditionally do not effectively handle present-day security threats in ASP.NET Core applications. The implementation of OAuth and JWT along with the Zero-Trust model proved to be highly effective methods for application security improvement. The authorization security of OAuth works through unobstructed third-party access pathways while JWT maintains an efficient stateless process for token-based authentication across different application parties. By adopting the Zero-Trust model developers must verify all user actions continuously while denying trust at all times to reduce exposure vulnerabilities.

Security enhancement is essential for ASP.NET developers who need these security frameworks in their development practice. The implementation of OAuth together with JWT decreases vulnerabilities regarding credential theft and session hijacking attacks. Developers should ensure proper configuration of JWT bearer authentication in ASP.NET Core to validate tokens effectively. Implementing the Zero-Trust model requires a shift in mindset towards continuous verification and least-privilege access, which can be achieved by leveraging ASP.NET Core's built-in security features. Additionally, employing multi-factor authentication (MFA) and AI-driven anomaly detection can further strengthen security measures.

Research in the future must critically inspect modern security methods that utilize Zero-Trust architecture while incorporating AI threat detection mechanisms to improve ASP.NET Core application defense. Analysis of machine learning systems in real-time threat monitoring has valuable applications for security prevention and response. The analysis of Zero-Trust implementation barriers alongside successful strategies applied across different organizational types would produce greater comprehension of its practical deployment.

ASP.NET developers need to implement novel security frameworks including OAuth and JWT and Zero-Trust models because cyber security threats will persist in their existing form. Developers who both embrace these security methods and track new security trends will achieve maximum protection of their applications despite growing digital complexity.

REFERENCES

[1]. Badhwar, R. (2021). Intro to API Security-Issues and Some Solutions!. In *The CISO's Next Frontier: AI, Post-Quantum Cryptography and Advanced Security Paradigms* (pp. 239-244). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-75354-2_29

[2]. Barabanov, A., & Makrushin, D. (2020). Authentication and authorization in microservice-based systems: survey of architecture patterns. arXiv preprint arXiv:2009.02114. https://doi.org/10.48550/arXiv.2009.02114

[3]. Basta, N., Ikram, M., Kaafar, M. A., & Walker, A. (2022, April). Towards a zero-trust micro-segmentation network security strategy: an evaluation framework. In NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium (pp. 1-7). IEEE. https://doi.org/10.1109/NOMS54207.2022.9789888

[4]. Bhawiyuga, A., Data, M., & Warda, A. (2017, October). Architectural design of token based authentication of MQTT protocol in constrained IoT device. In *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)* (pp. 1-4). IEEE. https://doi.org/10.1109/TSSA.2017.8272933

[5]. Bojinov, H., & Boneh, D. (2011, March). Mobile token-based authentication on a budget. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications* (pp. 14-19). https://doi.org/10.1145/2184489.2184494

[6]. Cirani, S., Picone, M., Gonizzi, P., Veltri, L., & Ferrari, G. (2014). Iot-oas: An oauth-based authorization service architecture for secure services in iot

scenarios. *IEEE sensors journal*, *15*(2), 1224-1234. https://doi.org/10.1109/JSEN.2014.2361406

[7]. Cirani, S., Picone, M., Gonizzi, P., Veltri, L., & Ferrari, G. (2014). Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios. *IEEE sensors journal*, *15*(2), 1224-1234. https://doi.org/10.1145/2976749.2978385

[8]. Dash, B. (2024). Zero-Trust Architecture (ZTA): Designing an AI-Powered Cloud Security Framework for LLMs' Black Box Problems. Available at SSRN 4726625. https://dx.doi.org/10.2139/ssrn.4726625

[9]. de Almeida, M. G., & Canedo, E. D. (2022). Authentication and authorization in microservices architecture: A systematic literature review. *Applied Sciences*, *12*(6), 3023. https://doi.org/10.3390/app12063023

[10]. Emerson, S., Choi, Y. K., Hwang, D. Y., Kim, K. S., & Kim, K. H. (2015, October). An OAuth based authentication mechanism for IoT networks. In *2015 International Conference on Information and Communication Technology Convergence (ICTC)* (pp. 1072-1074). IEEE. https://doi.org/10.1109/ICTC.2015.7354740

[11]. Ferry, E., O Raw, J., & Curran, K. (2015). Security evaluation of the OAuth 2.0 framework. *Information & Computer Security*, *23*(1), 73-101.

[12]. Greitzer, F. L., Moore, A. P., Cappelli, D. M., Andrews, D. H., Carroll, L. A., & Hull, T. D. (2008). Combating the insider cyber threat. *IEEE Security & Privacy*, *6*(1), 61-64.https://doi.org/10.1109/MSP.2008.8

[13]. Haekal, M. (2016, October). Token-based authentication using JSON web token on SIKASIR RESTful web service. In *2016 International Conference on Informatics and Computing (ICIC)* (pp. 175-179). IEEE. https://doi.org/10.1109/IAC.2016.7905711

[14]. He, Y., Huang, D., Chen, L., Ni, Y., & Ma, X. (2022). A survey on zero trust architecture: Challenges and future trends. Wireless Communications and Mobile Computing, 2022(1), 6476274. https://doi.org/10.1155/2022/6476274

[15]. Heiland, R., Koranda, S., Marru, S., Pierce, M., & Welch, V. (2015, June). Authentication and authorization considerations for a multi-tenant service. In *Proceedings of the 1st Workshop on The Science of Cyberinfrastructure: Research, Experience, Applications and Models* (pp. 29-35). https://doi.org/10.1145/2753524.2753534

[16]. Herceg, T. (2024). *Modernizing. NET Web Applications: Everything You Need to Know About Migrating ASP. NET Web Applications to the Latest Version of. NET*. Springer Nature. https://doi.org/10.1177/1550147717712627

[17]. Indu, I., PM, R. A., & Bhaskar, V. (2017). Encrypted token based authentication with adapted SAML technology for cloud web services. *Journal of Network and Computer Applications*, *99*, 131-145. https://doi.org/10.1016/j.jnca.2017.10.001

[18]. Jones, M., Bradley, J., & Sakimura, N. (2015). Rfc 7519: Json web token (jwt). https://doi.org/10.17487/RFC7519

[19]. Jung, S. W., & Jung, S. (2017). Personal OAuth authorization server and push OAuth for Internet of Things. *International Journal of Distributed Sensor Networks*, *13*(6), 1550147717712627. https://doi.org/10.1177/1550147717712627

[20]. Kang, H., Liu, G., Wang, Q., Meng, L., & Liu, J. (2023). Theory and application of zero trust security: A brief survey. Entropy, 25(12), 1595. https://doi.org/10.3390/e25121595

[21]. Kauser, S., Rahman, A., Khan, A. M., & Ahmad, T. (2019). Attribute-based access control in web applications. In Applications of Artificial Intelligence Techniques in Engineering: SIGMA 2018, Volume 1 (pp. 385-393). Springer Singapore. https://doi.org/10.1007/978-981-13-1819-1_36

[22]. Kavitha, D., & Thejas, S. (2024). Ai enabled threat detection: Leveraging artificial intelligence for advanced security and cyber threat mitigation. *IEEE Access*.https://doi.org/10.1109/ACCESS.2024.3493957

[23]. Kendyala, S. H. (2020). THE ROLE OF MULTI FACTOR AUTHENTICATION IN SECURING CLOUD BASED ENTERPRISE APPLICATIONS. Available at SSRN 5074876 . https://dx.doi.org/10.2139/ssrn.5074876

[24]. Leiba, B. (2012). Oauth web authorization protocol. *IEEE Internet Computing*, *16*(1), 74-77. https://doi.org/10.1109/MIC.2012.11

[25]. Machireddy, J. R. (2024). Machine Learning and Automation in Healthcare Claims Processing. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *6*(1), 686-701. https://doi.org/10.60087/jaigs.v6i1.335

[26]. Matta, V., Di Mauro, M., Longo, M., & Farina, A. (2018). Cyber-threat mitigation exploiting the birth–death–immigration model. *IEEE Transactions on Information Forensics and Security*, *13*(12), 3137-3152.https://doi.org/10.1109/TIFS.2018.2838084

[27]. Munonye, K., & Péter, M. (2022). Machine learning approach to vulnerability detection in OAuth 2.0 authentication and authorization flow. *International Journal of Information Security*, *21*(2), 223-237. https://doi.org/10.23919/EECSI48112.2019.8977061

[28]. Machireddy, Jeshwanth, Harnessing AI and Data Analytics for Smarter Healthcare Solutions (January 14, 2023). International Journal of Science and Research Archive, 2023, 08(02), 785-798 , Available at SSRN: http://dx.doi.org/10.2139/ssrn.5159750

[29]. Norberg, S. (2020). *Advanced ASP .NET Core 3 Security*. Apress. https://doi.org/10.1007/978-1-4842-6014-2

[30]. Norberg, S., & Norberg, S. (2020). Introducing ASP. NET Core. *Advanced ASP. NET Core 3 Security: Understanding Hacks, Attacks, and Vulnerabilities to Secure Your Website*, 1-29. https://doi.org/10.1007/978-1-4842-6014-2_1

[31]. Oh, S. R., & Kim, Y. G. (2020). AFaaS: Authorization framework as a service for Internet of Things based on interoperable OAuth. *International Journal of Distributed Sensor Networks*, *16*(2),

1550147720906388. https://doi.org/10.1177/1550147720906388

[32]. Pai, S., Sharma, Y., Kumar, S., Pai, R. M., & Singh, S. (2011, June). Formal verification of OAuth 2.0 using Alloy framework. In *2011 International Conference on Communication Systems and Network Technologies* (pp. 655-659). IEEE. https://doi.org/10.1109/CSNT.2011.141

[33]. Polo, L. (2024). Revolutionizing sales and operations planning with artificial intelligence: Insights and results. International Journal For Multidisciplinary Research, 6(6). https://doi.org/10.36948/ijfmr.2024.v06i06.34053

[34]. Paul, B., & Rao, M. (2022). Zero-trust model for smart manufacturing industry. Applied Sciences, 13(1), 221. https://doi.org/10.3390/app13010221

[35]. Poudel, B. P., Mustafa, A., Bidram, A., & Modares, H. (2020). Detection and mitigation of cyber-threats in the DC microgrid distributed control system. *International Journal of Electrical Power & Energy Systems*, *120*, 105968.https://doi.org/10.1111/risa.13900

[36]. Poudel, B. P., Mustafa, A., Bidram, A., & Modares, H. (2020). Detection and mitigation of cyber-threats in the DC microgrid distributed control system. *International Journal of Electrical Power & Energy Systems*, *120*, 105968.https://doi.org/10.1016/j.ijepes.2020.105968

[37]. Qazi, F. A. (2022, December). Study of zero trust architecture for applications and network security. In 2022 IEEE 19th international conference on smart communities: improving quality of life using ICT, IoT and AI (HONET) (pp. 111-116). IEEE. https://doi:10.1109/HONET56683.2022.10019186.

[38]. Rozaliuk, T., Kopyl, P., & Smołka, J. (2022). Comparison of ASP. NET Core and Spring Boot ecosystems. *Journal of Computer Sciences Institute*, *22*, 40-45. https://doi.org/10.35784/jcsi.2794

[39]. Sadqi, Y., Belfaik, Y., & Safi, S. (2020, March). Web oauth-based SSO systems security. In *Proceedings of the 3rd International Conference on Networking, Information Systems & Security* (pp. 1-7). https://doi.org/10.1145/3386723.3387888

[40]. Sendor, J., Lehmann, Y., Serme, G., & de Oliveira, A. S. (2014, March). Platform-level support for authorization in cloud services with OAuth 2. In *2014 IEEE International Conference on Cloud Engineering* (pp. 458-465). IEEE. https://doi.org/10.1109/IC2E.2014.60

[41]. Sharif, A., Carbone, R., Scia rretta, G., & Ranise, S. (2022). Best current practices for OAuth/OIDC Native Apps: A study of their adoption in popular providers and top-ranked Android clients. *Journal of Information Security and Applications*, *65*, 103097. https://doi.org/10.1016/j.jisa.2021.103097

[42]. Sheffer, Y., Hardt, D., & Jones, M. (2020). RFC 8725: JSON web token best current practices. https://doi.org/10.17487/RFC8725

[43]. Shibli, M. A., Masood, R., Habiba, U., Kanwal, A., Ghazi, Y., & Mumtaz, R. (2014). Access control as a service in cloud: challenges, impact and strategies. Continued Rise of the Cloud: Advances and Trends in Cloud Computing, 55-99. https://doi.org/10.1007/978-1-4471-6452-4_3

[44]. Singh, J., & Chaudhary, N. K. (2022). OAuth 2.0: Architectural design augmentation for mitigation of common security vulnerabilities. *Journal of Information Security and Applications*, *65*, 103091. https://doi.org/10.1016/j.jisa.2021.103091

[45]. Solapurkar, P. (2016, December). Building secure healthcare services using OAuth 2.0 and JSON web token in IOT cloud scenario. In *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)* (pp. 99-104). IEEE. https://doi.org/10.1109/IC3I.2016.7917942

[46]. Sudarsan, S. V., Schelén, O., & Bodin, U. (2023). Multilevel subgranting by power of attorney and oauth authorization server in cyber–physical systems. *IEEE internet of things journal*, *10*(17), 15266-15282. https://doi.org/10.1109/JIOT.2023.3265407

[47]. Syed, N. F., Shah, S. W., Shaghaghi, A., Anwar, A., Baig, Z., & Doss, R. (2022). Zero trust architecture (zta): A comprehensive survey. IEEE access, 10, 57143-57179. https://doi:10.1109/ACCESS.2022.3174679.

[48]. Tanvi, P., Sonal, G., & Kumar, S. M. (2011, June). Token based authentication using mobile phone. In *2011 International Conference on Communication Systems and Network Technologies* (pp. 85-88). IEEE. https://doi.org/10.1109/CSNT.2011.24

[49]. Tassanaviboon, A., & Gong, G. (2011, November). Oauth and abe based authorization in semi-trusted cloud computing: aauth. In *Proceedings of the second international workshop on Data intensive computing in the clouds* (pp. 41-50). https://doi.org/10.1145/2087522.2087531

[50]. Wang, C. (2022, December). Design and Implementation of Ideological and Political Education Network Platform for College Students under ASP. NET. In *2022 3rd International Conference on Artificial Intelligence and Education (IC-ICAIE 2022)* (pp. 923-930). Atlantis Press. https://doi.org/10.2991/978-94-6463-040-4_139

[51]. Wang, Y., & Huang, Y. (2018). Research on education and teaching resources management system based on ASP. NET. In *Lecture Notes in Real-Time Intelligent Systems* (pp. 425-431). Springer International Publishing. https://doi.org/10.1007/978-3-319-60744-3_46

[52]. Whitesell, S., Richardson, R., Groves, M. D., Whitesell, S., Richardson, R., & Groves, M. D. (2022). ASP. NET Core Overview. *Pro Microservices in. NET 6: With Examples Using ASP. NET Core 6, MassTransit, and Kubernetes*, 29-49. https://doi.org/10.1007/978-1-4842-7833-8_2