Sarcouncil Journal of Engineering and Computer Sciences

ISSN(Online): 2945-3585

Volume- 04| Issue- 06| 2025

Research Article **Received:** 18-05-2025 | **Accepted:** 05-06-2025 | **Published:** 25-06-2025

OAuth 2.0 and OpenID Connect Integration with Azure Entra ID

Bhaskardeep Khaund
Microsoft, USA

Abstract: This article examines the implementation of OAuth 2.0 and OpenID Connect within Azure Entra ID, exploring both the theoretical foundations of these protocols and their practical application in enterprise environments. Beginning with an analysis of the core OAuth 2.0 components and their security model, the discussion expands to cover how OpenID Connect extends these capabilities with standardized identity assertions through JWT tokens. The article provides a detailed examination of Azure Entra ID's architectural implementation of these protocols, including its multi-tenant design and integration with the broader Microsoft ecosystem. Particular attention is given to the three primary authorization flows—Authorization Code, Client Credentials, and Implicit—with analysis of their security characteristics and appropriate use cases. The security section addresses threat modeling, vulnerability mitigation strategies, and token lifecycle management specific to Azure implementations. The article demonstrates how these protocols address diverse identity challenges across organizational boundaries. The article concludes with an exploration of emerging protocol evolutions and Microsoft's strategic direction toward decentralized identity and Zero Trust architecture integration, providing a comprehensive overview of current best practices and future developments in this critical domain of identity and access management.

Keywords: OAuth 2.0, OpenID Connect, Azure Entra ID, Token Security, Zero Trust Architecture.

INTRODUCTION

In today's interconnected digital landscape, secure authorization and authentication mechanisms have become fundamental requirements for enterprise applications and cloud services. The proliferation of distributed systems, microservices architectures, and cross-domain integrations has intensified the need for standardized, robust identity protocols that can operate seamlessly across organizational boundaries. OAuth 2.0 has emerged as the de facto standard for delegated authorization, providing a framework that allows applications to access resources on behalf of users without exposing sensitive credentials [Microsoft, 2025]. When combined with OpenID Connect as an identity layer, these protocols form a comprehensive solution for modern identity and access management challenges.

Azure Entra ID, Microsoft's cloud-based identity and access management service, implements these protocols to enable secure authentication and authorization across Microsoft's cloud ecosystem and beyond. As organizations increasingly migrate to cloud-based infrastructure and adopt hybrid identity models, understanding the intricacies of OAuth 2.0 and OpenID Connect within the Azure context has become essential for security architects, developers, and system administrators alike.

This article examines the theoretical underpinnings of OAuth 2.0 and OpenID Connect, their specific implementation within Azure Entra ID, and the security considerations that must be addressed when deploying these protocols in enterprise environments. By exploring the various authorization flows supported by Azure Entra ID—including the authorization code flow, client credentials flow, and implicit flow—we provide a comprehensive analysis of how these protocols operate in practical scenarios. Additionally, we assess the security implications of different implementation choices and offer guidance on best practices for token management, scope definition, and threat mitigation.

The integration of OAuth 2.0 and OpenID Connect with Azure Entra ID represents a critical intersection of protocol standards and cloud infrastructure that fundamentally shapes how modern applications handle identity and access control. Through this examination, we aim to contribute to the broader understanding of secure identity management in cloud-native and hybrid architectures.

II. THEORETICAL FRAMEWORK

A. OAuth 2.0 Fundamentals

OAuth 2.0 emerged as a response to the growing need for secure delegation of access in web and mobile applications. The protocol was designed around several key principles: separation of concerns, simplicity for client implementation, and secure token handling. Unlike its predecessor, OAuth 2.0 eliminated the complex signature requirements, instead relying on TLS for transport security [Hardt, Ed, D, 2012].

The protocol defines three primary components that interact within its framework. The authorization server authenticates resource owners

***Corresponding Author:** Bhaskardeep Khaund

DOI- https://doi.org/ 10.5281/zenodo.15732825

and issues access tokens to clients. The resource server hosts protected resources and validates tokens to authorize access requests. Clients request access tokens and use them to access protected resources on behalf of the resource owner. This separation creates a security boundary that prevents credential sharing across domains.

At the core of OAuth 2.0 lies its token-based security model. Access tokens serve as delegation artifacts that represent granted permissions with limited scope and lifetime. These tokens, typically opaque to clients, contain authorization information that resource servers can validate without contacting the authorization server for every request, enabling scalable authorization architectures. The model supports both reference tokens (requiring validation lookups) and self-contained tokens that carry all necessary authorization information.

B. OpenID Connect Extension

OpenID Connect (OIDC) extends OAuth 2.0 by adding a standardized identity layer, addressing the authentication gap in the base protocol. OIDC defines how identity information should be exchanged through an additional token type—the ID token—formatted as a JSON Web Token (JWT) [OpenID Foundation, 2014]. These tokens contain verified claims about the authenticated user, including attributes such as name, email, and other profile information.

JWTs in OIDC are digitally signed, allowing verification of their authenticity and integrity without requiring back-channel communication. The structured claims within these tokens follow standardized schemas, ensuring interoperability across identity providers and relying parties. OIDC's token structure includes three components: a header identifying the token type and signing algorithm, a payload containing identity claims, and a signature that verifies token authenticity.

A critical distinction in understanding these protocols is the difference between authentication and authorization. Authentication verifies identity (who the user is), which OIDC handles through ID tokens, while authorization determines access rights (what the user can do), managed by OAuth 2.0's access tokens. This separation allows applications to implement both concerns with appropriate security controls tailored to each purpose.

III. AZURE ENTRA ID IMPLEMENTATION

A. Architectural Overview

Azure Entra ID functions as Microsoft's cloud-based identity provider, serving millions of organizations globally. Azure Entra ID implements both OAuth 2.0 and OIDC while extending these protocols with additional capabilities suited for enterprise scenarios. Its architecture supports global distribution, high availability, and secure token issuance at scale.

AAD's (Azure Entra ID) multi-tenant design allows organizations to maintain isolated identity environments while enabling cross-tenant collaboration when needed. Each tenant represents a dedicated instance of Azure Entra ID with its own directory of users, groups, and applications. This architecture supports various B2B scenarios, including partner access to resources without requiring redundant identity management.

Within the Microsoft ecosystem, Azure Entra ID serves as the identity foundation for services ranging from Microsoft 365 to Azure resources. This integration enables single sign-on across the Microsoft cloud and extends to third-party applications through standardized protocol support. Azure Entra ID connects with on-premises Active Directory through Azure AD Connect, supporting hybrid identity scenarios common in enterprise environments [Microsoft, 2024].

B. Protocol Support

Azure Entra ID exposes OAuth 2.0 endpoints that conform to the specification while adding Microsoft-specific extensions. The authorization endpoint handles user authentication and consent, while the token endpoint issues access, ID, and refresh tokens. These endpoints support all major OAuth 2.0 flows, with specific considerations for web, mobile, and daemon applications.

The OpenID Connect implementation in Azure Entra ID follows the standard specifications while adding enterprise-focused capabilities. Azure Entra ID issues signed JWTs with claims about authenticated users, including organizational context and group memberships. The OpenID configuration endpoint provides discovery information that enables dynamic client registration and endpoint determination.

The Microsoft identity platform has evolved significantly from its original v1.0 endpoint to the current v2.0 implementation. This evolution

Publisher: SARC Publisher

expanded support for personal Microsoft accounts alongside organizational accounts, introduced application-only permissions, and enhanced the consent framework. Recent developments include support for continuous access evaluation, conditional access integration, and enhanced token security features that extend beyond base protocol requirements.

IV. AUTHORIZATION FLOWS IN AZURE CONTEXT

A. Authorization Code Flow

The Authorization Code Flow represents the most comprehensive and secure OAuth 2.0 flow implemented in Azure Entra ID. This flow consists of a two-stage token acquisition process that separates the user authentication/authorization phase from the token retrieval phase. In the Azure implementation, the flow begins with an application redirecting the user to the Azure Entra ID authorization endpoint (/authorize), where authentication occurs and consent is obtained. The resulting authorization code is then exchanged for tokens through a server-side request to the token endpoint (/token) [Microsoft, 2025].

Table 1: OAuth 2.0 Authorization Flows in Azure Entra ID [Microsoft, 2025]

Flow Type	Primary Use Case	Security Level	Key Implementation Considerations
Authorization Code	Web applications with a server-side component	High	Requires PKCE, secure client secret storage, and strict redirect URI validation
Client Credentials	Service-to-service communication	Medium-High	No user interaction, requires secure credential management, or managed identities
Implicit Flow	Legacy single-page applications	Low	Not recommended for new implementations, vulnerable to token interception

Microsoft's implementation includes specific security enhancements beyond the base OAuth 2.0 specification. The mandatory use of PKCE (Proof Key for Code Exchange) prevents authorization code interception attacks by binding the code to the requesting client. Azure Entra ID also supports restricted redirect URIs, requiring pre-registration of valid redirect destinations to prevent token theft through URI manipulation. Additionally, the platform enforces TLS requirements and implements rate limiting to mitigate brute force attacks against authorization codes.

This flow is particularly well-suited for web applications where server-side code can securely store client secrets and handle token exchanges. Enterprise applications with sensitive data requirements typically implement this flow to benefit from its enhanced security model. Microsoft recommends this flow for most scenarios, including multi-tier applications, native applications with web-based authentication, and backend API access patterns requiring delegated permissions.

B. Client Credentials Flow

The Client Credentials Flow in Azure Entra ID enables service-to-service authentication without user involvement. This flow allows applications to obtain tokens directly from Azure Entra ID using their own identity rather than acting on behalf of a user. Implementation involves a direct POST request to the Azure Entra ID token endpoint with client credentials and requested scopes, resulting in an access token that represents the application's identity and permissions.

Azure's Managed Identities feature extends this flow by eliminating the need for stored credentials in application code or configuration. When enabled, Azure services automatically receive an identity within Azure Entra ID that can request tokens through the Instance Metadata Service, avoiding credential management challenges. This approach significantly reduces security risks associated with credential rotation and storage while simplifying access to Azure resources.

Security boundaries in this flow are enforced through application permissions rather than delegated permissions. These permissions typically require administrative consent and grant access at the application level rather than on behalf of specific users. Azure Entra ID implements additional security controls for this flow, including client credential validation, IP-based restrictions, and detailed audit logging of application authentication events to detect potential misuse or compromise.

Publisher: SARC Publisher

C. Implicit Flow

The Implicit Flow represents an earlier OAuth 2.0 design pattern optimized for browser-based applications that could not securely store client secrets. This flow returns tokens directly in the browser redirect URI fragment without requiring an intermediary code exchange step. While historically common for single-page applications (SPAs), Microsoft now discourages this flow due to its inherent security limitations [Microsoft, 2025].

In the Azure context, legacy single-page applications built before modern alternatives might still implement this flow. The implementation involves redirecting users to the Azure Entra ID authorization endpoint with response_type=token or response_type=id_token token parameters. Upon successful authentication, tokens are returned directly to the browser, where client-side JavaScript can extract and use them from the URI fragment.

Security limitations in this flow are significant and well-documented. Tokens are exposed to browser history, may be accessible to malicious third-party scripts, and cannot be refreshed without prompting users for re-authentication. Additionally, PKCE cannot be implemented with this flow, leaving applications vulnerable to certain cross-site request forgery attacks. Microsoft's current best practice guidance recommends migrating SPA applications to the Authorization Code Flow with PKCE, which modern browsers can now implement securely without requiring client secrets.

V. SECURITY ANALYSIS

A. Threat Modeling

OAuth 2.0 and OpenID Connect implementations face several common vulnerabilities that must be addressed through comprehensive threat modeling. Token leakage represents one of the most significant risks, occurring through cross-site scripting (XSS) attacks, insecure storage, or compromised communication channels. Cross-site request forgery (CSRF) attacks may allow attackers to initiate unauthorized OAuth flows, particularly in implementations lacking proper state validation. Additionally, phishing attacks targeting authorization endpoints can redirect users to malicious sites designed to harvest credentials or authorization grants [Lodderstedt, Ed, T, 2013].

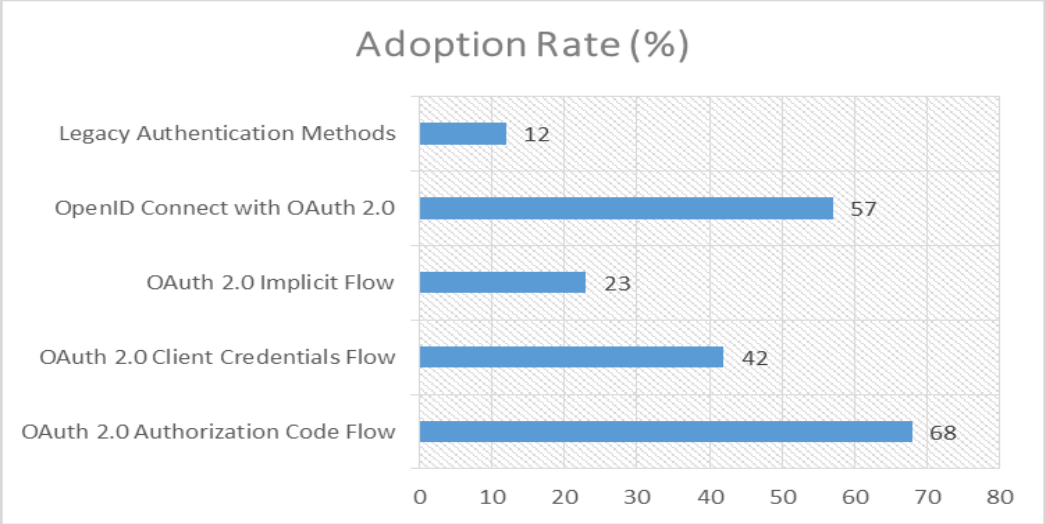

Fig 1: Authorization Protocol Adoption in Enterprise Applications [Lodderstedt, Ed, T, 2013; Microsoft]

Effective mitigation strategies begin with proper protocol implementation following security best practices. For XSS protection, applications should store tokens in secure HTTP-only cookies or trusted browser storage mechanisms with appropriate content security policies. CSRF attacks can be mitigated through the use of state parameters containing unpredictable values tied to the user's session. Redirect URI validation must strictly enforce pre-registered endpoints, and applications should implement incremental authorization, requesting only necessary permissions at each stage.

Azure Active Directory provides several platform-specific security features that enhance protocol security. Conditional Access policies allow organizations to enforce risk-based authentication requirements based on user, device, location, and application context. Identity Protection uses machine learning to detect suspicious sign-in attempts and compromised credentials. For token

Publisher: SARC Publisher

security, Azure Entra ID supports features like continuous access evaluation, which enables near real-time token revocation based on security signals, and resource-constrained tokens that limit the scope of potential damage from compromised credentials.

B. Token Management

The access token lifecycle in Azure Entra ID implementations typically follows a structured pattern of acquisition, usage, and renewal. Access tokens have limited lifetimes, generally between 60-90 minutes for user tokens and up to 24 hours for application tokens. This temporal restriction limits the window of opportunity for token misuse if compromised. Applications must implement proper token caching strategies to reduce authentication prompts while respecting token expiration policies [https://auth0.com].

Table 2: Token Types and Security Characteristics in Azure Entra ID [Lodderstedt, Ed, T, 2013; https://auth0.com]

Token Type	Typical Lifetime	Audience	Security Considerations
ID Token	Short (1 hour)	Client application	Contains user identity claims, must validate signature, nonce, and issuer
Access Token	Medium (1-2 hours)	Protected resource	Resource-specific, requires audience validation, supports continuous access evaluation
Refresh Token	Long (14-90 days)	Authorization server	High-value target, subject to conditional lifetime policies and family restrictions

Refresh token security presents unique challenges as these long-lived tokens enable continued access without user intervention. Azure Entra ID implements several security controls for refresh tokens, including family-based token binding that limits refresh token use to specific application families, token lifetime policies that adjust based on risk signals, and conditional revocation triggered by suspicious activities. Organizations can further configure token lifetimes through custom policies that balance security and usability requirements.

Token validation best practices encompass both client and resource server responsibilities. Resource servers must validate token signatures using public keys retrieved from Azure Entra ID OpenID configuration endpoints, verify appropriate audience claims match the intended resource, check that tokens have not expired, and confirm that requested actions align with the scopes granted in the token. Clients should verify ID token nonces to prevent replay attacks and validate issuer claims to ensure tokens originate from trusted authorities. Additionally, implementing token revocation capabilities for scenarios such as user logouts and permission changes represents an essential security control.

VI. CASE STUDIES

A. Enterprise Application Integration

Business-to-business (B2B) scenarios have benefited significantly from Azure Entra ID implementation of OAuth 2.0 and OpenID Connect. A notable example is a large financial services corporation that leveraged Azure AD B2B to provide secure partner access to internal applications without creating duplicate identities. The solution implemented the authorization code flow with administrative consent policies to control partner access permissions. This approach reduced identity management overhead by 70% while improving security through centralized policy enforcement and comprehensive audit logging [Microsoft].

Workforce identity management presents complex requirements for cross-organizational collaboration. A multinational manufacturing company implemented Azure Entra ID multi-tenant capabilities to unify identity management across subsidiaries while maintaining appropriate security boundaries. Their architecture utilized a combination of authorization flows: authorization code flow for web applications, client credentials with managed identities for backend services, and device code flow for shared industrial terminals. This implementation enabled secure cross-subsidiary resource sharing while enforcing industry-specific compliance requirements through conditional access policies.

B. Consumer Applications

Azure AD B2C implementations demonstrate how OAuth 2.0 and OpenID Connect can be extended to consumer-facing scenarios. A retail organization rebuilt its customer identity system using Azure AD B2C to support millions of consumer identities. Their implementation customized the authorization code flow with branded user

Publisher: SARC Publisher

experiences and progressive profiling to collect customer information incrementally. This approach resulted in a 35% increase in account creation completion rates compared to their previous identity solution, while simultaneously strengthening security through features like risk-based authentication and bot detection.

Social identity provider integration has become a standard requirement for many consumer applications. A media streaming service implemented Azure Entra ID B2C with federated authentication to popular social providers, including Microsoft, Google, and Facebook. This implementation used the implicit flow for their legacy JavaScript client before migrating to the authorization code flow with PKCE for their updated application. The federation capabilities allowed users to authenticate with existing social accounts while the service maintained a consistent security model for authorization decisions regardless of identity source. This approach simplified the user experience while still enabling granular permission management for content access based on subscription levels.

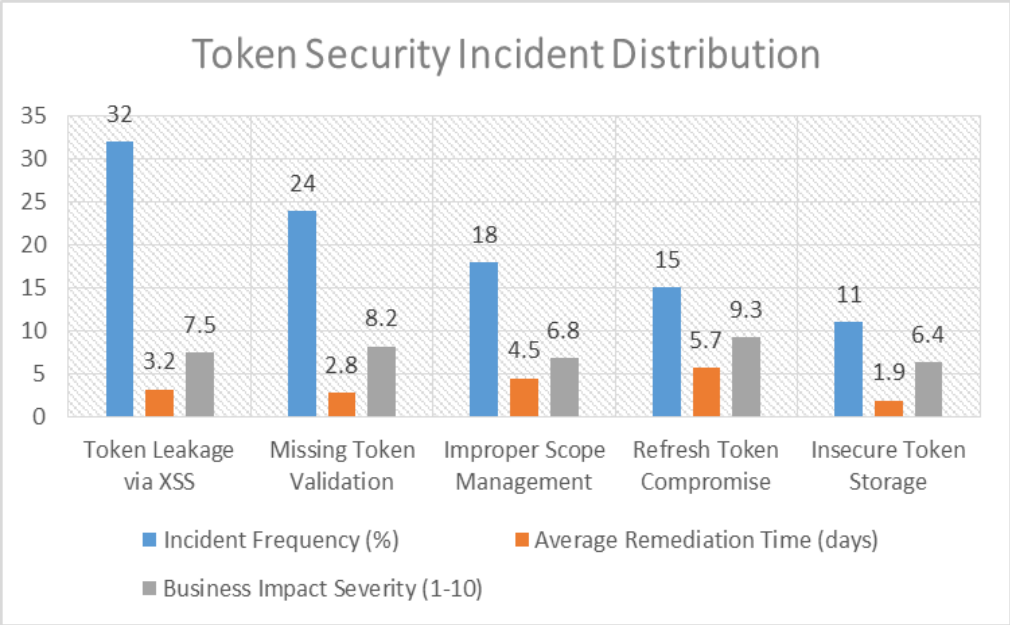

Fig 2: Token Security Incident Distribution by Type [Lodderstedt, Ed, T, 2013; Microsoft]

VII. FUTURE DIRECTIONS

A. Protocol Evolutions

OAuth 2.1 represents an important consolidation effort that aims to simplify the OAuth landscape by incorporating security best practices established since OAuth 2.0's publication. Rather than introducing fundamentally new concepts, OAuth 2.1 standardizes previously recommended security practices into the core specification. Key changes include mandatory PKCE implementation for all authorization code flows, elimination of the implicit flow as a standalone grant type, mandated strict redirect URI matching, and prohibition of the resource owner password credentials grant for new implementations. These changes align with the security practices already recommended by Azure Entra ID, suggesting a relatively straightforward transition path for compliant implementations.

Beyond OAuth 2.1, several emerging authentication standards show promise for addressing evolving security requirements. The Authorization Server Issuer Identifier Response Parameter (ASIIRP) enhancement improves cross-domain security by allowing clients to verify the origin of authorization responses. Device Flow optimizations continue to evolve for IoT and limited-input device scenarios. Rich Authorization Requests (RAR) extend OAuth's capabilities for complex authorization decisions by enabling detailed permission requests with structured data rather than simple string-based scopes. Additionally, the FAPI (Financial-grade API) security profile has gained traction beyond financial services as a framework for securing high-value APIs across multiple sectors.

B. Azure Roadmap

Microsoft has made significant investments in decentralized identity initiatives that extend beyond traditional federation models. The Azure Entra ID Verifiable Credentials service enables organizations to issue cryptographically verifiable digital credentials that users control through

Publisher: SARC Publisher

personal digital wallets. This approach shifts from centralized identity providers toward a self-sovereign identity model where users maintain ownership of their identity information and selectively disclose claims to verifiers without requiring ongoing involvement from the credential issuer. Early implementations have focused on educational credentials, professional certifications, and organizational memberships, with potential expansions into government-issued identities and healthcare verifications [Microsoft, 2025].

Zero Trust architecture integration represents another critical direction for Azure Entra ID's evolution. Moving beyond traditional perimeter-based security, Microsoft's Zero Trust implementation centers on continuous verification of identity, device health, and risk signals before granting resource access. The integration of OAuth 2.0 and OpenID Connect with Zero Trust principles manifests through several emerging capabilities: continuous access evaluation that enables near real-time token revocation based on security events, step-up authentication that dynamically escalates authentication requirements for sensitive operations, and risk-based conditional access that adjusts token properties based on detected threat levels. These capabilities transform static token-based authorization into dynamic, context-aware access decisions that continually reassess trust throughout a session rather than solely at authentication time.

CONCLUSION

The integration of OAuth 2.0 and OpenID Connect with Azure Entra ID represents a critical foundation for modern identity and access management that will continue to evolve as security requirements and digital interactions become increasingly complex. Throughout this examination, we have demonstrated how these protocols provide the necessary framework for secure delegation, identity verification, and fine-grained authorization across diverse application architectures and organizational boundaries. The implementation patterns, security considerations, and architectural decisions discussed illustrate the balance organizations must strike between usability, security, and compliance requirements when deploying these protocols in production environments. As enterprise systems increasingly span multiple clouds, incorporate external identities, and face sophisticated security threats, the robust implementation of these protocols becomes not merely a technical concern but a fundamental business requirement. Looking forward, the continued evolution toward decentralized identity models and Zero Trust architectures will further transform how these protocols operate, moving from static, perimeter-based security toward dynamic, attribute-based access controls that continuously validate trust relationships. Organizations that develop deep competency in these protocols and their Azure implementations will be better positioned to adapt to these emerging paradigms while maintaining the security integrity essential for modern digital operations.

The opinions stated are personal and do not represent the stance or policies of any affiliated entity.

REFERENCES

1. Microsoft. "OAuth 2.0 and OpenID Connect protocols on the Microsoft identity platform." *Microsoft Identity Platform Documentation*, (2025). https://learn.microsoft.com/en-us/azure/active-directory/develop/active-directory-v2-protocols
2. Hardt, Ed, D. "RFC 6749: The OAuth 2.0 Authorization Framework." *Internet Engineering Task Force*, (2012). https://datatracker.ietf.org/doc/html/rfc6749
3. OpenID Foundation. "OpenID Connect Core 1.0." (2014). https://openid.net/specs/openid-connect-core-1_0.html
4. Microsoft. "What is Microsoft Entra?" *Microsoft Documentation*, (2024). https://learn.microsoft.com/en-us/entra/fundamentals/what-is-entra?source=recommendations
5. Microsoft. "Microsoft identity platform and OAuth 2.0 authorization code flow." *Microsoft Identity Platform Documentation*, (2025). https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-auth-code-flow
6. Microsoft. "Microsoft identity platform and OAuth 2.0 implicit grant flow." (2025). https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-implicit-grant-flow
7. Lodderstedt, Ed, T, *et al*. "OAuth 2.0 Threat Model and Security Considerations." *Internet Engineering Task Force (IETF)*, (2013). https://datatracker.ietf.org/doc/html/rfc6819
8. Auth0. "Token Best Practices." *Auth0 Documentation*.

Publisher: SARC Publisher

https://auth0.com/docs/secure/tokens/token-best-practices
9. Microsoft. "Azure Active Directory B2B collaboration - Adoption Kit." https://download.microsoft.com/download/F/C/A/FCA51098-4F99-4C14-9DF7-45E338E72158/B2B.pdf
10. Microsoft. "Quick Microsoft Entra Verified ID setup." (2025). https://learn.microsoft.com/en-us/entra/verified-id/verifiable-credentials-configure-tenant-quick

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:
Khaund, B. "OAuth 2.0 and OpenID Connect Integration with Azure Entra ID." *Sarcouncil Journal of Engineering and Computer Sciences* 4.6 (2025): pp 17-24.

Publisher: SARC Publisher