International Journal of Business Quantitative Economics and Applied Management Research
Volume-6, Issue-06, 2020 ISSN No: 2349-5677

ADVANCEMENTS IN IDENTITY AND ACCESS MANAGEMENT: EMBRACING OAUTH2 AND OPENID CONNECT

Ritesh Kumar
Independent Researcher
Pennsylvania, USA
ritesh2901@gmail.com

Abstract

Identity and Access Management (IAM) is a critical component of enterprise security, ensuring robust authentication and authorization across distributed systems. With the increasing adoption of cloud computing, API-driven architectures, and Zero Trust security models, OAuth 2.0 and OpenID Connect (OIDC) have emerged as the dominant standards for secure authorization delegation and identity federation. This paper provides a technical analysis of OAuth 2.0 and OIDC, examining their core principles, security mechanisms, and architectural considerations. Key advancements, including Proof Key for Code Exchange (PKCE), the deprecation of the Implicit Flow, and OAuth 2.0 security best practices, are explored in detail. Additionally, we analyze common security challenges such as token interception, replay attacks, and improper access delegation, along with mitigation strategies for securing authentication workflows. By offering a structured evaluation of these protocols, this paper aims to provide insights into their role in modern IAM frameworks and best practices for secure and scalable implementation.

Keywords: OAuth 2.0, OpenID Connect, Identity and Access Management (IAM), Authentication, Authorization, API Security, Federated Identity, Multi-Factor Authentication (MFA), Zero Trust, Security Best Practices

I. INTRODUCTION

In the evolving landscape of digital security, Identity and Access Management (IAM) has become a crucial component in protecting enterprise applications and user identities. IAM frameworks enable organizations to securely authenticate and authorize users, ensuring that access to sensitive resources is granted only to legitimate entities. Traditional authentication methods, such as password-based authentication and session-based access controls, are increasingly proving insufficient in modern distributed environments due to their susceptibility to security breaches, scalability limitations, and user experience challenges [4].

With the rapid adoption of cloud computing, microservices, and API-driven architectures, enterprises require scalable and secure authentication solutions that support decentralized access control while maintaining a seamless user experience [6]. OAuth 2.0 and OpenID

Connect (OIDC) have emerged as the dominant standards for achieving secure authentication and authorization in these environments. OAuth 2.0 provides a robust framework for delegated authorization, allowing applications to securely access resources on behalf of users without exposing their credentials [1]. OpenID Connect, built on top of OAuth 2.0, adds an authentication layer, enabling Single Sign-On (SSO), federated identity management, and standardized identity assertions across applications [3].

This paper presents a technical analysis of OAuth 2.0 and OpenID Connect, focusing on their core principles, security mechanisms, architectural considerations, and best practices. We explore key advancements such as Proof Key for Code Exchange (PKCE) [5] and the deprecation of the Implicit Flow [2], along with security challenges like token interception, replay attacks, and improper access delegation [4]. Additionally, the integration of these protocols into Zero Trust security models, API security, and federated IAM frameworks is examined to highlight their role in modern enterprise security architectures [9].

The structure of this paper is as follows: Section 2 introduces the fundamental concepts of OAuth 2.0 and OpenID Connect. Section 3 discusses security enhancements, best practices, and common security threats. Section 4 explores the role of these protocols in IAM architectures, including API security and federated authentication. Section 5 highlights challenges and emerging trends, followed by Section 6, which presents the conclusion.

II. FUNDAMENTALS OF OAUTH 2.0 AND OPENID CONNECT

Identity and Access Management (IAM) frameworks rely on standardized authentication and authorization protocols to ensure secure and seamless access to resources [6]. OAuth 2.0 and OpenID Connect (OIDC) are two of the most widely adopted protocols in modern IAM implementations [1], [3]. While OAuth 2.0 focuses on authorization delegation, OpenID Connect extends it with an authentication layer, making it suitable for identity federation and Single Sign-On (SSO)scenarios [3]. This section provides a detailed overview of both protocols, including their core principles, components, and flows.

A. OAuth 2.0: Authorization Framework

1. Overview of OAuth 2.0

OAuth 2.0 is an industry-standard framework designed for delegated authorization, allowing third-party applications to access resources on behalf of users without exposing their credentials [1]. Instead of using passwords, OAuth 2.0 employs access tokens, which are granted by an Authorization Server and used to access protected resources [1]. This approach enhances security, reduces password exposure risks, and enables seamless user experiences across multiple applications [5].

2. Key Components of OAuth 2.0

OAuth 2.0 defines four main components that interact in the authorization process [1]:

a) **Resource Owner:** The user who grants permission to access their protected resources.

b) **Client:** The application requesting access to the user's resources.
c) **Authorization Server:** The entity responsible for issuing access tokens after validating the client's request.
d) **Resource Server:** The API or service that validates the access token and provides requested resources.

3. OAuth 2.0 Grant Types

OAuth 2.0 provides different authorization flows, known as grant types, designed for various application architectures and security requirements [1], [5].

a) Authorization Code Grant (Recommended for Web and Mobile Apps)

- The most secure flow as it prevents exposure of credentials and tokens in the browser [1].
- Requires an authorization code from the Authorization Server before exchanging it for an access token.
- Often combined with Proof Key for Code Exchange (PKCE) to mitigate code interception attacks [5].

b) Implicit Flow (Deprecated)

- Originally designed for browser-based apps where tokens were directly returned via the redirect URI.
- Deprecation reason: High risk of token leakage due to exposure in URL fragments [2].

c) Client Credentials Grant (For Machine-to-Machine Communication)

- Used when no end-user is involved, such as when a service requests access to another service.
- Tokens are issued based on client authentication (client ID and secret) [1].

d) Resource Owner Password Credentials (Not Recommended)

- Directly exchanges the user's username and password for an access token.
- Security risk: Increases credential exposure and is discouraged in favor of more secure flows [1], [4].

4. OAuth 2.0 Token Types

OAuth 2.0 relies on different token types to facilitate secure authentication and authorization workflows [1], [3].

a) **Access Token:** Short-lived credential used to access protected resources.
b) **Refresh Token:** Long-lived token used to obtain new access tokens without requiring re-authentication [1].
c) **ID Token (only in OpenID Connect):** Contains identity claims about the authenticated user [3].

B. OpenID Connect: Authentication Layer over OAuth 2.0

1. Overview of OpenID Connect

OpenID Connect (OIDC) is an identity layer built on top of OAuth 2.0, providing authentication capabilities [3]. While OAuth 2.0 only handles authorization, OIDC enables applications to verify user identities and obtain user attributes through JSON Web Tokens (JWTs) [11]. This makes OIDC the preferred solution for Single Sign-On (SSO), identity federation, and decentralized authentication [3].

2. Key Components of OpenID Connect

OIDC extends OAuth 2.0 with additional components to support authentication [3]:

a) **OpenID Provider (OP):** The authorization server that authenticates users and issues ID tokens.
b) **Relying Party (RP):** The application (client) that requests authentication from the OpenID Provider.
c) **ID Token:** A JWT containing user identity claims, such as the user's email, name, and authentication time [3], [11].

3. OpenID Connect Authentication Flows

Similar to OAuth 2.0, OIDC supports multiple flows based on application type [3]:

a) Authorization Code Flow (Recommended)

- The most secure and commonly used authentication flow.
- Requires an authorization code before obtaining ID and access tokens [3].
- Supports PKCE for improved security in mobile and browser applications [5].

b) Implicit Flow (Deprecated)

- Initially designed for SPAs (Single Page Applications) but now discouraged due to security concerns [2].

c) Hybrid Flow

- A combination of Authorization Code and Implicit Flow.
- Allows immediate access to ID tokens while maintaining security benefits of authorization codes.

4. OpenID Connect Scopes and Claims

OIDC introduces scopes and claims to define the level of access requested [3]:

a) **Scopes:** Indicate the type of user data requested (e.g., openid, profile, email) [11].
b) **Claims:** Specific user attributes included in the ID token (e.g., name, email, sub) [3].

5. Security Considerations in OpenID Connect

OIDC introduces security enhancements to prevent token misuse and improve authentication robustness [3], [11]:

a) **ID Token Validation**: Ensuring proper signature verification to prevent token forgery [3].
b) **Nonce Parameter:** Used to mitigate replay attacks [3].
c) **Token Expiry and Revocation:** Implementing session timeouts and refresh policies [11].

III. SECURITY ENHANCEMENTS AND BEST PRACTICES IN OAUTH 2.0 AND OPENID CONNECT

Security is a critical concern in Identity and Access Management (IAM), especially when dealing with authentication and authorization protocols that control access to sensitive resources [4]. While OAuth 2.0 and OpenID Connect (OIDC) provide robust mechanisms for delegated access and identity verification, they are also susceptible to various security threats if not implemented correctly [5]. This section examines key security enhancements, best practices, and mitigation strategies to address common threats such as token interception, replay attacks, and improper access delegation [7].

A. OAuth 2.0 Security Best Practices

OAuth 2.0 has undergone several security improvements to address known vulnerabilities and enhance its robustness [1]. The following best practices should be followed to secure OAuth 2.0 implementations:

1. Proof Key for Code Exchange (PKCE)

a) **Purpose:** Prevents authorization code interception attacks, particularly in public clients (e.g., mobile and single-page applications) [5].

b) How It Works:

- The client generates a random code verifier and derives a code challenge from it.
- During the authorization request, the code challenge is sent to the Authorization Server.
- When exchanging the code for an access token, the code verifier is sent to prove the legitimacy of the request.
- The Authorization Server ensures the code verifier matches the challenge before issuing tokens [5].

2. Deprecation of Implicit Flow

a) **Security Risk**: Tokens are exposed in URLs, making them vulnerable to man-in-the-middle (MITM) and token leakage attacks [2].

b) **Mitigation**: Use the Authorization Code Flow with PKCE instead of Implicit Flow for public clients [2].

3. Access Token Scoping and Audience Restriction

a) **Principle:** Limit the access granted to tokens to only the necessary resources.

b) Implementation [3]:

- Use scopes to define fine-grained access control.
- Implement token binding to restrict tokens to specific clients or sessions.
- Define audience (aud) claims in tokens to prevent token misuse across APIs [11].

4. Secure Storage and Transmission of Tokens

- Access and refresh tokens should be stored securely using encrypted storage mechanisms [1].
- Avoid storing tokens in local/session storage in browsers, as they can be accessed by malicious scripts [4].
- Use TLS (Transport Layer Security) encryption for all token transmissions to prevent eavesdropping [7].

B. OpenID Connect Security Best Practices

Since OpenID Connect builds on OAuth 2.0, it inherits its security concerns while introducing additional risks related to identity assertion and authentication tokens (ID Tokens) [3]. The following best practices enhance OpenID Connect security:

1. ID Token Validation and Signature Verification

a) Why It's Important: Prevents unauthorized token forgery and tampering [3].
b) How to Implement [3]:

- Verify the signature of ID Tokens using JSON Web Key Sets (JWKS) from the OpenID Provider.
- Check the issuer (iss) and audience (aud) claims to ensure the token is intended for the relying party.
- Validate the expiration time (exp) claim to prevent token reuse.

2. Use of Nonce to Prevent Replay Attacks

a) Threat: Attackers may reuse an intercepted ID Token to impersonate a legitimate user [3].
b) Solution: The nonce parameter ensures that an ID Token is valid only for the original authentication request [3].

3. Secure Client Authentication

a) **Best Practice:** Public clients should use PKCE [5], while confidential clients should use client authentication mechanisms like:

- Mutual TLS (mTLS) for enhanced security [8].
- Private Key JWT authentication instead of client secrets for improved protection [8].

C. Common Security Threats and Mitigation Strategies

1. Token Interception and Replay Attacks

a) Attack: An attacker captures an access or ID token and reuses it to gain unauthorized access [4].
b) Mitigation [7]:

- Use short-lived access tokens and require frequent refresh token rotation.
- Implement aud and iss claims in tokens to prevent cross-service misuse.
- Enforce TLS encryption for all token transmissions.

2. Authorization Code Injection and Fixation

a) Attack: An attacker tricks a user into authorizing an application using a pre-obtained authorization code [5].

b) Mitigation [5]:

- Implement PKCE to ensure that authorization codes are bound to the requesting client.
- Enforce strict validation of redirect URIs to prevent redirection attacks.

3. Access Token Leakage and Misuse

a) Attack: An access token is leaked due to weak storage or improper exposure in logs [4].

b) Mitigation [7]:

- Store tokens securely using hardware security modules (HSMs) or secure enclaves.
- Implement Token Binding to ensure tokens can only be used within their intended context.

4. Security Considerations for Single Sign-On (SSO)

a) Threat: SSO can be a single point of failure if improperly secured [3].

b) Best Practices [9]:

- Use multi-factor authentication (MFA) to strengthen user identity verification.
- Apply session expiration policies to mitigate session hijacking risks.
- Monitor SSO events and enforce anomalous login detection.

IV. OAUTH 2.0 AND OPENID CONNECT IN IAM ARCHITECTURES

As modern enterprises transition towards cloud-native, microservices, and API-driven architectures, Identity and Access Management (IAM) plays a crucial role in ensuring secure authentication and authorization [6]. OAuth 2.0 and OpenID Connect (OIDC) have become fundamental components of IAM frameworks, providing secure API access, identity federation, and Single Sign-On (SSO). This section explores the integration of OAuth 2.0 and OpenID Connect in various IAM architectures, including API security, federated identity management, and Zero Trust security models [9].

A. OAuth 2.0 for API Security and Access Control

1. The Role of OAuth 2.0 in API Security

OAuth 2.0 is widely used to secure RESTful APIs, ensuring that only authorized clients can access protected resources [1]. Instead of handling credentials directly, APIs rely on access tokens issued by an Authorization Server, which enforces authentication and authorization policies [3].

2. API Gateway and OAuth 2.0 Integration

- API Gateways act as intermediaries between clients and backend services, enforcing authentication and authorization policies [7].
- OAuth 2.0 is integrated with API gateways to validate access tokens, ensuring that API requests are properly authenticated [8].

- API Gateway capabilities include:
  - Token validation before processing API requests.
  - Rate limiting and quota enforcement to prevent API abuse.
  - Scope-based access control to restrict API operations based on the token's scope.

3. Role-Based and Attribute-Based Access Control (RBAC & ABAC)

OAuth 2.0 tokens can be leveraged for fine-grained access control in API security [11]:

a) Role-Based Access Control (RBAC): Uses roles assigned to users (e.g., Admin, User, Viewer) to restrict API actions.

b) Attribute-Based Access Control (ABAC): Uses additional attributes (claims) within OAuth 2.0 tokens (e.g., department, geolocation) to enforce dynamic access policies [10].

B. OpenID Connect in Federated Identity Management

1. Identity Federation with OpenID Connect

Federated identity management allows users to authenticate once and access multiple services without re-entering credentials. OpenID Connect simplifies identity federation by enabling trust relationships between Identity Providers (IdPs) and Service Providers (SPs) [11].

2. Components of Federated Identity Using OIDC

a) Identity Provider (IdP): The trusted entity that authenticates users and issues ID Tokens (e.g., Google, Microsoft Azure AD, Okta) [3].

b) Service Provider (SP): The relying party (RP) that consumes identity assertions from the IdP [3].

c) Claims and Scopes: Service Providers request specific identity claims (e.g., email, name, groups) from the IdP to grant appropriate access levels [11].

3. OpenID Connect Discovery and Dynamic Client Registration

- OIDC Discovery enables automatic retrieval of IdP metadata, simplifying integration with federated identity systems.
- Dynamic Client Registration allows new clients (applications) to securely register with an IdP without manual configuration [11].

4. Single Sign-On (SSO) with OpenID Connect

- OpenID Connect enables SSO across multiple applications, allowing users to log in once and access multiple services without repeated authentication.
- Session management and logout handling prevent unauthorized access when users sign out from a service [11].

C. OAuth 2.0 and OIDC in Zero Trust Security Models

1. Introduction to Zero Trust Security

Zero Trust is a security paradigm that assumes no user or device should be inherently trusted, regardless of whether they are inside or outside the corporate network [9]. OAuth 2.0 and

OpenID Connect play a crucial role in implementing Zero Trust access policies by enforcing strict authentication and authorization controls [8].

2. OAuth 2.0 and Zero Trust Access Controls

a) Least Privilege Access: OAuth 2.0 scopes ensure that clients only receive minimal permissions required for their tasks [1].
b) Continuous Authorization: Access tokens can be short-lived, requiring continuous reauthorization based on risk analysis [7].
c) Token-Based Policy Enforcement: Security policies are enforced dynamically by evaluating claims within OAuth 2.0 tokens [10].

3. Continuous Authentication with OpenID Connect

- OpenID Connect enables adaptive authentication mechanisms, such as Multi-Factor Authentication (MFA), when risk levels change.
- Identity Providers (IdPs) evaluate risk factors (e.g., geolocation, device fingerprinting) before issuing authentication tokens.

4. OAuth 2.0 and OIDC in Microservices Security

a) Decentralized IAM: Each microservice validates OAuth 2.0 tokens independently, enforcing access policies at the service level.
b) Token Propagation: When a microservice calls another microservice, the original OAuth 2.0 token can be forwarded or exchanged for a service-specific token [10].
c) Mutual TLS (mTLS) for OAuth 2.0: Enhances security by ensuring only trusted services can request access tokens [8].

V. CHALLENGES AND FUTURE CONSIDERATIONS

While OAuth 2.0 and OpenID Connect (OIDC) have become the de facto standards for modern authentication and authorization, their implementation comes with various challenges related to security, scalability, interoperability, and compliance [4]. Additionally, as security landscapes evolve, these protocols must adapt to emerging trends such as decentralized identity, next-generation OAuth improvements, and passwordless authentication [9]. This section explores the challenges associated with OAuth 2.0 and OIDC and discusses future advancements that could shape the evolution of Identity and Access Management (IAM).

A. Challenges in OAuth 2.0 and OpenID Connect Implementations

1. Complexity in Managing Token Lifecycle

OAuth 2.0 uses access tokens and refresh tokens to manage authentication sessions, but improper handling of token expiration and revocation can lead to security risks [5]:

a) Challenges [4], [7]:

- If access tokens are too long-lived, they increase the risk of token leakage and misuse.
- If access tokens are too short-lived, frequent reauthorization can degrade user experience.
- Token revocation is not always immediate, leading to potential session hijacking risks.

b) Mitigation:

- Implement short-lived access tokens with refresh token rotation [1].
- Enforce token revocation policies and token binding mechanisms to reduce the risk of token theft [8]

2. Interoperability Across Different IAM Systems

Organizations often need to integrate OAuth 2.0 and OIDC with existing IAM frameworks (e.g., SAML, LDAP, Kerberos) [6] :

a) Challenges:

- Legacy applications may not support OAuth 2.0 or require significant modifications [6].
- Different IdPs may implement OAuth 2.0/OIDC with slight variations, causing compatibility issues [11].

b) Mitigation:

- Use OIDC Discovery and Dynamic Client Registration to improve interoperability [11].
- Implement protocol transition mechanisms to bridge OAuth 2.0 with legacy authentication protocols [7].

3. Compliance and Regulatory Constraints

OAuth 2.0 and OIDC implementations must comply with data protection regulations such as GDPR, HIPAA, and CCPA [9]:

a) Challenges:

- Ensuring user consent and data minimization when sharing identity attributes [10].
- Protecting personally identifiable information (PII) stored in ID tokens.

b) Mitigation:

- Implement scoped access control and encrypt sensitive identity claims in JWT tokens [3].
- Ensure auditable logging of authentication and authorization events [9].

4. OAuth 2.0 in Distributed and Microservices Architectures

OAuth 2.0 is widely used in microservices security, but applying it effectively can be challenging:

a) Challenges:

- Each microservice must validate OAuth 2.0 tokens independently.
- Challenge: Secure token propagation across multiple microservices without exposing sensitive information.

b) Mitigation:

- Use service-to-service authentication via JWT-based OAuth tokens or mutual TLS (mTLS) [8].
- Implement fine-grained token scopes and audience (aud) restrictions to prevent token misuse [11].

B. Emerging Trends and Future Directions

1. Future OAuth Enhancements: Standardizing Security Best Practices

The next iteration of OAuth is expected to incorporate security best practices and remove deprecated features to improve implementation consistency and security [2].

a) Expected Enhancements in Future OAuth Versions:

- Deprecation of Implicit Flow: Improves security by enforcing Authorization Code Flow with PKCE.
- Enhanced Token Binding: Strengthens token security by linking tokens to specific clients.
- Simplified Core Specification: Reduces unnecessary complexity in OAuth 2.0 implementations.
- Future Impact: Future OAuth updates are anticipated to enhance security, simplify the authorization process, and adopt modern authentication techniques to ensure better scalability and resilience against emerging threats.

2. Decentralized Identity and Self-Sovereign Identity (SSI)

Traditional IAM models rely on centralized identity providers (Google, Microsoft, Okta), but Decentralized Identity (DID) and Self-Sovereign Identity (SSI) aim to give users full control over their identities:

a) Concept: Users store identity credentials in digital wallets (e.g., W3C Verifiable Credentials) instead of relying on third-party providers.

b) Integration with OAuth 2.0/OIDC:

- OAuth and OIDC are being adapted to work with decentralized identity frameworks, such as DIDComm and Hyperledger Indy.
- Future IAM solutions may shift towards blockchain-based identity verification.

3. Passwordless Authentication and FIDO2/WebAuthn

OAuth 2.0 and OIDC are evolving towards passwordless authentication, reducing reliance on traditional passwords [9]:

a) FIDO2 and WebAuthn allow authentication via biometric sensors (fingerprint, face ID) or security keys.

b) OAuth/OIDC Adaptation:

- IdPs are integrating FIDO2-based authentication for OAuth login flows.
- Users authenticate with hardware-backed security credentials instead of passwords.
- Future Impact: Enhanced security and elimination of password-based phishing attacks.

4. AI-Driven Adaptive Authentication

AI and machine learning are being integrated into OAuth 2.0/OIDC systems for real-time risk assessment [10]:

a) Context-Aware Authentication:

- Analyzes user behavior, device fingerprints, and location to determine authentication risk levels.
- Dynamically enforces Multi-Factor Authentication (MFA) based on risk scores.

b) Anomaly Detection in OAuth Flows:

- AI models monitor OAuth 2.0 transactions for suspicious activity (e.g., unauthorized token exchanges).

c) **Future Impact:** AI-driven security enhancements will strengthen IAM resilience against evolving threats [10].

VI. CONCLUSION

Identity and Access Management (IAM) is a fundamental component of modern security architectures, ensuring secure authentication and authorization across distributed systems [6]. With the increasing reliance on cloud computing, API-driven architectures, microservices, and Zero Trust models, robust authentication mechanisms have become essential [9]. OAuth 2.0 and OpenID Connect (OIDC) have emerged as the dominant standards for secure authorization and identity federation, enabling organizations to manage user access in a scalable and secure manner [3].

This paper provided a technical analysis of OAuth 2.0 and OpenID Connect, covering their core principles, security mechanisms, implementation best practices, and architectural integrations [1], [3]. We examined critical security enhancements such as Proof Key for Code Exchange (PKCE), token lifecycle management, and the deprecation of the Implicit Flow to mitigate common threats like token interception, replay attacks, and unauthorized access [5]. Additionally, the integration of OAuth 2.0 and OIDC in API security, federated identity, and Zero Trust security models was explored, demonstrating their crucial role in enterprise IAM strategies [9].

Despite their widespread adoption, OAuth 2.0 and OIDC face challenges related to token management, interoperability, and regulatory compliance [7]. Future advancements, including next-generation OAuth improvements, decentralized identity frameworks, passwordless authentication via FIDO2/WebAuthn, and AI-driven adaptive authentication, will continue to refine and strengthen IAM frameworks [10]. As identity-based security threats evolve, the adoption of best practices, continuous monitoring, and advanced authentication mechanisms will be essential for securing digital ecosystems [11].

In conclusion, OAuth 2.0 and OpenID Connect remain pivotal in the IAM landscape, offering scalability, security, and interoperability for modern applications [3]. By implementing secure authentication workflows, enforcing strong access control policies, and adapting to emerging IAM technologies, organizations can future-proof their identity management strategies and mitigate evolving security threats [9].

REFERENCES

1. Hardt, "The OAuth 2.0 Authorization Framework," Internet Engineering Task Force (IETF), RFC 6749, Oct. 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749
2. M. Jones, B. Campbell, and C. Mortimore, "OAuth 2.0 Token Exchange," Internet Engineering Task Force (IETF), RFC 8693, Jan. 2020. [Online]. Available: https://tools.ietf.org/html/rfc8693
3. N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "OpenID Connect Core 1.0," OpenID Foundation, Nov. 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html
4. Fett, R. Küsters, and G. Schmitz, "A Comprehensive Formal Security Analysis of OAuth 2.0," in Proc. ACM Conf. Comput. Commun. Security (CCS), 2016, pp. 1204–1215. DOI: 10.1145/2976749.2978385.
5. M. Jones and D. Hardt, "OAuth 2.0 Proof Key for Code Exchange (PKCE)," Internet Engineering Task Force (IETF), RFC 7636, Sept. 2015. [Online]. Available: https://tools.ietf.org/html/rfc7636
6. Nadalin, P. Hallam-Baker, J. Hodges, R. Philpott, and E. Maler, "Security Assertion Markup Language (SAML) v2.0," OASIS Standard, Mar. 2005. [Online]. Available: https://docs.oasis-open.org/security/saml/v2.0/
7. K. L. Kuppusamy and K. Thilagavathi, "Security Challenges in OAuth 2.0 and OpenID Connect Protocols," International Journal of Information Security Science, vol. 7, no. 3, pp. 96-106, 2018.
8. M. Jones and M. Scurtescu, "OAuth 2.0 for Native Apps," Internet Engineering Task Force (IETF), RFC 8252, Oct. 2017. [Online]. Available: https://tools.ietf.org/html/rfc8252
9. FIDO Alliance, "Web Authentication: An API for Accessing Public Key Credentials Level 2," World Wide Web Consortium (W3C), Apr. 2019. [Online]. Available: https://www.w3.org/TR/webauthn-2/
10. Chadwick and G. Inman, "Attribute Aggregation in Federated Identity Management," IEEE Computer Society Transactions on Dependable and Secure Computing, vol. 9, no. 3, pp. 409-422, May 2012.
11. OpenID Foundation, "OpenID Connect Discovery 1.0," OpenID Standard, Feb. 2018. [Online]. Available: https://openid.net/specs/openid-connect-discovery-1_0.html