# OAuth 2.0 Authentication Framework

## Overview

Genesys Cloud's Platform API implements authorization flows described in the OAuth 2.0 standard (RFC 6749), which is an authorization framework that enables external applications to obtain limited access to HTTP services with user consent. OAuth makes a clear distinction between three key entities:

- **Resource Owner**: The Genesys Cloud user who owns the data
- **Client**: The application requesting access (your integration)
- **Authorization Server**: Genesys Cloud authentication service
- **Resource Server**: Genesys Cloud Platform API

---

## OAuth 2.0 Concepts & Terminology

Key OAuth 2.0 Components:

Resource Owner:
├─ The user who owns the data
├─ Genesys Cloud user account
├─ Grants permission to applications
└─ Has specific permissions/roles

Client (Application):
├─ The application requesting access
├─ Mobile app, web app, service, bot
├─ Registered as OAuth client in Genesys Cloud
├─ Has Client ID and Client Secret
└─ Requests scopes during authorization

Authorization Server:
├─ Genesys Cloud authentication service
├─ Validates user credentials
├─ Issues access tokens
├─ Manages token lifecycle
└─ Enforces scope limits

Resource Server:
├─ Genesys Cloud Platform API
├─ Protects API resources
├─ Validates access tokens
├─ Enforces token scopes
└─ Returns authorized data

Access Token:
├─ Short-lived credential (1 hour typical)
├─ Proves authorized access
├─ Sent with every API request
├─ HTTP Authorization header
├─ Bearer token format: "Bearer {token}"
└─ Automatically revoked on expiration

Refresh Token:
├─ Long-lived credential (30 days to 450 days)
├─ Used to obtain new access token
├─ Never expires unless explicitly revoked
├─ Not sent with API requests
├─ Stored securely server-side
└─ Can be rotated on each refresh

Scope:
├─ Granular permission specification
├─ Limits application access
├─ Implements principle of least privilege
├─ Combined as space-separated list
├─ Examples: "conversations:readonly" "users:manage"
├─ Requested at authorization time
└─ Enforced on every API call

Authorization Code:
├─ Short-lived code (valid ~10 minutes)
├─ Single-use only
├─ Exchanged for access token
├─ Valid only with client_secret
├─ Returned via redirect URI
└─ Not the access token itself

Client ID:
├─ Public identifier for application
├─ Shared in browser/mobile clients
├─ Used in authorization requests
├─ Identifies which app is making request
└─ Can be displayed in logs

Client Secret:
├─ Confidential application credential
├─ NEVER exposed to browser/mobile
├─ Used to exchange code for token
├─ Server-side only
├─ Rotated monthly minimum
├─ Regenerate if exposed
└─ View-once-only after creation (March 2026)

---

# Why OAuth 2.0?

Problem Solved:
├─ Users don't share passwords with applications
├─ Applications get limited, scoped access
├─ Users can revoke access without changing password
├─ Multiple applications can access same data
└─ No full account access needed

Key Advantages:
├─ Industry standard (RFC 6749)
├─ Secure delegation of access
├─ Granular permission control

├─ User consent and transparency
├─ Token-based (stateless for API)
├─ Revocation support
├─ Works across multiple protocols
└─ Widely implemented and trusted

Historical Context:
├─ OAuth 1.0: Complex signature-based (deprecated)
├─ OAuth 2.0: Token-based, simpler (current standard)
└─ OAuth 2.1: Emerging standard (future replacement)

Genesys Cloud Implementation:
├─ OAuth 2.0 standard compliant
├─ RFC 6749 Authorization Code Grant
├─ RFC 7636 PKCE (Proof Key for Code Exchange)
├─ RFC 6750 Bearer Token usage
└─ Multiple grant types supported

---

# Comparison: Basic Auth vs OAuth

Basic Authentication (DEPRECATED):

How it works:
├─ Username and password Base64 encoded
├─ Sent with every API request
└─ No expiration

Security Issues:
├─ Hash of credentials sent every request
├─ Increases exposure of sensitive data
├─ Hash never expires (unless password changed)
├─ Anyone intercepting hash gains full access
├─ Requires user to share credentials with app
├─ User has no way to revoke without password reset
└─ No granular permission control

Status in Genesys Cloud:

├─ HTTP Basic access authentication disabled
├─ Not supported for new implementations
└─ All integrations should use OAuth

OAuth 2.0 (RECOMMENDED):

How it works:
├─ User authenticates once
├─ App receives limited-access token
├─ Token sent with API requests
├─ Token expires automatically
└─ User never shares credentials with app

Security Advantages:
├─ Credentials only shared with authorization server
├─ Access tokens are short-lived (1 hour)
├─ Tokens can be revoked independently
├─ Granular scopes limit what app can do
├─ User can revoke without password reset
├─ Different apps have different access levels
└─ Transparent to user (they control permissions)

Status in Genesys Cloud:
├─ Primary authentication mechanism
├─ Recommended for all new integrations
├─ Multiple grant types supported
└─ Security best practices enforced

---

# OAuth 2.0 Security Principles

Core Security Concepts:

1. Never Trust the Client
├─ Always authenticate the application
├─ Verify client identity before issuing tokens
├─ Use client_secret for server-to-server

└ Don't assume client behaves correctly

2. Limit Token Scope
├ Grant only permissions needed
├ Principle of least privilege
├ Separate scopes for different functions
└ Can revoke individual scopes if needed

3. Short Token Lifespan
├ Access tokens: 1 hour (can be configured)
├ Reduces window for token misuse
├ Requires refresh token for persistence
├ Token becomes useless after expiration
└ Automatic cleanup reduces risk

4. Protect Sensitive Data
├ Client_secret: Server-side only
├ Refresh token: Secure storage required
├ Access token: HTTP Authorization header
├ Never log token values
└ Encrypt at rest and in transit

5. Validate All Input
├ Verify state parameter (CSRF protection)
├ Validate redirect URIs
├ Check token claims before trusting
├ Verify signature on tokens
└ Sanitize all user input

6. Use HTTPS Always
├ OAuth exchanges must use HTTPS
├ Unencrypted transmission compromises security
├ Validate SSL/TLS certificates
├ Protect against man-in-the-middle attacks
└ Never fall back to HTTP

7. Prevent Code Interception
├ PKCE adds code_verifier proof
├ Only original verifier can exchange code
├ Prevents authorization code theft

├─ Mandatory for public clients
└─ Best practice for all clients

8. Implement Audit Logging
├─ Log all authentication events
├─ Track token creation/refresh
├─ Monitor for anomalies
├─ Never log sensitive tokens
└─ Retain logs for compliance

---

# OAuth 2.0 vs Other Standards

Comparison with Alternative Approaches:

OAuth 2.0 vs SAML 2.0:
├─ OAuth: For API access, tokens
├─ SAML: For authentication, assertions
├─ OAuth: REST/lightweight
├─ SAML: XML/enterprise
├─ OAuth: Better for APIs
├─ SAML: Better for enterprise SSO
├─ Can be used together: SAML assertion → OAuth token

OAuth 2.0 vs OpenID Connect:
├─ OAuth 2.0: Authorization only
├─ OpenID Connect: OAuth 2.0 + Authentication
├─ OAuth: For API access
├─ OIDC: For user identity + API access
├─ Can use both in same implementation
└─ OIDC is superset of OAuth

OAuth 2.0 vs JWT (JSON Web Tokens):
├─ OAuth: Authorization framework
├─ JWT: Token format/encoding
├─ OAuth defines flow/process
├─ JWT is common OAuth token format
├─ Can use OAuth with non-JWT tokens

├─ Can use JWT outside OAuth
└─ Genesys uses Bearer tokens (could be JWT)

OAuth 2.0 vs Sessions/Cookies:
├─ OAuth: Stateless tokens (APIs)
├─ Sessions: Server-side state (web apps)
├─ OAuth: Better for distributed systems
├─ Sessions: Better for traditional web apps
├─ OAuth: No session storage needed
├─ Sessions: Requires shared session store
└─ Modern apps use both (API + web)

---

# Genesys Cloud OAuth Implementation

Genesys Cloud OAuth Features:

Supported Grant Types:
├─ Authorization Code Grant (RECOMMENDED)
├─ Authorization Code with PKCE (BEST for public clients)
├─ Client Credentials Grant (Service-to-service)
├─ Token Implicit Grant (DEPRECATED - May 2027 deadline)
└─ SAML2 Bearer Grant (Enterprise SSO)

Token Configuration:
├─ Token duration: 300 to 172,800 seconds (default 3600)
├─ SCIM special: Up to 38,880,000 seconds (450 days)
├─ HIPAA override: 15-minute idle timeout
├─ Automatic expiration handling
└─ Refresh token support

Scope Management:
├─ 100+ available scopes
├─ Granular permission control
├─ User consent on authorization
├─ Enforced on every API call

└─ Cannot exceed user permissions

OAuth Client Creation:
├─ Admin UI: Admin → Integrations → OAuth
├─ Up to 125 authorized redirect URIs
├─ Role assignment for Client Credentials
├─ Scope selection for all grant types
├─ Division scoping for role-based access
└─ Audit trail of creation/modifications

Security Features:
├─ Client secret: View-once-only (March 2026)
├─ IP whitelisting: Available for sensitive clients
├─ Multi-factor authentication: For administrators
├─ Audit logging: All OAuth events logged
├─ Token revocation: DELETE /oauth/sessions/me
└─ Scope enforcement: Both user AND token permissions required

Multi-Tenant Support:
├─ Separate OAuth clients per tenant
├─ Tenants can't cross-authenticate
├─ Isolated token stores
├─ Per-tenant rate limits
└─ Tenant-specific configuration

---

# Key Takeaways: Chapter 1

- **OAuth 2.0 is Standard** - Industry-best secure authorization framework
- **Three Key Entities** - Resource Owner (user), Client (app), Authorization Server
- **Five Key Concepts** - Access tokens, refresh tokens, scopes, authorization codes, client credentials
- **Security by Design** - Short-lived tokens, scope limitation, HTTPS required, credential protection
- **Industry Standard** - RFC 6749 compliant, widely implemented, trusted globally
- **Genesys Implementation** - 4+ grant types, granular scopes, audit logging, token management
- **Better than Basic Auth** - Never expose passwords, granular control, revocation support
- **Stateless & Scalable** - No session storage, works across distributed systems

# Interview Prep: OAuth 2.0 Concepts

| Question | Answer |
|---|---|
| What is OAuth 2.0? | Authorization framework enabling secure delegated access without sharing passwords |
| Three key entities? | Resource Owner (user), Client (app), Authorization Server (Genesys) |
| What's an access token? | Short-lived (1hr) proof of authorization sent with every API request |
| What's a refresh token? | Long-lived (30-450 days) credential used to obtain new access tokens |
| What are scopes? | Granular permissions limiting what an app can do (conversations:readonly, users:manage) |
| Why OAuth over Basic Auth? | Credentials never exposed, short-lived tokens, granular control, easy revocation |
| Why HTTPS required? | Unencrypted transmission exposes credentials and tokens |
| What is PKCE? | Proof Key for Code Exchange - prevents authorization code interception attacks |
| Why short token lifespan? | Reduces window for token misuse, requires refresh for persistence |
| How prevent CSRF? | Use state parameter in authorization code grant flow |

# Additional Resources

- **OAuth 2.0 Specification**: https://tools.ietf.org/html/rfc6749
- **PKCE (RFC 7636)**: https://tools.ietf.org/html/rfc7636
- **Bearer Token Usage (RFC 6750)**: https://tools.ietf.org/html/rfc6750
- **Genesys Developer Center**: https://developer.genesys.cloud/authorization/platform-auth/
- **Genesys Cloud Resource Center**: https://help.genesys.cloud

# Document Version

**Chapter**: 1 of 8
**Last Updated**: March 2026
**Status**: Current with RFC 6749, RFC 7636, RFC 6750
**Scope**: OAuth 2.0 Framework, Concepts, Principles

---

Revision #1
Created 14 March 2026 19:31:02 by Cesar Gzz
Updated 14 March 2026 19:31:17 by Cesar Gzz