OAuth for Model Context Protocol

Authorization Essentials for MCP Server Developers

What We'll Cover

- OAuth 2.1 fundamentals and authorization code flow
- Server discovery with Protected Resource Metadata (RFC9728)
- Authorization server capabilities with Server Metadata (RFC8414)
- Dynamic client registration (RFC7591)
- Security requirements for MCP implementations

Why OAuth for MCP?

MCP enables clients to access restricted servers on behalf of resource owners.

Authorization is OPTIONAL for MCP

- HTTP-based transports SHOULD use this OAuth-based specification
- STDIO transports SHOULD use environment credentials instead
- Alternative transports MUST follow protocol-specific security practices

OAuth 2.1: Evolution

OAuth 2.1 consolidates OAuth 2.0 with security best practices from multiple RFCs.

Required

- PKCE for all clients
- HTTPS for all endpoints
- Exact redirect URI matching

Removed

- Implicit grant flow
- Resource owner password flow
- Bearer tokens in query strings

Key OAuth 2.1 Concepts

Resource Owner

The user who authorizes access to protected resources

Client

Application requesting access (MCP client)

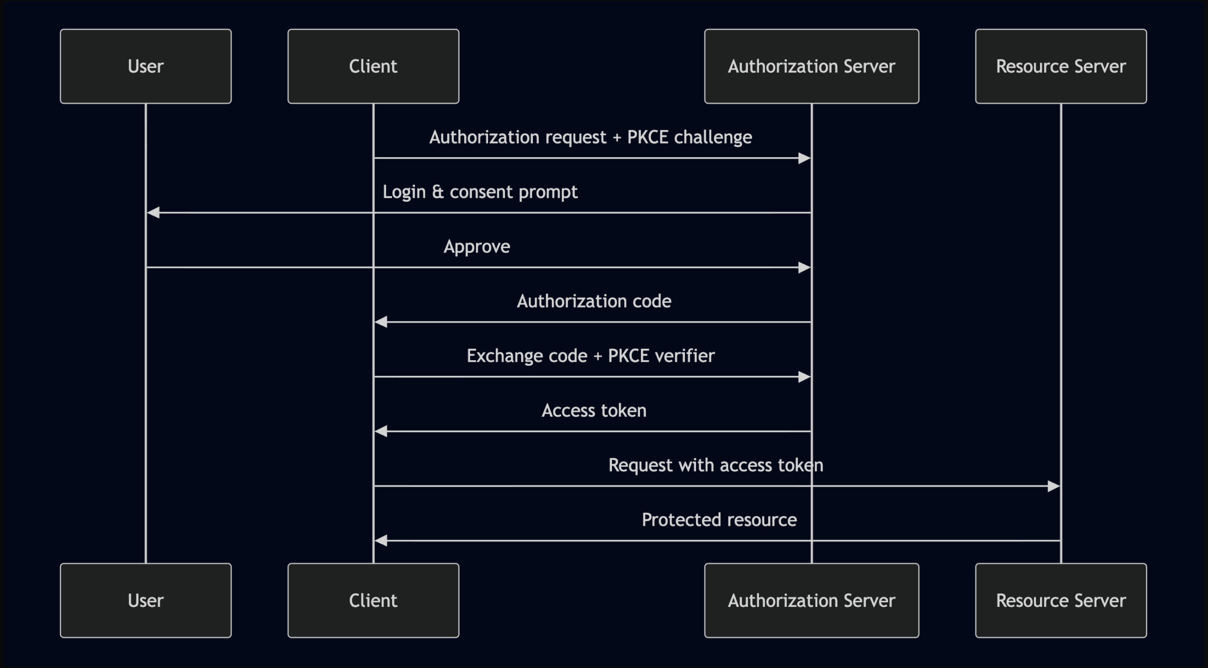
Authorization Server

Issues access tokens after authenticating the resource owner

Resource Server

Hosts protected resources (MCP server)

Authorization Code Flow



MCP-specific flow see e.g.
https://docs.scalekit.com/mcp/overview/#the-authorization-flow-in-practice

PKCE: Proof Key for Code Exchange

(See separate presentation for details)

PKCE prevents authorization code interception attacks.

Challenge

Client generates random code verifier, creates challenge (SHA256 hash), sends challenge with auth request

Verification

Client sends verifier with token request, server verifies hash matches original challenge

Required for ALL OAuth 2.1 clients

MCP OAuth Roles

MCP Client → OAuth Client

Makes protected resource requests on behalf of the user

MCP Server → OAuth Resource Server

Accepts and responds to requests using access tokens

Authorization Server

Issues access tokens (may be hosted with MCP server or separately)

Discovery Overview

Clients need to discover two things:

1

Where is the authorization server?

RFC9728: Protected Resource Metadata

2

What can it do?

RFC8414: Authorization Server Metadata

RFC9728: Protected Resource Metadata

MCP servers advertise their authorization servers using this standard.

MCP servers MUST:

- Implement Protected Resource Metadata
- Include authorization_servers field with at least one server
- Provide metadata document URL in WWW-Authenticate header on 401 response

MCP clients MUST:

- Parse WWW-Authenticate headers
- Use Protected Resource Metadata for authorization server discovery

WWW-Authenticate Header

When an MCP server returns 401 Unauthorized, it includes this header:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer
  as_uri="https://auth.example.com/.well-known/oauth-protected-resource"
```

The client fetches this URL to get the Protected Resource Metadata document.

RFC8414: Authorization Server Metadata

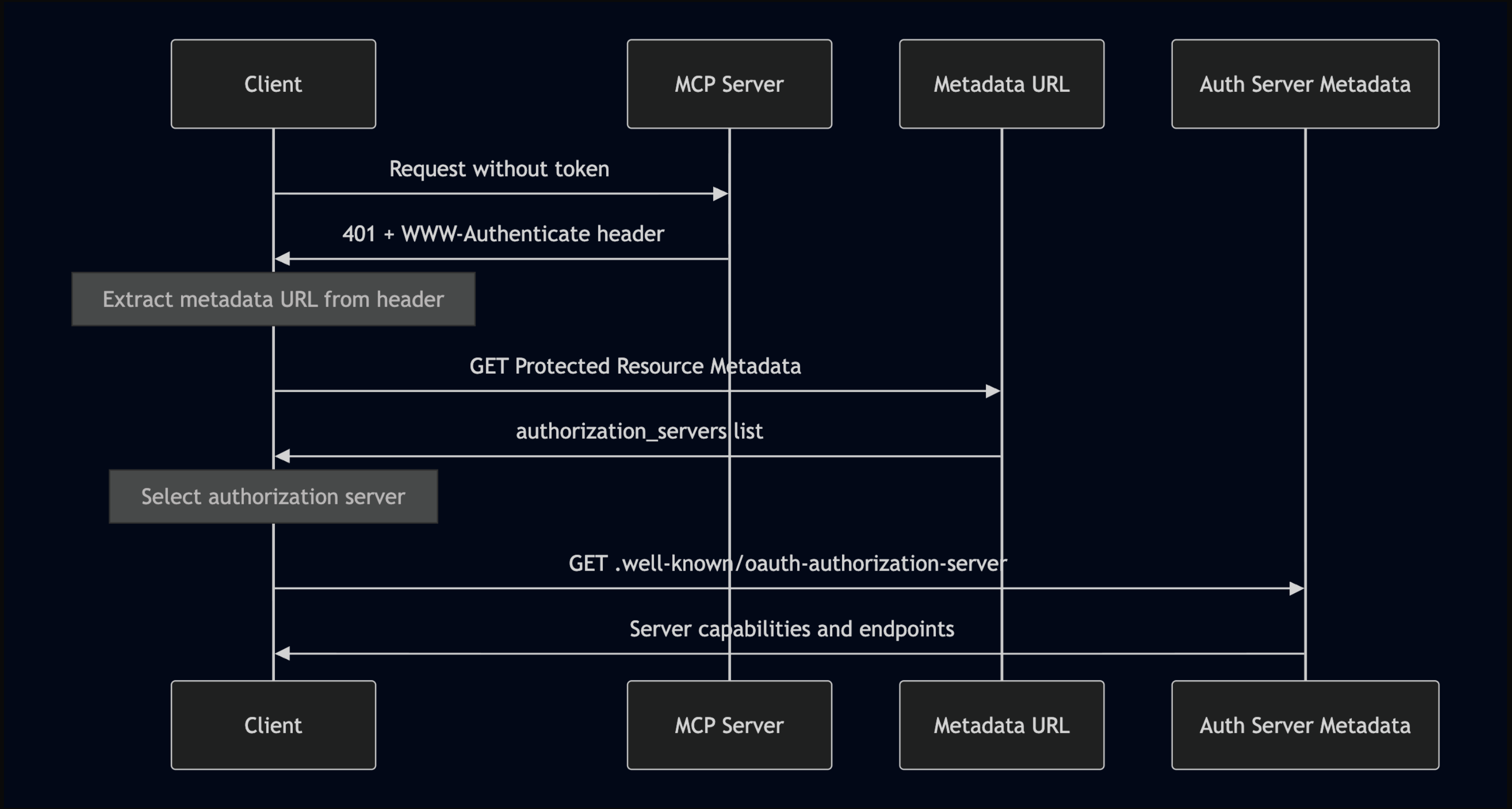
Once clients know where the authorization server is, they discover its capabilities.

Key metadata includes:

- Authorization endpoint URL
- Token endpoint URL
- Supported grant types and response types
- Supported scopes
- PKCE methods supported

Both authorization servers MUST provide this and clients MUST use it.

Metadata Discovery Flow


Client
MCP Server
Metadata URL
Auth Server Metadata
Request without token
401 + WWW-Authenticate header
Extract metadata URL from header
GET Protected Resource Metadata
authorization_servers list
Select authorization server
GET .well-known/oauth-authorization-server
Server capabilities and endpoints
Client
MCP Server
Metadata URL
Auth Server Metadata

RFC7591: Dynamic Client Registration

MCP clients and servers SHOULD support dynamic registration.

Why it matters for MCP:

- Clients may not know all possible MCP servers in advance
- Manual registration creates friction for users
- Enables seamless connection to new servers
- Authorization servers control their own registration policies

Without this, clients must hardcode client IDs or require manual configuration.

Dynamic Registration Process

1. Request

Client sends registration request with metadata (redirect URIs, grant types, etc.)

2. Response

Server returns client_id and optionally client_secret

3. Use

Client uses credentials for all subsequent auth requests

Registration happens automatically without user interaction

Complete MCP Authorization Flow

1. Discovery

Get 401, fetch metadata, discover auth server

2. Registration

Dynamically register to get client_id

3. Authorization Request

Redirect user with resource parameter and PKCE

4. User Consent

User authenticates and authorizes access

5. Token Exchange

Exchange code + PKCE verifier for access token

6. Resource Access

Use token in Authorization header

Resource Parameter (RFC8707)

MCP clients MUST use the resource parameter to bind tokens to specific MCP servers.

Purpose:

- Explicitly specifies which MCP server the token is for
- Prevents token misuse across different services
- Enables authorization servers to bind tokens to intended audience

Usage:

Include in both authorization requests and token requests

Canonical Server URI

The resource parameter value should be the most specific URI identifying the MCP server.

Valid examples:

- https://mcp.example.com
- https://mcp.example.com:8443
- https://mcp.example.com/server/mcp

Invalid examples:

- mcp.example.com (missing scheme)
- https://mcp.example.com#fragment

Lowercase scheme and host preferred

Access Token Usage

MCP clients MUST:

- Use Authorization header with Bearer scheme
- Never include tokens in URI query strings
- Only send tokens issued for the specific MCP server

Example:

```
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI...
```

Token Validation

MCP servers MUST:

- Validate tokens were issued by their authorization server
- Verify tokens were issued specifically for them (audience validation)
- Check token expiration and validity
- Return 401 for invalid or expired tokens
- Return 403 for valid tokens with insufficient permissions

Never accept tokens intended for other services

Security: Token Audience Binding

Token audience binding prevents confused deputy attacks.

The Problem:

Without audience validation, attackers could use tokens from Service A to access Service B.

The Solution:

- Use resource parameter to specify target
- Auth server binds token to audience
- Servers validate intended audience
- Never pass tokens to upstream services

Security: PKCE & Redirect Protection

PKCE (Required):

- Prevents authorization code interception
- Protects against code injection attacks
- Required for all MCP clients

Redirect URI Protection:

- Register redirect URIs with server
- Server validates exact URI matches
- Use state parameter for validation
- Prevents open redirection attacks

Security: Communication Requirements

HTTPS Everywhere:

- All authorization server endpoints MUST use HTTPS
- All redirect URIs MUST be localhost or HTTPS
- No exceptions for production deployments

Token Storage:

- Implement secure token storage
- Use short-lived access tokens
- Rotate refresh tokens for public clients

Common Pitfalls to Avoid

Skipping audience validation

Always verify tokens are for your server

Omitting resource parameter

Required in auth and token requests

Forgetting PKCE

Mandatory for all OAuth 2.1 clients

Token passthrough

Never forward tokens to upstream services

Using HTTP instead of HTTPS

All endpoints must use secure transport

Summary & Resources

Key Takeaways:

- OAuth 2.1 provides secure authorization for MCP over HTTP
- Discovery uses RFC9728 and RFC8414
- Dynamic registration (RFC7591) enables seamless onboarding
- Token audience binding prevents vulnerabilities
- PKCE, HTTPS, and validation are mandatory

Resources:

- MCP Spec: modelcontextprotocol.io/specification
- OAuth 2.1: draft-ietf-oauth-v2-1-13
- RFC8414, RFC7591, RFC9728, RFC8707