oauth
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2026

J. Chen
M. Chen
L. Su
China Mobile

1 March 2026

Structured and Constraint Extensions for OAuth Scopes
draft-chen-oauth-scope-agent-extensions-00

Abstract

This specification defines an extension to the OAuth 2.0 scope parameter, as specified in RFC 6749, which is used to express the permissions granted to an access token. The proposed extension introduces a structured syntax for scope values to enable fine-grained authorization for the installation and execution of Modular Capability Units (encapsulated and reusable functional modules such as skills) within Agent ecosystems. However, current mechanisms for authorizing such modules generally lack a standardized way to express and obtain consent for the complex, fine-grained permissions they require during installation.

This document addresses this limitation by defining a structured format for the scope parameter. The format extends the simple space-delimited strings with a colon-separated syntax: [resource_type]:[action]:[target][:constraints]. This syntax allows for the precise description of permissions for operations such as file system access, command-line execution, network access, tool invocation, and scheduled tasks. These extensions provide a standardized, machine-readable way to request, convey, and validate detailed permissions, thereby enhancing users' security control over their resources while maintaining compatibility with the existing OAuth token issuance and validation flows.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

The OAuth 2.0 scope parameter [RFC6749] serves as a widely adopted mechanism for expressing permissions, defining the access token's authority. However, in emerging scenarios such as the installation and execution of third-party skills within Agent ecosystems which require the declaration of complex, fine-grained operational permissions its expressive power becomes limited.

1.1. The Need for scope Extensions in AI Agent Ecosystems

In current Agent architectures, users typically extend their Agent's functionality by installing third-party developed Modular Capability Units. These Modular Capability Units MAY be specifically called "Skills" etc.,

but their essence is installable components that add specific capabilities to an Agent. However, this installation process presents significant security and privacy risks:

* Lack of Source Authentication: Users cannot verify the identity of the capability module's publisher or the integrity of its code.

* Coarse-Grained or No Permission Control: During installation, capability modules either obtain all requested permissions (often presented as a broad, unstructured list) or rely entirely on user trust, lacking a standardized process for fine-grained, interactive authorization confirmation at installation time.

* Runtime Permission Abuse Risk: Once installed, capability modules MAY access local or remote resources with permissions exceeding user expectations.

  The OAuth 2.0 framework [RFC6749] provides a standardized mechanism for delegated authorization. This proposal applies this framework to the Agent skill authorization model and, crucially, extends the OAuth scope parameter to support the expression of fine-grained permissions required by Modular Capability Units such as skills. This allows the installation and execution of a skill to be modeled as a process where a client obtains user consent for a specific set of structured permissions (scope), which are then executed during resource access.

1.2. Role Mapping for the Skill Installation and Authorization Context

This proposal applies the OAuth framework to a specific interaction pattern within Agent ecosystems: the installation and subsequent execution of Modular Capability Units (such as skills). To provide context for how the extended scopeparameter is used within this pattern, the entities involved are mapped to standard OAuth 2.0 roles as follows:

Resource Owner (RO): The end-user, who grants permissions.

Client: The personal Agent. It initiates skill installation and execution requests on behalf of the user.

Authorization Server (AS): A trusted entity responsible for verifying the skill’s source/integrity, managing scope policies, authenticating the user, and issuing access tokens containing the granted scope.

Resource Server (RS): (1) a Modular Capability Unit Management Server that hosts the unit repository and performs installation upon valid token presentation, and (2) a User Resource Server that controls access to the user’s specific resources (e.g., file system, network, tools) during a unit’s execution.

1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the terms "Authorization Server" (AS), "Client", "Resource Server" (RS), and "Access Token" as defined in The OAuth 2.0 Authorization Framework [RFC6749].

2. Authorization Flow Illustrating scope Negotiation

The following abstract flow illustrates how the extended scope parameter is negotiated and used within a modified OAuth Authorization Code flow:

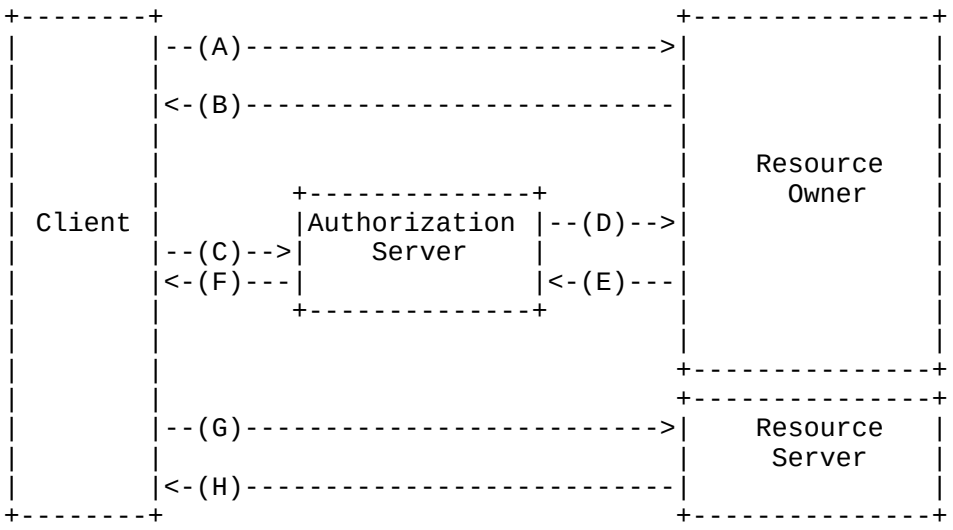

Figure 1: Skill Installation and Authorization Flow

(A) Client Requests Installation from Resource Owner.

(B) Resource Owner Initiates Authorization.

(C) Client Initiates Authorization Request to AS.

(D) AS Verifies Module and Negotiates Permissions: The Authorization Server performs the following sub-steps (omitting the AS's authentication of the Resource Owner):

* Fetches and verifies the capability module's metadata (including its requested initial permission list scope_initial) and its source and integrity (e.g., verifying publisher signature).

* Based on internal permission scope policies, maps or tailors scope_initial to a set of permissions scope_request (scope_request subset of scope_initial) that can be requested from the Resource Owner.

* Generates authorization consent interface information, presenting scope_request clearly to the Resource Owner. (E) Resource Owner Authorizes and Provides Consent: The Resource Owner reviews the authorization consent interface, can choose to accept, reject, or further adjust the authorization scope, forming the final consented permission set scope_granted (scope_granted subset of scope_request). After the Resource Owner confirms authorization, this authorization result (i.e., the authorization grant) is returned to the Client via interface interaction.

(F) AS Issues Authorization Code to Client.

(G) Client Requests Access Token.

(H) AS Issues Access Token.

(H) AS Issues Access Token:The AS issues an access token whose scope parameter value is set to scope_granted.

RS Validates Token and Scope:During skill execution, the RS validates the token and ensures the operation is within the token’s scope_granted.

3. Structured scope Format Extension

To meet the needs of Agent skills for describing complex, fine-grained operation permissions, this specification defines a structured extension to the OAuth scope parameter value syntax, building upon the ABNF defined in [RFC6749] :

The value of the scope parameter is expressed as a list of space-delimited, case-sensitive *structured scope tokens*. The syntax for a structured scope token is defined as:

```
structured-scope-token = resource-type
                         ":" action
                         ":" target
                         [ ":" constraints ]
                         [ ":" reserve ]


resource-type  = 1*( %x21-2F    ; visible chars except ":" and ";"
                 / %x30-3A
                 / %x3C-7E
                 )

action         = 1*( %x21-2F / %x30-3A / %x3C-7E )
target         = 1*( %x21-2F / %x30-3A / %x3C-7E )

constraints    = constraint *( ":" constraint )
constraint     = 1*( %x21-2F    ; often key=value pairs
                 / %x30-3A
                 / %x3C-7E
                 )
```

* resource-type: Identifier for the type of resource. This field is REQUIRED. New types introduced by this extension include:

  - fs: File system operations.

  - cmd: Command-line execution.

  - net: Network access.

  - tool: Tool/API invocation.

  - scheduler: Scheduled tasks.

  - othertype: Other resource types are reserved for resource types not defined in this specification. the specific semantics of other resource types are defined by the target component and/or through agreement between the AS and RS.

* action: The operation permitted on the specific resource type. This field is REQUIRED. For example:

  - For fs: read, write, list, delete.

  - For cmd: execute.

  - For net: connect, send, receive.

- For tool: invoke.

- For scheduler: create, read, update, delete.

- For all resource-type: otheraction (Other operations. The precise operation is defined by the combination of resource-type and target. Implementations MUST be prepared to handle unknown action values for a given resource-type‘.)

* target: Identifier for the target of the operation. This field is REQUIRED. Its syntax and meaning depend on resource-type and action. For example:

  - File path: /home/user/documents/*

  - Command line: /usr/bin/git

  - Network endpoint: api.example.com:443

  - Tool ID: weather_forecast

  - Scheduled task ID: daily_backup

  - When resource-type is othertype or action is otheraction, the target field MAY contain an implementation-specific or a more complex structured identifier (e.g., a URI or a name spaced string.

* constraints (Optional): Additional constraints on the authorization, in the form of key=value pairs. This field is OPTIONAL. Multiple constraints are separated by :. For example:

  - Time constraints: expires=2026-12-31T23:59:59Z, duration=PT2H (2 hours).

  - Logical constraints: if_condition=user_consented.

  - Regex constraints: path_regex=^/home/user/[^/]+/\.config$.

  - Recursion and depth limits: recursive=true:max_depth=5.

  - Authorization and resource servers SHOULD ignore any constraint key they do not understand.

* reserve (Optional): A reserved field for future extensibility. This field is OPTIONAL. This field contains an opaque string whose syntax and semantics are defined by a future specification. Implementations that do not understand the content of the reserve field MUST ignore it. This provides a standardized location for future extensions without breaking existing parsers.

3.1. Forward Compatibility Principle:

This structured extension itself is optional to implement. When parsing the scope value, the authorization server and resource server SHOULD first verify if a token's format conforms to the structured syntax defined in this specification. A token is interpreted and processed as a structured scope only if the format is strictly compliant; otherwise, it MUST be treated as a plain (non-extended) scope token.

Clients, AS, and RS implementing this specification SHOULD be designed to tolerate unknown values in the resource-type, action, and constraint keyfields. The preferred behavior upon encountering an unknown component is to treat that specific structured scope token as not granting the requested permission, rather than failing the entire request or token validation, unless a policy explicitly mandates strict validation of all components.

3.2. Example scope parameter values

Example scope parameter values (each line represents one possible space-delimited token within a scope string):

fs:read:/home/user/documents/:recursive=true:max_depth=5

cmd:execute:/usr/bin/git

net:connect:api.example.com:443

tool:invoke:weather_forecast

scheduler:create::interval=P1D (create a daily task)

The authorization server MAY fully or partially ignore the scope requested by the client, based on the authorization server policy or the resource owner's instructions. If the issued access token scope is different from the one requested by the client, the authorization server MUST include the "scope" response parameter to inform the client of the actual scope granted, as per [RFC6749].

3.3. Security Considerations

* scope Validation and Least Privilege: Authorization Servers and Resource Servers MUST strictly enforce validation of the structured scope. The granted permissions scope_granted MUST be a subset explicitly consented to by the user. When processing requests, the Resource Server MUST verify that the token's scope contains a structured token that precisely matches the requested operation, including resource-type, action, target, and any constraints.

* Skill Metadata Integrity: The Authorization Server's authentication of the skill's source and integrity (Step D) is critical. Strong cryptographic signatures (e.g., code signing certificates) SHOULD be used to prevent injection or tampering with malicious skills, especially their advertised scope_initial.

* User Consent Interface: The authorization consent interface (Step E) MUST clearly and unambiguously present the specific meaning of the structured scope_request tokens to the user, avoiding situations where users over-authorize due to misunderstanding.

4. Error Response

When a request is denied due to a failure in validating the structured scope(e.g., malformed syntax, unsupported resource-typeor action), the AS returns an error response with the following error code:

Error Code: scope_validation_failed

Description: The requested structured scope is invalid, malformed, or contains unsupported values.

The AS SHOULD include an error_description parameter to provide developers with more specific diagnostic information (e.g., "Unrecognized resource-type: 'custom_db'", "Malformed constraints segment").

5. IANA Considerations

5.1. OAuth Authorization Server Metadata

This specification requests the registration of the following values in the OAuth Authorization Server Metadata registry established by [RFC8414] : - structured_scope_resource_types_supported:

OPTIONAL. JSON array containing a list of the resource-typevalues (e.g., fs, cmd, net, tool, scheduler) that this authorization server supports and can process within structured scope tokens.

* structured_scope_actions_supported:

OPTIONAL. JSON array containing a list of the actionvalues (e.g., read, write, execute, connect) that this authorization server supports and can process within structured scope tokens.

5.2. OAuth Extensions Error Registry

This specification requests the registration of the following error code in the OAuth Extensions Error Registry:

* Error Code: scope_validation_failed

6. Acknowledgements

This document based on RFC6749

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/rfc/rfc8414>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

Authors' Addresses

Jia Chen
China Mobile
BeiJing
China
Email: chenjia@chinamobile.com

Meiling Chen
China Mobile
BeiJing
China
Email: chenmeiling@chinamobile.com

Li Su
China Mobile
BeiJing
China
Email: suli@chinamobile.com