authorization-control-inter HTTP/TLS/JSON

Interface Design Description

Abstract
This document describes a HTTP protocol with TLS payload security and JSON payload encoding variant of the **authorization-control-inter** service.

Contents

1 Overview

This document describes the **authorization-control-inter** service interface, which enables authorization control between local clouds. It's implemented using protocol, encoding as stated in the following table:

Profile type	Type	Version
Transfer protocol	HTTP	1.1
Data encryption	TLS	1.3
Encoding	JSON	RFC 8259 [1]
Compression	N/A	-

Table 1: Communication and semantics details used for the **authorization-control-inter** service interface

This document provides the Interface Design Description IDD to the *authorization-control-inter – Service Description* document. For further details about how this service is meant to be used, please consult that document.

The rest of this document describes how to realize the *authorization-control-inter* service HTTP/TLS/JSON interface in details.

2 Interface Description

The service responses with the status code `200 Ok` if called successfully. The error codes are, `400 Bad Request` if request is malformed, `401 Unauthorized` if improper client side certificate is provided, `500 Internal Server Error` if Authorizator is unavailable.

```
POST /authorization/intercloud/check HTTP/1.1

{
  "cloud": {
    "name": "string",
    "operator": "string",
  },
  "providerIdsWithInterfaceIds": [
    {
      "id": 0,
      "idList": [
        0
      ]
    }
  ],
  "serviceDefinition": "string"
}
```

Listing 1: An authorization-control-inter invocation.

```
{
  "authorizedProviderIdsWithInterfaceIds": [
    {
      "id": 0,
      "idList": [
        0
      ]
    }
  ],
  "cloud": {
    "name": "string",
    "operator": "string"
  },
  "serviceDefinition": "string"
}
```

Listing 2: An authorization-control-inter response.

3 Data Models

Here, all data objects that can be part of the service calls associated with this service are listed in alphabetic order. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is meant to denote a JSON Object that must contain certain fields, or names, with values conforming to explicitly named types. As a complement to the primary types defined in this section, there is also a list of secondary types in Section 3.5, which are used to represent things like hashes, identifiers and texts.

3.1 struct CheckAuthRuleRequest

Field	Type	Mandatory	Description
cloud	CloudDescriptor	yes	Descriptor of the consumer cloud.
providerIdsWithInterfaceIds	List<ProviderInterfaceIds>	yes	Array of provider and interface reference objects
serviceDefinition	Name	yes	Service definition name.

3.2 struct CloudDescriptor

Field	Type	Mandatory	Description
name	Name	yes	Name of the cloud
operator	Name	yes	Name of the cloud operator

3.3 struct ProviderInterfaceIds

Field	Type	Mandatory	Description
id	Number	yes	Database record identifier of the provider system
idList	List<Number>	yes	List of interface database record identifiers.

3.4 struct CheckAuthRuleResponse

Field	Type	Mandatory	Description
authorizedProviderIdsWithInterfaceIds	List<ProviderInterfaceIds>	yes	Array of the authorized provider and interface reference objects
cloud	CloudDescriptor	yes	Descriptor of the consumer cloud.
serviceDefinition	Name	yes	Service definition name.

3.5 Primitives

As all messages are encoded using the JSON format [2], the following primitive constructs, part of that standard, become available. Note that the official standard is defined in terms of parsing rules, while this list only concerns syntactic information. Furthermore, the Object and Array types are given optional generic type parameters, which are used in this document to signify when pair values or elements are expected to conform to certain types.

JSON Type	Description
Value	Any out of Object, Array, String, Number, Boolean or Null.
Object <A>	An unordered collection of [String: Value] pairs, where each Value conforms to type A.
Array <A>	An ordered collection of Value elements, where each element conforms to type A.
String	An arbitrary UTF-8 string.
Number	Any IEEE 754 binary64 floating point number [3], except for *+Inf*, *-Inf* and *NaN*.
Boolean	One out of `true` or `false`.
Null	Must be `null`.

With these primitives now available, we proceed to define all the types specified in the **authorization-control-inter** SD document without a direct equivalent among the JSON types. Concretely, we define the **authorization-control-inter** SD primitives either as *aliases* or *structs*. An *alias* is a renaming of an existing type, but with some further details about how it is intended to be used. Structs are described in the beginning of the parent section. The types are listed by name in alphabetical order.

3.5.1 alias Address = String

A string representation of a network address. An address can be a version 4 IP address (RFC 791), a version 6 IP address (RFC 2460) or a DNS name (RFC 1034).

3.5.2 alias List <A> = Array<A>

There is no difference.

3.5.3 alias Name = String

A String identifier that is intended to be both human and machine-readable.

3.5.4 alias PortNumber = Number

Decimal Number in the range of 0-65535.

4 References

[1] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 8259, Dec. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8259.txt

[2] ——, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 7159, 2014, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7159

[3] M. Cowlishaw, “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, July 2019. [Online]. Available: https://doi.org/10.1109/IEEESTD.2019.8766229

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	YYYY-MM-DD	4.6.0		Xxx Yyy

5.2 Quality Assurance

No.	Date	Version	Approved by
1	YYYY-MM-DD	4.6.0	Xxx Yyy