Authorization Decision Log

Logius Standard
Draft February 24, 2026

This version:
https://logius-standaarden.github.io/authorization-decision-log/

Latest published version:
https://gitdocumentatie.logius.nl/publicatie/dk/authorization-decision-log/

Latest editor's draft:
https://logius-standaarden.github.io/authorization-decision-log/

Editor:
Authors:
Maikel Hofman (VNG Realisatie)
Guus van der Meer (Vecozo)
Michiel Trimpe (VNG Realisatie)

Participate:
GitHub Logius-standaarden/authorization-decision-log
File an issue
Commit history
Pull requests

This document is also available in these non-normative format: PDF

Status of This Document

This is a draft that could be altered, removed or replaced by other documents. It is not a recommendation approved by TO.

Table of Contents

§ Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *MUST NOT*, *RECOMMENDED*, and *SHOULD* 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.

Abstract

This document defines a standardized method for logging decisions to allow or deny API requests to enable reconstruction and analysis of historical decisions.

§ 1. Introduction

The Authorization Decision Log standard defines a standard method for logging authorization decisions. The standard builds on the information models defined by [*AuthZen*] to log decisions.

It augments this with a structure for recording environmental factors that affected the evaluation decision. These are separated into active policies, additional information sources and configuration of the evaluation engine as per the architectural components introduced by [*NIST.SP.800-162*], also known as the "PxP" architecture.

Additionally, it includes a non-normative outline introducing the concerns, principles and requirements for developing and maintaining such a log in concordance with legislation.

§ 1.1 Purpose of this standard

This standard defines a uniform approach for logging authorization decisions, enabling organizations to provide effective accountability for historical decisions.

The standard provides a structured format for all contextual and environmental parameters that affect decisions. A full implementation of the standard allows historical decisions to be replayed for analysis.

§ 1.1.1 Applicability

Functionally, the standard is applicable to any API request. Organizationally, it applies to Dutch governmental bodies — such as the central government, provinces, municipalities, and water boards — as well as institutions in the public or semi-public sector.

§ 1.1.2 Target Audience

The target audience for the standard includes organizations involved in making and accounting for access decisions, both within their own organization and in collaborations with other parties. This may include the Dutch government and other relevant authorities.

§ 1.2 Terminology

The following list defines terminology used throughout this document.

Authorization

Authorization is the process of deciding whether to, fully or partially, allow or deny requests for processing (API requests) and enforcing these decisions.

Externalized Authorization Management

Externalized Authorization Management (EAM) is an architectural pattern in which authorization decisions are made outside of the applications that enforce the decision.

Log

A structured record of events, called log records, generated by a service. Logs are intended for operational monitoring, auditing, and troubleshooting.

Log record

A single unit of information within a log, representing one recorded event. Each log record is immutable once written and collectively forms the log.

Policies

A set of one or more rules that determine whether a request should be allowed or denied.

Source

A source of one or more pieces of information, such as attributes or policies, which affected an authorization decision and which come from a single source.

When a source is later queried for historical information, a single identifier, such as a version, hash, or timestamp, should be sufficient to retrieve all historical information from that source.

Replay

The recreation of the environment in which an authorization decision took place, allowing for analysis of historical decisions.

§ 2. Architecture

This section describes the common EAM architecture within which the authorization decisions are logged.

§ 2.1 Context

The Authorization Decision Log defines a standard format for recording decisions made by Externalized Authorization Management (EAM) systems.

The goal of the standard is to enable the recreation of the environment in which historical authorization decisions were made to enable analysis of decisions while preventing unnecessary data duplication.

The inputs for records in the Authorization Decision Log come from the following standard EAM or PxP components as introduced in [*NIST.SP.800-162*] and adopt the information model introduced in [*AuthZEN*].

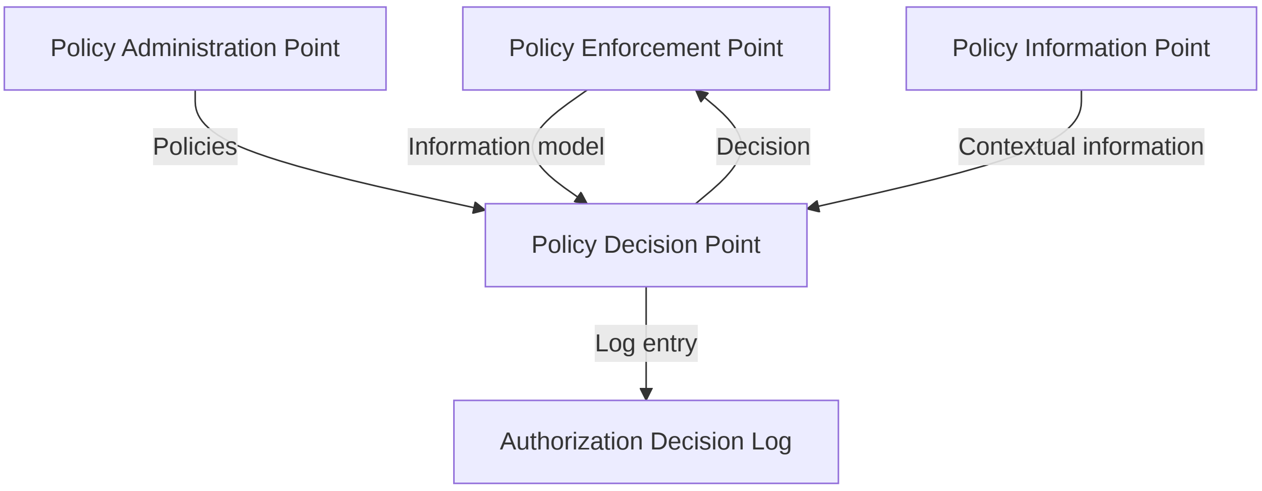

Figure 1 EAM or PxP Architecture

In a federated context, such as introduced by [*FSC-Core*], both the consumer outway and provider inway function as a PEP for incoming and outgoing requests. Both the consumer and the provider ask an internal PDP to decide on allowing the request. Both of these decisions can be logged using this standard.

When combined with tracing headers such as [*trace-context*] introduced by [*Logboek dataverwerkingen*] and the FSC Transaction ID used in [*FSC-Logging*], this enables full traceability across complex multi-organizational processing chains.

See the sequence diagram below for an example of such a flow.

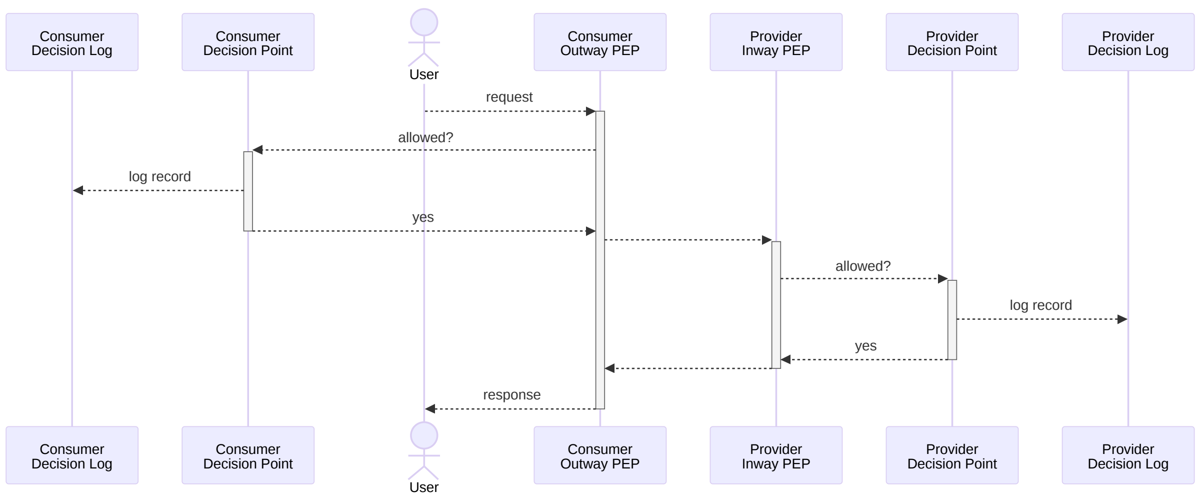

Figure 2 Decision logging in federated context

§ 2.2 Components

The standard EAM architecture has the following conceptual components. These can be deployed as standalone applications, combined in various configurations, or even implemented within a single monolithic application.

§ 2.2.1 Authorization Decision Log

The Authorization Decision Log contains all information that was used in the authorization decision. Using the Authorization Decision Log it *SHOULD* be possible to accurately recreate environmental factors that affected historical authorization decisions.

§ 2.2.2 Policy Enforcement Point (PEP)

A Policy Enforcement Point (PEP) intercepts a user's request, sends it to the Policy Decision Point (PDP) for evaluation, and then enforces the resulting "permit" or "deny" decision. A PEP is typically implemented as an API gateway or as a component within the application itself.

§ 2.2.3 Policy Administration Point (PAP)

A Policy Administration Point (PAP) is where access policies are authored, managed, and stored. It is responsible for distributing current policies to the PDP and archiving previous versions for traceability. This can be a dedicated commercial or open-source tool or a version control system like a Git repository.

§ 2.2.4 Policy Information Point (PIP)

A Policy Information Point (PIP) enriches access requests with additional attributes needed to make a decision. For example, it might retrieve a user's role or the sensitivity level of a data record from an external source. PIPs are often integrated directly into the PDP.

§ 2.2.5 Policy Decision Point (PDP)

A Policy Decision Point (PDP) evaluates incoming requests from the PEP against the relevant policies (from the PAP) and contextual data (from the PIP) to make a "permit" or "deny" decision. The PDP is often a separate application or sidecar container.

> NOTE: EAM components within a monolithic application
>
> It is important to keep in mind that these are architectural components. They can also be physically implemented in a single application. In such a case the authorization interceptor can be considered the PEP; the authorization handler, the PDP; the services it invokes, the PIP; and the Git repository of the application itself, the PAP.

§ 2.3 Scope

This section delineates the scope of the standard.

§ 2.3.1 No specification for the management of logs

The specification defines an interface for persisting log entries. This is the component that *MUST* be consistent across organizations to ensure interoperability.

The management of a log, however, is left to the discretion of individual implementations. Consequently, the specification does NOT define behavior or interfaces for:

- deleting or modifying log entries
- managing access to the log
- ensuring long-term accessibility
- handling archival and retention periods
- ensuring integrity and non-repudiation
- maintaining time-synchronization

See 5. Information Management and Compliance for an overview of various aspects which *MAY* be required for legal and regulatory compliance.

§ 2.4 Flows

§ 2.4.1 Writing a log record after an authorization decision

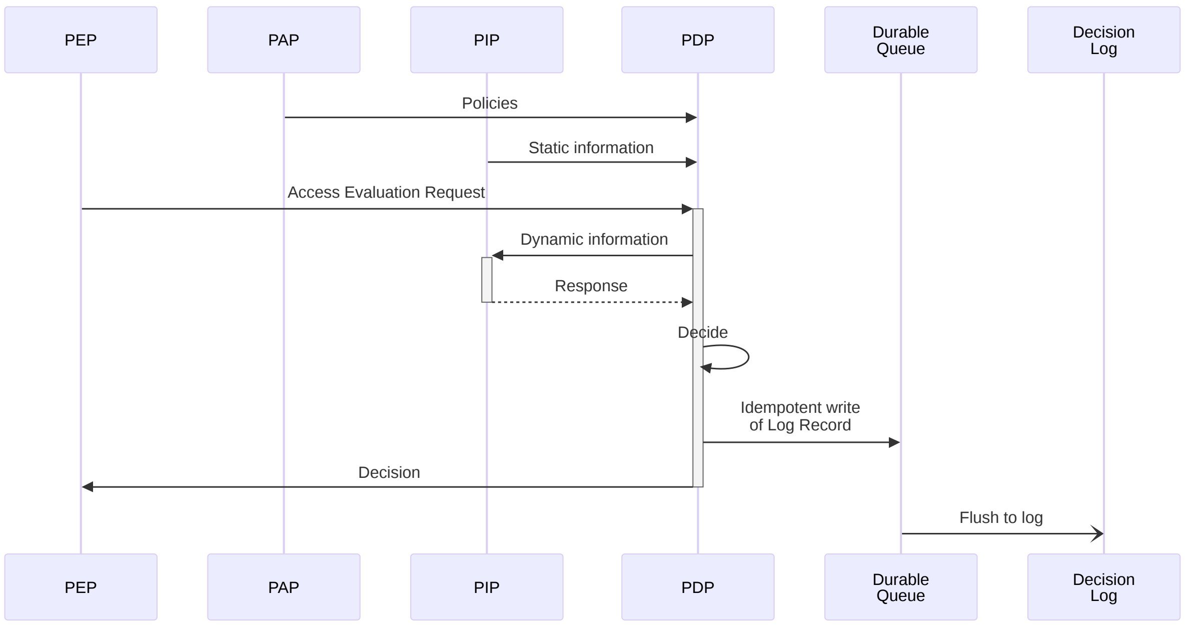

Figure 3 Writing a log record after an authorization decision

To provide accountability for historical authorization decisions it must be possible to recreate the information and environment that affected the decision. The PDP provides the information required for this to the Authorization Decision Log in the form of a Log Record, as defined in the specification below.

The PDP *SHOULD* ensure that a Log Record has been persisted to durable storage before providing the Policy Enforcement Point with the decision. The PDP then flushes this durable storage to the Authorization Decision Log, ensuring that the log record has been persisted and can be used to provide accountability when needed.

§ 3. Specifications

This section provides the specification for the protocols and interfaces to be used and the expected behavior of the components.

§ 3.1 Protocols

The protocols used between the engine and the log are not prescribed in this standard.

> NOTE
> Note, by "the protocols" we mean the method of delivering messages between components. This standard does describe the interfaces of the messages themselves. The components *MUST* comply with the interfaces to ensure interoperability between component functionalities. The standard does not prescribe how that information is passed between components, as this depends on the technical/architectural choices made by software developers. This provides the freedom to add the standard to almost any software solution.

It is *RECOMMENDED* to use the OpenTelemetry Protocol (OTLP) for the interaction between the Application and the log.

> NOTE
> OpenTelemetry is a standard and open-source framework for managing, generating, collecting, and exporting telemetry data. Using this open standard can prevent vendor-specific integrations. OpenTelemetry is a CNCF incubating project.

§ 3.2 Behavior

Each decision log entry *SHOULD* be persisted to durable storage before the PDP provides a decision response to the PEP. If persistence of log entries is not confirmed, historical decisions may end up not being logged.

The log *MUST* enforce TLS on connections, in accordance with the standard practice established within the organization.

See 5. Information Management and Compliance for an overview of additional behavior that *MAY* be required for legal and regulatory compliance.

§ 3.3 Interface

The interface *MUST* have fields that can identify the request.

It is *RECOMMENDED* to use [*trace-context*] to identify requests by implementing the following fields:

Field	Type	Description
`trace_id`	16 byte	Unique identifier of trace that follows data processing
`span_id`	8 byte	Unique identifier of span within the data processing

When requests incorporate [*FSC-Logging*], the following field *SHOULD* be implemented:

Field	Type	Description
`transaction_id`	string	Unique identifier of FSC transaction id of this request

The interface *MUST* implement the following fields describing the request and its response:

Field	Type	Description
`timestamp`	timestamp	The exact point in time when the authorization decision was made
`type`	string	The type of request as defined in the [*AuthZen*] standard
`request`	object	Input for the decision in [*AuthZen*] format
`response`	object	Output of the decision in [*AuthZen*] format

The interface *MAY* implement the following fields:

Field	Type	Description
`policies`	object	Policy sources that affected the decision
`information`	object	Information sources used in the decision
`configuration`	object	Policy Decision Point configuration and metadata

Implementations *MAY* include additional fields. Such fields *MUST NOT* alter the semantics of the defined fields and *SHOULD* be ignored by recipients that do not recognize them.

§ 3.3.1 Identifiers

Each log record should be uniquely identified. Using these identifiers the log entry can be related to other logs and vice-versa.

The specification supports a number of different identifiers: a W3C Trace Context to integrate with [*logboek dataverwerkingen*], a transaction id to integrate with [*FSC-Logging*], and a generic ID for when neither of the other identifiers are available.

§ *3.3.1.1 W3C Trace Context*

The `trace_id` field is a 16-byte unique identifier that represents the trace context for the data processing operation. This field *SHOULD* follow the W3C Trace Context specification. The trace ID enables correlation of the authorization decision with related operations.

The `span_id` field is an 8-byte unique identifier that represents the specific span within the trace context. This field *SHOULD* follow the W3C Trace Context specification. The span ID identifies the particular operation or step within the data processing flow where the authorization decision was made.

§ *3.3.1.2 FSC Transaction ID*

The `transaction_id` field is a string value that represents the FSC transaction to which this request belongs. If a W3C trace context is available it should also be included.

> NOTE
> The Authorization Decision Log and the FSC Log have the same granularity and can thus be combined into a single physical log. This specification ensures that no fields are defined that conflict with those defined in [*FSC-Logging*].

§ *3.3.1.3 Generic identifier*

If none of the other identifiers can be supported, a generic fallback identifier can be included in the `id` field.

This can be any value, simple or complex, which can contain any kind of request identifier.

§ **3.3.2 Request and response**

The minimal information required for an entry in the log consists of a request for a decision and a response with the evaluated decision.

This section describes the key fields required for logging this request/response cycle.

§ *3.3.2.1 Timestamp*

The `timestamp` field represents the exact point in time when the authorization decision was made. The timestamp *SHOULD* be in [*RFC3339*] format to ensure consistent interpretation across different systems and regions.

§ *3.3.2.2 Type*

The `type` field represents the type of request that was made. This value identifies the AuthZEN endpoint that was invoked.

Its value *MUST* be a string containing the key value of the relevant endpoint as defined in "Endpoint Parameters" of the "Policy Decision Point Metadata" as defined in [*AuthZEN*] with the `_endpoint` suffix omitted.

For example, a request to the URL defined by the `search_subject_endpoint` in the PDP metadata would have the `type` of `search_subject`.

§ *3.3.2.3 Request*

The `request` field is an object that represents the input to the decision. This field *MUST* be in [*AuthZen*] format as defined for the given request type.

Portions of the request *MAY* be omitted for privacy reasons. If information is omitted, this omission *SHOULD* be documented or indicated in the log record. If the omitted information was used by the Policy Decision Point, then full accountability can no longer be provided.

§ *3.3.2.4 Response*

The `response` field is an object that represents the output of the decision. This field *MUST* be in [*AuthZen*] format as defined for the given request type.

Portions of the response *MAY* be omitted for privacy reasons. If information is omitted, this omission *SHOULD* be documented or indicated in the log record. If information that was used by the Policy Enforcement Point is omitted then full accountability can no longer be provided.

§ *3.3.2.5 Examples (non-normative)*

In the following example a manager called Alice attempts to approve a holiday request for a team member called Bob, but the request is denied because she does not have signing authority.

Her browser submits the following request to the API:

EXAMPLE 1: HTTP request for holiday approval

```
POST /users/bob/holiday-requests/446epbc8y7 HTTP/1.1
Host: hr.example.com
Authorization: Bearer <alice-auth-token>
traceparent: 00-28dbeec32e77635cc19bc3204ec56c41-dec5220770f8f4f4-01

{"action":"approve"}
```

The application, acting as the PEP, then submits the following HTTP request to the PDP:

EXAMPLE 2: HTTP request from the PEP to the PDP

```
POST /access/v1/evaluation HTTP/1.1
Host: pdp.example.com
Content-Type: application/json
Authorization: Bearer <pep-auth-token>
traceparent: 00-28dbeec32e77635cc19bc3204ec56c41-893e1b2ac52d712f-01

{
    "subject": {
        "type": "user",
        "id": "alice"
    },
    "action": {
        "name": "approve"
    },
    "resource": {
        "type": "holiday-request",
        "id": "446epbc8y7",
        "properties": {
            "employee": "bob"
        }
    },
    "context": {
        "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-dec5220770
    }
}
```

The PDP then determines that Alice can't sign on behalf of the company and thus cannot approve the holiday request. It returns the following response:

EXAMPLE 3: Response from the PDP to the PEP

```
{
    "decision": false,
    "context": {
        "reason": {
            "48": "No signing authority"
        }
    }
}
```

The following JSON object contains a non-normative example of a log record describing this example:

EXAMPLE 4: Log record of denied holiday approval

```
{
    "timestamp": "2025-09-07T10:14:18Z",
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "893e1b2ac52d712f",
    "type": "evaluation",
    "request": {
        "subject": {
            "type": "user",
            "id": "alice"
        },
        "action": {
            "name": "approve"
        },
        "resource": {
            "type": "holiday-request",
            "id": "446epbc8y7",
            "properties": {
                "employee": "bob"
            }
        },
        "context": {
            "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-dec522
        }
    },
    "response": {
        "decision": false,
        "context": {
            "reason": {
                "48": "No signing authority"
            }
        }
    }
}
```

§ 3.3.3 Policy Sources

The `policies` field represents a versioned reference to the policies that the PDP used to evaluate the request. In a PxP architecture, this represents the information that would come from the Policy Administration Point (PAP).

A PDP can have one or more sources of policies which can be individually versioned. To accommodate that the `policies` field is an object in which each key identifies a specific, versioned, policy source.

All policy sources that have affected the decision *MUST* be included. The value associated with each key refers to a unique version of the policy source. The information in this field *MUST* be sufficient to retrieve all policies from the policy sources that were used in the authorization decision.

Non-normative examples include:

- Timestamp
- Unique identifier
- Semantic version
- Git hash

§ *3.3.3.1 Examples (non-normative)*

We can extend the example of the holiday-approval request by adding a reference to a Git repository in which current HR approval policies are documented. In the example below the git hash of the version currently deployed together with the PDP is `6266d07750c44b4c9b05d0801b752c0ef884e4f6`.

EXAMPLE 5: Git-versioned policy source

```
{
    "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6"
}
```

More complex references can be achieved by using an object as the version identifier. If, for example, the HR application takes part in a federation with predefined policies for different maturity levels. The following non-normative example shows how those policies can be referenced using a semantic version combined with a filter for policies relevant to the current maturity level.

EXAMPLE 6: Complex policy source reference

```
{
    "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6",
    "federation": {
        "version": "2.7.1",
        "filter": "maturity_level <= 3"
    }
}
```

The following JSON object contains a non-normative example of a log record describing this example:

EXAMPLE 7: Log record of denied holiday approval

```json
{
    "timestamp": "2025-09-07T10:14:18Z",
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "893e1b2ac52d712f",
    "type": "evaluation",
    "request": {
        "subject": {
            "type": "user",
            "id": "alice"
        },
        "action": {
            "name": "approve"
        },
        "resource": {
            "type": "holiday-request",
            "id": "446epbc8y7",
            "properties": {
                "employee": "bob"
            }
        },
        "context": {
            "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-dec522
        }
    },
    "response": {
        "decision": false,
        "context": {
            "reason": {
                "48": "No signing authority"
            }
        }
    },
    "policies": {
        "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6",
        "federation": {
            "version": "2.7.1",
            "filter": "maturity_level <= 3"
        }
    }
}
```

§ 3.3.4 Information Sources

The `information` field represents all the supporting information used in the evaluation of the access decision. In a PxP architecture, this field represents the information that would come from Policy Information Points (PIPs).

It is an object in which each key identifies an information source. All information sources that have affected the decision *SHOULD* be included. The value of this field *SHOULD* either contain the information that was used in the access decision or be sufficient to retrieve the information.

§ *3.3.4.1 Examples (non-normative)*

In the example of the holiday approval, the ability to sign is accessed through the `can_sign` field of the user. The Policy Information Point (PIP) called `can-sign-api` requests this via an API from the HR application using the request below:

EXAMPLE 8: PIP's request to the Managers API

```
GET /users/alice?fields=can_sign HTTP/1.1
Host: hr.example.com
traceparent: 00-28dbeec32e77635cc19bc3204ec56c41-836ff5286112f460-01
```

And the API returns the following response.

EXAMPLE 9: Managers API response to the PIP's request

```
{
    "can_sign": false
}
```

The following JSON object contains a non-normative example of a log record describing this example.

EXAMPLE 10: Log record of denied holiday approval

```
{
    "timestamp": "2025-09-07T10:14:18Z",
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "893e1b2ac52d712f",
    "type": "evaluation",
    "request": {
        "subject": {
            "type": "user",
            "id": "alice"
        },
        "action": {
            "name": "approve"
        },
        "resource": {
            "type": "holiday-request",
            "id": "446epbc8y7",
            "properties": {
                "employee": "bob"
            }
        },
        "context": {
            "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-dec522
        }
    },
    "response": {
        "decision": false,
        "context": {
            "reason": {
                "48": "No signing authority"
            }
        }
    },
    "policies": {
        "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6",
        "federation": {
            "version": "2.7.1",
            "filter": "maturity_level <= 3"
        }
    },
    "information": {
        "can-sign-api": {
            "can_sign": false
            }
        }
    }
```

NOTE: Source references to reduce data duplication

In this example the entire response is stored in the log record as it is a small response without sensitive data. In most cases it is recommended to use a reference to the data instead. See 3.4 Sources and referencing for more information.

§ 3.3.5 Configuration Sources

The `configuration` field represents the information required to recreate the software configuration that evaluated the original decision. In a PxP architecture, this primarily represents the configuration of the Policy Decision Point (PDP), but *MAY* also include configuration of Policy Information Points (PIPs) and Policy Administration Points (PAPs).

It is an object in which each key identifies a configuration source. All configuration sources that have affected the decision *SHOULD* be included. The value of this field *SHOULD* either contain the configuration that was used in the access decision or be sufficient to retrieve the configuration.

Non-normative examples include:

- Configuration of the policy engine (PDP)
- Version of the policy language
- Identifier or hostname of the PDP in case multiple PDPs are used
- Configuration of Policy Information Points, such as API endpoints.
- Git hash of an IaaS definition, such as a Terraform repository.

§ *3.3.5.1 Examples (non-normative)*

In the example below we extend the holiday approval request example by describing the version of the language used by the PDP and the configuration of the `can-sign-api` PIP.

EXAMPLE 11: Log record of denied holiday approval

```
{
    "timestamp": "2025-09-07T10:14:18Z",
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "893e1b2ac52d712f",
    "type": "evaluation",
    "request": {
        "subject": {
            "type": "user",
            "id": "alice"
        },
        "action": {
            "name": "approve"
        },
        "resource": {
            "type": "holiday-request",
            "id": "446epbc8y7",
            "properties": {
                "employee": "bob"
            }
        },
        "context": {
            "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-dec522
        }
    },
    "response": {
        "decision": false,
        "context": {
            "reason": {
                "48": "No signing authority"
            }
        }
    },
    "policies": {
        "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6",
        "federation": {
            "version": "2.7.1",
            "filter": "maturity_level <= 3"
        }
    },
    "information": {
        "can-sign-api": {
            "can_sign": false
        }
    },
    "configuration": {
        "opa_version": "1.10.0",
        "can-sign-api": "https://hr.example.com/users/{subject.id}?fie
    }
}
```

NOTE: Source references to reduce data duplication

In this example the configuration is stored in the log record itself. To reduce data duplication it is generally recommended to use a reference to the configuration instead. See 3.4 Sources and referencing for more information.

§ 3.4 Sources and referencing

Policy, information and configuration sources *MAY* be included in the log directly.

This is generally undesirable however as it introduces duplication, increases the size of the log and increase security requirements for the log by including sensitive data.

To address this we describe several methods of referencing sources from the log below.

§ 3.4.1 Versioned sources

Some information sources offer the ability to 'time-travel' by providing a version at which to query. In such cases, the data itself may be omitted and the version can be stored instead.

The version identifier can be a simple value, such as a string or number, or a complex object, such as an array or object containing multiple version identifiers.

The following non-normative example shows a reference to a specific semantic version of a policy source.

EXAMPLE 12: Policy source reference using semantic versioning

```
{
    "traffic-policy": "gmb-2025-94604@1.1"
}
```

In a complex case, such as limiting requests for open data per IP per minute across a large number of servers, the version could also consist of an array of partition offsets in a Kafka stream of HTTP request.

EXAMPLE 13: Complex versioned information sources using Kafka partition offsets

```
{
    "nginx-requests": [ 8376912, 8368118, 8377785, 8386285, 8383526 ]
}
```

§ 3.4.2 Temporal sources

In case the source of information offers the ability to 'time-travel' by providing a timestamp at which to query, then the data itself may be omitted.

It is *RECOMMENDED* to use the timestamp defined in the `time` field in the `context` of the `request` as the base time. In that case the information source *MAY* be omitted fully.

If a different timestamp is used, then it *SHOULD* be included in [*RFC3339*] format.

NOTE: Inter-system clock inconsistencies

When system clocks are not aligned properly, a system may be asked to provide information for a timestamp that lies in the future. This can be mitigated by requesting the policies of a few seconds or minutes ago at the expense of reducing the speed with which policy changes can be deployed.

NOTE: Usage of REST API Design Rules

In the context of REST APIs developed by the Dutch government the Temporal extension of the [*ADR*] can be used for this purpose.

§ 3.4.3 Logged sources

For information sources that are logged in an external log, a request identifier is needed to look up the corresponding request in the external log.

It is *RECOMMENDED* to use the W3C Trace Context standard as the request identifier. Such a request *SHOULD* have the same `trace_id` as the request to the PDP, in which case the source reference can consist of only the value of the `span_id`.

It is *RECOMMENDED* to log requests in the [*WARC*] format as it includes all request and response headers that may be used in the authorization decision.

The following non-normative example shows a log record for a request to find all subjects capable of approving a holiday request:

EXAMPLE 14: Log record of a search request for managers with approval rights

```
{
    "timestamp": "2025-09-07T10:15:36Z",
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "17c59821784ee492",
    "type": "search_subject",
    "request": {
        "subject": {
            "type": "user"
        },
        "action": {
            "name": "approve"
        },
        "resource": {
            "type": "holiday-request",
            "id": "446epbc8y7",
            "properties": {
                "employee": "bob"
            }
        }
    },
    "response": {
        "results": [
            {
                "type": "user",
                "id": "carol"
            },
            {
                "type": "user",
                "id": "dan"
            }
        ]
    },
    "policies": {
        "git": "e4c15a063048367da367d5588d703b5e4a6b760e"
    },
    "information": {
        "managers-api": "45deb36022f53afa"
    }
}
```

Which would result in the following WARC entries logging the REST API call to the HR system:

EXAMPLE 15: WARC entries for REST API call to HR system

```
WARC/1.1
WARC-Type: request
WARC-Date: 2025-09-07T10:15:31Z
WARC-Record-ID: <urn:uuid:48bafbce-8d2a-45c1-9d7a-1a851c36e1c8>
Content-Type: application/http; msgtype=request
Content-Length: 142

GET /users/bob/managers?fields=can_sign HTTP/1.1
Host: hr.example.com
traceparent: 00-28dbeec32e77635cc19bc3204ec56c41-45deb36022f53afa-01

WARC/1.1
WARC-Type: response
WARC-Date: 2025-09-07T10:15:32Z
WARC-Record-ID: <urn:uuid:4a381180-21a7-4712-8706-5b321c17e3f8>
WARC-Concurrent-To: <urn:uuid:48bafbce-8d2a-45c1-9d7a-1a851c36e1c8>
Content-Type: application/http; msgtype=response
Content-Length: 175

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 107

{
    "alice": {
        "can_sign": false
    },
    "carol": {
        "can_sign": true
    },
    "dan": {
        "can_sign": true
    }
}
```

§ 4. Data Verifiability and Level of Detail

The ability to provide accountability depends on the log's level of detail. A balance must be struck between capturing enough information to accurately replay historical decisions and practical challenges like data duplication and scalability. The appropriate level of detail depends on the organization's specific context and legal requirements, as the highest level is not always necessary.

To ensure historical accuracy while minimizing data storage, referencing external information (e.g., via a timestamp or version number) is preferred over storing copies. This approach keeps logs lean but is contingent on the ability of source systems to provide versioned historical data.

For full replayability, the log must also identify the exact version and configuration of the policy engine that evaluated the decision; however, providing reliable versioning for the engine may not always be feasible, depending on the infrastructure.

§ 4.1 Definition of Levels

We have identified four levels of detail, in order from least to most detail. Each level builds on the information from the previous level.

§ 4.1.1 Level 1: Decision Request/Response

At the most basic level only the decision request and the decision response are logged.

NOTE: Engine boundaries

The decision request and response *MAY* contain all information required for an audit log, as described by [*ISO/IEC 27002:2022*] and [*BIO2*]. If that is the case, and all auditable actions are decided on by the PDP, the Authorization Decision Log *MAY* be used as an audit log.

At this level of detail log requests contain the following keys, as defined in 3. Specifications:

Field	Required	Reference
`trace_id`	optional	3.3.1.1 W3C Trace Context
`span_id`	optional	3.3.1.1 W3C Trace Context
`timestamp`	required	3.3.2.1 Timestamp
`type`	required	3.3.2.2 Type
`request`	required	3.3.2.3 Request
`response`	required	3.3.2.4 Response

§ 4.1.2 Level 2: Decision and Policies

In addition to the request and response, the exact version of the policies that were used to evaluate the request can be programmatically retrieved.

At this level of detail log requests contain the following keys, as defined in 3. Specifications:

Field	Required	Reference
`trace_id`	optional	3.3.1.1 W3C Trace Context
`span_id`	optional	3.3.1.1 W3C Trace Context
`timestamp`	required	3.3.2.1 Timestamp
`type`	required	3.3.2.2 Type
`request`	required	3.3.2.3 Request
`response`	required	3.3.2.4 Response
`policies`	required	3.3.3 Policy Sources

§ 4.1.3 Level 3: All Information Sources

All information used in the evaluation can be programmatically retrieved. This allows full replayability, assuming the engine (PDP) behaves identically or can be manually recreated in the correct state, which is generally achievable.

At this level of detail log requests contain the following keys, as defined in 3. Specifications:

Field	Required	Reference
`trace_id`	optional	3.3.1.1 W3C Trace Context
`span_id`	optional	3.3.1.1 W3C Trace Context
`timestamp`	required	3.3.2.1 Timestamp
`type`	required	3.3.2.2 Type
`request`	required	3.3.2.3 Request
`response`	required	3.3.2.4 Response
`policies`	required	3.3.3 Policy Sources
`information`	required	3.3.4 Information Sources

§ 4.1.4 Level 4: Full Environment

In addition to all information used in the evaluation, the environment and configuration of the system that evaluates the decision can also be accurately recreated. This provides full, guaranteed replayability and maximum accountability.

NOTE: System boundaries

The configuration of all components that influence the decision should be included. While this may be limited to the configuration of the PDP, it often also requires configuration of the PIP and sometimes the PAP as well.

At this level of detail log requests contain the following keys, as defined in 3. Specifications:

Field	Required	Reference
`trace_id`	optional	3.3.1.1 W3C Trace Context
`span_id`	optional	3.3.1.1 W3C Trace Context
`timestamp`	required	3.3.2.1 Timestamp
`type`	required	3.3.2.2 Type
`request`	required	3.3.2.3 Request
`response`	required	3.3.2.4 Response
`policies`	required	3.3.3 Policy Sources
`information`	required	3.3.4 Information Sources
`configuration`	required	3.3.5 Configuration Sources

§ 4.2 Implications of levels

The higher the level of detail, the more useful the log is for determining the context of an authorization decision. On the other hand, higher levels of detail also introduce challenges around scalability, technical feasibility, and security.

Conversely, the lowest level of detail may not be sufficient to provide effective accountability. This depends on the data processing which is being authorized and legal requirements for it.

For that reason it's important to decide for different use cases which level of detail is required and appropriate. Aiming for the highest level of detail for all authorization decisions is thus not necessary.

§ 5. Information Management and Compliance

This, non-authoritative, section addresses the governance, security, and privacy-compliance aspects of managing an Authorization Decision Log (ADL).

Conformance to this standard does not, by itself, guarantee legal or regulatory compliance. The implementing organization is solely responsible for ensuring its implementation adheres to all applicable frameworks, such as the General Data Protection Regulation (GDPR / AVG) and relevant security baselines, such as [*ISO/IEC 27001:2022*], [*ISO/IEC 27002:2022*], and [*BIO2*]. Each organization is responsible for its own Authorization Decision Log. There is no central log, although logs of several organizations can be aggregated if desired.

The following sections list aspects that should be taken into consideration when creating a compliant and secure logging solution.

§ 5.1 Legal and Privacy Compliance

Logging authorization decisions creates a new processing of data, which must be compliant with privacy regulations if personal data is involved.

§ 5.1.1 Purpose Limitation

When collecting personal data, the specific purposes for logging (e.g., operational auditing, forensic analysis, citizen accountability) must be defined and documented in a formal policy before implementation. Data collection must be limited to these defined purposes.

§ 5.1.2 Data Minimization

A core principle is to avoid storing unnecessary information, especially sensitive data. The goal is to make the log precise, compact, and manageable. Instead of duplicating large amounts of (personal) data, prefer storing only the data used in the decision or even just a reference that allows the state at the time of the decision to be recreated.

A logging policy must be implemented to manage what is logged based on risk. When logging sensitive data for high-risk use cases, this should be explicitly documented and approved.

Implementers should consider pseudonymization and anonymization for personal data. For example, personal identifiers can be hashed or aliased and location data can be randomized.

When aggregate statistics, such as decisions per type per time period, are sufficient, implementers should consider using aggregation as a method of data minimization and anonymization.

§ 5.1.3 Data Retention

A data retention policy must be defined and enforced. Retention periods must be based on the defined purposes and legal obligations. These may differ for different types of log entries.

As a general guideline, operational logs (for debugging, support) should be retained for a short period (months), while forensic/audit logs may be retained for longer periods (years).

Logs that have exceeded their defined retention period must be automatically and securely purged.

§ 5.1.4 Transparency and Subject Rights

If logs contain personal data, they must be able to comply with data subject access requests.

The log's structure, purpose, and retention must be documented in the organization's Register of Processing Activities.

A Data Protection Impact Assessment (DPIA) must be conducted for any implementation that involves large-scale or high-risk processing of personal data.

§ 5.2 Access Control

Access to log data should be restricted based on the principle of least privilege.

It is essential to define clear policies for authorizing access to the Authorization Decision Log or parts thereof. These decisions to provide or deny access to the log should also be included in the Authorization Decision Log.

Common and important usage policies include:

- **(Forensic) Audits**: The log is a critical tool for auditing and forensic analysis after a security incident or data breach. It can help determine what actions were permitted at a specific time and on what basis, even if that permission was technically correct but improper in hindsight.
- **Observability in Trust Frameworks**: The log offers a structured method for data users to provide insight into their data usage to data providers when required, as may be required in trust frameworks. It thus offers an implementation standard for "Observability services" as defined for data spaces under the EU Data Act.
- **Debugging and Support**: The log can be a useful tool for determining why the authorization is not working as expected. It can also contain highly sensitive data, however, so it's essential to carefully define if, and under which conditions, the Authorization Decision Log can be used for this purpose.

§ 5.3 Security and Integrity

The log must be protected against unauthorized access, modification, and deletion.

Key concerns for ensuring security and integrity include:

- **Transport Security**: It's recommended to use mTLS (Mutual Transport Layer Security) for network connections that transport log data to ensure authenticated and encrypted transport.
- **Encryption at Rest**: All log data should be encrypted at rest. It's recommended to manage this using a Key Management System (KMS).
- **Data Integrity**: The log should be configured as append-only storage (WORM) to prevent undetected modification or deletion. Mechanisms such as a cryptographic hash chains and periodic cryptographic sealing can be used to ensure integrity and non-repudiation of logs.
- **Time Synchronization**: All systems involved in generating and storing logs should be synchronized to a trusted Network Time Protocol (NTP) source to ensure a reliable and accurate timeline of events.
- **Ingestion**: The logging endpoint should be implemented as idempotent writes to an asynchronous, buffered service (e.g., using a durable queue) to mitigate latency and availability risks in the event of log-ingest failures.

§ 6. List of Figures

§ A. Index

§ A.1 Terms defined by this specification

§ A.2 Terms defined by reference

§ B. References

§ B.1 Normative references

[AuthZen]
Authorization API 1.0 – draft 03. O. Gazitt; D. Brossard; A. Tulshibagwale. URL: https://openid.net/specs/authorization-api-1_0-03.html

[FSC-Core]
FSC - Core. Eelco Hotting; Ronald Koster; Henk van Maanen; Niels Dequeker; Edward van Gelderen; Pim Gaemers. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/fsc/core/2.0.0/

[FSC-Logging]
FSC - Logging. Eelco Hotting; Ronald Koster; Henk van Maanen; Niels Dequeker; Edward van Gelderen; Pim Gaemers. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/fsc/logging/1.1.0/

[logboek dataverwerkingen]
Logboek Dataverwerkingen. Logius. URL: https://logius-standaarden.github.io/logboek-dataverwerkingen/

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc2119

[RFC3339]
Date and Time on the Internet: Timestamps. G. Klyne; C. Newman. IETF. July 2002. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc3339

[RFC8174]
Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174

[trace-context]
Trace Context. Sergey Kanzhelev; Morgan McLean; Alois Reitbauer; Bogdan Drutu; Nik Molnar; Yuri Shkuro. W3C. 23 November 2021. W3C Recommendation. URL: https://www.w3.org/TR/trace-context-1/

[WARC]
Information and documentation — WARC file format. International Organization for Standardization (ISO). August 2017. Published. URL: https://www.iso.org/standard/68004.html

§ B.2 Informative references

[ADR]
API Design Rules. Jasper Roes; Joost Farla. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/api/adr/2.1

[BIO2]
Baseline Informatiebeveiliging Overheid 2. 24 september 2025. URL: https://www.bio-overheid.nl/media/cs5ctudu/20250924-baseline-informatiebeveiliging-overheid-2-bio2-v12-def.pdf?csf=1&web=1&e=9JoWOT

[ISO/IEC 27001:2022]
Information security, cybersecurity and privacy protection — Information security management systems — Requirements. 2022-10. URL: https://www.iso.org/standard/27001.html

[ISO/IEC 27002:2022]
Information security, cybersecurity and privacy protection — Information security controls. 2022-02. URL: https://www.iso.org/standard/75652.html

[NIST.SP.800-162]
Guide to Attribute Based Access Control (ABAC) Definition and Considerations. Chung Tong Hu; David F. Ferraiolo; David R. Kuhn. February 25, 2019. URL: https://www.nist.gov/publications/guide-attribute-based-access-control-abac-definition-and-considerations-1

↥