LOGPOINT

Integrations

Universal REST API

V3.4.0

CONTENTS

CHAPTER

ONE

UNIVERSAL REST API FETCHER

Universal REST API Fetcher provides a generic interface to fetch logs from cloud sources via REST APIs. The cloud sources can have multiple endpoints, and every configured source consumes one device license.

CHAPTER

TWO

INSTALLING UNIVERSAL REST API FETCHER

Prerequisite

Logpoint v7.4.0 and later

To install:

1. Download the .pak file from the *Download* section in Release Notes.
2. Go to *Settings >> System Settings* from the navigation bar and click **Applications**.
3. Click **Import**.
4. **Browse** to the downloaded .pak file.
5. Click **Upload**.

After installing Universal REST API Fetcher, you can find it under *Settings >> System Settings >> Plugins*.

CHAPTER

THREE

UNINSTALLING UNIVERSAL REST API FETCHER

To uninstall Universal REST API Fetcher, you must first remove its configurations and also uninstall Duo Security and Cybereason.

To remove Universal REST API Fetcher configurations:

1. Go to *Settings >> Log Sources* from the navigation bar.
2. Click the (⋮) icon of Universal REST API Fetcher and click **Delete**.
3. Click **Delete**.

To uninstall Universal REST API Fetcher, Duo Security and Cybereason:

1. Go to *Settings >> System Settings* from the navigation bar and click **Applications**.
2. Click the **Uninstall** (🗑) icon in **Actions** of Universal REST API Fetcher, Duo Security and Cybereason.

CHAPTER

FOUR

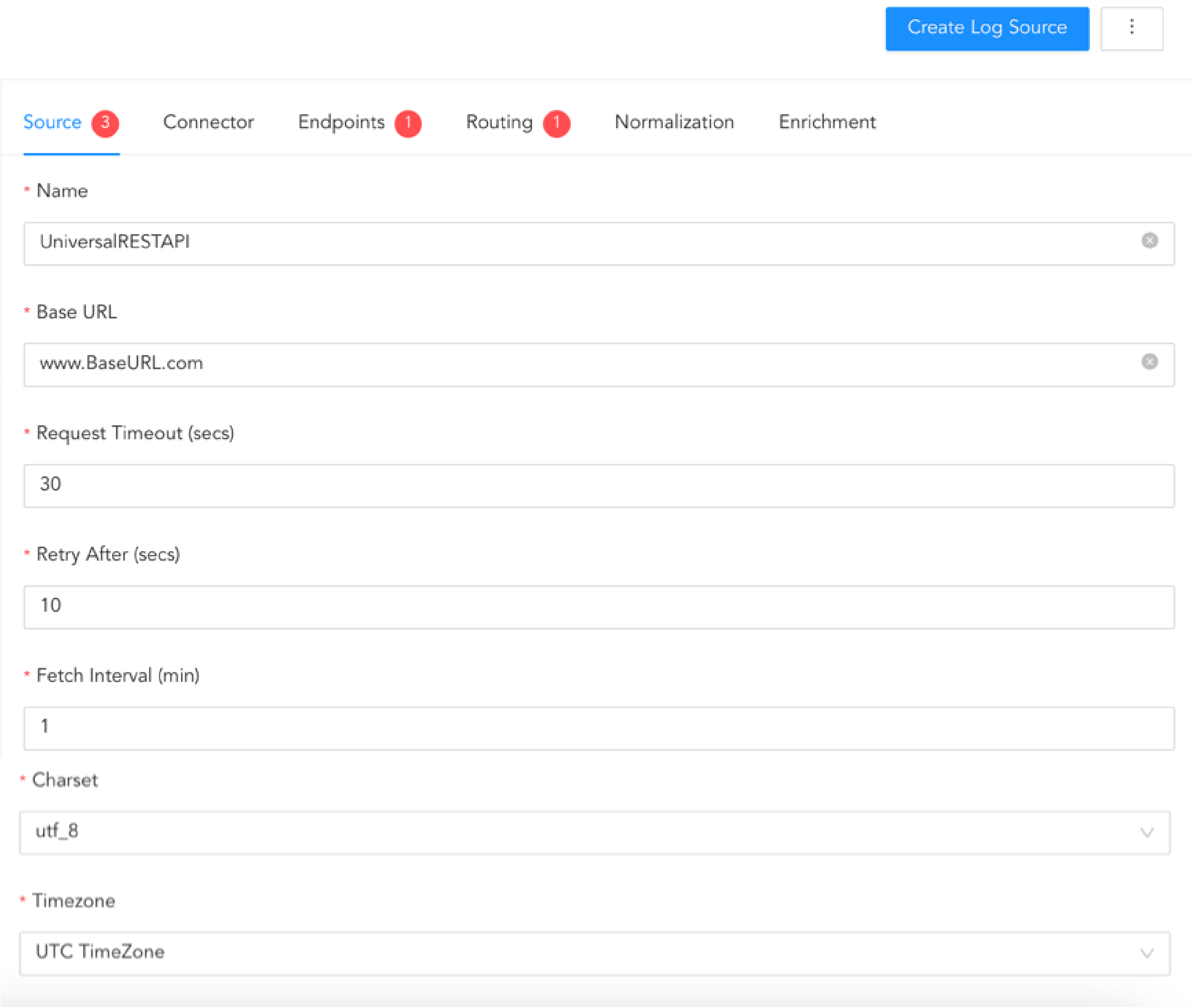
CONFIGURING UNIVERSAL REST API FETCHER

1. Go to *Settings >> Log Sources* from the navigation bar and click **Add Log Source**.
2. Click **Create New** and select **Universal Rest API**.

4.1 Source

In source, add details about the log source from where the Universal REST API Fetcher fetches logs.

1. Click **Source**.
2. Enter the Log Source's **Name**.
3. In **Base URL**, enter the RESTful API.
4. Enter **Request Timeout (secs)** for the API request.
5. In **Retry After(secs)**, enter the time to wait after an error or timeout.
6. Enter the **Fetch Interval (min)** and select **Charset**.
7. Select the **Timezone** of the log source if its response time is not in UTC. Universal REST API Fetcher automatically sets the time in case of UTC.



* Name

UniversalRESTAPI

* Base URL

www.BaseURL.com

* Request Timeout (secs)

30

* Retry After (secs)

10

* Fetch Interval (min)

1

* Charset

utf_8

* Timezone

UTC TimeZone

Fig. 1: Configuring Source

4.2 Connector

In Connector, you must configure how the Universal REST API Fetcher and the log source communicate with each other.

1. Click **Connector**.
2. Select the **Authorization Type**. For more information on supported Authorization types in URAF, go *here*.

    2.1. Select **No Auth** if no authentication is required.

    2.2. Select **Basic** to use a username and password to authenticate.

    2.2.1. In **Credentials**, enter the **Username** and **Password**.

2.3. Select **OAuth2** to authenticate using OAuth authentication. Enter the following details in **OAUTH 2.0 BASIC INFORMATION**.

2.3.1. Enter the **Token URL** of the server.

2.3.2. Select either **Client Credentials** or **Password Credentials** as the **Grant Type**.

2.3.2.1. If you select **Client Credentials**, enter the OAuth secret password in **Client Secret**.

2.3.2.2. If you select **Password Credentials**, enter the OAuth **Username** and **Password**.

2.3.3. In **Client ID**, enter the OAuth application ID or client ID. If the vendor requires a client secret, enter **Client Secret**.

2.3.4. In **API Key Prefix**, enter the prefix to add to the authorization header before the API Key or Token.

2.3.5. Select whether to send the client credentials as a basic auth header or in the body.

2.3.6. Enter the extra parameters key and its value in **ADDITIONAL BODY FOR OAUTH 2.0**.

2.4. Select the **API Key** to authenticate using an API Key.

2.4.1. In **Secret Key**, enter **API Key**. This API key is used in the authorization header.

2.4.2. In **API Key Prefix**, enter the prefix to add to the authorization header before API Key or Token. This is optional.

2.5. Select **Digest** to authenticate using digest access authentication.

2.5.1. In **Credentials**, enter the **Username** and **Password**.

2.6. Select **Custom** to authenticate using integration that applies custom authentication mechanisms and request handling.

2.6.1. Select a **Product**. Here, you can see the integrations supported by Universal REST API Fetcher, such as Duo Security and Cybereason, that require custom authentication mechanisms and request handling. They must also be installed on Logpoint.

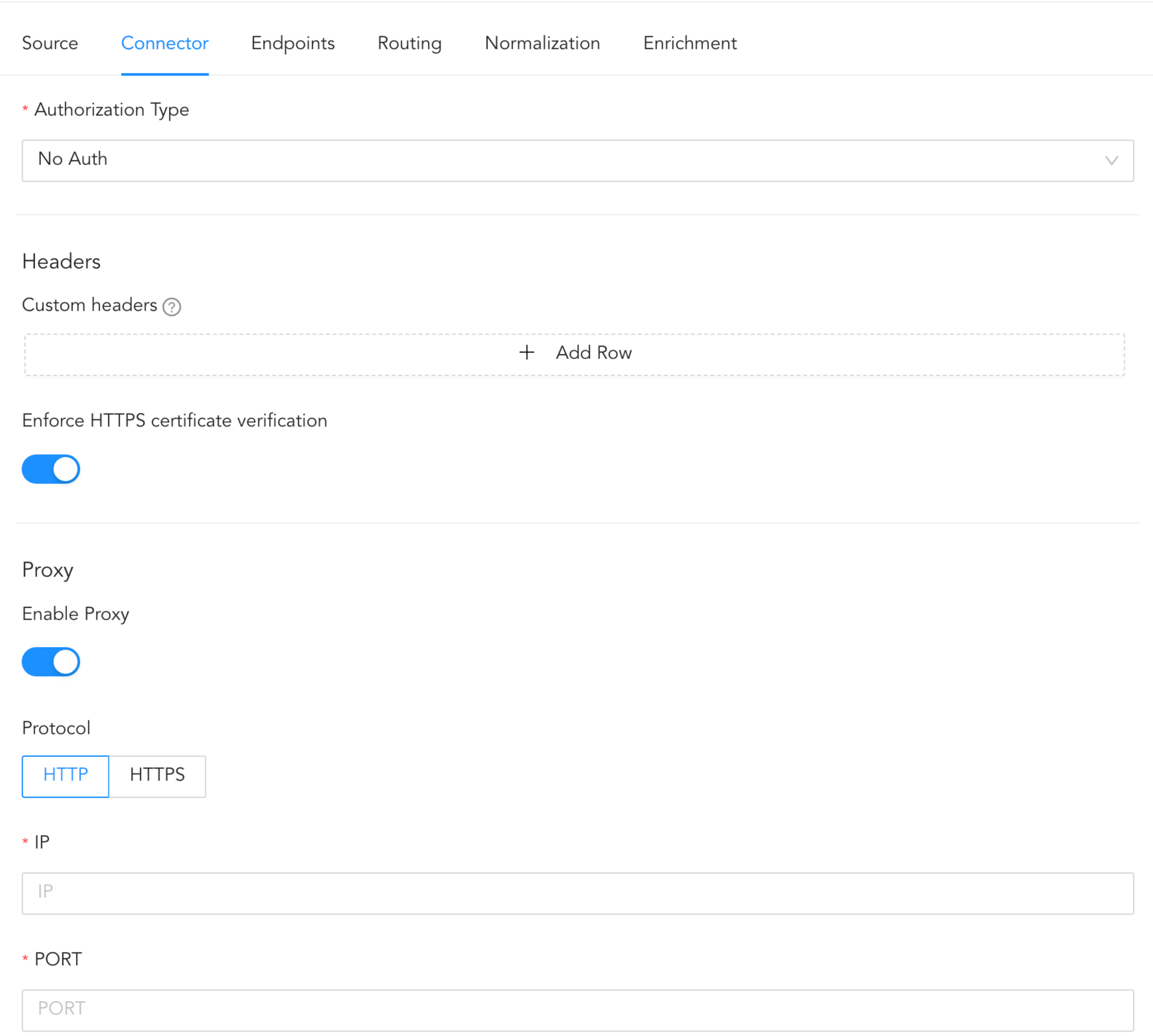
3. Enter the RESTful API custom headers in **Key** and **Value**.

4. Enable **Enforce HTTPS certificate verification** to enable a secure connection.

5. **Enable Proxy** to use a proxy server.

5.1. Select either **HTTP** or **HTTPS** protocol.

5.2. Enter the proxy server **IP** address and the **PORT** number.



Source | Connector | Endpoints | Routing | Normalization | Enrichment

* Authorization Type

No Auth

Headers

Custom headers

+ Add Row

Enforce HTTPS certificate verification

Proxy

Enable Proxy

Protocol

HTTP | HTTPS

* IP

IP

* PORT

PORT

Fig. 2: Configuring Connector

4.3 Endpoints

In enpoints, configure details about the log source endpoints.

1. Click **Endpoints** and **Add Row**.
2. Enter the endpoint's **Name**.
3. Select the HTTP request **Method**.
    - 3.1. If you select **GET**, continue to Step 4.
    - 3.2. If you select **POST**, enter the **Post request body** in JSON format.

You can define the time range for fetching logs in the **Post request body**:

- Use the Jinja keyword *{{start}}* for beginning of the log fetching window.
- Use *{{end}}* for endpoint of the time range.

Example:

```
{
 "filters": [
  {
   "fieldName": "<field>",
   "operator": "<operator>",
   "values": "[value]"
  }
 ],
 "search": "<value>",
 "sortingFieldName": "<field>",
 "sortDirection": "<sort direction>",
 "limit": "<limit>",
 "offset": "<page number>"
}
```

Important: On the first request, *{{start}}* is replaced with the *Initial Fetch* value set in the endpoint. In later requests, it is replaced with the *check sum* value. The *{{end}}* value is always replaced with the timestamp when the request is sent.

Example:

```
{
 "filters": [
  {
   "fieldName": "StartTimestamp",
   "operator": "equals",
   "values": "{{start}}"
  },
  {
   "fieldName": "EndTimestamp",
   "operator": "equals",
   "values": "{{end}}"
  }
 ]
}
```

In this example, *StartTimestamp* and *EndTimestamp* are the beginning and end of the fetch window.

4. Enter the **Endpoint** part of the previously *added* Base URL .

5. Enter a **Description** for the endpoint.
6. Under **Headers**, click **+ Add Row**. Enter the **Key** and **Value** for each custom header.

> **Note:**
>
> - Avoid using common log filtering fields like *start_date* or *end_date* as headers.
> - Do not add *Authorization* as a custom header.

7. In **Query Parameters**, click **+ Add Row**.

   7.1. Enter the **Key** and **Value** as required by the API.

   Example:
   For a query like */api/alerts?$filter=(severity eq 'High') or (severity eq 'Medium')*, enter:

   - **Key**: *$filter*
   - **Value**: *(severity eq 'High') or (severity eq 'Medium')*

   Query parameters are sent as part of the request URL.

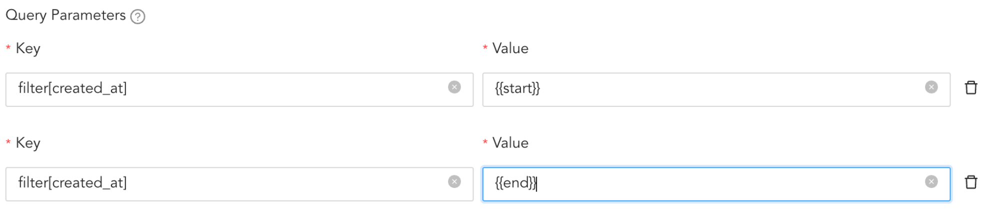
   Important: To define a time range using query parameters:

   - Use *{{start}}* for the start timestamp.
   - Use *{{end}}* for the end timestamp.

   Example:

Key	Value
StartTimestamp	*{{start}}*
EndTimestamp	*{{end}}*

   You can also send multiple query parameters with the same key.
   Example:



Query Parameters

* Key	* Value
filter[created_at]	{{start}}
filter[created_at]	{{end}}

8. In **Increment Value / Check Sum**, enter the path to the field that tracks progress in log fetching.

   Example:

   If the field is *event_date* within an *Events* object, enter *Events.event_date*. This field's value from the most recent log is stored in the CheckSum database. During the next collection cycle, it becomes the *{{start}}* value, ensuring no duplicate logs are collected.

9. Enter the **Response Key**. This is used to locate and extract log records from the API response.

10. Enter the **Custom Date Format** expected in the API response.

Some of them are:

Date Type	Format	Example
UTC	%Y-%m-%dT%H:%M:%SZ	2023-04-27T07:18:52Z
ISO-8601	%Y-%m-%dT%H:%M:%S%z	2023-04-27T07:18:52+0000
RFC 2822	%a, %d %b %Y %H:%M:%S %z	Thu, 27 Apr 2023 07:18:52 +0000
RFC 850	%A, %d-%b-%y %H:%M:%S UTC	Thursday, 27-Apr-23 07:18:52 UTC
RFC 1036	%a, %d %b %y %H:%M:%S %z	Thu, 27 Apr 23 07:18:52 +0000
RFC 1123	%a, %d %b %Y %H:%M:%S %z	Thu, 27 Apr 2023 07:18:52 +0000
RFC 822	%a, %d %b %y %H:%M:%S %z	Thu, 27 Apr 23 07:18:52 +0000
RFC 3339	%Y-%m-%dT%H:%M:%S%z	2023-04-27T07:18:52+00:00
ATOM	%Y-%m-%dT%H:%M:%S%z	2023-04-27T07:18:52+00:00
COOKIE	%A, %d-%b-%Y %H:%M:%S UTC	Thursday, 27-Apr-2023 07:18:52 UTC
RSS	%a, %d %b %Y %H:%M:%S %z	Thu, 27 Apr 2023 07:18:52 +0000
W3C	%Y-%m-%dT%H:%M:%S%z	2023-04-27T07:18:52+00:00
YYYY-DD-MM HH:MM:SS	%Y-%d-%m %H:%M:%S	2023-27-04 07:18:52
YYYY-DD-MM HH:MM:SS am/pm	%Y-%d-%m %I:%M:%S %p	2023-27-04 07:18:52 AM
DD-MM-YYYY HH:MM:SS	%d-%m-%Y %H:%M:%S	27-04-2023 07:18:52
MM-DD-YYYY HH:MM:SS	%m-%d-%Y %H:%M:%S	04-27-2023 07:18:52

11. In **Logs Filtering Parameters**, select the parameters to filter the incoming logs.

    11.1. Select a **Data format**.

    11.1.1. Select **ISO Date** to represent data using the International Standards Organization (ISO) format of "yyyy-MM-dd". Example: 2017-06-10.

    Note: If you select **ISO Date**, then its value must be in the string format in the **Post request body**.

    11.1.2. Select **UNIX Epoch** to represent data using the UNIX epoch time format. It is a system for measuring time as the number of seconds that have elapsed since January 1, 1970, at 00:00:00 UTC (Coordinated Universal Time). Example: 1672475384.

    11.1.3. Select **UNIX Epoch (ms)** to represent data using the UNIX epoch time format with milliseconds precision. It is a system for measuring time as the number of milliseconds that have elapsed since January 1, 1970, at 00:00:00 UTC (Coordinated Universal Time). Example:1672475384000.

    11.1.4. Select **Custom Format** to define your own format for representing the data. The custom format can be created using *Date/Time patterns*.

    11.1.5. Select **Unique ID** to represent data using a unique ID.

    Note: If you select **Unique ID** here, then its value must be in the number format in the **Post request body**.

12. Select an **Initial Fetch** date. Logs are fetched for the first time from this date.

13. For *Link-based pagination*, in **Pagination Key**, enter the URL or URI from the API response that points to the next page of results. This value can come from response body, response header links, or response headers.

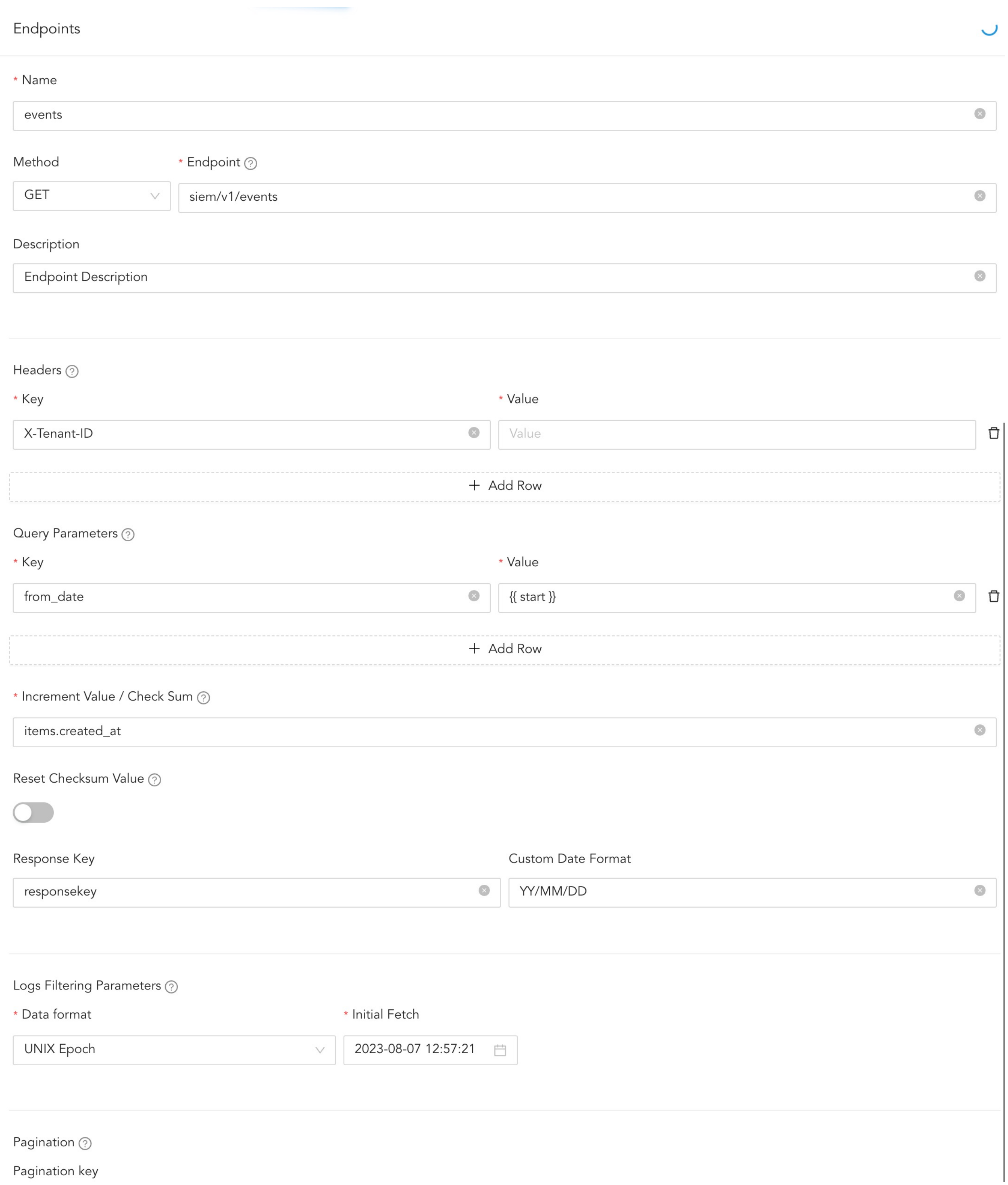
    However, the pagination key could also originate from other fields in the API response depending on how the API implements pagination.

    For *page-based*, *offset-based* and *cursor-based* paginations, users only have to configure Key and Value in **Query Parameters**.

    - **Key**: The name of the query parameter used by the API to indicate the position or page of results to fetch. This could represent a page number, an offset, or a cursor, depending on the pagination type.

- **Value**: The dynamic value for the query parameter that determines the next set of results. It is usually derived from the API response and can be a page number, an offset count, or a cursor pointing to the next item in the dataset.

14. Click **Save Changes**.



Endpoints

* Name

events

Method: GET

* Endpoint: siem/v1/events

Description

Endpoint Description

Headers

* Key	* Value
X-Tenant-ID	Value

Query Parameters

* Key	* Value
from_date	{{ start }}

* Increment Value / Check Sum

items.created_at

Reset Checksum Value

Response Key: responsekey

Custom Date Format: YY/MM/DD

Logs Filtering Parameters

* Data format: UNIX Epoch

* Initial Fetch: 2023-08-07 12:57:21

Pagination

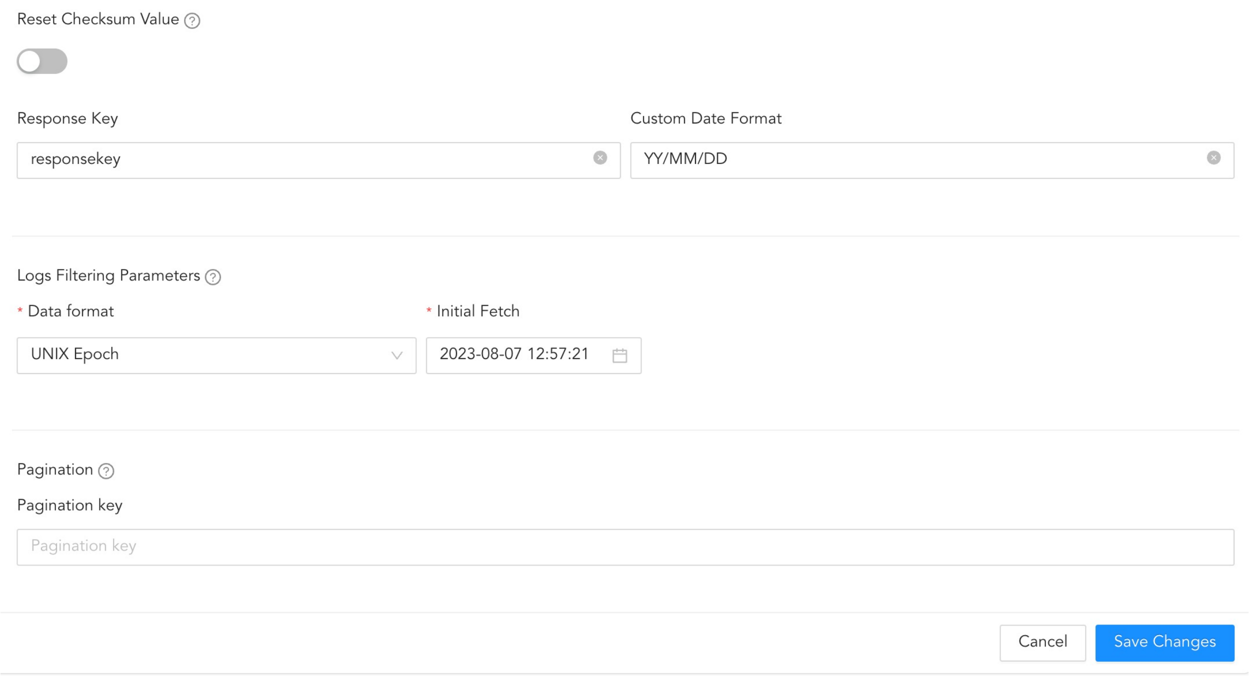
Pagination key

Fig. 3: Configuring Endpoint

To edit the endpoint configuration, click the (⋮) icon under **Action** and click **Edit**. Make the necessary changes and click **Save Changes**.

To delete the endpoint configuration, click the (⋮) icon under **Action** and click **Delete**.

To reset the Checksum values, toggle **Reset**.



Reset Checksum Value

Response Key: responsekey

Custom Date Format: YY/MM/DD

Logs Filtering Parameters

* Data format: UNIX Epoch

* Initial Fetch: 2023-08-07 12:57:21

Pagination

Pagination key: Pagination key

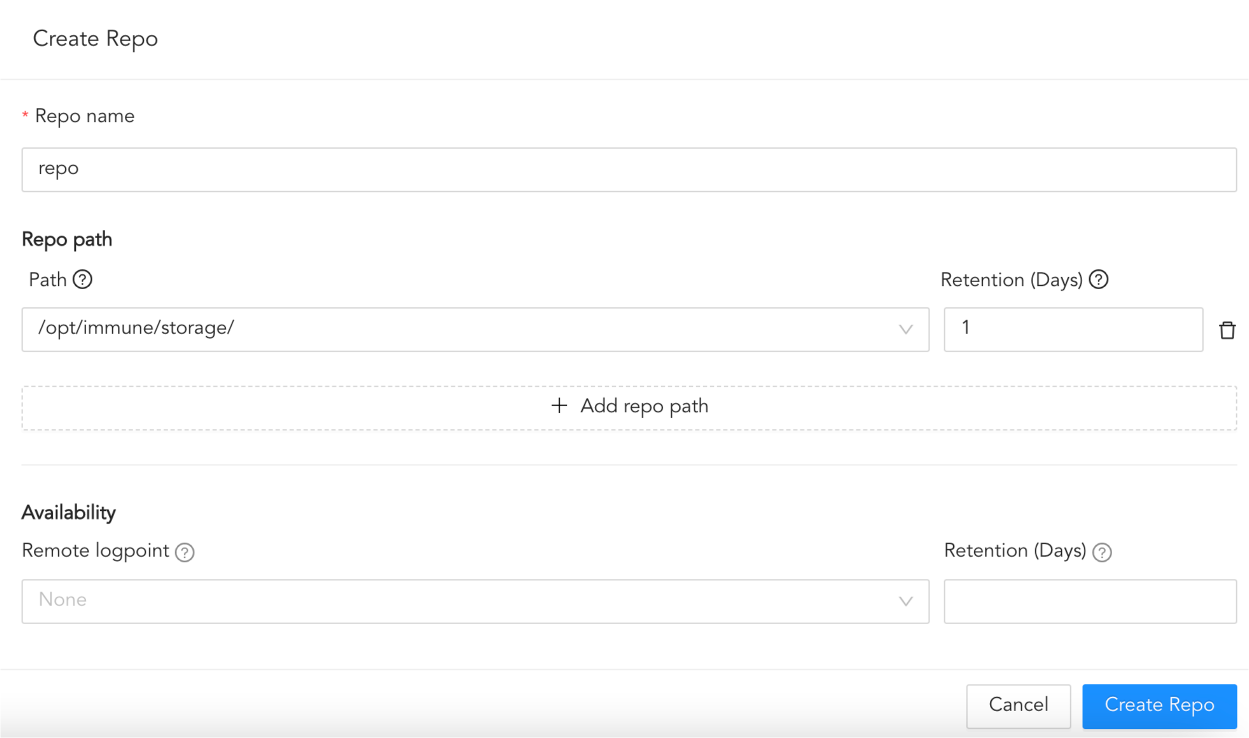
Fig. 4: Reseting Checksum

4.4 Routing

Routing lets you create repos and routing criteria for URAF. Repos store incoming logs and routing criteria determines where the logs are sent.

To create a repo:

1. Click **Routing** and **+ Create Repo**.
2. Enter a **Repo name**.
3. In **Path**, enter the location to store incoming logs.
4. In **Retention (Days)**, enter the number of days logs are kept in a repository before they are automatically deleted.
5. In **Availability**, select the **Remote logpoint** and **Retention (Days)**.
6. Click **Create Repo**.



Create Repo

* Repo name

repo

Repo path

Path

/opt/immune/storage/

Retention (Days)

1

Availability

Remote logpoint

None

Retention (Days)

Fig. 5: Creating a Repo

In **Repo**, select the created repo to store logs.

To create Routing Criteria:

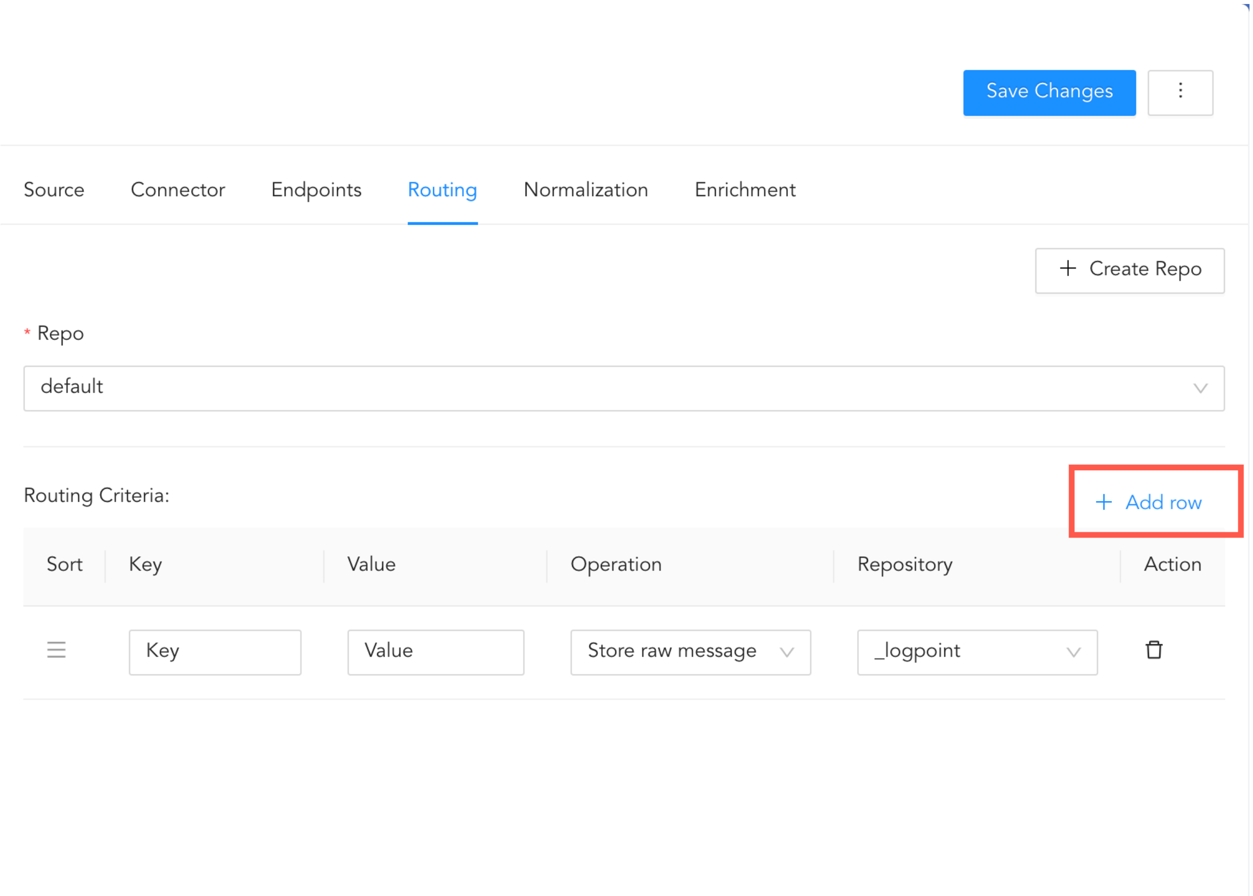
1. Click **+ Add row**.
2. Enter a **Key** and **Value**. The routing criteria is only applied to those logs which have this key-value pair.
3. Select an **Operation** for logs that have this key-value pair.

   3.1. Select **Store raw message** to store both the incoming and the normalized logs in the selected repo.

   3.2. Select **Discard raw message** to discard the incoming logs and store the normalized ones.

   3.3. Select **Discard entire event** to discard both the incoming and the normalized logs.

5. In **Repository**, select a repo to store logs.



* Repo

default

Routing Criteria:

Sort	Key	Value	Operation	Repository	Action
☰	Key	Value	Store raw message	_logpoint	🗑

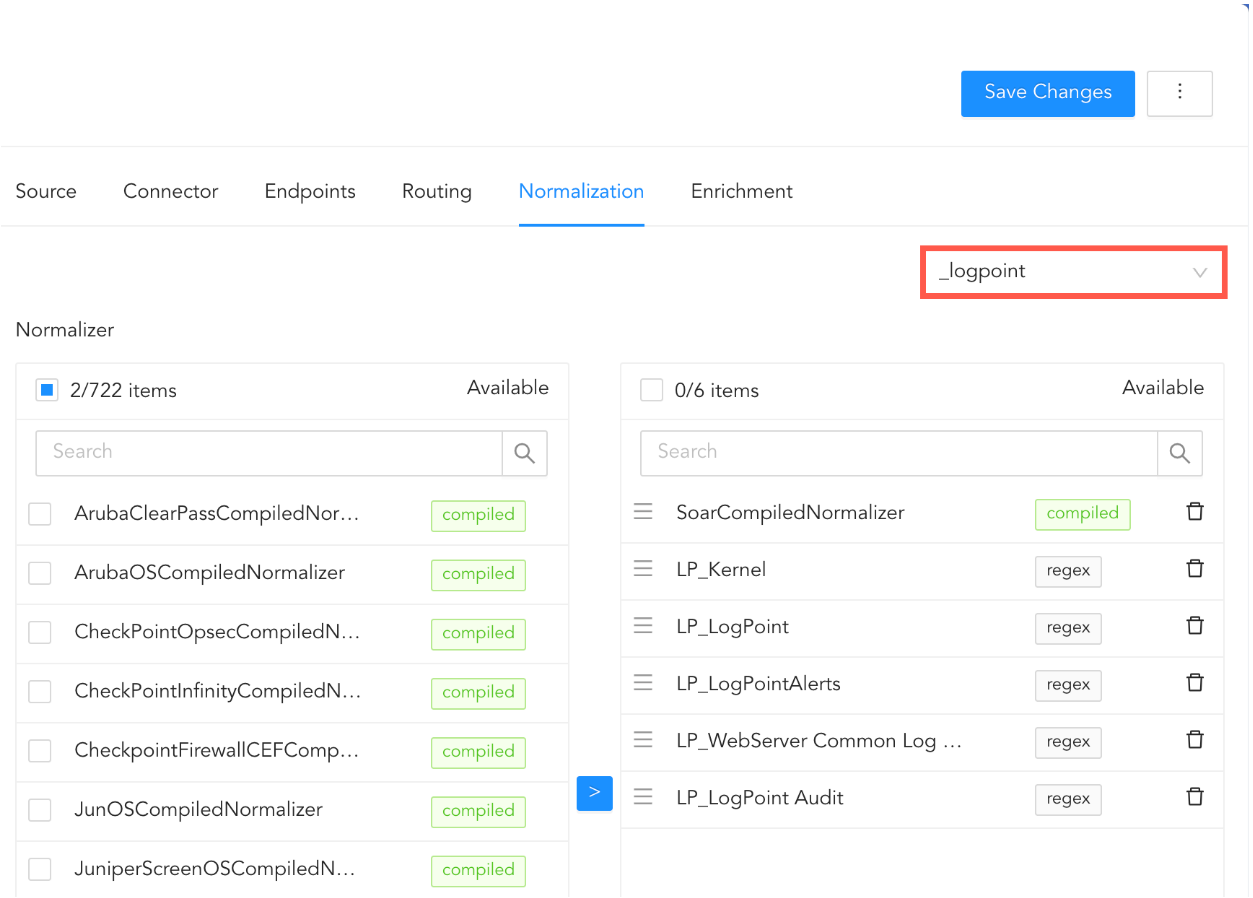
Fig. 6: Creating a Routing Criteria

Click the (🗑) icon under **Action** to delete the created routing criteria.

4.5 Normalization

In normalization, you must select normalizers for the incoming logs. Normalizers transform incoming logs into a standardized format for consistent and efficient analysis.

1. Click **Normalization**.
2. You can either select a previously created normalization policy from the **Select Normalization Policy** dropdown or select a **Normalizer** from the list and click the swap(>) icon.



■ 2/722 items	Available
□ ArubaClearPassCompiledNor…	compiled
□ ArubaOSCompiledNormalizer	compiled
□ CheckPointOpsecCompiledN…	compiled
□ CheckPointInfinityCompiledN…	compiled
□ CheckpointFirewallCEFComp…	compiled
□ JunOSCompiledNormalizer	compiled
□ JuniperScreenOSCompiledN…	compiled

□ 0/6 items	Available
SoarCompiledNormalizer	compiled
LP_Kernel	regex
LP_LogPoint	regex
LP_LogPointAlerts	regex
LP_WebServer Common Log …	regex
LP_LogPoint Audit	regex

Fig. 7: Adding Normalizers

4.6 Enrichment

In enrichment, select an enrichment policy for the incoming logs. Enrichment adds details like user information or geolocation to logs before analysis.

1. Click **Enrichment**.
2. Select an **Enrichment Policy**.

Click **Create Log Source** to save the configurations of Source, Connector, Endpoints, Routing, Normalization and Enrichment.

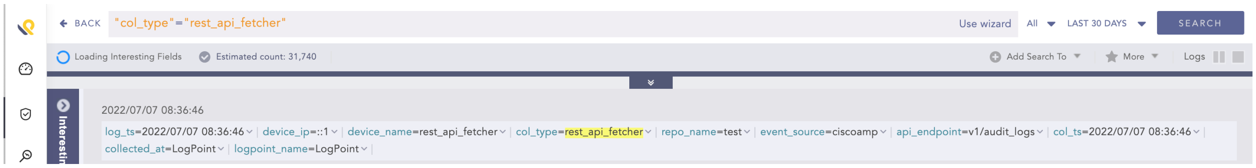
CHAPTER

FIVE

ACCESSING THE UNIVERSAL REST API FETCHER LOGS

Use the following query to access the logs:

```
col_type = rest_api_fetcher
```



```
2022/07/07 08:36:46
log_ts=2022/07/07 08:36:46 | device_ip=::1 | device_name=rest_api_fetcher | col_type=rest_api_fetcher | repo_name=test | event_source=ciscoamp | api_endpoint=v1/audit_logs | col_ts=2022/07/07 08:36:46 |
collected_at=LogPoint | logpoint_name=LogPoint |
```

Fig. 1: Universal REST API Fetcher Sample Log

CHAPTER

SIX

SUPPORTED PAGINATION TYPES

Pagination is used in API responses to divide a large dataset into smaller segments, like pages, for consistent navigation. API responses include **pagination fields** that specify details, including the current page, page size, limit, next page link, and next cursor.

Supported Pagination Types

1. Page-based pagination
2. Offset-based pagination
3. Link-based pagination
4. Cursor-based pagination

6.1 Page-based Pagination

Page-based pagination divides a large dataset into small pages. To retrieve a specific page, you must mention the page number in the API request.

Response example containing page information

```
{
  "items": [
    {
      "created_at": "2025-03-25T10:51:17.043463+00:00",
      "id": 73,
      "value": "Item 73"
    },
    {
      "created_at": "2025-03-26T10:51:17.043469+00:00",
      "id": 74,
      "value": "Item 74"
    },
    ....
```

(continues on next page)

(continued from previous page)

```
  ],
  "pagination": {
    "hasPotentiallyAnotherPage": true,
    "page": 0,
    "pageSize": 20
  }
}
```

1. **hasPotentiallyAnotherPage** indicates if there is a next page. If **hasPotentiallyAnotherPage** is true, the next request adds 1 to the page query parameter for the subsequent fetch.
2. **Page** displays the current page number.
3. **pageSize** denotes the number of items per page.

Note: When configuring **Query Parameter** in URAF for the above response example, the **Key** is page and the **Value** is {% if pagination__hasPotentiallyAnotherPage %}{{ pagination__page + 1 }}{% endif %}.

6.2 Offset-based Pagination

Offset-based pagination uses an offset value that represents the number of pages to skip from the start before fetching items.

Response example containing next offset

```
  "items": [
    {
      "created_at": "2025-04-05T09:05:43.925874+00:00",
      "id": 84,
      "value": "Item 84"
    },
    {
      "created_at": "2025-04-06T09:05:43.925876+00:00",
      "id": 85,
      "value": "Item 85"
    },
    .....
  ],
  "metadata": {
```

(continues on next page)

(continued from previous page)

```
    "limit": 10,
    "next_offset": 10,
    "offset": 0,
    "total": 17
  }

}
```

1. **limit** refers to the number of items per page.
2. **next_offset** value is the numerical value you must use in the query parameter offset to fetch the next set of pages.
3. **total** is the total number of items in a dataset.

Note: When configuring **Query Parameter** in URAF for the above response example, the **Key** is offset and the **Value** is **{{ metadata__next_offset }}**.

6.3 Link-based Pagination

Link-based pagination uses complete URLs or URI paths in API response to access paginated logs. These links refer to the **next**, **previous**, and **first or last pages** in a dataset.

Response example containing complete URLs

```
 "data": [...],
 "metadata": {
  "links": {
   "next": "https://example.com/v1/audit_logs?limit=10&offset=10",
   "prev": null
  },
}
```

Response example containing URI path

```
 "data": [...],
 "metadata": {
  "links": {
   "next": "v1/audit_logs?limit=10&offset=10",
   "prev": null
  },
}
```

Response example containing next page URL in response header link

```
Link: <http://ip_address:port/logs?page=2>; rel="next",
    <http://ip_address:port/logs?page=1>; rel="prev"

Representated as json below

headers = {
  'Link': '<http://ip_address:port/logs?page=2>; rel="next", <http://ip_address:port/logs?
↪page=1>; rel="prev"'
}
```

Response example containing next page URI in response header link

```
Link: <logs?page=2>; rel="next",
    <logs?page=1>; rel="prev"

Represented as json below

headers = {
  'Link': '<logs?page=2>; rel="next", <logs?page=1>; rel="prev"'
}
```

Response example of next page URL in a separate header key other than header link

```
NextPageUri: https://ip:port/items?page=2&limit=10

Represented as json below

headers = {
  'NextPageUri': 'https://ip:port/items?page=2&limit=10'
}
```

Response example of next page URI in a separate header key other than header link

```
NextPageUri: /items?page=2&limit=10

Represented as json below

headers = {
  'NextPageUri': '/items?page=2&limit=10'
}
```

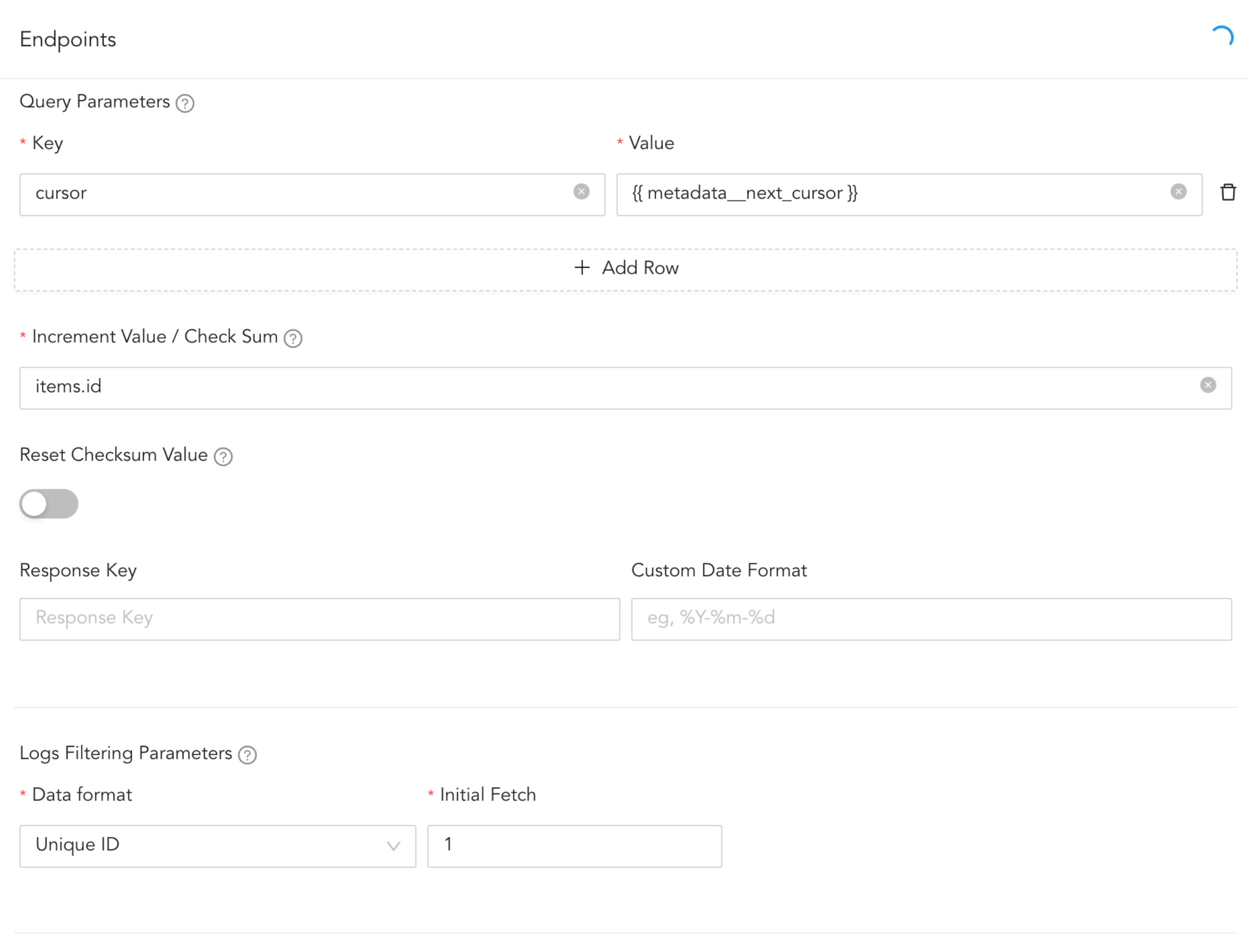
6.4 Cursor-based pagination

Cursor-based pagination fetches the next page of results starting from the last item of the previous page, referenced as a cursor. The cursor is a field such as a timestamp or database ID.

Response example containing next_cursor

```
    "items": [
        {
            "id": 1,
            "value": "Item 1"
        },
        {
            "id": 2,
            "value": "Item 2"
        },
        ......
    ],
    "metadata": {
        "next_cursor": 10
    }
}
```

next_cursor is used to fetch the next page of results.



Endpoints

Query Parameters

* Key: cursor

* Value: {{ metadata__next_cursor }}

* Increment Value / Check Sum: items.id

Reset Checksum Value

Response Key: Response Key

Custom Date Format: eg, %Y-%m-%d

Logs Filtering Parameters

* Data format: Unique ID

* Initial Fetch: 1

Fig. 1: Cursor-based Pagination

Response example containing boolean field indicating there is next cursor

```
"items": [
  {
    "created_at": "2025-04-05T06:10:11.319668+00:00",
    "id": 84,
    "value": "Item 84"
  },
  {
    "created_at": "2025-04-06T06:10:11.319671+00:00",
    "id": 85,
    "value": "Item 85"
  },
  .....
],
"metadata": {
  "has_more": true,
```

(continues on next page)

(continued from previous page)

```
      "next_cursor": 93
   }
}
```

The field **has_more** contains a boolean value, such as true or false. If it is true, the next_cursor is used to request the next page. If it is false, no additional page is requested.

Note: When configuring **Query Parameter** in URAF for the above response example, the **Key** is cursor and the **Value** is **{% if metadata__has_more %}{{ metadata__next_cursor }}{% endif %}**.

CHAPTER

SEVEN

SUPPORTED AUTHORIZATION TYPES

Authorization grants or denies access to a system or application based on a user's permissions. Each authorization type uses a unique process to exchange and validate credentials or tokens.

Supported Authorization Types

1. API Key
2. Basic
3. Digest
4. Oauth2
5. Custom

7.1 API Key authorization

API Key authorization uses a secret key (the API key) to identify and authorize a client when making API requests.

Example 1: Sending API Key in the header with the key Authorization

```
"headers": {
        "Authorization": "Bearer abc123securekey"
    }
}
```

Example 2: Sending API key in the header with a different key name

```
"headers": {
        "x-api-key": "Bearer abcdsecret123key"
    }
}
```

7.2 Basic Authorization

Basic Authorization is an HTTP authentication method that sends a Base64-encoded string containing the username and password in the "Authorization" header.

Example:

```
"headers": {
          "Authorization": "Basic MTIxOWEyYjczYmExYmlLWE5YjItMzg2NjkwYmFjMjZj"
      }
}
```

7.3 Digest Authorization

Digest Authorization is an HTTP authentication method that applies hash and nonces to the username and password to transmit credentials securely.

Example:

```
"headers": {
 "Authorization": "Digest username=\"alice\", realm=\"example.com\", nonce=\
↪"dcd98b7102dd2f0e8b11d0f600bfb0c093\", uri=\"/data\", response=\
↪"6629fae49393a05397450978507c4ef1\", qop=auth, nc=00000001, cnonce=\"0a4f113b\""
}
```

7.4 Oauth2 Authorization

OAuth 2.0 is an authorization framework that uses access tokens in headers and optional refresh tokens. It allows applications limited access to a user's data without exposing their credentials.

Supported Grant Types in URAF 3.4.0

1. Client Credentials
2. Password Credentials

Example:

```
"headers": {
  "Authorization": "Bearer ya29.a0AfH6SMCq_jJwEXAMPLE-TOKEN123456789"
 }
```

7.5 Custom Authorization

If an API of a source requires an authentication method other than the supported methods, URAF supports it via lightweight applications or integrations that run along with URAF. After installing the application, you can select the vendor template in URAF to create a log source and enter the credentials. For example, Cybereason and DuoSecurity.