Event Integration Guide

About Events API

The Argus events API provides endpoints for searching and fetching events, as well as event statistics.

The Event API uses role based access control, so any search will be limited to the customers for which the user has permission to view events.
In addition, fetching events by case provides access to the events associated with that case, provided that the user has read access to the case.

Please read the General Integration Guide to learn the general concepts and common data structures used throughout the Argus API.

Detailed API documentation

The Swagger API documentation is always up-to-date and lets you try out any query with your user session or an API-key.

Integration guide

Fetch a single event

To fetch an event, append the event ID to the base URL

```
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v1/AGGR
/1569185283911/2171/6c36deb6-d5e3-461c-bab4-fca397006cb9
```

This returns the entire event:


```
{
  "responseCode": 200,
  "limit": 0,
  "offset": 0,
  "count": 0,
  "metaData": {},
  "messages": [],
  "data": {
    "customerInfo": {
      "id": 2171,
      "shortName": "demo",
      "name": "Demo"
    },
    "properties": {
      "http_protocol": "HTTP/1.1",
      "event.original.signature": "SNORT-1:41336",
      "agent.parseTimestamp": "1569185281851",
      ...
    },
    "associatedCase": null,
    "location": {
      "shortName": null,
      "name": null,
      "timeZone": "Europe/Oslo",
      "id": 0
    },
    "attackInfo": {
      "alarmID": 58129,
      "alarmDescription": "MALWARE - Andr.Trojan.Sysch variant activity",
      "attackCategoryID": 363,
      "attackCategoryName": "Callback traffic (Check-in) (infected client
/server)",
      "signature": "SNORT-1:41336"
    },
    "domain": null,
    "uri": null,
```

```
    "count": 1,
    "source": {
      "port": 12342,
      "geoLocation": null,
      "networkAddress": {
        "host": true,
        "ipv6": false,
        "maskBits": 32,
        "multicast": false,
        "public": false,
        "address": "10.5.97.201"
      }
    },
    "destination": {
      "port": 80,
      "geoLocation": {
        "countryCode": "NO",
        "countryName": null,
        "locationName": "",
        "latitude": 52.3824,
        "longitude": 4.8995
      },
      "networkAddress": {
        "host": true,
        "ipv6": false,
        "maskBits": 32,
        "multicast": false,
        "public": true,
        "address": "94.127.56.1"
      }
    },
    "protocol": "0",
    "timestamp": 1569185283911,
    "startTimestamp": 1569185283910,
    "endTimestamp": 1569185283910,
    "lastUpdatedTimestamp": 1569186046339,
    "flags": [
      "BLOCKED",
      "CREATED_BY_ANALYSIS_FILTER",
      "FINALIZED",
      "CUSTOM_SOURCE_AGGREGATION",
      "CUSTOM_DESTINATION_AGGREGATION",
      "SOURCE_IS_CUSTOMERNET"
    ],
    "severity": "high",
    "detailedEventIDS": [],
    "id": "AGGR/1569185283911/2171/6c36deb6-d5e3-461c-bab4-fca397006cb9"
  },
  "size": 0
}
```


Searching for events

To search for events, query the "aggregated" endpoint to search among all security events.

```
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v1
/aggregated
```

See Swagger API documentation for details on available query parameters

Advanced search

To do more advanced filtering, use the advanced search endpoint.

The advanced search endpoint requires a startTimestamp.

```
curl -H "Argus-API-Key: my/api/key" -XPOST https://api.mnemonic.no/events
/v1/aggregated/search -d '{
  "signature": ["SNORT-1:41336"],"startTimestamp":1567288800000
}'
```

See Swagger API documentation for details on available search parameters

Searching by IP - CIDR ranges

Argus will by default resolve any CIDR IP range to match any IP contained within that range.

Moreover, Argus will also widen the search to include any *covering ranges*, i.e. any CIDR range with lower maskbits than specified which will contain the specified range/IP.

Example: Searching for 192.168.1.32/28

- Will match CIDR range 192.168.1.32/28
- Will match IP 192.168.1.33 (part of 192.168.1.32/28)
- Will match CIDR range 192.168.1.40/29 (part of 192.168.1.32/28)
- Will match CIDR range 192.168.1.0/24 (covers 192.168.1.32/28)
- Will match CIDR range 0.0.0.0/0 (covers 192.168.1.32/28)

To control this behaviour, use parameters `sourceIPMinBits` and `destinationIPMinBits`. Setting these parameters will limit how wide CIDR ranges will be considered.

```
# Search for any event with source or destination IP within 192.168.1.0
/24.
# Do not match any CIDR networks wider than 192.168.0.0/20 (neither for
source nor destination)
curl -H "Argus-API-Key: my/api/key" -XPOST https://api.mnemonic.no/events
/v1/aggregated/search -d '{
  "ip": ["192.168.1.0/24"],"startTimestamp":1567288800000,
  "sourceIPMinBits":20,  "destinationIPMinBits":20
}'
```

Any CIDR range specified in the search criteria will be searched for, even if the CIDR range has a lower maskbits than specified in these criteria.

Fetching events associated to a case

To fetch events associated with a case, use the eventsByCase endpoint

```
#list the first 10 events associated with case 12342345
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v1/case
/12342345?limit=10
```

Fetching large amounts of events

To pull out large amounts of events, it is important to select a strategy which is resource-efficient for both the server and client, as well as a robust strategy which will leave you with a complete dataset. There are several ways for doing this, all depending on your use case.

Common for all strategies

- Use a medium-size limit to avoid fetching too big resultsets, but big enough to make transfer efficient
- Use a clear sorting and limiting strategy to ensure that you will end up with a complete dataset

Pulling events continuously

This strategy fetches events in the order they are written in the event store. If an event is updated and written again, it will be returned again.
This strategy is well suited for fetching events continously and near real-time.

```
curl -H "Argus-API-Key: my/api/key" -X POST -H"Content-Type: application
/json" https://api.mnemonic.no/events/v1/aggregated/search -d '{
  "startTimestamp": "2020-01-01T00:00:00Z", //need to set a lower bound
for the search
  "limit":1000,
  "sortBy":"lastUpdatedTimestamp"
}'
```

This will fetch the first 1000 events sorted by event lastUpdatedTimestamp. The response

```
{
  "responseCode": 200,
  ...
  "data": [
    ...
    {..., "lastUpdatedTimestamp": 1581721863924, ... }
  ]
}
```

Subsequent requests should use the lastUpdatedTimestamp value of the last event in the previous result, to fetch the next 1000 events.

```
curl -H "Argus-API-Key: my/api/key" -X POST -H"Content-Type: application
/json" https://api.mnemonic.no/events/v1/aggregated/search -d '{
  "startTimestamp": "2020-01-01T00:00:00Z", //need to set a lower bound
for the search
  "lastUpdatedTimestamp": 1581721863924
  "limit":1000,
  "sortBy":"lastUpdatedTimestamp"
}'
```

Submitting events via API

The event collector API is designed to allow integration of 3rd party agents to deliver events into Argus.

The API allows submitting single events or a bulk of events, and will handle events asynchronously.

Notes about collector APIs:@tor

- Collector APIs are created for high-capacity, asychronous operation.
- The receiving endpoint will validate the data and the users permissions to add them, and will reject invalid data and data which the user does not have the permission to add.
- However, the API will return successful result once the data is *enqueued* for storage, so the results may not be immediately available. If the backend engine is under pressure, the actual storage and indexing of the data may be delayed.
- The API does not distinguish between add or update. When storing the events, if the same event ID already exists, the event will be updated with changes from the incoming event.

Note: It is not possible to update a FINALIZED event.
If submitting updates to an existing event which has previously been marked as FINALIZED, the update will be silently ignored.
The submit-API will still return status "accepted".

The ID of an event is a separate object in the submission request, with properties

- type
- customerID
- timestamp
- UUID

The id of an event can also be written using an Argus event ID string, e.g. AGGR/1570715141521/1/1679fbfc-1792-4992-8428-3cb4df642fa8

Submitting an event

Submit a new event by calling the single event submission API:

```
#create a new event with signature MySignature
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v2
/event -d '{
  "signature": "MySignature"
}'
```

By default, this event will be submitted with the following fields:

- Time: timestamp, startTimestamp and endTimestamp "now"
- Customer: Will be set to the current users customer
- Type: Default event type is "aggregated". Use type "raw" to submit raw events.
- Severity: low
- Count: 1
- Properties: agent.id, agent.timestamp and engine.timestamp

Remaining fields have default value null (not set).

To submit an event for a specific customer, you need to populate the "id" object of the event:

```
#create a new event with signature MySignature, for customer "shortname",
with a specific event timestamp.
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v2
/event -d '{
  "id":{
    "customer": "shortname",
    "timestamp": "2019-10-05T15:07:10Z"
  },
  "signature": "MySignature"
}'
```

The endpoint will reject the event if the user does not have access to submit events for the specified customer.

Submission response

The endpoint will respond with code 201, with a status object containing the generated ID of the accepted event.
If the event contained invalid fields, the endpoint will return 412.

Each of the identifying fields (timestamp, customerID, type and UUID) will be given a default value if not specified in the request.

```
{
 "responseCode":201,
 ...
 "data":{
   "id":"AGGR/1570715141521/1/1679fbfc-1792-4992-8428-3cb4df642fa8",
   "status":"accepted",
   "message":null
 }
}
```

Submitting an update

Submitting an update on a previous event requires the type, timestamp, customer and UUID to be the same as the existing event.

```
#update the previous event with an updated count
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v2
/event -d '{
  "id": {
    "customer": "shortname",
    "timestamp": "2019-10-05T15:07:10Z",
    "uuid": "1679fbfc-1792-4992-8428-3cb4df642fa8" #type "aggregated" is
implicit
  },
  "count": 2
}'
```

For simplicity, you can use the "id" field to submit the exact ID string returned in the submission status, which is a general Argus event ID:

```
#update the previous event with an updated count
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v2
/event -d '{
  "id": "AGGR/1570715141521/1/1679fbfc-1792-4992-8428-3cb4df642fa8",
  "count": 2
}'
```

Submitting with properties

Events can contain an arbitrary number of properties, and each property can have a list of values.

> ⚠ The V1 event search/get API does currently not support multiple values on properties, so an event with multiple property values will be returned as a comma-separated property string

Properties are accumulated on update, so updating an event with new properties will cause the event to contain the accumulated set of properties from all updates/submissions.
Updating an existing property will overwrite the existing value.

The properties are submitted as a map of property keys, each key mapping to a list of values.
For simplicity, sending a single value instead of a list is also supported.

```
#submit an event with properties
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v2
/event -d '{
  "properties": {
    "myprop": ["val1", "val2"],
    "otherprop": "singlevalue"
  }
}'
```

Submitting with flags

Events can contain a set of flags, as defined in the Swagger API documentation.

Flags are accumulated on update, so updating an event with new flags will cause the event to contain the accumulated set of flags from all updates/submissions.
It is not possible to DELETE a flag through an event submission.

> ⚠ Some flags have special handling, called "partial flags".
>
> E.g. for the flag "SOURCE_IS_CUSTOMERNET":
>
> If an event is submitted *without* this flag, and then updated *with* this flag, the resulting event will be marked with the partial flag SOURCE_IS_PARTIAL_CUSTOMERNET
>
> Submitting another update with SOURCE_IS_CUSTOMERNET will reset the flag of the event to SOURCE_IS_CUSTOMERNET.

```
#submit an event with properties
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v2
/event -d '{
  "flags": ["SOURCE_IS_CUSTOMERNET"]
}'
```

Submitting a bulk of events

When submitting many events, the bulk API is a more efficient solution.

The bulk API accepts a list of events, and an **error mode**.

The default error mode is "rejectAll", which will reject the entire request with a 412 status if any event is invalid.
Alternatively, error mode "dropInvalid" will drop invalid events, and submit the valid events, before returning status 201.

> ⚠ In mode "dropInvalid", the endpoint will drop events which fail to validate customer or location, or other referenced fields.
> However, invalid JSON or invalid fields in the JSON request will still cause a 412 on the entire request.

The status result from the bulk endpoint will list the number of accepted and rejected events, as well as a list of submission status objects, one for each event.

```
#submit an event with properties
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v2/event
/bulk -d '{
  "events": [
    {"signature": "FirstAlarm"},
    {"signature": "SecondAlarm"},
    {"signature": "ThirdAlarm", "location": "invalid"}
  ],
  "onError": "dropInvalid"
}'
```

Response from bulk submit with dropped invalid events:

```
{
  "responseCode": 201,
  ...
  "data": {
    "accepted": 2,
    "rejected": 1,
    "events": [
      {
        "id": "AGGR/1570715932641/1/f222ed6c-dc7b-473d-9bf0-20bb35cb1030",
        "status": "accepted",
        "message": null
      },
      {
        "id": "AGGR/1570715932642/1/3a38e04d-d58b-4d74-9c12-b41c4c435f97",
        "status": "accepted",
        "message": null
      },
      {
        "id": null,
        "status": "rejected",
        "message": "Invalid location: invalid"
      }
    ]
  }
}
```

The third event was invalid, and is therefore not assigned an ID. The order of the event status objects in the bulk status object will correspond to the events in the request.

Finalizing an event

To finalize an event, submit the event with the "finalized" property.

Setting the finalized property has the same effect as adding the event flag FINALIZED.

```
#finalize this event
curl -H "Argus-API-Key: my/api/key" https://api.mnemonic.no/events/v2
/event -d '{
  "id": "AGGR/1570715141521/1/1679fbfc-1792-4992-8428-3cb4df642fa8",
  "finalized": true
}'
```

Finalizing events prevents further updates on this event through later event submissions.

If an agent does not expect events to be updated, they should be finalized in the initial submission.