D4.5 SPHINX Embedded SIEM v1

WP4 – SPHINX Toolkits

Version: 1.00

SPHINX

A Universal Cyber Security Toolkit for
Health-Care Industry

Disclaimer

Any dissemination of results reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Copyright message

Document information

<table>
<tr><th>Grant Agreement Number</th><th colspan="3">826183</th><th colspan="3">Acronym</th><th colspan="6">SPHINX</th></tr>
<tr><td><b>Full Title</b></td><td colspan="12">A Universal Cyber Security Toolkit for Health-Care Industry</td></tr>
<tr><td><b>Topic</b></td><td colspan="12">SU-TDS-02-2018 Toolkit for assessing and reducing cyber risks in hospitals and care centres to protect privacy/data/infrastructures</td></tr>
<tr><td><b>Funding scheme</b></td><td colspan="12">RIA - Research and Innovation action</td></tr>
<tr><td><b>Start Date</b></td><td colspan="4">1stJanuary 2019</td><td colspan="4"><b>Duration</b></td><td colspan="4">36 months</td></tr>
<tr><td><b>Project URL</b></td><td colspan="12">http://sphinx-project.eu/</td></tr>
<tr><td><b>EU Project Officer</b></td><td colspan="12">Reza RAZAVI (CNECT/H/03)</td></tr>
<tr><td><b>Project Coordinator</b></td><td colspan="12">National Technical University of Athens - NTUA</td></tr>
<tr><td><b>Deliverable</b></td><td colspan="12">D4.5 SPHINX Embedded SIEM v1</td></tr>
<tr><td><b>Work Package</b></td><td colspan="12">WP4 – SPHINX Toolkits</td></tr>
<tr><td><b>Date of Delivery</b></td><td colspan="3"><b>Contractual</b></td><td colspan="3">M20</td><td colspan="3"><b>Actual</b></td><td colspan="3">M20</td></tr>
<tr><td><b>Nature</b></td><td colspan="4">R - Report</td><td colspan="4"><b>Dissemination Level</b></td><td colspan="4">P - Public</td></tr>
<tr><td><b>Lead Beneficiary</b></td><td colspan="12">PDMFC</td></tr>
<tr><td rowspan="2"><b>Responsible Author</b></td><td colspan="5">Eli de Lima</td><td colspan="2"><b>Email</b></td><td colspan="5">eli.lima@pdmfc.com</td></tr>
<tr><td colspan="5"></td><td colspan="2"><b>Phone</b></td><td colspan="5"></td></tr>
<tr><td><b>Reviewer(s):</b></td><td colspan="12"></td></tr>
<tr><td><b>Keywords</b></td><td colspan="12">SIEM, logs, event management</td></tr>
</table>

Document History

Version	Issue Date	Stage	Changes	Contributor
0.10	09/06/2020	Draft	ToC	Eli de Lima (PDMFC)
0.20	20/08/2020	Draft	Content	Eli de Lima (PDMFC)
0.30	24/08/2020	Draft	Content	Bianca Nóbrega (PDMFC)
0.40	27/08/2020	Draft	Internal Review 1	Acarali Dilara (TEC)
0.40	27/08/2020	Draft	Internal Review 1	Waqar Asif (TEC)
0.50	27/08/2020	Pre - Final	Adjustments suggested by internal reviewers (TEC)	Eli de Lima (PDMFC)
0.50	28/08/2020	Pre - Final	Quality Control	George Doukas (NTUA), Michael Kontoulis (NTUA)
1.00	28/08/2020	Final	Final	Christos Ntanos (NTUA)

Executive Summary

This document reports the development status for the SPHINX System Information and Event Management (SIEM) component. Moreover, it presents the design, architecture and implementation of the SIEM component and its subcomponents, by providing technical details and explaining the level of interaction performed by the subcomponents within the SIEM system and demonstrating how users and external components can interact with the SIEM platform.

This deliverable shows the results accomplished on the first iteration (M13-M20) of the associated task (T4.5). However, the SIEM component is continuously growing and evolving, thus in the next iteration this document will incorporate refinements and updates of the SIEM component, integration efforts and use cases, for demonstrating all the functionalities encompassed by the system.

Contents

Table of Tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Table of Figures

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Table of Abbreviations

SIEM - Security Information and Event Management

API - Application Programming Interface

REST - Representational state transfer

UI - User Interface

MQL - Metano Query Language

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

1 Introduction

1.1 Purpose & Scope

This document reports the development of the Security Information and Event Management component, which is responsible for implementing a query interface where other components or users can distinguish between normal and abnormal operations. In order to accomplishing that, the SIEM component collects data from some key components present in the SPHINX ecosystem, normalizing all this information into a data repository, where it can be queried and monitored, triggering actions when specific events occur.

1.2 Structure of the Deliverable

This document is structured as follows: Section 2 provides an overview of the SIEM component from a technical perspective, where we introduce the SIEM scope and architecture, with all of its subcomponents and what role they play in the SPHINX ecosystem. Section 3 presents the SIEM Web user interface component that provides a friendlier interaction with SIEM's main functionalities. In Section 4 we illustrate some uses cases covered by the SIEM component. Section 5 is reserved to show the conclusions of this deliverable and future steps.

1.3 Relation to Other WPs & Tasks

This document is within WP4 - SPHINX Toolkits, namely as task T4.5 SPHINX Security Information and Event Management. Within the scope of the SPHINX project, the tasks which relate to this task are T3.5 – D3.5: SPHINX Automated Cybersecurity Certification, T3.3 - D3.3: Vulnerability Assessment as a Service, T4.1 - D4.1: Anomaly Detection, T4.2 - D4.2: Sandbox, T4.3 - D4.3: BBTR – Blockchain Based Threats Registry, T4.4 - D4.4: Honeypot and T5.3 – D5.3: Security Incident/Attack Simulator.

2 Overview of SIEM

2.1 Scope of SIEM

The main objective behind the SPHINX Security Information and Event Management component is to provide a solid log management tool for security-related events that the security centre administrators can rely on for responding to security incidents as early as possible. Thus, the SIEM ecosystem will encompass log collection and normalization, data correlation, alert management and reporting, in order to guarantee near real-time analysis of security alerts, which have been generated by network hardware and applications.

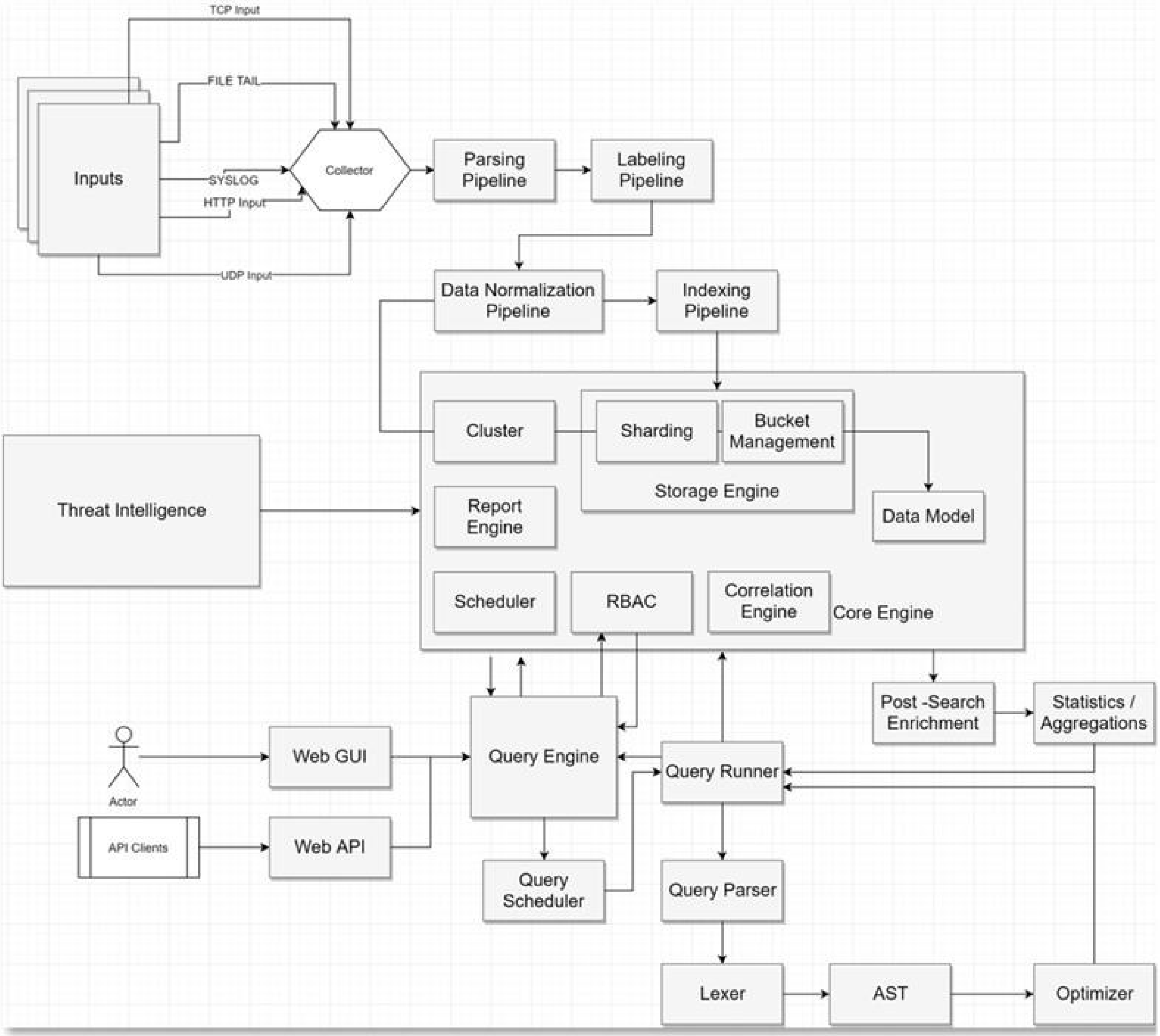


Figure 1. SIEM Architecture Overview

Figure 1 illustrates the interactions between all subcomponents comprising in the SIEM main component. The top-left corner shows the log files being sent through the network and then being collected by the component responsible for gathering logs, normalizing, and saving the data into the SIEM database. The Core Engine in the middle is responsible for managing the data correlations, monitoring security events, and applying the defined actions whenever security alerts are raised. Finally, the Query Engine is the environment that executes MQL queries, which is designed to be a high performing query language for log files.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

2.1.1 Design Principles

Considering the design and software development lifecycle (SLDC) principles described in deliverable D6.1, we structured and started the development focusing on security, code maintainability, interoperability, scalability and ease of deployment. The SIEM component is divided into four different sub-components, each one has its own processes and project structures, which means they are all managed independently.

2.1.1.1 Stakeholders' Requirements Fulfilment

According to deliverable D2.6 - SPHINX Architecture v2, the SIEM component has six (6) basic functional requirements, as shown in Table 1, and they intend to fulfil some of the basic requirements raised by the stakeholders.

Technical Specification ID	Stakeholder Requirement ID	Observations
SIEM-F-010	***STA-F-250*** ***STA-F-260***	*Categorisation of cyber events* *Patterns of incidents (including external sources)*
SIEM-F-040	***STA-F-100*** ***STA-F-290***	*Concentrate data and performance* *Collection of evidence (logs, records, registries)*
SIEM-F-050	***STA-F-290***	*Collection of evidence (logs, records, registries)*
SIEM-F-060	***STA-F-220***	*Data analysis and visualisation*
SIEM-F-070	***STA-F-240*** ***STA-F-290***	*Deal with known cyberattacks* *Collection of evidence (logs, records, registries)*
SIEM-F-080	***STA-F-560***	*Query features*

Table 1 – Functional requirements traceability (SPHINX Project. D2.6 – SPHINX Architecture v2)

2.1.1.2 Technical Details

The SIEM was divided into several components, each having a specific objective. Because of this, we have several technologies involved in development.

Table 2 shows the technologies that we use in the SIEM components.

Technology	Where	Why
Angular	*SIEM Web*	*Angular is one of the most popular JavaScript frameworks for developing single page applications. As it is based on components, it becomes highly reusable, productive and easy to maintain.*
Nest.JS	*Document Manager and Event Manager*	*NestJS is a progressive Node.js framework for building efficient, reliable and scalable server-side applications.* *With its defined architecture, it is clear and easy to transform into microservices.*
Flask	*MQL*	*Flask is a micro-framework for Python. It has a simple and expansive core which allows a project to have only the resources necessary for its execution.*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Elastic Search	*Document Manager, Event Manager and MQL*	*Elasticsearch is a distributed, open source search and analytics engine for all types of data.* *It supports a large volume of data without losing the performance.*
Redis	*Event Manager*	*Redis is a fast, open-source, in-memory key-value data store for use as a database, cache, message broker, and queue.* *Due to its speed and ease of use, Redis is in high demand for applications that require the best performance in the market.*
Swagger	*Document Manager and Event Manager*	*Swagger is a framework for description, consumption and visualization of Rest API.* *With Swagger, the documentation can evolve at the same pace as the implementation.*
Docker	*All Components*	*Docker is an open-source platform that facilities the creation and administration of isolated environments. It makes it possible to package an application or environment inside a container, making it portable for any other host that has Docker installed.*
Docker Compose	*All Components*	*Compose is a tool for defining and running multi-container Docker applications. With it, we can set up the entire environment quickly.*

Table 2 – Technologies uses in SIEM components.

2.2 Background

A SIEM system is a management security approach that provides a comprehensive view of an organization's security information system. Furthermore, these systems are an important component of company networks, IT infrastructures and the cybersecurity domain. In fact, they allow to consolidate and to evaluate messages and alerts of individual components of an IT system. At the same time messages of specialized security systems (firewall-logs, VPN gateways etc.) can be considered. However, practice has shown that these SIEM systems are extremely complex and only operable with large personnel effort. Many times, SIEM systems are installed but neglected in continuing operation. The SIEM being developed for SPHINX aims to mitigate those flaws present in other SIEM systems, providing a practical tool that seamlessly integrates with as many components as necessary, and offers a user-friendly interface to manage technical and analytical aspects of the entire system.

2.3 SIEM in SPHINX

The SIEM component in SPHINX is, so far, subdivided into 3 main components: Document Manager, Event Manager and Metano Query Language (MQL), where each of them is responsible for crucial process feature of a SIEM system, namely data normalization, event management and text search, respectively. They were designed this way in order to segment the project structure and improve component development, scalability and maintainability as a whole. The following subtopics present a more in-depth description of each one of them.

2.3.1 Document Manager

The Document Manager subcomponent is responsible for controlling the entry and search of all logs, providing input methods for file upload, file and/or directory monitoring, and TCP and HTTP connections.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

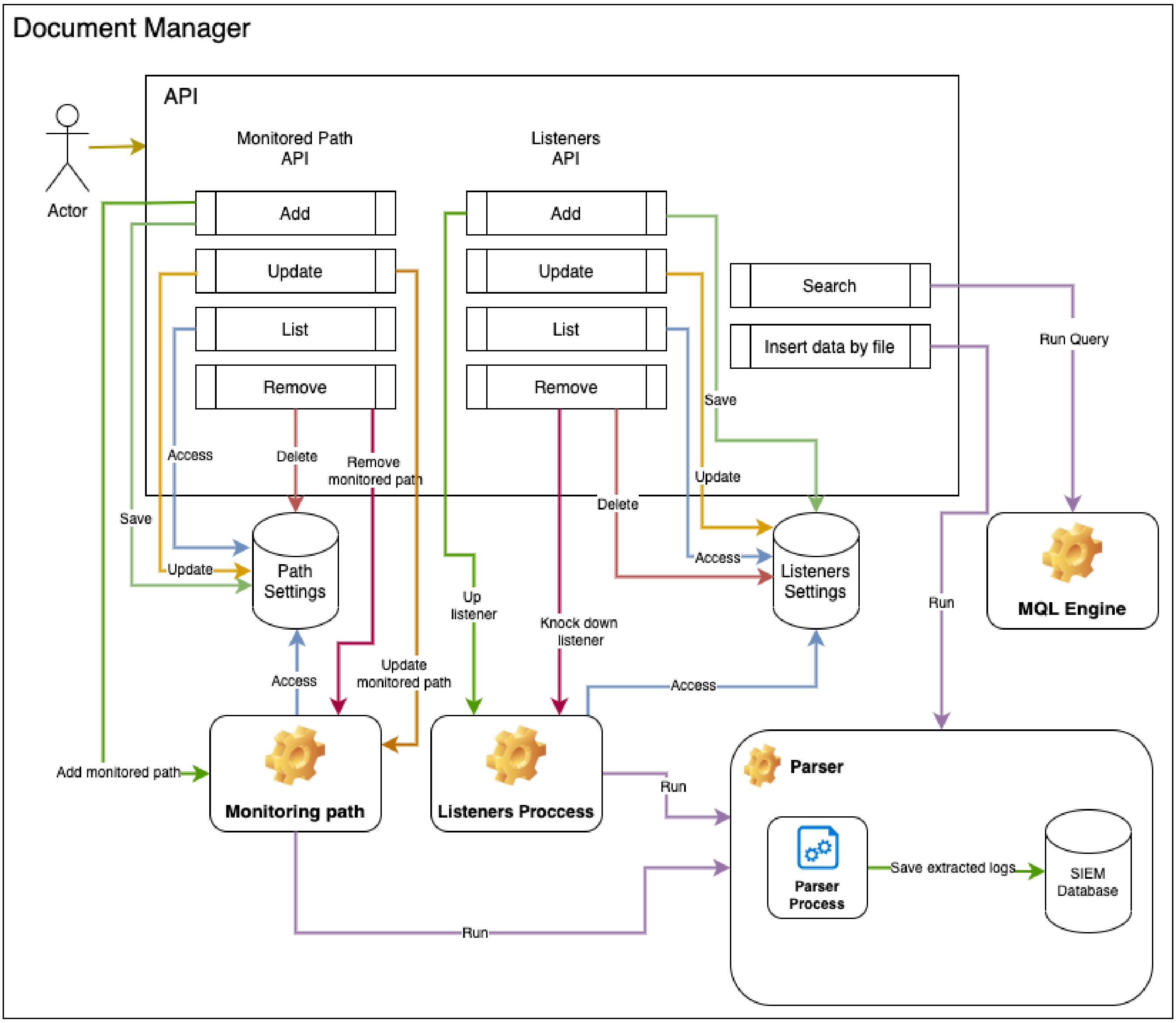


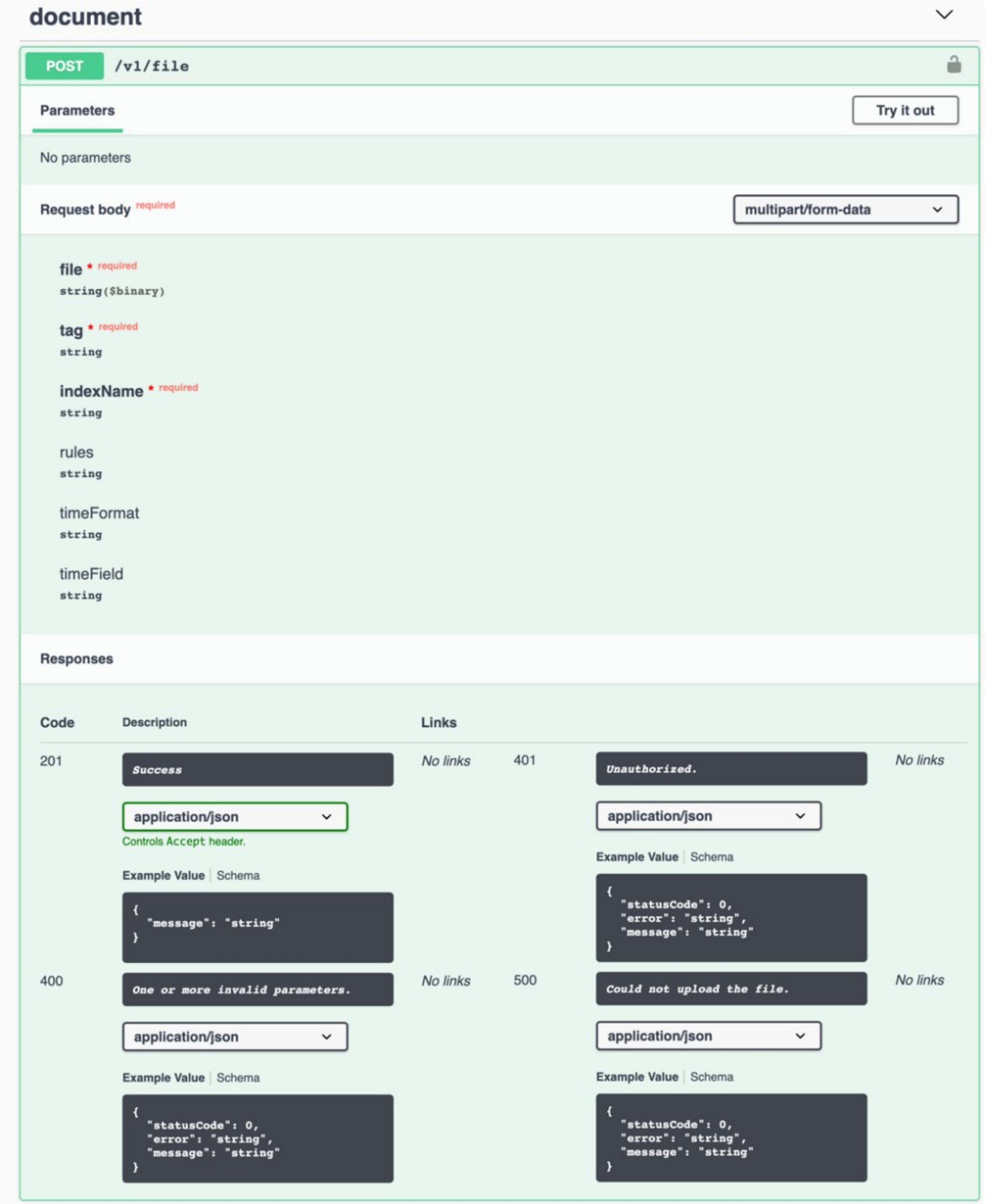
Figure 2. Document Manager Architecture

Figure 2 depicts the architecture designed for the Document Manager subcomponent, with its main activities and possible interactions. The API area lists all the methods that are externally available for other components to call. The ***Monitoring path*** process is used for watching for changes in files and/or directories. Furthermore, whenever any new change is detected, the parser method is invoked for that specific file, and the normalized data is saved to the SIEM database. Regarding the ***Listeners Process***, it is responsible for starting up the listening servers based on the saved settings. Once a listener is up and running, it will capture all incoming data sent by the agents installed on the external components, parse the log files and save it to the SIEM database. The ***MQL Engine*** is used exclusively by the search method for executing the MQL queries, and it is responsible for accessing the SIEM common databases to retrieve information based on the specified query.

In the following section the API methods are presented in detail, covering their purpose, inputs and response.

2.3.1.1 Swagger API Methods

This section illustrates the API methods exposed in the Document Manager subcomponent, where the external components can interact with its functionality, mostly for inserting and querying logs.



document

POST /v1/file

Parameters

Try it out

No parameters

Request body required

multipart/form-data

file * required
`string($binary)`

tag * required
`string`

indexName * required
`string`

rules
`string`

timeFormat
`string`

timeField
`string`

Responses

Code | Description | Links

201 — *Success* — No links

application/json
Controls Accept header.

Example Value | Schema

```
{
  "message": "string"
}
```

401 — *Unauthorized.* — No links

application/json

Example Value | Schema

```
{
  "statusCode": 0,
  "error": "string",
  "message": "string"
}
```

400 — *One or more invalid parameters.* — No links

application/json

Example Value | Schema

```
{
  "statusCode": 0,
  "error": "string",
  "message": "string"
}
```

500 — *Could not upload the file.* — No links

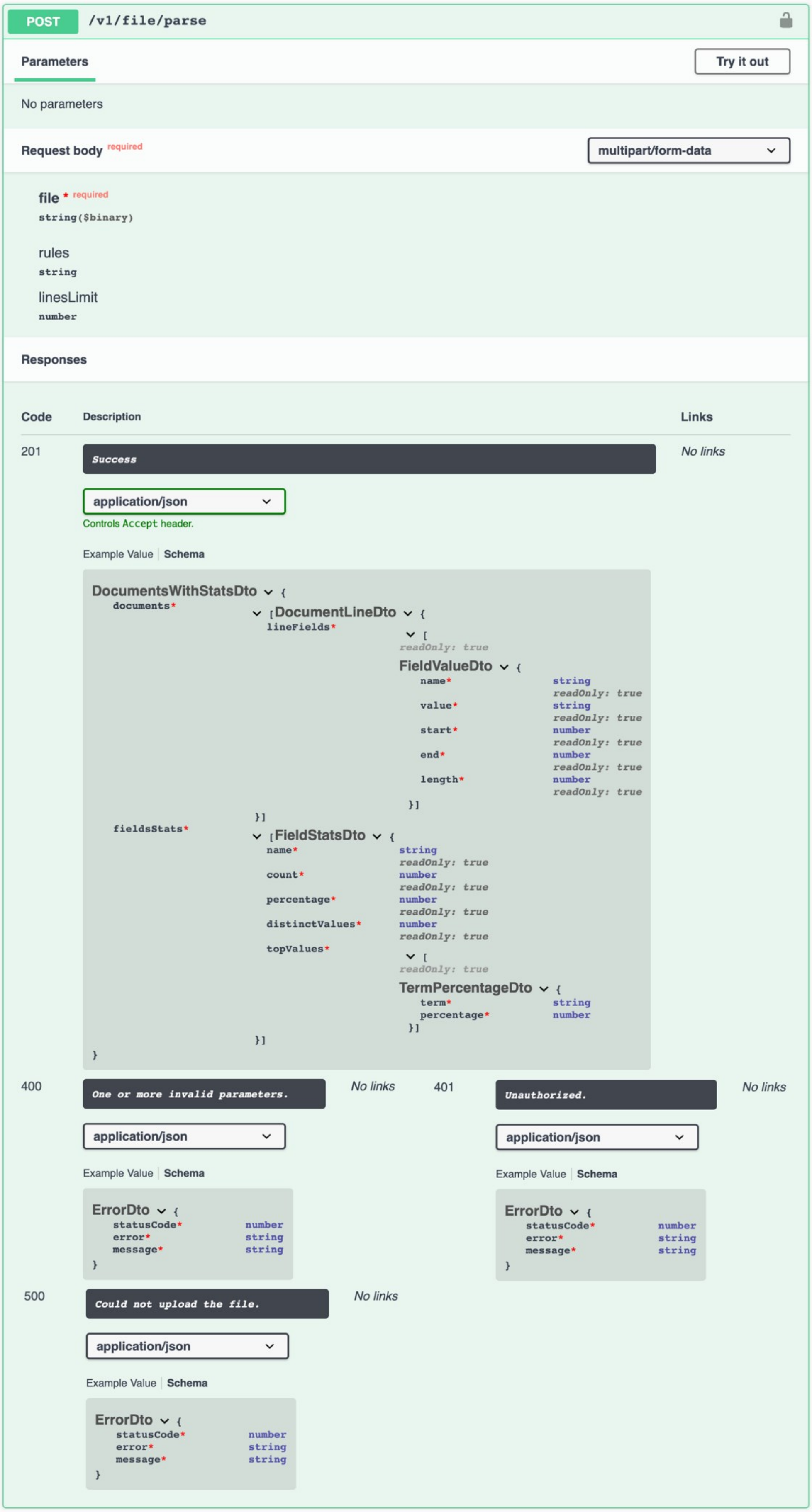
application/json

Example Value | Schema

```
{
  "statusCode": 0,
  "error": "string",
  "message": "string"
}
```

Figure 3. API method for File Upload

Figure 3 shows the method for transforming and saving data from a file upload. This method will go through the entire file, applying the informed normalizing rules. Once the parsing process is done, the extracted document will be written into the SIEM database for future searches.



POST /v1/file/parse

Parameters — Try it out

No parameters

Request body required — multipart/form-data

file * required
string($binary)

rules
string

linesLimit
number

Responses

Code	Description	Links
201	Success application/json Controls Accept header. Example Value | Schema	No links

```
DocumentsWithStatsDto {
   documents*        [DocumentLineDto {
                        lineFields*         [
                                            readOnly: true
                                            FieldValueDto {
                                               name*              string
                                                                  readOnly: true
                                               value*             string
                                                                  readOnly: true
                                               start*             number
                                                                  readOnly: true
                                               end*               number
                                                                  readOnly: true
                                               length*            number
                                                                  readOnly: true
                                            }]
                     }]
   fieldsStats*      [FieldStatsDto {
                        name*               string
                                            readOnly: true
                        count*              number
                                            readOnly: true
                        percentage*         number
                                            readOnly: true
                        distinctValues*     number
                                            readOnly: true
                        topValues*          [
                                            readOnly: true
                                            TermPercentageDto {
                                               term*              string
                                               percentage*        number
                                            }]
                     }]
}
```

Code	Description	Links
400	One or more invalid parameters. application/json Example Value | Schema	No links
401	Unauthorized. application/json Example Value | Schema	No links
500	Could not upload the file. application/json Example Value | Schema	No links

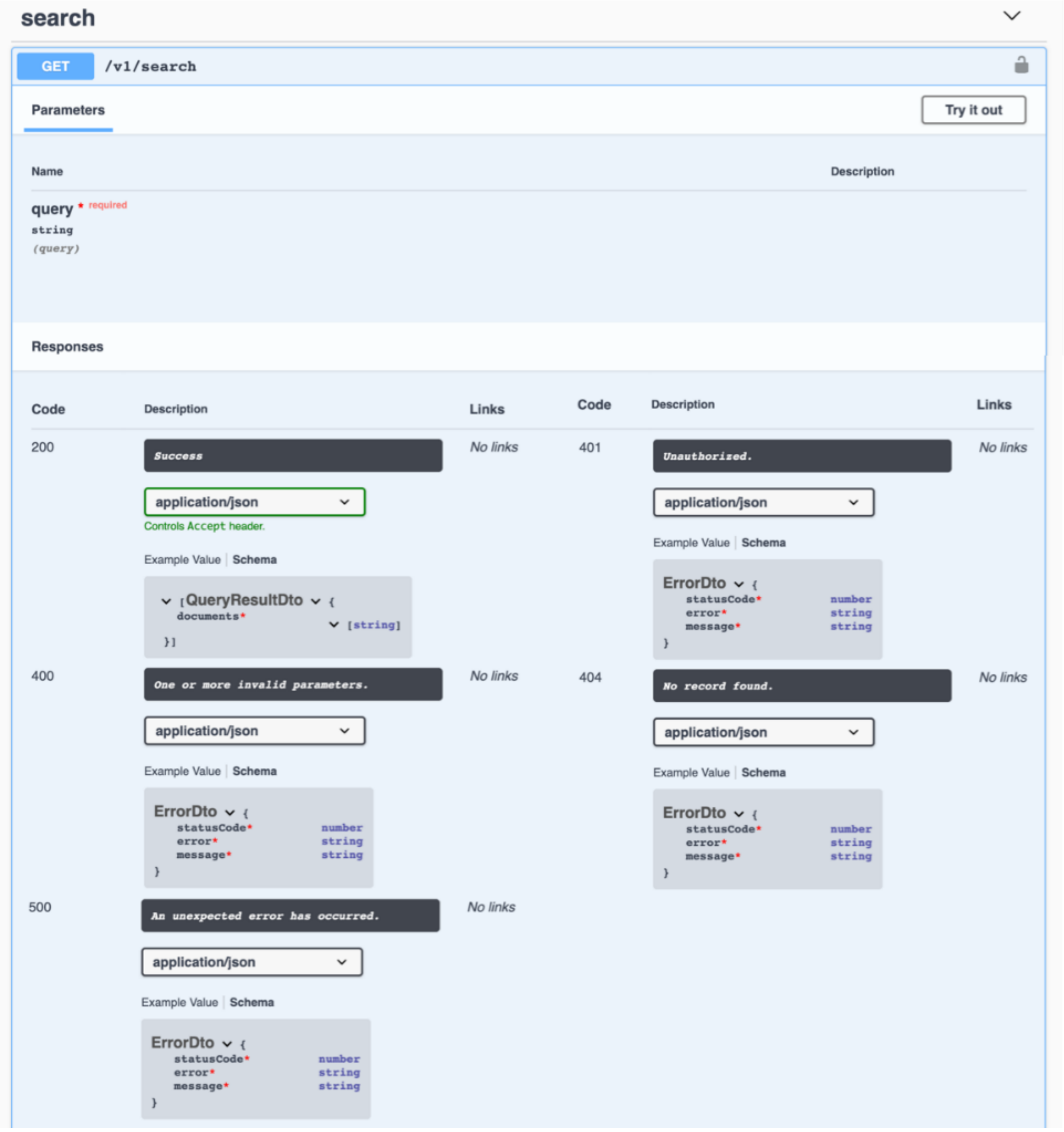
```
ErrorDto {
   statusCode*        number
   error*             string
   message*           string
}
```

```
ErrorDto {
   statusCode*        number
   error*             string
   message*           string
}
```

```
ErrorDto {
   statusCode*        number
   error*             string
   message*           string
}
```

Figure 4. Parse method

Error! Reference source not found. shows the method for previewing how a set of field extraction rules affects a document. This method executes the same rules as in **Error! Reference source not found.**. However, it will not save the parsed document to the data source, since this is used only for previewing the normalizing process.



search

GET /v1/search

Parameters

Try it out

Name	Description
query * required string *(query)*	

Responses

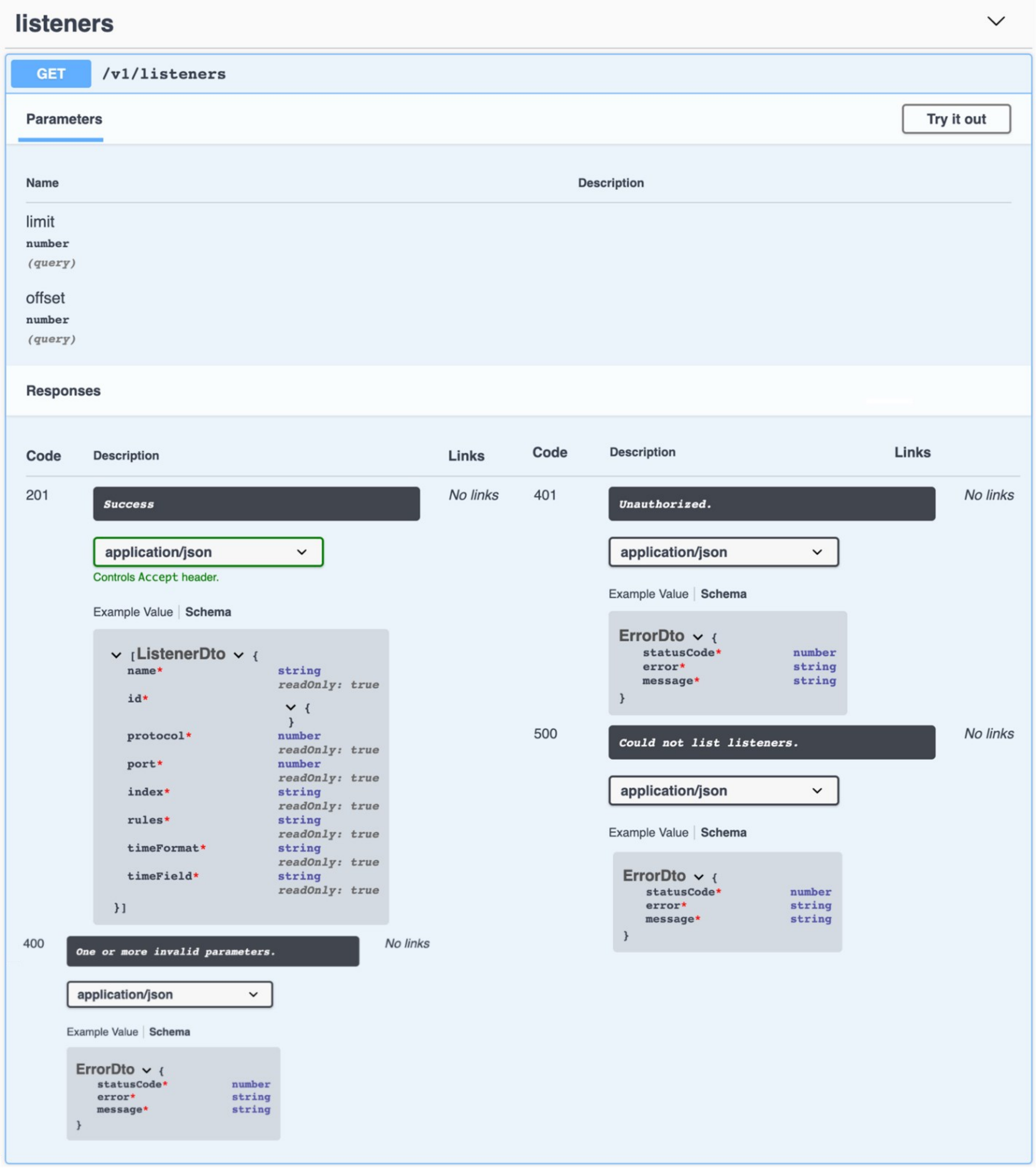
Code	Description	Links
200	*Success* application/json Controls Accept header. Example Value | Schema [QueryResultDto { documents* [string] }]	*No links*
400	*One or more invalid parameters.* application/json Example Value | Schema ErrorDto { statusCode* number, error* string, message* string }	*No links*
500	*An unexpected error has occurred.* application/json Example Value | Schema ErrorDto { statusCode* number, error* string, message* string }	*No links*

Code	Description	Links
401	*Unauthorized.* application/json Example Value | Schema ErrorDto { statusCode* number, error* string, message* string }	*No links*
404	*No record found.* application/json Example Value | Schema ErrorDto { statusCode* number, error* string, message* string }	*No links*

Figure 5. Search method

Figure 5 shows the query method. Based on the informed query, which is a mandatory parameter, this method searches the information in the SIEM databases.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



listeners

GET /v1/listeners

Name	Description
limit number *(query)*	
offset number *(query)*	

Responses

Code	Description	Links
201	*Success*	*No links*
400	*One or more invalid parameters.*	*No links*
401	*Unauthorized.*	*No links*
500	*Could not list listeners.*	*No links*

201: application/json. Controls Accept header. Example Value | Schema

```
∨ [ListenerDto ∨ {
    name*           string
                    readOnly: true
    id*             ∨ {
                    }
    protocol*       number
                    readOnly: true
    port*           number
                    readOnly: true
    index*          string
                    readOnly: true
    rules*          string
                    readOnly: true
    timeFormat*     string
                    readOnly: true
    timeField*      string
                    readOnly: true
}]
```

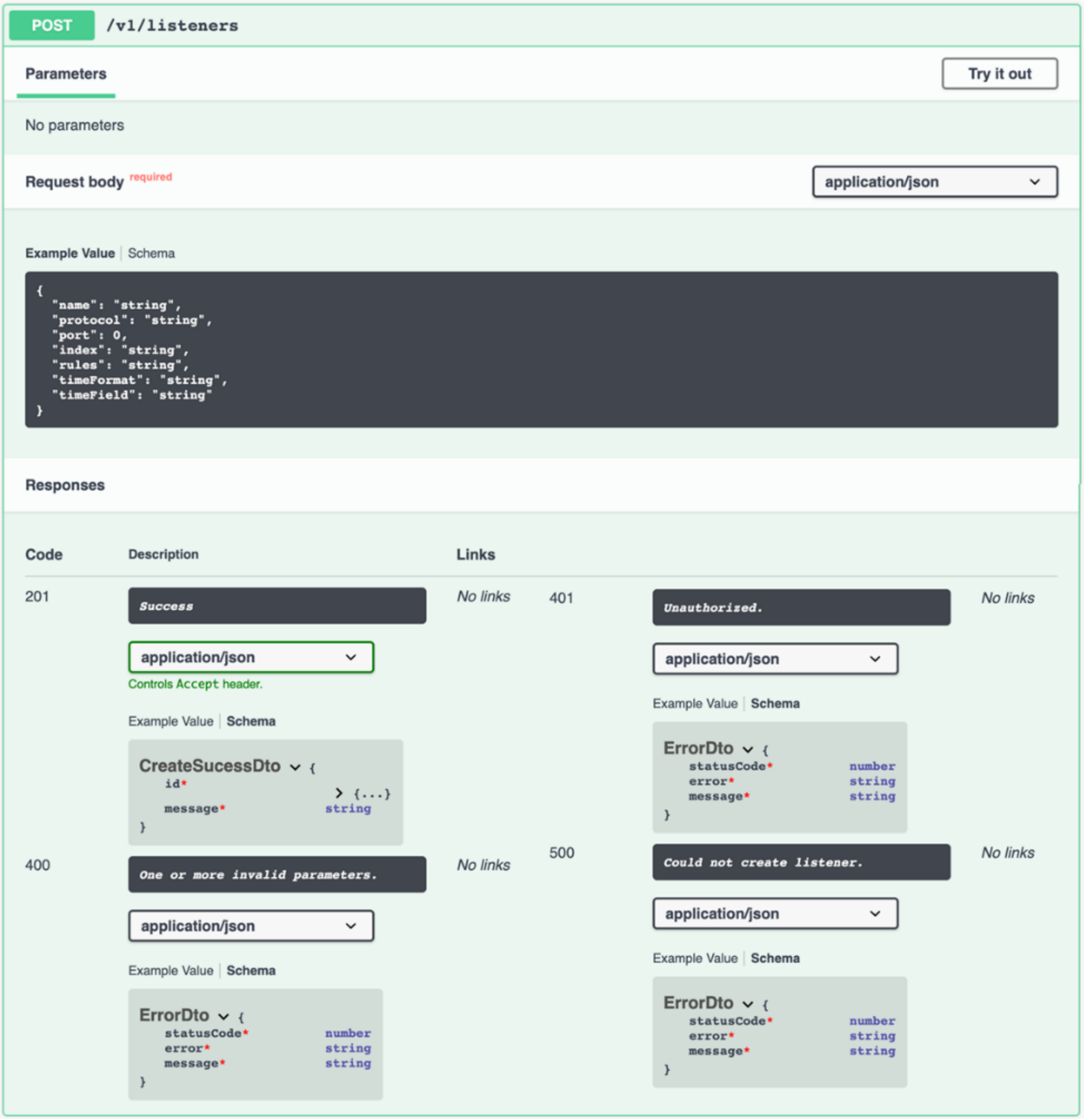
400, 401, 500: application/json. Example Value | Schema

```
ErrorDto ∨ {
    statusCode*     number
    error*          string
    message*        string
}
```

Figure 6. Method for listing all listeners.

Figure 6 shows the method for listing all listeners. Furthermore, the query parameters allow the users to filter the results, and the response sections illustrate the format in which the user should receive the results.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



POST /v1/listeners

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "name": "string",
  "protocol": "string",
  "port": 0,
  "index": "string",
  "rules": "string",
  "timeFormat": "string",
  "timeField": "string"
}
```

Responses

Code	Description	Links
201	Success	No links
400	One or more invalid parameters.	No links
401	Unauthorized.	No links
500	Could not create listener.	No links

201 – application/json

Controls Accept header.

Example Value | Schema

```
CreateSucessDto {
   id*                    > {...}
   message*               string
}
```

400 – application/json

Example Value | Schema

```
ErrorDto {
   statusCode*            number
   error*                 string
   message*               string
}
```

401 – application/json

Example Value | Schema

```
ErrorDto {
   statusCode*            number
   error*                 string
   message*               string
}
```

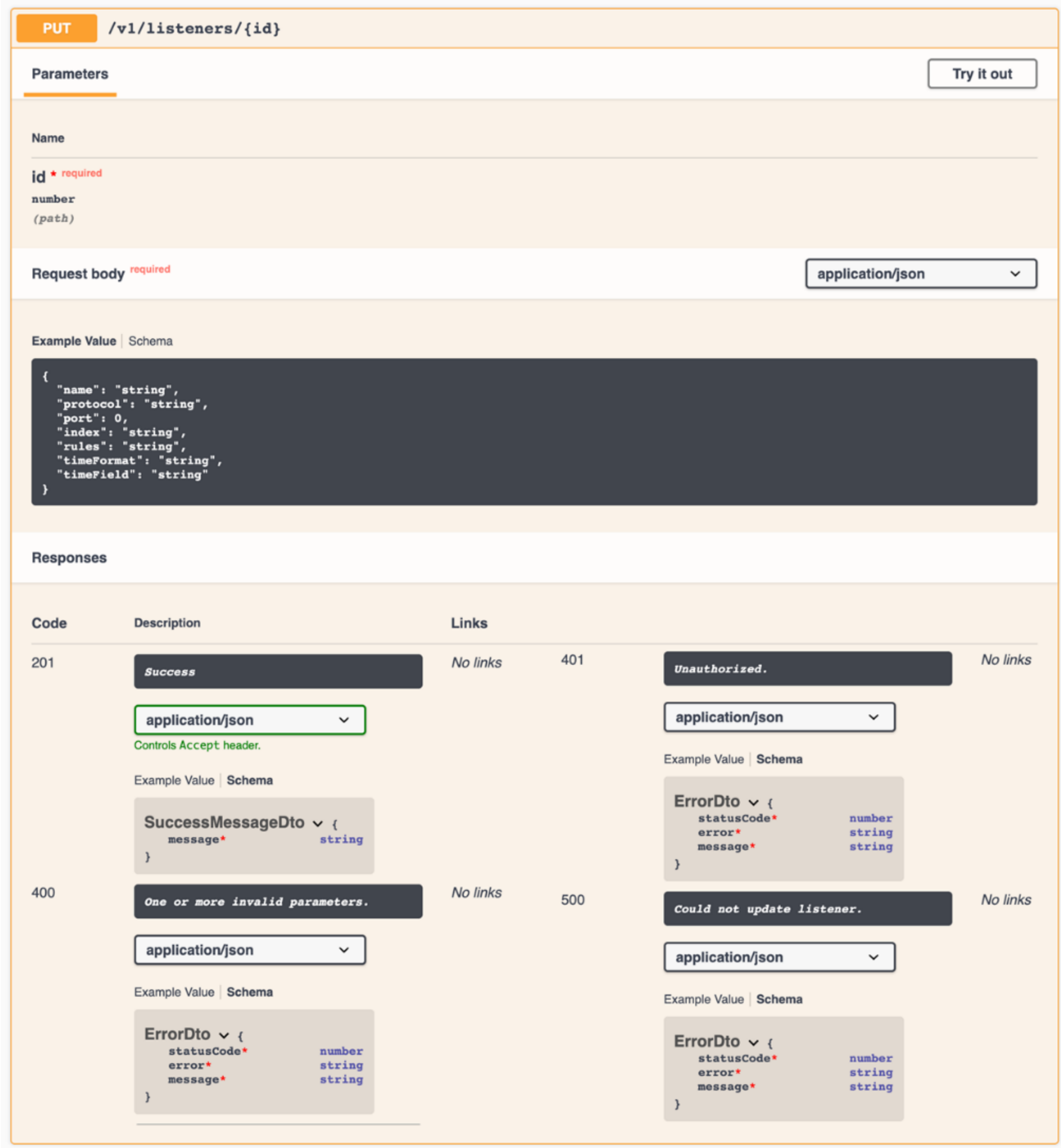
500 – application/json

Example Value | Schema

```
ErrorDto {
   statusCode*            number
   error*                 string
   message*               string
}
```

Figure 7. Method to add listeners

Figure 7 shows the method for adding a new HTTP or TCP listener. This method is responsible for recording the settings in the database and starting the listeners. Whenever the server is restarted, it obtains the configurations on the database to restart the listeners again.



PUT /v1/listeners/{id}

Parameters

Try it out

Name

id * required
number
(path)

Request body required

application/json

Example Value | Schema

```
{
  "name": "string",
  "protocol": "string",
  "port": 0,
  "index": "string",
  "rules": "string",
  "timeFormat": "string",
  "timeField": "string"
}
```

Responses

Code | Description | Links

201

Success

application/json

Controls Accept header.

Example Value | Schema

```
SuccessMessageDto ⌄ {
    message*                 string
}
```

No links

400

One or more invalid parameters.

application/json

Example Value | Schema

```
ErrorDto ⌄ {
    statusCode*              number
    error*                   string
    message*                 string
}
```

No links

401

Unauthorized.

application/json

Example Value | Schema

```
ErrorDto ⌄ {
    statusCode*              number
    error*                   string
    message*                 string
}
```

No links

500

Could not update listener.

application/json

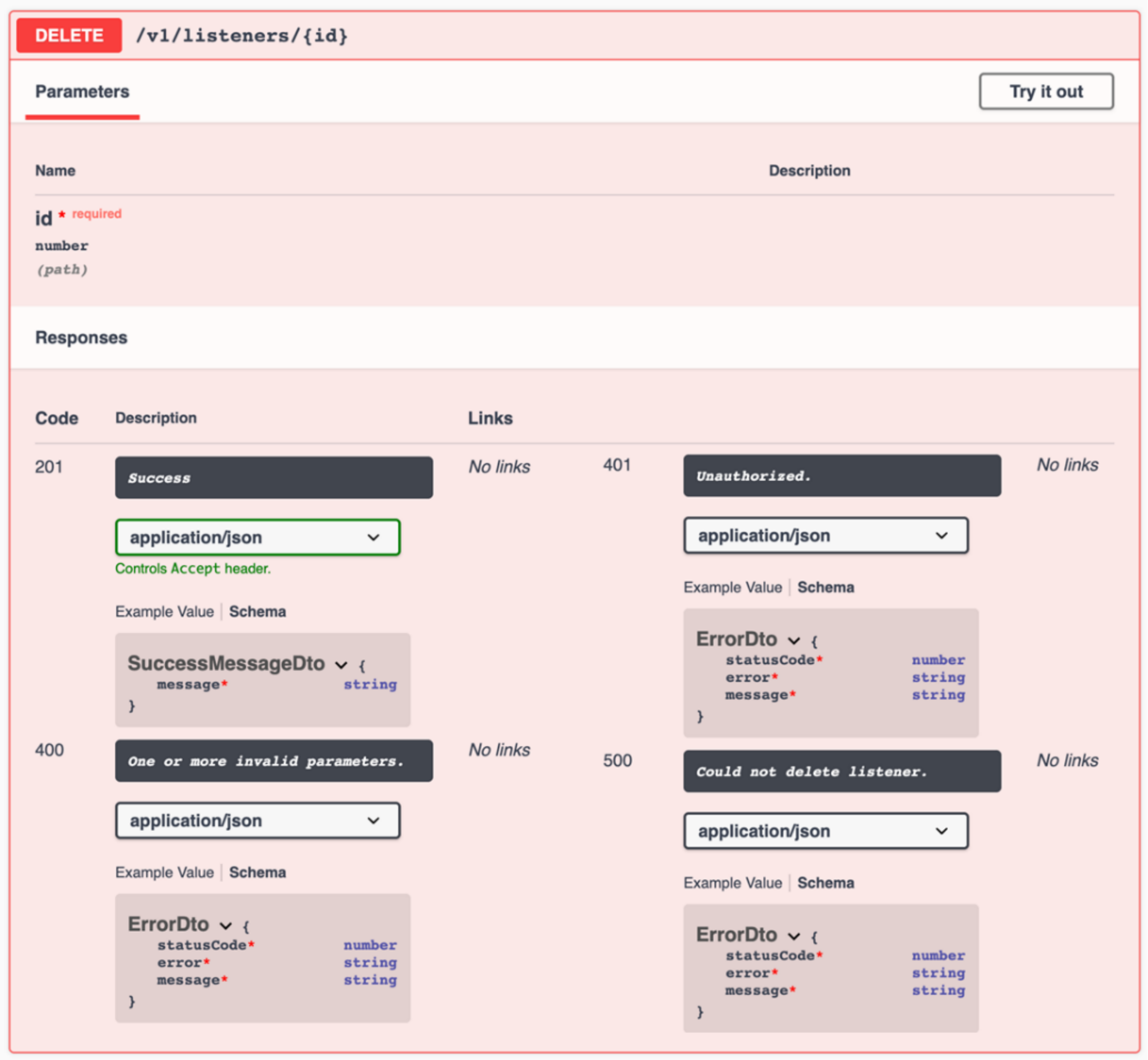
Example Value | Schema

```
ErrorDto ⌄ {
    statusCode*              number
    error*                   string
    message*                 string
}
```

No links

Figure 8. Method for updating listener

Figure 8 shows the API method for updating a HTTP or TCP listener configuration. Upon update, the current listener server is stopped, and immediately restarted, in order to be up-to-date with the recently changed settings.



`/v1/listeners/{id}`

Name	Description
id * required number (path)	

Responses

Code	Description	Links
201	Success application/json Controls Accept header. Example Value | Schema	No links
400	One or more invalid parameters. application/json Example Value | Schema	No links
401	Unauthorized. application/json Example Value | Schema	No links
500	Could not delete listener. application/json Example Value | Schema	No links

201:

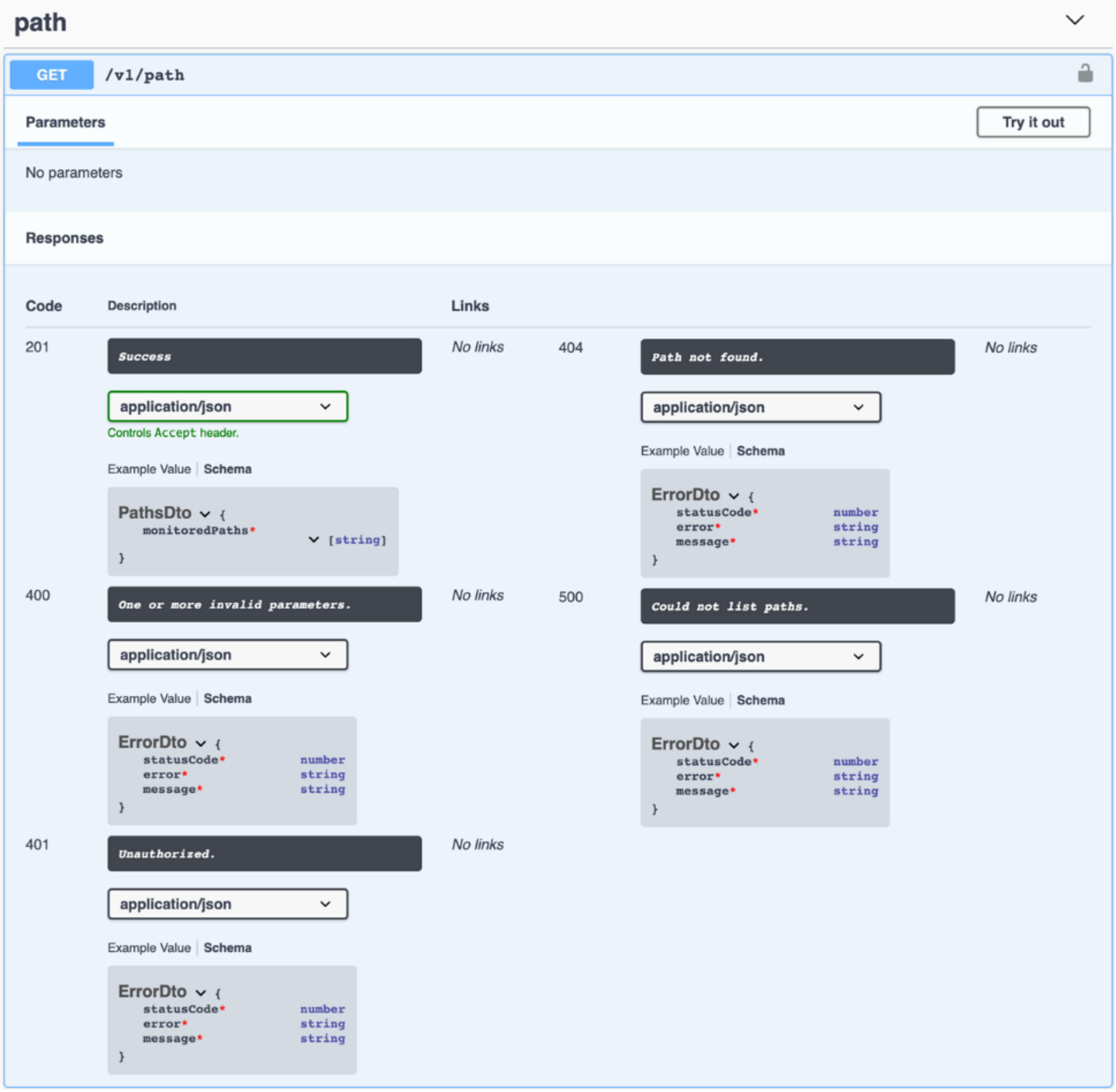
```
SuccessMessageDto {
    message*                 string
}
```

400, 401, 500:

```
ErrorDto {
    statusCode*              number
    error*                   string
    message*                 string
}
```

Figure 9. Method for removing a listener

Figure 9 shows the method for deleting a listener. In addition to removing it from the database, this method is also responsible for shutting down the running listener.



path

GET /v1/path

No parameters

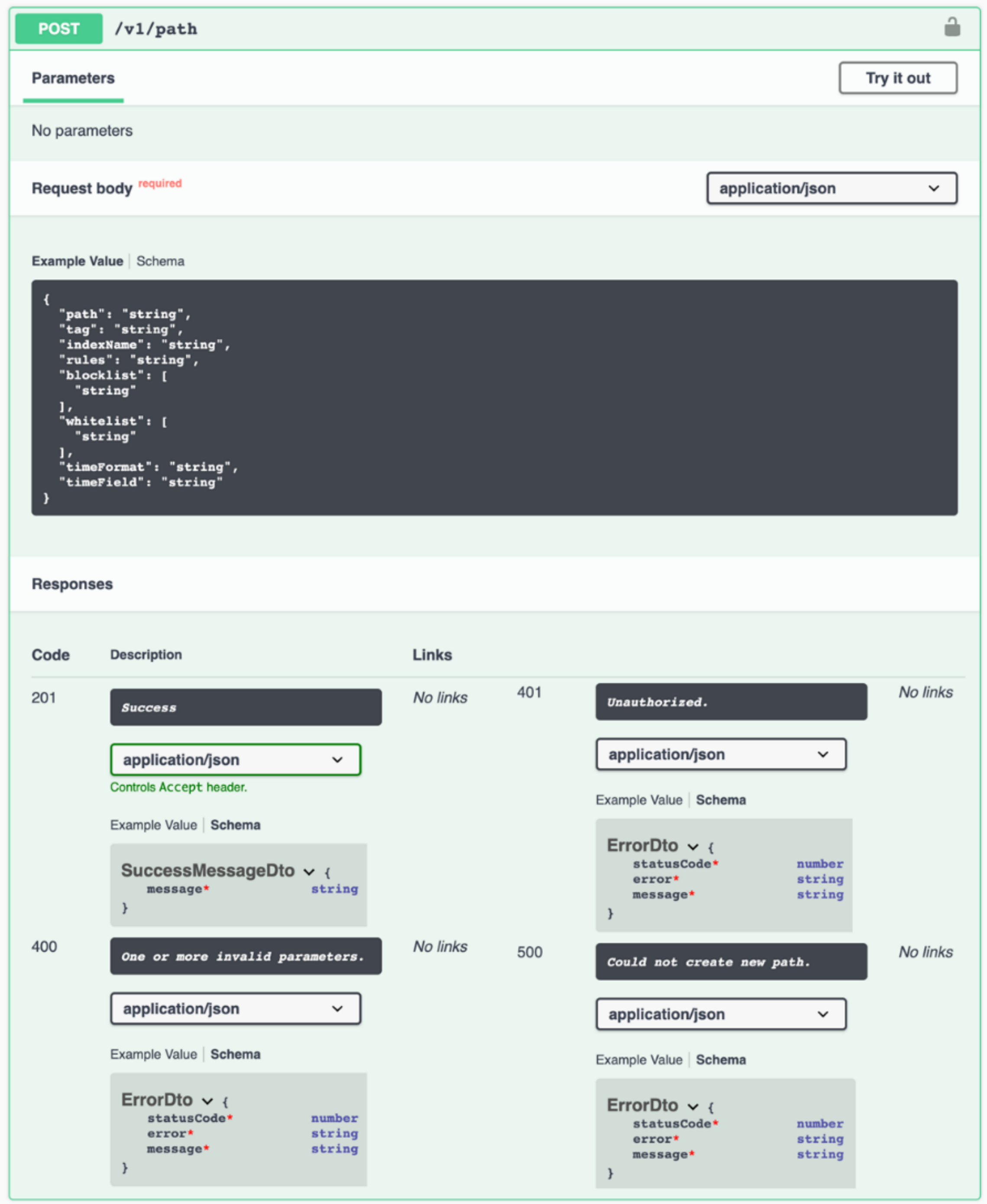
Responses

Code	Description	Links
201	*Success* application/json Controls Accept header. Example Value | Schema PathsDto { monitoredPaths* [string] }	*No links*
400	*One or more invalid parameters.* application/json Example Value | Schema ErrorDto { statusCode* number, error* string, message* string }	*No links*
401	*Unauthorized.* application/json Example Value | Schema ErrorDto { statusCode* number, error* string, message* string }	*No links*
404	*Path not found.* application/json Example Value | Schema ErrorDto { statusCode* number, error* string, message* string }	*No links*
500	*Could not list paths.* application/json Example Value | Schema ErrorDto { statusCode* number, error* string, message* string }	*No links*

Figure 10. Method for listing all monitored paths

Figure 10 shows the method for listing all monitored paths. Furthermore, the query parameters allow the users to filter the results, and the response sections illustrate the format in which the user should receive the results.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



POST /v1/path

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "path": "string",
  "tag": "string",
  "indexName": "string",
  "rules": "string",
  "blocklist": [
    "string"
  ],
  "whitelist": [
    "string"
  ],
  "timeFormat": "string",
  "timeField": "string"
}
```

Responses

Code | Description | Links

201

Success

application/json

Controls Accept header.

Example Value | Schema

```
SuccessMessageDto {
   message*                string
}
```

No links

400

One or more invalid parameters.

application/json

Example Value | Schema

```
ErrorDto {
   statusCode*             number
   error*                  string
   message*                string
}
```

No links

401

Unauthorized.

application/json

Example Value | Schema

```
ErrorDto {
   statusCode*             number
   error*                  string
   message*                string
}
```

No links

500

Could not create new path.

application/json

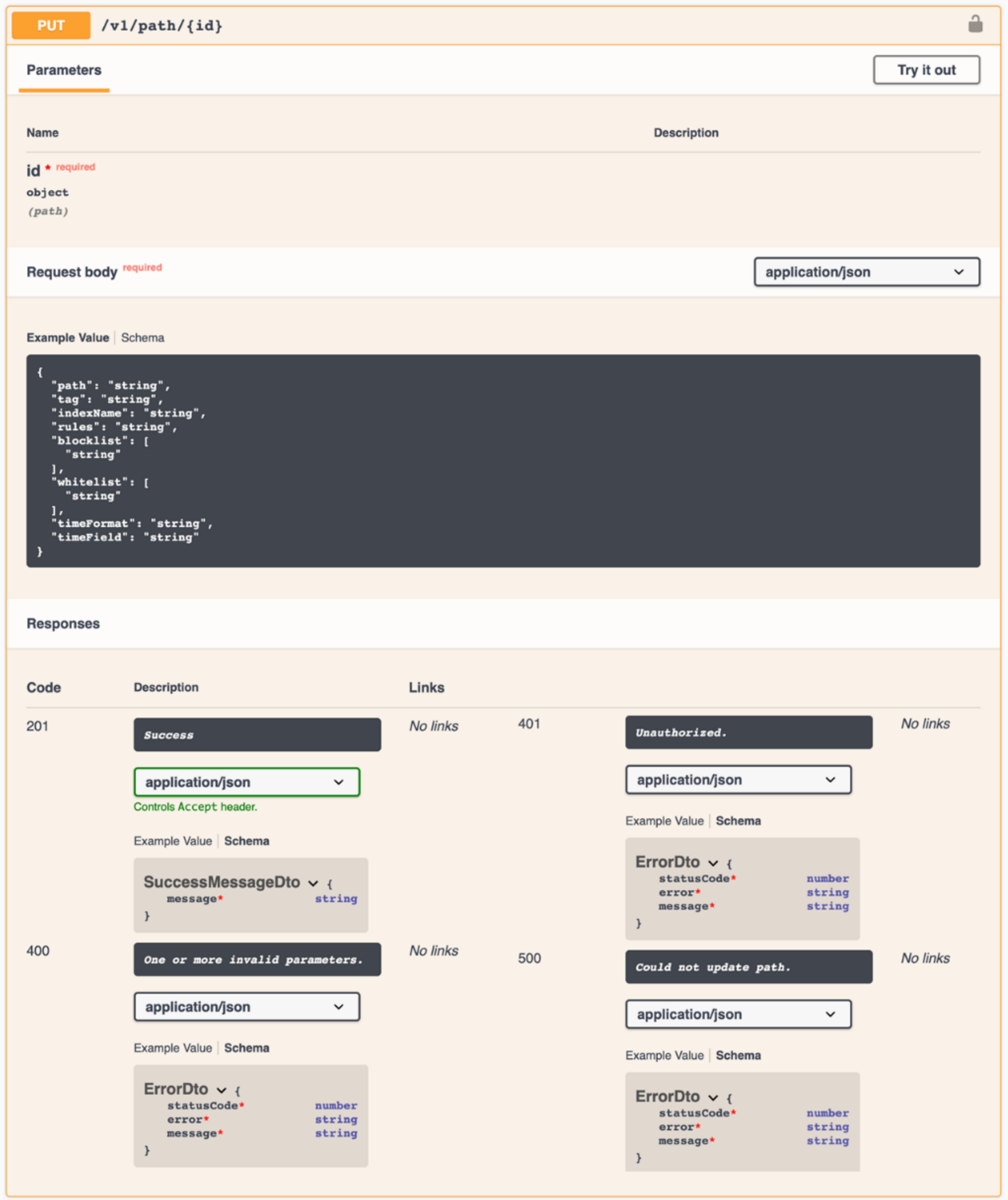
Example Value | Schema

```
ErrorDto {
   statusCode*             number
   error*                  string
   message*                string
}
```

No links

Figure 11. Method to add monitored path

Figure 11 shows the method for monitoring a file and/or path. This method is responsible for adding the file/path on the monitoring process and recording its configurations in the database. Whenever the monitoring process is restarted, it obtains the configurations on the database to restart the processes again.



PUT /v1/path/{id}

Parameters

Try it out

Name	Description
id * required object *(path)*	

Request body required

application/json

Example Value | Schema

```
{
  "path": "string",
  "tag": "string",
  "indexName": "string",
  "rules": "string",
  "blocklist": [
    "string"
  ],
  "whitelist": [
    "string"
  ],
  "timeFormat": "string",
  "timeField": "string"
}
```

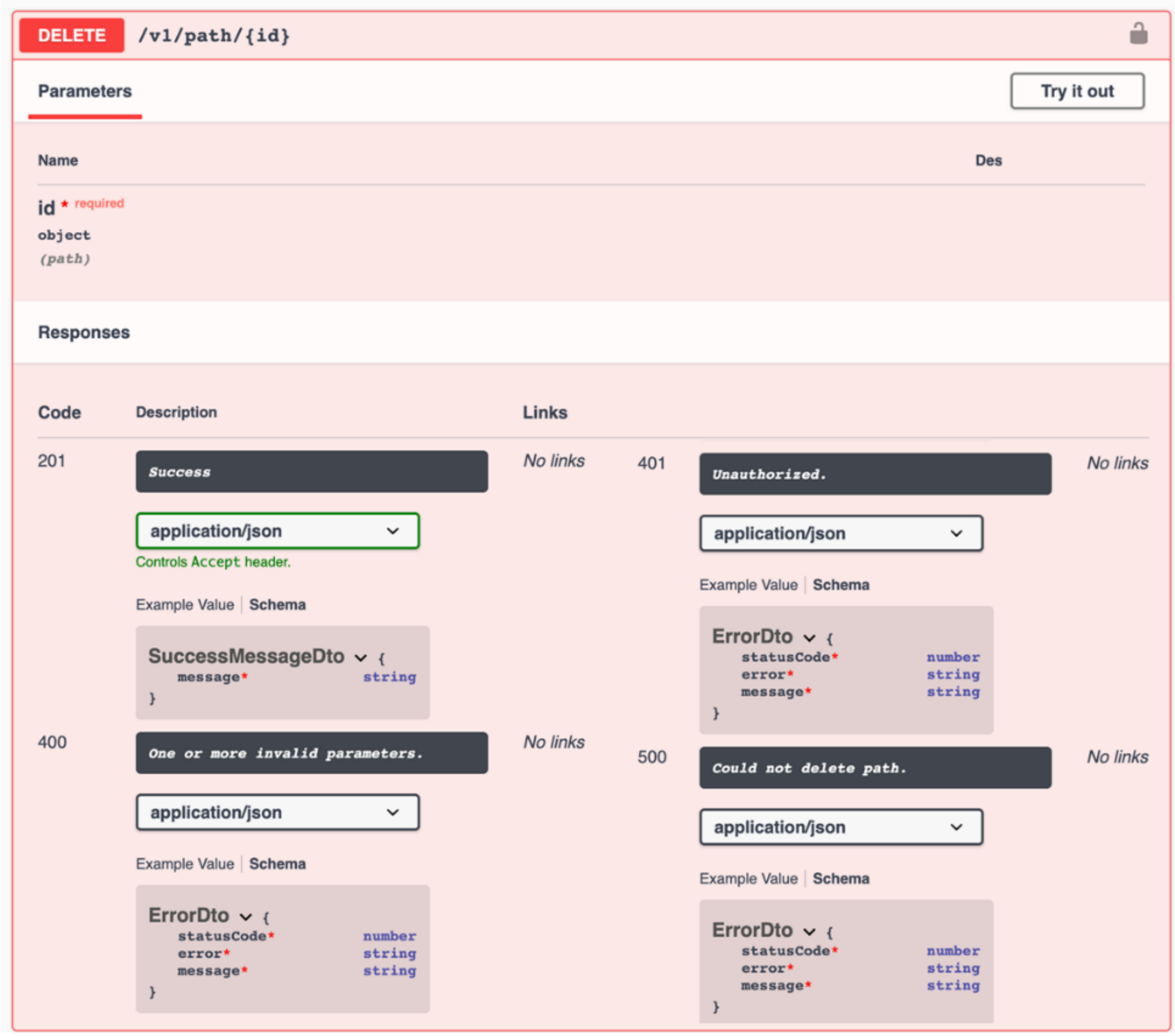
Responses

Code	Description	Links
201	*Success* application/json Controls Accept header. Example Value | Schema SuccessMessageDto { message* string }	*No links*
400	*One or more invalid parameters.* application/json Example Value | Schema ErrorDto { statusCode* number error* string message* string }	*No links*
401	*Unauthorized.* application/json Example Value | Schema ErrorDto { statusCode* number error* string message* string }	*No links*
500	*Could not update path.* application/json Example Value | Schema ErrorDto { statusCode* number error* string message* string }	*No links*

Figure 12. Method for updating a monitored path

Figure 12 shows the method for deleting a monitored path. In addition to updating the database, this method is also responsible for updating the path from the monitoring process that is running.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



DELETE /v1/path/{id}

Name	Des
id * required object (path)	

Responses

Code	Description	Links
201	Success application/json Controls Accept header. Example Value | Schema SuccessMessageDto { message* string }	No links
400	One or more invalid parameters. application/json Example Value | Schema ErrorDto { statusCode* number error* string message* string }	No links
401	Unauthorized. application/json Example Value | Schema ErrorDto { statusCode* number error* string message* string }	No links
500	Could not delete path. application/json Example Value | Schema ErrorDto { statusCode* number error* string message* string }	No links

Figure 13. Method for removing a monitored path

Figure 13 shows the method for deleting a monitored path. In addition to removing it from the database, this method is also responsible for removing the path from the monitoring process that is running.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

2.3.2 Event Manager

The Event Manager subcomponent is responsible for gathering all the queries that are going to run periodically, in order to monitor specific events described in the scheduled query, generate alerts and take actions when abnormal operations are detected. Additionally, this subcomponent saves all query executions and alerts to a database, and all saved data is available through the exposed API listing methods.

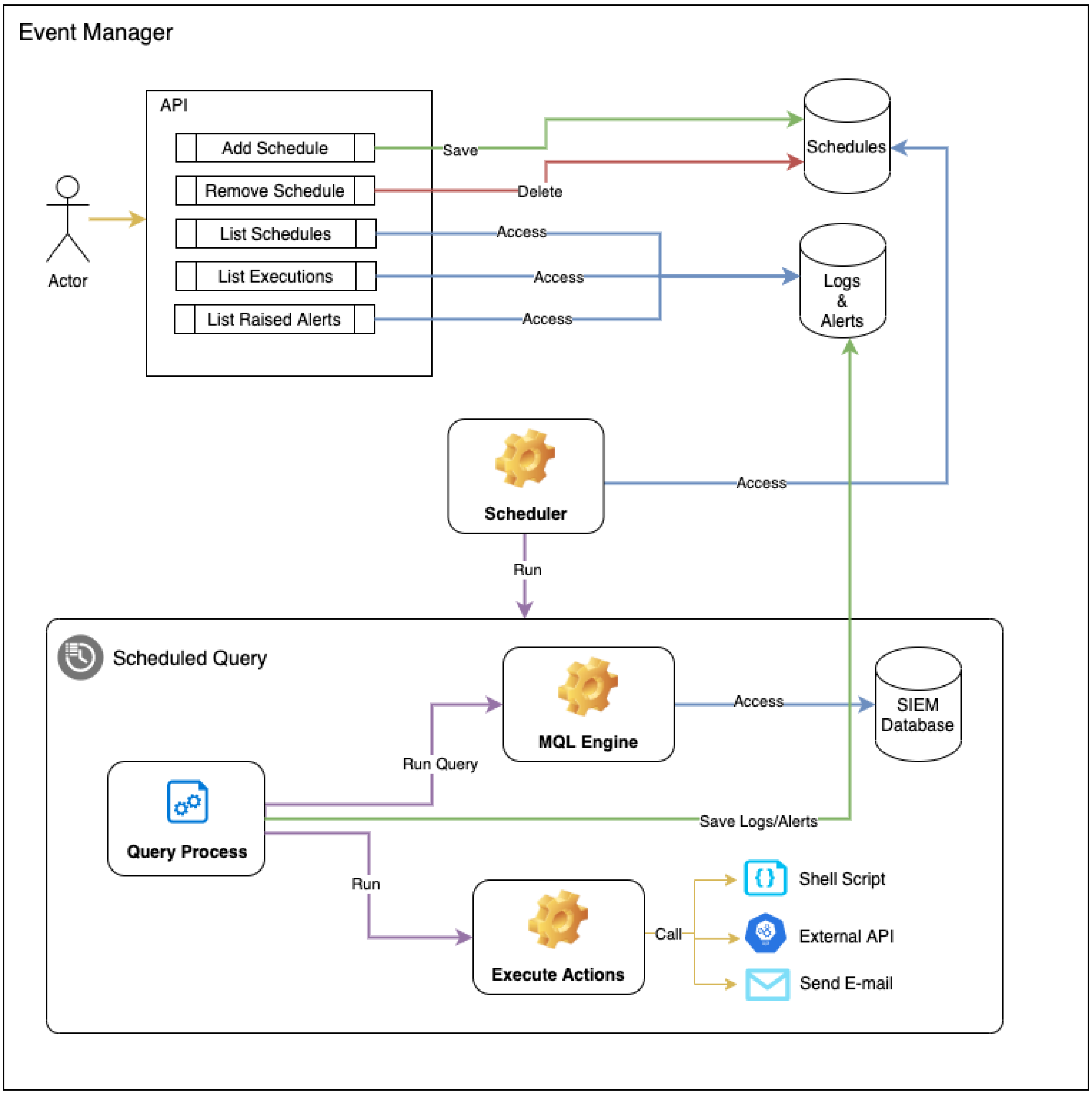


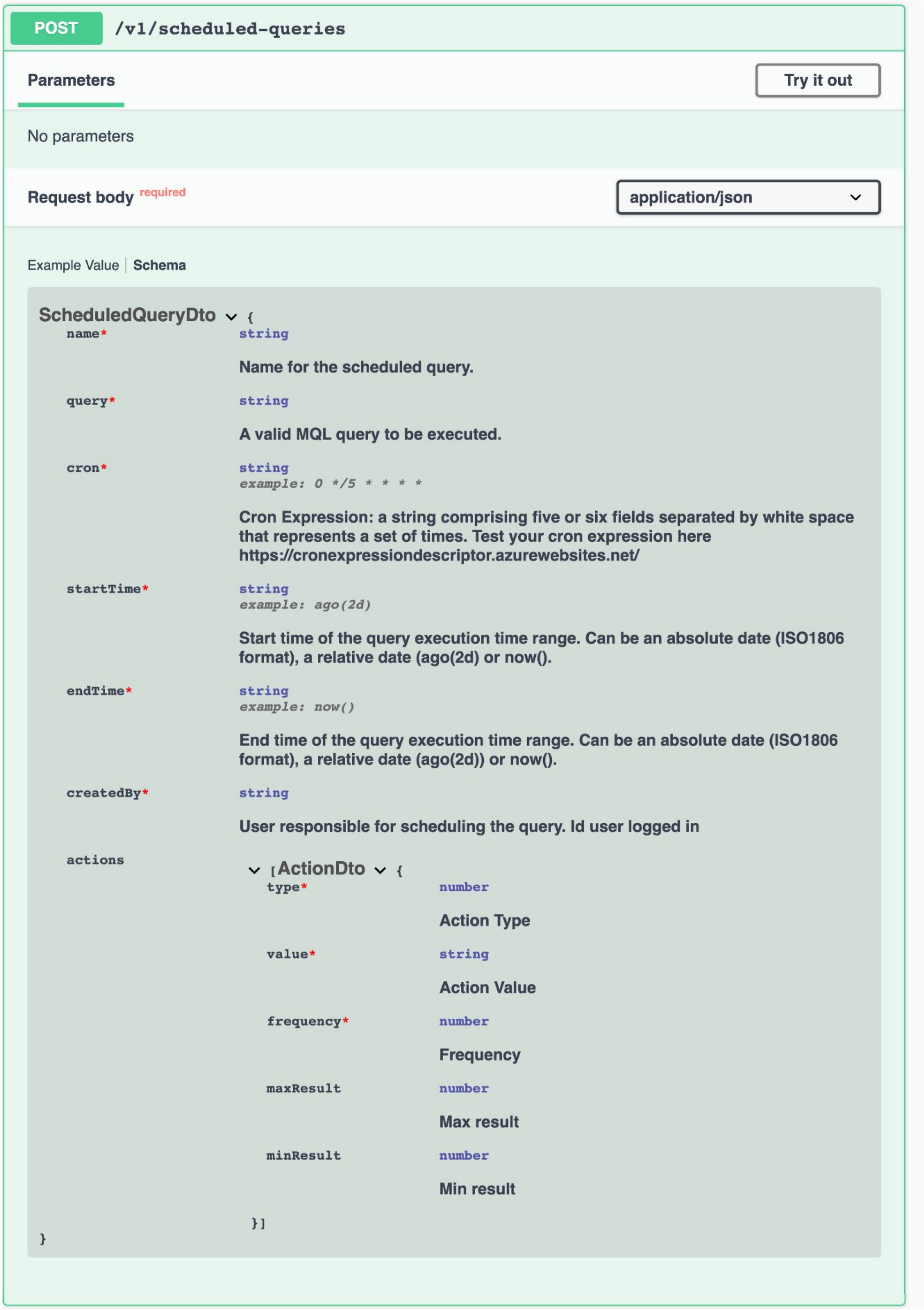
Figure 14. Event Manager Architecture

Figure 14 depicts the architecture designed for the Event Manager subcomponent, with its main activities and possible interactions. The API methods expose all the actions that can be accessed by external components. For instance, when the method ***Add Schedule*** is invoked, a new configuration is added to the database and also to the scheduler process, meaning that the query job will be executed periodically based on what was specified for that schedule. Each execution consists of a call to the MQL Engine for running the query and executing the configured actions for the retrieved results, always following what is configured on the schedule definition. Calling the method ***Remove Schedule*** will exclude the schedule execution workflow. In the following section, the API methods are presented in detail.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

2.3.2.1 Swagger API Methods

This section illustrates the API methods exposed in the Event Manager subcomponent, where the external components can interact with its functionality, mostly for scheduling queries and listing alerts and execution logs.



/v1/scheduled-queries

Parameters

No parameters

Request body required

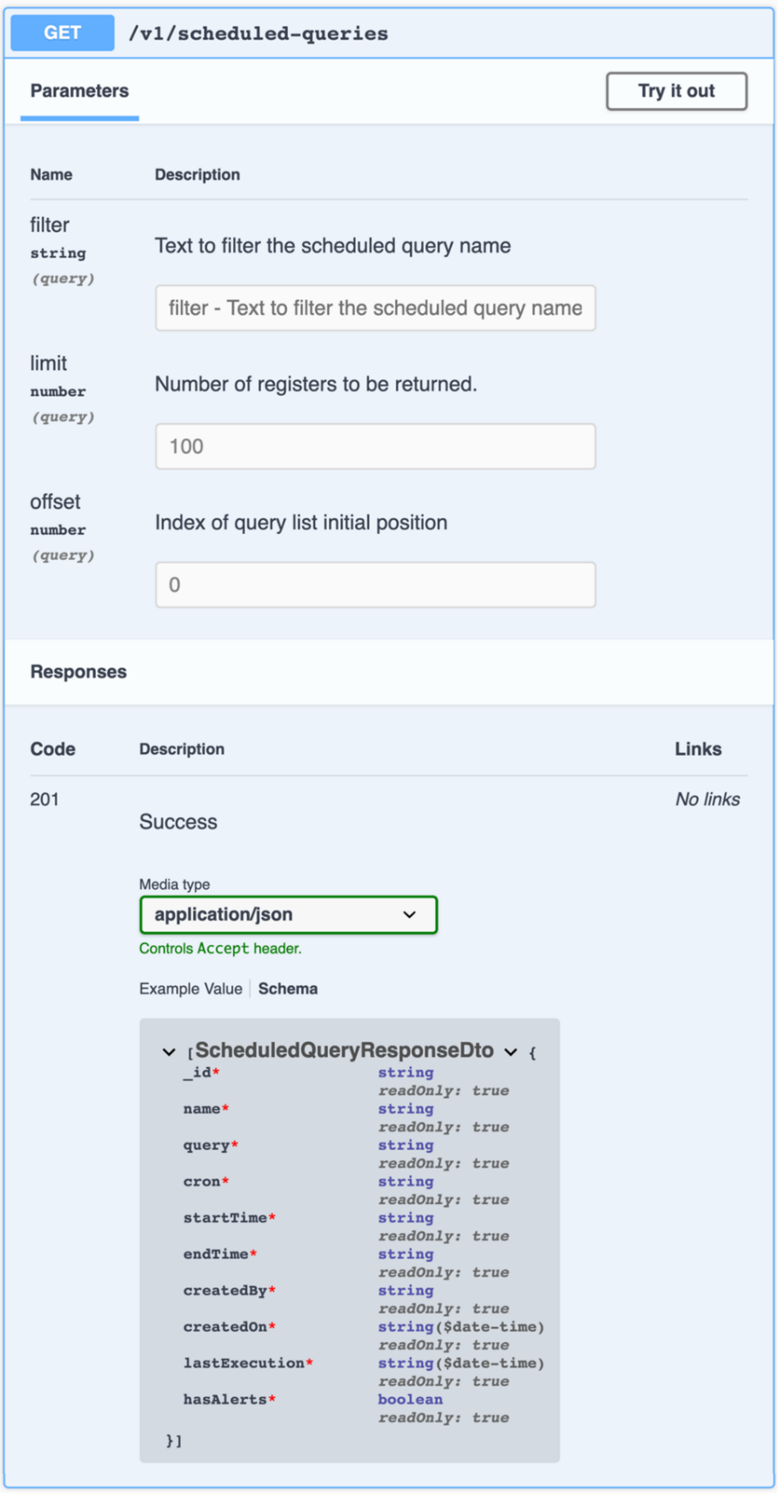
application/json

Example Value | Schema

```
ScheduledQueryDto {
    name*                string
                         Name for the scheduled query.
    query*               string
                         A valid MQL query to be executed.
    cron*                string
                         example: 0 */5 * * * *
                         Cron Expression: a string comprising five or six fields separated by white space
                         that represents a set of times. Test your cron expression here
                         https://cronexpressiondescriptor.azurewebsites.net/
    startTime*           string
                         example: ago(2d)
                         Start time of the query execution time range. Can be an absolute date (ISO1806
                         format), a relative date (ago(2d) or now().
    endTime*             string
                         example: now()
                         End time of the query execution time range. Can be an absolute date (ISO1806
                         format), a relative date (ago(2d)) or now().
    createdBy*           string
                         User responsible for scheduling the query. Id user logged in
    actions                [ActionDto {
                           type*            number
                                            Action Type
                           value*           string
                                            Action Value
                           frequency*       number
                                            Frequency
                           maxResult        number
                                            Max result
                           minResult        number
                                            Min result
                         }]
}
```

Figure 15. Add Schedule POST method

Figure 15 shows the method for scheduling a new query. Moreover, in the Request Body we can see all the required parameters, alongside its description, for adding a new query definition into the Event Manager scheduler process.



GET /v1/scheduled-queries

Parameters

Try it out

Name	Description
filter string *(query)*	Text to filter the scheduled query name filter - Text to filter the scheduled query name
limit number *(query)*	Number of registers to be returned. 100
offset number *(query)*	Index of query list initial position 0

Responses

Code	Description	Links
201	Success	*No links*

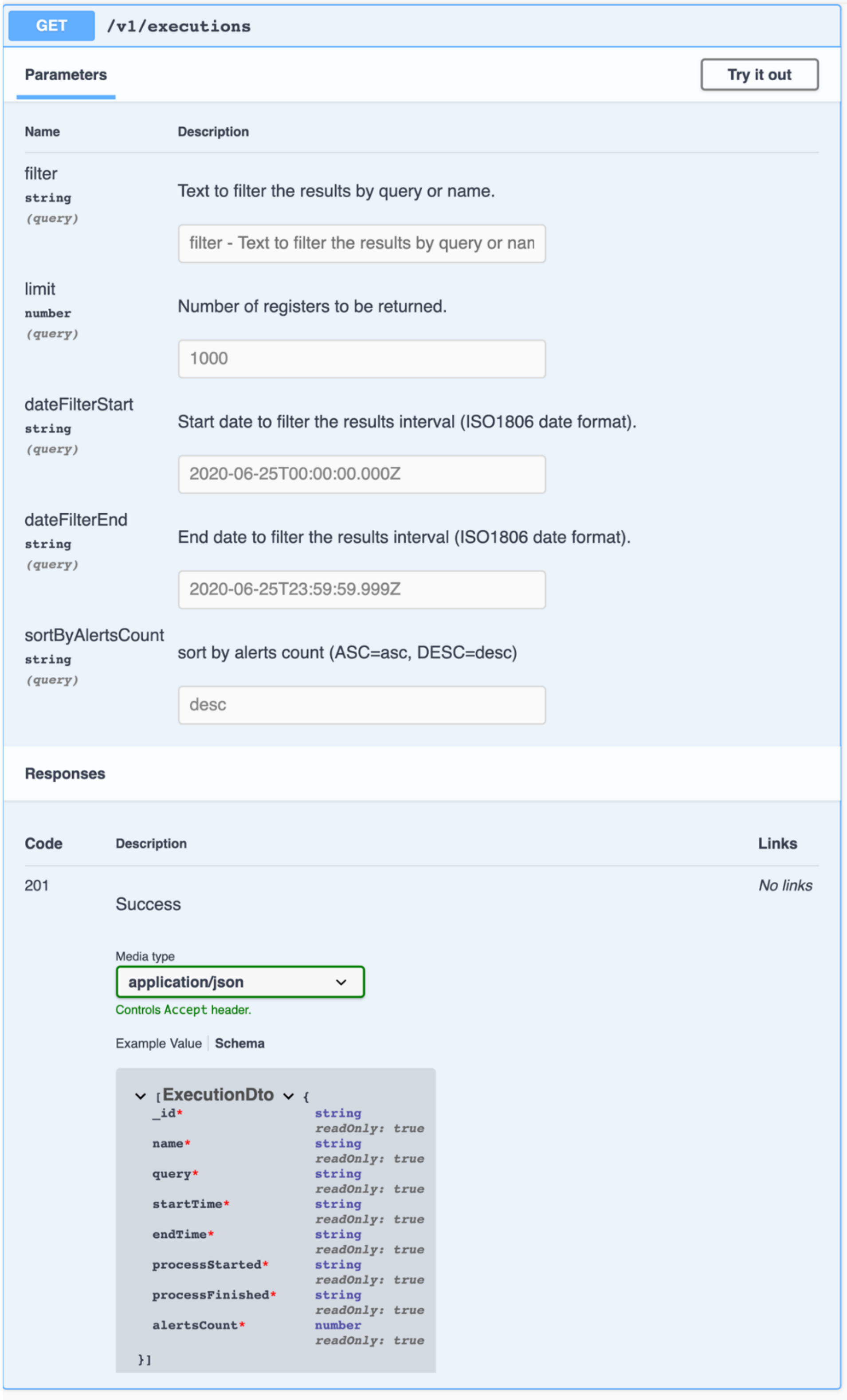
Media type
application/json
Controls Accept header.

Example Value | Schema

```
⌄ [ScheduledQueryResponseDto ⌄ {
   _id*                 string
                        readOnly: true
   name*                string
                        readOnly: true
   query*               string
                        readOnly: true
   cron*                string
                        readOnly: true
   startTime*           string
                        readOnly: true
   endTime*             string
                        readOnly: true
   createdBy*           string
                        readOnly: true
   createdOn*           string($date-time)
                        readOnly: true
   lastExecution*       string($date-time)
                        readOnly: true
   hasAlerts*           boolean
                        readOnly: true
 }]
```

Figure 16. REST method for listing all scheduled queries

Figure 16 shows the REST method for listing all scheduled queries. Furthermore, the query parameters allow the users to filter the results, and the response sections illustrate the format in which the user should receive the results.



/v1/executions

Parameters

Name	Description
filter string *(query)*	Text to filter the results by query or name. filter - Text to filter the results by query or nan
limit number *(query)*	Number of registers to be returned. 1000
dateFilterStart string *(query)*	Start date to filter the results interval (ISO1806 date format). 2020-06-25T00:00:00.000Z
dateFilterEnd string *(query)*	End date to filter the results interval (ISO1806 date format). 2020-06-25T23:59:59.999Z
sortByAlertsCount string *(query)*	sort by alerts count (ASC=asc, DESC=desc) desc

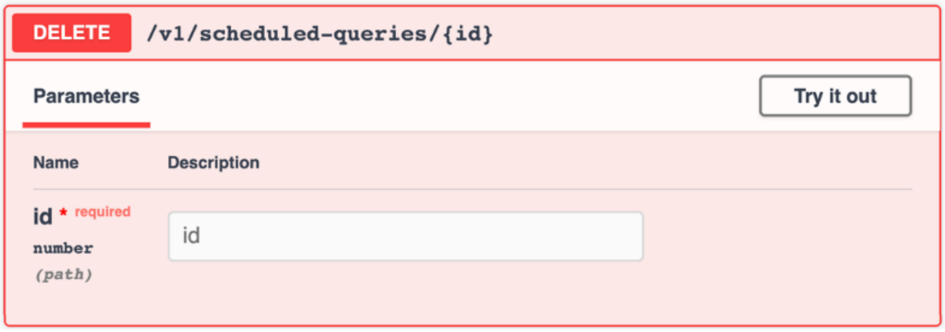
Responses

Code	Description	Links
201	Success Media type application/json Controls Accept header. Example Value | Schema	*No links*

```
[ExecutionDto {
   _id*                string
                       readOnly: true
   name*               string
                       readOnly: true
   query*              string
                       readOnly: true
   startTime*          string
                       readOnly: true
   endTime*            string
                       readOnly: true
   processStarted*     string
                       readOnly: true
   processFinished*    string
                       readOnly: true
   alertsCount*        number
                       readOnly: true
}]
```

Figure 17. API method for listing all Executions Logs

Figure 17 shows the API method for listing all execution logs. In the parameters section, the user can find all the fields they can use to filter the results, and the response section describes the format of a successful request.



DELETE /v1/scheduled-queries/{id}

id * required
number
(path)
id

Figure 18. API method for removing a scheduled query

Figure 18 shows the REST method for deleting a scheduled query. In this case the only mandatory parameter is the scheduled query id.

2.3.3 MQL – Metano Query Language

The MQL component is also the Search Engine of the SPHINX SIEM system. It shares its name with the language used to build the search queries. Furthermore, MQL is inspired by CAL (internal PDM language), SPL (Splunk search processing language) and KQL (Kusto Query Language) and aims to be a high-performing query language for log files. This component has only one method, called Run, which is used to execute the query. This method is divided into four (4) major steps, namely: Interpretation, used for interpreting the query, Get settings, used for loading data sources settings and configurations, Fetch data, used for querying data in the selected data source and finally, Transform data, used for parsing the data based on the criteria informed on the query. In addition, the next lines will give a bit more detail on each of the steps.

Interpretation: In this step, MQL extracts from the query all the information necessary for performing the search, such as data source settings, fields required in the response, filter conditions and even internal methods necessary for filtering and/or transforming the data.

Get settings: From the previous stage, the workflow has the all of the information for connecting to the chosen data source. If in the query the user did not provide the data source settings, MQL searches an internal table with all previously registered data sources and if it is not found, a default database is used.

Fetch data: After being ready to connect to the selected database, the next step is to execute the query search on the selected database and retrieve the results.

Transform data: Finally, in this step, any needed transformation is applied to the retrieved results. It could be a simple filter on the extracted data, as well as an extraction on read, which means extracting new fields from the existing ones using regular expressions. In summary, it can transform the originally-retrieved data, based on what was defined in the search query.

3 SIEM Web – User Interface

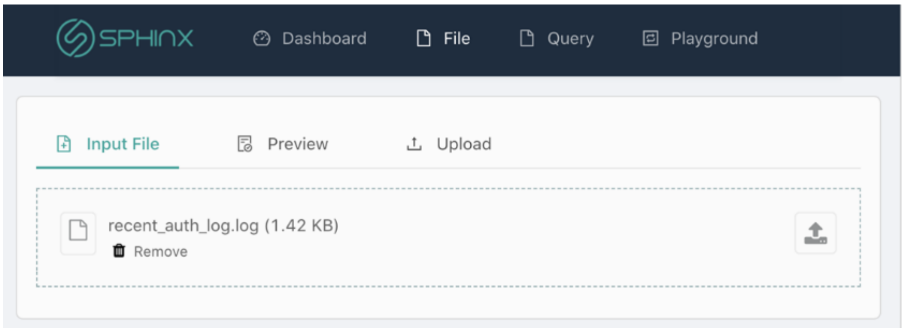
The SIEM Web is a user-friendly graphical interface that was designed to be the SIEM central point, where administrators and analysts can visually interact with the whole SIEM ecosystem. From this tool, the user will be able to define rules for data collection, query the SIEM database, define rules for event correlation and monitoring, watch and list alerts, manage incidents, generate graphs and advanced reports.

3.1 Data Collection

Data collection is a crucial functionality for the whole SIEM ecosystem, consisting of gathering and normalizing data from log files and saving it to the SIEM database. The next sections are going to illustrate the three options available for sending data to the SIEM platform: file upload, path monitoring and port listeners.

3.1.1 File Upload

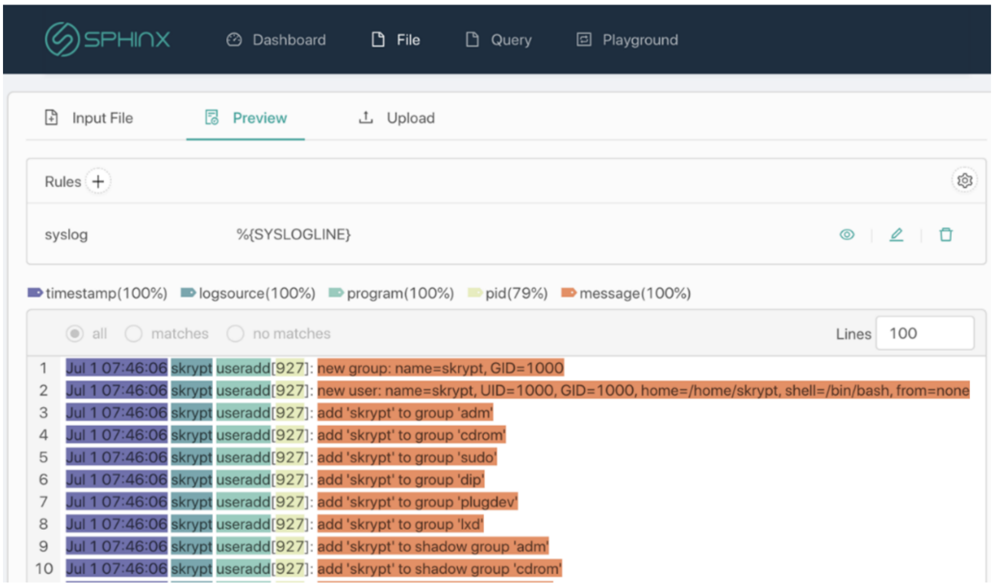
This option consists of manually selecting a single file, adding the parsing rules and submitting it to the SIEM database. The idea behind this option is to provide a simple and easy way to save log data to the platform, even if the user has never used the system before.



recent_auth_log.log (1.42 KB)

Figure 19. File Upload

Figure 19 shows the first page of the File Upload workflow, where the user is expected to select a single file. When a valid file is selected, the user is redirected to the Preview tab, to define one or more rules for parsing the chosen file data.



syslog %{SYSLOGLINE}

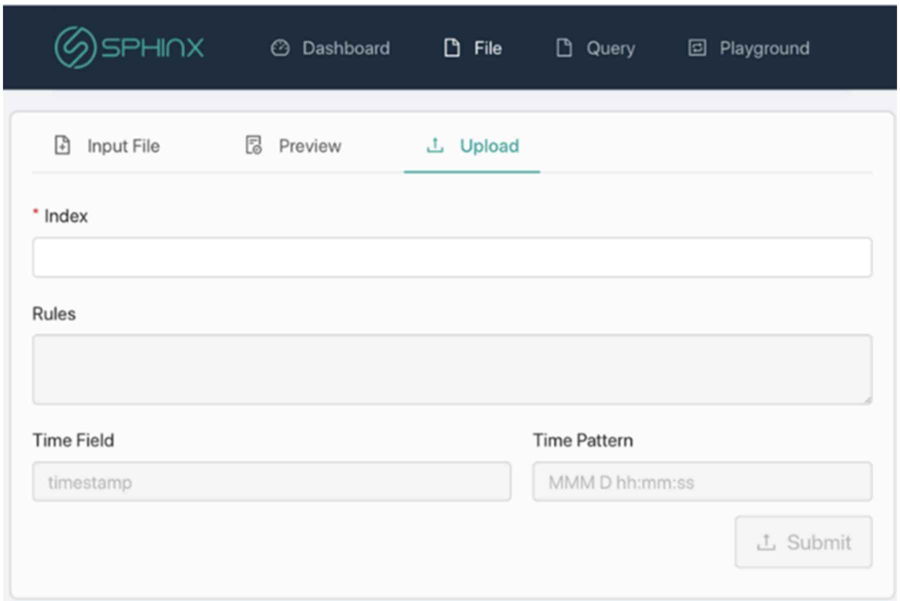
Lines 100

```
Jul 1 07:46:06 skrypt useradd[927]: new group: name=skrypt, GID=1000
Jul 1 07:46:06 skrypt useradd[927]: new user: name=skrypt, UID=1000, GID=1000, home=/home/skrypt, shell=/bin/bash, from=none
Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'adm'
Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'cdrom'
Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'sudo'
Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'dip'
Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'plugdev'
Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'lxd'
Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to shadow group 'adm'
Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to shadow group 'cdrom'
```

Figure 20. Log file normalization preview

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

The Preview page, illustrated in Figure 20, is where the user can configure rules, based on Grok expressions and/or regular expressions, to extract fields from the file lines. After defining which rules they want to apply, the system gives instant feedback for all matches found, including some statistics on the fields extracted, for example, the number of occurrences for each field. Additionally, the user can add and customize as many rules as they see fit.



* Index

Rules

Time Field: timestamp

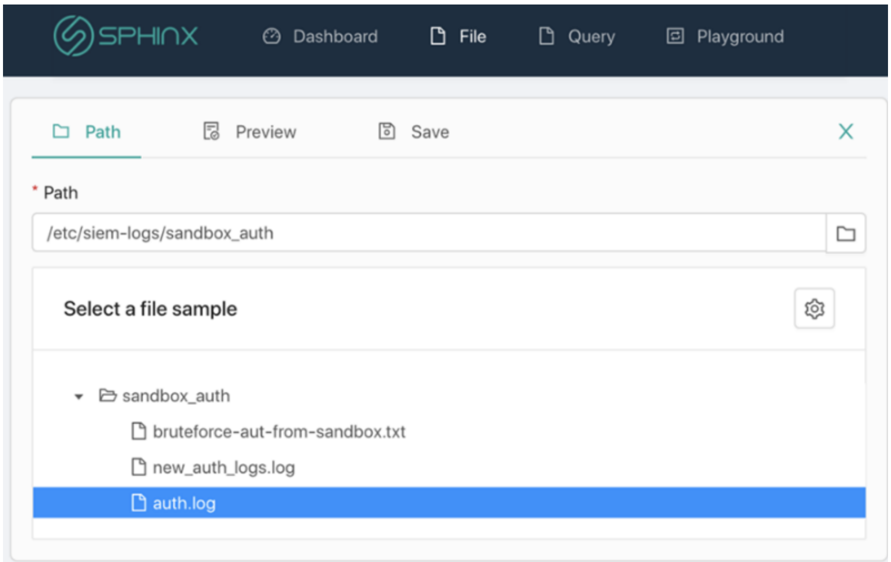
Time Pattern: MMM D hh:mm:ss

Figure 21. File Upload Summary - last step prior to submission

Figure 21 shows the upload page, where the user can see a summary of the options and parsing rules selected to be applied on the file. Also, on this page, it is mandatory to select an index on the database where the parsed files will be saved to. This third step concludes the File Upload workflow.

3.1.2 Path Monitoring

The Path Monitoring method is an option where the user is presented with a directory, hosted on the server side, from where they can pick a file, directory or subdirectory that they want to monitor. In the monitoring definitions, the user can set up the parsing rules they wish to apply, and the index where the normalized data will be saved to.



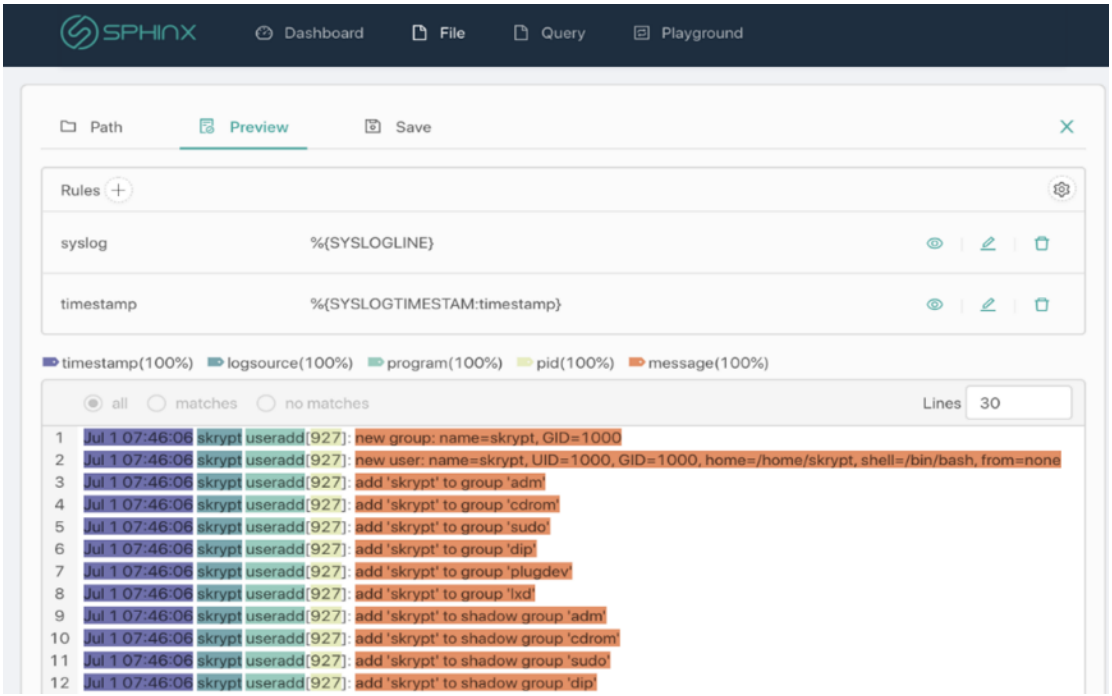
* Path: /etc/siem-logs/sandbox_auth

Select a file sample

- sandbox_auth
  - bruteforce-aut-from-sandbox.txt
  - new_auth_logs.log
  - auth.log

Figure 22. Path and file sample selection

Figure 22 displays the first tab of the Path Monitoring workflow, where a directory is selected, and from that directory, the user picks one file as sample to visualize the parsing rules and extra configuration they wish to apply on those files.



Rules	
syslog	%{SYSLOGLINE}
timestamp	%{SYSLOGTIMESTAM:timestamp}

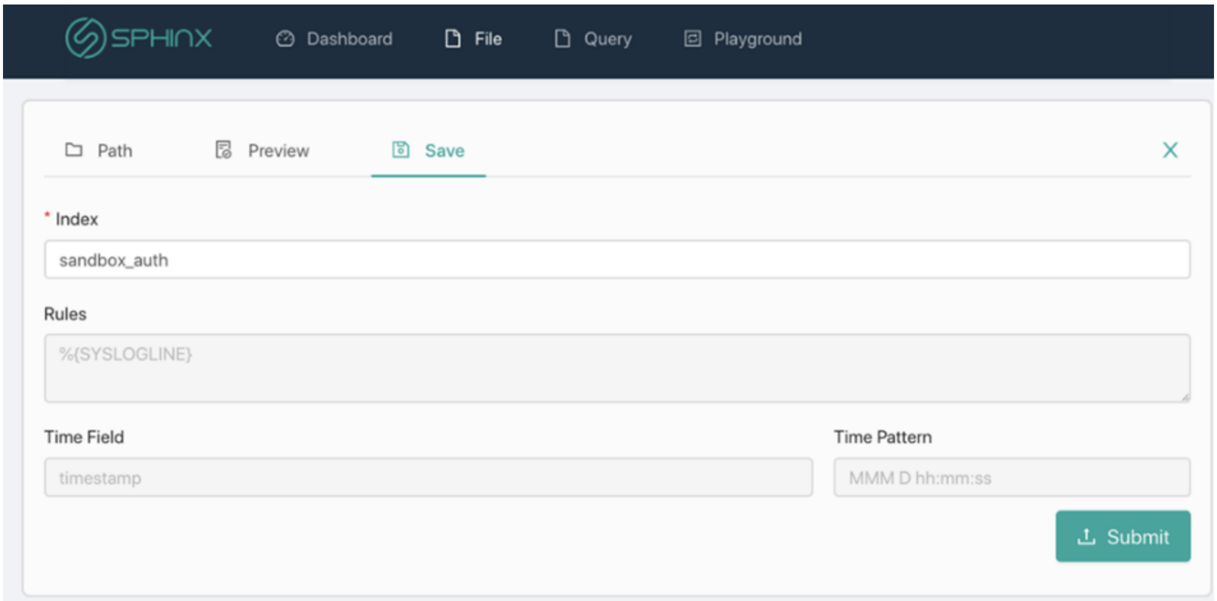
timestamp(100%) logsource(100%) program(100%) pid(100%) message(100%)

Lines 30

```
1  Jul 1 07:46:06 skrypt useradd[927]: new group: name=skrypt, GID=1000
2  Jul 1 07:46:06 skrypt useradd[927]: new user: name=skrypt, UID=1000, GID=1000, home=/home/skrypt, shell=/bin/bash, from=none
3  Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'adm'
4  Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'cdrom'
5  Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'sudo'
6  Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'dip'
7  Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'plugdev'
8  Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to group 'lxd'
9  Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to shadow group 'adm'
10 Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to shadow group 'cdrom'
11 Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to shadow group 'sudo'
12 Jul 1 07:46:06 skrypt useradd[927]: add 'skrypt' to shadow group 'dip'
```

Figure 23. Log file normalization preview

The Preview component works the same way for any data collection method. As shown in Figure 23, the user can configure rules selected from a pre-set, or even add custom rules, based on Grok expressions and/or regular expressions, to extract fields from the file lines. After defining which rules they want to apply, the system gives instant feedback for all matches found, including some statistics on the fields extracted, for example, the number of occurrences for each field. Additionally, the user can add and customize as many rules as they see fit.



* Index: sandbox_auth

Rules: %{SYSLOGLINE}

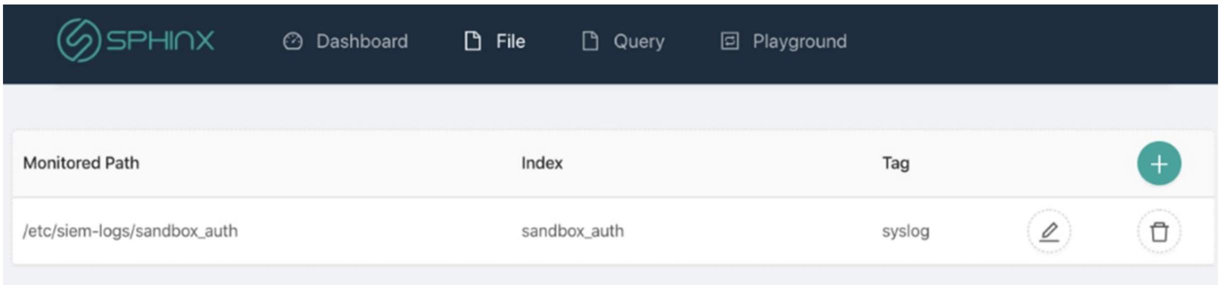
Time Field: timestamp

Time Pattern: MMM D hh:mm:ss

Figure 24. Path monitoring – summary and submission

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

The Save tab, as shown in Figure 24, is the last step for configuring a new Path Monitoring process. At this stage, the user can see a summary of the selected rules and extra details and must provide the index name where the normalized data will be saved to.



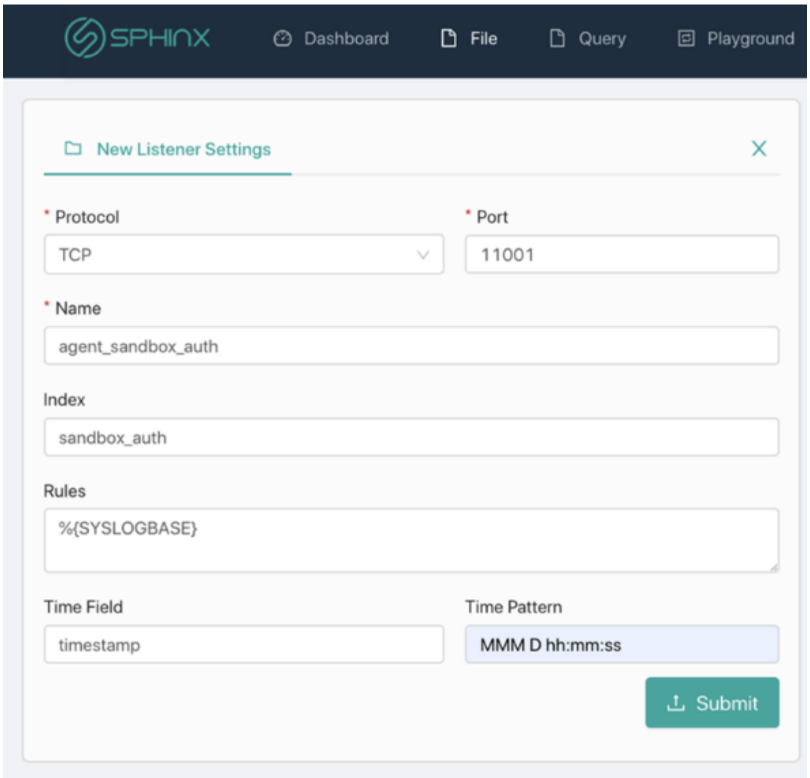
Monitored Path	Index	Tag
/etc/siem-logs/sandbox_auth	sandbox_auth	syslog

Figure 25. Path Monitoring List

After submitting the new item, the Path Monitored List is updated with the recently added item, as can be seen in Figure 25. From this list, the user can edit or delete the existing configurations. To sum up the options presented in this section, whenever any change is detected on the monitored file or directory, the defined set of rules are applied to the new lines, and the normalized new lines are saved to the selected index.

3.1.3 Listeners

Listeners are used to insert the logs collected on the SPHINX components into the SIEM database. So, configuring a listener is basically defining a channel where the components will put through the data collected on each component. Moreover, all data will be normalized according to the pre-defined rules, and then inserted into the SIEM database.

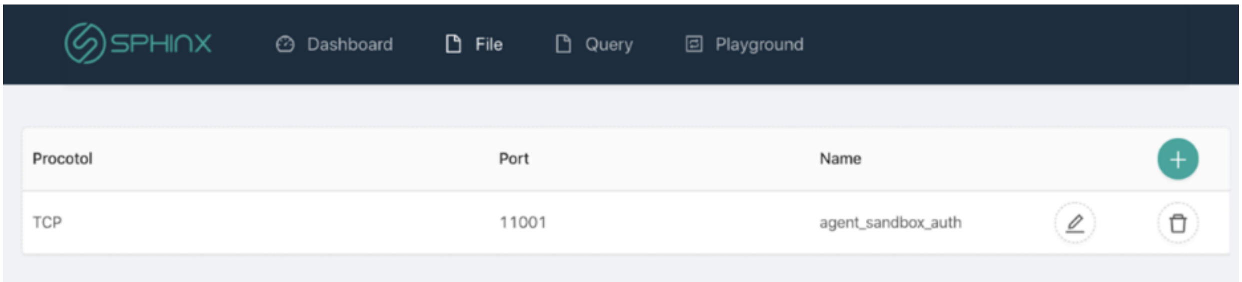


* Protocol	TCP
* Port	11001
* Name	agent_sandbox_auth
Index	sandbox_auth
Rules	%{SYSLOGBASE}
Time Field	timestamp
Time Pattern	MMM D hh:mm:ss

Figure 26. New Listener Form

Figure 26 exemplifies what fields are mandatory to create a new listener that will work as a channel to ingest data collected from external components into the SIEM platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



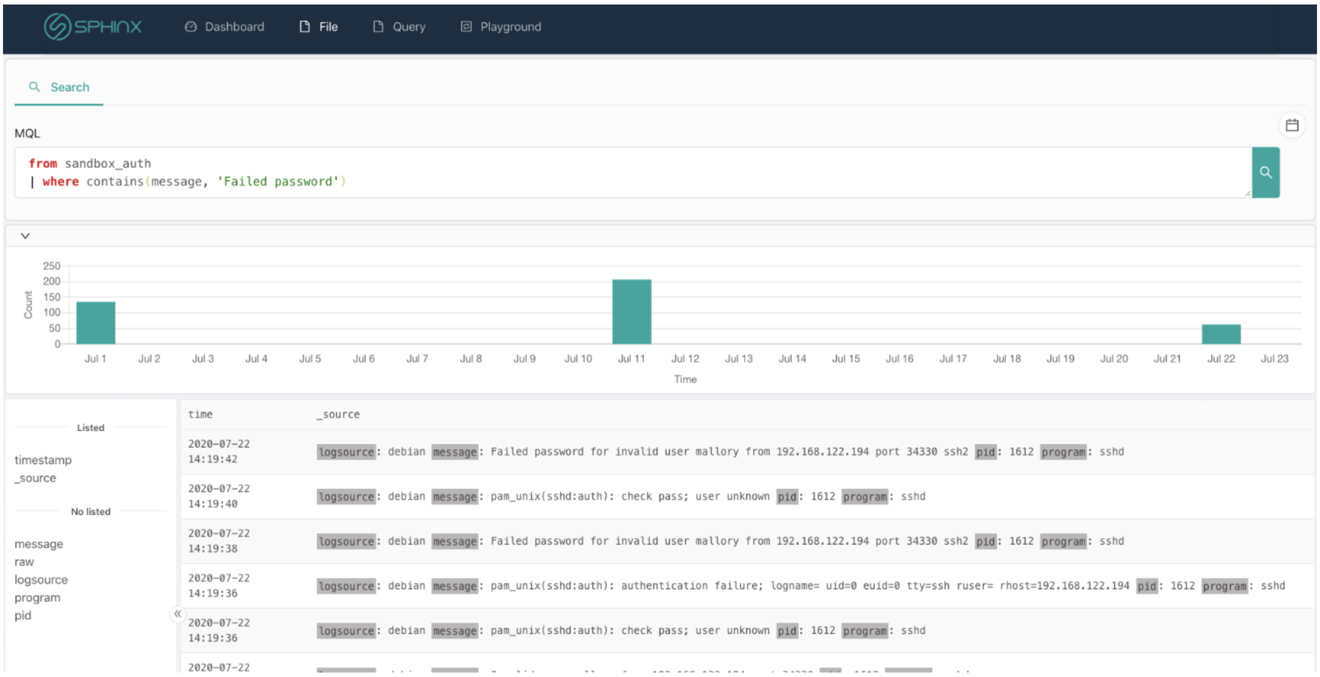
Procotol	Port	Name
TCP	11001	agent_sandbox_auth

Figure 27. Listeners List

From the Listeners List, as shown in Figure 27, the user can edit or remove the available listeners. Once a new listener is created, and is up and running, the agents installed on the SPHINX components can use them to send data through the network following the information configured on the listener settings.

3.2 Search

This functionality implements a query interface where users can run MQL (Metano Query Language) queries on the SIEM database in order to distinguish between normal and abnormal operations. MQL is inspired by CAL (internal PDM language), SPL (Splunk search processing language) and KQL (Kusto Query Language) and aims to be a high performing query language for log files. This feature will be the entry point from where the user will define which events they wish to find and monitor. Once they know the search query works and brings the expected results, they can move on to the scheduling stage, which is going to be discussed in the next section.



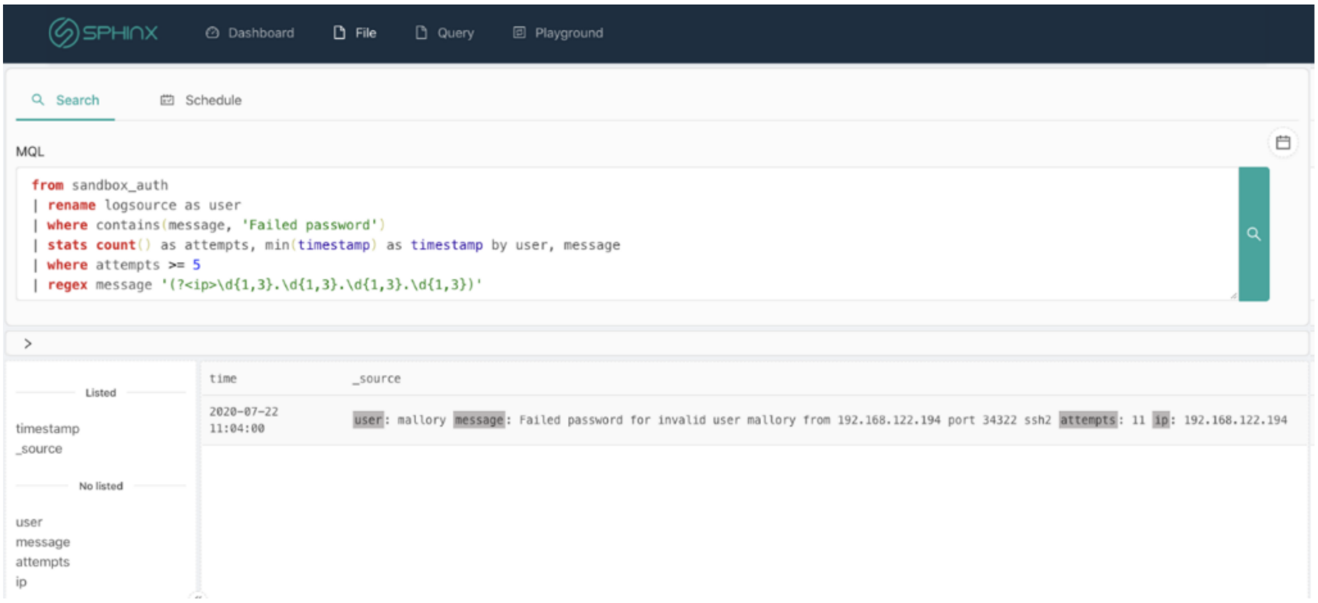
```
from sandbox_auth
| where contains(message, 'Failed password')
```

time	_source
2020-07-22 14:19:42	logsource: debian message: Failed password for invalid user mallory from 192.168.122.194 port 34330 ssh2 pid: 1612 program: sshd
2020-07-22 14:19:40	logsource: debian message: pam_unix(sshd:auth): check pass; user unknown pid: 1612 program: sshd
2020-07-22 14:19:38	logsource: debian message: Failed password for invalid user mallory from 192.168.122.194 port 34330 ssh2 pid: 1612 program: sshd
2020-07-22 14:19:36	logsource: debian message: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=192.168.122.194 pid: 1612 program: sshd
2020-07-22 14:19:36	logsource: debian message: pam_unix(sshd:auth): check pass; user unknown pid: 1612 program: sshd
2020-07-22	

Figure 28. Search Page: MQL form

As can be seen in Figure 28, the user can use MQL to build queries, on a wide range of complexity, and search through the available indexes. The histogram component is used to group the results in date buckets, from where the user can filter the documents even more. The result table can have its columns arranged based on the fields selected on the fields listed, placed to the left of the main result table.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



```
from sandbox_auth
| rename logsource as user
| where contains(message, 'Failed password')
| stats count() as attempts, min(timestamp) as timestamp by user, message
| where attempts >= 5
| regex message '(?<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})'
```

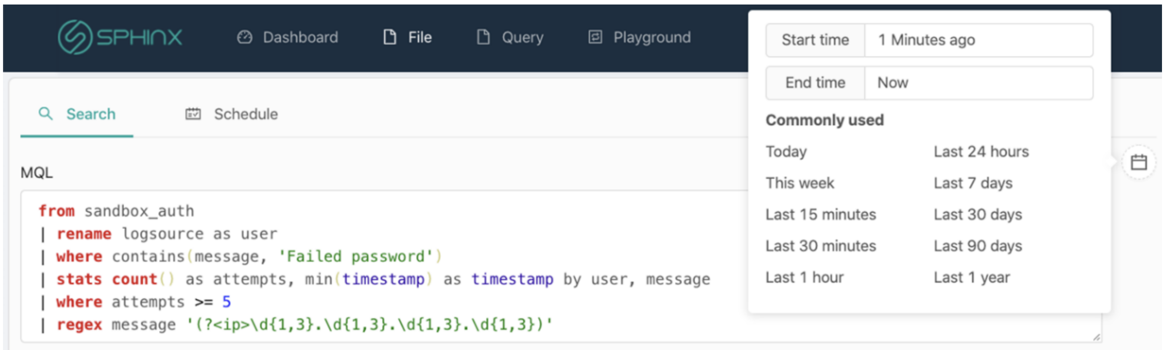
```
time                 _source
2020-07-22           user: mallory message: Failed password for invalid user mallory from 192.168.122.194 port 34322 ssh2 attempts: 11 ip: 192.168.122.194
11:04:00
```

Figure 29. MQL query example

Figure 29 demonstrates an example of a more elaborate query, as it also shows how the user can hide some components on the screen in order to gain more space, thus increasing the tool usability. This functionality is still evolving, and the development of several new features are currently in progress, such as creating charts from queries and listing the available indexes.

3.3 Scheduling

This functionality is the entry point for event monitoring and alerts generation. As discussed in the previous section, the user can search for specific events that they wish to monitor using the query tool. Once they find the exact query that matches their purpose, they are ready to move on to the schedule stage, that consists of configuring a time frame for filtering the query results, and also the periodicity for running that query. Last but not least, they define the actions they want to take whenever the query runs.



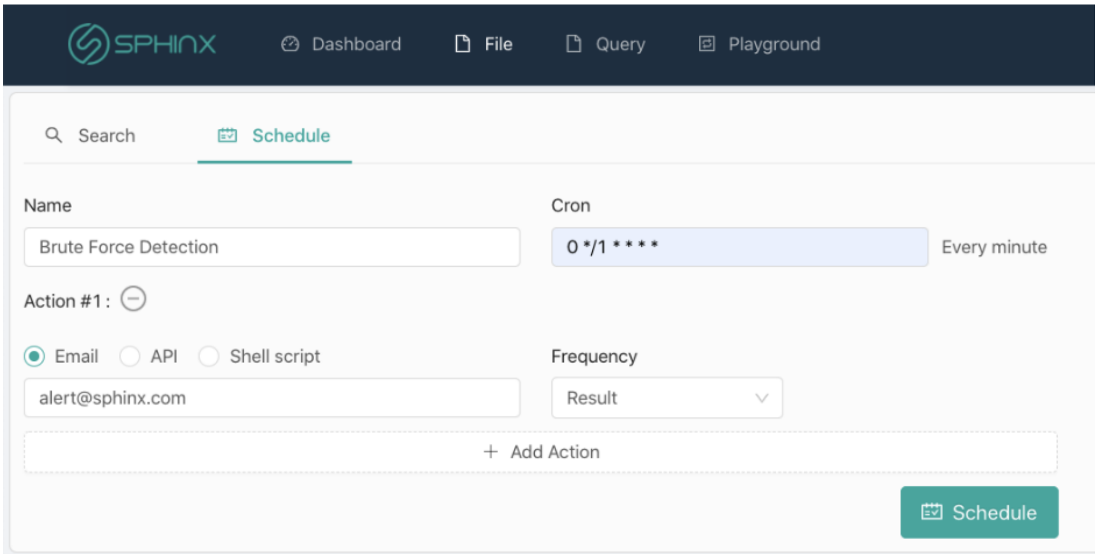
Start time: 1 Minutes ago

End time: Now

```
from sandbox_auth
| rename logsource as user
| where contains(message, 'Failed password')
| stats count() as attempts, min(timestamp) as timestamp by user, message
| where attempts >= 5
| regex message '(?<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})'
```

Figure 30. Query with time range definition

First, in Figure 30, we have an example where the user wishes to query for brute force attempts events. For doing so, he has defined a query that searches for messages that contains the words ‘Failed Password’, and those messages are going to be counted by user, and filtered where the number of attempts is equal to or greater than five in the last one minute till now.



Name: Brute Force Detection

Cron: 0 */1 * * * Every minute

Action #1 :

● Email ○ API ○ Shell script

alert@sphinx.com

Frequency: Result

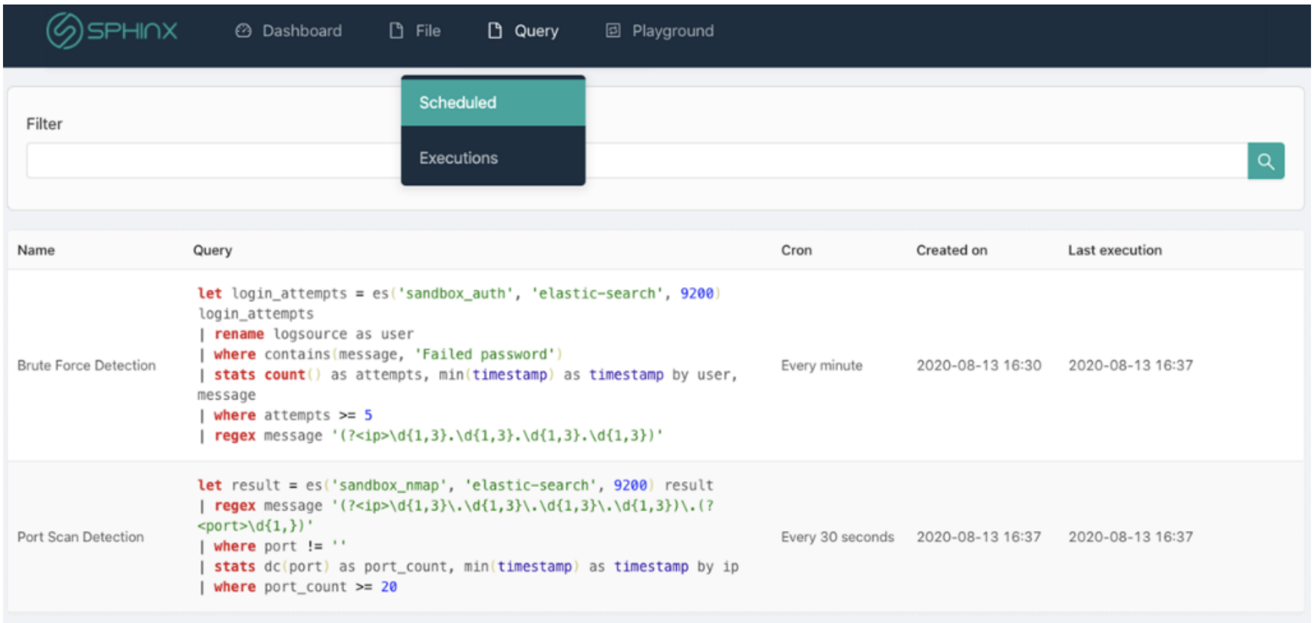
+ Add Action

Schedule

Figure 31. Query Schedule settings

Secondly, after defining the query criteria and the time frame they wish to filter the data from, the user can access the Schedule tab, as in Figure 31, and finish the schedule settings. At this stage, the user must provide a name for the schedule, also a CRON expression, which is a string composed of five or six fields separated by white space. This represents a set of times, normally as a schedule to execute some routine, thus indicating the periodicity with which the query is going to run.

Finally, the user must inform the system of what actions they wish to take whenever the defined query has returned results. For that matter, the system lists three (3) options, namely e-mail, API call and Shell script execution. Additionally, the user can define as many actions as they see fit.

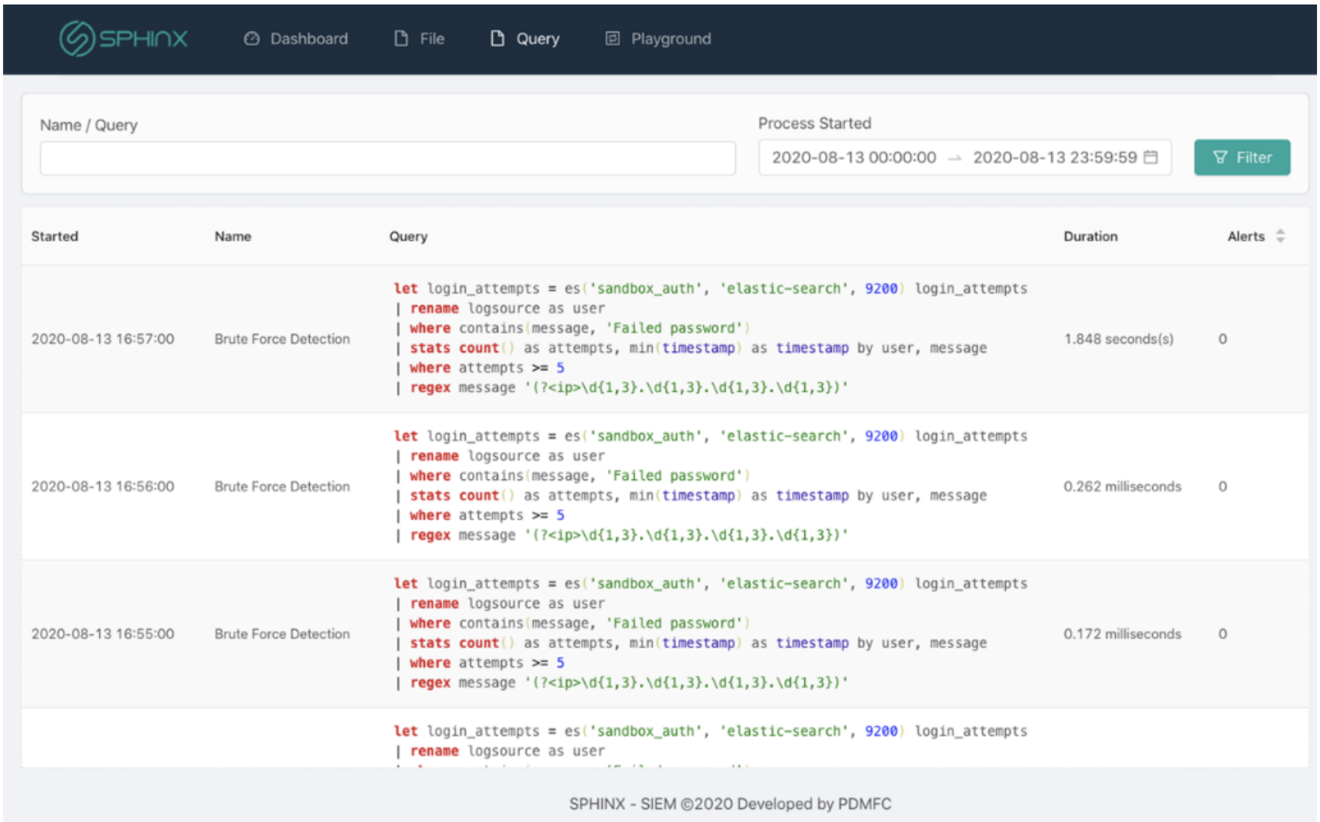


<table>
<tr><th>Name</th><th>Query</th><th>Cron</th><th>Created on</th><th>Last execution</th></tr>
<tr><td>Brute Force Detection</td><td>let login_attempts = es('sandbox_auth', 'elastic-search', 9200)
login_attempts
| rename logsource as user
| where contains(message, 'Failed password')
| stats count() as attempts, min(timestamp) as timestamp by user, message
| where attempts >= 5
| regex message '(?<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})'</td><td>Every minute</td><td>2020-08-13 16:30</td><td>2020-08-13 16:37</td></tr>
<tr><td>Port Scan Detection</td><td>let result = es('sandbox_nmap', 'elastic-search', 9200) result
| regex message '(?<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.(?<port>\d{1,})'
| where port != ''
| stats dc(port) as port_count, min(timestamp) as timestamp by ip
| where port_count >= 20</td><td>Every 30 seconds</td><td>2020-08-13 16:37</td><td>2020-08-13 16:37</td></tr>
</table>

Figure 32. All Scheduled Queries

All the queries being monitored can be listed under the menu "***Query > Scheduled"***, as shown in Figure 32. Moreover, from this page the user can see the last time the query ran, and if it presented any results. In the future they will also be able to edit the schedule settings.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



Name / Query

Process Started

2020-08-13 00:00:00 → 2020-08-13 23:59:59

<table>
<tr><th>Started</th><th>Name</th><th>Query</th><th>Duration</th><th>Alerts</th></tr>
<tr><td>2020-08-13 16:57:00</td><td>Brute Force Detection</td><td>let login_attempts = es('sandbox_auth', 'elastic-search', 9200) login_attempts
| rename logsource as user
| where contains(message, 'Failed password')
| stats count() as attempts, min(timestamp) as timestamp by user, message
| where attempts >= 5
| regex message '(?<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})'</td><td>1.848 seconds(s)</td><td>0</td></tr>
<tr><td>2020-08-13 16:56:00</td><td>Brute Force Detection</td><td>let login_attempts = es('sandbox_auth', 'elastic-search', 9200) login_attempts
| rename logsource as user
| where contains(message, 'Failed password')
| stats count() as attempts, min(timestamp) as timestamp by user, message
| where attempts >= 5
| regex message '(?<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})'</td><td>0.262 milliseconds</td><td>0</td></tr>
<tr><td>2020-08-13 16:55:00</td><td>Brute Force Detection</td><td>let login_attempts = es('sandbox_auth', 'elastic-search', 9200) login_attempts
| rename logsource as user
| where contains(message, 'Failed password')
| stats count() as attempts, min(timestamp) as timestamp by user, message
| where attempts >= 5
| regex message '(?<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})'</td><td>0.172 milliseconds</td><td>0</td></tr>
<tr><td></td><td></td><td>let login_attempts = es('sandbox_auth', 'elastic-search', 9200) login_attempts
| rename logsource as user</td><td></td><td></td></tr>
</table>

SPHINX - SIEM ©2020 Developed by PDMFC

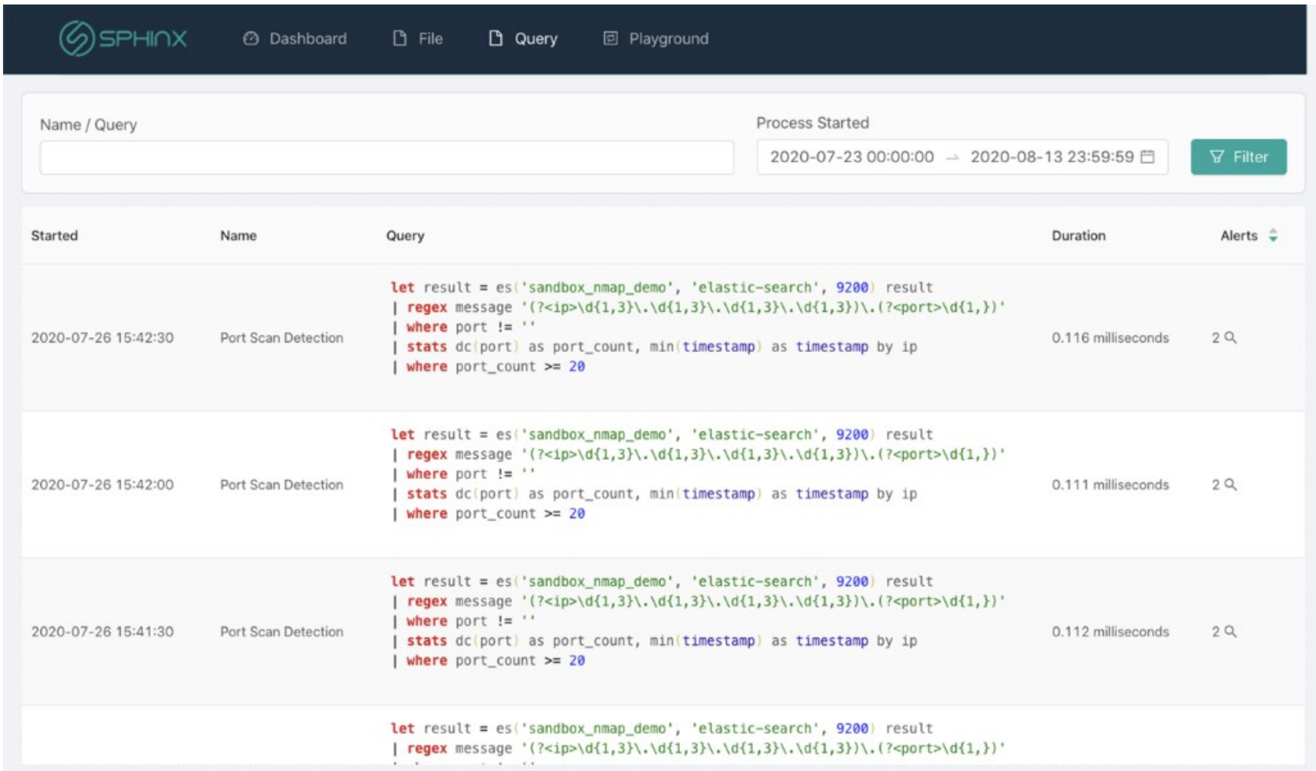
Figure 33. Query Executions Logs

Every scheduled query execution generates log information, and they can be found in the menu “***Query > Executions”***, as displayed in Figure 33, where the user can filter by name, query and/or datetime.

3.4 Alerts

Whenever a scheduled query runs and find results matching the criteria defined in the schedule settings, it is going to raise alerts and execute the configured actions. For instance, it could send an e-mail, or call an external API passing the results found as parameters. Additionally, the SIEM Web will be able to display all the logs referring to executions that raised alerts in order to centralize a point in the application where the user can go to list and filter all the alerts triggered in the system.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



Name / Query

Process Started

2020-07-23 00:00:00 → 2020-08-13 23:59:59

<table>
<tr><th>Started</th><th>Name</th><th>Query</th><th>Duration</th><th>Alerts</th></tr>
<tr><td>2020-07-26 15:42:30</td><td>Port Scan Detection</td><td>let result = es('sandbox_nmap_demo', 'elastic-search', 9200) result
| regex message '(?<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.(?<port>\d{1,})'
| where port != ''
| stats dc(port) as port_count, min(timestamp) as timestamp by ip
| where port_count >= 20</td><td>0.116 milliseconds</td><td>2</td></tr>
<tr><td>2020-07-26 15:42:00</td><td>Port Scan Detection</td><td>let result = es('sandbox_nmap_demo', 'elastic-search', 9200) result
| regex message '(?<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.(?<port>\d{1,})'
| where port != ''
| stats dc(port) as port_count, min(timestamp) as timestamp by ip
| where port_count >= 20</td><td>0.111 milliseconds</td><td>2</td></tr>
<tr><td>2020-07-26 15:41:30</td><td>Port Scan Detection</td><td>let result = es('sandbox_nmap_demo', 'elastic-search', 9200) result
| regex message '(?<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.(?<port>\d{1,})'
| where port != ''
| stats dc(port) as port_count, min(timestamp) as timestamp by ip
| where port_count >= 20</td><td>0.112 milliseconds</td><td>2</td></tr>
<tr><td></td><td></td><td>let result = es('sandbox_nmap_demo', 'elastic-search', 9200) result
| regex message '(?<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.(?<port>\d{1,})'</td><td></td><td></td></tr>
</table>

Figure 34. Scheduled query executions that raised Alerts

Figure 34 shows a list with executions that raised alerts. Upon clicking on the magnifying glass icon beside the alerts count, the user will be redirected to the search page, where they will see the alerts information, based on the query definition.

3.5 Playground

The Playground section has the purpose to enrich the platform, providing ways for the user to test some of the systems functionalities in an isolated environment. For instance, the first candidate to be presented in the playground area is the parsing tool, where it will be possible to use a file or a piece of text, and to apply Grok rules or regular expressions in an experimental way, without having to use any of the workflows for data collection. However, this feature is still under discussion, and all extra details on its development will be provided in the next iteration.

4 Use Cases

During the analysis stage, all partners working on the SPHINX project listed several use cases with real case scenarios to be demonstrated by integrating the platform components. However, this section is going to present two extra use cases that, despite not being in the official list, have the same potential to show how the SPHINX SIEM system can integrate with other components to solve problems and detect real world security threats.

4.1 Brute Force Attack

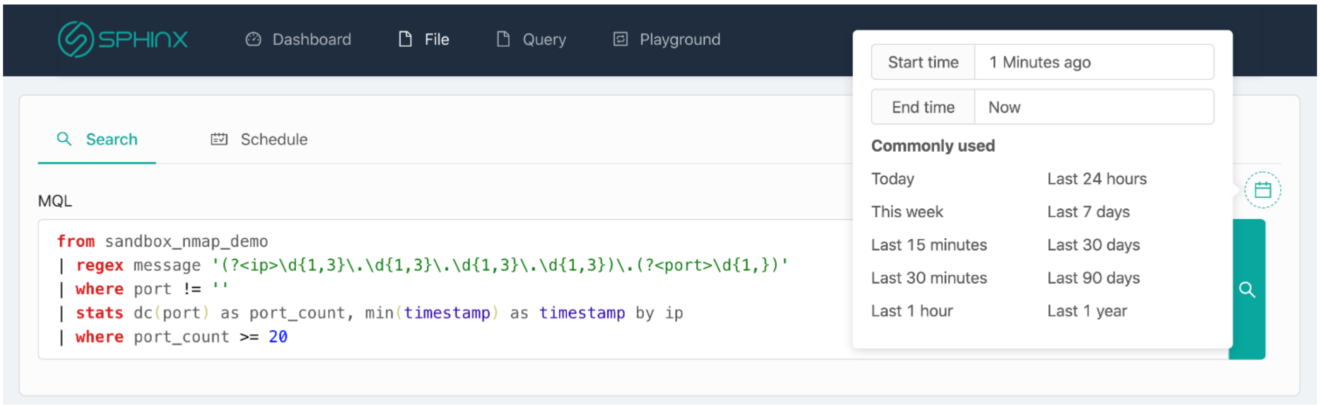
The main idea behind this use case is to monitor events where there are a certain number of failed login attempts. When this specific event occurs, it should trigger an alert and send an email to the security team. In this scenario we worked together with the SPHINX Sandbox component, which is responsible for providing all logs gathered from the user authentication system.

The first step in the process is configuring the rules to normalize and save the log files to the SIEM common database. Secondly, as shown in Figure 29??, it is necessary to set up the query for searching for Brute Force attempts. The next step is to set up the timeframe we want to use to monitor new events. In this case, we want to monitor log events from one minute ago till now, as in Figure 30. Finally, as shown in Figure 31, we need to provide information for scheduling the new query and save it. As a result, whenever a new event matching the configured criteria is found, an email is sent to the predefined address and the user can also list all the executions that raised alerts from the SIEM Web application (see Figure 34).

4.2 Port Scan Detection

In this scenario, the Event Manager is going to be watching for access attempts on different ports from the same source IP in a very short timeframe. This case also used logs provided by the SPHINX Sandbox component, that is in charge of gathering all network monitoring logs and sending it to the SIEM system.

After defining the rules for collecting the log files, normalizing, and saving the data to the SIEM database, it is necessary to setup the query find the events we want to monitor using one of the methods described in section 3.1.



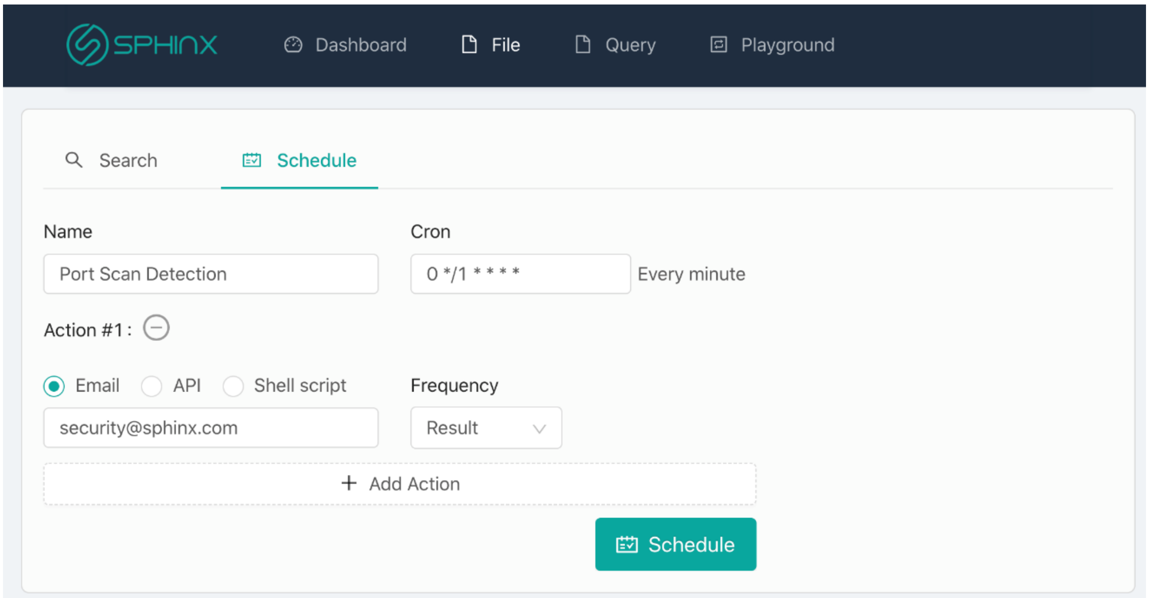
Start time: 1 Minutes ago
End time: Now

MQL

```
from sandbox_nmap_demo
| regex message '(?<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.(?<port>\d{1,})'
| where port != ''
| stats dc(port) as port_count, min(timestamp) as timestamp by ip
| where port_count >= 20
```

Figure 35. MQL query for detecting Port Scans attempts in the last minute

In Figure 35, we define the query that searches for situations where twenty or more access attempts on different ports from the same IP address were made in the past minute.



Name: Port Scan Detection

Cron: 0 */1 * * * Every minute

Action #1 :

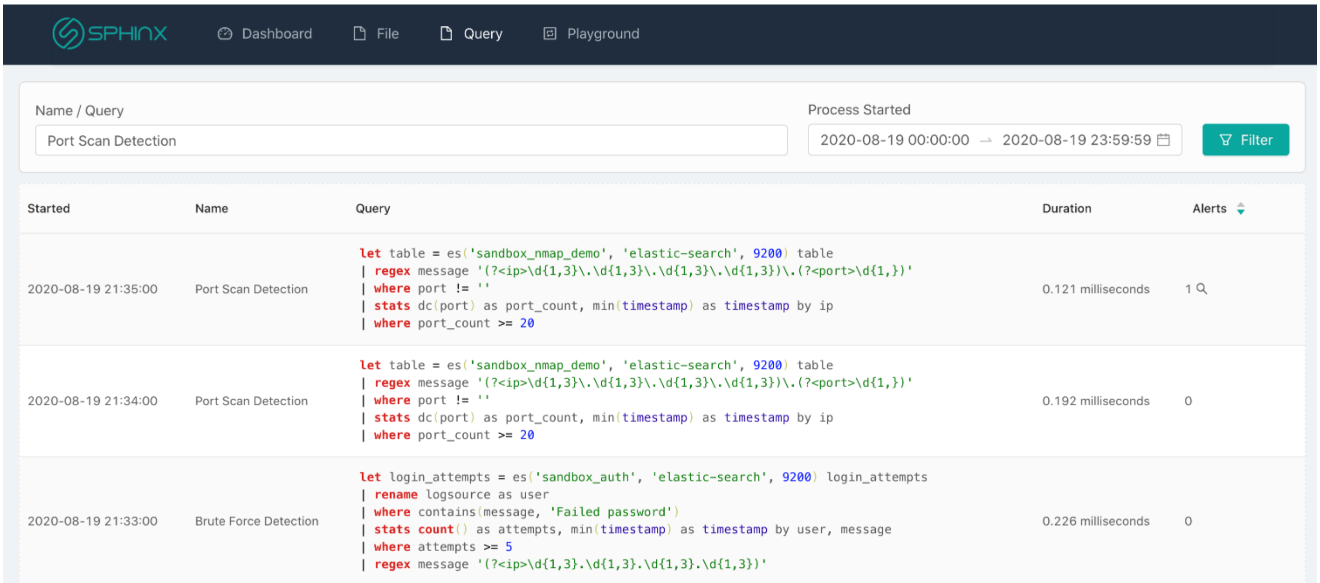
● Email ○ API ○ Shell script

security@sphinx.com

Frequency: Result

Figure 36. Schedule Query configuration

Figure 36 shows the Schedule settings, meaning that this query will be executed every minute, and whenever an alert is raised, the configured email box is going to receive a message with the alert's details.



Name / Query: Port Scan Detection

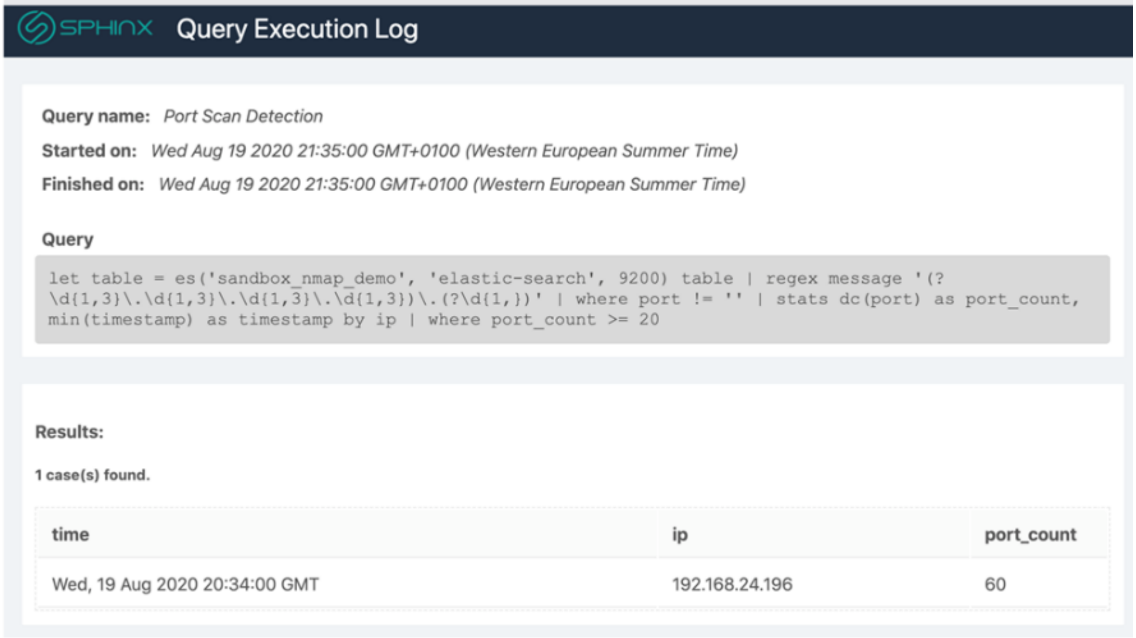
Process Started: 2020-08-19 00:00:00 → 2020-08-19 23:59:59

<table>
<tr><th>Started</th><th>Name</th><th>Query</th><th>Duration</th><th>Alerts</th></tr>
<tr><td>2020-08-19 21:35:00</td><td>Port Scan Detection</td><td>let table = es('sandbox_nmap_demo', 'elastic-search', 9200) table
| regex message '(?<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.(?<port>\d{1,})'
| where port != ''
| stats dc(port) as port_count, min(timestamp) as timestamp by ip
| where port_count >= 20</td><td>0.121 milliseconds</td><td>1</td></tr>
<tr><td>2020-08-19 21:34:00</td><td>Port Scan Detection</td><td>let table = es('sandbox_nmap_demo', 'elastic-search', 9200) table
| regex message '(?<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.(?<port>\d{1,})'
| where port != ''
| stats dc(port) as port_count, min(timestamp) as timestamp by ip
| where port_count >= 20</td><td>0.192 milliseconds</td><td>0</td></tr>
<tr><td>2020-08-19 21:33:00</td><td>Brute Force Detection</td><td>let login_attempts = es('sandbox_auth', 'elastic-search', 9200) login_attempts
| rename logsource as user
| where contains(message, 'Failed password')
| stats count() as attempts, min(timestamp) as timestamp by user, message
| where attempts >= 5
| regex message '(?<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})'</td><td>0.226 milliseconds</td><td>0</td></tr>
</table>

Figure 37. Alerts raised for Port Scan Detection

After configuring all the settings for the scheduled query, a port scan was simulated in the Sandbox in order to test the event monitoring process. As can be seen in Figure 37, a new alert was raised after the port scan was executed.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



Query Execution Log

Query name: *Port Scan Detection*

Started on: *Wed Aug 19 2020 21:35:00 GMT+0100 (Western European Summer Time)*

Finished on: *Wed Aug 19 2020 21:35:00 GMT+0100 (Western European Summer Time)*

Query

```
let table = es('sandbox_nmap_demo', 'elastic-search', 9200) table | regex message '(?
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.(?\d{1,})' | where port != '' | stats dc(port) as port_count,
min(timestamp) as timestamp by ip | where port_count >= 20
```

Results:

1 case(s) found.

time	ip	port_count
Wed, 19 Aug 2020 20:34:00 GMT	192.168.24.196	60

Figure 38. E-mail received from the SIEM platform, alerting about a new threat

Figure 38 displays the message received from the SIEM platform via e-mail, alerting about a possible port scan threat. Additionally, more actions could be taken from this point, such as executing a pre-defined shell script, or calling an external API method. And with this last picture we conclude the use case for monitoring Port Scan Threats.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

5 Summary and Conclusions

This deliverable presented the overall development status of the SIEM component. Considering this as a versioned document, in its first version it covers all the progress made so far, explaining in depth the SIEM system architecture and lifecycle, diving into its subcomponents and their interdependencies, and also discussing potential future functionalities. In its next version, this document will aim to conclude and exemplify all the predicted features that are still under development, thus providing a complete snapshot of the SPHINX SIEM system as whole.