Development of a GraphQL-based API for querying security advisories for Common Security Advisory Framework (CSAF)

Waldemar Wiegel

Supervisors

Prof. Dr. Jörg Keller
FernUniversität in Hagen

Dr. Klaus Biß
Thomas Schmidt
Federal Office for Information Security (BSI) Germany

September 30, 2023

Master thesis in the master program practical computer science
Matriculation number: 3279308
FernUniversität in Hagen

Abstract

Security advisories can help to close vulnerabilities more quickly. However, relevant security advisories have to be sifted out from the mass of security advisory documents. Both Common Security Advisory Framework (CSAF) distribution methods Resource-Oriented Lightweight Information Exchange (ROLIE) and directory-based do not provide opportunities to increase the document relevance. The creation date is practically the only filter criterion. The new Application Programming Interface (API) makes the content of security advisory documents searchable. Thus, it is possible to include these document contents in the API query as a condition. With the help of the API, expensive IT security personnel can be noticeably relieved by only having to process relevant security advisories.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

API	Application Programming Interface
API1:2019	Broken Object Level Authorization
API2:2019	Broken User Authentication
API3:2019	Excessive Data Exposure
API4:2019	Lack of Resources & Rate Limiting
API5:2019	Broken Function Level Authorization
API6:2019	Mass Assignment
API7:2019	Security Misconfiguration
API8:2019	Injection
API9:2019	Improper Assets Management
API10:2019	Insufficient Logging & Monitoring
API3:2023	Broken Object Property Level Authorization
API4:2023	Unrestricted Resource Consumption
API6:2023	Unrestricted Access to Sensitive Business Flows
API7:2023	Server Side Request Forgery
API10:2023	Unsafe Consumption of APIs
BSI	Federal Office for Information Security Germany
CIO	Chief Information Officer
CISO	Chief Information Security Officer
CSAF	Common Security Advisory Framework
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DPO	Data Protection Officer
FIRST	Forum of Incident Response and Security Teams
GDPR	General Data Protection Regulation

JSON	JavaScript Object Notation
OWASP	Open Web Application Security Project
REST	Representational State Transfer
ROLIE	Resource-Oriented Lightweight Information Exchange
SQL	Structured Query Language
TLP	Traffic Light Protocol
URL	Uniform Resource Locator

1 Introduction

IT systems and software components may have one or more vulnerabilities. Knowing that an IT system or software component, is vulnerable and how to mitigate the problem is a central challenge for a person responsible for IT security. Those responsible can find out whether an IT system or software component is affected by a vulnerability from a security advisory, for example. A security advisory consists of vulnerabilities and the IT systems and software components affected by the vulnerability. Security advisories are issued by manufacturers, IT security researchers or coordinating authorities. They should help to better understand a vulnerability in order to in the best case close or at least mitigate the vulnerability.

Different manufacturers resulted in different security advisory formats, which makes working with security advisories more difficult for IT security responsibles than it would be the case with a standardized format. With CSAF, such a standard for security advisories exists, so that security advisories are partially additionally issued in the CSAF format.

Over time, the number of security advisories issued in the CSAF format increased, so that IT security responsibles now have to check significantly more security advisories for relevance. Relevant security advisories are those relating to IT systems or software components used in the own area of responsibility.

To determine relevance, each security advisory must be queried and reviewed. Queries would become more efficient if only documents with specific content, rather than all documents, could be queried. With *GraphQL*, an open source data query language exists that may be able to query CSAF documents more efficiently. Therefore, the following research question arise:

> *How could a GraphQL-based API make querying CSAF documents more efficient?*
>
> Research question

1.1 Motivation

With security advisories, the reader gets an overview of the vulnerabilities and the products affected by them. Security Advisories are published by various manufacturers in relation to their products or by IT security researchers. The CSAF serves as a framework for security advisories, with defined roles, document structures, and distribution methods.

The CSAF standard supports two types of distribution: directory-based and ROLIE-based. In both cases, searching for CSAF documents with relevant content (e.g. products) is not possible. All documents must be downloaded to make the decision whether one's product is part of the document and is it affected by a vulnerability with a relevant rating. This has to be done continuously for all new and updated CSAF documents and is probably not the most exciting part of the job. Especially because, as of today, only parts of this process can be automated.

This is a waste of valuable time and human resources. Therefore, this master thesis follows the leitmotif of keeping the time between the release of the security advisory, knowledge of being affected, and getting it into the right hands short. This can shift valuable time and human resources from searching to acting.

To achieve this goal, an API is needed that provides the ability to query for relevant security advisories. And, if possible, all relevant security advisories for all products used should be downloaded at once. There is also prior work on the CSAF API.

1.2 Related work

Schmidt describes in *Development of an API to request security advisories for CSAF 2.0, Bachelor's thesis, University of Applied Sciences, Offenburg* a Representational State Transfer (REST) API based on routes [32]. This conceptual API defines routes for predefined use cases. All of these routes provide clear added value in the use cases described. Therefore, the new CSAF API must also provide these values. Schmidt has encountered three main problems "REST API Limitation", "CSAF Versioning" and "missing unique product identifier" [33].

1.3 Value of the new solution and opportunities

The powerful CSAF API query will be used to search not only for any CSAF document parts and contents, but also for properties with dependent content. For example, it will be possible to query by product name for products whose product ID occurs in the product status of a vulnerability. This eliminates one step and speeds up the CSAF consumer process. To reduce the abundance of information, the filter parameters will be used to further limit the result set. With the help of logical operators `AND`, `OR`, `XOR` or `NOT` it will be possible to formulate queries even more precisely to meet the requirements even more accurately (e.g. combining or excluding queries). With the additional functionality (exist) it will be possible to query for the existence of CSAF document components and their absence. This enable further use cases that can be used in quality management (e.g. query missing `/product_ull_product_names[]/product_identification_helper`). To achieve all this, a structured approach is needed.

1.4 Structure of the thesis

Chapter 2 provides background information by describing the different terms, going into more detail about CSAF and the related work on a CSAF API. The chapter concludes with an overview of *OWASP API Security Project* and CSAF document migration to a database.

Chapter 3 focuses on the design and the individual components of the CSAF API. *OWASP API Security Project* plays a crucial role in some design decisions. This chapter concludes with concrete design decisions.

In the most comprehensive chapter, the specific implementations are explained by using many examples. It is shown how to convert the CSAF schema into a *GraphQL* type definition, how the *Elasticsearch* connector was implemented, and concludes with the migration of the existing CSAF documents into the API database.

The fifth chapter contains a brief summary of all the goals achieved. Here, it becomes evident that the workload of those responsible for IT security is significantly reduced. The chapter concludes with possible further areas of application for the CSAF API.

The Discussion chapter addresses problems that could not be resolved by the CSAF API because CSAF schema must be modified first. For the implementation of the CSAF API a principle of *GraphQL* had to be violated, especially this point and the associated risks are discussed in detail. The chapter concludes with the greatest achievement and an unachieved goal or mitigated problem.

The last chapter concludes this master thesis with the central and most valuable achievement.

2 Background

To understand why CSAF and in particular a CSAF API is necessary, the background must be explained. The following subsections therefore explain what a security advisory is, what it consists of, and how it becomes a CSAF document. The different roles that work with CSAF documents must be discussed to understand that the work begins with the publication of the security advisory and ends with the remediation of the vulnerability or its mitigation. Because an earlier concept for the CSAF API exists, it must also be considered in order to incorporate its experiences. To ensure that the API can be operated relatively securely in the end, at least the common faults must be avoided. Therefore, the Open Web Application Security Project (OWASP) must be presented [25]. Vulnerabilities are one of the three pillars of the CSAF document, so it should be started with.

2.1 Vulnerabilities

A weakness is not automatically a vulnerability. It becomes a vulnerability when it can be demonstrably exploited. When a vulnerability becomes known and listed in the *Common Enumeration of Vulnerabilities Program* database, it receives a Common Vulnerabilities and Exposures (CVE) identifier [17]. CVE identifiers can be grouped under a Common Weakness Enumeration (CWE) category. Vulnerabilities become relevant if they affect a product that is in the organization's own area of responsibility. The vulnerabilities are also relevant if they are components of one's own product.

2.2 Affected products

If a vulnerability of a third-party software component becomes known, then manufacturers who use this third-party software components in their product check whether their product is affected by the vulnerability in the same or similar way. If this is the case and a solution or mitigating measures are available, these vendors will issue a security advisory and list their products that are somehow affected by the vulnerability. The product tree is the second pillar of the CSAF document.

Usually, not all of a manufacturer's products are equally affected by a vulnerability. Therefore, CSAF offers the possibility to specify the product status per vulnerability in the CSAF document. There are different vulnerability product statuses in the CSAF standard (see table 1). With the product status `not_affacted` the manufacturer can make clear which products are not affected by the vulnerability in order to avoid unnecessary calls from customers. This status `not_affacted` is not very spectacular, and yet it is soothing for those responsible for IT security. Vulnerabilities and products affected by them are important components of a security advisory.

CSAF product status	Description
first_affected	These are the first versions of the releases known to be affected by the vulnerability.
first_fixed	These versions contain the first fix for the vulnerability but may not be the recommended fixed versions.
fixed	These versions contain a fix for the vulnerability but may not be the recommended fixed versions.
known_affected	These versions are known to be affected by the vulnerability.
known_not_affected	These versions are known not to be affected by the vulnerability.
last_affected	These are the last versions in a release train known to be affected by the vulnerability. Subsequently released versions would contain a fix for the vulnerability.
recommended	These versions have a fix for the vulnerability and are the vendor-recommended versions for fixing the vulnerability.
under_investigation	It is not known yet whether these versions are or are not affected by the vulnerability. However, it is still under investigation - the result will be provided in a later release of the document.

Table 1: CSAF product statuses and descriptions by OASIS Open

2.3 Security advisory

Security advisories are issued not only by manufacturers, but also by IT security researchers and coordinating authorities (e.g. Federal Office for Information Security (BSI)). They contain the vulnerability or a collection of vulnerabilities, a list of affected products, and information about the security advisory document itself and the publisher. This is the third pillar of the CSAF document. In the best case, the security advisories contain a final solution or at least mitigation measures. This makes it easier for IT security responsibles to understand the problem and derive recommendations for action or to argue the need for action.

The CSAF security advisory pulls all these information together into a CSAF document. If new findings come from research, then the security advisory is expanded. If the criticality of a vulnerability rises or falls, the security advisory is adjusted.

In addition to the CVE identifier, the security advisory can also contain other information about the vulnerability, such as a rating and a brief description. The brief description is addressed to the consumer, the consumer interprets the relevance. If a rating is present, then the relevance has been established in an external context. Whether this context applies to the consumer and thus also the rating, the consumer still has to find out for himself. Nevertheless, this external evaluation gives the consumer the opportunity to pre-filter for supposedly more critical vulnerabilities.

2.4 Common Security Advisory Framework

A CSAF document contains a product tree, vulnerabilities, information about the publisher and the document, and in the best case a vendor fix. More and more vendors (see table 2) and other security advisory providers are additionally providing security advisories as CSAF documents.

Vendors and security advisory providers	**Providing CSAF documents**
Cisco Systems, Inc. [4]	at least since January 2018
TIBCO Software Inc. [38]	at least since November 2018
Nozomi Networks Inc. [18]	at least since November 2019
Siemens Aktiengesellschaft [36]	at least since April 2021
SICK AG [35]	at least since June 2021
Arista Networks, Inc. [3]	at least since July 2021
Festo SE & Co. KG [10]	at least since September 2021
Schneider Electric SE [34]	at least since December 2021
Oracle Corporation [23]	at least since April 2022
Red Hat, Inc. [30]	at least since May 2022
Hitachi Energy Ltd. [15]	at least since December 2022

Table 2: Providers with additional CSAF documents

CSAF extends the security advisory with a framework. It defines roles (*CSAF publisher*, *CSAF provider*, *CSAF trusted provider*, *CSAF lister* and *CSAF aggregator*) [21], distribution methods (ROLIE and directory based) and standardizes the security advisory JavaScript Object Notation (JSON) document (`/document`, `/product_tree` and `/vulnerabilities`).

2.4.1 CSAF roles

The CSAF consumer is the client, the user or in simple terms the consumer of the CSAF documents. The CSAF documents can be provided, for example, by IT security researchers as *CSAF publisher* or by product manufacturers as *CSAF provider*. A *CSAF lister* is similar to a phone book, it lists *CSAF providers* and *CSAF publishers* but does not provide CSAF documents itself.

The *CSAF aggregator* is different, it collects documents from different *CSAF publishers* and *CSAF providers* and makes them available for download, if the *CSAF publishers* and *CSAF providers* agree. A *CSAF publisher* or *CSAF provider* agrees if the `/mirror_on_CSAF_aggregators` attribute in the `provider-metadata.json` explicitly allows aggregation of CSAF documents.

2.4.2 CSAF document

As mentioned above (see subsection 1.3), the CSAF document consists of three parts (document, product tree and vulnerabilities). The intended API is to search document contents when querying, so relevant contents of the CSAF document are explained in the coming subsections.

2.4.2.1 Document

The document part of a CSAF document contains information about the publisher, called metadata in the further course. This metadata is no new information as long as the document was retrieved directly from the *CSAF publisher* or *CSAF provider*. If the CSAF document is not obtained directly from the publisher but via a *CSAF aggregator*, this information becomes relevant in a completely new way, namely as a search criterion in the CSAF API.

In addition to the publisher, the canonical URL `/document/references[category: "self"]/url` is given. Only then is it possible for the *CSAF consumer* to go directly to the source of the information. First-hand information may be more comprehensive and available more quickly.

The title of the document provides a good search criterion because it should be chosen as unique as possible. The Traffic Light Protocol (TLP) label is also suitable as a search and sharing criterion. Various definitions can be used for the definition of the TLP. The CSAF documents from Federal Office for Information Security Germany (BSI) use the TLP definition from FIRST.ORG, Inc [11]. Whereby only `TLP:WHITE` labeled documents may be shared without restriction (see table 3 for short descriptions and table 13 for definitions).

TLP label	**Short description**
TLP:RED	Not for disclosure, restricted to participants only.
TLP:AMBER	Limited disclosure, restricted to participants' organizations.
TLP:GREEN	Limited disclosure, restricted to the community.
TLP:WHITE	Disclosure is not limited.

Table 3: TLPv1 label short descriptions by FIRST.ORG, Inc

A valuable attribute is the `/document/tracking/current_release_date`, which can be used especially for automation. This is always the case when it comes to latest documents. Interfaces that should fetch latest documents automatically will query for this attribute.

Each CSAF document has a unique ID assigned by the publisher and located in the attribute `/document/tracking/id`. Strictly speaking, the ID is unique in the context of the publisher. Together with the publisher namespace `/document/publisher/namespace`, this ID can be used to query the current state of a CSAF document. An impression of the entire CSAF document object `/document` is given in figure 1.

Not only the `/document/tracking/current_release_date` can increase the relevance of CSAF documents found through the API. The products contained in the CSAF documents can contribute decisively.

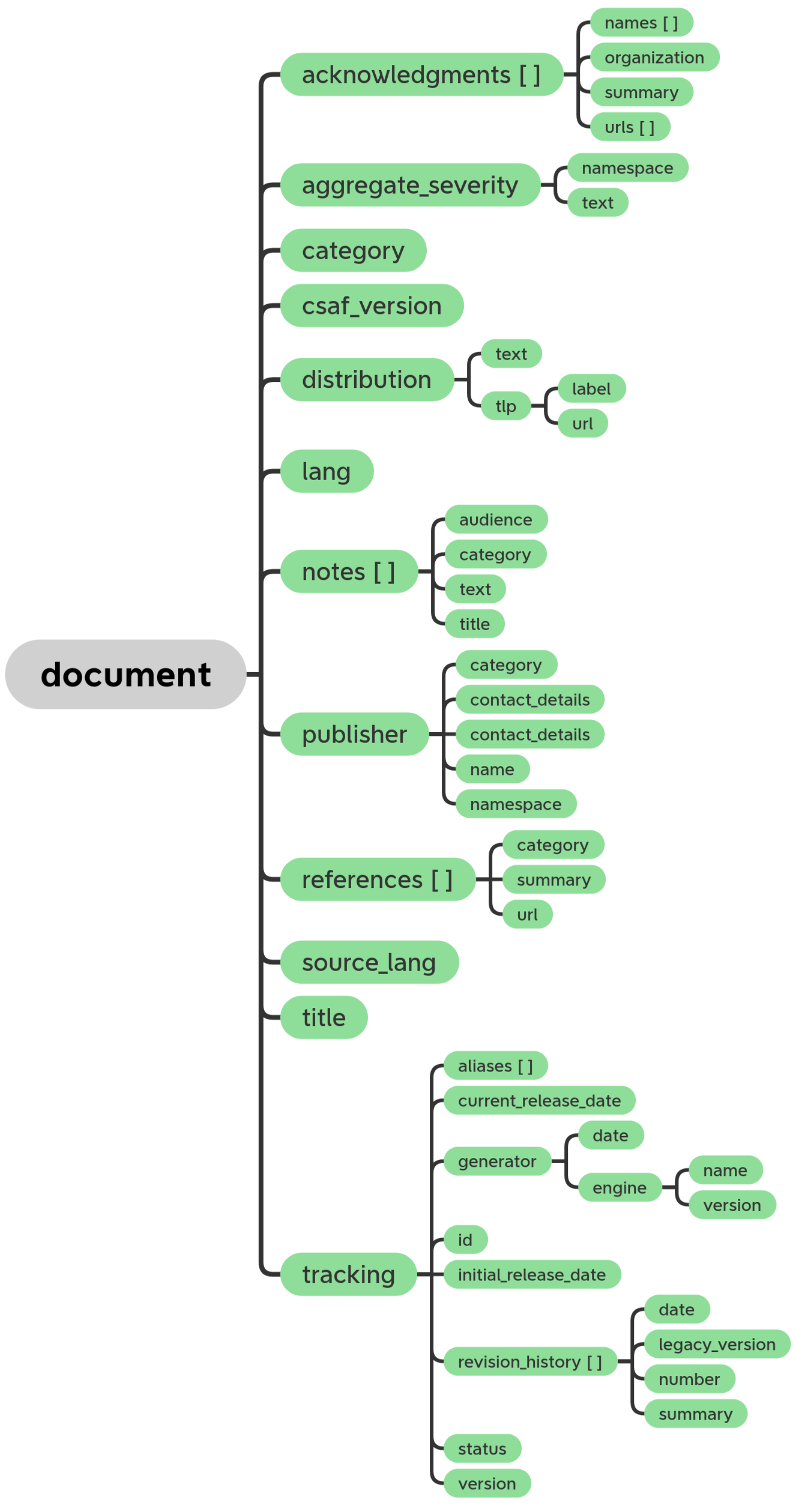


Figure 1: Document object of CSAF document

2.4.2.2 Product tree

Products can be defined in a CSAF document at different places (see table 4). In product tree branches `/product_tree/branches[]` whole product families can be defined. This is a special case in a CSAF document, since an infinite nesting of branches is possible. Branches are not suitable for querying because users cannot know in advance how many branch nestings actually exist and need to be queried.

Product ID definitions
/product_tree/branches[ ](/branches[ ])*/product/
/product_tree/full_product_names[ ]/
/product_tree/relationships[ ]/full_product_name/

Table 4: Possibilities for product definition in the CSAF document

If a product is defined in the CSAF document, it can be referenced in the CSAF document via the product ID (see references in table 5). For example, products can be grouped into product groups under the path `/product_tree/product_groups[]/product_ids[]`.

Product ID references
/product_tree/product_groups[ ]/product_ids[ ]
/product_tree/relationships[ ]/product_reference
/product_tree/relationships[ ]/relates_to_product_reference
/vulnerabilities[ ]/flags[ ]/product_ids[ ]
/vulnerabilities[ ]/product_status/first_affected[ ]
/vulnerabilities[ ]/product_status/first_fixed[ ]
/vulnerabilities[ ]/product_status/fixed[ ]
/vulnerabilities[ ]/product_status/known_affected[ ]
/vulnerabilities[ ]/product_status/known_not_affected[ ]
/vulnerabilities[ ]/product_status/last_affected[ ]
/vulnerabilities[ ]/product_status/recommended[ ]
/vulnerabilities[ ]/product_status/under_investigation[ ]
/vulnerabilities[ ]/remediations[ ]/product_ids[ ]
/vulnerabilities[ ]/scores[ ]/products[ ]
/vulnerabilities[ ]/threats[ ]/product_ids[ ]

Table 5: CSAF document elements referencing a product by product ID

Figure 2 shows the product tree with the infinite `../branches[]` nesting, highlighted in light blue.

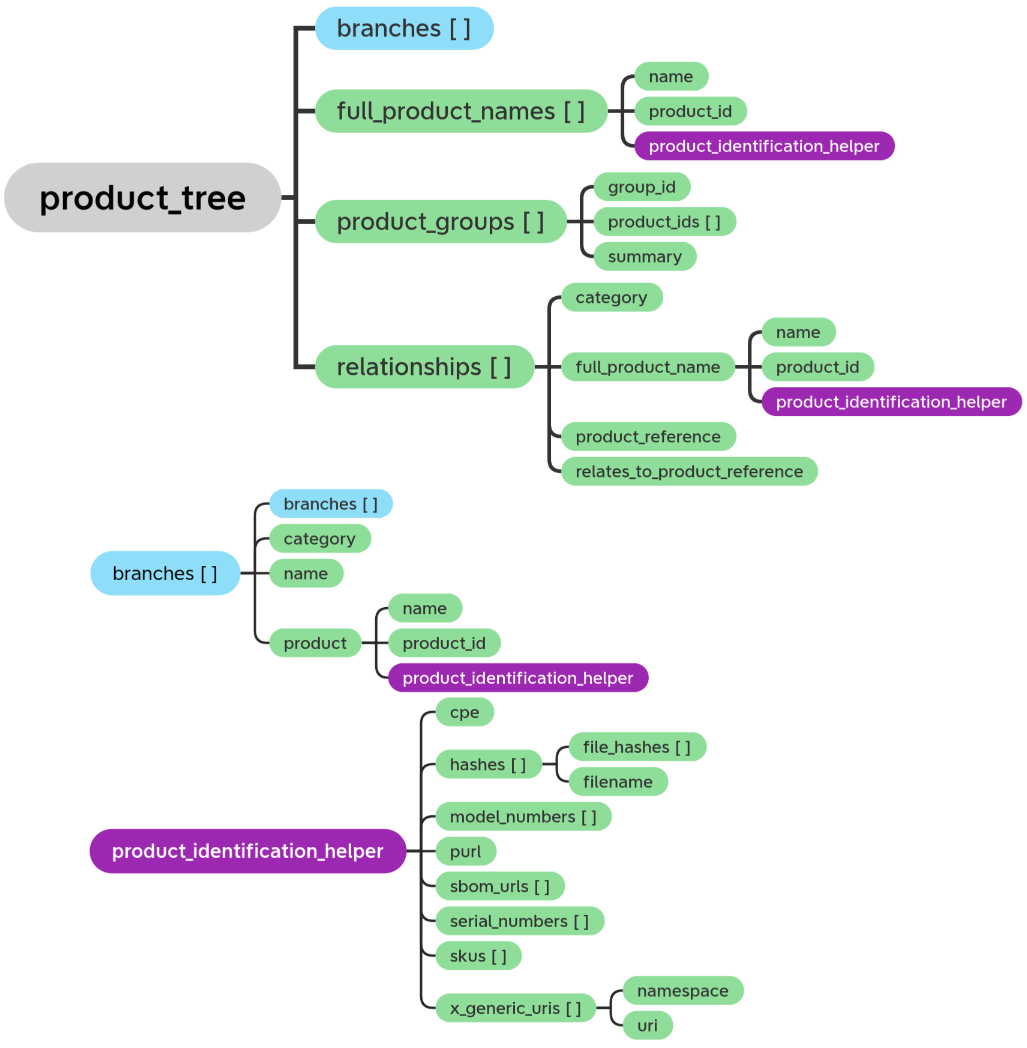


Figure 2: CSAF document - product_tree

Under the CSAF array `/product_tree/full_product_names[]` all products are listed that are not listed under `/product_tree/branches[]`. The product itself consists of a `name`, a `product_id` and a `product_identification_helper`. The product name is suitable for querying whether there are dependencies on the product that is used in one's own area of responsibility. Restricting to products increases the relevance of the results when products are affected by vulnerabilities.

2.4.2.3 Vulnerabilities

Whether a product is affected by a vulnerability is stated in the vulnerability product status `/vulnerabilities[]/product_status` (e.g. `fixed` or `known_not_affected`). Since not all versions of a product are equally affected by vulnerabilities, the various versions could be listed individually as branches in the CSAF document in order to assign them individually and thus clearly to a vulnerability.

The vulnerabilities object `/vulnerabilities[]` is an array of vulnerabilities. If a CVE has been assigned and this information is maintained at the vulnerability, then it can be found under the path `/vulnerabilities[]/cve`. If a CWE category has been determined and assigned for the vulnerability, then this information can be found under the path `/vulnerabilities[]/cwe/id`.

CVE and CWE are globally unique IDs and therefore suitable to search for CSAF documents containing them. Even without a product reference, these can be used to answer statistical questions about CVE and CWE (e.g. How many security advisories are there, with vulnerabilities assigned to this CVE or CWE?).

A significant role for the API query is played by the `/vulnerabilities[]/product_status` part. A vulnerability has a list of eight possible statuses (see table 1). The status contains product IDs for which the vulnerability status applies. Alternative lists and collections can be specified as sources under `/vulnerabilities[]/ids[]` (see example in listing 1), alternatively to CVE.

```
{
  "document": {...},
  "product_tree": {...},
  "vulnerabilities": [{
    ...
    "ids": [{
      "system_name": "Github Issue",
      "text":        "csaf-tools/CVRF-CSAF-Converter#78"
    }]
  }]
}
```

Listing 1: Vulnerabilities ids example

Probably the nicest, because most soothing, part of a CSAF document is the remediations part `/vulnerabilities[]/remediations[]`. A look at the details `/vulnerabilities[] /remediations[]/details` tells an IT security responsible what to do to fix or mitigate the vulnerability (see listing 2). The existence of this attribute is suitable as a criterion if a person responsible for IT security is only interested in CSAF documents that contain a vendor fix `/vulnerabilities[] /remediations["category":"vendor_fix"]`.

The document parts `/vulnerabilities[]/scores[]/cvss_v2` and `/vulnerabilities[] /scores[]/cvss_v3` have already been addressed in Motivation (see subsection 1.1). IT security responsibles can usefully restrict the search queries to supposedly more critical vulnerabilities. It must not be forgotten that these scores were created in a context that differs from their own. Despite this, the scores are excellent for queries, as they highlight supposedly critical vulnerabilities and thus increase the relevance of the result documents. For a complete overview of all vulnerability elements, see figure 3.

An API that also considers the content of CSAF documents during querying can increase the relevance of result documents and complement existing distribution methods well.

```
{
  "document": {...},
  "product_tree": {...},
  "vulnerabilities": [{
    ...
    "remediations": [{
      "category": "vendor_fix",
      "date": "2022-03-14T13:10:55.000+01:00",
      "details":  "Update to the latest version of the product.
        At least version 1.0.0-rc2",
      "product_ids": [...],
      "url": "https://github.com/csaf-tools/CVRF-CSAF-Converter/
        releases/tag/1.0.0-rc2"
    }]
  }]
}
```

Listing 2: Vulnerabilities remediation example

2.4.3 CSAF distribution methods

The consumer can access and download CSAF documents directly through a browser (see figure 4). There is a directory structure and the files are sorted by name. CSAF documents can change over time (tracked in `../changes.csv` in the main directory adjacent to the year folders), so the consumer must also check the timestamp of the files to ensure all updated CSAF documents are downloaded. Whether the found CSAF document is still interesting and relevant, the consumer sees only after he has downloaded the CSAF document and checking the content.

In the ROLIE based distribution the topicality is in the foreground, it is sorted by date. The first entry is therefore the most recent CSAF document. The most recent CSAF documents still need to be downloaded to determine relevance (see figure 5).

To enable a CSAF consumer to determine the integrity of CSAF documents, hash values are formed for each CSAF document, signed and additionally offered for download. They are relevant for the API in that these documents are to be queried as required.

2.4.4 CSAF document hashing

Each CSAF document is stored on the file system with the *OpenPGP Message Format* [16] signature and one or more hashed values (e.g. SHA256, SHA512 [37]). Using the *OpenPGP Message Format* key and hash values, a consumer or IT security responsible can check whether the CSAF document has been tampered with in transit from the publisher to the consumer. The self-generated hash value must match the generated value. IT security responsibles have an additional and crucial task that influences the speed of remediation implementation.

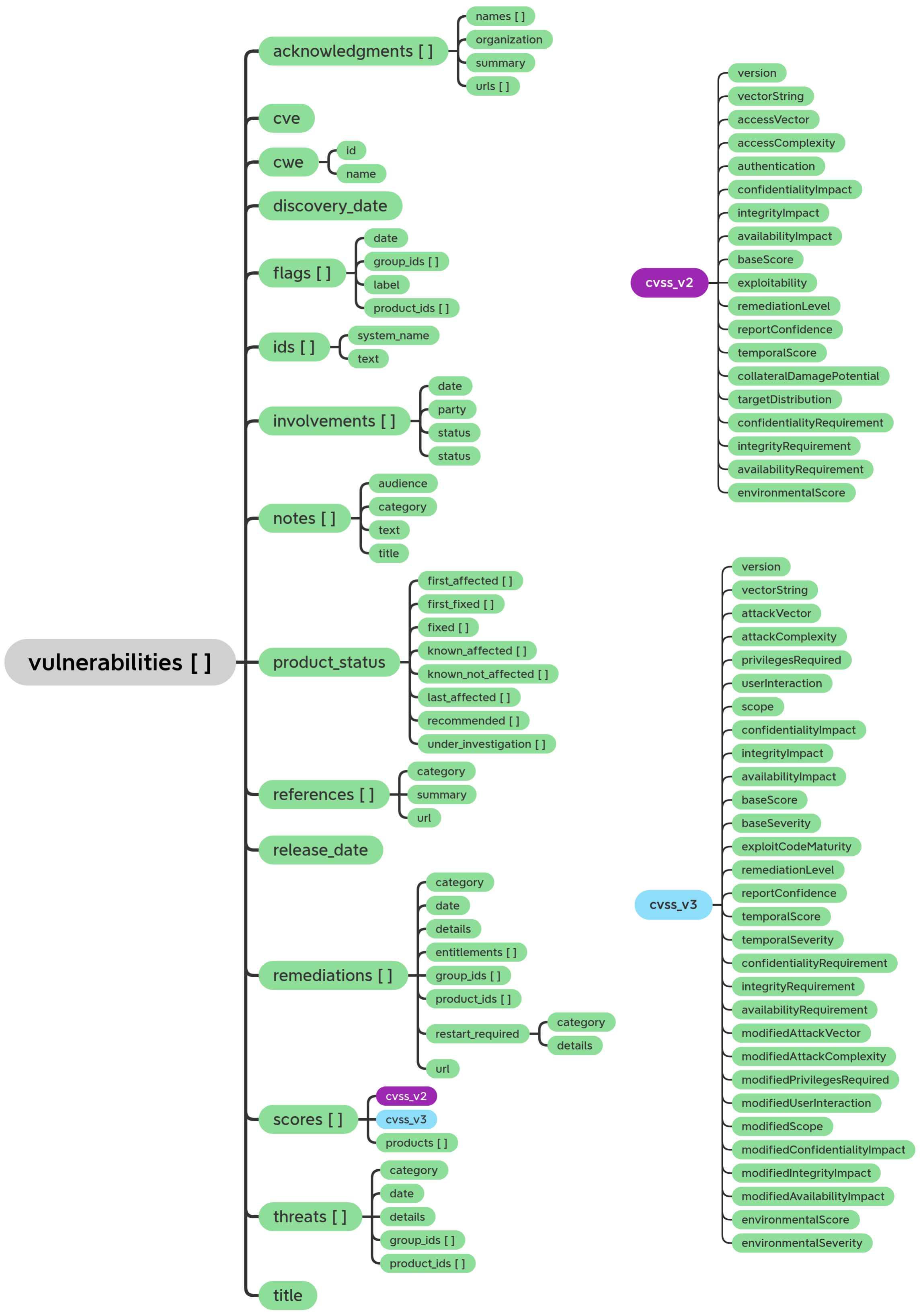
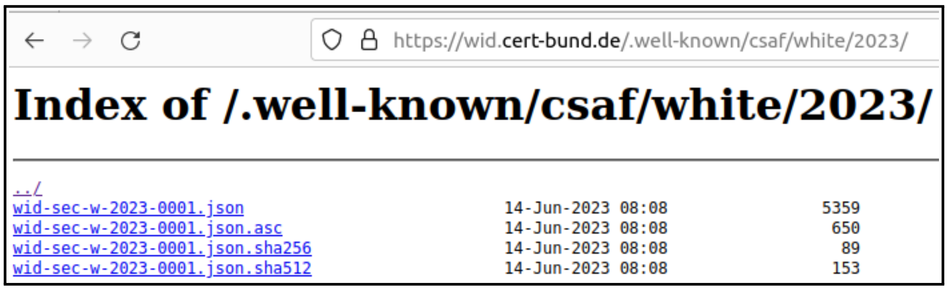


Figure 3: CSAF document - vulnerabilities

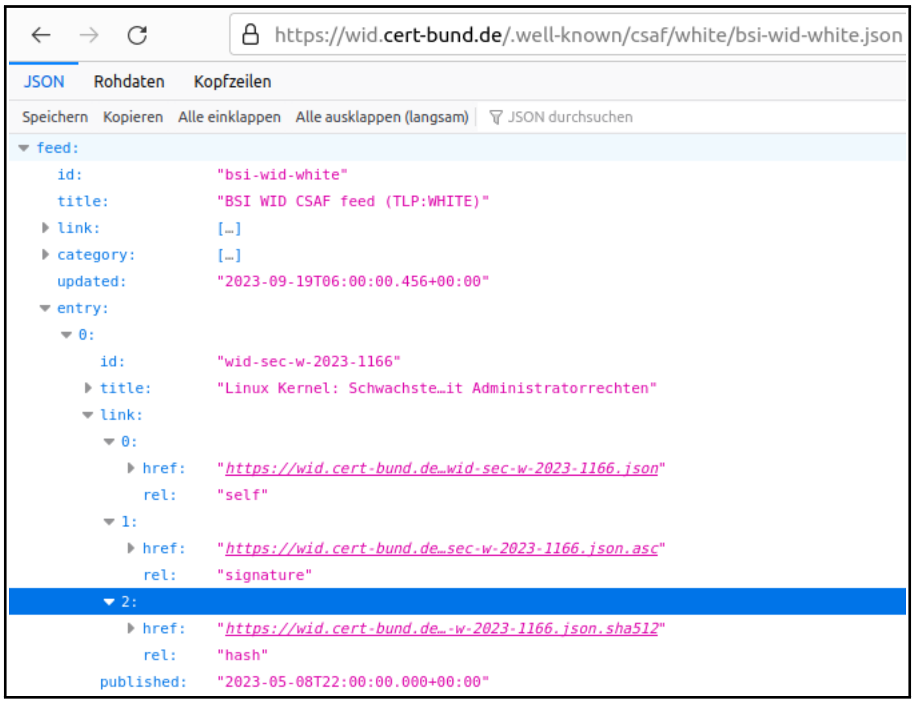


https://wid.cert-bund.de/.well-known/csaf/white/2023/

Index of /.well-known/csaf/white/2023/

```
../
wid-sec-w-2023-0001.json                         14-Jun-2023 08:08           5359
wid-sec-w-2023-0001.json.asc                     14-Jun-2023 08:08            650
wid-sec-w-2023-0001.json.sha256                  14-Jun-2023 08:08             89
wid-sec-w-2023-0001.json.sha512                  14-Jun-2023 08:08            153
```

Figure 4: CSAF directory based distribution by Federal Office for Information Security (BSI)



https://wid.cert-bund.de/.well-known/csaf/white/bsi-wid-white.json

JSON Rohdaten Kopfzeilen
Speichern Kopieren Alle einklappen Alle ausklappen (langsam) JSON durchsuchen

```
▼ feed:
    id:               "bsi-wid-white"
    title:            "BSI WID CSAF feed (TLP:WHITE)"
  ▶ link:             […]
  ▶ category:         […]
    updated:          "2023-09-19T06:00:00.456+00:00"
  ▼ entry:
    ▼ 0:
        id:           "wid-sec-w-2023-1166"
      ▶ title:        "Linux Kernel: Schwachste…it Administratorrechten"
      ▼ link:
        ▼ 0:
          ▶ href:     "https://wid.cert-bund.de…wid-sec-w-2023-1166.json"
            rel:      "self"
        ▼ 1:
          ▶ href:     "https://wid.cert-bund.de…sec-w-2023-1166.json.asc"
            rel:      "signature"
        ▼ 2:
          ▶ href:     "https://wid.cert-bund.de…-w-2023-1166.json.sha512"
            rel:      "hash"
        published:    "2023-05-08T22:00:00.000+00:00"
```

Figure 5: CSAF ROLIE based distribution by Federal Office for Information Security (BSI)

2.5 Persons responsible for IT security

The security of IT is spread over many shoulders. Above a certain size, this creates different roles in a company. The smaller the company, the more tasks are handled by a single person. No matter how many different people or roles are involved, it is important that the person acting also gets the opportunities to act. The following five roles have different tasks and yet at least one common goal: To make their IT more secure.

The IT administrator is the key person who keeps IT products up to date and closes vulnerabilities by installing patches and applying mitigations. If a vulnerability is known and one's own IT system or software component is affected, then an IT administrator responsible for this product is instructed to eliminate the vulnerability. IT administrators are part of the IT department in larger companies, with a department head who also does IT project work in addition to IT operations. IT projects compete for IT budget. Since it is difficult to calculate the return on investment for IT security projects, particularly good arguments must be made here.

This is where the Chief Information Security Officer (CISO) comes in. One of his most important skills is communication, so that necessary measures are planned and taken. In German, there is a kind of play on words. You can make a good living from fortune-telling (in German: Wahrsagen), but not from telling the truth (in German: Wahrheit sagen). Here is the original German version:

> *"Vom Wahrsagen läßt sich wohl leben in der Welt, aber nicht vom Wahrheit sagen."*
>
> Georg Christoph Lichtenberg

It is CISO's job to argue and vigorously pursue the need with a mixture of `fortune-telling` and `telling the truth`. The CISO is supported by the Data Protection Officer (DPO) in many respects, because many of the systems and software components involved also process personal data protected by the General Data Protection Regulation (GDPR) in the EU. With the clear goal in mind, armed with the arguments from the security advisory and flanked by the GDPR, the CISO can go to the Chief Information Officer (CIO) and get the necessary budget and resources to do his job now and do it well.

In order to cover the needs of all IT security roles, it must be possible to query the API for as many CSAF document parts as possible. Not all document parts are always required by all IT security roles and this is where the strength of *GraphQL* comes into play.

2.6 GraphQL

GraphQL is in a way a view of things, in parts comparable with a filter, only certain things are let through. If an object consists of several elements, then the user gets exactly those elements displayed that he explicitly asked for (see Description of GraphQL).

> *"GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools."*
>
> Description of GraphQL by GraphQL Foundation [13]

In the intended API context this is exactly where a problem arises. If the searched element is included, the CSAF consumer sometimes wants to get all elements of a CSAF document, without knowing which of them exist and without having to query all possible elements. Thus, the intended API would no longer provide a predictable result and would violate the following principle (see Principle of GraphQL).

> *"Send a GraphQL query to your API and get exactly what you need, nothing more and nothing less. GraphQL queries always return predictable results. Apps using GraphQL are fast and stable because they control the data they get, not the server."*

Principle of GraphQL by GraphQL Foundation [13]

Nevertheless, *GraphQL* is auspicious because individual elements can be targeted for querying, inputs and outputs are validated, and experience values with APIs based on *GraphQL* exist on the Internet. These experiences are especially valuable for the secure operation of the intended API.

Implementing the API based on *GraphQL* is not the first attempt. Therefore, experience from the conceptual API based on REST is available.

2.7 Related work

Schmidt describes in his Bachelor's thesis a conceptual REST API (see subsection 1.2) and defines routes for use cases (see table 6).

HTTP-Method	**Route**	**Use case**
GET	../by-cve/{cve}	Find the CSAF document(s) that contains the following CVE.
POST	../from-device-list	Find all documents containing any device in the device list.
GET	../match-property	Find all documents where the property has this value/type.
POST	../match-properties	Find all documents where all of the properties have this value/type.
GET	../by-id/ {publisher_ns}/ {tracking_id}	Find the CSAF document with the global ID supplied.
GET	../by-title/{title}	Find the CSAF document(s) having the title specified.
GET	../by-publisher/ {publisher_name}	Find all documents from publisher X.

Table 6: Routes and their use case, defined by Schmidt

The intended API should serve these use cases without running into the three problems ("REST API Limitation", "CSAF Versioning" and "missing unique product identifier") Schmidt mentioned.

2.7.1 REST API Limitations

Schmidt's conceptual REST API is limited by design to the defined routes and their intended purpose (e.g. find the CSAF document(s) that contains the following CVE). New use cases (such as find CSAF document(s) with a missing CVE) must be implemented additionally.

`OR`-concatenation of routes (`../by-cve/{cve} OR ../from-device-list`) is only possible if both routes are queried separately from each other. This creates the distinct union problem for REST API user, Schmidt mentions [33]. If two or more routes are retrieved, then duplicates cannot be avoided, since it is not possible to prevent documents from being retrieved twice. Complex queries (e.g.: XOR concatenated) are impossible. The terms (`device` and `device list`) are replaced by the terms (`product` and `product list`) in this master thesis to reflect the CSAF terminology (see `/product_tree` in subsection 2.4.2.2).

2.7.2 CSAF Versioning

Versioning of the CSAF document does not take place. Differences between once and now downloaded CSAF document must be found on the consumer side by comparing both CSAF document versions.

> *"However, the API lacks one important feature in its current state: tracking document updates."*
>
> Missing CSAF versioning by Schmidt

2.7.3 Missing unique product identifier

The third challenge identified by Schmidt is the lack of a unique product identifier. For example, the CSAF document attribute `/product_tree/full_product_names[]/product_id` is only a CSAF document-internally unique ID (see Product id description).

> *"Token required to identify a full_product_name so that it can be referred to from other parts in the document. There is no predefined or required format for the product_id as long as it uniquely identifies a product in the context of the current document."*
>
> Product id description by OASIS Open

Therefore, the attribute `/product_tree/full_product_names[]/product_id` cannot be used to search across publishers. Only the full product name seems to be suitable for searching across all publishers (see Product name description).

> *"The value should be the product's full canonical name, including version number and other attributes, as it would be used in a human-friendly document."*
>
> Product name description by OASIS Open

There is no reliable naming convention for full product names, so product names may vary depending on which CSAF publisher creates the document. The matter is complicated even more by the fact that the product ID can change during the life cycle of a CSAF document; after all, the product ID only needs to be unique in the current version of the document.

`branches` nesting can alleviate the problem to some extent. In the example of listing 3, vendor name, product name and the version number are included in the full product name (FastStone Image Viewer 7.6) in a human-readable way (see listing 3). These three informations can also be structured hierarchically in a CSAF document. OASIS Open recommended a hierarchical structure e.g. `vendor -> product_name -> product_version` (marked teal in listing 3) [20]. But other hierarchical structures are also conceivable (e.g.

vendor -> product_family -> product_name -> product_version). Therefore, the category product_name may vary in the branches nesting depth. This circumstance must be taken into account in the intended API.

```
{
  "document": {...},
  "product_tree": {
    "branches": [{
      "category": "vendor", "name": "FastStone",
      "branches": [{
        "category": "product_name", "name": "Image Viewer",
        "branches": [
          {
            "category": "product_version", "name": "7.6",
            "product": {
              "name": "FastStone Image Viewer 7.6",
              "product_id": "CSAFPID-0001"
            }
          },
          {
            "category": "product_version_range", "name": "<7.6",
            "product": {
              "name": "FastStone Image Viewer <7.6",
              "product_id": "CSAFPID-0002"
            }
          }
        ]
      }]
    }]
  },
  "vulnerabilities": [...]
}
```

Listing 3: Product_tree snippet from Federal Office for Information Security (BSI)

IT security responsibles can be responsible for one or more products. If they are responsible for multiple products, then there is an obvious interest in retrieving all security advisories that affect all those products.

2.8 Product list

A product list in the terms of this master thesis is a list of all systems and software components used in an organization. Any system or software component may have vulnerabilities. Therefore, the CSAF documents must be queried for each product (system and software component). Knowing the contents of the product list can open various attack vectors.

Classifying informations makes it easier to decide whether and to whom this informations may be passed on. There are no rigid specifications for the classification of information. Often four clusters are formed (see table 7).

Public	**Internal**	**Confidential**	**Strictly confidential**
already published information or free to share	information for internal use only	access is restricted to named groups	access is limited to named individuals

Table 7: Rough clustering of information classification

The classification of a product list is essential for the use of the intended API. The product list can disclose everything about the products used by the CSAF consumer. Therefore, the product list must be protected from attackers. When the product list is passed to the API, risks arise. If an attacker succeeds in intercepting this product list or hijacks it by manipulating the API, he gets an accurate picture of possible attack vectors related to the CSAF customer's vulnerable products in use. If the attacker additionally captures identity information (e.g. company names), the risk increases even further. Therefore, the CISO must be diligent in classifying the product list. Here it must be considered that a too restrictive classification of the product list prevents a rational use of the API.

For this reason, it makes sense not only to specify functional requirements, but also non-functional requirements for secure API operation.

2.9 OWASP API Security Top 10 - 2019

APIs are critical components of modern web applications and therefore targets for attackers. OWASP, through the OWASP API Security Project [26] , presents the top ten mistakes made by API developers. These top 10 (API1:2019 to API10:2019) need to be analyzed and applied to the intended CSAF API to eliminate potential flaws in the API. OWASP Foundation additionally assists in this process with its special cheat sheet for *GraphQL* [24].

Broken Object Level Authorization (API1:2019) is about access restrictions. These restrictions must not be circumvented by manipulating the query. Transferred to the intended CSAF API, it means that only CSAF documents that are approved for the specific user may be accessed. So it must be prevented that the API user gets access to protected CSAF documents by manipulating the queries.

Broken User Authentication (API2:2019) is about authentication. Authentication is an upstream process and not part of the intended CSAF API, so the consideration here is limited to recommendations. Unauthenticated users have no token or no valid token. If a user becomes an attacker and tries to send a token, there is a risk that a valid token will be hit. Therefore, token generation must generate sufficiently long tokens that are not continuous and thus difficult to guess. Following the OWASP Foundation recommendation for session IDs, the token should have a length of at least 16 bytes [28].

Excessive Data Exposure (API3:2019) is to remind that internal information should remain internal. So there is a risk that error messages pass internal information (e.g. the chosen database) to an API user. To prevent data disclosure, database-specific error messages should be replaced with API's own error messages before they are propagated. This can prevent internal structure information from leaking out.

With Lack of Resources & Rate Limiting (API4:2019), the focus is on availability. Limitations are intended to make the API resilient. To prevent resource consumption from increasing uncontrollably, limiting parameters must be foreseen. Transferred to the intended CSAF API, for example, the maximum number of CSAF documents that can be retrieved with one query should be specified. To further limit resource consumption, it is recommended to run an API in a virtualized environment, then consumptions (such as memory, CPU, . . .) can be further limited.

Broken Function Level Authorization (API5:2019) draws attention to the fact that there may be functions intended for an administrator that could be abused by an attacker. For the intended CSAF API no administrative API user is needed, it should be distinguished between authenticated and unauthenticated users. Since the function set remains the same whether authenticated or not, this point does not seem to have any relevance to the design of the intended API.

With Mass Assignment (API6:2019), careless source code is pointed out. It is reckless to assume that every value sent, whether expected or not, can also be taken over into one's own database. The CSAF API is not intended to persistently take over user data, so the relevance of this point is rather low. Although no data of the user is supposed to be taken over into the database when using the CSAF API, this carelessness can cause problems at one point or another in the source code. Therefore, this point should be considered during implementation.

Security Misconfiguration (API7:2019) includes, for example, sending an error traces to API user. The CSAF API must be configured to prevent these misconfigurations, which can be prevented by the API implementation.

Injection (API8:2019) draws attention to the fact that APIs are not safe from injections either. Whenever user input has to be interpreted, it can result in a malicious execution. Since *GraphQL* validates API queries against the API schema, the risk seems to be low with respect to API8:2019. If *GraphQL* validation is deviated from, this point must be considered again.

Improper Assets Management (API9:2019) deals with previous API versions and any remaining functionality that was not included everywhere. The intended CSAF API is a new API, with no previous versions at all, so this point should not matter. Should proof of concept or similar occur during the implementation phase, then these versions must be cleaned up so that the point mentioned here does not occur.

Insufficient Logging & Monitoring (API10:2019) deals with a topic that seems to be neglected. Since logging does not have to meet any functional requirements, it is difficult to define requirements. The CSAF API should consist of different modules, each module should have at least one error handling and should generate at least one log entry for monitoring. Whether additional logging and monitoring is required beyond that must be shown in practice. OWASP Foundation has released a new API Top 10 list while working on this thesis. The changes from version 2019 to 2023 must be duly considered [27].

2.10 OWASP API Security Top 10 - 2023

OWASP Foundation released a new version on July 3, 2023. In this brand new version, two categories API3:2019 and API6:2019 have been merged under a new category Broken Object Property Level Authorization (API3:2023) because they share a root cause. The category API4:2019 has been renamed to Unsafe Consumption of APIs (API10:2023) because more emphasis is placed on resource consumption than the pace at which they are exhausted. Three new categories Unrestricted Resource Consumption (API4:2023), Unrestricted Access to Sensitive Business Flows (API6:2023) and Server Side Request Forgery (API7:2023) have been added. Figure 6 draws the complete picture.

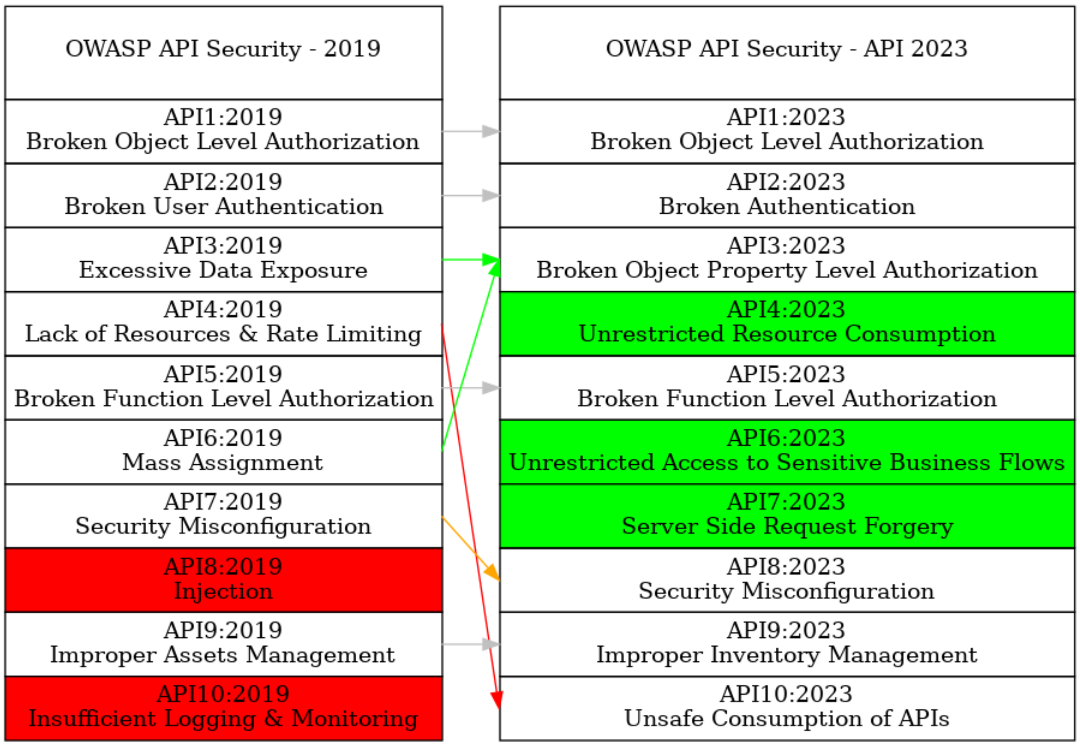

OWASP API Security - 2019
API1:2019
Broken Object Level Authorization
API2:2019
Broken User Authentication
API3:2019
Excessive Data Exposure
API4:2019
Lack of Resources & Rate Limiting
API5:2019
Broken Function Level Authorization
API6:2019
Mass Assignment
API7:2019
Security Misconfiguration
API8:2019
Injection
API9:2019
Improper Assets Management
API10:2019
Insufficient Logging & Monitoring
OWASP API Security - API 2023
API1:2023
Broken Object Level Authorization
API2:2023
Broken Authentication
API3:2023
Broken Object Property Level Authorization
API4:2023
Unrestricted Resource Consumption
API5:2023
Broken Function Level Authorization
API6:2023
Unrestricted Access to Sensitive Business Flows
API7:2023
Server Side Request Forgery
API8:2023
Security Misconfiguration
API9:2023
Improper Inventory Management
API10:2023
Unsafe Consumption of APIs

Figure 6: *OWASP API Security Project* changes from 2019 to 2023

API4:2023 deals with the fact that resources cost money and in times of cloud services these resources now cause transparent costs that should not be unnecessarily or unmanageably raised by API users. Therefore, points (e.g. execution timeouts) must be taken into account in the implementation.

API6:2023 addresses a point that seems to be aimed at the intended CSAF API (see Closing recommendation), since one of the intended API goals is automation.

"Secure and limit access to APIs that are consumed directly by machines (such as developer and B2B APIs). They tend to be an easy target for attackers because they often don't implement all the required protection mechanisms."

Closing recommendation of *Unrestricted Access to Sensitive Business Flows* [29]

Since the implementation of the CSAF API should also consider users who could become attackers by manipulating the query, this kind of neglect should not arise.

API7:2023 is about the fact that the API server could be used as a gateway if an attacker succeeds in providing the server with his own target addresses. The CSAF API should essentially address three services (file system, database and authentication) in the back-end. None of these targets should be dependent on the user's input to prevent API7:2023.

3 Design

The design of the CSAF API is kept simple. The client queries the API using *GraphQL*. The reason for this choice is described in chapter ¡2.6¿ This request is put into a simplified intermediate form. The API has one or more connectors (see figure 7). The connector's job is

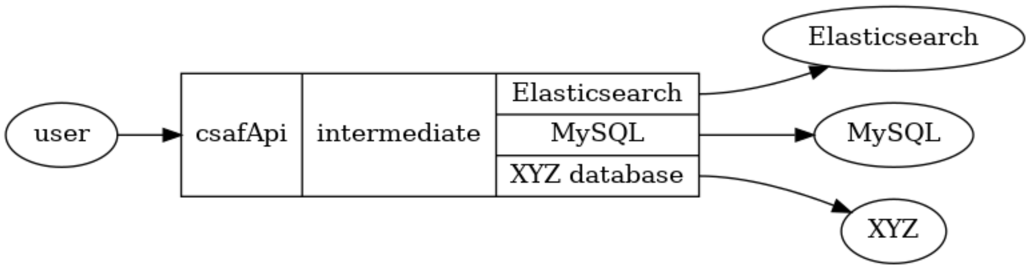


Figure 7: Simple CSAF API design

to put the intermediate form into a database-specific form. Depending on the connector, the *GraphQL* query (see listing 4) becomes an *Elasticsearch* (see listing 5), *MySQL* (see listing 6)

```
query {
  csafApi {
    findDocuments {
      must {
        product_tree {
          full_product_names {
            name (should: "Linux")
}}}}}}
```

Listing 4: GraphQL query where the full product name should contain "Linux"

or other database query.

```
{
  "query": {
    "bool": { "should": [{
      "bool": { "must": [{
        "match": { "product_tree.full_product_names.name": "Linux" }
      }]}
    }]}
  }
}
```

Listing 5: Elasticsearch query where the full product name should match "Linux"

Since queries can fail, error messages must be taken into account on the way back from the database to the client. Not all error messages should arrive unfiltered at the client, some of them are intended for administrators and must remain on the server (see figure 8). At least one database is required to run the API.

```
SELECT * FROM PRODUCT_TREE
    LEFT JOIN FULL_PRODUCT_NAMES
WHERE PRODUCT_TREE.ID = FULL_PRODUCT_NAMES.PRODUCT_TREE_ID
    AND FULL_PRODUCT_NAMES.NAME LIKE "Linux"
```

Listing 6: SQL query where the full product name should contain ”Linux”

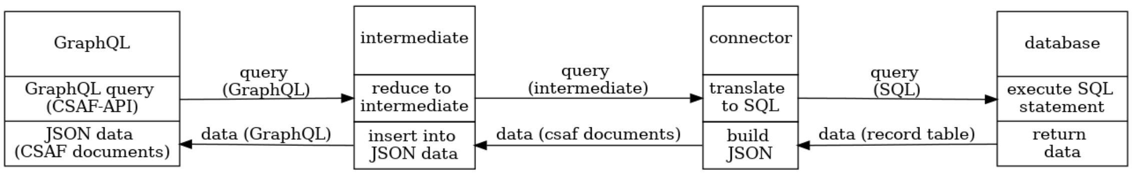


Figure 8: CSAF API data flow - with MySQL connector

3.1 Eleasticsearch

Elasticsearch is the storage component of the Elastic stack.

> *”Elasticsearch is the distributed search and analytics engine at the heart of the Elastic Stack. It provides near real-time search and analytics for all types of data. Whether you have structured or unstructured text, numerical data, or geospatial data, Elasticsearch can efficiently store and index it in a way that supports fast searches. Elasticsearch provides a REST API that enables you to store data in Elasticsearch and retrieve it. The REST API also provides access to Elasticsearch's search and analytics capabilities.”*

elasticsearch B.V.

There are alternatives to *Elasticsearch* such as the open-source project *OpenSearch* from Amazon Web Services, Inc. [1]. *Elasticsearch* was chosen based on prior experience. The API is to be designed to be open to databases, so all database-specific points are outsourced to connectors (see figure 7). To make it easier to implement connectors, an intermediate layer is included.

3.2 Intermediate

The intermediate stage is to be based on the *GraphQL* structure and reduce it to the absolutely necessary. Equivalent logical operators are to be harmonized. This means that a `csafAnd` should remain as it is and a `must` should be harmonized into a `csafAnd`. A complete list of logical operators and corresponding harmonization specifications is given in table 8.

Each connector gets the intermediate version of the *GraphQL* query, generates the database-specific query from it and converts the results into the structure expected by *GraphQL*. *GraphQL* expects the results as JSON objects under the path `/data`. Therefore, all found CSAF documents must be inserted into a JSON object under the path `/data`. On the one hand, the query is specified in *GraphQL*, while the response is returned as a JSON object.

GraphQL logical operator	**Intermediate harmonization**
csafAnd	csafAnd
must	csafAnd
csafOr	csafOr
should	csafOr
csafNot	csafNot
must_not	csafNot
exist	existp
existp	existp

Table 8: Harmonization of logical CSAF API operators

Because CSAF documents are also in JSON format, it makes sense to define JSON as the CSAF API format.

3.3 GraphQL

Queries are written in *GraphQL*. The query must be able to pass search parameters, filter parameters and a desired result structure. *GraphQL* must be able to cover requirements from the REST API. The new requirements include querying all CSAF document parts, dependent parts and, in the best case, future requirements. Strictly speaking, the query should not be unnecessarily restricted so that as many future use cases as possible can be covered with a *GraphQL* query.

Schmidt describes a REST API based on routes (e.g. `/csaf-documents/by-cve/`) in his thesis (see table 6). All these routes must be mapped with *GraphQL* in order to leverage their added value. Also, the CSAF objects have to be translated into *GraphQL* types.

The CSAF distribution brings several schema definitions. These have to be converted to *GraphQL* type definitions. Only then, *GraphQL* can use the objects to validate the input and output and structure the output. Table 9 lists all schema definitions. The schema definitions marked with `MUST` in the second column must be transferred. The remaining schema definitions are marked as `NICE TO HAVE`, because they are not part of the API.

CSAF schema definition file	**MUST or NICE TO HAVE**
aggregator_json_schema.json	NICE TO HAVE
csaf_json_schema.json	MUST
cvss-v2.0.json	MUST
cvss-v3.0.json	MUST
cvss-v3.1.json	MUST
provider_json_schema.json	NICE TO HAVE
ROLIE_feed_json_schema.json	NICE TO HAVE

Table 9: CSAF schema definitions by OASIS Open

3.3.1 Search and filter criteria

Since all components of a CSAF document are to be searched for, the various CSAF types and formats must be taken into account. Depending on the combinations of types, formats and enumerations, different search and filter parameters must be added as attributes. There are the following combinations of types, formats and enumerations:

- type: "string", format: "date-time"
- type: "string", format: "uri"
- type: "string", enum: []
- type: "string"
- type: "object"
- type: "number"
- type: "array"

When searching for a date string, four cases (before, exactly, after and in between) must be distinguished. It is also useful to be able to specify the date relative to now, so this is to be implemented as well. This simplifies automation on the client side, since the date no longer has to be determined there. Strings should be searched for in two ways. The exact search should find CSAF documents whose field matches exactly the searched term. This search is especially suitable for searching in `ID` fields. In the full text search, the search term must only be contained in the search field.

Objects can consist of other objects, strings, numbers, arrays or dates. They cannot be searched for directly, but their components can. A number can be less than, equal to, greater than, or within a range. Therefore, it should be possible to search according to these criteria. Arrays are collections of other arrays, strings, numbers, dates or other objects. As with objects, it is not possible to search directly for an array, but for its entries. It must be possible to filter array entries so that only array entries that the user requests are displayed.

When using an array filter in combination with the `must_not` condition, the user expects that all array elements that do not meet this condition will be filtered out. This must not result in filtering out CSAF documents that contain these unwanted array entries. Consequently, the `must_not` condition must not be taken into account when searching for array entries.

There are fourteen enumerations defined in the CSAF schema. The query with an enumeration value should behave exactly as in the exact search. Since there are no entries that differ from it, an approximate full text search makes no sense. An additional query for the exact value is superfluous, since *GraphQL* ensures that only values from the enumeration are considered. The search criteria should be passed as enumeration values and not as strings (see c0-c6 in listing 7).

```
query {
  categoryEnumeration: csafApi {
    findDocuments {
      csafOr {
        vulnerabilities {
          notes {
            c0: category (enum: DESCRIPTION)
            c1: category (enum: DETAILS)
            c2: category (enum: FAC)
            c3: category (enum: GENERAL)
            c4: category (enum: LEGAL_DISCLAIMER)
            c5: category (enum: OTHER)
            c6: category (enum: SUMMARY)
}}}}}}
```

Listing 7: Enumeration example query with GraphQL

Since *GraphQL* expects an enumeration in upper case (see *GraphQL* enumeration specification), all CSAF enumerations must be translated for the *GraphQL* type definition. For database queries, the *GraphQL* enumerations must be translated back to the CSAF enumeration. This translation must take place in the intermediate so that it is passed to all connectors in the same way.

> *"Enum values are represented as unquoted names (ex. MOBILE_WEB). It is recommended that Enum values be "all caps". Enum values are only used in contexts where the precise enumeration type is known. Therefore it's not necessary to supply an enumeration type name in the literal."*
>
> GraphQL enumeration specification by GraphQL Foundation

To comply with the *GrahpQL* enumeration notation, the enumerations are to be renamed. In most cases, capitalization is sufficient (see table 10). In a case `csaf_version: 2.0` a complete renaming must be done, because neither the dot as separator nor a number as start character is accepted by *GrahpQL*. In one case, nothing needs to be converted because the values there are already in uppercase. In this particular case, additionally the enumeration type can be set as *GraphQL* object type, because in this case the validation would not fail. It is not only the enumerations that need to be translated. To reduce the user's typing work, the short notation in the product queries must also be translated into the long notation.

JSON path	**Transformation**
.../branches[ ]/category	UPPERCASE
.../notes[ ]/category	UPPERCASE
.../references[ ]/category	UPPERCASE
/document/csaf_version	RENAME
/document/distribution/tlp/label	DO NOT TRANSFORM
/document/publisher/category	UPPERCASE
/document/tracking/status	UPPERCASE
/product_tree/relationships[ ]/category	UPPERCASE
/vulnerabilities[ ]/flags[ ]/label	UPPERCASE
/vulnerabilities[ ]/involvements[ ]/party	UPPERCASE
/vulnerabilities[ ]/involvements[ ]/status	UPPERCASE
/vulnerabilities[ ]/remediations[ ]/category	UPPERCASE
/vulnerabilities[ ]/remediations[ ]/restart_required/category	UPPERCASE
/vulnerabilities[ ]/threats[ ]/category	UPPERCASE

Table 10: Different CSAF enumeration cases

Products can be defined in three places in the CSAF document. By nesting `branches` elements, an infinite number of places in the CSAF document can contain product definitions. CSAF consumers should be able to formulate queries like the one in listing 8, with significantly less text.

```
query{ csafApi {
    findDocuments {
      csafOr {
        path1: product_tree { branches {
          product { name (should: "Linux") }
        } }
        path2: product_tree { branches { branches {
          product { name (should: "Linux") }
        } } }
        path3: product_tree { branches { branches { branches {
          product { name (should: "Linux") }
        } } } }
        path4: product_tree { branches { branches { branches { branches {
          product { name (should: "Linux") } }
        } } } }
        path5: product_tree { full_product_names {
          name (should: "Linux")
        } }
        path6: product_tree { relationships { full_product_names {
          name (should: "Linux")
        } } }
      }
    }
}}
```

Listing 8: Example search query for a product

So the query function `byProduct` is intended to avoid typing work for the CSAF consumer (see listing 9), especially because the CSAF consumer cannot know how many nesting levels exist. This avoided typing work has to be done by the intermediate (see subsection 3.2), so

```
query{
  byProductExampleQuery: csafApi {
    findDocuments {
      byProduct { name (should: "Linux") }
}}}
```

Listing 9: GraphQL query byProduct

that the connectors get the full long version of the query. The query is passed to the connectors as if the CSAF consumer had queried each CSAF product name path individually. To enable complete search queries, the maximum nesting depth must be determined in advance and stored in the intermediate configuration.

As with the nesting depth, it can also be assumed for the components of a CSAF document that the CSAF consumer does not know their existence or absence in advance. To avoid that CSAF consumers have to include all elements of a CSAF document in their queries, a shorthand notation is required.

3.3.2 Show all elements

GraphQL returns only that part of a JSON object that is part of the query (see figure 9). If

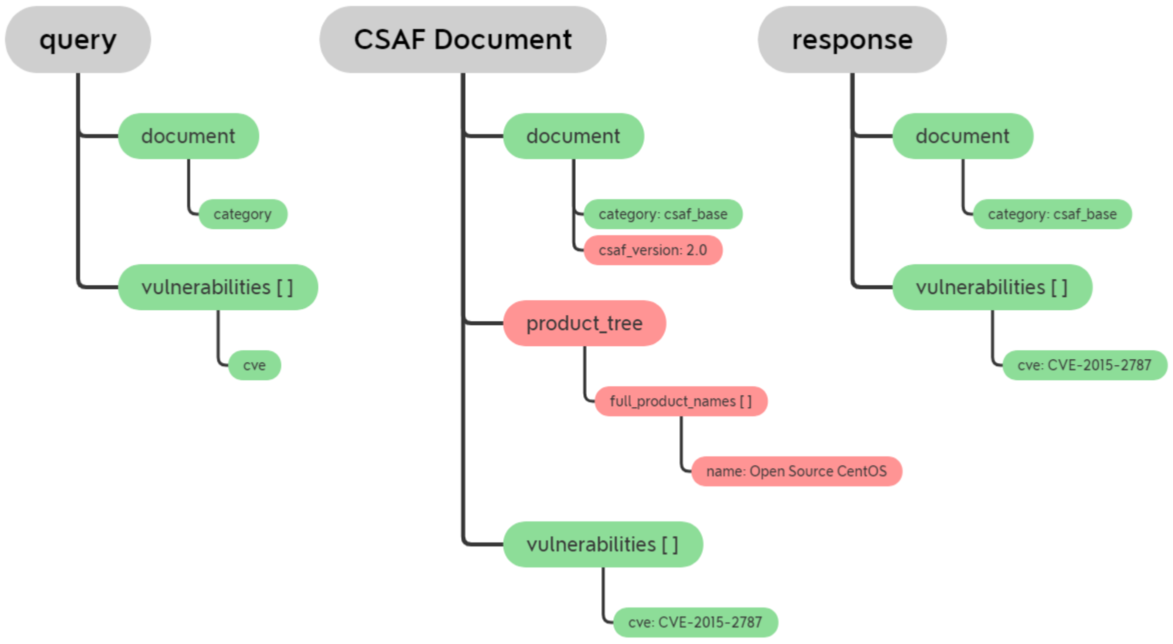


Figure 9: A GraphQL query and response for /document/category and /vulnerabilities[]/cve

the query contains only the paths `/document/category` and `/vulnerabilities[] /cve` and the found CSAF document contains other JSON elements `/document/csaf_version` and `/product_tree/full_product_names[]/name`, then only the requested JSON paths will be displayed in the results. The elements marked in red will not be included. To get the complete object all paths of the object must be part of the query. The CSAF consumer should not be required to type in the complete CSAF object structure, therefore a user-friendly solution is required with significantly less writing effort, which outputs all elements of the CSAF document on demand without having to request them individually. If the CSAF consumer does not specify a structure or paths, then all possible paths, i.e. the complete structure, should be displayed.

Before passing the connector result CSAF documents to *GraphQL*, the query of *GraphQL* must be manipulated (see figure 10). The query must contain all existing attributes of the result set in order for *GraphQL* to display all attributes.

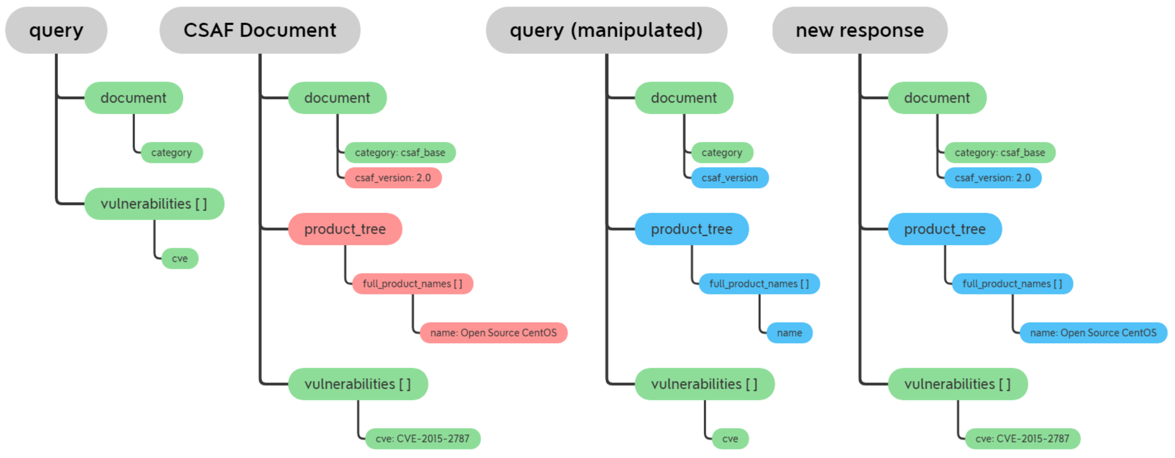


Figure 10: A *GraphQL* extended query and response

No matter what query a CSAF consumer enters, the response from intermediate to *GraphQL* must be the complete CSAF document. It is the job of *GraphQL* to take care of hiding, even if it has to be manipulated here and there by the intermediate.

The intermediate's responsibilities also include token verification. If, in addition to the query, the API user sends a token that he has received from the API operator, then this token must also be taken into account in the query, since authenticated users may be offered more CSAF documents.

3.3.3 Authentication

If the CSAF consumer, as a customer of the CSAF provider, has received a token from it. This token must be included with every CSAF API query in the headers (see token in listing 10). This token should be used to implement restrictions that affect the TLP label, so that only certain CSAF documents with the corresponding TLP label can be queried.

CSAF uses the TLP to specify the distribution. The Forum of Incident Response and Security Teams (FIRST) distinguishes four TLP labels (see table 3). Only CSAF documents with the TLP label `WHITE` will be delivered to an arbitrary and unauthenticated API user. To prevent CSAF documents with other TLP labels from being supplied, an additional condition must be appended to each query. For arbitrary users it is the restriction to the TLP

```
POST / HTTP/1.1
Host: localhost:4000
token: ***

query{
  byProductExampleQuery: csafApi {
    findDocuments {
      byProduct { name (should: "Linux") }
}}}
```

Listing 10: Token send via HTTP headers

label `WHITE` and for CSAF consumers additional granted TLP labels should be added (see listing 11). Which TLP labels the CSAF API operator grants to its CSAF consumers must be told to the API in the intermediate layer so that the TLP labels can be appended by the intermidiate accordingly.

```
query{
  csafApi {
    findDocuments {
      must {
        csafConsumerOriginalQueryExample:
          product_tree {full_product_names {name (should: "Linux")}}

        byIntermediateAppendedMustRestriction: csafOr {
          default:    document {distribution {tlp {label (enum: WHITE) }}}
          ifGranted1: document {distribution {tlp {label (enum: GREEN) }}}
          ifGranted2: document {distribution {tlp {label (enum: AMBER) }}}
          ifGranted3: document {distribution {tlp {label (enum: RED)   }}}
        }
}}}}
```

Listing 11: CSAF API append TLP

There are similar definitions (e.g. from US Cybersecurity & Infrastructure Security Agency [39] or Federal Office for Information Security (BSI) [8]), these can also be used. It is important that the TLP used is specified in the CSAF document as an attribute `/document/distribution/tlp/url`, but only the attribute `/document/distribution/tlp /label` is in the *CSAF json schema definition, Version 2.0* [22] marked as required.

Authenticity of the user to the API is important, just as important as the authenticity of the API to the user. Hash values were generated for each CSAF document (see subsection 2.4.4).

The hash values generated after the creation of the CSAF documents cannot be used for the validation of the *GraphQL* results, because the *GraphQL* results have gone through several manipulations before they are output and thus never match the original. So that the possibility for the validation is not completely lost, the intended CSAF API must offer these documents (CSAF document, PGP key and hashes) optionally as attachment.

3.4 CSAF API

The basic structure of CSAF API shall provide a search function `findDocuments` and a filter function `filterParameter` and shall allow structuring of the output `documents`. The fourth element `originalDocuments` is shown as needed and is not relevant to the query. Therefore, it is grayed out in figure 11. The gray marked `must`, `should` and `must_not` under `findDocuments` are equivalences to `csafAnd`, `csafOr` and `csafNot` and therefore marked in gray.

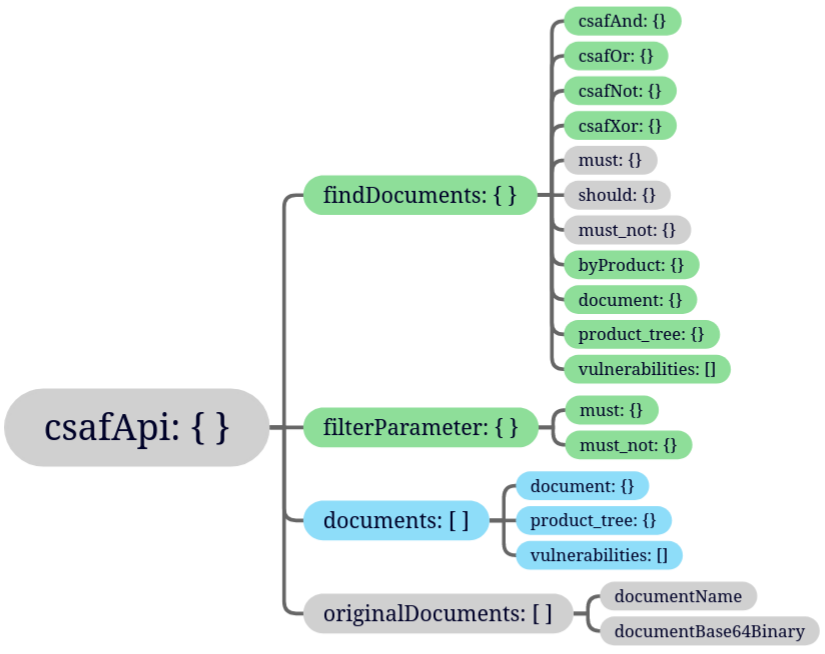


Figure 11: The basic design of the CSAF API

With `findDocuments` it shall be possible to search for CSAF documents directly by using the CSAF document structure and setting one or more arguments as search criteria (see figure 12). Paths without attributes accordingly do not contain any search criteria, they are irrelevant for the query and can be ignored. In figure 12, they are therefore grayed out.

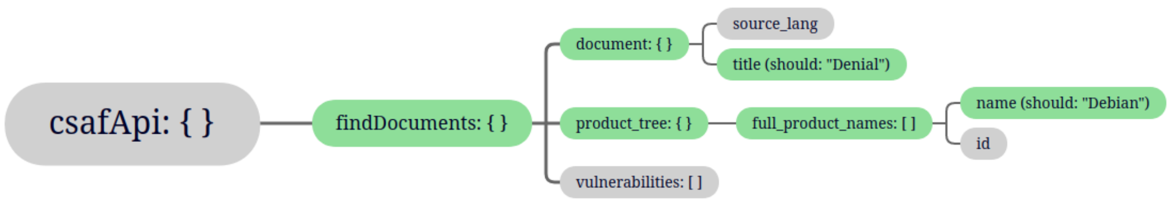


Figure 12: CSAF API - findDocuments

Search criteria can be specified under the `findDocuments` function. How these criteria relate to each other, whether they must all apply or only one of them, has yet to be determined. Listing 12 shows an `example` with two product names. These two products are obviously so different that they permit only one decision, it is searched for both products, these conditions must be `OR` concatenated.

```
query{ csafApi {
  example: findDocuments {
    product_tree { full_product_names {
      p1: name (should: "Linux")
      p2: name (should: "Windows")
    }}
  }
  extended: findDocuments {
    document { lang (should: "DE") }
    product_tree { full_product_names {
      p1: name (should: "Linux")
      p2: name (should: "Windows")
    }}
  }
  confusion: findDocuments {
    document { lang (should: "DE") }
    p1: product_tree { full_product_names { name (should: "Linux")   }}
    p2: product_tree { full_product_names { name (should: "Windows") }}
  }
}}
```

Listing 12: CSAF API function findDocuments examples

An additional search criterion is added, see `extended` example in listing 12. In this extended example, the search is obviously for the two products and the document language should be German. In this case, `findDocuments` must be an `AND` concatenation.

The `confusion` example shows that neither the `AND` concatenation nor the `OR` concatenation allow an intuitive usage. It is confusing because with no concatenation the obvious search result can be achieved, CSAF documents in German language for the two products `Linux` and `Windows`.

This confusion cannot arise if there is always a logical operator between `findDocuments` and a CSAF path (see listing 13). Since the `AND` concatenation yields fewer results than the `OR` concatenation, misbehavior with the `AND` concatenation would be noticed more quickly, `findDocuments` is therefore defined as an `AND` concatenation.

The logical operations `AND`, `OR` and `NOT` are sufficient for complex search queries. To make the query easier for the user or to save him typing work, the logical operator `XOR` should be implemented. This will make the notation much shorter and less complex. All logical operators can be nested within each other.

For readability, the equivalences `MUST`, `SHOULD` and `MUST_NOT` are to be implemented for the logical operators `AND`, `OR` and `NOT`. The operators `MUST`, `SHOULD` and `MUST_NOT` are redundant. They have been taken from the *Elasticsearch* context and have been left in because they are more meaningful in certain cases.

Here is an example: The searched document must contain `Linux` and `Kernel` in the title (see `example1` in listing 14). The query would start with `findDocuments` followed by `must`. Both criteria must apply, so these two query conditions can also be concatenated with the

```
query{ csafApi {
  example: findDocuments {
    csafOr {
      product_tree { full_product_names {
        p1: name (should: "Linux")
        p2: name (should: "Windows")
  }}}}
  extended: findDocuments {
    csafAnd {
      document { lang (should: "DE") }
      csafOr { product_tree { full_product_names {
        p1: name (should: "Linux")
        p2: name (should: "Windows")
      }}}
  }}
  nonconfusing: findDocuments {
    csafAnd {
      document { lang (should: "DE") }
      csafOr {
        p1: product_tree{ full_product_names{ name (should: "Linux")  }}
        p2: product_tree{ full_product_names{ name (should: "Windows")}}
  }}}
}}
```

Listing 13: CSAF API function findDocuments non confusing examples

logical `AND` operator (see `example2` in listing 14). For a query with only one criterion, the query would start with `findDocuments` followed by a `MUST` (see `example3` in listing 14). In this case, there is no `AND` operation, so `AND` instead of `MUST` would cause confusion (see example `confusing` in listing 14). In addition to searching for existing CSAF objects, it should also be possible to search for non-existing objects.

In order for the user to query for missing elements, the function `exist` must be implemented. The user should be able to select elements from the CSAF JSON structure as desired (see example e3 in listing 15). All selected CSAF elements should be present if the element `exist` is in a `must`. All CSAF elements must be missing if it is in a `must_not` (see example e4 in listing 15). Since existence is limited to only two states (exist or not), the use case with `should` makes no sense and therefore shall not be implemented.

Since *GraphQL* validates the query and the selection of objects is accepted only if at least one contained scalar is selected as well, an alternative function `existp` must be implemented. This alternative function `existp` should pass a JSON path as an attribute `path` (see example e2 in listing 15). This is to bypass the restriction of *GraphQL*. Instead of an array, it should also be possible to pass only a single value (see example e1 in listing 15).

This not only circumvents a restriction of *GraphQL* to meet the need, but also creates a vulnerability to injection attacks. This vulnerability must be eliminated by appropriate implementation. Vulnerability can be prevented by first validating the passed path against a

```
query{ csafApi {
  example1: findDocuments {
    must { document {
      criterion1: title (should: "Linux")
      criterion2: name (should: "Kernel")
  }}}
  example2: findDocuments {
    csafAnd { document {
      criterion1: title (should: "Linux")
      criterion2: name (should: "Kernel")
  }}}
  example3: findDocuments {
    must { document { title (should: "Linux") }}
  }
  confusing: findDocuments {
    csafAnd {document { title (should: "Linux") }}
  }
}}
```

Listing 14: CSAF API reasons for redundancies

```
query{ exist: csafApi { findDocuments {
  must {
    e1: existp (path: "/document/title")
    e2: existp (path: ["/document/lang", "/document/title"])
    e3: exist {
      document {
        lang
        title
    }}
  }
  must_not {
    e4: exist {
      document { title }
  }}
}}}
```

Listing 15: GraphQL exist query

full list of allowed paths and then passing it to the database. A validation routine must be implemented in the intermediate layer.

If the criteria from the `findDocuments` match, then the found CSAF documents are returned completely. If a specific array content is to be searched for, then a complete array list can lead to confusion, since not only the searched contents are listed. Therefore, and in order to make the overview better, an array filter is to be implemented. This array filter is to clean the results, so that in the results arrays only entries are to be found, for which one searched.

As with search attributes (see subsection 3.3.1), filter attributes must also be differentiated by type. The array can consist of JSON basic types `string`, `date` or `number` or complex objects. Filters for the following JSON types, formats and enumerations are required:

- type: "string" format: "date-time"
- type: "string" format: "uri"
- type: "string" enum: [ ]
- type: "string"
- type": "number"

No additional attributes need to be created in the *GraphQL* type definition for filtering, the attributes from the search function should be reused. Array filters on integrated arrays and objects should not be implemented (see red marked in figure 13). If all elements of an array

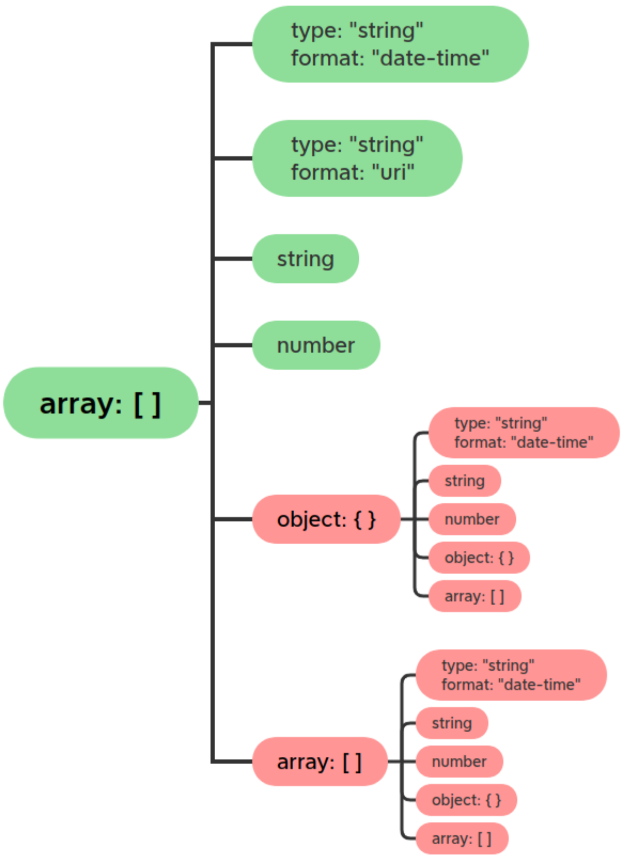


Figure 13: CSAF API array filter

are filtered out due to filter criteria, obviously the parent object was not searched for. The parent object would have to be removed, since consciously set filter criteria do not apply to this object. Since filtering happens after the database query, it must be implemented in the intermediate or the step after it, at the latest in the last step before handover. Because `filterParameter` distinguish by `must` and `must_not` (see figure 11), they must be taken into account in the filter function. Since a `should` would include both results of a `must` and results of a `must_not`, filtering with `should` makes no sense. Therefore, only these two operators `must` and `must_not` are to be implemented.

With the CSAF API it should be possible to search for all parts of a CSAF document (e.g. `/document/category` and `/vulnerabilities[]/cve`), the results should be filtered (e.g. for `/product_tree/full_product_names[]/name`) and in the best case only the desired parts of the document are displayed, as in figure 14.

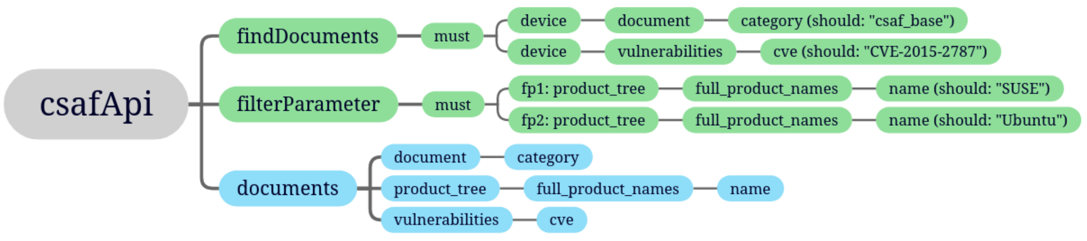


Figure 14: CSAF API Query with filtered array elements

To keep the overview `filterParameter` intervenes in the results. To increase this overview even more, the output structure can be customized with `documents`. If the `documents` element is missing in the query, then all CSAF document elements should be displayed. If `documents` is present, then only the elements defined in `documents` should be displayed (like the `/document/title` in figure 15). With the `documents` function, an API user can selec-

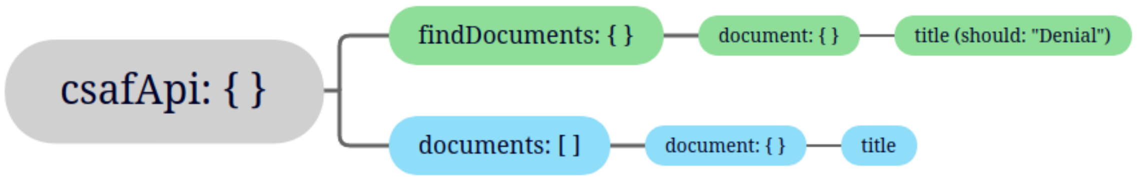


Figure 15: Show only the document title

tively retrieve elements of the CSAF document. The possibility to query original documents should not disappear completely.

To be able to check authenticity on the client side, clients need to be able to download original documents, signatures and hashes (see subsection 2.4.4). In such a case, the attribute `originals` can be added to the query (see listing 16).

```
query {
  max_documents_1: csafApi (max_documents: 1, originals: true) {
    findDocuments { document {title (should: "Linux") } }
  }
}
```

Listing 16: GraphQL query - originals

The original documents should appear in the response under the item `/data/[csafApi || alias]/originals`. So that *GraphQL* does not modify the original CSAF document, they must be passed `Base64` encoded (see listing 17). The functionality of the API is now complete

```
{
  "data": {
    "max_documents_1": {
      "documents": [...],
      "originalDocuments": [
        {
          "documentName":         "XYZ.json",
          "documentBase64Binary": "ewogICJkb2N1bWVudCIgOiB7C ..."
        },
        {
          "documentName":         "XYZ.json.asc",
          "documentBase64Binary": "LS0tLS1CRUdJTiBQR1AgU01HT ..."
        },
        {
          "documentName":         "XYZ.json.sha256",
          "documentBase64Binary": "ZTI0NGFlOWUwYzA1ZDRiMmYxZ ..."
        },
        {
          "documentName":         "XYZ.json.sha512",
          "documentBase64Binary": "Y2FlZWRkYjYxOGRkMGJjYWFmM ..."
        }
      ]
}}}
```

Listing 17: GraphQL response - originals

with this `originals` functionality. For the next subsection, a step back must be taken so that process speed-up potentials become visible.

3.4.1 Automation

After a CSAF document is created, it can be published by placing it on the server. It may take an indefinite time before the document is retrieved by a CSAF consumer. It takes time for the CSAF consumer to determine if any of their products are affected by the vulnerability inside the CSAF document. Depending on the downstream process, it may take an indeterminate amount of time for this information to reach a responsible administrator.

Therefore, manufacturers could come up with the idea of integrating CSAF API queries into their product, e.g. in a management dashboard if one exists. Relevant information would reach key persons (e.g. administrators, application or system owners, or other roles with responsibility) directly.

References to CSAF documents could come in by ticker. For a ticker query the CSAF elements `/document/title`, `/document/tracking/revision_history[]/date` and `/document/tracking/revision_history[]/summary` seems to be well suited and for more detailed information the element `/document/notes[]/text` (see listing 18). Other useful information can also be included (e.g.: `/vulnerabilities[]/scores[]/cvss_v3/baseScore`); Only if the administrator wants more information the complete CSAF document would be displayed.

```
query { tickerQuery: csafApi {
  findDocuments { byProduct { name (exact: "Red Hat Enterprise Linux")}}
  documents {
    document {
      title
      tracking {
        revision_history {
          date
          summary
        }
      }
      notes { text }
    }
  }
}}
```

Listing 18: Ticker query

These queries can be narrowed down specifically to the product used (see `byProduct` example in listing 18). As the manufacturer knows the third-party components used in his product, these products can additionally be included in the query. Manufacturers, CSAF consumers and also API operators are interested in stable and secure API operation.

3.4.2 Security

The CSAF API offers infinite nesting (see `branches` in figure 2), this creates the risk that Apollo Graph, Inc warns about under `Mitigate malicious queries` [2] . This risk should be mitigated by limiting the nesting depth in a configuration file. OASIS Common Security Advisory Framework (CSAF) Technical Committee says a maximum nesting depth of 25 is sufficient [19], so the configuration value should not exceed this recommendation.

In addition to protecting the CSAF API resources and thus protect the availability of CSAF API from unconsidered queries, a maximum of `10` CSAF documents should be delivered by default. This value must be configurable so that it can be adjusted by the API administrator. This preset must be allowed to be overridden by the CSAF consumer up to a certain threshold.

There is a potential risk that the authenticity of the original files is checked (see subsection 3.4), but then the unverifiable API response (see subsection 3.3.3) is processed further. CSAF consumers should continue the process with the original CSAF documents whenever possible, despite the many conveniences of the API.

The CSAF API receives requests from the CSAF consumer, converts them into database queries, and returns the results. Except for writing the log files, no write access is required. Therefore, all permissions of the user running the API should be restricted to read-only, except for the log directory. Additional functionalities have to be implemented for the realization of the security requirements.

3.4.3 Set number of results - max_documents

Since the maximum number of results should be set to a default value for security reasons (see previous subsection 3.4.2), the CSAF consumer must be given the option to set a different value. It should be possible to increase the limit of the results with the `max_documents` argument (see listing 19).

```
query {
  max_documents_1: csafApi (max_documents : 1){
    findDocuments { document {title (should: "Linux") } }
  }
  max_documents_2: csafApi (max_documents : 2) {
    findDocuments { document {title (should: "Windows") } }
  }
}
```

Listing 19: GraphQL max_documents query

As a result, the first query `max_documents_1` would return exactly one CSAF document and the second query `max_documents_2` would return two CSAF documents (see listing 20). Since the information about how many documents can be found with the query is lost when

```
{
  "data": {
    "max_documents_1": {
      "documents": [
        {...}
      ]
    },
    "max_documents_2": {
      "documents": [
        {...},
        {...}
      ]
    }
}}
```

Listing 20: GraphQL max_documents results

the result list is limited, additional functionality must be implemented.

3.4.4 Show results metadata

The default limitation of the search results (see previous subsection 3.4.3) could give the wrong impression that only this amount of CSAF documents could be found. Therefore, the CSAF consumer should be given the possibility to display the real results size using `metadata` (see listing 21). The real results size of found CSAF documents should be attached as a `/data/[csafApi || alias]/metadata/total` object (see listing 22). Errors can occur when processing metadata and also in other places, so error handling must be implemented.

```
query {
  max_doc_1: csafApi (max_documents: 1, metadata: true) {
    findDocuments { document {title (should: "Linux") } }
  }
  max_doc_2: csafApi (max_documents: 2, metadata: true) {
    findDocuments { document {title (should: "Windows") } }
  }
}
```

Listing 21: GraphQL metadata query

```
{
  "data": {
    "max_doc_1": { "documents":[{...}],       "metadata": { "total": 753}},
    "max_doc_2": { "documents":[{...},{...}], "metadata": { "total": 34}}
}}
```

Listing 22: GraphQL metadata results

3.4.5 Error handling

As in any application, error messages can occur in the API. Two types of error messages must be distinguished, error messages for the user and for the administrator. Error messages for administrators must be written to log files. Error messages for the user must be passed to the user, they should be part of the response (see figure 16). As mentioned before (see

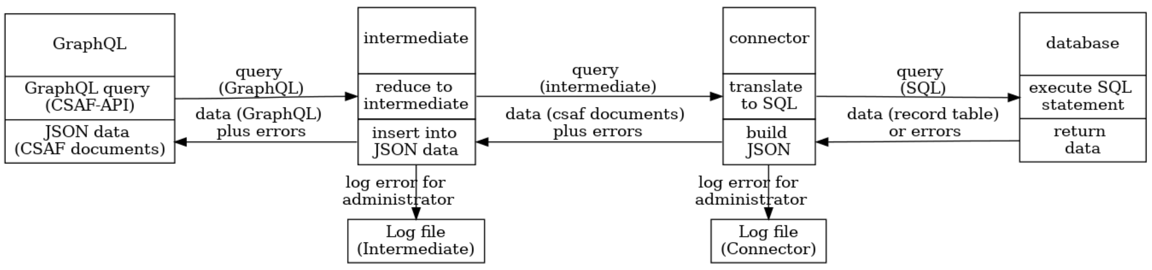


Figure 16: Error flow

subsection 3.4.2), writing the log files is supposed to be the only write access of the API. If an error occurs, as with the query in listing 23, a `csafXor` with three conditions instead

```
query{
  erroneousExample: csafApi {
    findDocuments {
      csafXor {
        condition1: existp (path: "/document")
        condition2: existp (path: "/product_tree")
        condition3: existp (path: "/vulnerabilities")
}}}}
```

Listing 23: Error-causing query

of two should be used, then the error message must be passed to the CSAF consumer (like listing 24).

```
{
  "data": {
    "erroneousExample": {
      "error": "csafXor allows exactly two conditions."
}}}
```

Listing 24: Response of the error-causing query

Depending on where the error occurred (database, connector or intermediate), it must be ensured that the CSAF consumer only receives the information it needs. Under no circumstances should information about the database, its structure, the application server or other security-relevant information be leaked. No error message may be forwarded unfiltered.

Even with `csafXor`, requirements (see subsection 1.3) cannot be implemented with exactly one database query. Especially the dependent queries can fail due to the possibilities of the database. To be able to serve these requirements nevertheless, an additional functionality must be implemented.

3.4.6 Pre-query the product IDs

For people responsible for IT security, the question is whether and how a product for which they are responsible is affected by a vulnerability. Whether a product ID is affected by a vulnerability is stored in the vulnerability product status `/vulnerabilities[]/product_status`. The people responsible for IT security do not know the IDs assigned in the CSAF document, so they cannot query them directly. The assigned product IDs for the product names must first be found out. This can be found out by searching all places where product ids are defined (see table 4) with the help of the product name.

In a relational database that supports Structured Query Language (SQL), the query could start on the products with a condition (e.g. `PRODUCTS.NAME LIKE "Linux"`) and the vulnerabilities product statuses could be joint (e.g. `ON P.ID = V.PRODUCT_ID`) (see listing 25 for the entire SQL Statement).

```sql
SELECT * FROM PRODUCTS P
  LEFT OUTER JOIN VULNERABILITIES_PRODUCT_STATUS V
    ON P.ID = V.PRODUCT_ID
WHERE P.NAME LIKE "Linux"
```

Listing 25: Show product status of vulnerabilities by specifying the product name

This advantage does not exist in databases like *Elasticsearch*, which store CSAF documents as complete JSON documents. In order to serve the request, a first query must find documents with the corresponding product names and then start a new query with the found product IDs. First, all products corresponding to the product name must be found in the CSAF document. With their IDs, a second query must be performed in the product statuses of the vulnerabilities. The situation is complicated by the fact that the data source is not a single CSAF document, rather many. All CSAF documents must be queried for products that match the search term. With the product IDs found from the many CSAF documents, a second query must be run.

Since the product ID must only be unique within the CSAF document (see document tracking ID description in subsection 2.4.2.1), additional search parameters must be provided for the second query the document tracking id `/document/tracking/id`. Aggregators that provide CSAF documents from different publishers must also include the publisher namespace `/document/publisher/namespace` to be unique. If the first query finds a large amount of results, the entire system can quickly reach its limits when executing the second query. The query for products whose ID is in a vulnerability product status must be implemented without running into the problem described.

This circumstance can be bypassed in the connector module. The first query should return CSAF documents containing the requested product names. A second query with the product IDs becomes unnecessary if CSAF documents are filtered out in the connector module that do not match the query. This shifts database activity to the Connector, but prevents oversized queries.

In order to test all functionalities, a database (in this case *Elasticsearch*) is required, so the existing CSAF documents have to be migrated into the database.

3.5 Migration respectively synchronization

The API design intends to use a database. This means that the existing CSAF documents must be migrated to a database so that the CSAF API can search for the documents there. In the case of *Elasticsearch*, the CSAF documents can be uploaded to an *Elasticsearch* index unchanged. After uploading, the CSAF documents are already fully queryable. In the case of *MySQL*, the CSAF documents must be fragmented into their individual components and inserted into corresponding database structures. A database structure is not given in this thesis, therefore it can differ from API operator to API operator. Consequently, the *MySQL* connector, which is to be implemented only rudimentarily, must be adapted to each future API operator.

4 Implementation

After the design in the broadest sense reflects the expectations, the implementation will show the concrete design of the CSAF API.

4.1 GraphQL

GraphQL is the language in which the CSAF API is queried (see figure 7), so it makes sense to start here. In order to query the API, the query itself `csafApi` must be defined first (see listing 26) and then the respective components of the query. The query should return CSAF

```
type Query {
  csafApi: [csafDocument]
}
```

Listing 26: CSAF API type definition

documents, so it gets the type `csafDocument`. Because it is intended to be an array of CSAF documents, the type is enclosed in square brackets `[csafDocument]`. The query is now ready, only the used type `csafDocument` and types following it have to be defined.

4.1.1 Type definition

GraphQL expects a type definition, on the one hand to be able to validate the query and on the other hand to be able to reduce the output of the results to defined types and thus approved objects. There are several tools on the Internet that can translate an JSON schema, and thus also the CSAF JSON schema, into an *GraphQL* type definition. Here the manual process is described. This example will show how easy it is to convert a JSON schema (see reduced CSAF document schema in listing 27) into a *GraphQL* type definition (see listing 28).

```
{
  "properties": {
    "document": {
      "type": "object"
    },
    "product_tree": {
      "type": "object"
    },
    "vulnerabilities": {
      "type": "array"
    }
  }
}
```

Listing 27: CSAF schema snippet - CSAF document

A CSAF document `csafDocument` consists of two objects `/document` and `/product_tree` and an array `/vulnerabilities[]`. These three parts can be found in the reduced CSAF document schema under `/properties` (see previous listing 27). Their names (`document`, `product_tree` and `vulnerabilities`) must be copied unchanged from the schema to the *GraphQL* type definition (see listing 28).

```
type csafDocument {
    document: csafDocumentDocument
    product_tree: csafDocumentProductTree
    vulnerabilities: [csafDocumentVulnerabilities]
}
type csafDocumentDocument {}
type csafDocumentProductTree {}
type csafDocumentVulnerabilities {}
```

Listing 28: translated CSAF schema snippet - CSAF document

A new type must be created for each of the properties. If the property type is an array (e.g. `/properties/vulnerabilities/type` in listing 27), the *GraphQL* property type is specified in square brackets `vulnerabilities: [csafDocumentVulnerabilities]`, as the `csafApi: [csafDocument]` before. The type definition of `csafDocument` is completed. The newly created but empty types `csafDocumentDocument`, `csafDocumentProductTree` and `csafDocumentVulnerabilities` must be filled with their properties and so on.

The elements of the type `string` are interesting (see `/properties/document/properties /category` in listing 29), because users should be able to search for them. Therefore, search

```
{
  "properties": {
    "document": {
      "type": "object",
      "properties": {
        "acknowledgments": { "$ref": "#/$defs/acknowledgments_t" },
        "aggregate_severity": {"type": "object"},
        "category": {"type": "string"},
        "csaf_version": {
          "type": "string",
          "enum": { "0": "2.0" }
        },
        "distribution": {"type": "object"},
        "lang": {"$ref": "#/$defs/lang_t"},
        "notes": {"$ref": "#/$defs/notes_t"},
        "publisher": {"type": "object"},
        "references": {"$ref": "#/$defs/references_t"},
        "source_lang": {"$ref": "#/$defs/lang_t"},
        "title": {"type": "string"},
        "tracking": {"type": "object"}
}}}}
```

Listing 29: CSAF schema snippet - CSAF document - document

parameters are defined in addition to the element type. There is a separate subsection on search criteria (subsection 4.1.2), so they are only briefly explained here. Contents of the type string can be searched for either exactly or approximately. Therefore, the user is given the possibility to specify an exact or an approximate search term as attributes `exact` or `should`.

Another special case is formed by the objects that are reused respectively referenced in the schema (e.g. `/properties/document/properties/acknowledgements/$ref` in listing 29). These predefined objects can and should be created with their own name as type name. The next special case is enumeration, which is described in more detail in subsection 4.1.1. A brief note here, the enumeration values cannot always be copied. This applies especially to the case `/properties/document/properties/csaf_version`, since a forbidden separator *dot* is contained and the enumeration value `2.0` begins with a number. The fully translated CSAF document object from listing 29 is in listing 30.

```
type csafDocumentDocument {
    acknowledgments: [acknowledgments_t]
    aggregate_severity: csafDocumentDocumentAggregateSeverity
    category( exact: String
              should: String): String
    csaf_version( exact: String
                  should: csav_versionEnum): String
    distribution: csafDocumentDocumentDistribution
    lang( exact: String
          should: String): String
    notes( audience: String
           category: String
           text: String
           title: String): [notes_t]
    publisher: csafDocumentDocumentPublisher
    references: [references_t]
    source_lang( exact: String
                 should: String ): String
    title( exact: String
           should: String ): String
    tracking( id: String): csafDocumentDocumentTracking
}
type acknowledgments_t {}
type csafDocumentDocumentAggregateSeverity {}
type csafDocumentDocumentDistribution {}
type notes_t {}
type csafDocumentDocumentPublisher {}
type references_t {}
type csafDocumentDocumentTracking {}
enum csav_versionEnum {}
```

Listing 30: Type definition of CSAF document type

The following listing 31 shows various type definitions occurring in the CSAF schema. For each of these definitions there is a corresponding entry in listing 32. This converts the CSAF JSON schema to the *GraphQL* type description. If CSAF documents are now passed

```
"object_t": {
  "properties": {
    "integer":             {"type": "number"},
    "text":                {"type": "string"}
  },
  "type": "object"
}
"object": {
  "properties": {
    "integer":             {"type": "number" },
    "floatnumber":         {"type": "number" },
    "text":                {"type": "string" },
    "true_or_false":      {"type": "boolean"},
    "date":                {"type": "string" },
    "complexObject":       {"$ref": "#/object_t"},
    "array_of_integergs": {"type": "array", "items": {"type": "number"  }},
    "array_of_floats":    {"type": "array", "items": {"type": "number"  }},
    "array_of_strings":   {"type": "array", "items": {"type": "string"  }},
    "array_of_booleans":  {"type": "array", "items": {"type": "boolean" }},
    "array_of_objects":   {"type": "array", "items": {"$ref": "#/object_t"}},
    "enumeration":         {"type": "enum"}
  },
  "type": "object"
}
```

Listing 31: JSON example

```
type object {
  #scalars
  integer:          Int
  floatnumber:      Float
  text:             String
  true_or_false:   Boolean
  #no special date type, just a string
  date:             String
  #objects
  complexObject:    object_t
  #arrays
  array_of_integergs: [Int]
  array_of_floats:    [Float]
  array_of_strings:   [String]
  array_of_booleans:  [String]
  array_of_objects:   [object_t]
  enumeration:            uppercaseStringEnum
}
type object_t {
  integer: Int
  text: String
}
enum uppercaseStringEnum {
  STRING_IN_UPPER_CASE_ONE
  STRING_IN_UPPER_CASE_TWO
}
```

Listing 32: JSON example translated into GraphQL

to *GraphQL* as a response, then *GraphQL* can validate, process and return them to a CSAF API user. For validation, the enumerations must be translated from the CSAF schema to *GraphQL* type definition. There are 14 enumerations in the CSAF schema (see table 10). One of them `/document/csaf_version` had to be renamed for *GraphQL*, so that it can be used as an enumeration in the `enum` search parameter (see table 33). The type of `csaf_version`

```
enum csafDocumentDocumentCsafVersionEnum {
  # 2.0 fully renamed
  V2_0
}
type csafDocumentDocument {
  ...
  csaf_version( enum: csafDocumentDocumentCsafVersionEnum ): String
  ...
}
```

Listing 33: Enumeration type definition with GraphQL

must not be set to `csafDocumentDocumentCsafVersionEnum`. If the type would also be `csafDocumentDocumentCsafVersionEnum`, then the results would fail during validation. Therefore it must remain of type `String`. In the case of `/document/distribution/tlp/label` not only the `enum` type but also the TLP label type was set to `csafDocumentDocumentDistributionTlpEnum` (see listing 34). In this case, the results must take exactly these

```
enum csafDocumentDocumentDistributionTlpEnum{
  # Enumeration was already in upper case,
  # it can be used as object type and attribute type
  AMBER
  GREEN
  RED
  WHITE
}
type csafDocumentDocumentDistributionTlp {
    label( enum: csafDocumentDocumentDistributionTlpEnum
        ): csafDocumentDocumentDistributionTlpEnum
    ...
}
```

Listing 34: GraphQL TLP Enumeration type definition

values. This completes the preliminary work for the CSAF API.

4.1.2 CSAF API

The CSAF API is defined as a query in the *GraphQL* type description (see listing 26). Since the CSAF API must provide many different functions (e.g. `findDocuments`, `filterParameter`, . . .), the `csafApi` type from listing 26 must be significantly extended. The API must be multi functional, like the egg-laying, milk-giving mammal with woolly base hair and a beak, the platypus. Therefore, the new `csafApi` type is named `csafApiPlatypus`. All functions are inserted there (see listing 35). The query `csafApi` itself gets three additional attributes.

```
type Query {
    csafApi(
        max_documents: Int
        originals: Boolean
        metadata: Boolean
    ): csafApiPlatypus
}
type csafApiPlatypus {
    findDocuments: csafApiPlatypusSearchCriteria
    filterParameter: csafApiPlatypusFilterCriteria
    documents: [csafDocument]
    originalDocuments: [csafOriginalDocuments]
    metadata: csafResultsMetadata
    error: String
}
```

Listing 35: CSAF API definition

These attributes are `max_documents` (see subsection 3.4.3), `originals` (see subsection 3.4) and `metadata` (see subsection 3.4.4). `max_documents` restricts the result set to the desired number of documents (detailed information is given later in subsection 4.3), `originals` ensures that the original documents are attached (detailed information is given later in subsection 4.2), and `metadata` tells the CSAF consumer the exact number of CSAF documents that could be retrieved with this query without any `max_documents` limitation.

It should be possible to search for elements directly (see subsection 3.3.1) or indirectly (see subsection 3.4.6). Attributes such as `should` have been added for direct searches (for example see `category` in the appendix in listing 83). Attributes such as `idInVulnerabilitiesProductStatusFixed` have been added for indirect search (for example see `product_id` in the appendix in listing 84).

If the document current release date `/document/tracking/current_release_date` is to be searched by, the date can be specified either approximately, exactly, or as a range (see example in listing 36).

```
query{ dateCriteria: csafApi {
  findDocuments {
    csafOr { document { tracking {
      approximately: current_release_date (should: "2023-03")
      exactly: current_release_date ( exact: "2023-03-15T23:00:00.000+00:00")
      asRange: current_release_date ( younger: "2023-03", older: "2023-04")
}}}}}}
```

Listing 36: CSAF API date search parameter

Search criteria can be `OR` concatenated with each other. Thus, not only different time periods can be defined in the query (see `csafOr` in listing 36), but also different products.

4.1.3 Product list

Several products can be affected by one vulnerability. If each product is queried separately, then it cannot be avoided that CSAF documents that have already been downloaded are downloaded a second time. This can be avoided by making a single query and downloading all relevant CSAF documents at once (see listing 37). The restriction `max_documents` must

```
query{
  productlist: csafApi (max_documents: 2000, metadata: true) {
    findDocuments {
      csafOr{
        p1: product_tree{full_product_names{name(should: "SUSE")}}
        p2: product_tree{full_product_names{name(should: "Ubuntu")}}
      }
    }
  }
}
```

Listing 37: GraphQL CSAF API Query - product list

be set accordingly high, so that all CSAF documents are delivered that match the query. The parameter `metadata` additionally returns the maximum number of documents matching the query. If the returned value is greater than `max_documents`, then not all documents were delivered. `max_documents` is preset to 10 for security reasons as long as the query does not specify a value. This query returns CSAF documents with the searched product name. The array `product_tree/full_product_names[]` may contain many additional products that do not match the search query. The array filter function can be used to remove them from the array.

4.1.4 Array Filtering

The function `filterParameter` (see subsection 3.4) is implemented as a resolver. There is one resolver for the query `csafApi` and one resolver per array parent (see example for `/document/acknowledgments[]` in listing 38). Resolvers are called by *GraphQL* for each recognized *GraphQL* type, if they exist respectively are implemented. In this example, the resolver is called on the CSAF document object `acknowledgements`. The `filterArray_V2` function return a cleaned respectively filtered array. In the filter function `filterParameter`, as well as in the search function `findDocuments` there is the possibility to query an unlimited number of nestings, this bears risks (see branches in subsection 2.4.2.2).

4.1.5 Security

To mitigate the risk of infinite nesting (see subsection 3.4.2), the nesting depth parameter must be set (see listing 39). This parameter describes the allowed nesting depth, all nestings

```
csafDocumentDocument: {
  // /document/acknowledgments[] has arrays
  // that is why it needs to be filtered
  acknowledgments:(root, obj, args, context, info) => {
    var filteredArray = root.acknowledgments;
    if(filteredArray){
      //names Filter
      filteredArray = graphQLhelper.filterArray_V2(
        context, filteredArray,  originalCsafApiFilter,
        "/document/acknowledgments", "names"
      );
      //urls Filter
      filteredArray = graphQLhelper.filterArray_V2(
        context,  filteredArray,  originalCsafApiFilter,
        "/document/acknowledgments", "urls"
      );
    }
    return filteredArray;
  }
}
```

Listing 38: GraphQL array filtering on /document/acknowledgments[]

```
const config = require("./config");
const depthLimit = require("graphql-depth-limit");
const { ApolloServer } = require("apollo-server");
const typeDefs = require("./typeDefs");
const resolvers = require("./resolvers");
const server = new ApolloServer({  typeDefs, resolvers,
  introspection: config.csafServer.csafServerStage !== "production",
  context:({req})=>{const token=req.headers.token || null; return {token};},
  validationRules: [ depthLimit(config.csafServer.csafDepthLimit) ]
});
server.listen().then(({ url }) => {console.log("Server ready at ${url}");});
```

Listing 39: depthLimit definition in the server startup script - server_V20.js

are counted starting from `findDocuments` (for determination of the maximum see table 11). It can be seen from the table 11 that the different CSAF document elements have different maximum nesting depths. The first three examples represent the deepest nesting depths (with out unlimited nesting) of the three CSAF document components. The two special cases show the real problem, the possibility of infinite nesting. In order to use all elements as query parameters, a minimum nesting depth of `8` must be allowed. To allow more complex logical operations like a XOR (coming later in listing 61), two more nesting depths must be allowed. Therefore, the maximum nesting depth of `10` was chosen here. This setting can be customized by the CSAF API administrator (see listing 40), but it does not need to be greater than `35` (see subsection 3.4.2).

GraphQL query	**depth**
query{csafApi{	
findDocuments{ must{	2
document{tracking{generator{engine{name}}}}	+5
}}	
}}	=7
query{csafApi{	
findDocuments{must{	2
product_tree{branches{product{product_identification_helper{	+6
hashes{filename}}	
}}}	
}}	
}}	=8
query{csafApi{	
findDocuments{must{	2
vulnerabilities{remediations{restart_required{details}}}	+4
}}	
}}	=6
special case I:	
query{csafApi{	
findDocuments{	1
csafAnd{csafAnd{csafAnd{csafAnd{csafAnd{. . . }}}}}	+∞
}	
}}	=∞
special case II:	
query{csafApi{	
findDocuments{	1
must{	+1
product_tree{branches{branches{. . . }}}}	+∞
}	
}}	=∞

Table 11: Examples of nesting depth

The next security consideration is file access. To enable authenticity for the CSAF customer, the CSAF documents are searched as originals on the file system and attached to the CSAF API results. For the search file system access is needed. If an attacker succeeds to pass his own file system search terms to this procedure, the attacker gains access to the file system with the rights of the user running the CSAF API. Therefore, on the one hand, the permissions of the user running the API must be limited to the least necessary. The user running CSAF API needs read permissions to the CSAF documents directory and must not be an administrator. And on the other hand, it must be impossible for a attacker to pass his own file names to the procedure.

The permissions for the user running the CSAF API must be restricted by the CSAF API administrator. File names cannot be passed to the procedure directly by the attacker. The file names are taken from the results `/document/tracking/id`. As long as an attacker does

```
/**
 * csafServerStage: "production" means, that the introspection of GraphQL is
 * deactivated for security reasons
 *  Just use "test", "integration", "engeneering" or something else to have
 *  the introspection active
 *
 * csafIntermediate:   connector: "./elasticsearch/connector"  OR
 * csafIntermediate:   connector: "./mysql/connector"
 */
 module.exports = {
    csafServer: {
        csafDepthLimit: 10,
        csafServerStage: "engeneering"
    },
    csafIntermediate:  {
        connector: "./elasticsearch/connector",
        defaultTLP: "WHITE",
        branchesMaxDepth: 3,
        defaultMaxDocuments: 10
    },
    csafElasticsearch:  {
        elasticsearchIndex: "csaf_documents", requestTimeout: 30000,
        host: "localhost", port: "9200",
        user: "elastic",   password: "***",
        maxPreQuerySize: 10000
    },
    csafMysql:  {
        database: "csaf", requestTimeout: 30000,
        host: "localhost", port: "3306",
        user: "csaf",      password: "***",
        insecureAuth : true
    }
};
```

Listing 40: Configuration file of the CSAF API - config.js

not succeed in uploading his own CSAF documents, this attack vector should be closed.

Additional protection is provided by disabling introspection (see listing 39). If introspection is disabled, CSAF consumers and attackers no longer receive any information about the structure of the API. As long as the CSAF API remains in the standard, disabling introspection makes no sense because the structure and thus all functions and elements are made public. If the operator extends the CSAF API with its own interfaces and functions, this restriction could become more relevant because this information is not public. This can make attacks and the design of legitimate queries more difficult. However, CSAF providers must then find another way to enable their customers to design the queries. A qualification or training system could become necessary in this case. All GraphQL queries are forwarded to the Intermediate.

4.2 Intermediate

GraphQL queries are forwarded to the intermediate, which has a central role in the CSAF API. The intermediate caches the *GraphQL* context before it is manipulated, unfolds the shorthand `byProduct` into full queries and restricts each query by adding a TLP label constraint to each query. It reduces the query to the bare minimum by translating the *GraphQL* context into its own intermediate form, enriches information that reduces the programming effort in the connectors and harmonizes logical operators. And it searches on the file system in cases the CSAF documents are needed as originals. Since the intermediate layer is intended to unify and simplify (see table 8), unnecessary case distinctions are avoided through harmonization (see listing 41).

```
function harmonizeIntermediateObjects(intermediateObject, graphQlContext){
  if(intermediateObject.name == "must") {
    intermediateObject.name = "csafAnd";
  } else if(intermediateObject.name == "must_not"){
    intermediateObject.name = "csafNot";
  } else if(intermediateObject.name == "should") {
    intermediateObject.name = "csafOr";
  } else if(intermediateObject.name == "exist"){
    intermediateObject.name = "existp";
    let newExistpArguments = createExistpArguments(graphQlContext);
    let i = 0;
    while(i < newExistpArguments.length){
      if(!intermediateObject.arguments){intermediateObject.arguments = [];}
      intermediateObject.arguments[intermediateObject.arguments.length] =
        newExistpArguments[i];
      i++;
    }
    // remove  because "exist" tree branche is now "existp" attribute
    delete graphQlContext.selectionSet;
  }
  return intermediateObject;
}
```

Listing 41: Intermediate harmonization

Harmonization is not the only renaming in the intermediate. The renamed enumerations (see enumerations in subsection 4.1.1) must be translated back so that they can be used by connectors. The challenge in translation was to get the right object for translation without having to approach an infinite number of paths for the mappings (see `branches` in table 11). Because the last two path parts are sufficient for uniqueness, the path is shortened to the last two parts and used for the translation (see listing 42).

Surprisingly, during the implementation it turned out that lower case is also accepted by *GraphQL*, although it should not be (see *GraphQL* enumeration specification in subsection 3.3.1). This would have the advantage that then only one enumeration must be translated and that then also these values could be checked automatically during the validation.

```
/**
 * @param CSAF document path json_path
 * @param Enumeration in UPPER_CASE argument_value
 * @returns
 */
function translateEnum(json_path, argument_value){
  let translatedEnum = "";
  let tempJsonPath = "";
  var tempArray = json_path.split("/");
  let i = tempArray.length-2;
  while(i < tempArray.length){
      if(i!=0){ tempJsonPath += "/" + tempArray[i]; }
      i++;
  }
  if(tempJsonPath == "/document/csaf_version") {
    if      (argument_value == "V2_0")    {translatedEnum = "2.0";}
    else                                  {translatedEnum = "csafUndefined"}
  }else if(tempJsonPath == "/references/category") {
    // /document/references[]/category
    if      (argument_value == "EXTERNAL"){translatedEnum = "external";}
    else if (argument_value == "SELF")    {translatedEnum = "self";}
    else                                  {translatedEnum = "csafUndefined"}
  }else if(tempJsonPath == "/branches/category") {
    // /product_tree/branches[]*/category
    ...
  }else if(tempJsonPath == "/notes/category") {
    // /vulnerabilities[]/notes[]/category
    ...
  }else if(tempJsonPath == "/publisher/category") {
  ...
  }else {
    //no translation, use it as it is
    translatedEnum = argument_value;
  }

  return translatedEnum;
}
```

Listing 42: Intermediate enumeration translation

Another type of translation takes place in the products, here the shorthand is translated into the longhand. The shorthand notation `byProduct` is intended to relieve the CSAF consumer of typing work (see listing 8). Therefore, the element `byProduct` is replaced in the intermediate by the long notation (see listing 43), in which the context is extended as if the CSAF consumer had specified all places with product reference individually. The translation from *GraphQL* query to the intermediate version is started afterwards.

Here, a fixed maximum nesting depth is assumed. This must be determined beforehand by the administrator and entered in the configuration file (see `/csafIntermediate`

```
const config = require("../config");
function createIntermediateByProductObject(graphQlContext, parent){
  var tempgraphQlContext = JSON.parse(JSON.stringify(graphQlContext));
  tempgraphQlContext.name.value = "csafOr";
  var tempIntermediateObject =
    createIntermediateObject(tempgraphQlContext, parent);
  var tempSubElements = [];
  // /product_tree/branches[]
  // /product_tree/branches[]/branches[]
  // /product_tree/branches[]/branches[]/branches[]
  // /product_tree/branches[]/branches[]/branches[]/branches[]
  let i = 0;
  while(i < config.csafIntermediate.branchesMaxDepth){
    tempSubElements[tempSubElements.length] =
      createByProductProductTreeBranches_V2(
        tempIntermediateObject, i
      );
    i++;
  }
  // /product_tree/full_product_names[]
  tempSubElements[tempSubElements.length] =
    createByProductProductTreeFullProductNames(tempIntermediateObject);
  // /product_tree/relationships[]/full_product_name
  tempSubElements[tempSubElements.length] =
    createByProductProductTreeRelationshipsFullProductName(
      tempIntermediateObject);
  return tempSubElements;
}
```

Listing 43: Unfolding of the byProduct shorthand

`/branchesMaxDepth` in listing 40). A setting that is too high unnecessarily increases resource consumption. The administrator should therefore determine the exact nesting depth of branches in advance. Fortunately, the CSAF API already provides all the possibilities to retrieve this information (see listing 44).

```
query{
  n0: csafApi (max_documents: 1, metadata:true) { findDocuments { must {
      existp (path:"/product_tree") }}}
  n1: csafApi (max_documents: 1, metadata:true) { findDocuments { must {
    existp (path:"/product_tree/branches") }}}
  n2: csafApi (max_documents: 1, metadata:true) { findDocuments { must {
    existp (path:"/product_tree/branches/branches") }}}
  n3: csafApi (max_documents: 1, metadata:true) { findDocuments { must {
    existp (path:"/product_tree/branches/branches/branches") }}}
  n4: csafApi (max_documents: 1, metadata:true) { findDocuments { must {
    existp (path:"/product_tree/branches/branches/branches/branches") }}}
  n5: csafApi (max_documents: 1, metadata:true) { findDocuments { must {
    existp (
      path:"/product_tree/branches/branches/branches/branches/branches") }}}
}
```

Listing 44: Manual query of the maximum depth

The results of the development database give a maximum nesting depth of 3 (see nesting n3 has hits and n4 has no hits in listing 45), therefore `3` is entered in the configuration file as `branchesMaxDepth` (see listing 40).

```
{
  "data": {
    "n0": { "documents": [...], "metadata": { "total": 5076 } },
    "n1": { "documents": [...], "metadata": { "total": 14 } },
    "n2": { "documents": [...], "metadata": { "total": 14 } },
    "n3": { "documents": [...], "metadata": { "total": 14 } },
    "n4": { "documents": [],    "metadata": { "total": 0 } },
    "n5": { "documents": [],    "metadata": { "total": 0 } }
  }
}
```

Listing 45: Results of the try depth query from listing 44

The unfolding of the `byProduct` shorthand is a convenience for the user, as he does not have to query each CSAF document path containing product names individually. For the next convenience it is required to violate a *GraphQL* principle (see subsection 2.6). As mentioned before (see subsection 3.3.2), the *GraphQL* context must be manipulated to display additional elements that the user has not explicitly requested. If the user has not made any specifications regarding the output `/query/csafApi/documents`, it is assumed that all elements should be displayed (see subsection 3.3.2). In order for all elements to be displayed, the *GraphQL* context must be changed as if the user has requested all elements. All elements are added to the *GraphQL* context (see listing 46). The content of the prepared static file `csaf_graphql_all_possible_elements.js` is inserted.

Not only do elements need to be added to be filled. Some elements must be removed from the *GraphQL* context so that they do not appear empty. The `findDocuments` part of the

```
// create a manipulated query to display all the elements of the response,
// not just the requested one
context.fieldNodes[0].selectionSet.selections = [{
  kind: "Field", name: { kind: "Name", value: "documents" },
  selectionSet: require("./csaf_graphql_all_possible_elements")
}];
```

Listing 46: Inserting the static list of all elements - csaf_graphql_all_possible_elements.js

query would always return an empty element if it is not deleted from the *GraphQL* context. This type of manipulation introduces a significant challenge. The moment the context is no longer resent because the query has not changed, the query would fail because it cannot be extracted from the now manipulated context. Therefore, the *GraphQL* context must be cached before manipulation in order to access the original context if needed. To distinguish between the original and the manipulated context, the manipulated context is assigned an additional attribute `manipulated` (see listing 47). Another manipulation of the *GraphQL*

```
function cacheContext(context){
  var contextCsafApi = context.fieldNodes[0];
  // If an alias is assigned, then take the alias
  // otherwise take the name of the query
  var tempAlias = contextCsafApi.alias ?
    contextCsafApi.alias.value : contextCsafApi.name.value;
  // cache the original query if not manipulated
  if (!contextCsafApi.manipulated) {
    originalContext[tempAlias]=JSON.parse(JSON.stringify(contextCsafApi));
    ...
    // mark this context as CSAF API manipulated
    contextCsafApi["manipulated"] = true;
  } else {
    // never cache the manipulated query as original
  }
  return tempAlias;
}
```

Listing 47: Intermediate caching of the GraphQL context and marking it as manipulated

context concerns authorization (see listing 48) or authentication. Authentication is part of

```
"accounts": [
  { "id": "12345", "token": "***", "roles": ["csaf_consumer"],
    "permissions": [{"tlpLabel":"RED"}, {"tlpLabel":"GREEN"},
      {"tlpLabel":"WHITE"}] },
  { "id": "67890", "token": "***", "roles": ["csaf_aggregator"],
    "permissions": [{"tlpLabel":"WHITE"}] }
]
```

Listing 48: User credential examples stored in accounts.js

the intermediate (see subsection 3.3.3). The intermediate includes a restriction to predefined TLP labels as an additional condition for each query (see listing 49).

```
function authentification(context, userData){
  let temp = context.selectionSet.selections.filter(
    e => e.name.value === "findDocuments");
  if(temp.length == 0){
    context.selectionSet.selections[context.selectionSet.selections.length]
    = { "arguments":[], "directives":[], "kind": "Field", "name":{
          "kind": "Name", "value": "findDocuments" },
        "selectionSet":{ "kind": "SelectionSet",  "selections": []} };
  }
  // filter and patch all findDocuments-queries
  var findDocuments = context.selectionSet.selections.filter(
    e => e.name.value === "findDocuments");
  let tlpLabelSelections = generateTlpLabelSelections(userData);
  let i = 0;
  while ( i < findDocuments.length ){
    let temp = findDocuments[i];
    let selections = temp.selectionSet.selections;
    // original query: parameter1 AND parameter2 AND ...
    let csafOr1 = {
      "arguments":[], "directives":[], "kind": "Field", "name":{
        "kind": "Name", "value": "csafAnd" },
      "selectionSet":{ "kind": "SelectionSet",  "selections": selections}
    };
    // restrictions = restriction1 OR restriction2 OR ...
    let csafOr2 ={
      "arguments":[], "directives":[], "kind": "Field", "name":{
        "kind": "Name", "value": "csafOr" },
      "selectionSet":{ "kind":"SelectionSet","selections":tlpLabelSelections}
    };
    // (original query parameter) AND (restrictions)
    var unionSelections = [];
    if(selections.length == 0){ unionSelections = [csafOr2]; }
    else {                      unionSelections = [csafOr1, csafOr2]; }
    let csafAnd = {
      "arguments":[], "directives":[], "kind": "Field", "name":{
        "kind": "Name", "value": "csafAnd" },
      "selectionSet":{ "kind": "SelectionSet", "selections":unionSelections}
    };
    findDocuments[i].selectionSet.selections =[csafAnd];
    i++;
  }
  return context;
}
```

Listing 49: Extension of the query for the purpose of authentication

The script first filters out all queries. If the query does not exist, then an empty query is added. Additional constraints are appended to each query so that after running the script, each query has the following structure:

```
( (original query parameter) AND (tlp1 OR tlp2 OR ..) )
```

This is to prevent that more files can be queried than allowed. Thus, conditions, or more precisely, restrictions were added. Thereby `findDocuments` is converted to `csafAnd` (see subsection 3.4). However, entire CSAF documents can also be added.

To be able to attach the original documents (CSAF documents, signatures and hashes), two extensions have been implemented. First, files are searched on the file system that

```
if (intermediateObject.originals) {
  // object originals is present, user wishes to get the original files
  results["originalDocuments"] =[];
  let i = 0;
  while (i < results.documents.length) {
    let documentTrackingId = results.documents[i].document.tracking.id;
    // fetch files belonging to (results: /documents[]/document/tracking/id)
    let fileList = searchOnFilesytem(documentTrackingId, csafDocumentsPath);
    let j = 0;
    while (j < fileList.length){
      results.originalDocuments[results.originalDocuments.length]
        = fileList[j];
      j++;
    }
    i++;
  }
}
```

Listing 50: Adding original files to the results - originals

match the attribute `/document/tracking/id`. The found documents are appended `base64` encoded to the result set (see listing 50). And secondly, the *GraphQL* context is extended

```
let originals = intermediateObject.arguments.filter(e=>e.name=="originals");
if(originals.length > 0 && originals[0].value){
  context.fieldNodes[0].selectionSet.selections[
    context.fieldNodes[0].selectionSet.selections.length
  ]={
    kind: "Field", name: {kind: "Name", value: "originalDocuments"},
    selectionSet: {kind: "SelectionSet", selections: [
      {kind: "Field", name: {kind: "Name", value: "documentName"}},
      {kind: "Field", name: {kind: "Name", value: "documentBase64Binary"}}
    ]}
  };
}
```

Listing 51: Adding originalDocuments to GraphQL context

to additionally display these attachments (see listing 51), i.e. the original CSAF documents also arrive at the clients response (see example in listing 52).

```
{
  "data": {
    "example": {
      "documents": [ ... ],
      "originalDocuments":[
        { "documentName": "wid-sec-w-2022-0517.json",
          "documentBase64Binary": "ewogICJk..." },
        { "documentName": "wid-sec-w-2022-0517.json.asc",
          "documentBase64Binary": "LS0tLS1C..." },
        { "documentName": "wid-sec-w-2022-0517.json.sha256",
          "documentBase64Binary": "NmM2ODgw..." },
        { "documentName": "wid-sec-w-2022-0517.json.sha512",
          "documentBase64Binary": "YThiY2Uz..." }
      ]
}}}
```

Listing 52: GraphQL example response for originalDocuments

Once all changes are made to the *GraphQL* context and the intermediate version of the *GraphQL* context is created, then the intermediate version of the query can be passed to the database connector.

4.3 Eleasticsearch

Elasticsearch was chosen as the database (see subsection 3.1). Therefore, a connector for *Elasticsearch* was implemented, which is called by the intermediate (see listing 53) and defined in the configuration file (see listing 40).

```
const config = require("../config");
const connector = require(config.csafIntermediate.connector);

function Intermediate(context, args){
  ...
  console.log("use connector (" + connector.name  + ")");
  return connector.connect(intermediateObject)
    .then( r => { ...; return r; } )
    .catch( e => { ... })
  ;
}
```

Listing 53: The intermediate passes the intermediate version of the query to the connector.

The connector takes the intermediate version of the query, converts it to an *Elasticsearch* query, and executes it. The return structure of *Elasticsearch* differs from the structure that Intermediate and also *GraphQL* expect, so the expected structure is created before the results are returned. At this point, the total number of CSAF documents (see metadata in listing 22) corresponding to the query is appended (see listing 54).

```
function Connector(intermediateObject){
  var elasticSearchQuery =
    intermediateToElasticsearchQuery(intermediateObject, undefined);
  var needPreQuery = testNeedPreQuery(intermediateObject);
  if(needPreQuery){
    // run special query (with pre-query)
    ...
  } else {
    // run normal query
    return ElasticSearchClient(elasticSearchQuery)
    .then(
      //change response comming from elasticsearch
      r => {
        let results = r['hits']['hits'];
        results.map((item, i) => results[i] = item._source);
        results = {
                    "documents": results,
                    "metadata": { "total": r.hits.total.value }
                  };
        return results;
      }
  )
  .catch( e => {...} );
}
```

Listing 54: Elasticsearch connector returns result to the intermediate

In order to translate the complete intermediate query into an *Elasticsearch* query, each object (e.g. `csafAnd`) of the intermediate query must be translated individually. A `csafAnd` corresponds to a `must` in *Elasticsearch*. Therefore, the *GraphQL* `csafAnd` query (see listing 55)

```
query {
  csafApi {
    findDocuments {
      csafAnd {
        condition1: document {title (should: "Linux")}
        condition2: document {title (should: "Kernel")}
}}}}
```

Listing 55: The logical AND operator in the CSAF API

is translated to an *Elasticsearch* must query (see listing 56).

```
"query": {
  "bool":{ "must":[
      {"match": {"document.title": "Linux"  }},
      {"match": {"document.title": "Kernel" }}
]}}
```

Listing 56: Elasticsearch representation of figure 55

The `should` element is for the logical `or` operator, as shown in listing 57, which is a

```
query { csafApi {
  findDocuments {
    csafOr {
      condition1: document {title (should: "Linux")}
      condition2: document {title (should: "Kernel")}
}}}}
```

Listing 57: The logical OR operator in the CSAF API

`csafOr` in the CSAF context. Therefore, it is converted into a should query (see listing 58).

```
"query": {
  "bool":{ "should":[
      {"match": {"document.title": "Linux"  }},
      {"match": {"document.title": "Kernel" }}
]}}
```

Listing 58: Elasticsearch representation of listing 57

A `csafNot` (see listing 59) corresponds to the `must_not` in the *Elasticsearch* context (see

```
query { csafApi { findDocuments {
    csafNot {
      condition1: document {title (should: "Linux")}
      condition2: document {title (should: "Kernel")}
}}}}
```

Listing 59: The logical NOT operator in the CSAF API

listing 60).

```
"query": {
  "bool":{ "must_not":[
      {"match": {"document.title": "Linux"  }},
      {"match": {"document.title": "Kernel" }}
]}}
```

Listing 60: Elasticsearch representation of figure 59

Since `csafXor` is intended to avoid the user typing work (see the short form in listing 61), the long form must be implemented in the *Elasticsearch* query (see listing 63).

```
query { csafApi { findDocuments {
    csafXor {
      condition1: document {title (should: "Linux")}
      condition2: document {title (should: "Kernel")}
}}}}
```

Listing 61: The logical XOR operator in the CSAF API

For better understanding, the first thing to do is to convert the `csafXor` to a logical equivalent (see listing 62).

```
query { csafApi { findDocuments {
  csafOr {
    csafAnd {
      condition1:           document {title (should: "Linux") }
      condition2: csafNot { document {title (should: "Kernel")} }
    }
    csafAnd {
      condition1: csafNot { document {title (should: "Linux") } }
      condition2:           document {title (should: "Kernel")}
    }
  }
}}}
```

Listing 62: The logical XOR equivalent

Now each logical operator is translated separately (see listing 63).

```
"query": {
  "bool": {"should": [
    {"bool": {"must": [
      {"bool": {"must":     [ {"match": {"document.title": "Linux"  }} ]}},
      {"bool": {"must_not": [ {"match": {"document.title": "Kernel" }} ]}}
    ]}},
    {"bool": {"must": [
      {"bool": {"must_not": [ {"match": {"document.title": "Linux"  }} ]}},
      {"bool": {"must":     [ {"match": {"document.title": "Kernel" }} ]}}
    ]}}
  ]}
}
```

Listing 63: Elasticsearch representation of figure 62

For the sake of completeness, the must is inserted parallel to the must_not, even though it is completely unnecessary from a logical point of view. Not only the Xor can be represented in different ways, also the existence of CSAF document elements can be queried in different ways. Both alternatives `alternative1` and `alternative2` that query the existence of a CSAF document elements (see listing 64) are mapped the same in the *Elasticsearch*

```
query { csafApi {
  findDocuments {
    must {
      alternative1: exist{
        document {
          title
          lang
        }
      }
      alternative2: existp (path: ["/document/title", "/document/lang"])
}}}}
```

Listing 64: The exist or existp function in the CSAF API

query (see listing 65). During the implementation it turned out that the effort for `exist` is significantly lower, if `exist` is already harmonized in the intermediate away (see `exist` in table 8). Therefore, this case no longer occurs in the connectors and can be ignored.

```
"query": {
  "bool":{ "must":[
    {"exists": {"field": "document.title"}},
    {"exists": {"field": "document.lang" }}
]}}
```

Listing 65: Elasticsearch representation of listing 64

Once the *GraphQL* query is converted to an *Elesticsearch* query, the *Elesticsearch* connector can query the *Elesticsearch* database. To avoid traffic between CSAF API and the database, each *Elesticsearch* query can be given a size. The number of hits can be limited by a user with the argument `max_documents` (see listing 19). *Elasticsearch* has an absolute limit of 10,000 `Results Per Query`, so a user can enter a larger number but will not get more results delivered (see *Limits* by elasticsearch B.V. [5]). If the `max_documents` attribute is set and it is set by default in the intermediate for performance reasons (see `defaultMaxDocuments` in listing 40), this value is appended to the *Elasticsearch* query to avoid unnecessary network traffic (see listing 66). Since the CSAF API is designed to be modular, additional connectors

```
let max_documents = getMaxDocuments(intermediateObject);
let elasticsearchQuery = {
  "size": max_documents,
  "query": {...}
};
```

Listing 66: Limiting Elasticsearch results to max_documents

(such as *MySQL*) can be implemented.

4.4 MySQL

The *MySQL* connector has not been implemented. Only a basic scaffold was built to show how a *MySQL* database is connected, which parameters are passed and how the connector switching works (see description in listing 40). To test the basic functionality, the same CSAF document is always delivered, no matter what the query is (see `fake_file` in listing 67).

```
function Connector(context, args, userData, intermediateObject){
  var sql = convertSearchParameterToSQLQuery(context.selectionSet.selections);
  // run query
  return mysqlClient(sql)
    .then(
      //change response comming from mysql
      r => {
        let _source = r.result;
        let _source_json = mysql_to_json(r.result);
        //fake_file
        _source_json = fake_file();
        _source = { "documents": [_source_json] };
        return _source;
      }
    )
    .catch(e => {...})
  ;
}
```

Listing 67: MySQL connector implementation

4.5 REST-API queries

Schmidt has defined 7 routes (see table 6). All of these routes must be implemented (see subsection 3.3). When using the CSAF API these routes do not need to be implemented separately. They can be queried using the functionality now available. The route `/csaf-documents/by-cve/{cve}` can be directly converted into a *GraphQL* query (see example query in listing 68). Also the optional parameters `CVSSv2` and `CVSSv3`, which Schmidt mentions in his thesis, can be given directly.

```
query { csafApi (max_documents: 2) {
  findDocuments { must {
    vulnerabilities {cve (exact: "CVE-2022-27193")}
    should {
     vulnerabilities {scores {
        cvss_v2 {baseScore (gte: 2.5, lte: 10)}
        cvss_v3 {baseScore (gte: 2.5, lte: 10)}
}}}}}}
```

Listing 68: REST-API ../by-cve/cve query as GraphQL query

As mentioned in subsection 2.7.1, the `devices` from Schmidt's thesis are called products here, because this term reflects the CSAF terminology. Products can be queried in many ways, the easiest way is to use the `byProduct` function (see listing 69).

```
query { csafApi (max_documents: 2) {
  findDocuments { must {
  pr1: byProduct {name (should: "Red Hat")}
  pr2: byProduct {name (should: "SUSE")}
}}}}
```

Listing 69: REST-API ../from-device-list query as GraphQL query

To match a property is the simplest query, all other queries build on it. The property is specified as a *GraphQL* query. The condition is set on the property itself. In this example (see listing 70), a product is searched for that contains *Red Hat* in the product name.

```
query { csafApi (max_documents: 2) { findDocuments {
  must {
    property: product_tree {full_product_names {name (should: "Red Hat")}}
}}}}
```

Listing 70: REST-API ../match-property query as GraphQL query

There are many ways to specify multiple properties, the simplest way is to number them (see properties p1-p3 in listing 71). The properties do not have to be numbered in this case,

```
query { csafApi (max_documents: 2) {
  findDocuments { must {
    p1: document {aggregate_severity {text (exact: "mittel")}}
    p2: product_tree {full_product_names {name (should: "SUSE")}}
    p3: vulnerabilities {cve (exact: "CVE-2022-47951")}
}}}}
```

Listing 71: REST-API ../match-properties query as GraphQL query

because they are different properties. If the properties are the same, e.g. `OR` concatenated, then the properties must be numbered consecutively. More precisely, they do not need to be numbered, but an alias must be assigned so that *GraphQL* can differentiate them. As

```
query { csafApi (max_documents: 2) {
  findDocuments { must {
    document {
      p1: publisher {namespace (exact: "https://www.bsi.bund.de")}
      p2: tracking {id (exact: "BSI-2022-0002")}
}}}}}
```

Listing 72: REST-API ../by-id/publisher_ns/tracking_id query as GraphQL query

already seen with match-properties, two or more properties can be queried, i.e. also the publisher namespace and the tracking ID (see listing 72). As shown for `match-property` (see

```
query { csafApi (max_documents: 2) {
  findDocuments { must {
    document {title (should: "Denial of Service")}}
}}}
```

Listing 73: REST-API (by-title) query as GraphQL query

listing 70),it is possible to query for any property, including the document title (see listing 73) or the document publisher name (see listing 74).

```
query { csafApi (max_documents: 2) {
    findDocuments  must {
      document {publisher {name (should: "Bundesamt")}}}
}}
```

Listing 74: REST-API (by-publisher) query as GraphQL query

In order for the CSAF API queries to search the database, the database must be populated with CSAF documents. The CSAF documents must be migrated into the database.

4.6 Migration respectively synchronization

At the time of migration, the CSAF documents are on the file system (see figure 4). They must be migrated into the corresponding database and kept up to date. To keep the status up to date, a synchronization must take place regularly. When a new CSAF document is created on the file system, this CSAF document must also be uploaded to the API database. When a CSAF document is deleted from the file system, it must also be deleted from the API database. This procedure saves server resources (computing load), since only absolutely necessary steps are performed.

Alternatively, the complete data-set can be deleted and rebuilt. This procedure seems to make sense especially for smaller databases, as it is reliable and clearly arranged. Database entries that no longer exist were completely deleted and only current CSAF documents were imported.

Because *Elasticsearch* was chosen as the database (see subsection 3.1), the creation of the *Elasticsearch* index and the import were implemented. In order for the CSAF documents to be uploaded to the *Elasticsearch* database, the first thing to do is create an index (see listing 75).

```
curl -k -u USERNAME:PASSWORD -X PUT "localhost:9200/csaf-documents?pretty"
```

Listing 75: Create Elasticsearch index for CSAF documents

Now the CSAF documents can be uploaded under the `csaf-documents` index by running the import script (see `elastic_upload.sh` in listing 76). The unique file name is used as ID

```
/bin/bash ~/elastic_upload.sh ~/UPLOADDIRECTORYPATH USERNAME PASSWORD
```

Listing 76: Start upload on shell with Elasticsearch user credentials

here (see listing 77). File names are unique in the context of a CSAF publisher. Because it cannot be avoided that file names are not unique in the case of a CSAF aggregator, the path must also be added as a distinguishing feature. The script runs through all directories and uploads all documents with the `.json` file extension.

```
#!/bin/bash
MYHOMEDIR=~
DIR=$1
USERNAME=$2
PASSWORD=$3
for FILE in $1/*; do
  # if is a folder, that look inside the folder
  if [ -d $FILE ]; then
    sh ./elastic_upload.sh $FILE;
  else
    # upload only in cases of .json files
    case $FILE in
      *.json )
        # use directory as ID in Elasticsearch,
        # without information about the homedirectory
        MYKEY=${FILE#"$MYHOMEDIR"*};
        MYKEY=$(echo "$MYKEY" | tr "/" "_" | tr "-" "_")
        # upload file to elasticsearch
        curl -k -u $USERNAME:$PASSWORD -X POST \
        -H 'Content-Type:application/json' \
        --data @"$FILE" "https://localhost:9200/csaf_document/_doc/{$MYKEY}";
    esac
  fi
done
```

Listing 77: CSAF documents upload shell-script (elastic_upload.sh)

Migration to *MySQL* was not implemented, because the *MySQL* Connector was only implemented schematically to demonstrate that the CSAF API can be connected to a *MySQL* database. The migration should be similar to *Elasticsearch*. All CSAF documents are iterated through and stored in the *MySQL* database. Depending on the database design, there may be one or many database tables that need to be filled.

5 Results

The main goal of this master thesis was to design and implement an API with *GraphQL* to demonstrate that this is possible, to show where the limits of *GraphQL* and the API are and what opportunities arise with the API. Most importantly, the API is incredibly intuitive to use. If the query is to search for CSAF documents that should have *Linux* in the document title, the query looks exactly like this (see listing 78).

```
query {
  csafApi {
    findDocuments {
      document {
        title (should: "Linux")
}}}}
```

Listing 78: Intuitive query

In this way, any element of interest can be searched for. There are so many elements that

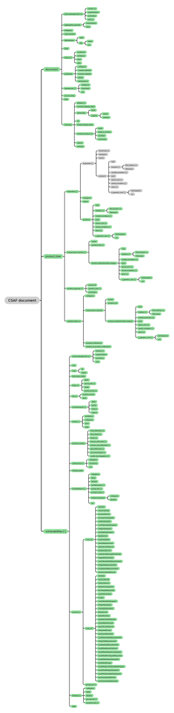


Figure 17: CSAF API queryable but no longer displayable elements

it cannot be properly displayed on one page (see figure 17).

Depending on the element type, type-specific search parameters can be used. In case of a date, CSAF documents can be searched for that are younger than the given date (see listing 79). Even complex queries are made possible by the logical operators (e.g. AND, OR, XOR and NOT).

```
query {
  csafApi {
    findDocuments {
      document {
        tracking {
          current_release_date (younger: "now-10d")
}}}}}
```

Listing 79: Query CSAF documents for the last 10 days.

With the possibility to search for existing or non-existing elements, not only tools for complex queries are provided (see listing 15), but also possibilities for quality improvement are created. Thus, for example, missing current release dates `/document/tracking/current_release_date` can be searched for.

See figure 17, there are to many elements with too much information to reasonably display it on one page. For an even smaller display (like mobile devices) it makes sense not to display all elements at once. Therefore, the response can be limited to certain elements. Only a selection of elements is returned, making the information displayable and readable even on small displays.

If an array element is defined as a filter criterion, the query returns the complete CSAF document on which the query matches. If the structure of the response is specified, then all JSON elements that were not queried are removed. If an array element is part of the response structure, then all array elements are returned because they are part of the searched CSAF document array. This filter criterion can also be used to remove these elements of the CSAF document array that are irrelevant for the CSAF consumer.

It is important to know whether a product is affected by a security vulnerability. It is tedious to search out product IDs in each CSAF document based on the product name and then use the product ID to determine whether a vulnerability is affected. Therefore, a kind of preliminary query was implemented that first queries all product names and then filters out the CSAF documents whose product IDs do not appear in the queried vulnerability product status (e.g. `/vulnerabilities[]/product_status/known_affected[]`). Only then the exact affected status can be determined.

For automation, characteristics are needed by which a change can be determined. Time stamps (like dates) are particularly well suited for this purpose. A complete list of CSAF document dates and their definitions can be found in the appendix (see table 14). Probably the most relevant date for automation is the document current release date `/document/tracking/current_release_date`. It shows when the document was last modified. Therefore, it is very suitable for queries about the latest CSAF documents. With the possibility to specify the date relatively, queries can be defined that always provide current CSAF documents (see listing 79).

6 Discussion

The fact that an API can be implemented with *GraphQL*, *Elasticsearch* as database and corresponding effort in the type description became apparent early in the design. Accordingly, the problem with the *GraphQL* principles appeared early, which could not be completely resolved until the end, without violating the *GraphQL* principles. Recommended extensions to the CSAF standard have also emerged and will be addressed here.

Eleasticsearch, which was chosen as database, had advantages. Among the notable disadvantages is the versioning. Although *Elasticsearch* stores the number of times a document was overwritten, it does not store the content of the overwritten document. Even the best API cannot compensate for versioning that is already missing in the database. Therefore, this chapter will conclude with the database recommendation.

6.1 Extension of the CSAF standard

In the course of the development of this document, the need for changes became apparent, as standards have changed in the meantime or the full potential of the CSAF API could not be leveraged. These change requirements are addressed in the following subsections.

6.1.1 TLP versioning

FIRST.ORG, Inc's TLP transitioned to version 2.0 in September 2022. The original TLP version cannot be read from the CSAF document itself because the version number is missing. There are only two properties for TLP the Uniform Resource Locator (URL) and the label (see listing 80).

```
"tlp": {
  "additionalProperties": false, "title": "Traffic Light Protocol (TLP)",
  "description": "Provides details about the TLP classification of the
    document.",
  "properties": {
    "label": {
      "description": "Provides the TLP label of the document.",
      "enum": [ "AMBER", "GREEN", "RED", "WHITE" ],
      "title": "Label of TLP", "type": "string" },
    "url": {
      "default": "https://www.first.org/tlp/",
      "description": "Provides a URL where to find the textual description
        of the TLP version which is used in this document. Default is the
        URL to the definition by FIRST.", "examples": [...],
      "format": "uri", "title": "URL of TLP version", "type": "string" }
  },
  "required": [ "label" ],  "type": "object"
}
```

Listing 80: CSAF schema TLP definition by OASIS Open

In this version change the label `TLP:WHITE` was replaced by `TLP:CLEAR`. Since the link refers always to the current version, the CSAF publisher is forced to regenerate all CSAF `TLP:WHITE` documents to change either `/document/distribution/tlp/url` or `/document/distribution/tlp/label` when such TLP version change occurs. In this case, the easiest way is to change the URL to the old version without having to have detailed knowledge about the version change itself (for the link to the old version see table 12).

Version number	**Link**	**Version**
2.0	https://www.first.org/tlp/	latest or current version
1.0	https://www.first.org/tlp/v1/	old version

Table 12: Different CSAF enumeration cases

The version number could be added to the TLP schema definition to avoid misunderstandings and need for action.

```
"tlp": {
  ...
  "properties": {
    "label": {...},
    "url":   {...},
    "version": {
      "$ref": "#/$defs/version_t"
    }
  },
  "required": [ "label" ],
  "title": "Traffic Light Protocol (TLP)",
  "type": "object"
}
```

Listing 81: Extended CSAF TLP definition

6.1.2 Enumeration

Enumerations are accepted in the *GraphQL* type definition in upper case (see subsection 3.3.1). Enumerations like CSAF version (see table 10) are not accepted under this condition and must be renamed to the type definition (see listing 33). Enumerations with a dot in the enumeration value or those that start with a number must be renamed necessarily, an adaptation of the CSAF standard could avoid unavoidable translations in the API.

For the CSAF API, all enumerations except TLP label have been renamed. This renaming costs resources in processing. Although the capitalization of 12 enumerations was not necessary, they were also renamed for completeness and policy compliance. Should the CSAF standard be adapted and renaming become completely obsolete, then it would have a positive impact on the performance of the API. A renaming in the CSAF standard is to be welcomed from the point of view of the API.

6.2 Violation of GraphQL principles

In order to meet all or as many requirements as possible, tricks had to be done, since some requirements violate the *GraphQL* principle (see subsection 2.6). The tricks applied contain additional risks. On the one hand, it is possible that this manipulation makes *GraphQL* itself more vulnerable. And on the other hand it cannot be guaranteed that the CSAF API still works after a *GraphQL* upgrade.

During manipulation, the context of *GraphQL* is changed. Entries are deleted and added as needed. Should *GraphQL* consciously block or ignore this kind of manipulation in a coming version, the following points must be refactored or accepted by CSAF API operators and consumers.

`Metadata` and `originalDocuments` are added to the *GraphQL* context as needed when a user sets the CSAF API attributes (`metadata` and `originals`) to true. If adding them to the *GraphQL* context is no longer possible, a API user must include them in the query for them to be displayed. Especially if the user should see all elements of a document, all elements of a CSAF document must be included in the query by the user. See figure 17, there are many of them.

The CSAF API query contains elements (`findDocuments` and `filterArrays`) that would appear empty in the output if not removed by the *GraphQL* context manipulation. This can cause confusion for CSAF consumers who expect a fixed number of elements. Downstream applications can fail if they stubbornly process the first element and suddenly an empty element appears first.

GraphQL can become vulnerable. The cheatsheet by OWASP Foundation shows that a hardening of *GraphQL* is advised [24]. With the manipulation of the *GraphQL* context comes a potential vulnerability with yet unknown extent. Since there are no empirical values with the new API yet, there is at least the possibility that some bugs or dependencies may still lead to crashes, data disclosure or denial of service. Even if no serious vulnerabilities exist now, new vulnerabilities may be added, e.g. when new connectors are implemented.

OWASP API Security Project API Top 10 [26], the *GraphQL Cheat Sheet* [24] and *GraphQL* own security recommendations were taken into account as far as possible. Thus, a reasonable nesting depth was determined (see table 11) and set up (see listing 39). Neither the *GraphQL Cheat Sheet* nor *GraphQL*'s own security recommendations assumed that such manipulations of the *GraphQL* context would take place, as they violate *GraphQL* principles, so some surprises might still occur.

6.3 CSAF document versioning

If a CSAF consumer receives a new CSAF document for review, then the complete document is reviewed (affectedness, impact, criticality). If this CSAF document is updated, then the check must take place again. Therefore, there might be a wish for versioning, more specifically change tracking. A CSAF consumer can read the change history from the element `/document/tracking/revision_history[]`. These entries are made for humans and therefore understandable for humans. For an automation these entries are not suitable, because they have no fixed reference to other elements in the document, no IDs which are taken over into other elements. In the end, the question remains open whether relevant changes have occurred.

If the relevance is determined by the product, then a change by an additional affected product is not automatically a relevant change. Relevant changes can include an additional product if it is in the own product list. Relevant changes may include a status change `/vulnerabilities[]/product_status` if it relates to its own product in use. But all these considerations are worth nothing if the actual change cannot be clearly determined in an automated way.

With the functionality of a pre-query (see subsection 3.4.6) it was shown that reliefs for the CSAF consumer can be implemented. These reliefs are possible at the expense of the CSAF provider. Many relevant changes are already covered by the design of the CSAF API query. If the vulnerability status for a product changes, CSAF documents are filtered out that were previously found or added if they were not previously included. Useless effort is created for the CSAF consumer if the changes to the CSAF document are not part of the query, for example, if additional affected products are added to the CSAF document that the CSAF consumer does not have in use. In this case, the CSAF consumer has to look through the CSAF document again without having had any added value to his own work. The CSAF consumer has to make this effort because the last change date has changed. The CSAF consumer cannot query a relevant change concerning a product. A selection feature seems to be missing for these changes to the product. A last change timestamp on the product would allow CSAF consumer to search more specifically for relevant updates without having to refer to the parent timestamp of the entire CSAF document.

By extending the revision history (see listing 82), it becomes possible to extend the CSAF API to allow specific querying using the pre-query functionality. With the pre-query,

```
"revision_history": {
  "description": "Holds one revision item ... including the initial one.",
  "items": {
    "additionalProperties": false,
    "description": "Contains all the information ... of a CSAF document.",
    "properties": {
      "date": {...},
      "legacy_version": {...},
      "number": {...}, "summary": {...},
      "affected_products": { "$ref": "#/$defs/products_t" }
    },
    "required": [ "date", "number", "summary" ], "title": "Revision",
    "type": "object"
  },
  "minItems": 1, "title": "Revision history", "type": "array"
}
```

Listing 82: Affected products extension of the document revision history

the product IDs are queried, as with the other pre-query. With these product IDs, CSAF documents are queried that have these product IDs in the revision history, which has a current timestamp. Unlike current queries, it is then no longer necessary to filter by the timestamp of the whole CSAF document.

6.4 Integration in other application

It could bear untold fruits, if software manufacturer integrates a connector for the CSAF API into their own product, perhaps in their product administration or monitoring panel. Security advisories would pop up there, where IT administrators regularly check system status. This is where the strengths of *GraphQL* come into play. The query can be set to retrieve only relevant parts of a CSAF document, which are perfectly suited to be integrated into existing applications.

6.5 What database should be chosen?

As already mentioned (see subsection 3.1), *Elasticsearch* was chosen as the database for this work. When querying the product status of a vulnerability using the product name, the *Elasticsearch* query quickly reaches its limits and a pre-query was needed. It was possible to use pre-queries to answer this type of query with *Elasticsearch*. However, the queries quickly become inefficient. This problem does not seem to arise with relational databases, since the queries using SQL provide for such relationships. How the increasing document volume will affect the performance of relational databases cannot be deduced from the rudimentary implementation of the MySQL connector.

If the CSAF document was found, then in the case of *Elasticsearch* it only needs to be returned. In the case of a relational database, the CSAF document is distributed over many tables and must first be exported in JSON format. Alternatively, only essential CSAF document elements could be transferred to database tables, which are needed for the most frequent queries. The query in the relational database can then no longer generate the complete CSAF document. The database must therefore hold references to the original CSAF documents and deliver them to the API. This prevents the need to migrate every CSAF document component to the relational database. However, the number of file accesses would then increase and this would affect the API performance. With *Elasticsearch*, on the other hand, file accesses only occur when the API user requests the original files.

The query itself seems to be much more efficient on a relational database. Especially if the relational database already exists and the CSAF documents are already generated from it. *Elasticsearch* is very simple and quickly operational from the initial setup. Both databases mentioned here have their advantages and disadvantages and thus a person responsible is spoilt for choice.

7 Conclusion

All of the work presented in this paper was conducted to find an answer to this research question:

> *How could a GraphQL-based API make querying CSAF documents more efficient?*
>
> Research question

This master thesis structures and names individual components of the API and their functional scope. After designing and implementing the *GraphQL*-based API, all CSAF document contents became searchable. This made it possible to search for CSAF documents with more relevant content, such as vulnerabilities with a specific CVE or minimum Common Vulnerability Scoring System (CVSS) score.

During the development of the targeted API, limitations of *GraphQL* were reached and exceeded. A basic principle of *GraphQL*, that *GraphQL* queries always return predictable results, was violated. Elements were inserted into the query as needed, while other elements were deleted. Inserting elements as needed saves the API user a lot of typing.

After the first breakthrough was achieved, the API can be queried and provides data, the requirements were gradually implemented. The work took place iteratively with alternating design and implementation phases. With the API developed here, the query can be restricted directly to more relevant CSAF documents. This eliminates a significant part of the daily work of sifting through irrelevant documents to determine their irrelevance.

The API could leverage more potential if document changes could be assigned to products, as consumers of CSAF documents determine their personal affectedness based on the affected products. However, *GraphQL* is suitable as a basis for the CSAF API, even if *GraphQL* principles had to be violated in some places. The deviation from these principles raises questions. Has the API become more vulnerable as a result of the deviation?

Future research could address the structure of a CSAF security advisory. The document structure makes sense from the manufacturer's point of view, as it allows them to record how products are affected by assigning them to a vulnerability product status. The consumers of the CSAF documents have a different view, they want to get the question answered, from which vulnerabilities is the product in use affected. It seems to be the same data, just from different perspectives. To enable more effective queries for consumers, their perspective must be taken.

Bibliography

[1] Amazon Web Services, Inc. *OpenSearch.* 2023. URL: `https://aws.amazon.com/what-is/opensearch/` (visited on 02/24/2023).

[2] Apollo Graph, Inc. *9 Ways To Secure your GraphQL API — GraphQL Security Checklist.* 2021. URL: `https://www.apollographql.com/blog/graphql/security/9-ways-to-secure-your-graphql-api-security-checklist/` (visited on 04/18/2022).

[3] Arista Networks, Inc. *Security Advisories.* 2021. URL: `https://www.arista.com/en/support/advisories-notices/security-advisory/11999-security-advisory-59` (visited on 09/15/2023).

[4] Cisco Systems, Inc. *Cisco Security Advisories.* 2018. URL: `https://sec.cloudapps.cisco.com/security/center/publicationListing.x` (visited on 03/12/2023).

[5] elasticsearch B.V. *Limits.* 2023. URL: `https://www.elastic.co/guide/en/app-search/current/limits.html` (visited on 06/26/2023).

[6] elasticsearch B.V. *Welcome to Elastic Docs.* 2023. URL: `https://www.elastic.co/guide/index.html` (visited on 01/04/2023).

[7] Federal Office for Information Security (BSI). *CSAF documents.* 2023. URL: `https://wid.cert-bund.de/.well-known/csaf` (visited on 07/07/2023).

[8] Federal Office for Information Security (BSI). *Merkblatt zum sicheren Informationsaustausch mit dem Traffic Light Protocol (TLP), Version 2.0.* 2022. URL: `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/TLP/merkblatt-tlp.pdf` (visited on 12/23/2022).

[9] Federal Office for Information Security (BSI). *Stack Buffer Overflow vulnerability in FastStone Image Viewer 7.5 and earlier.* 2022. URL: `https://wid.cert-bund.de/.well-known/csaf/white/2022/bsi-2022-0002.json` (visited on 08/10/2023).

[10] Festo SE & Co. KG. *Downloads for FestoSecurityAdvisory.* 2021. URL: `https://www.festo.com/de/en/search/?tab=SUPPORT_PORTAL&q=FestoSecurityAdvisory&documentTypeGroup=EXPERT_KNOWLEDGE&documentTypes=&supportPortalQ=` (visited on 09/15/2023).

[11] FIRST.ORG, Inc. *TRAFFIC LIGHT PROTOCOL (TLP), FIRST Standards Definitions and Usage Guidance — Version 1.0.* 2022. URL: `https://www.first.org/tlp/v1/` (visited on 03/28/2023).

[12] FIRST.ORG, Inc. *TRAFFIC LIGHT PROTOCOL (TLP), FIRST Standards Definitions and Usage Guidance — Version 2.0.* 2022. URL: `https://www.first.org/tlp/` (visited on 03/28/2023).

[13] GraphQL Foundation. *A query language for your API.* 2022. URL: `https://graphql.org/` (visited on 05/16/2023).

[14] GraphQL Foundation. *Specification for GraphQL.* 2018. URL: `http://spec.graphql.org/June2018/%5C#sec-Enum-Value` (visited on 07/18/2023).

[15] Hitachi Energy Ltd. *Hitachi Energy.* 2022. URL: https://www.hitachienergy.com/products-and-solutions/cybersecurity/alerts-and-notifications (visited on 02/28/2023).

[16] J. Callas, L. Donnerhacke, H. Finney, D. Shaw and R. Thayer. *OpenPGP Message Format.* 2007. URL: https://datatracker.ietf.org/doc/html/rfc4880 (visited on 09/19/2023).

[17] MITRE Corporation. *Common Enumeration of Vulnerabilities Program.* 1999. URL: https://www.cve.org/ (visited on 04/16/2022).

[18] Nozomi Networks Inc. *Nozomi Networks Security Advisories.* 2019. URL: https://security.nozominetworks.com/csaf/ (visited on 09/15/2023).

[19] OASIS Common Security Advisory Framework (CSAF) Technical Committee. *Consider a set of proven values for maxima of string and array lengths.* 2021. URL: https://github.com/oasis-tcs/csaf/issues/204%5C#issuecomment-815975781 (visited on 08/31/2023).

[20] OASIS Open. *Branches Type.* 2022. URL: https://docs.oasis-open.org/csaf/csaf/v2.0/os/csaf-v2.0-os.html#312-branches-type (visited on 09/19/2023).

[21] OASIS Open. *Common Security Advisory Framework Version 2.0.* 2022. URL: https://docs.oasis-open.org/csaf/csaf/v2.0/os/csaf-v2.0-os.html (visited on 09/17/2023).

[22] OASIS Open. *CSAF json schema definition, Version 2.0.* 2022. URL: https://github.com/csaf-poc/csaf_distribution/blob/main/csaf/schema/csaf_json_schema.json (visited on 04/08/2022).

[23] Oracle Corporation. *Critical Patch Updates, Security Alerts and Bulletins.* 2022. URL: https://www.oracle.com/security-alerts/ (visited on 03/12/2023).

[24] OWASP Foundation. *GraphQL Cheat Sheet.* 2020. URL: https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html (visited on 02/23/2023).

[25] OWASP Foundation. *OWASP API Security Project.* 2019. URL: https://owasp.org/API-Security/editions/2019/ (visited on 08/31/2023).

[26] OWASP Foundation. *OWASP API Security Project.* 2019. URL: https://owasp.org/www-project-api-security/ (visited on 12/03/2022).

[27] OWASP Foundation. *OWASP API Security Project.* 2023. URL: https://owasp.org/API-Security/editions/2023/en/0x00-header/ (visited on 08/14/2023).

[28] OWASP Foundation. *SessionManagement.* 2019. URL: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html (visited on 08/27/2023).

[29] OWASP Foundation. *Unrestricted Access to Sensitive Business Flows.* 2023. URL: https://owasp.org/API-Security/editions/2023/en/0xa6-unrestricted-access-to-sensitive-business-flows/ (visited on 08/16/2023).

[30] Red Hat, Inc. *Common Security Advisory Framework (CSAF) beta files now available.* 2022. URL: https://www.redhat.com/en/blog/common-security-advisory-framework-csaf-beta-files-now-available (visited on 08/10/2023).

[31] Leon Schmidt. *Development of an API to request security advisories for CSAF 2.0, Bachelor's thesis, University of Applied Sciences, Offenburg.* 2022. URL: https://opus.hs-offenburg.de/files/6011/CSAF_API_development_v1.pdf (visited on 11/01/2022).

[32] Leon Schmidt. *Development of an API to request security advisories for CSAF 2.0, Bachelor's thesis, University of Applied Sciences, Offenburg.* 2022. Chap. 3.3.2. URL: https://opus.hs-offenburg.de/files/6011/CSAF_API_development_v1.pdf.

[33] Leon Schmidt. *Development of an API to request security advisories for CSAF 2.0, Bachelor's thesis, University of Applied Sciences, Offenburg.* 2022. Chap. 6.1. URL: https://opus.hs-offenburg.de/files/6011/CSAF_API_development_v1.pdf.

[34] Schneider Electric SE. *Cybersecurity support portal.* 2021. URL: https://www.se.com/ww/en/work/support/cybersecurity/security-notifications.jsp (visited on 02/28/2023).

[35] SICK AG. *SICK PSIRT Security Advisories.* 2021. URL: https://sick.com/.well-known/csaf/white/ (visited on 09/15/2023).

[36] Siemens Aktiengesellschaft. *Siemens Security Advisories.* 2021. URL: https://new.siemens.com/global/en/products/services/cert.html#SecurityPublications (visited on 02/28/2023).

[37] National Institute of Standards and Technology. *FIPS PUB 180-4: Federal Information Processing Standards PUBLICATION: Secure Hash Standard (SHS).* 2015. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (visited on 09/19/2023).

[38] TIBCO Software Inc. *TIBCO Security Advisories.* 2018. URL: https://www.tibco.com/.well-known/csaf/ (visited on 09/15/2023).

[39] US Cybersecurity & Infrastructure Security Agency. *Traffic Light Protocol (TLP) Definitions and Usage.* 2022. URL: https://www.us-cert.gov/tlp (visited on 03/28/2023).

A Appendix

A.1 CSAF list of dates

TLP label	**Definition**
TLP:RED	Not for disclosure, restricted to participants only. Sources may use TLP:RED when information cannot be effectively acted upon by additional parties, and could lead to impacts on a party's privacy, reputation, or operations if misused. Recipients may not share TLP:RED information with any parties outside of the specific exchange, meeting, or conversation in which it was originally disclosed. In the context of a meeting, for example, TLP:RED information is limited to those present at the meeting. In most circumstances, TLP:RED should be exchanged verbally or in person.
TLP:AMBER	Limited disclosure, restricted to participants' organizations. Sources may use TLP:AMBER when information requires support to be effectively acted upon, yet carries risks to privacy, reputation, or operations if shared outside of the organizations involved. Recipients may only share TLP:AMBER information with members of their own organization, and with clients or customers who need to know the information to protect themselves or prevent further harm. Sources are at liberty to specify additional intended limits of the sharing: these must be adhered to.
TLP:GREEN	Limited disclosure, restricted to the community. Sources may use TLP:GREEN when information is useful for the awareness of all participating organizations as well as with peers within the broader community or sector. Recipients may share TLP:GREEN information with peers and partner organizations within their sector or community, but not via publicly accessible channels. Information in this category can be circulated widely within a particularcommunity. TLP:GREEN information may not released outside of the community.
TLP:WHITE	Disclosure is not limited. Sources may use TLP:WHITE when information carries minimal or no foreseeable risk of misuse, in accordance with applicable rules and procedures for public release. Subject to standard copyright rules, TLP:WHITE information may be distributed without restriction.

Table 13: TLPv1 label definitions by FIRST.ORG, Inc

JSON PATH	**DESCRIPTION**
document.tracking.current_release_date	The date when the current revision of this document was released
/document/tracking/generator/date	This SHOULD be the current date that the document was generated. Because documents are often generated internally by a document producer and exist for a nonzero amount of time before being released, this field MAY be different from the Initial Release Date and Current Release Date.
/document/tracking/initial_release_date	The date when this document was first published.
/document/tracking/revision_history/date	The date of the revision entry
/vulnerabilities/discovery_date	Holds the date and time the vulnerability was originally discovered.
/vulnerabilities/flags/date	Contains the date when assessment was done or the flag was assigned.
/vulnerabilities/involvements/date	Holds the date and time of the involvement entry.
/vulnerabilities/release_date	Holds the date and time the vulnerability was originally released into the wild.
/vulnerabilities/remediations/date	Contains the date from which the remediation is available.
/vulnerabilities/threats/date	Contains the date when the assessment was done or the threat appeared.

Table 14: CSAF list of dates

A.2 CSAF API type defintions

```
type csafDocumentDocument {
    acknowledgments: [acknowledgments_t]
    aggregate_severity: csafDocumentDocumentAggregateSeverity
    category(
        exact: String
        should: String): String
    csaf_version(
        enum: csafDocumentDocumentCsafVersionEnum): String
    distribution: csafDocumentDocumentDistribution
    lang(
        exact: String
        should: String): String
    notes(
        audience: String
        category: String
        text: String
        title: String): [notes_t]
    publisher: csafDocumentDocumentPublisher
    references: [references_t]
    source_lang(
        exact: String
        should: String): String
    title(
        exact: String
        should: String): String
    tracking(id: String): csafDocumentDocumentTracking
}
```

Listing 83: GraphQL type definitions of CSAF /document

```
type full_product_name_t {
    name(
    exact: String
    should: String): String
    product_id(
    exact: String
    should: String
    idInProductTreeProductGroupsProductIds: Boolean
    idInProductTreeRelationshipsProductReference: Boolean
    idInProductTreeRelationshipsRelatesToProductReference: Boolean
    idInVulnerabilitiesProductStatusFirstAffected: Boolean
    idInVulnerabilitiesProductStatusFirstFixed: Boolean
    idInVulnerabilitiesProductStatusFixed: Boolean
    idInVulnerabilitiesProductStatusKnownAffected: Boolean
    idInVulnerabilitiesProductStatusKnownNotAffected: Boolean
    idInVulnerabilitiesProductStatusLastAffected: Boolean
    idInVulnerabilitiesProductStatusRecommended: Boolean
    idInVulnerabilitiesProductStatusUnderInvestigation: Boolean
    idInVulnerabilitiesFlagsProductIds: Boolean
    idInVulnerabilitiesRemediationsProductIds: Boolean
    idInVulnerabilitiesScoresProducts: Boolean
    idInVulnerabilitiesThreatsProductIds: Boolean
    ): String
    product_identification_helper:
    full_product_name_tProductIdentificationHelper
}
```

Listing 84: GraphQL type definitions of CSAF ./full_product_name_t

B Appendix: Data medium content

The *JavaScript* code and this master thesis on the CSAF API were burned to the disk. The folder structure of the disk is as follows:

- `/csaf_api`: The .js source files.
- `/paper`: The pdf version of this master thesis and the Selbständigkeitserklärung.