# Offenburg University of Applied Sciences
## Media Faculty

in cooperation with the

## Federal Office of Information Security (BSI)

### Bachelor's thesis

# Development of an API to request security advisories for CSAF 2.0

by

Leon Schmidt

Enterprise and IT Security

Supervision

Prof. Dr. Daniel Hammer, Offenburg University of Applied Sciences
Dr. Klaus Biß, Federal Office for Information Security (BSI) Germany
Thomas Schmidt, Federal Office for Information Security (BSI) Germany

Submission date

August 04, 2022

# Abstract

This work addresses the conceptualization, design, and implementation of an Application Programming Interface (API) for the Common Security Advisory Framework (CSAF) 2.0, introducing another method for distributing CSAF documents in addition to two already existing methods. These don't allow the use of flexible queries as well as filtering, which makes it difficult for operators of software and hardware to use CSAF. An API is intended to simplify this process and thus advance the automation goal of CSAF.

First, it is evaluated whether the current standard allows the implementation of an API. Any conflicts are highlighted and suggestions for standard adaptations are made. Based on these results, the API is designed to meet the previously defined requirements. Subsequently, a proof of concept is successfully developed according to the design and extensively tested with specially prepared test data. Finally, the results and the necessary standard adjustments are summarized and justified.

The conceptual design and the implementation were successfully completed. However, during the implementation of the proof of concept, some routes could not be fully implemented.

# Contents

# List of Figures

# List of Tables

# List of Listings

# List of Abbreviations

**API** Application Programming Interface

**CA** Certificate Authority

**CERT** Computer Emergency Response Team

**CGI** Common Gateway Interface

**CORS** Cross Origin Ressource Sharing

**CPE** Common Platform Enumeration

**CSAF** Common Security Advisory Framework

**CSIRT** Computer Security Incident Response Team

**CVE** Common Vulnerabilities and Exposures

**CVSS** Common Vulnerability Scoring System

**CWE** Common Weakness Enumeration

**DBMS** Database Management System

**DNS** Domain Name System

**HTML** Hypertext Markup Language

**HTTP** Hypertext Transfer Protocol

**JSON** JavaScript Object Notation

**NAT** Network Address Translation

**npm** Node Package Manager

**NTIA** National Telecommunications and Information Administration

**OAS** OpenAPI Standard

**PoC** Proof of Concept

**PURL** Package URL

**REST** Representational State Transfer

**ROLIE** Resource-Oriented Lightweight Information Exchange

**SBOM** Software Bill of Materials

**SKU** Stock Keeping Unit

**SNI** Server Name Indication

**SOC** Security Operations Center

**SSH** Secure Shell

**TLP** Traffic Light Protocol

**TLS** Transport Layer Security

**TOML** Tom's Obvious, Minimal Language

**URI** Unique Ressource Identifier

**URL** Unique Ressource Locator

**VEX** Vulnerability Exploitability eXchange

**VM** Virtual Machine

**YAML** YAML Ain't Markup Language

# 1 Introduction

## 1.1 Motivation

In the industrial environment, there is a large number of vendors with a correspondingly diverse range of products. Over their lifecycle, usually security vulnerabilities are discovered. As an administrator of a large company, it can be difficult to keep track of mitigating them, as the corresponding security advisories are usually published in a human-readable format. This requires the manual retrieval and processing of these advisories, which gets more difficult with every product – or deviating product version. This can result in multiple delays before an asset owner can respond to a vulnerability. This can lead to it remaining exposed after disclosure and being exploited by an attacker, potentially leading to a corporate disruption.

This problem is to be solved by the Common Security Advisory Framework (CSAF). Currently, the framework offers two methods for the distribution of security advisories. However, these only serve the purpose of finding all existing documents of a manufacturer – a search for advisories that are relevant for one's own operation is currently not possible with them before downloading. An Application Programming Interface (API) as a third distribution method is intended to address and solve this problem.

## 1.2 Objective and methodology

In this work, it is evaluated whether the current CSAF standard 2.0 allows the implementation of an API with extensive search and filter options. The necessary adjustments are to be named. A Proof of Concept (PoC) will also be developed, which is based on the source code of the CSAF distribution PoC [25] and extends it accordingly. Furthermore, it is briefly explained what added value an API has compared to the current distribution methods. The API is intended to be used in the CSAF 2.1 standard.

This leads to the following guiding questions, which are answered in this thesis:

- What are the benefits of establishing a CSAF API and which Computer Emergency Response Teams (CERTs) / Computer Security Incident Response Teams (CSIRTs) would use it?
- Are the contents of the current CSAF standard suitable for running an API on it?
- What needs to be adjusted to ensure operation and enable complex search queries?
- How well can an API be integrated into the current PoC?

This thesis is divided into the following sections in order to answer these guiding questions:

In chapter 2 "Basics of CSAF 2.0", the recent CSAF standard is discussed first. In order to lay the foundation for the following chapters, the areas which are relevant for the development of an API will be worked out and described. In this work, the Committee Specification 02 [17] is used as a reference, since the product branch type category `product_version_range` was added here. This allows products to be identified by a version range instead of just a specific version only [17, section 3.1.2.3.2] which can be of great advantage for the development of an API.

The following areas of the standard will be described:

**Schema Element Definitions** [17, section 3.1]

**Schema Element Properties** [17, section 3.2]

**CSAF Document Profiles** [17, section 4]

**Distributing CSAF documents** [17, section 7]

**Tests and Conformance** (partially) [17, sections 6, 9]

In addition, at the end of this chapter, the envisaged CSAF infrastructure will be described in order to show the areas of application of an API. It will become clear who the actors and profiteers are.

The following chapter 3 "Design" deals with the conceptual design of the API based on the basics explained in chapter 2.

First, the requirements for the development of an API are worked out. This is followed by the actual conceptual design of the API routes. It is important to mention that these are designed according to the black box principle. This means that the actual functionality of the API is not yet in the focus. Only the interface to the outside is defined. This ensures that access to the API is logical and independent of the actual technical conditions behind it. Accordingly, the conceptual design follows the "spec-first" approach: This means that the implementation is done after a specification has been created. The design will be produced in the OpenAPI 3.0 format [20] with the help of the Swagger online editor [23].

Subsequently, the feasibility of the design will be evaluated and it will be checked to which degree the requirements are fulfilled. If necessary, adaptations to the standard will be suggested informally. Here, a possible implementation will be explicitly discussed. It will also be considered to which degree the design would fit into the standard and how comprehensible it is to end users. With the help of this methodology, the design is continuously revised and adjusted.

A proof-of-concept will be developed and documented in chapter 4 “Proof of Concept”. This will be based on the source code of the CSAF distribution PoC [25] and extends it accordingly.

First, the CSAF distribution PoC will be used to set up a test environment with a CSAF provider and, ideally, a CSAF aggregator. This will be followed by the actual implementation and its documentation.

When the implementation is complete, a basic user interface will be prepared with which the API can be accessed for test purposes. The user interface will not be implemented by the developer itself, but will be set up with the help of existing tooling for creating API clients, such as Swagger UI [24].

Subsequently, the informal change proposals in the “Design” chapter are taken up again. Explicit standard adaptations are to be derived from them in chapter 5 “Integration into the specification”, which are to be incorporated into the next version of the CSAF standard. This will result in the specific requirements that the next version must meet so that the selected route design can be applied to it.

The resulting new standard will initially include all three distribution methods: Directory-based, Resource-Oriented Lightweight Information Exchange (ROLIE) feed, and the API. Finally, these are set up against each other in order to make the advantages and disadvantages of the API visible again.

Finally, the results of the work are summarized in chapter 6 “Conclusion and Outlook”. It is explained again which weaknesses the standard currently has in connection with the API and which improvements and adjustments are necessary. Likewise, it is described how it must be extended to allow the implementation.

The difficulties that arose during the work are considered and evaluated in the follow-up. Finally, it is explained how the results of this work can be dealt with and what added value is created as a result.

# 2 Basics of CSAF 2.0

This chapter examines the goal of CSAF, as well as the basics about the components of the standard. It serves as the basis for all further chapters.

The CSAF standard describes the format of a single security advisory or similar document (hereinafter "CSAF document") and their distribution. The sections 2.1 - 2.4 deal specifically with the document format, while section 2.5 focuses on with the distribution system.

## 2.1 Goals of CSAF

“The Common Security Advisory Framework (CSAF) Version 2.0 is the definitive reference for the language which supports creation, update, and interoperable exchange of security advisories as structured information on products, vulnerabilities, and the status of impact and remediation among interested parties.” [17, Abstract]

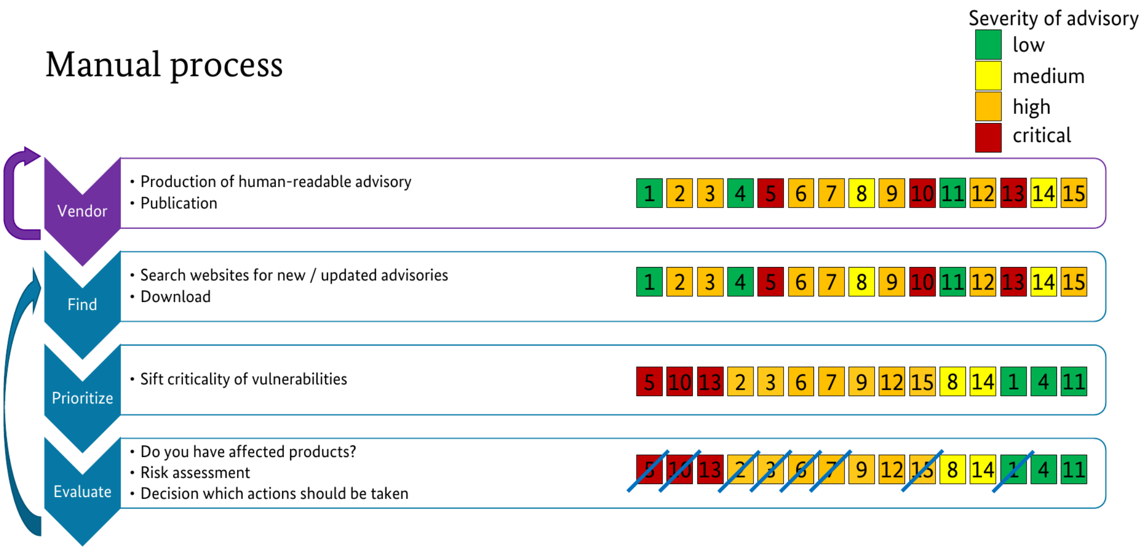


Figure 2.1: Manual process for handling security advisories without CSAF. Finding, prioritizing and evaluating advisories must be done manually. [18]

Figure 2.1 shows the manual process for handling security advisories. This describes how they are currently handled. Finding, prioritizing and evaluating security advisories today has to be done manually. First of all, the advisories must be searched for on the manufacturer’s website or retrieved by any other means. Depending on the size and heterogeneity of the infrastructure, this can be very difficult and, above all, time-consuming. A major problem here is the inconsistent format used by manufacturers to issue their security warnings. Once all the advisories have been found, they have to be processed and prioritized. Likewise, of course, vulnerabilities that do not affect products in the infrastructure must be sorted out. This process also has to be done manually. This does not scale well as the business grows.

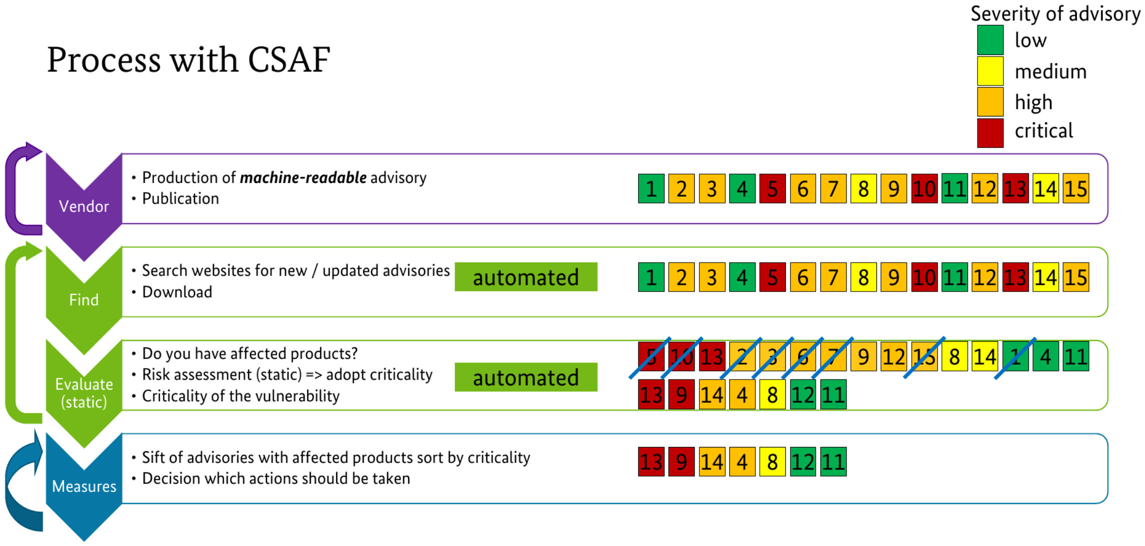

Process with CSAF
Severity of advisory
low
medium
high
critical
Vendor
• Production of machine-readable advisory
• Publication
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Find
• Search websites for new / updated advisories
• Download
automated
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Evaluate (static)
• Do you have affected products?
• Risk assessment (static) => adopt criticality
• Criticality of the vulnerability
automated
5 10 13 2 3 6 7 9 12 15 8 14 1 4 11
13 9 14 4 8 12 11
Measures
• Sift of advisories with affected products sort by criticality
• Decision which actions should be taken
13 9 14 4 8 12 11

Figure 2.2: Process for handling security advisories with CSAF with the automated steps “Find” and “Prioritize” [18]

With CSAF, the “Find” and “Evaluate” steps can be fully automated, as shown in figure 2.2. How the advisories are handled must and should continue to be decided by the customers themselves. This makes update management easier and scales with the size of the company.

CSAF supports the creation and exchange of security advisories in the machine-readable JavaScript Object Notation (JSON) format. The goal is to allow security advisory retrieval and processing to be automated. As a first step, an operator determines his product inventory. With the help of simple queries, security advisory information related to this inventory can be requested. This information can be used to carry out a more detailed risk assessment. Furthermore, the time frame between the design and dissemination of an advisory by the manufacturer to its retrieval by the consumer is significantly shortened by automation.

The CSAF 2.0 standard currently specifies two ways to distribute CSAF documents: the ROLIE-based and directory-based distribution. In this work, an API is introduced as a third distribution method.

## 2.2 Type definitions

This section explains the individual type definitions for a CSAF document [17, section 3.1]. They serve as reusable schemas which are later used in the property definitions of a CSAF document. These are addressed in the next section.

**acknowledgments_t** Array containing objects. Each object represents an acknowledgment, which can contain any number of names and Unique Ressource Locators (URLs), as well as an organization name and a summary.

**branches_t** Array containing objects, which itself can contain an element of type `branches_t` under the property `branches`. If it is not specified, the object is considered the leaf element of that branch. The property `product` must be filled with an element of type `full_product_name_t`. Each branch must end with a leaf containing a product. Adjacent to those properties, an enumerated `category` and a `name` must be specified. This results in a tree whose branches each represent a complete product.

**full_product_name_t** Object representing a product. The properties `name` and `product_id` are mandatory. The latter one has the type `product_id_t`. Thus, the type `full_product_name_t` is used to assign a unique product ID to a product. In addition, the property `product_identification_helper` can be specified to make it easier to identify this product in an asset database. A Common Platform Enumeration (CPE), Hashes, Model Numbers, a Package URL (PURL), Software Bill of Materials (SBOM) URLs, Serial Numbers, Stock Keeping Units (SKUs) and generic Unique Ressource Identifiers (URIs) can be specified here.

**lang_t** String specifying a language based on `IETF BCP 47 / RFC 5646` [2].

**notes_t** Array with notes that allows the input of free text. A note `category` must also be selected from the following enumeration: `description`, `details`, `faq`, `general`, `legal_disclaimer`, `other` and `summary`. Additionally, `audience` and `title` can also be specified for a note.

**product_group_id_t** String, which uniquely identifies a product group.

**product_groups_t** Array containing only elements of type `product_group_id_t`. Therefore, it lists product group IDs.

**product_id_t** String, which uniquely identifies an element of type `full_product_name_t`.

**products_t** Array containing elements of type `product_id_t`, listing product IDs.

**references_t** Array containing objects. Each object represents a reference. For each reference, a `url` and a `summary` must be specified. A reference to the current document can also be specified. For this, the property `category` must have the value `self`.

**version_t** String which allows the specification of a version number. Either integer or semantic versioning [15] must be used here.

## 2.3 Document properties

In this section, the individual properties of a CSAF document are briefly explained. This has the purpose of initially evaluating which of the properties would be suitable as search or filter parameters for an API. The parameters that define the initial data set before filtering are described as search parameters. They contain values that are easy to find in the CSAF document and that are independent of other parameters. Filter parameters, on the other hand, are suitable for reducing the acquired data set. They are mostly parameters that depend on other search parameters, such as the existence of a specific remediation *(filter parameter)* for a vulnerability *(search parameter)*.

The primary concern of this section is to recognize whether the properties are machine-readable and thus parsable. Especially for filter options, it is indispensable that the property values, which serve as data source, are deterministic. This is achieved primarily by string enumerations or standardized formats. Properties that contain free text are difficult to parse, or at least lower the quality of search results significantly. This is also true for properties whose approximate value is deterministic, but they do not follow a fixed format or enumeration. In this and the following sections and chapters, this problem will be discussed in detail.

### 2.3.1 The `/document` property

The `/document` property contains metadata about the CSAF document itself. It is the only property required by the CSAF JSON schema [17, section 3.2]. The following subproperties are mandatory:

**`/document/category`** Specifies the type and profile of the CSAF document. It defines which properties are required in addition to the `/document` property. Since this field defines how a CSAF document must be structured, it is of great importance.

**`/document/csaf_version`** Defines the CSAF version of the document. Its value is always `2.0` in the current standard.

**`/document/publisher`** Provides information about the publisher of the document. The name, category and namespace (e.g. website) of the issuing party is mandatory.

**`/document/title`** Title of the CSAF document.

**`/document/tracking`** This field contains all the management attributes required for tracking the document. At least the following properties must be specified here: `id`, `current_release_date`, `initial_release_date`, `revision_history`, `status` and `version`. The dates can be particularly useful as search parameters, because they can be used to narrow down the search to a specific time period. The `id` is also part of the global identifier used in the CSAF network. The `status` property, which is enumerated, can also be used to determine whether a security advisory is complete.

In addition, the following subproperties can be specified optionally:

**`/document/acknoledgements`** Other persons or organizations that contributed to the content of the CSAF document are referenced here.

**`/document/aggregate_severity`** Defines the criticality of the document. It conveys the urgency with which the one or more vulnerabilities reported should be addressed and must be specified as free text in the `text` property. Optionally, a `namespace` can be supplied.

**`/document/distribution`** Here, the rules for distributing the document are defined. These can be specified either as free text in the `distribution/text` subproperty or by assigning a Traffic Light Protocol (TLP) [1] label to `distribution/tlp/label`.

**`/document/lang`** Sets the language of this document.

**`/document/source_lang`** Sets the language of the original document, if this document is a translation.

**`/document/notes`** Provides the possibility to record annotations to the document as free text.

**`/document/references`** External reference documents (also non-CSAF) can be listed here.

Most of the fields in the `/document` property are searchable or deterministic, since the values are defined as string enumerations. Many of them are therefore suitable to act as data sources for simple filter functions. Searching for a specific manufacturer should be trivial, especially because of the `/document/publisher/namespace` property. Likewise, global findability should be possible and unproblematic. The properties `/document/publisher/namespace` and `/document/tracking/id` are used to globally identify a single CSAF document [17, section 3.2.1.12.4].

However, it becomes problematic with the optional properties `/document/aggregate_severity` and `/document/distribution`: The first one requires only the subproperty `text`, in which a severity level of the document can be specified using free text. This may make it difficult to process. The latter one has the subproperties `text` and `tlp`, but only one of them is required. The `tlp` subproperty is easy to parse because its `label` subproperty is standardized and enumerated [1]. In the `text` subproperty, on the other hand, free text is allowed. This makes it impossible for a computer to deterministically determine whether a document may be distributed or not. To prevent information leakage, a restriction according to `TLP:RED` must be assumed if only the `text` subproperty is set!

The optional `/document/lang` property, on the other hand, has to conform to the IETF BCP 47 / RFC 5646 standards [2], and is therefore suitable as a data source for filter functions.

The remaining properties `/document/{acknowledgments,notes,references,source_lang}` are not useful as a source for filter functions. Despite them having a high importance inside the CSAF document itself, it is uncommon for them to be used as search parameters.

### 2.3.2 The `/product_tree` property

The `/product_tree` property is used in the document as a definition point for products. Here, all products are defined that are, for example, affected by or related to a vulnerability. Each object receives a unique ID [17, section 3.2.2], which can be used to reference it elsewhere in the document.

Products can be defined in these ways:

**`/product_tree/full_product_names`** Allows the uncomplicated specification of a single product including all necessary information (manufacturer, product name, version). Each defined product receives a unique product ID. This is the simplest way to specify products.

**`/product_tree/branches`** Enables the hierarchical specification of products. For example, several versions of a single product or several products from a single product family can be specified. This results in branches, which in the end results in a complete product with its own ID. Listing 2.1 shows an example of the `branches` property.

Listing 2.1: Simple example for the `/product_tree/branches` property with Product XY from Vendor XY in version 2

```
...
"branches": [
    {
        "name": "Vendor XY",
        "category": "vendor",
        "branches": [
            {
                "name": "Product XY",
                "category": "product_name",
                "branches": [
                    {
                        "name": "2",
                        "category": "product_version",
                        "product": {
                            "name": "Product XY v2",
                            "product_id": "SP1",
                            "product_identification_helper": {
                                ...
                            },
                        }
                    }
                ]
            }
        ]
    }
]
...
```

Each step on the branch requires the properties `name`, `category` and `branches`. The only exception is the last element of the branch ("leaf element"): Here the `product` property with the type `full_product_name_t` must be specified instead of `branches`. The type is the same as the one that must be used in the `full_product_names` property. Consequently, the `product_id` is assigned here.

To avoid complications when parsing this representation, the standard recommends the approximate order of categories: `vendor -> product_name -> product_version`. However, this is just a recommendation and not a requirement. There is an informative test that states that the other categories (`architecture`, `host_name`, `language`, `legacy`, `patch_level`, `product_family`, `product_version_range`, `service_pack` and `specication`) [17, section 3.1.2.2] can be used before, in between or after the recommended order [17, section 6.3.9]. All categories can also be defined multiple times in a branch, which could complicate parsability in a real world scenario.

**`/product_tree/relationships`** Within this array, previously defined products can be placed in relation to each other. For example, this is necessary for a software which is only affected by a vulnerability if it is part of another system or is otherwise connected to it. The relationship categories that can be used are: `default_component_of`, `external_component_of`, `installed_on`, `installed_with` and `optional_component_of`. A relation yields a new product ID that can be used to reference it like a product defined in the `branches` or `full_product_names` property. [17, section 3.2.2.4]

**`/product_tree/product_groups`** Additionally, the defined products can be grouped in the property `product_groups`. Compared to the other properties, the group does not create a new ID of type `product_id_t`, but a group ID of type `product_group_id_t`.

### 2.3.3 The `/vulnerabilities` property

The `/vulnerabilites` property contains an array of vulnerability information for this CSAF document. More than one can be specified, since e.g. during a security audit several vulnerabilities can be discovered at once, which are then all distributed in a single security advisory.

Each entry in the vulnerability property can be of any kind: it can be a fully parsable element containing Common Vulnerabilities and Exposures (CVE), Common Weakness Enumeration (CWE) and references, or just free text information about the vulnerability. According to the standard, the only requirement is that at least one of the following subproperties must be included. However, the document profiles provide further rules on how a vulnerability object must be structured (see section 2.4).

**`/vulnerabilities[]/acknowledgments`** The `acknowledgments` property contains a list of parties associated with the vulnerability. It is not important in this thesis as it is not suitable to be used in search queries.

**/vulnerabilities[]/cve** Contains the MITRE standard Common Vulnerabilities and Exposures (CVE) string to identify the vulnerability. Only one can be specified per item in the /vulnerabilities array. This property is eligible to be used as a search parameter.

**/vulnerabilities[]/cwe** Contains the MITRE standard Common Weakness Enumeration (CWE) Weakness ID and its name. Like the /vulnerabilites/cve property, it can be used as a search parameter.

**/vulnerabilities[]/discovery_date** Date on which the vulnerability was discovered. This may be a better alternative to the /document/tracking/initial_release_date property for date filtering, as it relates specifically to the vulnerability.

**/vulnerabilities[]/flags** Allows setting machine-readable flags to specify the impact of a vulnerability on a specific product or product group. The following values in the label subproperty are possible:

- component_not_present
- inline_mitigation_already_exist
- vulnerable_code_cannot_be_controlled_by_adversary
- vulnerable_code_not_in_execute_path
- vulnerable_code_not_present

In addition to this, the respective product or product group can then be specified via its ID, as well as the date when the flag was assigned.

The flags usually provide a justification as to why a product is not affected by a vulnerability. It is unusual for such information to be used as a search parameter. However, the property can be useful as a filter parameter.

**/vulnerabilities[]/ids** The /vulnerabilites/ids property is a list of arbitrary IDs or labels, if any. They serve the purpose of identifying a vulnerability which, for example, have not yet been assigned a CVE. Each list entry requires the properties system_name and text. The former specifies the ID system and the latter the actual ID inside this system (e.g. system_name="Github Issue", text="oasis-tcs/csaf#210"). Both properties are defined as free text here. Thus, a reliable search for CSAF documents using these ID systems is problematic. An enumeration is presumably hardly realizable, since this property has the function to assign IDs on the basis of arbitrary systems. Nevertheless, this property can be useful as a search parameter as the values of system_name are somewhat deterministic although they are not enumerated.

**/vulnerabilities[]/involvements** This property can be used to specify third parties (coordinator, discoverer, other, user or vendor) who were involved in the vulnerability disclosure process. Furthermore, the status of the communication (completed, contact_attempted, disputed, in_progress, not_contacted or open) must be specified for each party. Additionally, a date of involvement and a free text summary can be provided. Eventually, the use case of finding CSAF documents in which a particular party was involved may arise from this property. However, it is unlikely that this will add value to the goal of automating advisories.

**/vulnerabilities[]/notes** This property can be used to store free text notes for the /vulnerability entry. Each note has to choose from the following selection in the category subproperty to categorize the note: description, details, faq, general, legal_disclaimer, other and summary. Additionally, the audiences subproperty can be used to specify the target audience of the note. The notes property, like the /document/notes property, is not suitable to be used as a search parameter for the same reason.

**/vulnerabilities[]/product_status** Describes the status of whether and how a product is affected by this vulnerability. The following states can be specified by creating subproperties from the following selection under /product_status: first_affected, first_fixed, fixed, known_affected, known_not_affected, last_affected, recommended and under_investigation. The product IDs from the /product_tree property are then specified under the respective properties. When requesting CSAF documents, these states can be useful as a filter option when transferring your own device lists. Thus, the following search query is created as an example "Find all CSAF documents that (possibly) affect my devices ({known,first,last}_affected (or under_investigation))".

**/vulnerabilities[]/references** Lists references to the vulnerability (e.g. documentation in other formats, details from other sources, etc.). Like the /document/references property, this is not suitable to be used as a search parameter.

**/vulnerabilities[]/release_date** Contains the date when the vulnerability was published. It may be used as a fallback to the /vulnerabilities/discovery_date property.

**/vulnerabilities[]/remediations** Under this property, remediations for the vulnerability can be specified with respect to one or more products or product groups, if applicable. The following categories can be selected: workaround, mitigation, vendor_fix, none_available and no_fix_planned. Free text details and at least one product or product group must be provided. Since this information is basically what you would expect as the result of a query, this property is also rather unsuitable as a search parameter. But when submitting a device list, for example, it can be useful to filter for vulnerabilities, for which a mitgation is available.

**/vulnerabilities[]/scores** Contains a list of Common Vulnerability Scoring System (CVSS) scores. CVSS v2, v3 or both can be specified. For each specified score, the products that are affected by it must be specified. The fact that multiple CVSSs can be specified here could make it difficult to use them as search parameters. However, they are indispensable as filter parameters.

**`/vulnerabilities[]/threats`** This property allows specifying multiple threats with the following categories: `exploit_status`, `impact` or `target_set`. Thus, it is possible to specify how mature exploits already are (`exploit_status`), what influence the vulnerability can have (`impact`), or which group of persons or devices could be affected (`target_set`). In addition to these properties, products or product groups can be referenced here again, and a date can be set. It is intended that the values of each entry in `threats` can change over time. Like `/vulnerabilites[]/remediations`, this property is not suitable as a search parameter for the same reason. Other than the remediations however, the `details` property is non-parsable which makes the `/vulnerabilities[]/threats` unsuitable as a filter parameter.

**`/vulnerabilities[]/title`** Allows to specify a canonical name for the vulnerability. May be used for direct document search ("Find the CSAF document where this vulnerability is listed with this title").

### 2.3.4 Summary

The following table 2.1 summarizes the detailed descriptions of the CSAF document properties and their search parameter suitability. For each property, it is indicated whether it is suitable (S) and useful (U) as a specification in a request to the API. The decisions are briefly justified in each case.

The decision of suitability is mainly based on properties like parsability and clearness. Usability, on the other hand, has more of a non-technical basis. For example, a user's expectation is important here: some properties are simply expected to be part of a response rather than a query. Likewise, some values are static, so they are of no use as part of a request. Generally, a property is marked as usable as soon as it has an informational added value as a search or filter parameter for the user.

Table 2.1: CSAF document property suitability (S) and usability (U) summary

| **CSAF document property suitability (S) and usability (U) summary** | | | |
|---|---|---|---|
| Property under `/document` | S | U | Reason |
| `/category` | ✓ | ✓ | Defines document category |
| `/csaf_version` | ✓ | ✕ | Is always `2.0` [(i)] |
| `/publisher` | ✓ | ✓ | Part of global ID |
| `/title` | ✓ | ✓ | Title of the document |
| `/tracking` | ✓ | ✓ | Part of global ID and contains document status |
| `/acknowledgments` | ✕ | ✕ | Usually expected as response |
| `/aggregate_severity` | ✕ | ✓ | Useful for filtering, but non-parsable |
| `/distribution` | ✕ | ✓ | Source for access control decisions, TLP property not required |
| `/lang` | ✓ | ✓ | Useful for filtering |
| `/source_lang` | ✓ | ✕ | Usually expected as response |
| `/notes` | ✕ | ✕ | Usually expected as response |
| `/references` | ✕ | ✕ | Usually expected as response |
| Property under `/product_tree` | S | U | Reason |
| `/full_product_names` | ✓ | ✓ | Easy to parse |
| `/branches` | ✓ | ✓ | Hard to parse, missing default order of categories |
| `/relationships` | ✓ | ✓ | Easy to parse |
| `/product_groups` | ✓ | ✓ | Easy to parse |
| Property under `/vulnerabilities[]` | S | U | Reason |
| `/acknowledgments` | ✕ | ✕ | Usually expected as response |
| `/cve` | ✓ | ✓ | Easy to parse |
| `/cwe` | ✓ | ✓ | Easy to parse |
| `/discovery_date` | ✓ | ✓ | Alternative for dates in `/document` |
| `/flags` | ✓ | ✓ | Expected as response, but also useful as filter parameter |
| `/ids` | ✕ | ✓ | Useful as search parameter; `system_id` not enumerated, but guessable |
| `/involvements` | ✕ | ✕ | Usually expected as response |

[(i)] Based on the current CSAF version; changes in later versions

| Property under `/vulnerabilities[]` | S | U | Reason |
|---|---|---|---|
| `/notes` | ✗ | ✗ | Usually expected as response |
| `/product_status` | ✓ | ✓ | Useful as filter parameters |
| `/references` | ✗ | ✗ | Usually expected as response |
| `/release_date` | ✓ | ✓ | Alternative for dates in `/document` |
| `/remediations` | ✓ | ✓ | Useful as filter parameter |
| `/scores` | ✓ | ✓ | Useful as filter parameter |
| `/threats` | ✗ | ✗ | `details` property is non-parsable |
| `/title` | ✓ | ✓ | Useful for filtering |

In theory, many of the properties are suitable for implementing an API on top of them. However, it becomes problematic with the properties that are marked as not suitable (S: ✗) but usable (U: ✓). For some of these, specification adaptations are suggested in the following chapters.

## 2.4 Document profiles

In addition to the single requirement that a CSAF document must have only the `/document` property, so-called document profiles are defined that specify further requirements for a document. A profile defines a use case that requires the necessary fields in the document to fulfill it. Inheritance of profiles is also possible, but an inheriting profile may only add further required properties to the inherited profile.

The individual profiles of the standard are briefly explained below. The resulting use cases can be used to make better decisions regarding the importance or suitability of individual properties.

The following profiles are defined by the standard [17, section 4]:

**CSAF Base** This profile lays the absolute foundation for a CSAF document. All other profiles inherit the properties of this profile. Furthermore, it serves as a "catch all" if the profile specified in `/document/category` does not exist and the document therefore does not belong to any other profile.

It is required that the document specifies a document category, CSAF version and a title. It is also required to specify the publisher and some tracking information in `/document/tracking`:

- `/document/category` must be `csaf_base`
- Publisher category, name, and namespace
- Tracking release date (current and initial)
- Tracking revision history
- Tracking status and version

CSAF Base documents therefore must only transport metadata about the publisher and the document itself, but must not contain any useful security information. The properties provided for that purpose are only enforced by the subsequent profiles. However, all non-enforced properties can still be set.

**Security Incident Response** This profile serves the use case to provide response for a security breach or incident. For example, a vulnerability in another vendor's product can be conveyed if it affects one of the vendor's own products.

The following rules are enforced by this profile:

- `/document/category` must be `csaf_security_incident_response`.
- At least one element in `/document/notes` with a `category` of `description`, `details`, `general` or `summary` must exist.

- At least one element in `/document/references` with a `category` of `external` must exist.

**Informational Advisory** This profile is suitable for transporting information that is not related to a vulnerability. For example, it can be used to point out misconfigurations.

The following rules are enforced by this profile:

- `/document/category` must be `csaf_informational_advisory`.
- At least one element in `/document/notes` with a `category` of `description`, `details`, `general` or `summary` must exist.
- At least one element in `/document/references` with a `category` of `external` must exist.
- The property `/vulnerabilities` shall not exist.

If the property `product_tree` exists, it must also be assumed that all specified products are affected. The property `/vulnerabilities` must not be specified.

**Security Advisory** This profile is suitable for transporting vulnerability information and its remediations - in other words, it represents the classic security advisory. This is to inform about vulnerabilities and their effects, and how they can be mitigated by users.

The following rules are enforced by this profile:

- `/document/category` must be `csaf_security_advisory`.
- All products related to this advisory must be listed in `/product_tree` regardless of their state. Thus, it works differently from the Informational Advisory profile, where all products in the `/product_tree` must be assumed to be affected.
- Each vulnerability object in `/vulnerabilities[]` must have a `notes` property.
- Each vulnerability object in `/vulnerabilities[]` must have a `product_status` property.

**VEX** This profile is suitable for transporting information according to the Vulnerability Exploitability eXchange (VEX) concept. The goal of VEX is to provide operators with additional information about whether a product is affected by a vulnerability or not and what countermeasures can be taken if necessary. In many cases, indicated vulnerabilities are not exploitable at all because, for example, the affected component is not present, or another security measure prevails against it. [10]

VEX was originally conceptualized by the National Telecommunications and Information Administration (NTIA) to find out whether a software is affected by an existing vulnerability via SBOM documents. Accordingly, a CSAF feed can be stored in an SBOM document for each software component. In the case of the SBOM format CycloneDX this specification is located in the property `/components[]/externalReferences[type="advisories"]/url` [5].

VEX is implemented in CSAF by the profile of the same name as follows:

- `/document/category` must be `csaf_vex`
- All products related to this advisory must be listed in `/product_tree` regardless of their state
- Each vulnerability in `/vulnerabilities[]` must be assigned either a CVE or an ID as mentioned in chapter 2.3.3: `/vulnerabilities[]/ids`. In addition, details must be provided in `/vulnerabilities[]/notes`.
- The products must be appropriately placed in the `/vulnerabilities[]/product_status/{fixed, known_affected, known_not_affected, under_investigation}` properties to set their status.
- For each product in `/vulnerabilities[]/product_status/known_affected` there must be either a machine-readable flag in `/vulnerabilities[]/flags` or a human-readable justification in `/vulnerabilities[]/threats` with a category of `impact`. This is called an impact statement.
- For each product in `/vulnerabilities[]/product_status/known_not_affected`, product-specific information must be specified in `/vulnerabilities[]/remediations` as to why the product is not affected. This is called an action statement.

These specifications are used to set a status for each product. In addition, reasons are given for the definite states `known_affected` and `known_not_affected`.

The fact that the profiles provide some kind of scheme ensures the combination of certain properties that would otherwise not provide any added value on their own. For example, the `/vulnerabilities[]/remediations` property is only enforced if `/vulnerabilities[]/product_status/known_affected` also exists. Otherwise, the `remediations` property would have no value, since it is not clear whether the product is affected by the named vulnerability at all.

However, the fact that the result of a request to the CSAF API cannot be guaranteed to consist only of homogeneous document types could be problematic. This could significantly limit the effectiveness of filter parameters, as some of these cannot be applied to CSAF documents of a particular profile. For example, filtering by VEX product status can only be effectively applied to documents that also satisfy the VEX profile.

## 2.5 Intended architecture

A CSAF infrastructure consists of the components “CSAF publisher”, “CSAF provider”, “CSAF trusted provider”, “CSAF lister” and “CSAF aggregator”. These components are also called roles in the standard [17, section 7.2]. They can be divided into the two groups “issuing parties” and “mirroring parties”. The former group is responsible for creating and issuing CSAF documents. It includes the CSAF publisher, provider and trusted provider. Usually, all documents originate from these entities. They can be operated by a manufacturer and therefore publish only those documents, which correlate with the products of this manufacturer. In addition, coordinators or discoverers can also distribute CSAF documents. A CSAF provider – in contrast to a CSAF publisher – additionally provides options that facilitate the automated retrieval of these documents. This includes above all the `provider-metadata.json` file, which must be located in a known or well-defined directory on a web server. It contains all the information about the provider itself, as well as the methods for distributing its CSAF documents.

The CSAF 2.0 standard currently defines two options to provide CSAF documents in a structured manner to allow their distribution [17, sections 7.1.11 - 7.1.17]: The ROLIE feeds and the directory-based distribution. ROLIE is a resource-oriented approach for security automation information publication, discovery, and sharing [7, Abstract]. It is built on top of the Atom Publishing Format and Protocol to ease discovery of security content as web-addressable resources and follows the Representational State Transfer (REST) architectural style. The CSAF standard currently uses the JSON representation of ROLIE feeds. These are referenced in the `provider-metadata.json` file instead of forcing the use of service documents mentioned in the ROLIE standard [7, section 5.1]. However, they are listed as an optional requirement [17, section 7.1.16]. The directory-based distribution only dictates where in the web directory the CSAF documents must be located. They must be sorted by year and TLP label.

Both methods intentionally do not provide search or filter functions to avoid information leakage to the server. All CSAF documents must always be downloaded and processed locally to obtain the relevant information. An API is to provide more flexible ways to query CSAF providers and aggregators.

Within the issuing parties, it is conceptually possible to implement an API without any issues. It only needs to be listed as a distribution method in `provider-metadata.json` so that queriers of this party can discover it. Each issuing party will accordingly operate its own API endpoint.

The mirroring parties ensure that these documents from the issuing instances can be retrieved at a single point. Operators of these parties decide by themselves which issuing parties are to be mirrored. A distinction is made between CSAF listers and aggregators. The former simply lists the issuing parties’ `provider-metadata.json` file in its own `aggregator.json` file so that they can be found more easily. However, requests for documents must still be made directly to the issuing party. A CSAF aggregator, on the other hand, mirrors all documents under a separate namespace on itself, so that CSAF documents can be requested directly from it. These namespaces are also listed in the `aggregator.json` file. The mirrored documents must be updated regularly.

The resulting namespace is composed of the name of the publisher, which is stored at the issuing party to be mirrored. An example for the publisher "Some_CERT" can be seen in listing 2.2.

Listing 2.2: Access paths to the `provider-metadata.json` on a provider versus on an aggregator

```
Provider:     /.well-known/csaf/provider-metadata.json
Aggregator:   /.well-known/csaf/Some_CERT/provider-metadata.json
```

This might introduce a problem later on: A single aggregator must then operate as many API endpoints as the underlying providers do. This also means that the APIs cannot communicate with each other, and thus the use case to find CSAF documents by publisher name might be more complicated to achieve. Since the API approach means that there are no longer any direct access paths to the CSAF documents, this problem can easily be circumvented by the implementation. An aggregator, like a provider, would therefore only need to access its local database of CSAF documents and merge them into a single API response.

The API will first be designed for CSAF providers only and adapted to aggregators after completion. The reason for this is that a provider functions completely independently and potential exchange functionality is only required by an aggregator. Basically the difference of the API routes between provider and aggregator should be minimal.

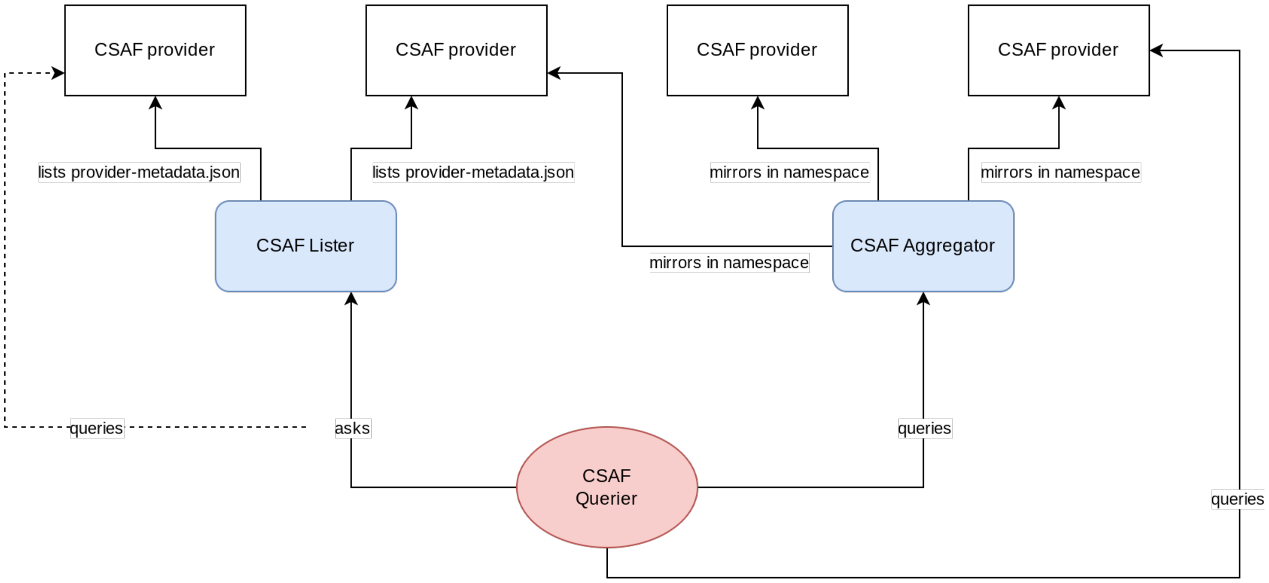


Figure 2.3: Data flow when querying CSAF providers, listers and aggregators

In summary, the diagram in figure 2.3 shows the simplified data flow of a CSAF infrastructure. It originates from a querier (red oval) and shows which options there are for requesting CSAF documents. The querier might be representative for an enterprise's Security Operations Center (SOC) or a similar facility.

The API to be conceptualized should not significantly change this data flow. A querier must still be able to request both providers and aggregators. As described, the API should not be implemented for listers, as they should only reference the implementations in the providers. CSAF publishers are also not considered, since they do not distribute CSAF documents in an automatable manner.

# 3 Design

This chapter covers the design and conception of the API. First, requirements are defined which must be fulfilled in order to serve the purpose of automating CSAF. The resulting design requirements are then fulfilled by the route design. Finally, it is evaluated and checked whether and how it can theoretically be implemented with the current standard.

The result of this chapter is a complete concept of the API, which is implemented by the Proof of Concept in chapter 4.

## 3.1 Requirements

The following requirements are not to be confused with the requirements defined in the standard for the CSAF roles [17, section 7.1] and are therefore not to be adopted into it. They serve as a methodology and guide for the design and implementation.

The methodology to set requirements is intended to define the "utopian end product" of the API. In the course of this thesis, this end product is to be conceptualized and then evaluated to what extent the requirements can actually be fulfilled by the standard. If this is not possible in a certain point, a change proposal is suggested. It should contain standard adjustments, by which the fulfillment of the requirement is made possible.

The following naming convention is used to define the requirements:

- `<category>-R<integer>`: a requirement (example: `INF-R10`: informal requirement 10)
- `<category>-R<integer>-A`: a change proposal by which the fulfillment of a requirement is made possible (example: `INF-R10-A`: Adjustment for informal requirement 10)

In the following, the requirements are divided into four categories. In the individual steps of this thesis, the requirements of the categories are to be fulfilled.

- **Informal requirements (`INF`)**: Basic requirements for a REST API. They have to be fulfilled by the operator of the server.
- **Design requirements (`DES`)**: These will be fulfilled solely by the route design made in section 3.2 - thus independent of the actual implementation. The category contains all requirements for the interface to the end user. Accordingly, only these are visible to the outside.

- **Implementation requirements (`IMP`)**: These are to be observed and fulfilled in chapter 4. They only concern the implementing authority of the API, not the end user or the operator of an existing implementation.
- **Requirements for embedding in the CSAF infrastructure (`EMB`)**: These requirements define rules for the API so that it can be embedded in the CSAF infrastructure and standard.

The informal requirements as defined in table 3.1 are not discussed further in this thesis. They are to be taken as given or fulfilled purely by the operator of the API. Nevertheless, it is important to mention them here in order to better understand design decisions. Although these requirements are to be taken as given, they still influence the design and implementation.

Table 3.1: Informal requirements

| **Informal requirements** | |
|---|---|
| Naming | Description |
| `INF-R1` | The entire API endpoint must be TLS encrypted according to modern standards. |
| `INF-R2` | Requests to the API must be idempotent: Two identical requests must generate an identical response. However, depending on the timing of the request, the response may vary, since the API is intended to serve the purpose of obtaining up-to-date CSAF documents. |
| `INF-R3` | The API must accept all Hypertext Transfer Protocol (HTTP) methods from RFC 7231 [16] – including Cross Origin Ressource Sharing (CORS) preflight requests with the `OPTIONS` method. |

Important in the selection of the design requirements as defined in table 3.2 was the evaluation of each property. Not every property is useful to be used in a search query. If possible, the use of properties that were marked as non-suitable should be avoided. However, as many use cases as possible should be covered, which form the interface to the user of the API. The following implementation requirements should not play a role in the definition of the design requirements.

Table 3.2: Design requirements

| **Design requirements** | |
|---|---|
| Naming | Description |
| `DES-R1` | All API routes must be versionable. |
| `DES-R2` | There must be a route that returns the CSAF role of the server. This is mainly for the purpose of distinguishing between aggregators and providers. |
| `DES-R3` | There must be a route that returns `aggregator.json` in the case of an aggregator, or `provider-metadata.json` in the case of a provider. |
| `DES-R4` | The API must allow at least one form of authentication on the routes, whose purpose is to return CSAF documents. However, this must not be enforced. |
| `DES-R5` | The routes must provide the ability to specify zero or more filter options in a single request. For example, these search queries must be possible in a single request:<br>• Search for CSAF documents. *(zero filter options)*<br>• Search for CSAF documents published after 2020. *(one filter option)*<br>• Search for CSAF documents published between 2020 and 2021. *(two filter options)* |
| `DES-R6` | There must be a filter option to match CSAF documents released before a specific time based on te initial release date. |
| `DES-R7` | There must be a filter option to match CSAF documents released after a specific time based on the initial release date. |
| `DES-R8` | There must be a filter option to match CSAF documents with a specific tracking state. |
| `DES-R9` | There must be a filter option to match CSAF documents of a specific document profile. |
| `DES-R10` | There must be a boolean parameter that specifies whether the hashes for each CSAF document should be included in the response. |
| `DES-R11` | There must be a boolean parameter that specifies whether the signature for each CSAF document should be included in the response. |
| `DES-R12` | There must be a function to pass a device list to the API to request all relevant documents. |

| Naming | Description |
|---|---|
| `DES-R13` | When providing a device list to the API there must be a filter option to only match the CSAF documents where the provided devices have the specified VEX status. |
| `DES-R14` | When providing a device list to the API there must be filter options to only match the CSAF documents, where any of the products have a CVSS score for any vulnerability in a specific score range. Both version 2 and 3 must be supported. |
| `DES-R15` | When providing a device list to the API there must be a filter option to only match the CSAF documents that provide a remediation of the specified type for any of the devices. |
| `DES-R16` | There must be a function to access arbitrary JSON properties to find CSAF documents with matching property values. |
| `DES-R17` | There must be a function to find the latest version of a CSAF document by its globally unique ID (`/document/publisher/namespace` + `/document/tracking/id`). |
| `DES-R18` | All API requests should contain an `/error` property signaling a machine-readable error code, even on responses not returning a status code in the range of 200-299. If no error occured, the property should have a value of `null` or be omitted. |
| `DES-R19` | The API must always return complete CSAF documents. |

Implementation requirements as defined in table 3.3 must be selected very carefully, as they determine how well the API integrates with the current implementations of the other CSAF components. They also form the guideline after the PoC implementation is made in this thesis in chapter 4.

In addition, the definition of the design requirements in table 3.2 already specifies how authentication must take place. Contrary to the original idea of providing different API endpoints for each TLP label [17, section 7.1.15], the approach chosen here was to optionally transmit authentication data. The API uses this data to decide which documents may be included in the calculation of the response. This response can then be filtered by TLP label, for example by using URL parameters. This simplifies the integration of the API into an automation process: An end user would no longer need to address multiple API endpoints to find all relevant documents. Similarly, access to documents that are not intended for the public, or where it is not obvious if they are, must remain denied to users without valid authentication data. The reason for this may be, for example, a missing TLP label in the document. To avoid information leakage, this needs to be well tested and verified.

Table 3.3: Implementation requirements

| **Implementation requirements** | |
|---|---|
| Naming | Description |
| `IMP-R1` | The API uses only the HTTPS request and the existing CSAF documents stored in the file system as data source. |
| `IMP-R2` | The API does not use a Database Management System (DBMS). |
| `IMP-R3` | Unauthenticated requests to the API may only process CSAF documents with a `TLP:WHITE` label. |
| `IMP-R4` | Authentication data, if available, must be verified before CSAF documents are processed. The permission level of the authentication data is used to decide which CSAF documents may be used to compute the response. |
| `IMP-R5` | If authentication data is submitted and it is not known to the server or it does not match the specified format, a `401 Unauthorized` error should be reported. In this case, not even `TLP:WHITE` documents should be processed. |

The requirements for embedding in the CSAF infrastructure as defined in table 3.4 serves to ensure that the API, as a third distribution method, fits in well with the existing ones. Among other things, this involves designing the API in such a way that it can function as far as possible in the same way on both CSAF (trusted) providers and aggregators without complex adaptations.

This can become a problem under certain circumstances, since the ROLIE feeds are already provided under their own namespace by aggregators and thus multiple `provider-metadata.json` files exist. However, the goal here should be that there is only one API endpoint per aggregator, which processes the documents of all (trusted) providers it mirrors. As it is at the moment, it is common for each provider to operate its own CSAF trusted provider. Should several of these be aggregated by a CSAF aggregator, it would thus also be possible to request documents from several manufacturers at once. This is especially helpful in infrastructures, where products from several vendors are used. Users having infrastructures with products mainly from a single vendor can still use the API of a CSAF (trusted) provider.

Table 3.4: Requirements for embedding in the CSAF infrastructure

| **Requirements for embedding in the CSAF infrastructure** | |
|---|---|
| Naming | Description |
| `EMB-R1` | The implementation of the API must work on both CSAF (trusted) providers and aggregators. Listers do not require an implementation, as they only reference the CSAF (trusted) provider instances, running the API. |
| `EMB-R2` | The API endpoint must be able to be stored in the `provider-metadata.json` file. |
| `EMB-R3` | The JSON schema of the `provider-metadata.json` must be adapted accordingly due to `EMB-R2`. |
| `EMB-R4` | The API endpoint must be able to be stored in the `aggregator.json` file. |
| `EMB-R5` | The JSON schema of the `aggregator.json` must be adapted accordingly due to `EMB-R4`. |

## 3.2 API design

Since the main goal of the API is to drive the automation of CSAF, the REST-JSON paradigm was chosen. REST is a very flexible paradigm that is, above all, easy to understand, well documentable, and offers a high level of compatibility with existing infrastructure and tooling. It is based on the HTTP and therefore allows easy integration into existing applications. REST can use any structured language to transport information, but JSON was chosen here because it is also the native language of CSAF. This ensures the uncomplicated transport of CSAF documents.

The design of the API is based on Microsoft's REST API guidelines [4], but the final product is not fully compliant with them. The reason for this is that the guideline is specifically oriented towards the transport of simple and hierarchically flat resources. The filtering of deeply-nested objects, as they occur in CSAF, is not intended. Consequently, the section of the guideline, which defines how to handle collections [4, section 9], is not considered. Furthermore, the recommendations on JSON standardizations [4, section 11] are not taken from the guideline, but directly from the CSAF standard. Advanced features such as delta queries [4, section 10], long running operations [4, section 13] and push notifications via webhooks [4, section 15] are also not considered, due to not being used.

### 3.2.1 Introduction to OpenAPI

The route designs are created using the OpenAPI Standard (OAS) version 3.0.1 [20] and the online editor "Swagger Editor" [23] suitable for this purpose. The latter enables automatic syntactic checking of the OpenAPI document. The creation is done in YAML Ain't Markup Language (YAML) format, while the final product is exported to the JSON format. Writing the specification in YAML is easier because of the syntax, but the native OpenAPI format is JSON. The Swagger editor supports this approach.

The resulting API specification is intended to serve as a single source of truth and is the source for the subsequent documentation in Hypertext Markup Language (HTML), as well as for the user interface for testing the API. Likewise, the specification defines the authentication requirements for the API routes.

The specification file should later be provided by the web server adjacent to the API endpoint URL – and thus to the `provider-metadata.json` file. This approach allows a requester to know the API route versions supported by the CSAF provider or aggregator and thus helps in prevailing the requirement `DES-R1`. A URL for this could look like this, for example:

```
https://www.example.com/.well-known/csaf/swagger.json
```

Using the tool "Swagger UI" [24], a browser-based client can be created based on the specification file. The interface can also be used to access the API directly and test authentication flows. Thus, it serves not only as API documentation, but also as an interface for directly calling the API. For this purpose, the endpoints in which the API has been implemented, must be specified in the OpenAPI specification.

Since there is no production-ready API during the design phase in this thesis, the **swagger.json** file is only examined in the Swagger Editor preview or in the local browser. A suitable test server can be started for this purpose with the following command:

```
npx open-swagger-ui /path/to/swagger.json
```

The **open-swagger-ui** package is temporarily downloaded from the Node Package Manager (npm) registry using **npx**. It creates a local webserver that displays a Swagger UI based on the given API specification. This is how the API documentation was reviewed in this thesis as well as how the API was tested. If npm is not installed, the official Docker image with the local specification can also be used:

```
docker run -p 8080:8080 -v /path/to/openapi.json:/app/openapi.json \
    -e SWAGGER_JSON=/app/openapi.json swaggerapi/swagger-ui
```

The API documentation and client can then be accessed at **http://localhost:8080**.

### 3.2.2 Component definitions

The OpenAPI specification allows the definition of so-called components under the **/components** property. They can be reused later in the definition of the API paths in the **/paths** property at almost any place.

This makes it possible to define frequently occurring design elements in advance. Only their utilization (= referencing) in the **/paths** property may differ. This creates a unified interface for processing these types in API requests and responses. Referencing works similar to JSON schema type definitions. First, the **#** symbol is used to indicate that it is a reference inside the current document. This is followed by the hierarchical path to the component. A example for referencing the schema component “Response” is shown in listing 3.1.

Listing 3.1: Usage of component references in OpenAPI 3.0.1

```
{
    "$ref": "#/components/schemas/Response"
}
```

The following component types can be specified [22]:

**schemas** Allows the definition of data models as in a JSON schema. They can also be used in other component types.

**examples** Allows specification of sample data for a given schema. These are useful for users of the browser-based client generated from the specification to see prebuilt datasets without actually having to perform a request.

**parameters** Allows the definition of reusable parameters for a HTTP request. Query, header, and path parameters can be created independently of the actual route and HTTP method.

**securitySchemes** Specifies possible authentication methods that can be assigned either to individual routes or to all routes at once.

**requestBodies** Allows the definition of reusable request bodies independent of the actual route and HTTP method. For instance, schema components can be used here to define a request body.

**responses** Allows the definition of reusable responses including response bodies. Schema components can also be used here in a meaningful way. In addition, a single response can also contain multiple content-type definitions.

**headers** Enables specification of reusable request and response headers.

**links** Enables non-technical linking of values across API requests. For example, a value of a response from route `X` can be marked as a usable value for a request for route `Y`.

**callbacks** Allows the non-technical tagging of callback URLs for documentary purposes. The use case for this would be, for example, a subscription mechanism in which a custom URL can be submitted to the API to receive status updates from it. This is generally known as a webhook.

The component types `links` and `callbacks` are not utilized in this design. The same applies to `header` component type, since only the `Authorization` header is used, which is automatically set by the `securitySchemes` component.

All components defined in the design are described in the following. The complete OpenAPI specification including the components can be found in appendix A.

### Schema components

**CSAFDocument** Simple entity that is to represent a CSAF document according to the current standard. There are only the three top level properties of a CSAF document defined with the required `/document` property.

**Error** Type containing detailed API error information. An enumerable error code should be placed in the `/errcode` property. Detailed free text information is stored in the optional `/errmsg` property.

**CSAFDocumentResponse** The type returned by all API routes used to request CSAF documents. It contains a nullable `/error` property of the schema `Error`, and an array of objects which contain the plain content of the document after the `CSAFDocument` schema in the `/content` property. Optionally, the document's hash and signature can be specified in /documents[]/hash and /documents[]/signature, respectively. For more convenient processing, the length of this array is also specified.

**DeviceList** Array type containing objects that schematically resemble the property product_identification_helper of the full_product_name_t type from the CSAF standard (see section 2.2) [17, section 3.1.3.3]. This can be used to define a device for the request body in one of the API routes.

**AdvancedMatching** Type that allows complex matching of multiple JSON properties. It is intended to be used in the route where CSAF documents are to be matched based on arbitrary JSON properties. A JSON document following this scheme could look like displayed in listing 3.2.

Listing 3.2: Example of a JSON object of the AdvancedMatching schema which matches the TLP:WHITE label and searches for CVEs beginning with CVE-2018-

```
 1  {
 2    "matching_default": "exact",
 3    "operator": "and",
 4    "matches": [
 5      {
 6        "path": "$.document.distribution.tlp.label",
 7        "type": "string",
 8        "value": "WHITE"
 9      },
10      {
11        "path": "$.vulnerabilities[*]/cve",
12        "value": "CVE-2018-",
13        "matching": "begins-with",
14        "include_missing": false
15      }
16    ]
17  }
```

The /matches property contains match objects that can match the value, type, or both of a property. The /matches[]/matching and /matching_default properties specify the matching mechanism. The following values are allowed here: exact, regex, begins-with, ends-with and contains.

All match objects can be logically linked as specified in the /operator property. For example, it can be set that all match objects or only one of them must match. This is to make complex search operations possible. It can also be specified whether match objects should be evaluated to true if the specified property does not exist with the /matches[]/include_missing property. This can be useful if it is unknown whether it exists, since it may be optional.

It is also possible to search arrays by using square brackets with an asterisk (see line 11 in listing 3.2). In doing so, all properties in the array are matched. As soon as one of them matches, the match object evaluates to true.

### Example components

Example components (/components/examples) can be used to represent specific example data. These can, for example, fill a schema, but can also be used completely independently. Example components can be used wherever working with data that could also correspond to a schema, which includes response and request bodies. The defined examples can later be viewed in the browser-based client to get a better understanding of the data.

In this design, the following four example components are defined:

**CSAFDocumentResponse0Documents** Represents sample data for the CSAFDocumentResponse scheme, which returns no documents.

**CSAFDocumentResponse2Documents** Represents sample data for the CSAFDocumentResponse scheme, which returns two documents.

**CSAFDocumentResponse2DocumentsWithHashAndSignature** Same as the example component CSAFDocumentResponse2Documents, but with hash and signature for every document in /documents[]/hash and /documents[]/signature, respectively.

**DeviceListWindows10** Represents sample data for the DeviceList schema, which includes only the CPE of Microsoft Windows 10.

**DeviceListWindows10AndKeycloak** Represents sample data for the DeviceList schema, which includes the CPE of Microsoft Windows 10, as well as the CPE and an SBOM URL of the software Keycloak.

The CSAFDocumentResponse* example components are used as response in almost every route. The DeviceList* example components, on the other hand, serve as an example for the request body for the search for CSAF documents based on a device list.

### Parameter components

Request parameters can be defined under the /components/parameters property. The components defined here are called filter parameters and are intended to be used to restrict the search query in addition to the request URL path. These parameters can then be specified in any route where filtering makes sense. This applies almost exclusively to all routes that respond with CSAF documents.

The following parameters are defined.

**Before** Filters the response so that only documents initially published before the defined date are included in the response. This parameter satisfies requirement DES-R6.

**After** Filters the response so that only documents initially published after the defined date are included in the response. This parameter satisfies requirement DES-R7.

**TrackingStatus** Filters the response for documents that have the specified tracking status. This parameter satisfies requirement DES-R8.

**ValueMatching** Parameter that determines the filtering behavior of another parameter. For example, if the title of the document is to be searched for, you can specify that the title string should be matched exactly, or that it is a regex pattern. The same values used in the AdvancedMatching scheme component can be specified here, as well.

**ProductStatus** Filter which filters transmitted device lists according to whether they are affected by a specific vulnerability. The values from the VEX standard, which are also used in the CSAF profile VEX, are used for this purpose.

**Profile** Filter that only includes documents of a certain document profile (see section 2.4). It can be used, for example, to explicitly search for security advisories. The advantage of this is that it gives an assurance that certain properties exist due to being enforced by the respective profile. All values valid for the /document/category property that start with csaf_ are accepted [17, section 4]. This parameter satisfies requirement DES-R9.

**CVSSv3Range** Allows the specification of a CVSSv3 base score range. For example, values such as >8 (“score greater than 8”), >=8 (“score greater than or equal to 8”), or even ranges like >5,<9 (‘scores greater than 5 and smaller than 9‘) can be specified. Specifying this parameter only makes sense if the request contains either a device or a CVE, since a CVSS score alone has no meaningfulness. This serves the following use cases: “Are there vulnerabilities *(unknown)* that have a CVSS score of X in combination with my devices *(known)*?” and “Which of my devices *(unknown)* are affected by a CVSS score of X due to this vulnerability *(known)*?”.

**CVSSv2Range** Serves the same purpose as CVSSv3Range, but for CVSS version 2. The specification of both parameters is possible and recommended, since only one of the two values must be specified in a CSAF document [17, section 3.2.3.13].

**RemediationCategory** Allows the specification of a remediation category [17, section 3.2.3.12.1] for which at least one must exist for each specified device that is affected. This covers the following use case: “Find all CSAF documents for which there is a vendor fix, mitigation, etc. for the affected devices that were transmitted”. If the API does not deliver any documents for one of the devices, there is no corresponding remediation.

**WithHash** Instructs the API to include each CSAF document’s hashes in the response. If there are no hashes, /documents[]/hashes will be null. Otherwise, each hash will be mapped to the algorithm used. As an example, a SHA256 hash would reside in /documents[]/hashes/sha256. This parameter satisfies the requirement DES-R10.

**WithSignature** Instructs the API to include each CSAF document’s signature in the response. If there is no signature, /documents[]/signature will be null. This parameter satisfies the requirement DES-R11.

### Security scheme components

In the OpenAPI format, the authentication flow can also be specified. In the browser-based client generated from the specification, the authentication data may be specified to interact with the API.

In order for `DES-R4` to be fulfilled, a security scheme must be applied to all routes whose purpose is to return CSAF documents. For illustration purposes, the Bearer authentication scheme was chosen as the only possible variant – authentication takes place using the `Authorization:` request header in Bearer format. It is a suitable method for authentication API requests because, unlike cookie-based authentication, the token does not have to be stored in the HTTP client's memory whilst still requiring each request to be authenticated. In addition, more complex mechanisms such as OAuth or OpenID can also be permitted by the CSAF server as a non-standard extension.

### Request body components

The request body components can be used to map schema components to a specific content type. For this design, the `DeviceListRequestBody` components are defined for sending device lists and `AdvancedMatchingRequestBody` for the route that matches arbitrary JSON properties in the document.

The schema components `DeviceList` and `AdvancedMatching` are used to define the content type `application/json` for both request body components, respectively. Furthermore, the example components `DeviceListWindows10` and `DeviceListWindows10AndKeycloak` are set as examples for the `DeviceListRequestBody` component.

### Response components

With response components, schemas can be assigned to a specific content type to define a HTTP body, as in request components. It is important to note that the HTTP status code is defined by the route, not by the response component. However, they can be enriched with examples to correspond to a certain HTTP status.

Basically, response components are only defined for routes that respond with CSAF documents in this design. For all others, the response is specified inline without the help of components, since no reusability is required here.

The response components `200Ok`, `400BadRequest`, `401Unauthorized` and `500UnknownError` are defined. All of them follow the schema set in the `CSAFDocumentResponse` component. For the `200Ok` component, the two examples `CSAFDocumentResponse0Documents` and `CSAFDocumentResponse2Documents` are specified, in which the `error` property is always `null`.

The remaining components have a non-`null` error and set the corresponding error codes. A list of all error codes can be found in appendix B.

This makes it easy for a user of the API to check if an error occurred during the request. If the `/error` property is `null`, the request was successful. If not, the error code in `/error/errcode` provides a machine-readable reason for the error.

### 3.2.3 Route definitions

First, the so-called base URL for the API must be defined. This contains the schema, the host and optionally a path prefix for all routes. The route definitions are then based on this URL. Since the standard requires the use of a `/.well-known/csaf/` path for the `provider-metadata.json` file [17, section 7.1.8], the API endpoint will also follow this requirement. Accordingly, this path is part of the base URL. Since this design defines only the first version of the API, the version prefix `v1` is also part of the base URL.

This results in the following base URL, which is specified in the OpenAPI property `/servers[0]/url`: `https://example.com/.well-known/csaf/api/v1` . Of course, the host must later be replaced by the host name of the respective CSAF server. This ensures the versionability of the API and thus fulfills the requirement `DES-R1`. The definition of another API version can then be done in a new OpenAPI document with a correspondingly adapted base URL.

The current design only serves the purpose of requesting CSAF documents. Functions for uploading or modifying documents are not available, since administrative use cases are not part of this thesis. Nevertheless, a corresponding extension is conceivable and possible.

The route definitions are divided into the following categories:

- Meta queries
- Macro queries
- Arbitrary queries
- Special queries

Meta queries serve the use case of getting information about the CSAF server on which the API is running. The routes within this category are the only ones that do not respond with the `CSAFDocumentResponse` schema.

Macro queries describe routes that provide information that can also be requested via other routes, but which serve a more general use case. They are used to improve readability and simplify usability of the API by allowing only specific queries.

Arbitrary queries comprise the routes that can be used to address nearly any properties in a CSAF document. They are meant to be able to map many use cases at once. Complex queries can be made, which other routes – especially the macro routes – would not allow.

Special queries describe the routes that can retrieve CSAF documents based on external information. They are categorized separately to make it obvious that these queries may operate outside the context of the CSAF server currently in use, e.g. by requesting external sources. These routes usually require a special request body or deviating path parameters.

Both the macro queries and the special queries do not follow the Microsoft REST API guidelines in the naming approach section [4, section 17.1]. In addition to the usual addressing of the property to be processed (`/csaf-documents`), the search action (for example `/by-id`) is appended to the route. This approach is not in the sense of the guideline, however, it creates a higher clarity in the case of CSAF, as the processed properties are fairly big.

In the following, the individual routes are named. They are partially accompanied by a literal description of some of the use cases that are to be addressed by them. The possibilities that arise through the specification of filter parameters are also taken into account. Through the totality of all routes, a collection of all use cases for this design is created, which can finally be evaluated.

**[Meta queries]**
**`GET /metadata`**

**Parameters:** -

This route returns the contents of the `provider-metadata.json` or `aggregator.json` file, depending on whether the request is made to a CSAF (trusted) provider or aggregator. It thus satisfies requirement `DES-R3`. The content is transported under a separate JSON property as described in listing 3.3 to determine which of the two files is returned. Only one of the metadata files will be contained in the response, as the API always corresponds to a single CSAF instance. If the server provides both (trusted) provider and aggregator capabilites, two API endpoints must be operated. An `error` according to the `Error` scheme is also supplied.

Listing 3.3: Format of the `/metadata` route response body

```
{
  "error": null,
  "provider_metadata": {...},
  "aggregator": {...}
}
```

This route ignores authentication data of any kind.

### [Meta queries]

**`GET /role`**

**Parameters:** -

This route returns which role this CSAF server fulfills, which satisfies `DES-R2`. Possible values are `csaf_provider`, `csaf_trusted_provider` and `csaf_aggregator`. An error property according to the error scheme is also provided here. The values are returned as described in listing 3.4.

Listing 3.4: Example for the `/role` route response body returning the role of a CSAF trusted provider

```
{
  "error": null,
  "role": "csaf_trusted_provider"
}
```

This route ignores authentication data of any kind.

### [Macro queries]

**`GET /csaf-documents/by-id/{publisher_namespace}/{tracking_id}`**

**Parameters:** publisher namespace, tracking id

**Use cases:** “Find the CSAF document with the global ID supplied”

This route finds CSAF documents by the global ID consisting of the publisher namespace and the tracking ID [17, section 3.2.1.12.4]. Thus, requirement `DES-R17` is satisfied. The response always follows the `CSAFDocumentResponse` schema with either zero or one CSAF document.

In this route, the common filter parameters before, after, profile, and tracking status cannot be specified because it addresses a single document in every case.

This route optionally accepts authentication data.

**[Macro queries]**

```
GET /csaf-documents/by-title/{title}?matching=<string>
  &before=<string>&after=<string>&profile=<string>&tracking_status=<string>
```

**Parameters:** title, matching, before, after, profile, tracking status

**Use cases:** "Find the CSAF document(s) having the title specified", "Find the CSAF document(s) whose title start with the one specified"

This route finds CSAF documents by the title in `/document/title`. It can also be used to request multiple CSAF documents. The `matching` parameter allows to specify how the title should be matched (see parameter component `Matching`). Thus, CSAF documents can also be searched for whose titles contain or begin with a substring, for example, which may yield multiple documents.

The response always follows the `CSAFDocumentResponse` scheme. This route optionally accepts authentication data.

**[Macro queries]**

```
GET /csaf-documents/by-publisher/{publisher_name}?matching=<string>
  &publisher_namespace=<string>&publisher_category=<string>
  &before=<string>&after=<string>&profile=<string>&tracking_status=<string>
```

**Parameters:** publisher name, matching, publisher namespace, publisher category, before, after, profile, tracking status

**Use cases:** "Find all documents from publisher X.", "Find all security advisories from publisher X in this namespace that were published after date Y."

This route finds all CSAF documents of a specific publisher, based on the properties in `/document/publisher`. Only the publisher name must be specified. Optionally, category and namespace can also be specified. The matching parameter applies only to the publisher name, as the category is enumerated and the namespace matching is always exact.

The response always follows the `CSAFDocumentResponse` scheme. This route optionally accepts authentication data.

### [Macro queries]

```
GET /csaf-documents/by-cve/{cve}?cvssv2=<string>&cvssv3=<string>
  &before=<string>&after=<string>&profile=<string>&tracking_status=<string>
```

**Parameters:** CVE, CVSSv2, CVSSv3, before, after, profile, tracking status

**Use cases:** “Find the CSAF document(s) that contains the following CVE.”, “Which devices have a CVSS score of X due to the vulnerability Y?”

This route finds CSAF documents that contain the specified CVE. It iterates over the `/vulnerabilities[]` property of each CSAF document to determine if the specified CVE is contained. The first encounter adds the entire document to the response. Specifying the `matching` parameter is not possible in this route, since CVEs are always matched exactly.

To limit the search, the `cvssv2` and `cvssv3` parameters can be used to specify score rules that must be met at least once by the CVE. Minimum, maximum or exact values can be specified (see parameter component `CVSSv2` and `CVSSv3`). Objects in `/vulnerabilities[]` not containing the `score` property alongside the `cve` will not be included.

The response always follows the `CSAFDocumentResponse` scheme. This route optionally accepts authentication data.

### [Arbitrary queries]

```
GET /csaf-documents/match-property?path=<string>&type=<string>
  &value=<string>&matching=<string>&include_missing
  &before=<string>&after=<string>&profile=<string>&tracking_status=<string>
```

**Parameters:** path, type, value, matching, `include_missing` flag, before, after, profile, tracking status

**Use cases:** “Find all documents where the property has this value/type.”, “Find all documents where the property has this value/type, include only the profile X and include documents missing the wanted property.”

This route is used to address an arbitrary JSON property, which must occur in the searched CSAF documents in a certain form. Thus, it satisfies `DES-R16`. The value, type or both can be matched. Likewise, the non-existence of a property can be checked using the `include_missing` flag. As in the `GET /csaf-documents/by-title` route, the matching parameter can be specified, for example to require only partial matching.

Furthermore, the search can be limited by searching only certain document profiles. For this purpose, the desired profile can be specified with the `profile` parameter. This has the advantage of having a higher confidence that the searched property really exists, since it may be prescribed by the document profile.

An example for a search query to find CSAF documents in the German language with the VEX profile is listed in listing 3.5.

Listing 3.5: Example request for the `/csaf-documents/match-property` route to search for german CSAF documents with the VEX profile (not URL-encoded)

```
GET /csaf-documents/match-property
    ?path=$.document.lang&value=de&profile=vex
```

The response always follows the `CSAFDocumentResponse` scheme. This route optionally accepts authentication data.

### [Arbitrary queries]
### `POST /csaf-documents/match-properties`

**Parameters:** -

**Use cases:** “Find all documents where all of the properties have this value/type.”, “Find all documents where at least one of the properties has this value/type.”

This route also allows the matching of arbitrary JSON properties, but here multiple properties can be specified at the once. The request is made as a request body according to the schema `AdvancedMatching` with the HTTP POST method. Although this is a `GET`-like request and no data is created, the use of this method is allowed because the request body contains a search statement, called a “command” [4, section 7.4].

In this route, the common search parameters before, after, profile, and tracking status cannot be specified because they can also be manually matched in the request body.

The specified match objects can be logically linked. Thus, for example, it can be achieved that a vulnerability object can be searched for without having to know which property is actually set in it. This can be necessary when working with documents with the VEX profile. It requires that either `/vulnerabilities[]/ids` or `/vulnerabilities[]/cve` must be set [17, section 4.5]. An example query to search for either a CVE or a Github issue is shown in listing 3.6.

Listing 3.6: Example of a response body for the `/csaf-documents/match-properties` route to search for issue `#210` in the `oasis-tcs/csaf` Github repository or CVE-2018-16476 in VEX documents

```
{
  "matching_default": "exact",
  "operator": "or",
  "matches": [
    {
      "path": "$.vulnerabilites[*]/ids[*]/text",
      "value": "oasis-tcs/csaf#210"
    },
      {
        "path": "$.vulnerabilites[*]/cve",
        "value": "CVE-2018-16476"
      }
    ]
}
```

The response always follows the `CSAFDocumentResponse` scheme. This route optionally accepts authentication data.

### [Special queries]

**`POST /csaf-documents/from-device-list?product_status=<string>&cvssv3=<string>&cvssv2=<string>&remediation_category=<string>&before=<string>&after=<string>&profile=<string>&tracking_status=<string>`**

**Parameters:** product status, CVSSv2, CVSSv3, remediation category, before, after, profile, tracking status

**Use cases:** “Find all documents containing any device in the device list.“, “Find all documents containing any device in the device list, which were released after X and before Y.”, “Find all documents where any of the devices in the device list is known to be affected by any vulnerability.“, “Find all documents where any of the devices in the device list is known to be affected by any vulnerability with a CVSSv2/v3 score of at least Z.”, “Find all documents where any of the devices in the device list is known to be affected and where at least one remediation of category X exists.”

This route allows the transmission of a device list, which is used as the primary search parameter to locate all CSAF documents that have anything to do with the specified devices. It is sent via the request body of the HTTP POST method following the `DeviceList` schema, that describes an array of `product_identification_helper` objects, which are also contained within `full_product_name_t` types in CSAF documents [17, section 3.1.3] [17, section 6.2.16]. This satisfies the requirement `DES-R12`.

Matching a single property within an object in this array adds the entire CSAF document to the response. For example, for a single device, both a CPE and a serial number can be specified in the request body. If the device is identified in CSAF solely by the CPE, it will still be added to the response. This prevents a user from having to know exactly which identification helpers were specified. For the request body the example components `DeviceListWindows10` and `DeviceListWindows10AndKeycloak` are defined.

The search can be further limited using additional URL parameters. For example, the VEX status, the devices must belong to, can be set to fulfill requirement `DES-R13`. Affected devices for which no remediation of a certain category exists can also be explicitly excluded from the results. This satisfies requirement `DES-R15`. Likewise, the search can be restricted to specific document profiles. Furthermore, a CVSS score can be specified to filter for documents, where a

CVE exists, which affects the devices in the score range mentioned. This satisfies requirement DES-R14. Objects in /vulnerabilities[] that do not contain the scores property are explicitly included in the result. The reason for this is the intended use case: vulnerabilities that affect the device are to be searched for. Consequently, vulnerabilities without CVSS should not be left out of this search, since they can also potentially affect the device(s) to a similar degree.

A more complex example of the following query can be seen in listing 3.7: Find all security advisory documents containing known vulnerabilities of the software Keycloak version 1.2.0 with a CVSSv3 score of at least 8.

Listing 3.7: Example of a request to the /csaf-documents/from-device-list route to search for security advisories containing critical vulnerabilites of Keycloak 1.2.0 (not URL-encoded)

```
POST /csaf-documents/from-device-list
    ?profile=csaf_security_advisory&product_status=known_affected&cvssv3=>8

[
    {
        "cpe": "cpe:2.3:a:redhat:keycloak:1.2.0:-:*:*:*:*:*:*",
        "sbom_urls": [
            "https://raw.githubusercontent.com/CycloneDX/bom-examples/master/
               SBOM/keycloak-10.0.2/bom.json"
        ]
    }
]
```

The response always follows the CSAFDocumentResponse scheme. This route optionally accepts authentication data.

This complete route design satisfies the following remaining requirements: All routes whose purpose is to return CSAF documents accept some form of authentication to satisfy requirement DES-R4. Similarly, filter parameters can be arbitrarily set according to requirement DES-R5 as long as it is applicable and reasonable. Each response contains an optional error property independent of the status code, thus satisfying requirement DES-R18. In routes that respond with CSAF documents, these are always delivered as a whole to satisfy requirement DES-R19.

Ultimately, the route design fulfills all design requirements.

## 3.3 Design feasibility

In order to check the feasibility of the route design, the following methodology is adopted in this section: First, it is worked out which properties are accessed by each individual route when all filter parameters are set. This should show which problems can arise, for example, due to the optionality of individual properties or other limitations. This section also serves as an orientation for the implementation.

For each property, the optionality is stated. According to the standard, the following enforcement types are possible, sorted by effectiveness:

- **Required by schema**: The property is enforced by the CSAF JSON schema and is guaranteed to exist.
- **Required by mandatory test**: The property is enforced by a mandatory test [17, section 6.1] and is guaranteed to exist.
- **Required by profile X**: The property is enforced by the document profile X and is therefore present in documents of this profile. Setting the `profile` filter parameter is recommended.
- **Required by optional test**: The existence of the property can be assumed, since optional tests are displayed as a warning [17, section 6.2].
- **Optional**: The existence of the property cannot be ensured. Searches based on optional properties may be unreliable.

### 3.3.1 Regularly used parameters

First, the reliability of the regularly used parameters is examined. In the following, the relevant CSAF document paths and corresponding enforcement type are listed for each of these parameters.

**`before / after`**
`/document/tracking/initial_release_date` (required by schema)

**`profile`**
`/document/category` (required by schema)

**`tracking_status`**
`/document/tracking/status` (required by schema)

**`with_hash` / `with_signature`**

These parameters do not access the contents of the CSAF documents. However, an attempt is made to read the hash and signature files of these. They are located on the file system in the same place, but with an additional suitable file suffix `.sha256`, `.sha512` for hash files and `.asc` for ASCII-armored PGP signatures [17, sections 7.1.18, 7.1.19]. The presence of these files is mandatory for CSAF trusted providers. Aggregators only have to copy already existing signatures and hashes. If they are not available on the mirrored server, the aggregator must create them itself. For normal CSAF providers and CSAF publishers, specifying `with_hash` and `with_signature` will return `null` in the corresponding response properties (see schema component `CSAFDocumentResponse`). However, this is not a major problem, since this response indicates the obvious non-existence of the signature or hash.

In conclusion, it can be said that all regularly used parameters are based on properties that are required by the document schema or do not access the document contents at all. Thus, they are reliable and their usage is deterministic.

### 3.3.2 Routes

Next, the individual routes will be examined. Both the required parameters and the route-specific, optional parameters are considered. Route-specific parameters are all additional parameters that are not already covered by the regularly used parameters mentioned in the previous section.

In this route evaluation, both path and query parameters are omitted from the route name for better readability. Only the method and path are specified.

#### `GET /metadata`

This route is necessary because it is the only measure to implement requirement `DES-R3`. It requires read access to the respective file in the file system of the web server, which is basically given. No CSAF documents need to be read or otherwise processed.

#### `GET /role`

This route is the only measure to implement requirement `DES-R2`. It does not require any access to the file system. The response is composed exclusively of runtime variables.

**`GET /csaf-documents/by-id`**

Relevant properties without specifying parameters:

- `/document/tracking/id` (required by schema)
- `/document/publisher/namespace` (required by schema)

Additional relevant properties with specification of route-specific parameters:

- none

This route requires only the specification of the mentioned path parameters. Both are enforced by the CSAF JSON schema. Consequently, this route can be implemented reliably.

**`GET /csaf-documents/by-title`**

Relevant properties without specifying parameters:

- `/document/title` (required by schema)

Additional relevant properties with specification of route-specific parameters:

- none

The additional `matching` parameter does not access or read any document property. This route only relies on properties enforced by the CSAF JSON schema. It is unlikely that this level of enforcement in going to change in future versions of CSAF. Thus, this route can be implemented reliably.

**`GET /csaf-documents/by-publisher`**

Relevant properties without specifying parameters:

- `/document/publisher/name` (required by schema)

Additional relevant properties with specification of route-specific parameters:

- `/document/publisher/namespace` (`publisher_namespace` parameter, required by schema)
- `/document/publisher/category` (`publisher_category` parameter, required by schema)

The additional `matching` parameter does not access or read any document property. This route only relies on properties enforced by the CSAF JSON schema, like the `GET /csaf-documents/by-title` route. Furthermore, the use of the route is flexible, since only the publisher name has to be specified, which allows a more "open" search. If the search is to be restricted to a unique publisher, the namespace and category can be specified optionally, which are also enforced by the schema. This route can be implemented reliably.

#### `GET /csaf-documents/by-cve`

Relevant properties without specifying parameters:

- `/vulnerabilities[]/cve` (partially required by mandatory test and profile VEX, checked by informative test)

Additional relevant properties with specification of route-specific parameters:

- `/vulnerabilities[]/scores[]/cvss_v3/baseScore` (`cvssv3` parameter, optional)
- `/vulnerabilities[]/scores[]/cvss_v2/baseScore` (`cvssv2` parameter, optional)

The specification of the CVE in a CSAF document is basically optional. There is a mandatory test which checks whether either a CVE or an ID exists for each vulnerability object [17, section 6.1.27.8]. The ID allows the unique identification of newly discovered vulnerabilities that have not yet been assigned a CVE. It can therefore be assumed that operators of CSAF providers also include a CVE in the vulnerabilities object as soon as one has been assigned to it. The search for a CVE presupposes the existence of it and corresponds to the use case of the route. Although the mandatory test is only applied to documents with the VEX profile, the assignment of a CVE is also advised in other profiles if `/vulnerabilities[]` contains at least one object. As long as publishers and providers adhere to this, this route can also be implemented reliably.

The procedure described in listing 3.8 shall apply when calculating the response for each CSAF document. Filtering using the regularly used parameters `before`, `after`, `profile` and `tracking_status` are omitted in the code example.

Listing 3.8: Pseudocode for the `GET /csaf-documents/by-cve` route implementation (excluding regularly used parameters)

```
for vuln_object in "/vulnerabilties[]"
    if "cve" in vuln_object and "cve" is searched_cve
        if searched_cvss_v3 or searched_cvss_v2
            # at least one cvss score specified
            if searched_cvss_v3
                if "scores/cvssv3" in vuln_object
                    if "scores/cvssv3/baseScore" is seached_cvss_v3
                        add_to_response(document)
                        continue
            if searched_cvss_v2
                if vuln_object has "scores/cvssv2"
                    if "scores/cvssv2/baseScore" is searched_cvss_v2
                        add_to_response(document)
                        continue
            continue
            # document was not added to response
        else
            # cve matches and cvss not specified
            add_to_response(document)
            continue
    else
        # cve does not match
        continue
```

### `GET /csaf-documents/match-property`

This route is redundant. Its function can be completely replaced by `POST /csaf-documents/match-properties`, but the call is simpler since no request body is required. It can access arbitrary properties. Consequently, the general feasibility cannot be evaluated and depends entirely on the usage.

### `POST /csaf-documents/match-properties`

This route serves as a general purpose route. Theoretically, it can be used to imitate the functions of the other routes and to implement more specific use cases. Here it is important that the user determines the feasability of the route based on his own request. The general feasability cannot be determined for the same reasons as for the previous route.

### `POST /csaf-documents/from-device-list`

Relevant properties without specifying parameters:

- `/product_tree/branches[](/branches[])*/product/product_identification_helper` (required by optional test)
- `/product_tree/full_product_names[]/product_identification_helper` (required by optional test)
- `/product_tree/relationships[]/full_product_name/product_identification_helper` (required by optional test)

Additional relevant properties with specification of route-specific parameters:

- `/vulnerabilities[]/scores[]/cvss_v3/baseScore` (`cvssv3` parameter, optional)
- `/vulnerabilities[]/scores[]/cvss_v2/baseScore` (`cvssv2` parameter, optional)

Using the `product_identification_helper` property may be unreliable here, since it is only required by optional tests. Documents that lack this property are not considered invalid. At this point, a standard adjustment would be necessary to enforce the property, for example, by a mandatory test or document profile specification. This will be tracked as `DES-R12-A`: "Require the specification of a product identification helper in every product.". The only question is whether this constraint can be implemented at all, since there may be an edge case where a product does not have a product identification helper.

The product identification helpers that match an object of type `full_product_name_t` are resolved to the product ID specified in it. The value of the `product_status` parameter is then used to search for this product ID in the corresponding property, which is required by the Security Advisory and VEX profiles. For the Informational Advisory profile, however, the `/vulnerabilities` property must not exist – all products in `/product_tree` must be assumed to be affected [17, section 4.3]. So in this case the following exception arises: If `product_status` has the value `known_affected` and the search hits a CSAF document of the Informational Advisory profile, all defined products are treated as affected without considering the `/vulnerabilities[]/product_status/known_affected` property. On all other document profiles, this parameter should have no effect. They will be skipped if this parameter is specified.

With regard to the CVSS scores, similar rules apply as for the `GET /csaf-documents/by-cve` route. However, the existence of the associated properties `/vulnerabilities[]/scores[]/cvss_v3` and `/vulnerabilities[]/scores[]/cvss_v2` is optional as stated in the route definition in section 3.2.3.

If the `remediation_category` parameter is set, it is checked if a remediation of the specified category exists within the same vulnerability object for each product in `/vulnerabilities[]/product_status/known_affected`. In some circumstances, the use of the `remediation_category` parameter is not possible. For example, if the `product_status` parameter has the value `known_not_affected`, there are no remediations. If then `remediation_category` is set to the value `mitigation`, a conflict arises and no CSAF documents are returned in any case. An example for such a faulty request can be seen in listing 3.9. Here, the implementation of a suitable error response is advised.

Listing 3.9: Faulty request to the device list endpoint caused by conflicting query paramter values (ommited request body)

```
POST /csaf-documents/from-device-list
    ?product_status=known_not_affected&remediation_category=mitigation
```

The procedure described in listing 3.10 shall apply when calculating the response for each CSAF document. Filtering using the regularly used parameters `before`, `after`, `profile` and `tracking_status` are omitted in the code example.

Listing 3.10: Pseudocode for the `POST /csaf-documents/from-device-list` route implementation (excluding regularly used parameters)

```
product_tree_object = document.product_tree
products = find_all_full_product_names_recursively(product_tree_object)

for product in products:

    matched = False
    for searched_product in searched_products:
        # compare every property of product with searched_product
        if any_prop_matches(product, searched_product):
            matched = True
            break
    if not matched:
        # do not add document to response
        continue

    vuln_objects = find_vuln_objects_with_product(product)
    # filter vuln_objects depending on parameters set in the request
    to_remove = []
    for vuln_object in vuln_objects:
        if is_set(product_status):
            if product not in vuln_object.product_status[product_status]:
                to_remove.append(vuln_object)
                continue
        if is_set(remediation_category):
            contains_matching_remediation = False
            for remediation in vuln_object.remediations:
                if remediation.category is remediation_category and product in remediation.product_ids:
                    contains_matching_remediation = True
                    break
            if not contains_matching_remediation:
                to_remove.append(vuln_object)
                continue
        if is_set(cvssv3) or is_set(cvssv2):
            cvss_matched = False
            for score in vuln_object.scores:
                if product in score.products:
                    if is_set(cvssv3) and score.cvss_v3:
                        if in_range(score.cvss_v3.baseScore, cvssv3):
                            cvss_matched = True
                            break
                            # cvss_v3 already matched, ignoring cvss_v2
                    if is_set(cvssv2) and score.cvss_v2:
                        if in_range(score.cvss_v2.baseScore, cvssv2):
                            cvss_matched = True
                if not cvss_matched:
                    to_remove.append(vuln_object)
                    continue
        vuln_objects.remove(to_remove)

        if len(vuln_objects) > 0:
            # at least one matched vuln_objects left after filtering
            # —> add document to response
            add_to_response(document)
            continue
```

With the current design of this route, products that do not have a product identification helper cannot be matched and thus cannot be found.

A possible alternative for solving this problem would be to specify the product names instead of the properties in `product_identification_helper` in the `POST` request body. However, this would result in further problems and challenges. The way versions and product names are specified by CSAF publishers or providers is unpredictable. This means that all version schemes available must be compatible with each other in order to be comparable and thus allow a reliable usage of this route. Similarly, string matching of product names must not be too strict, otherwise minor input errors such as case-sensitivity can lead to unexpected errors. In this case, fuzzy search algorithms would be unavoidable. Regular expressions would also be comparatively inappropriate for such a search, since this algorithm also partially requires knowledge of the search target – and especially its format.

A compromise would be to specify the product name and the version separately. This way the product name can be found using fuzzy search. The associated `branches_t` objects with category `product_version` or `product_version_range` can be found using the version specification variants. The CSAF standard currently proposes two variants for this purpose: Version Range Specifier (vers) and vers-like Specifier (vls), with the former being strongly recommended [17, section 3.1.2.3.2]. vers is a community project tasked with parsing and comparing versions and version ranges across packaged ecosystems such as PyPi and npm. A concrete implementation of vers is “univers” [11].

In this thesis, however, such a route has not been implemented, since CPEs as well as PURLs can be defined in the product identification helper, which can also contain a version number themselves. It is assumed that only documents in which all products contain a product identification helper are worked with.

The following standard adjustments were defined in this section while evaluating the design feasability:

Table 3.5: Design specification adjustments

| **Design specification adjustments** | |
|---|---|
| Naming | Description |
| `DES-R12-A` | Require the specification of a product identification helper in every product. |

# 4 Proof of Concept

## 4.1 Testing environment

A test infrastructure fulfills the purpose of being able to test and implement the API in a running CSAF environment. It should integrate seamlessly with the function of CSAF (trusted) providers and aggregators. In addition, a certain number of CSAF documents must be available as a data source.

The infrastructure was assembled from the components of the official CSAF PoC [25]. To ensure a disruption-free and at the same time traceable environment, a fork of the Git repository was created at the commit hash `006f088082f615dfd975e024a7da9869f4fac2b4`. This ensures that any breaking changes in the upstream repository that may occur while working on the infrastructure will not interfere with the original repo. The fork is located at `https://github.com/MexHigh/csaf_distribution` on the `dev-api` branch.

The environment consists of a trusted provider and an aggregator that mirrors the provider. In general, the operation of an aggregator which mirrors only one other instance is not allowed [17, section 7.1.22]. However, the PoC implementation also allows operation with only one instance for testing purposes.

The entire infrastructure is implemented on Virtual Machines (VMs) using VirtualBox to simulate an IP network on a single computer. The network is set up using VirtualBox's Network Address Translation (NAT) driver in a `10.0.2.0/24` subnet. A Caddy web server [3] is run in a dedicated VM, which acts as a reverse proxy. It takes care of Transport Layer Security (TLS) termination with certificates from a local Certificate Authority (CA) and forwards the requests to the respective CSAF component based on the Domain Name System (DNS) name using Server Name Indication (SNI). Thus, the endpoints `https://provider-1.csaf` and `https://aggregator-1.csaf` [(i)] are provided. The HTTPS endpoint of the web server and the Secure Shell (SSH) services of the VM are exposed to the VirtualBox host via port forwarding.

The trusted provider was set up using the guide in the `README.md` file in the repository [25]. Fulfilling the standard requirement 10 to locate the `provider-metadata.json` via a well-defined DNS hostname [17, section 7.1.10] was skipped, as it is not relevant to the implementation of the API. The aggregator was also set up according to this guide. However,

[(i)] These hostnames are used for local testing only. `.csaf` is not an officially approved top-level-domain!

periodically querying the data to be mirrored via the crontab was not necessary – the mirroring process was manually executed when changes were made to the trusted provider.

In the course of the implementation, the `provider-metadata.json` file must be adapted. Since this would then no longer conform to the provider JSON schema, all validation measures within the aggregator code have been temporarily commented out. The changes can be viewed and revised via GitHubs compare feature: `https://github.com/csaf-poc/csaf_distribution/compare/006f088082f615dfd975e024a7da9869f4fac2b4...MexHigh:csaf_distribution:dev-api`.

## 4.2 Test data set

The provider was partially filled with slightly modified documents copied from the Siemens CSAF endpoint at `https://cert-portal.siemens.com/productcert/csaf/` using the `csaf_downloader` [25]. Only CSAF documents from the `TLP:WHITE` feed and that were released in 2022 were used, resulting in a total of 70 documents. In each document, the `/document/publisher` field was replaced with that of the test provider. Checksums and signatures were created by the test provider itself.

Of the `TLP:WHITE` documents, five were selected to serve as copies in a `TLP:RED` feed as sample data. For this purpose, the value of the `/document/distribution/tlp/label` field was changed to `"RED"`. It is intended that two documents with the same tracking ID exist as a result. This makes it easier to test certain routes, even if this does not represent a real scenario. All documents were uploaded with the `csaf_uploader` [25].

Since only a few product identification helpers were found in the Siemens CSAF documents, some more were added for testing the `POST /csaf-documents/from-device-list` route. For this, the first document `ssa-111512.json` of the `TLP:WHITE` feed was modified. The following products were assigned these randomly chosen CPEs:

- Product with ID 1:
  `cpe:2.3:a:ntp:ntp:4.2.8:p3:*:*:*:*:*:*`
- Product with ID 2:
  `cpe:2.3:o:microsoft:windows_7:-:sp2:*:*:*:*:*:*`
- Product with ID 3:
  `cpe:2.3:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*`

In addition, the remaining properties in the product identification helper were set for the product with product ID 1. The resulting `product_identification_helper` object is shown in listing 4.1.

Listing 4.1: Complete product identification helper object excluding `x-generic-uris` for the test data set

```
"product_identification_helper": {
    "cpe": "cpe:2.3:a:ntp:ntp:4.2.8:p3:*:*:*:*:*:*",
    "hashes": [{
        "filename": "ntp.exe",
        "file_hashes": [
            {
                "algorithm": "sha256",
                "value": "c8133b84d1c5cc5ced08..."
            }
        ]
    }],
    "model_numbers": [
        "1234",
        "5678"
    ],
    "purl": "pkg:github/package-url/purl-spec@244fd47e07d1004f0aed9c",
    "sbom_urls": [
        "https://example.com/sbom.json"
    ],
    "serial_numbers": [
        "1234",
        "5678"
    ],
    "skus": [
        "UGG-BB-PUR-07"
    ]
}
```

## 4.3 Implementation

The PoC repository uses the module system introduced in Go 1.11. The root of the repository is also the root of the Go module. In the file `go.mod` all dependencies are listed. The folder `csaf/` forms the library, which provides the functions and constants for all CSAF components. The `cmd/` folder contains the packages for each component. An independent executable file is created from each of these packages when the Go module is compiled.

The API is implemented as a separate component package in `cmd/csaf_api/`. Thus, a large part of the implementation takes place in its own segregated area of the repository.

The intent of this implementation is to add as few dependencies to the module as possible and to use the ones already included in a prioritized way. For the API development, however, a routing framework is required to facilitate the implementation and readability of the code. For this, Gorilla Mux [9] is used, as its way of designing HTTP handlers is very close to Go's standard library `net/http`. This makes the code easier to understand for users who have no knowledge of specific routing frameworks. The dependency of Gorilla Mux was added to the `go.mod` file.

### 4.3.1 CSAF document management

At the time of the fork, there was no data structure for deserialized CSAF documents. The content of them was always stored in the `interface` datatype, because access to the contained properties was rarely required. For small queries to document properties, the gval library [12] with JSONPath extension [13] was used. However, since the API needs to access the properties frequently and also wants to make use of Go's type-safety, a Go model had to be made for it, which facilitates the filtering of CSAF documents.

First, an attempt was made to automatically generate a Go model using the official JSON schema. After several failed attempts, the tool `github.com/atombender/go-jsonschema` achieved satisfactory results after some adjustments. This resulted in the `CsafJson` data type, which, as seen in listing 4.2, finds application in the `documents` field of the `CSAFDocumentCollection`. `CsafJson` was added to the `csaf/` package.

In order to fulfill the implementation requirements `IMP-R1` and `IMP-R2` defined in section 3.1, a suitable data structure was developed. It is responsible for finding, loading, filtering and reading CSAF documents and is accordingly named `CSAFDocumentCollection`. The schema of the data structure can be seen in listing 4.2. It is implemented in the `csaf/` package as well and should be used in all API routes that process CSAF documents.

Listing 4.2: `CSAFDocumentCollection` struct responsible for finding, loading, filtering and reading CSAF documents with function signatures

```go
// CSAFDocumentWrapper contains the actual document payload
// and it's hashes, signatures and a path relative to the
// CSAFDocumentCollections basePath.
type CSAFDocumentWrapper struct {
	Path      string
	Document  *CsafJson
	Hashes    *map[string]string
	Signature *string
}

// CSAFDocumentCollection holds all CSAF documents for the
// provider or aggregator and provides methods to interact
// with them.
type CSAFDocumentCollection struct {
	documents []CSAFDocumentWrapper
	filters   []func(doc *CsafJson) (bool, error)
}

```

```
// AddFilterFunc adds at least one filter function to the collection object
// without executing it. It can be called multiple times or with more than
// one function at once.
//
// To execute all filter functions, call StartFiltering().
func (dc *CSAFDocumentCollection) AddFilterFunc(f ...func(doc *CsafJson) (bool, error))

// ClearFilterFuncs removes all filter functions. When calling
// StartFiltering(), the filter functions get cleared
// automatically afterwards.
func (dc *CSAFDocumentCollection) ClearFilterFuncs()

// StartFiltering executes all filter functions registered by
// AddFilterFunc() and returns the result. Afterwards, the
// registered filter functions are deleted.
func (dc *CSAFDocumentCollection) StartFiltering(verbose bool) ([]CSAFDocumentWrapper, error)

// NewCSAFDocumentCollection walks the basePath directory recursively to
// gather all CSAF documents within it and returns a new
// CSAFDocumentCollection instance.
func NewCSAFDocumentCollection(basePath string, verbose bool) (*CSAFDocumentCollection, error)
```

The function `NewCSAFDocumentCollection` recursively traverses the `basePath` specified in the configuration. All JSON documents that match the schema are transferred as CSAF documents to the generated Go model. The hash and signature files, if any, are also loaded.

With `AddFilterFunc`, filters can be added in the form of the specified function signature. The filter function receives the filled CSAF document model as a function parameter for each loaded document, which can then be examined in there. The boolean return value is used to instruct the filtering engine whether the document should be included in the response or not. Any number of filter functions can be added. All given functions must return true for this document to be included in the response. With the `StartFiltering` function, all previously specified filters are executed, and the filter result is returned. Afterwards, the previously defined filters are removed again, as it would also be possible manually via the `ClearFilterFuncs` function.

Due to the way CSAF documents are loaded from the file system, the problem mentioned in sections 2.5 and 3.1 regarding the operation of the API on an aggregator is no longer applicable. The structure of how the documents are stored there on the file system is fully compatible with the loading algorithm of the `CSAFDocumentCollection`. Thus, it is possible to operate a single API endpoint on an aggregator, which mirrors several providers, as initially intended. The differentiation of the providers takes place thereby exclusively with the help of the properties in `/document/publisher` in each individual document.

### 4.3.2 Configuration

To follow the practice of the other CSAF components, the configuration of the API is done in its own Tom's Obvious, Minimal Language (TOML) file `api.toml`. The path to the file is passed to the binary via the `-c` / `--config` flag. The TOML library `github.com/BurntSushi/toml` already used in the other components is used to deserialize the file. The associated data structures are listed and explained in listing 4.3. A suitable example configuration can be seen in listing 4.4.

Listing 4.3: `Config` struct used as unmarshalling target for the configuration file `api.toml`

```go
type AuthData struct {
    // The token without the "Bearer" part
    Token string ‘toml:"token"‘
    // Slice containing all TLP labels this token has clearence for
    // (TLP:WHITE is always implicitly included)
    AllowedTLPLabels []csaf.TLPLabel ‘toml:"allowed_tlp_labels"‘
}

type Config struct {
    // Whether to print verbose logs
    Verbose bool ‘toml:"verbose"‘ // default: false (implicit)
    // The address with port, the API should listen on
    BindAddress string ‘toml:"bind_address"‘ // default: 0.0.0.0:8080
    // The path, where all CSAF documents reside in
    // (see ’web’ provider option (https://github.com/csaf-poc/csaf_distribution/blob/main/docs/csaf_provider.md))
    CSAFDocumentsPath string ‘toml:"csaf_documents_path"‘ // default: /var/www
    // Slice containing tokens that can be used to request
    // TLP:GREEN, TLP:AMBER or TLP:RED documents
    Auth []AuthData ‘toml:"auth"‘
    // Defines, in which CSAF component the API is used in
    UsedIn csaf.MetadataRole ‘toml:"used_in"‘
}
```

Listing 4.4: Example configuation for the CSAF API in `api.toml` on a trusted provider with two valid authentication tokens

```toml
verbose = true
bind_address = "0.0.0.0:8081"
used_in = "csaf_trusted_provider"
csaf_documents_path = "/var/www"

[[auth]]
token = "abc123"
allowed_tlp_labels = ["GREEN"]

[[auth]]
token = "def456"
allowed_tlp_labels = ["GREEN", "AMBER", "RED"]
```

A validation mechanism for the configuration file was not implemented in the proof of concept.

### 4.3.3 Implementation of the authentication middleware

The used routing framework Gorilla Mux supports the implementation of middleware [9, section “Middleware”]. Middleware is used to modify incoming requests before they reach the appropriate route implementation. A schematic of this process is shown in figure 4.1. It can be chained and is executed in the order they were added to the router object.

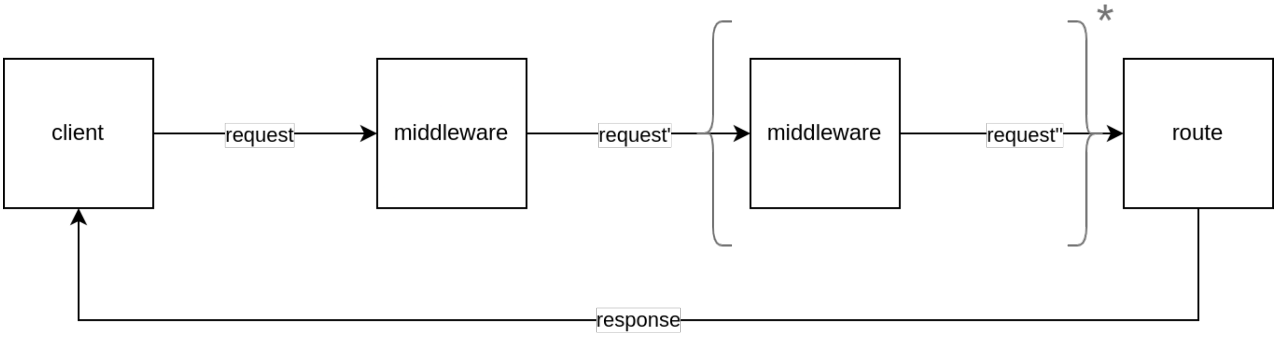


Figure 4.1: HTTP request and response handling process with middleware and routing. A client’s request can be intercepted by any number of middleware handlers.

In Gorilla Mux middleware can be implemented using Go functions. Such a function receives the next handler in the chain as parameter, which again can be a middleware or the route implementation. A handler must be returned, which receives the response writer and a pointer to the original request object as parameters. This function to be returned can now modify the request object as desired before it is passed on to the next handler.

The authentication mechanism prescribed by the requirements `IMP-R3`, `IMP-R4` and `IMP-R5` is implemented as middleware. It reads the `Authorization` request header and compares it to the tokens specified in the configuration file. If the token matches, the respective permissions for the TLP labels are granted. The following cases are specifically covered by the middleware:

- `Authorization` header is empty or not present: authorization for `TLP:WHITE` is granted (satisfies requirement `IMP-R3`)
- `Authorization` header holds a valid token: authorization for `TLP:WHITE` and all additionally specified labels is granted (satisfies requirement `IMP-R4`)
- `Authorization` header contains an invalid token: Middleware chain is interrupted with error `401 Unauthorized` – no authorization is granted (satisfies requirement `IMP-R5`)

As mentioned in section 2.3.1, the middleware will sort out CSAF documents without a TLP label – a TLP:RED label will be assumed. This can lead to unpredictable errors, since important documents may be excluded from queries even though they are actually intended for the public. To address this problem, TLP labels should always be considered. This is tracked in the IMP-R3-A and IMP-R4-A standard adaptations: "The specification of the /document/distribution/tlp/label should be enforced by the JSON schema or a mandatory test."

The permissions calculated in the middleware have to find their way to the route implementation. For this purpose, Go's context API is used. Among other things, it can be used to transport request-scoped values across API boundaries [6]. Each request object has a context by default, to which the permission values are added and passed to the next handler. In the route implementation, the context permission values can be retrieved using the getContextVars helper function. It can then be used for document filtering.

The authentication middleware is used for all routes under the /csaf-documents namespace.

### 4.3.4 Route implementation

The meta routes do not reside under the /csaf-documents namespace. Consequently, they do not follow the same response scheme. The GET /metadata and GET /role routes each define their own scheme here, as defined in the OpenAPI specification. Both route implementations depend on the mandatory used_in configuration parameter. The GET /role returns it directly, while GET /metadata uses it to determine whether to return the contents of the provider-metadata.json or aggregator.json file. The respective files, unlike the CSAF documents, are read at route execution, not beforehand. They are also validated using the validation functions from the csaf/ package. The paths used by the csaf_provider, or csaf_aggregator are used to locate the files.

If the value of used_in is not correct or cannot be read for other reasons, the error code ROLE_UNDEFINED is returned.

The implementation of the meta routes was uncomplicated.

The implementation of the macro routes was also largely unproblematic. Only for the GET /csaf-documents/by-cve route the implementation of the CVSS score range parameters became complex. For the comparison of the CVSS ranges, gval was used with the languages Basic and Arithmetic to allow for statements like cvss > 8.0 or cvss != 6.5. Go's typing system also complicated the parsing of scores, since some of them are deeply nested in the CSAF standard. This is also evident from the pseudocode in listing 3.8. Likewise, only the CVSS base scores can be handled so far – for more complex information matching like environmental score or the attack vector there is no implementation yet.

In general, the approach of filtering via the `CSAFDocumentCollection` makes sense and could also be implemented almost uniformly in each route. Finally, the implementation was successful for this route group.

During the implementation of the arbitrary `GET /csaf-documents/match-property` and `POST /csaf-documents/match-properties` routes, some difficulties arose due to Go's type-safety. The following rules apply to the parameterization of the routes:

- `path` is required
- either `type`, `value` or both must be specified

In the `type` parameter, the JSON type must be specified as a string, not the Go type. If only `path` and `type` were specified, the Go type must be evaluated via type assertion and by using the `reflect` package from the standard library. A mapping from Go type to JSON type had to be created. This problem can possibly also affect other strongly-typed programming languages and make implementation more difficult.

It was also necessary to decide in which syntax the path should be specified. In the other CSAF components and the `csaf/` package, gval with the JSONPath language was used. This was adopted for both the `GET /csaf-documents/match-property` and `POST /csaf-documents/match-properties` route. At this point the problem arose, that the JSONPath language is actually responsible for the request of paths in JSON documents, as well as for the matching of the JSON values. Accordingly, it can also be used to address arrays or whole objects. However, the design of this route only allows it to handle primitive JSON types, such as strings or numbers. Matching is then handled by the route, not by JSONPath itself. Accordingly, value matching with an array or object cannot take place. Also, with the current approach, it is not possible to search for a specific property in an array of objects. This becomes necessary, for example, when searching for a CVE in the `/vulnerabilities` array. The result of such a JSONPath query, as shown in listing 4.5, would in any case be an array that could not be mapped using either the `?value=` query parameter or the current `AdvancedMatching` scheme. To cover these cases as well, JSONPath must be fully available to the API user. In this design, JSONPath is just used as a path specification. Addressing directly accessible properties like `/document/title`, however, works without any problems.

Listing 4.5: JSONPath query to search for `CVE-2022-30190` in `/vulnerabilities[]`

```
$.vulnerabilities[? @.cve=="CVE-2022-30190"]
```

Finally, the arbitrary routes can only partially fulfill their purpose.

The implementation of the special query `POST /from-device-list` required the implementation of many auxiliary functions, but could be implemented comparatively effortlessly. The following procedure for filtering the documents takes place:

1. Adding the TLP filter from the authentication middleware.
2. Adding the `before`, `after`, `profile` and `tracking_status` filters.
3. Recursively find all objects of type `full_product_name_t` in the CSAF document.
4. Check if a product identification helper exists in each of these objects.
5. If yes, check for each defined product in the request body whether at least one of the specified properties matches one in the product identification helper.
6. If the `product_status` parameter is set, find all objects in `/vulnerabilities[]` in which the matched product is listed with the given status. Otherwise, find all objects in `/vulnerabilities[]` in which the matched product occurs at all.
7. Filter the objects from `/vulnerabilities[]` using the `score` property, as performed in the `GET /csaf-documents/by-cve` route.
8. If the `remediation_category` parameter is specified, check if this category occurs at least once in the remaining objects from `/vulnerabilities[]`. If not, discard the object.
9. If at least one of the objects from `/vulnerabilities[]` remains after filtering, add the entire CSAF document to the response.

Problems occurred especially in step five. Matching the properties proved to be non-trivial, since some of the product identification helpers formed deep nestings. An example of this is the `hashes` property. It is an array containing objects whose properties partially contain arrays with objects again. This requires deep nesting of `for` loops to parse due to Go's type-system.

Matching one-dimensional arrays in the product identification helper, on the other hand, proved to be very helpful. Specifying a single entry in this array would be sufficient to match the product. For example, products with multiple model numbers could be matched by specifying only one of them in the request body. The accuracy of the device list increases accordingly.

In the end, however, this route could also be fully implemented.

### 4.3.5 API operation

Unlike the other CSAF components, the `csaf_api` binary is a long-running task. The `csaf_provider` acts as a Common Gateway Interface (CGI) process whose lifetime is only as long as the HTTP request and response. The `csaf_aggregator` is normally called via a regular cronjob. The `csaf_api`, on the other hand, must listen permanently on the port for HTTP requests. Theoretically, a CGI implementation of the API would also be possible, but this would result in CSAF documents being read on every request. In high-traffic environments, the load on the file system would therefore be very high. Thus, the operation of the API process is not possible in stateless environments, such as on WordPress or with simple static web hosting.

The `csaf_api` component does not support TLS encryption of HTTP traffic. Accordingly, it relies on a TLS-terminating reverse proxy. In this way, the API is also included in the path of the CSAF server host, as described in listing 4.6 for nginx as an example. In addition, the `Access-Control-Allow-Origin` response header must be set to the value  to avoid blocking cross-origin access. This is especially necessary if a web application, for example a graphical frontend for CSAF, wants to include the API. In this case, cross-origin requests are issued by the application. When testing the API with the web client generated from the OpenAPI specification, this is also necessary.

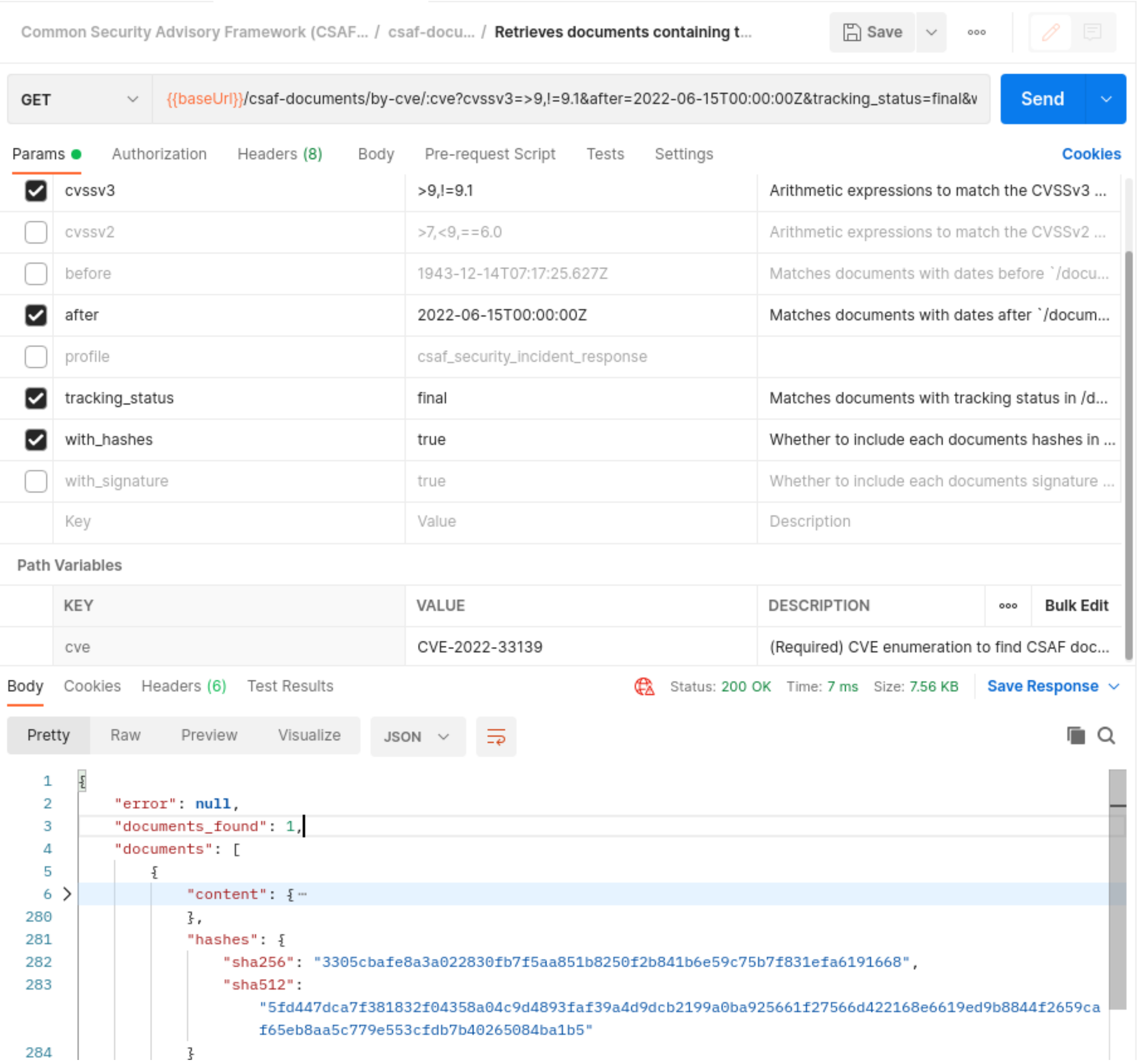
Listing 4.6: Example configuration for nginx on how to mount the API reverse proxy to a path

```
server {
    ...
    location /.well-known/csaf/api/ {
        add_header Access-Control-Allow-Origin "*";
        proxy_pass http://localhost:8081/;
    }
    ...
}
```

During the implementation, all implementation requirements could be covered. However, due to the matching difficulties in the arbitrary routes, the purpose behind the design requirement `DES-R16` could not be fully satisfied.

## 4.4 Testing and user interface

The OpenAPI format, in which the API was specified, is a well-known standard that is supported by many applications. For extensive testing of the API, popular testing tools such as SoapUI / ReadyAPI [19] or Postman (see figure 4.2) can be used. Likewise, as mentioned in section 3.2.1, the Swagger UI client [24] can be used for testing. All of them also provide a graphical interface.



Common Security Advisory Framework (CSAF... / csaf-docu... / Retrieves documents containing t...
Save
GET
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies

{{baseUrl}}/csaf-documents/by-cve/:cve?cvssv3=>9,!=9.1&after=2022-06-15T00:00:00Z&tracking_status=final&w

| | Key | Value | Description |
|---|---|---|---|
| ☑ | cvssv3 | >9,!=9.1 | Arithmetic expressions to match the CVSSv3 ... |
| ☐ | cvssv2 | >7,<9,==6.0 | Arithmetic expressions to match the CVSSv2 ... |
| ☐ | before | 1943-12-14T07:17:25.627Z | Matches documents with dates before `/docu... |
| ☑ | after | 2022-06-15T00:00:00Z | Matches documents with dates after `/docum... |
| ☐ | profile | csaf_security_incident_response | |
| ☑ | tracking_status | final | Matches documents with tracking status in /d... |
| ☑ | with_hashes | true | Whether to include each documents hashes in ... |
| ☐ | with_signature | true | Whether to include each documents signature ... |
| | Key | Value | Description |

Path Variables

| KEY | VALUE | DESCRIPTION |
|---|---|---|
| cve | CVE-2022-33139 | (Required) CVE enumeration to find CSAF doc... |

Status: 200 OK Time: 7 ms Size: 7.56 KB

```
{
    "error": null,
    "documents_found": 1,
    "documents": [
        {
            "content": {⋯
            },
            "hashes": {
                "sha256": "3305cbafe8a3a022830fb7f5aa851b8250f2b841b6e59c75b7f831efa6191668",
                "sha512":
                    "5fd447dca7f381832f04358a04c9d4893faf39a4d9dcb2199a0ba925661f27566d422168e6619ed9b8844f2659ca
                    f65eb8aa5c779e553cfdb7b40265084ba1b5"
            }
```

Figure 4.2: Testing of the `GET /csaf-documents/by-cve` route in Postman after importing the OpenAPI specification

Only Postman and Swagger UI were used for testing. It was important to test every possible combination of query parameters in every route. The required parameters however were set to a fixed value, so that checking the correctness of the response when arbitrarily setting the filter parameters is easier and more unambiguous. All routes for which this was possible were tested with randomly selected documents from the test data set (see section 4.2). Less attention was paid to the more modified documents in order to retain the real scenario of Siemens as much as possible, where the documents originally came from. For testing the `POST /csaf-documents/from-device-list` route, however, almost exclusively the modified documents could be used due to the frequent lack of product identification helpers.

The tests were performed manually using Postman. The parameters `with_signature` and `with_hashes`, as well as the regularly used parameters `before`, `after`, `profile` and `tracking_status` were not tested separately, because they are implemented the same for each route. Only the path parameters and route-specific parameters were tested for each route. The tests of the routes were carried out according to the expected response principle. That means, first a CSAF document was selected, which should be found by a defined API query. In some cases, a certain number of documents was also expected, for example in the case of the `GET /csaf-documents/by-publisher` route. This route was also used to test the authentication middleware, since the number of returned documents alone provides information about whether it works correctly. In addition, it was ensured that no non-`TLP:WHITE` documents were included if no token was transmitted. For testing the `POST /csaf-documents/from-device-list` route, the product identification helper objects added to the test data set in section 4.2 were searched for in different configurations in the response body.

Except for the arbitrary route problems mentioned in 4.3.4, all tests were successful.

During this implementation chapter, the following standard adjustments were defined in this section while developing the proof of concept:

Table 4.1: Implementation specification adjustments

| **Implementation specification adjustments** | |
|---|---|
| Naming | Description |
| `IMP-R3-A` | The specification of the `/document/distribution/tlp/label` should be enforced by the JSON schema or a mandatory test. |
| `IMP-R4-A` | See `IMP-R3-A`. |

The two adjustments are enforced by the same amendment. Both `IMP-R3` and `IMP-R4` are dependent on the presence of a TLP label in each CSAF document.

# 5 Integration into the specification

This chapter will look at the design and the implementation as a whole. It will be examined which adjustments are necessary to transition the API to the next version of the standard and which conflicts could arise in the process.

The previously mentioned standard adjustments (tables 3.5 and 4.1) are revisited, as well as the satisfiability of the requirements for embedding in the CSAF infrastructure (table 3.4). These are summarized once again in the following table 5.1:

Table 5.1: Specification adjustments summary

| **Specification adjustments summary** | |
|---|---|
| Naming | Description |
| `DES-R12-A` | Require the specification of a product identification helper in every product. |
| `IMP-R3-A` | The specification of the `/document/distribution/tlp/label` should be enforced by the JSON schema or a mandatory test. |
| `IMP-R4-A` | See `IMP-R3-A`. |
| `EMB-R1` | The implementation of the API must work on both CSAF (trusted) providers and aggregators. Listers do not require an implementation, as they only reference the (trusted) provider instances, running the API. |
| `EMB-R2` | The API endpoint must be able to be stored in the `provider-metadata.json` file. |
| `EMB-R3` | The JSON schema of the `provider-metadata.json` must be adapted accordingly due to `EMB-R2`. |
| `EMB-R4` | The API endpoint must be able to be stored in the `aggregator.json` file. |
| `EMB-R5` | The JSON schema of the `aggregator.json` must be adapted accordingly due to `EMB-R4`. |

## 5.1 Application requirements

### Specification changes

The specification adjustments and embedding requirements DES-R12-A, IMP-R3-A, IMP-R4-A, EMB-R3, and EMB-R5, as summarized in table 5.1, describe requirements for standardization. Accordingly, they must be enforced by the OASIS Open CSAF TC if the design proposed in chapter 3 is to be implemented. These mentioned requirements will thus not be discussed further in this thesis. The specific adaptations that must be made in order to fulfill the requirements EMB-R3 and EMB-R5 are described below in this chapter.

### API implementation in aggregators

During the proof of concept implementation and the subsequent testing of the API, it was found that the operation of the API on the aggregator is unproblematic. The prerequisite for this is that the configuration parameter `csaf_documents_path` has been set to the directory to which the CSAF documents are mirrored. The implementation finds all valid documents on its own. There are no routes or functions in the current design that work exclusively on aggregators. Accordingly, no further measures need to be taken here.

Thus, the requirement EMB-R1 is completely fulfilled.

### Adjustment of provider-metadata.json

The `provider-metadata.json` already provides an array with the different distribution methods under the `/distributions[]` property. It makes sense to specify the API method here as well. To do this, an object with the single property `api` is added to the array. It specifies the base URL, as well as all supported versions of the endpoint. An example is shown in listing 5.1. A client that intends to use the API of this (trusted) provider must now concatenate the base URL, as well as the highest version it supports, in order to be able to access the routes.

As with specifying the path to a ROLIE feed, for example, the API endpoint does not have to reside within the `/.well-known/` directory. Any path and even a different host can be specified as well.

The matter of whether a (trusted) provider gives permission to an aggregator to use its CSAF documents in the aggregator's API endpoint should be covered by the `/mirror_in_CSAF_aggregator` property already. If only listing is allowed, the API endpoint of the (trusted) provider must be used. Consequently, running the API in a lister is redundant.

Listing 5.1: Proposed adjustments to the `provider-metadata.json` to allow the discovery of the API endpoint

```
{
    "canonical_url": "https://provider-1.csaf:4431/.well-known/csaf/
        provider-metadata.json",
    "metadata_version": "2.0",
    "distributions": [
        {
            "rolie": { ... }
        },
        {
            "api": {
                "url": "https://provider-1.csaf:4431/.well-known/csaf/api",
                "supported_versions": ["v1"]
            }
        }
    ],
    "last_updates": "2022-07-18T11:34:19Z",
    ...
}
```

## Adjustment of aggregator.json

For the adjustment of the `aggregator.json` the same structure is used as in the `provider-metadata.json` as mentioned above. However, there is no dedicated distribution array, since they are already defined in the mirrored `provider-metadata.json` files in the `/csaf_providers[]/mirrors[]` property. However, since the API should unify all mirrored providers into one endpoint, embedding the configuration structure as a top-level property is proposed. A complete `aggregator.json` file could look like shown in listing 5.2 accordingly.

Listing 5.2: Proposed adjustments to the `aggregator.json` to allow the discovery of the API endpoint

```
{
    "aggregator": { ... },
    "aggregator_version": "2.0",
    "canonical_url": "https://aggregator-1.csaf:4431/.well-known/csaf-
        aggregator/aggregator.json",
    "csaf_providers": [ ... ],
    "api": {
        "url": "https://aggregator-1.csaf:4431/.well-known/csaf-aggregator/
            api",
        "supported_versions": ["v1"]
    },
    "last_updated": "2022-07-17T10:16:21Z"
}
```

If aggregators should later be extended by more distribution methods, which combine the documents from all mirrored providers, the use of a `/distributions[]` array would also be useful here. The schema could be similar to the one in the `provider-metadata.json`. However, this would be a major adaptation and is not part of this thesis.

As noted before, the PoC implementation of the aggregator in the test infrastructure completely omits validating the `provider-metadata.json` of the providers being mirrored. Accordingly, the customizations of the two metadata files must be incorporated into the JSON schema to re-enable validation. This is tracked in `EMB-R3` and `EMB-R5`.

### Specification of the API routes

Irrespective of the adaptations and requirements listed in table 5.1, the API design must of course also be specified so that CSAF infrastructure operators have the option of setting up their own implementation. The easiest variant would be to continue to provide the OpenAPI specification with additional descriptions and explanations for the individual components and routes and then to prescribe the implementation of it in the standard. Alternatively, the OpenAPI specification could be adopted in the format of the current standard, but this would cause the standard to grow considerably in size.

It is important to explain the behavior of all parameters in each route, as they may well behave differently. This is particularly evident from the `cvssv3` and `cvssv2` parameters in the `GET /csaf-documents/by-cve` and `POST /csaf-documents/from-device-list` routes: the first one requires the existence of the `/vulnerabilities[]/scores` properties, while the second one does not. Such a distinction is not apparent from the route naming alone. An inaccurate or even missing documentation of the parameters could accordingly lead to unequal behavior of two implementations.

## 5.2 Comparison with the existing distributions

It is clear that the API is currently the only distribution method capable of filtering based on the contents of the CSAF documents. This feature is especially important for the vulnerability management and SOCs, as they no longer have to search for the relevant CSAF documents themselves. Likewise, the API can also be more easily embedded into existing infrastructure – for example, in alerting systems or the process management systems for administrators.

However, one of the major problems with using the API are the privacy concerns, which would not occur in the other distribution methods. In principle, the added value of the API is that the filtering of CSAF documents is outsourced from the requesting client to the server. The consequence of this is that in some cases sensitive data must be sent to the server. This includes above all the device list, which allows conclusions to be drawn about the hardware and software used by the client. When using the API on a (trusted) provider belonging to the vendor category, this is rather negligible, since the provider usually only receives device lists with products from its own repertoire. On an aggregator, this has a much higher impact, as it combines all vendor endpoints and therefore also allows conclusions to be drawn about the "vendor hererogeneity" of the customer network. Most likely, the entire product inventory is transmitted to the aggregator. The route `POST /csaf-documents/from-device-list` is particularly affected by this, but also the macro route `GET /csaf-documents/by-cve`. The latter could indicate that a certain vulnerability has been found in the company and the impact of this is now being investigated.

However, whether these routes are used – and thus the information is transmitted – is left entirely to the user. The API also allows requesting all advisories at once to filter them locally, as the ROLIE feed distribution does. The following request would return all documents of a provider or aggregator without conveying any crucial information: `GET /csaf-documents/match-property?path=$.document&type=object`. This circumstance should be made clear in the standard.

Furthermore, this raises the question of how the API should be integrated into the distribution roles. For example, the CSAF provider must support either a ROLIE-based or directory-based distribution [17, section 7.2.2]. The API could also be added here as an optional variant. Alternatively, the specification of a new role would be conceivable, such as "CSAF filterable provider", which would be based on the trusted provider role. The advantage of this would be that a CSAF querier would know in advance whether the server actually supports document filtering. For the purpose of convincing manufacturers to use CSAF, the operation of the CSAF API must not be enforced in principle, which is why it is important to make its existence known if it is provided. Additionally, since the API basically works most sensibly and efficiently within an aggregator, it would also be conceivable to enforce it only in the aggregator role instead of defining a new role. This already has the requirement to provide hashes and signatures for each CSAF document, as does the trusted provider.

However, the API lacks one important feature in its current state: tracking document updates. With the directory-based distribution, the existence of a `changes.csv` file is prescribed [17, section 7.1.13], in which the current release date of each document must be indicated. This changes accordingly with updates. ROLIE feeds also carry the update date in the `/feed/entry[]/updated` property. The API only reveals this information through the `/document/tracking/current_release_date` property required in the CSAF document. However, there is no meta-dating outside the document, as in the other distribution methods.

Another difference, which has been pointed out several times in this work, is that the API does not keep different feeds per TLP label. Due to the design, distinctions can be easily matched at runtime to enforce access control, for example. Both the ROLIE-based and directory-based distribution methods keep all CSAF documents separated by TLP label.

The biggest problem with the integration into the CSAF standard will most likely be the standardization of the API routes, as there is no obviously practical way to implement this. The adaptations of the respective metadata files and their JSON schemas is not problematic. The PoC can also be run in all relevant CSAF components without any problems. If the adjustments from `DES-R12-A`, `IMP-R3-A` and `IMP-R4-A` are not realizable, alternatives can also be introduced in the route design.

In conclusion, it can be said that the API integrates well with the standard.

# 6 Conclusion

This chapter concludes the results of this work. It elaborates on the remaining problems of the concept, as well as the implementation of the proof of concept. The guiding questions are answered in the summary section. The end of this chapter covers opportunities for further research on the CSAF API, as well as other ways to enhance the CSAF ecosystem.

## 6.1 Problems concerning the concept itself

Although the route design allows the arbitrary arrangement of different parameters according to requirement `DES-R5` as defined in section 3.1, certain use-cases can still not be fulfilled in one query. For example, if all vulnerabilities are to be found for a product for which either a vendor fix or a workaround exists, the corresponding route must be called twice with different parameter values. The reason for this is that the `remediation_catgory` parameter only allows a single value. So the route must be called twice with different values for this parameter. This is problematic because there might be overlaps in the two results, which would force the user to build a union from the results to avoid duplicates. This case should ideally also be mapped by the API.

Another conspicuous aspect that emerged during testing of the PoC was that some queries generated very large responses with sometimes more than 50000 lines after JSON formatting. The reason for this is the consequence of requirement `DES-R19`. The number can of course increase indefinitely, depending on the number of CSAF documents handled by the API. This can lead to performance issues in larger aggregators, which could be addressed via the pagination principle mentioned in the Microsoft REST API guidelines [4, section 9.8]. Only a subset of the CSAF documents, along with a token for the next subset, would be sent. A client can now “paginate” over these records with multiple requests, either until the end of results or until the searched document is found. However, such an implementation would be more complex.

The specification of a filter parameter referring to the `/vulnerabilities[]/flags` property is not provided in this design. The property is used to convey a justification for the VEX status `known_not_affected`. While the VEX status is more important in the first place, the flags can additionally be used to reduce the response size and to narrow down the request.

A major problem also arose when testing the PoC. When searching for CSAF documents using the product identification helper, it became apparent that the use case of searching for the product name did exist, as already mentioned in section 3.3.2. In contrast to the properties in the product identification helper, the search for the product name is less deterministic, since neither the format nor the spelling is specified. However, it can be useful to allow both at the same time, especially to counteract the case that a product has no product identification helper.

As already documented in section 4.3.4, the implementation of the `GET /csaf-documents/match-propery` and `POST /csaf-documents/match-properies` routes was not completely possible due to the inappropriate design not matching the intended usage of JSONPath. The problem here was that the concept always assumed `path-type`, or `path-value` pairs. However, this use-case is not supported by JSONPath, since it is not only an addressing mechanism but also a querying mechanism. Since the type of every property is mostly ensured by the schematization of CSAF documents, the `type` parameter can be omitted. JSONPath should therefore be used for addressing and matching standalone and not only as a path in `path-value` pairs.

Another thing that may not be obvious by design is the composition of the CSAF documents in the response. While no errors were found during testing of the PoC, it was sometimes difficult to comprehend why some documents were included in the response. Some search and filter options search for sometimes very deep lying properties, which can also be located in multidimensional arrays. As a result, the entire document also contains many objects that do not match the query at all, since an advisory can also contain several products and vulnerabilities. A suitable data structure next to the CSAF documents in the response, which indicates the reason for the inclusion, would be conceivable, although not necessary.

On the other hand, the implementation of the macro routes turned out to work very well. Due to the narrower scope, these were easier to understand and implement. The device list route was also implemented well and works good, except for the aforementioned problem that the search for product names is not possible by design.

## 6.2 Problems concerning the proof of concept

The proof of concept is fully functional, except for the problems with the matching routes mentioned in section 4.3.4. The points mentioned below serve as subsequent improvements and as a guideline for an own implementation.

The `CSAFDocumentCollection` data structure for managing CSAF documents loads all documents from the file system into memory when the API binary is started in order to access them faster. Two problems arise from this: First, for very large aggregators, RAM usage can become correspondingly high. Similarly, the CSAF documents on the file system can also not be reloaded at runtime. Here, the implementation of a periodic refetch or file watcher would be necessary to quickly apply changes. This is particularly useful for new security advisories, which are still updated very regularly.

Another performance problem could arise from the filter mechanism. As described in section 4.3.1, several filter functions can be created, which are then applied to each CSAF document. It is required that all filter functions return ‘true‘ for a document to be included in the response. Currently, a filter function does not dilute the dataset, but only marks the document for exclusion. Thus, the next filter function processes all documents again. Here, it would make more sense algorithmically to remove documents directly as soon as the first filter determines its exclusion. Functionally, however, this makes no difference.

Currently, the PoC lacks the capability to handle version ranges as they are not part of the API design. These ranges are located in `branches_t` objects with the category `product_version_range` in the name property, but these are not utilized. Instead, only the product identification helper in `full_product_name_t` objects is considered. The CPEs that can be stored there also allow version matching. For example, if a security advisory exists with a product X with an arbitrary version, it can also be matched if a CPE containing the version is provided, since the latter is a subset of the CPE of product X. This is made possible by the CPE Name Matching specification [14]. In listing 6.1 an example implementation in Go with the library “go-cpe” [8] is described.

Listing 6.1: Checking, if the CPE of Microsoft Edge with any version is a subset of the CPE of Microsoft Edge in version 8.0.6001-beta in Go

```
package main

import (
    "fmt"

    "github.com/knqyf263/go−cpe/matching"
    "github.com/knqyf263/go−cpe/naming"
)

func main() {
    wfnSuperset, _ := naming.UnbindFS(
        ‘cpe:2.3:a:microsoft:edge:*:*:*:*:*:*:*:*‘)
    wfnSubset, _ := naming.UnbindFS(
        ‘cpe:2.3:a:microsoft:edge:8.0.6001:beta:*:*:*:*:*:*‘)
    fmt.Println(
        matching.IsSubset(wfnSubset, wfnSuperset)) // true
}
```

In the current PoC, CPEs are matched by string comparison. An implementation of CPE name matching would be a useful improvement. However, it should be noted that version matching only works with the CPE product identification helper. If products are searched for by serial number, the version matching is omitted, depending on the manufacturer’s rules.

During implementation, the use of group IDs in all documents was dispensed with for reasons of simplicity. It was noticed that in an object in `/vulnerabilities[]/remediations[]` both product IDs and group IDs can be stored, to which the remediation applies. The product matching however resolves the respective found product only to its product ID. It will then be searched in `/vulnerabilities[]/remediations[]/product_ids[]` in order to match the remediation object. However, if a product is part of a product group and only the group ID is specified in `/vulnerabilities[]/remediations[]/group_ids[]`, the remediation is not matched. This must be corrected in the implementation.

Ultimately, however, the proof of concept fulfills its purpose and succeeds in following the design. The only exception is the aforementioned matching routes, which are a design issue. After the adjustments to CSAF 2.1 have been made, the implementation is ready to act as part of the overall proof of concept. The API can already be used as a non-standard extension for CSAF 2.0.

## 6.3 Summary

This work aimed to answer the question whether an API can be built on top of CSAF as an additional distribution method. If yes, necessary adaptations should be documented and discussed. Despite the problems with the concept and implementation listed in this chapter, it could be shown that an API is both possible and useful. It serves as a feasible and flexible extension to the current distribution methods with the potential to replace them.

For both CERTs and CSIRTs, a CSAF API is a useful addition. The target group is primarily the companies that already use CSAF and those that intend to do so. The complexity and the effort, which are caused by the operation of the API, do not apply to the users. They can familiarize themselves with CSAF and obtain security advisories more easily and quickly, as there is no longer any need for local searching or filtering, as it would be the case with ROLIE feeds and the directory-based distribution. The entry barrier is thus slightly lower.

It turned out that only a few adjustments to the CSAF standard are necessary to run an API on it. These are listed in chapter 5 and are partially optional. Using the implementation as a PoC for CSAF 2.1 would probably also be possible after making the necessary adjustments.

The conceptual and implementation problems that arose have been extensively documented in this work and will help to eliminate them in subsequent work.

## 6.4 Outlook

As explained in chapter 5, the API can also function as a privacy-preserving alternative to the ROLIE feeds, if certain filtering mechanisms are not transmitted. Similarly, the API could also function as the primary exchange mechanism between (trusted) providers and aggregators – or even aggregators among themselves – if the concept is extended accordingly. Also, the functions of the entry point files `provider-metadata.json` and `aggregator.json` could be replaced by API routes. However, this approach would be impractical at the current state, as the advantage of static files would be lost. More than just a simple web server would be required, which would greatly increase the inhibition for many vendors to run a CSAF endpoint. As an alternative, however, this function can be considered.

What the overall CSAF ecosystem is still missing – and what has also come up through the API's authentication mechanism – is a suitable management system, both for CSAF documents themselves, and a system for controlling access to non-`TLP:WHITE` documents. The API uses bearer tokens in this implementation, which allows granular access to specific TLP labels. So far, these are specified in the `api.toml` configuration file. However, a central place to specify them would make more sense, since the access rights could be extended to the other distribution methods as well. Furthermore, access rights can also be scoped for user identities – for example, from an Active Directory – which would make the authorization model more flexible. Accordingly, the authentication scheme was intentionally not specified with requirement `DES-R4` in section 3.1, since the mechanism in general was also not defined specifically [17, section 7.1.5]. This leaves open how an access control system will handle this in the future. This topic, as well as the conceptualization of a management system for CSAF documents, offers much room for further research.

Another idea to extend the functionality of the API would be to use a subscribe mechanism in conjunction with a device list. This would allow the specification of an own URL as webhook endpoint, to which the CSAF API server regularly sends updates to advisories about the submitted devices. This could be new advisories as well as updates to them. OpenAPI allows documentation of this behavior via callback components [21]. This would further shorten the time to react to vulnerabilities. Again, there is much room for further research.

Finally, it can be said that this work offers far-reaching possibilities for further research. It lays the foundation for the API-based distribution of CSAF documents for the CSAF 2.1 standard.

# Bibliography

[1] FIRST Traffic Light Protocol Special Interest Group (TLP-SIG). *Traffic Light Protocol (TLP)*. Version 1.0. FIRST. URL: `https://www.first.org/tlp/`.

[2] Ed. A. Phillips et al. *Tags for Identifying Languages*. RFC 5646. Sept. 2009. URL: `https://datatracker.ietf.org/doc/html/rfc5646`.

[3] Caddy. *Caddy Webserver Homepage*. `https://caddyserver.com/`. 2022.

[4] Dave Campbell et al. *Guidelines.md in microsoft/api-guidelines*. `https://github.com/microsoft/api-guidelines/blob/8465bf242ad1743beb49caa90a5a36c9190dc648/Guidelines.md`. 2022.

[5] *CycloneDX*. Version 1.4. OWASP, Jan. 2022. URL: `https://cyclonedx.org/docs/1.4/json/`.

[6] Golang Package Documentation. *context (standard library)*. July 12, 2022. URL: `https://pkg.go.dev/context` (visited on 07/28/2022).

[7] J. Field et al. *Resource-Oriented Lightweight Information Exchange (ROLIE)*. RFC 8322. Feb. 2018. URL: `https://datatracker.ietf.org/doc/html/rfc8322`.

[8] Teppei Fukuda. *knqyf263/go-cpe*. `https://github.com/knqyf263/go-cpe`. 2020.

[9] gorilla. *gorilla/mux*. `https://github.com/gorilla/mux`. 2022.

[10] Derek Kruszewski. *Understanding Vulnerability Exploitability eXchange (VEX)*. `https://www.csa.gov.sg/-/media/Csa/Documents/Events/OTCEP-2021/aDolus-White-Paper--Introduction-to-VEX--V0100.pdf`. aDolus Technology Inc., Sept. 2021.

[11] nexB. *nexB/univers*. `https://github.com/nexB/univers`. 2022.

[12] PaesslerAG. *PaesslerAG/gval*. `https://github.com/PaesslerAG/gval`. 2022.

[13] PaesslerAG. *PaesslerAG/jsonpath*. `https://github.com/PaesslerAG/jsonpath`. 2022.

[14] Mary C. Parmelee et al. *Common Platform Enumeration: Name Matching Specification*. Version 2.3. National Institute of Standards and Technology (NIST), Aug. 2011. URL: `https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7696.pdf`.

[15] Tom Preston-Werner. *Semantic Versioning*. Version 2.0.0. June 2013. URL: `https://semver.org`.

[16] Ed. R. Fielding et al. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231. June 2014. URL: `https://datatracker.ietf.org/doc/html/rfc7231`.

[17] Langley Rock, Stefan Hagen, and Thomas Schmidt, eds. *Common Security Advisory Framework*. Version 2.0 Committee Specification (CS) 02. OASIS, June 2022. URL: https://docs.oasis-open.org/csaf/csaf/v2.0/cs02/csaf-v2.0-cs02.html.

[18] Thomas Schmidt and Jens Wiesner. *Securing the Supply Chain Together - Through Automation of Advisories and Vulnerability Management*. FIRST. June 27, 2022. URL: https://www.first.org/conference/2022/program#pSecuring-the-Supply-Chain-Together-Through-Automation-of-Advisories-and-Vulnerability-Management (visited on 07/28/2022).

[19] Smartbear. *SoapUI Docs: Working With REST Services and WADL*. https://www.soapui.org/docs/rest-testing/working-with-rest-services/. 2022.

[20] Swagger. *OpenAPI Specification*. Version 3.0.3. Swagger, Feb. 2020. URL: https://swagger.io/specification/.

[21] Swagger. *OpenAPI Specification: Callbacks*. Version 3.0.3. Swagger, Feb. 2020. URL: https://swagger.io/docs/specification/callbacks/.

[22] Swagger. *OpenAPI Specification: Components Section*. Version 3.0.3. Swagger, Feb. 2020. URL: https://swagger.io/docs/specification/components/.

[23] Swagger. *Swagger editor*. https://editor.swagger.io/. 2022.

[24] Swagger. *Swagger UI*. https://swagger.io/tools/swagger-ui/. 2022.

[25] Sascha L. Teichmann et al. *csaf-poc/csaf_distribution*. https://github.com/csaf-poc/csaf_distribution. 2022.

# Statutory Declaration

I herewith declare that I have composed the present thesis myself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The thesis in the same or similar form has not been submitted to any examination body and has not been published. This thesis was not yet, even in part, used in another examination or as a course performance.

Leon Schmidt

Offenburg, August 2, 2022

# A OpenAPI specification

This is the full OpenAPI specification document specifying the API in YAML format.

```
openapi: 3.0.1

info:
  title: Common Security Advisory Framework (CSAF) 2.0 Distribution API
  description: 'This is the OpenAPI definition of the CSAF 2.0 Distribution API v1 designed to be implemented in (trusted) providers and aggregators.'
  # termsOfService: https://example.com/pending
  # contact:
  #   email: pending@example.com
  version: 0.1.0

#externalDocs:
#  description: Swagger documentation (JSON)
#  url: https://example.com/.well-known/csaf/swagger.json

servers:
  - url: http://localhost:8080/.well-known/csaf/api/v1
  - url: https://provider-1.csaf:4431/.well-known/csaf/api/v1
  - url: https://aggregator-1.csaf:4431/.well-known/csaf-aggregator/api/v1

tags:
  - name: Meta queries
    description: These queries are used to retrieve information for the CSAF server itself. They do not convey any CSAF documents and do not offer authentication.
  - name: Macro queries
    description: These queries (under the ‘/by*‘ path) can be used as shorter variants of the arbitrary queries.
  - name: Arbitrary queries
    description: These queries are more flexible and can query almost every property in a CSAF document.
    #externalDocs:
    #  description: Docs pending
    #  url: https://example.org/pending
  - name: Special queries
    description: These queries (under the ‘/from*‘ path) use external informations to retrieve CSAF documents.

```

```
paths:

  /metadata:
    get:
      tags:
        - Meta queries
      summary: Retrieve the provider-metadata.json or aggregator.json file
      description: Retrieve the `provider-metadata.json` or `aggregator.json` file from this server. This depends on the role (see `/role` route).
      operationId: getMetadata
      responses:
        200:
          description: Object containing either a `provider-metadata.json` or `aggregator.json` file
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    $ref: '#/components/schemas/Error'
                  provider_metadata:
                    type: object
                    description: "See https://docs.oasis-open.org/csaf/csaf/v2.0/provider_json_schema.json"
                  aggregator:
                    type: object
                    description: "See https://docs.oasis-open.org/csaf/csaf/v2.0/aggregator_json_schema.json"
        400:
          description: Bad request
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    $ref: '#/components/schemas/Error'
              example:
                error:
                  errcode: BAD_REQUEST
                  errmsg: "..."
        # do not yield 401, as there is no authentication
        500:
          description: Server error
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    $ref: '#/components/schemas/Error'
              example:
                error:
                  errcode: UNKNOWN_ERROR
                  errmsg: "..."

  /role:
    get:
      tags:
        - Meta queries
      summary: Get the role of the CSAF server
      description: Returns the role of this CSAF server. See section 7.2 of the CSAF standard for details.
      operationId: getRole
      responses:
        200:
          description: Role of this CSAF server
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    $ref: '#/components/schemas/Error'
                  role:
                    type: string
                    enum:
                      - csaf_publisher
                      - csaf_provider
                      - csaf_trusted_provider
                      - csaf_lister
                      - csaf_aggregator
                    default: csaf_trusted_provider
        400:
          description: Bad request
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    $ref: '#/components/schemas/Error'
              example:
                error:
                  errcode: BAD_REQUEST
                  errmsg: "..."
        # do not yield 401, as there is no authentication
        500:
          description: Server error
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    $ref: '#/components/schemas/Error'
              example:
                error:
                  errcode: UNKNOWN_ERROR
                  errmsg: "..."

  /csaf-documents/by-id/{publisher_namespace}/{tracking_id}:
    get:
      tags:
        - Macro queries
      summary: Retrieves documents matching the globally unique document ID
      description: Retrieves CSAF documents matching the globally unique document ID according to sections 3.2.1.8.5 and 3.2.1.12.4 of the standard. Matching is always exact.
      operationId: getByID
      security:
        - bearerAuth: []
      parameters:
        - name: publisher_namespace
          in: path
          required: true
          schema:
            type: string
            format: uri
        - name: tracking_id
          in: path
          required: true
          schema:
            type: string
        - $ref: '#/components/parameters/WithHashes'
        - $ref: '#/components/parameters/WithSignature'
      responses:
        200:
          $ref: '#/components/responses/200Ok'
        400:
          $ref: '#/components/responses/400BadRequest'
        401:
          $ref: '#/components/responses/401Unauthorized'
        500:
          $ref: '#/components/responses/500UnknownError'

  /csaf-documents/by-title/{title}:
    get:
      tags:
        - Macro queries
      summary: Retrieves documents matching its title
```

```yaml
      description: Retrieves CSAF documents matching its title. Matching is exact by default.
      operationId: getByTitle
      security:
        - bearerAuth: []
      parameters:
        - name: title
          in: path
          required: true
          schema:
            type: string
        - $ref: '#/components/parameters/ValueMatching'
        - $ref: '#/components/parameters/Before'
        - $ref: '#/components/parameters/After'
        - $ref: '#/components/parameters/Profile'
        - $ref: '#/components/parameters/TrackingStatus'
        - $ref: '#/components/parameters/WithHashes'
        - $ref: '#/components/parameters/WithSignature'
      responses:
        200:
          $ref: '#/components/responses/200Ok'
        400:
          $ref: '#/components/responses/400BadRequest'
        401:
          $ref: '#/components/responses/401Unauthorized'
        500:
          $ref: '#/components/responses/500UnknownError'

  /csaf-documents/by-publisher/{publisher_name}:
    get:
      tags:
        - Macro queries
      summary: Retrieves documents matching the publisher name
      description: Retrieves CSAF documents documents matching the publisher name and optionally the publisher namespace and publisher category. The matching paramter only applies to the `publisher_name` parameter. This route only makes sense when querying an aggregator.
      operationId: getByPublisher
      security:
        - bearerAuth: []
      parameters:
        - name: publisher_name
          in: path
          required: true
          schema:
            type: string
        - $ref: '#/components/parameters/ValueMatching'
        - name: publisher_namespace
          in: query
          schema:
            type: string
            format: uri
        - name: publisher_category
          in: query
          schema:
            type: string
        - $ref: '#/components/parameters/Before'
        - $ref: '#/components/parameters/After'
        - $ref: '#/components/parameters/Profile'
        - $ref: '#/components/parameters/TrackingStatus'
        - $ref: '#/components/parameters/WithHashes'
        - $ref: '#/components/parameters/WithSignature'
      responses:
        200:
          $ref: '#/components/responses/200Ok'
        400:
          $ref: '#/components/responses/400BadRequest'
        401:
          $ref: '#/components/responses/401Unauthorized'
        500:
          $ref: '#/components/responses/500UnknownError'


  /csaf-documents/by-cve/{cve}:
    get:
      tags:
        - Macro queries
      summary: Retrieves documents containing the CVE specified
      description: Retrieves CSAF documents containing the CVE specified. Matching is always exact.
      operationId: getByCVE
      security:
        - bearerAuth: []
      parameters:
        - name: cve
          in: path
          description: CVE enumeration to find CSAF documents for
          required: true
          schema:
            type: string
        - $ref: '#/components/parameters/CVSSv3Range'
        - $ref: '#/components/parameters/CVSSv2Range'
        - $ref: '#/components/parameters/Before'
        - $ref: '#/components/parameters/After'
        - $ref: '#/components/parameters/Profile'
        - $ref: '#/components/parameters/TrackingStatus'
        - $ref: '#/components/parameters/WithHashes'
        - $ref: '#/components/parameters/WithSignature'
      responses:
        200:
          $ref: '#/components/responses/200Ok'
        400:
          $ref: '#/components/responses/400BadRequest'
        401:
          $ref: '#/components/responses/401Unauthorized'
        500:
          $ref: '#/components/responses/500UnknownError'

  /csaf-documents/match-property:
    get:
      tags:
        - Arbitrary queries
      summary: Retrieve documents matching arbitrary properties by value and/or type
      description: Retrieve CSAF documents matching an arbitrary property by value and/or type. Documents not containing the property will be skipped from evaluation by default. For combining multiple property-value pairs, use the `/match-properties` route.
      operationId: getDocumentByJSONMatch
      security:
        - bearerAuth: []
      parameters:
        - name: path
          in: query
          description: Path to the JSON property in JSONPath syntax (see https://tools.ietf.org/id/draft-goessner-dispatch-jsonpath-00.html)
          required: true
          schema:
            type: string
            example: $.document.distribution.tlp.label
        - name: type
          in: query
          description: Type of the property definded in `path`
          schema:
            type: string
            enum:
              - string
              - number
              - object
              - array
              - boolean
            example: string
        - name: value
          in: query
          description: Value of the property definded in `path`. To match `null`, use `type=object&value=null`.
          schema:
            type: string
            example: WHITE
        - $ref: '#/components/parameters/ValueMatching'
        - name: include_missing
```

```
          in: query
          description: Whether to include documents in the result that do not have the property specified in `path`
          schema:
            type: boolean
            default: false
        - $ref: '#/components/parameters/Before'
        - $ref: '#/components/parameters/After'
        - $ref: '#/components/parameters/Profile'
        - $ref: '#/components/parameters/TrackingStatus'
        - $ref: '#/components/parameters/WithHashes'
        - $ref: '#/components/parameters/WithSignature'
      responses:
        200:
          $ref: '#/components/responses/200Ok'
        400:
          $ref: '#/components/responses/400BadRequest'
        401:
          $ref: '#/components/responses/401Unauthorized'
        500:
          $ref: '#/components/responses/500UnknownError'

  /csaf-documents/match-properties:
    post:
      tags:
        - Arbitrary queries
      summary: Retrieve documents matching arbitrary properties by value and/or type
      description: Retrieve CSAF documents matching arbitrary properties by value and/or type. Documents not containing the property will be skipped from evaluation by default.
      operationId: getDocumentByJSONMatches
      security:
        - bearerAuth: []
      parameters:
        - $ref: '#/components/parameters/WithHashes'
        - $ref: '#/components/parameters/WithSignature'
      requestBody:
        $ref: '#/components/requestBodies/AdvancedMatchingRequestBody'
      responses:
        200:
          $ref: '#/components/responses/200Ok'
        400:
          $ref: '#/components/responses/400BadRequest'
        401:
          $ref: '#/components/responses/401Unauthorized'
        500:
          $ref: '#/components/responses/500UnknownError'

  /csaf-documents/from-device-list:
    post:
      tags:
        - Special queries
      summary: Retrieves documents matching the provided device list
      description: Retrieves CSAF documents matching the provided device list. With the `product_status` parameter, the product status according to section 3.2.3.9 of the standard can be used to filter the results.
      operationId: getDocumentsByDeviceList
      security:
        - bearerAuth: []
      parameters:
        - $ref: '#/components/parameters/ProductStatus'
        - $ref: '#/components/parameters/CVSSv3Range'
        - $ref: '#/components/parameters/CVSSv2Range'
        - $ref: '#/components/parameters/RemediationCategory'
        - $ref: '#/components/parameters/Before'
        - $ref: '#/components/parameters/After'
        - $ref: '#/components/parameters/Profile'
        - $ref: '#/components/parameters/TrackingStatus'
        - $ref: '#/components/parameters/WithHashes'
        - $ref: '#/components/parameters/WithSignature'
      requestBody:
        $ref: '#/components/requestBodies/DeviceListRequestBody'
      responses:
        200:
          $ref: '#/components/responses/200Ok'
        400:
          $ref: '#/components/responses/400BadRequest'
        401:
          $ref: '#/components/responses/401Unauthorized'
        500:
          $ref: '#/components/responses/500UnknownError'


components:

  parameters:

    Before:
      name: before
      in: query
      description: Matches documents with dates before `/document/tracking/initial_release_date`
      schema:
        type: string
        format: date-time

    After:
      name: after
      in: query
      description: Matches documents with dates after `/document/tracking/
```

```
          initial_release_date‘
      schema:
        type: string
        format: date-time

    ValueMatching:
      name: matching
      in: query
      description: Sets the mechanism how values are matched with the
          acutal value. This is only applied if the value that is to be
          matched is of or can be inferred to type string. Otherwise
          matching is always exact.
      schema:
        type: string
        enum:
          - exact
          - regex
          - begins-with
          - ends-with
          - contains
        default: exact

    ProductStatus:
      name: product_status
      in: query
      schema:
        type: string
        enum:
          - first_affected
          - first_fixed
          - fixed
          - known_affected
          - known_not_affected
          - last_affected
          - recommended
          - under_investigation

    Profile:
      name: profile
      in: query
      schema:
        type: string
        enum:
          - csaf_base
          - csaf_security_incident_response
          - csaf_informational_advisory
          - csaf_security_advisory
          - csaf_vex

    TrackingStatus:
      name: tracking_status
      in: query
      description: Matches documents with tracking status in /document/tracking/status according to section 3.2.1.12.7 in the standard.
      schema:
        type: string
        enum:
          - draft
          - final
          - interim

    CVSSv3Range:
      name: cvssv3
      description: "Arithmetic expressions to match the CVSSv3 with. Specify more than one to match (AND relation) with ‘,‘. Supported operators are: ‘>‘, ‘<‘, ‘==‘, ‘>=‘, ‘<=‘"
      in: query
      schema:
        type: string
      example: ">7,<9"

    CVSSv2Range:
      name: cvssv2
      description: "Arithmetic expressions to match the CVSSv2 with. Specify more than one to match (AND relation) with ‘,‘. Supported operators are: ‘>‘, ‘<‘, ‘==‘, ‘>=‘, ‘<=‘"
      in: query
      schema:
        type: string
      example: ">7,<9"

    RemediationCategory:
      name: remediation_category
      in: query
      schema:
        type: string
        enum:
          - mitigation
          - no_fix_planned
          - none_available
          - vendor_fix
          - workaround

    WithHashes:
      name: with_hashes
      description: Whether to include each documents hashes in ‘/documents[]/hashes‘
      in: query
      schema:
        type: boolean

    WithSignature:
      name: with_signature
      description: Whether to include each documents signature in `/documents[]/signature`
      in: query
      schema:
        type: boolean


  schemas:

    CSAFDocument:
      type: object
      properties:
        document:
          type: object
        product_tree:
          type: object
        vulnerabilites:
          type: object
      required:
        - document

    Error:
      type: object
      default: null
      properties:
        errcode:
          type: string
          description: Parsable, enumerated error code
        errmsg:
          type: string
          description: Stateful error message
      required:
        - errcode

    CSAFDocumentResponse:
      type: object
      properties:
        error:
          $ref: '#/components/schemas/Error'
        documents_found:
          type: integer
        documents:
          type: array
          items:
            type: object
            required:
              - content
            properties:
              content:
                $ref: '#/components/schemas/CSAFDocument'
              signature:
                type: string
              hashes:
                type: object
                properties:
                  sha256:
                    type: string
                  sha512:
                    type: string

    DeviceList:
      type: array
      items:
        type: object
        properties:
          cpe:
            type: string
          hashes:
            type: array
            items:
              type: object
          model_numbers:
            type: array
            items:
              type: string
          purl:
            type: string
          sbom_urls:
            type: array
            items:
              type: string
          serial_numbers:
            type: array
            items:
              type: string
          skus:
            type: array
            items:
              type: string
          x_generic_uris:
            type: array
            items:
              type: string

    AdvancedMatching:
      type: object
      properties:
        matching_default:
          type: string
          enum:
            - exact
            - regex
            - begins-with
            - ends-with
            - contains
          default: exact
        operator:
          type: string
          enum:
            - and
            - or
          default: and
        matches:
          type: array
          items:
            type: object
            properties:
              path:
                type: string
                example: $.document.distribution.tlp.label
              type:
                type: string
                enum:
                  - string
                  - number
                  - object
                  - array
                  - boolean
                default: string
              value:
                type: object
                example: "WHITE"
              matching:
                type: string
                enum:
                  - exact
                  - regex
                  - begins-with
                  - ends-with
                  - contains
              include_missing:
                type: boolean
                default: false
            required:
              - path
              - value
      required:
        - matches


  examples:

```

```
CSAFDocumentResponse0Documents:
  value:
    error: null
    documents_found: 0
    documents: []

CSAFDocumentResponse2Documents:
  value:
    error: null
    documents_found: 2
    documents: [
      {
        "content": {}
      },
      {
        "content": {}
      },
    ]

CSAFDocumentResponse2DocumentsWithHashesAndSignature:
  value:
    error: null
    documents_found: 2
    documents: [
      {
        "content": {},
        "hashes": {
          "sha265": "998c2d8ede020bf881cfad80316a21ff58748409389011893ca15f9e3e2e38e2"
        },
        "signature": "-----BEGIN PGP SIGNATURE-----\nabc123\n-----END PGP SIGNATURE-----"
      },
      {
        "content": {},
        "hashes": {
          "sha265": "4a4ad15670aefb7b9dca55c5c4974b2ed250f2a719af30bc8ad15b41c7efcd6b"
        },
        "signature": "-----BEGIN PGP SIGNATURE-----\ndef456\n-----END PGP SIGNATURE-----"
      },
    ]

DeviceListWindows10:
    value: [
      {
        "cpe": "cpe:2.3:o:microsoft:windows_10:-:*:*:*:*:*:x86:*"
      }
```

```
        ]

    DeviceListWindows10AndKeycloak:
      value: [
          {
            "cpe": "cpe:2.3:o:microsoft:windows_10:-:*:*:*:*:*:x86:*"
          },
          {
            "cpe": "cpe:2.3:a:redhat:keycloak:1.2.0:-:*:*:*:*:*:*",
            "sbom_urls": [
              "https://raw.githubusercontent.com/CycloneDX/bom-examples/master/SBOM/keycloak-10.0.2/bom.json"
            ]
          }
        ]


  requestBodies:

    DeviceListRequestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/DeviceList'
          examples:
            Windows 10:
              $ref: '#/components/examples/DeviceListWindows10'
            Windows 10 and Keycloak:
              $ref: '#/components/examples/DeviceListWindows10AndKeycloak'

    AdvancedMatchingRequestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/AdvancedMatching'


  responses:

    200Ok:
      description: Success. ‘error‘ is always ‘null‘.
      content:
        application/json:
          schema:
              $ref: '#/components/schemas/CSAFDocumentResponse'
          examples:
            No documents:
              $ref: '#/components/examples/CSAFDocumentResponse0Documents'
            Two documents:
              $ref: '#/components/examples/CSAFDocumentResponse2Documents'
            Two documents with hashes and signature:
```

```yaml
              $ref: '#/components/examples/CSAFDocumentResponse2DocumentsWithHashesAndSignature'

    400BadRequest:
      description: Bad request
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/CSAFDocumentResponse'
          example:
            error:
              errcode: BAD_REQUEST
              errmsg: "..."

    401Unauthorized:
      description: Bearer token is invalid
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/CSAFDocumentResponse'
          example:
            error:
              errcode: AUTH_INVALID
              errmsg: "The specified API token is not known by the server"

    500UnknownError:
      description: Server error
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/CSAFDocumentResponse'
          example:
            error:
              errcode: SERVER_ERROR
              errmsg: "..."


  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      #bearerFormat: JWT    # optional, arbitrary value for documentation purposes
```

# B API error codes

| Error code | Usage |
|---|---|
| BAD_REQUEST | Malformed request body, malformed parameter (e.g. enum ignored), invalid format (e.g. dates, URI) |
| MISSING_PARAMETER | Lack of required parameter (suberror of BAD_REQUEST) |
| DUPLICATE_PARAMETER | Parameter key was specified multiple times (suberror of BAD_REQUEST) |
| URL_DECODE_ERROR | URL parameter not encoded correctly (suberror of BAD_REQUEST) |
| MISSING_ACCEPT_HEADER | Accept header was not set (suberror of BAD_REQUEST) |
| AUTH_INVALID | Authentication data is not valid or not known to the server |
| SERVER_ERROR | Server error that cannot be resolved by the user (e.g. unmarshalling, internal context exceeded) |
| UNKNOWN_ERROR | Unknown error, fallback if other codes do not fit |
| METHOD_NOT_ALLOWED | A HTTP method was used that is not allowed for this route |
| NOT_FOUND | The requested resource does not match a route with any HTTP method |
| ROLE_UNDEFINED | The CSAF role of the server, the API is used in, was not defined |

## Notes

- This list contains only error codes for CSAF API specific cases, not HTTP specific cases (e.g. host header mismatch). Errors of this type are handled by the web server or client and might not return a response body of content type application/json or a response body at all.
- Errors like “method not allowed” and “route not found” also return error codes and are of content type application/json, even if no route matched the request URL.