




# Mining Security Documentation Practices in OpenAPI Descriptions

Diana Carolina Muñoz Hurtado   
Software Institute  
USI Lugano, Switzerland  
carolina.munoz@usi.ch

Souhaila Serbout   
Software Institute  
USI Lugano, Switzerland  
souhaila.serbout@usi.ch

Cesare Pautasso   
Software Institute  
USI Lugano, Switzerland  
c.pautasso@ieee.org

**Abstract**—Security is an integral requirement of any trustworthy software architecture, particularly critical for application programming interfaces (APIs). In this paper, we survey security documentation practices, specifically API security schemes related to authentication and authorization, by mining a large collection of OpenAPI descriptions retrieved from open-source GitHub repositories. Our study focuses on detecting existing security schemes and evaluating their prevalence and positioning within API descriptions. We distinguish whether security schemes are introduced locally (at the path or operation level) or globally (for the entire API). Our analysis highlights scenarios where security schemes are featured in APIs in different proportions over time, thus tracking whether the API documentation tends to include more (or less) security details as the API evolves.

**Keywords:** Web APIs, OpenAPI descriptions, API keys, Security schemes, Security documentation practices

## I. INTRODUCTION

APIs play an important role in open software architectures, as they promote reuse and composition and facilitate access to data sources, information repositories, and computational services [36, 9, 29]. Public Web APIs are meant to be invoked from any Internet-connected client, which often creates a need to control, protect, and limit access to known authenticated clients [24, 22]. These security measures should be communicated to clients to ensure they can select trusted APIs they can rely upon and properly configure secure access to them [13, 25]. From the client’s perspective, the absence of security documentation can negatively impact the API’s perceived trustworthiness and overall quality [4, 15, 27, 2, 35]. And, from the provider’s perspective, this can lead to misuse of the API, potentially resulting in unintended exposure of sensitive information [5, 16, 7, 34], vulnerability exploitation, or unauthorized access [12, 33, 3]. Having security documentation can also speed up onboarding new clients and reduce the integration process cost [32, 17].

In this paper, we investigate the state of security documentation practices in Web API descriptions. We focus on a large collection of API models mined from open-source GitHub repositories represented using OpenAPI [1], a widely adopted specification for modeling APIs in a structured, machine-readable format allowing large-scale fully automated analysis [30, 19, 21]. We aim to analyze how API designers decide to convey security information within these descriptions. We also track how the security coverage of APIs [26] evolves

over time to determine whether security is documented from the beginning of the API history, or it is included only later once the API reaches a given size.

This large-scale study addresses the following questions:

RQ1) To what extent are security aspects documented in Web APIs described using Swagger 2.0 or OpenAPI 3.0?

This question aims to examine whether and how security-relevant design decisions are documented in API specifications written in different versions of the OpenAPI standard.

RQ2) How does the level of detail in security documentation within OpenAPI descriptions vary along API histories?

Security schemes can be easily applied uniformly across the entire API. Alternately, each operation can be secured in a different way. The goal is to systematically examine all possible alternative granularity levels at which security can be applied and explore possible factors influencing this decision.

RQ3) How does security coverage correlate with API size and evolve over time?

The goal is to reveal patterns affecting whether API providers improve or modify security documentation, reflecting evolving security needs, and whether security is present from the start of an API lifecycle or added as the API grows.

To answer these questions, we perform a detailed analysis of security-related fields and keywords in OpenAPI descriptions. The analysis is performed on a snapshot of the APISitic dataset [28] composed of 915 988 OpenAPI specifications corresponding to commits pushed to Github between 2015 and 2023, and belonging to 270 564 distinct APIs. The collection contains only the commits that have affected the specification and not all of the repository, which fits with the goal of the analysis in this paper.

Our findings reveal that a significant number of specifications lack security documentation, with a very slow improvement in the security coverage over time. Moreover, APIs described using OpenAPI 3.0 (OAS 3.0) tend to incorporate more security features compared to Swagger 2.0 (OAS 2.0). In terms of the granularity of how security mechanisms are described, we found that they tend to be applied globally and uniformly to all API operations.

The paper is outlined as follows: Sec. II provides background on security documentation in OpenAPI specifications.

Sec. III reviews related work in API security. Sec. IV introduces the definition of key detectors and metrics employed in the analysis. In Sections V, VI, and VII we present the analysis results related to each of the three research questions, which are further discussed in Section VIII. Section IX addresses potential threats to validity in our study, outlining limitations and assumptions. Finally, Section X draws conclusions based on the findings and discusses future research directions.

## II. BACKGROUND

### A. OpenAPI Security Components and Schemes

The modularity of the OpenAPI language allows defining the security schemes in the `components` section, under the `securitySchemes` key in the case of OAS 3.0 and under the `definitions` section in the case of OAS 2.0. A security scheme defines how an API is secured by documenting the authentication and authorization methods that the API supports. Each scheme is part of the API documentation, enabling API consumers to understand how to interact with the API securely.

The current version of OpenAPI supports the definition of five types of security schemes:

- **API Key:** The client obtains and sends its unique API key as part of a request header, query parameter, or cookie [23].
- **HTTP Authentication:** includes standard HTTP authentication methods like Basic (username/password), Bearer (such as a JWT), and others.
- **OAuth2:** a more complex authorization framework that supports several "flows" (or grant types), like implicit, password, client credentials, and authorization code, for different types of applications and security requirements [10].
- **OpenID Connect:** an identity layer on top of OAuth2, enabling clients to verify the identity of end-users based on the authentication performed by an authorization server [11].
- **Mutual TLS:** a protocol where both client and server authenticate each other's identities using digital certificates [31].

The definition in the `securitySchemes` section depends on the type of Security Scheme used. In Listing 1 we show an example of a security scheme definition under OpenAPI 3.0 where an API uses OAuth2 [18] with the authorization code flow to provide secure access to social media posts.

### B. Security Documentation in Swagger 2.0 vs. OpenAPI 3.0

We categorize and evaluate security documentation practices based on two versions of the specification language to assess the impact of OAS 3.0's enhancements. OAS 2.0 already allowed reusable security definitions for API keys, basic authentication, and OAuth2, but with limited configuration options. OAS 3.0 introduced the `securitySchemes` component within a modular `components` section, enabling more flexible and reusable security definitions across endpoints. Additionally, OAS 3.0 expanded OAuth2 support with multiple flow types (implicit, password, clientCredentials, and authorizationCode) and introduced new authentication types, such as HTTP authentication schemes (e.g., bearer and basic), OpenID Connect, and Mutual TLS.

## III. RELATED WORK

A recent systematic literature review of API security research by Díaz-Rojas et al. identifies 66 distinct threats, most notably spoofing and tampering, and explored a suite of countermeasures including 21 techniques, 11 design patterns, and 34 methods to fortify APIs against these vulnerabilities [8]. More in detail, Cheh and Chen propose a semi-automated method for identifying security flaws in the design of API specifications using the OpenAPI standard: the endpoints related to sensitive data fields are analyzed for exposure [5]. They define risk exposure as the case where a field is sent through an endpoint that does not require authorization or authentication. Their technique, applied to the Open Bank Project API which includes 304 API operations in 142 endpoint paths and 345 data schemas containing 402 data fields identified 31 sensitive data fields, 29 insufficiently protected API calls, and 34 high-risk calls prone to exposing sensitive data. In our case, we aim to map the endpoint landscape by analyzing how security requirements are described across a very broad spectrum of APIs.

Bermbach and Wittern performed a study that revisited the performance, availability, and security configurations of popular web APIs, and underscored the dynamic and varied nature of web API ecosystems [4]. A particularly relevant finding for our paper was the evolution of Transport Layer Security (TLS) preferences among API providers, reflecting shifts toward stronger security postures. In this study we aim at analyzing how such trend is reflected in OpenAPI descriptions.

Chen et al. introduce API Prober 2.0, a tool for collaborative annotation and testing of security schemes [6]. AP2 was tested on nine real-world APIs, evaluating their security scheme attributes. In this study, we statically analyzed different decisions made by developers to implement security schemes by providing a systematic classification of API commits in which security schemes can be detected. This approach allows us to automatically classify a large number of API commits into different levels of granularity.

**Listing 1:** Example of Security Component definition in OpenAPI in OAS 3.0

```

components:
  securitySchemes:
    OAuth2: # Custom name for the OAuth2
            security scheme
            type: oauth2
            flows:
              authorizationCode:
                authorizationUrl: https://
                    socialmedia.com/oauth/authorize
                tokenUrl: https://socialmedia.com/
                    oauth/token
                refreshUrl: https://socialmedia.com/
                    oauth/refresh
            scopes:
              read: Read access to posts
              write: Write access to posts

```

## IV. DEFINITIONS

### A. Security coverage granularity

OpenAPI allows the application of security schemes at different levels within the API documentation, providing flexibility in how to enforce authentication and authorization across the API. Once defined, these security schemes can be applied with the `security` keyword, either globally to the entire API or to individual operations (like GET, POST, PUT, etc.) of specific endpoints. In this section, we introduce the possible alternatives for decisions taken regarding the level at which API designers decide to apply security mechanisms. In Tab. I, we summarize the different granularity levels adopted when documenting security in OAS, which we explain in details in the rest of this section. The color coding is selected to reflect the gradual transition from ● (a specification where the security component is completely absent) to ● (a security-rich documentation combining both local and global security properties to cover all the operations).

1) *Globally secured (GSec)*: A security scheme can be applied to the entire API. This means that every operation in the API requires clients to adopt the specified security scheme. This is defined by including the `security` section at the root level of the OpenAPI document (Listing 2). In the class *Globally Secured* (● GSec) we include only the commits of API specifications that exclusively enforce security at the global level.

More precisely, we define the  $\text{Global}(api)$  detector:

$$\text{Global}(api) = \begin{cases} \text{True} & \text{if a security scheme is referenced at the} \\ & \text{root of the } api \text{ document} \\ \text{False} & \text{otherwise} \end{cases}$$

We define the  $\text{Local}(p)$  detector as:

$$\text{Local}(p) = \begin{cases} \text{True} & \text{if } \exists m \in \text{methods}(p) \text{ sec}(p, m) = \text{True} \\ \text{False} & \text{if } \forall m \in \text{methods}(p) \text{ sec}(p, m) = \text{False} \end{cases}$$

where  $\text{sec}(p, m)$  returns True if a security scheme is explicitly defined for the specific method  $m$  in a given path  $p$ ; otherwise, it returns False.

An  $api$  can be classified as *globally secure* if and only if:

- 1)  $\text{Global}(api)$  is True.
- 2) For all paths  $p$  of the  $api$ ,  $\text{Local}(p)$  is False.

Formally, the rule can be expressed as:

$$(\text{Global}(api)) \wedge (\forall p \in \text{paths}(api), \neg \text{Local}(p)) \implies \text{GSec}(api) \quad (1)$$

In the example of Listing 2, OAuth2 is applied globally with read and write scopes, indicating that all operations require OAuth 2.0 authentication with at least one of these scopes.

2) *Locally Secured*: Security schemes can also be applied at the operations level. This fine-grained specification of different authentication requirements for specific API endpoints is done by including a security section within the operation definition. This approach is useful when only specific operations require authentication or when different operations require different levels of access. In the example (Listing 2), only read access

**Table I:** Security Granularity Level Classification

Class	Acronym
<span style="color: green;">●</span> Globally Secured and Fully Locally Secured	GFLSec
<span style="color: limegreen;">●</span> Fully Locally Secured	FLSec
<span style="color: lightgreen;">●</span> Globally Secured and Partially Locally Secured	GPLSec
<span style="color: yellow;">●</span> Globally Secured	GSec
<span style="color: orange;">●</span> Partially Locally Secured	PLSec
<span style="color: brown;">●</span> Not Secured	NoSec
<span style="color: lightblue;">●</span> Global security applied but no paths	GSec-No-Paths
<span style="color: blue;">●</span> Only component defined but no paths	C-No-Paths
<span style="color: red;">●</span> Missing Security Component Definition	UndefSec

**Listing 2:** Example of security scheme application

```
openapi: 3.0.3
info:
  ...
security:
  - OAuth2: [read, write]
paths:
  /posts:
    get:
      security:
        - OAuth2: [read]
    post:
      security:
        - OAuth2: [write]
```

is required for the GET operation, while the POST operation requires write access. Since some paths can be secured while others can have no local security defined, we distinguish:

● *Fully Locally Secured (FLSec)*: A commit of an API specification is classified as *Fully Locally Secured* if all paths have at least one method with an explicitly defined local security scheme. This way, the entire API is covered without relying on global security settings.

Using the previously defined detectors:

$$(\neg \text{Global}(api)) \wedge (\forall p \in \text{paths}(api), \text{Local}(p)) \implies \text{FLSec}(api) \quad (2)$$

● *Partially Locally Secured (PLSec)*: A commit of API specification is classified as *Partially Locally Secured* if at least one path has a method with a security scheme applied to it, while at least another different path does not have such a method. While some paths in the API enforce security locally, others have no security defined, not even globally.

$$(\neg \text{Global}(api)) \wedge (\exists p, p' \in \text{paths}(api), p \neq p', \text{Local}(p) \neq \text{Local}(p')) \implies \text{PLSec}(api) \quad (3)$$

3) *Hybrid Combinations*: API descriptions may combine global and local security schemes to describe different security policies across various paths and operations. Two main combinations arise from the interaction between globally and locally applied security schemes.

● *Globally Secure and Fully Locally Secured (GFLSec)*: In this case, the API applies a global security scheme at the root

level of the OAS document while also defining local security schemes at every path:

$$\begin{aligned} & \text{Global}(api) \wedge (\forall p \in \text{paths}, \text{Local}(p)) \\ & \implies \text{GFLSec}(api) \end{aligned} \quad (4)$$

● *Globally Secure and Partially Locally Secured (GPLSec)*: In this case, there exist some paths where a local specification of security properties is missing. These paths rely solely on the global security scheme.

$$\begin{aligned} & \text{Global}(api) \wedge \\ & (\exists p, p' \in \text{paths}(api), p \neq p', \text{Local}(p) \neq \text{Local}(p')) \\ & \implies \text{GPLSec}(api) \end{aligned} \quad (5)$$

4) *Lack of security definitions*: ● *Not Secured (NoSec)*: A commit of an API specification is classified as *Not Secured* if neither global nor local security schemes are defined for any of its paths. This means that – although a security component definition is present – the API description never uses it as it does not document any form of security enforcement, making all paths open to unauthenticated and unauthorized access.

$$\begin{aligned} & (\neg \text{Global}(api)) \wedge (\forall p \in \text{paths}(api), \neg \text{Local}(p)) \\ & \implies \text{NoSec}(api) \end{aligned} \quad (6)$$

● *Global Security Applied but Without Path in Specification (GSec-No-Paths)*: In this configuration, the API defines a global security scheme at the root level of the specification, but there are no paths listed in the API description. This means that while a global security policy exists, no specific operations or endpoints are available in the specification to actually use the globally declared security schemes.

$$\text{Global}(api) \wedge (\text{paths}(api) = \emptyset) \implies \text{GSec-No-Paths}(api)$$

● *Only Component Defined Without Paths in Specifications (C-No-Paths)*:

Also in this case the API specification defines security components, but these cannot be referenced from any path, because there are no paths listed.

$$(\neg \text{Global}(api)) \wedge (\text{paths}(api) = \emptyset) \implies \text{C-No-Paths}(api) \quad (7)$$

● *Missing Component Definition (UndefSec)*:

The last category includes any API specification in which there are no security components defined at all.

### B. Security Coverage Metrics

To quantitatively evaluate the proportion of security coverage [26], for each *api* specification, we count the *number of secured operations SO* covered by a security scheme applied either globally, locally, or both. The *security coverage* of the API operations *SOC* is defined as:

$$\text{SOC}(api) = \frac{\text{SO}(api)}{\text{TO}(api)}$$

where *TO* is the total number of operations described in the specification.

**Table II:** APIs/Commits with or without security components

APIs	270 564	100%
APIs Without Security Component	166 585	62%
APIs With Security Component	104 606	39%
→ OAS 2.0	46 305	17%
→ OAS 3.0	58 301	21%
→ APIs with both OAS 2.0 and OAS 3.0 descriptions	697	1%
APIs With and Without Security Component	6 242	2%
APIs With Security Component	97 737	36%
Commits	915 988	100%
Commits Without Security Component	482 700	53%
Commits With Security Component	433 288	47%
→ OAS 2.0	144 771	16%
→ OAS 3.0	288 517	32%

**Table III:** Security Schemes in APIs and Commits

Security Scheme	#APIs		#Commits	
	OAS 2.0	OAS 3.0	OAS 2.0	OAS 3.0
apiKey	27 318	27 329	87 393	127 594
oauth2	20 813	19 444	52 133	83 390
http	-	26 096	-	140 920
basic	12 257	-	39 431	-
openIdConnect	-	895	-	4 241
mutualTLS	-	13	-	14

### C. API Evolution and Security

To study the introduction of security documentation features along the lifecycle of an API, we collected the entire history of the OpenAPI artifacts as reflected in its git commits.

We define the API history  $H(api) := \{c, c \in \text{git}(api)\}$  as the ordered set of commits of the API specification found in its GitHub repository.  $t(c)$  indicates the commit timestamp.

The results of our analysis can be read at the commit level, where every API specification is classified and compared individually, or at the API level, where the history of each API is considered and the classification of its commits is aggregated accordingly.

### V. RQ1: SECURITY COMPONENT DEFINITION

In Tab. II, we present an overview of the dataset, where we classify APIs and their commits based on whether a security component has been detected or not. For APIs and commits with a security component we also further split the classification depending on the specification language version. 38.6% of the APIs have in their history at least one commit where the specification contains a security component, totaling 433 288 commits, while 36% of the API consistently include security component definitions in all of their commits. The majority of APIs (61.6%) however never had a defined security component in their whole commit history. If we compare Swagger (OAS2.0) vs OpenAPI (OAS3.0), we find a larger set of artifacts that use security in the more recent version of the API specification language.

The different types of security schemes that have been defined in our dataset are listed in Table III where we consider the types of security schemes in both APIs and commits. We note that developers have the tendency to adopt the security

scheme `apikey`, which appears in 214 987 commits of 54 647 APIs. There is also a tendency towards the frequent use of `oauth2` in both OAS2.0 and OAS3.0 versions. Still, the `HTTP basic` scheme was only the third choice for APIs documented with OAS 2.0, but after it was renamed to `http` it became the most frequent scheme among the commits of OAS 3.0 descriptions. For the rest of the schemes introduced in OAS 3.0, such as `OpenIdConnect` or `mutualTLS`, we notice significantly less uptake.

## VI. RQ2: GRANULARITY LEVELS

### A. Granularity levels of commits using OAS 2.0 vs. OAS 3.0

Once the decision to introduce security components in the API documentation has been made for 47% of the commits (Tab. II), the next decision involves whether the security component should be applied globally to the entire API or locally to specific API operations. In Tab. IV we present a detailed classification of those commits based on the classes defined in Section IV-A.

The smallest classes include 0.3% (C-No-Paths) and 0.5% (GSec-No-Paths) of the commits, which include a security component but do not list any endpoint paths. These might represent documentation templates with default security settings, or API specifications still at the initial stage of the API documentation evolution. The majority of commits (66%) presents an API whose entire surface is protected by some security scheme. This is achieved following different approaches. For example, the most common one (32%) introduces security globally and uniformly across the whole API (GSec), while 27% annotate each and every endpoint with a security property (FLSec). Some API designers instead choose to only partially cover the API endpoints with security annotations (26%). In Section VII, we study more in detail this class by measuring exactly how many operations do not use security.

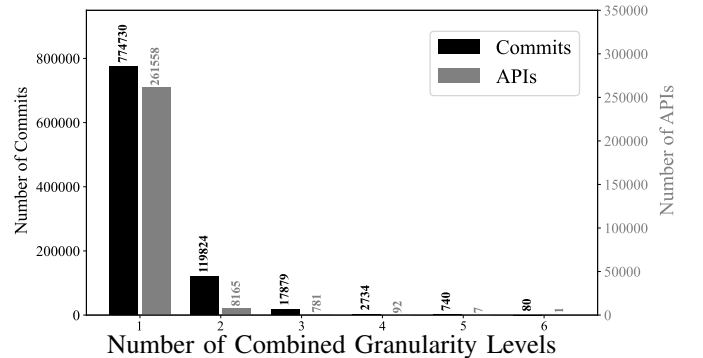
When comparing how the constraint of using OAS 2.0 or OAS 3.0 to document the API impacts the granularity level decision, we see that – relatively speaking – partial coverage is more frequent with OAS 2.0 (30% vs. 24%), while security is introduced globally in 41% of the commits for OAS 3.0, and only in 32% for OAS 2.0. The complete lack of security coverage (NoSec) is on a similar level for both versions.

### B. APIs Classification by Granularity Level

We classify the API by the different combinations of granularity levels in their histories. In Fig. 1, we plot the distribution of the number of APIs and the number of API commits across the number of combined granularity levels. More in detail, in Tab. V we list the commonly used granularity level that has been used in at least one commit in the API history. While many APIs fall under the ● UndefSec class because in all their commits no security component was ever defined, we found that 6 242 APIs (totaling 96 093 commits) have commits that have security components and other commits that fall into the other categories where a security component exists and either not applied ● (1 169 APIs, 15 336 commits), or applied at some level ●●●● (4 732 APIs, 72 122 commits). There

**Table IV:** Classification of API Commits Based on Security Granularity Levels

Class	OAS 2.0		OAS 3.0		OAS 2.0 & 3.0
	#Commits	%	#Commits	%	#Commits
<span style="color:green">●</span> GFLSec	1 120	1	8 633	3	9 753
<span style="color:yellow">●</span> FLSec	40 809	28	77 676	27	118 485
<span style="color:lightgreen">●</span> GPLSec	3 539	2	14 895	5	18 464
<span style="color:yellow">●</span> GSec	41 895	29	95 756	33	137 651
<span style="color:orange">●</span> PLSec	43 790	30	68 759	24	112 549
<span style="color:orange">●</span> NoSec	11 598	8	21 580	7	33 178
<span style="color:lightblue">●</span> GSec-No-Paths	830	1	875	-	1 705
<span style="color:blue">●</span> C-No-Paths	1 160	1	343	-	1 503
<b>Subtotal</b>	<b>144 771</b>		<b>288 517</b>		<b>433 288</b>
<span style="color:red">●</span> UndefSec	228 881	47	253 819	53	482 700
<b>Total</b>	<b>373 652</b>		<b>542 336</b>		<b>915 988</b>



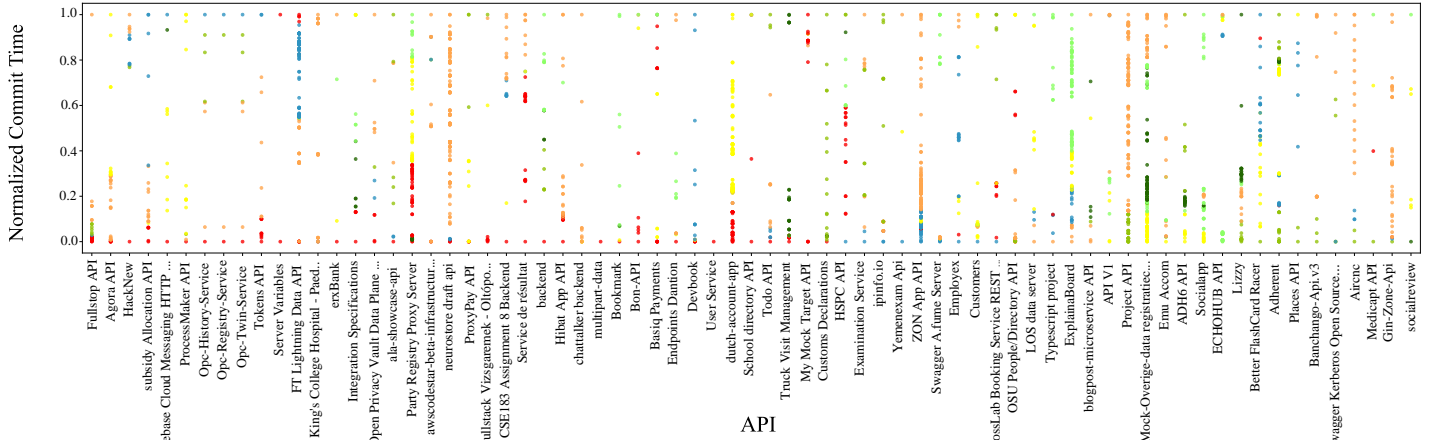
**Figure 1:** Number of APIs (and the total number of commits in these APIs) combining multiple granularity levels defined in Table I. Table V shows the number of APIs and commits for the most common combinations.

exist also 341 APIs (totaling 8 635 commits) with a history in which there exists at least one commit with no defined security component ●, at least another one where a security component is defined but never applied at any level ●, then at least another commit where the defined security scheme is locally applied either partially ● or fully ●.

These results indicate that most APIs remain stable according to the granularity level classification as their maintainers never change the decision of where and how security should be documented. The most frequent changes involve commits completely lacking security documentation and commits with security documented at some level.

### C. Example APIs: granularity level transitions

In Fig. 2 we show 73 examples of APIs that showcase a history with changing decisions regarding their granularity levels. The APIs have been selected as representatives of the classes identified in Fig. 1 with more than 3 granularity level combinations, sorted by their occurrence in distinct APIs. For each API, by coloring the granularity level of the corresponding sequence of commits, we illustrate whether the granularity level decision remains stable from the one taken for the initial commit (placed at the bottom) or changes through the lifetime of the API until the most recent commit (on the top row).



**Figure 2:** Example APIs transitioning between at least four different granularity levels during their evolution history

**Table V:** Classification of APIs Based on Security Granularity Levels Detected in Their History. The first column “#” counts the number of combined granularity levels.

#	Granularity Level Classification	#APIs	#Commits
1	Undefined	166585	443010
1	PLSec	29461	80163
1	GSec	28838	108671
1	FLSec	22176	101941
1	NoSec	7569	19996
1	GPLSec	2858	11011
1	GFLSec	1979	7315
2	Undefined, PLSec	1933	25624
2	Undefined, GSec	1626	29834
1	GSec-No-Paths	1318	1602
2	Undefined, NoSec	1169	15336
2	PLSec, FLSec	842	16019
1	C-No-Sec	774	1021
2	Undefined, FLSec	572	5265
2	NoSec, PLSec	495	4920
2	GSec, GPLSec	211	4832
2	NoSec, GSec	187	4395
2	Undefined, GPLSec	157	2308
2	NoSec, FLSec	157	1356
2	PLSec, GSec	132	1761
3	Undefined, PLSec, FLSec	128	2415
3	Undefined, NoSec, PLSec	116	2541
Total of 22 above classes with more than 100 APIs			891336
Total of remaining 86 classes			24652

Each commit is positioned on the Y axis according to the “normalized commit time“ of the API  $t_i = \frac{t(c_i) - t(c_0)}{t(c_N) - t(c_0)}$  where  $t(c_i)$  is the timestamp of the commit  $c_i$ . This projection preserves the commit order and makes it possible to compare APIs with histories of different durations. The APIs are sorted based on their initial granularity level.

We observe that the transition between the lack of security documentation and the presence of security documentation is not one-directional, as there are many cases of APIs which gain security documentation as they evolve, while we also have detected fewer APIs which shed their security documentation.

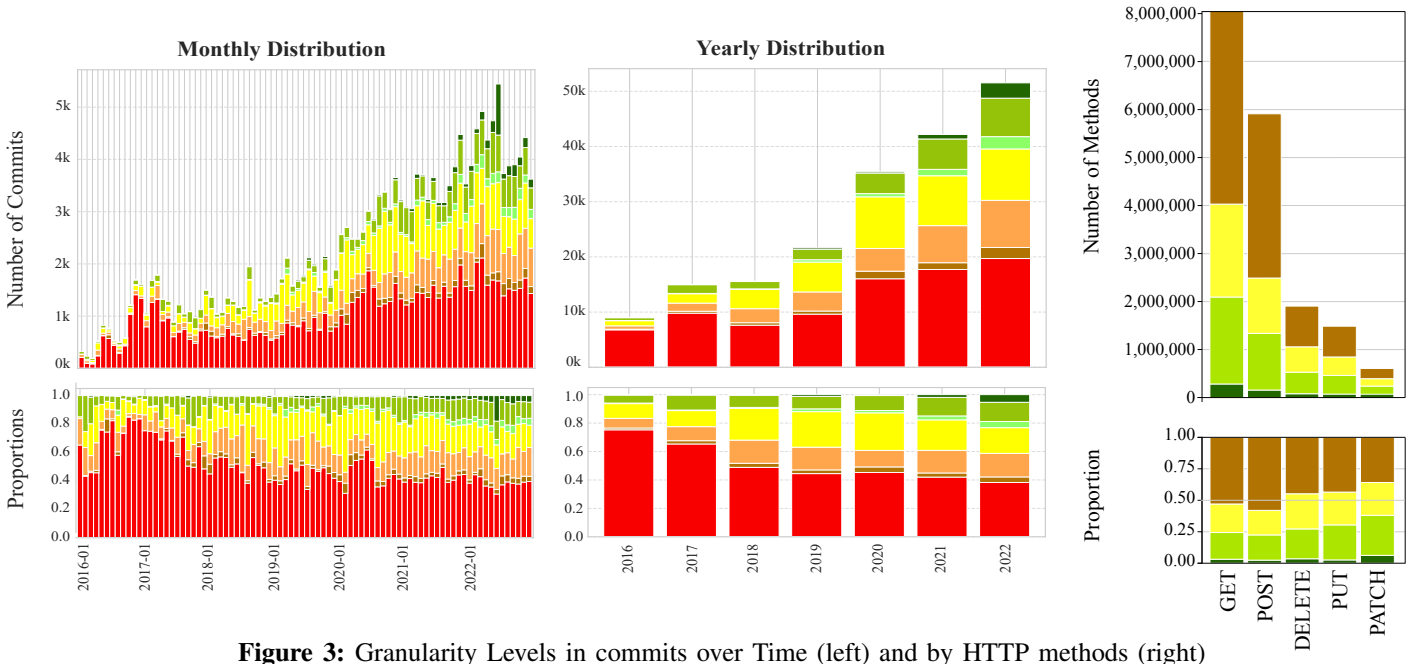
#### D. Granularity Levels over Time and API Size

To study whether and how the commit time affects the granularity level decision we have segmented the classification of the commits based on the month and year of the commit timestamp. The resulting monthly and yearly distributions have been visualized on the left side of Fig. 3, both in absolute (top) and relative (bottom) terms. The commits from 2023 have not been included in the visualization as the dataset did not include a complete crawl of GitHub for that year. We observe a slight increase over the year of the proportion of commits which include security documentation. In particular, the classes GFLSec and FPLSec representing an overlap of globally defined security with its local redefinition are more frequently detected in recent years.

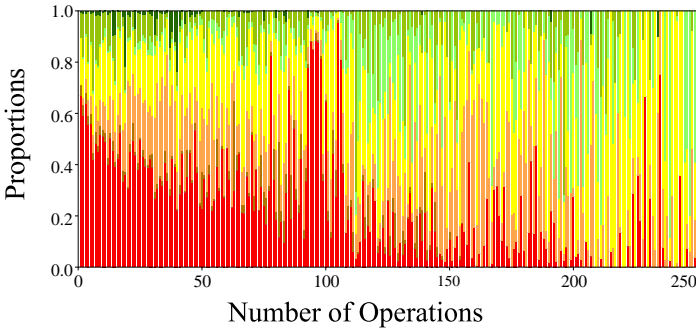
Fig. 4 precisely maps the impact of the API size on the granularity level decision. The plot visualizes the probability for a commit of the given number of operations to be classified in one of the possible granularity levels. With some exceptions, we observe that larger APIs tend to include security documentation more often, while more than 50% of commits including fewer than 5 operations completely lack security components in their documentation.

#### E. Granularity Levels by HTTP methods

We observed how each operation has been secured depending on its HTTP method. Note that the classification has been applied at the finest possible level of granularity: the PLSec class is not applicable at the method level. Fig. 3, shows that GET – a read-only method – is the most present one but only secured in 50% of the cases. Operations which mutate the state of the API (i.e., PUT, PATCH, or DELETE) are significantly less frequent but also more often require clients to be authenticated and authorized. Compared to the other ones, the POST method – often used within RPC-style APIs – is the one which lacks security documentation most frequently.



**Figure 3:** Granularity Levels in commits over Time (left) and by HTTP methods (right)



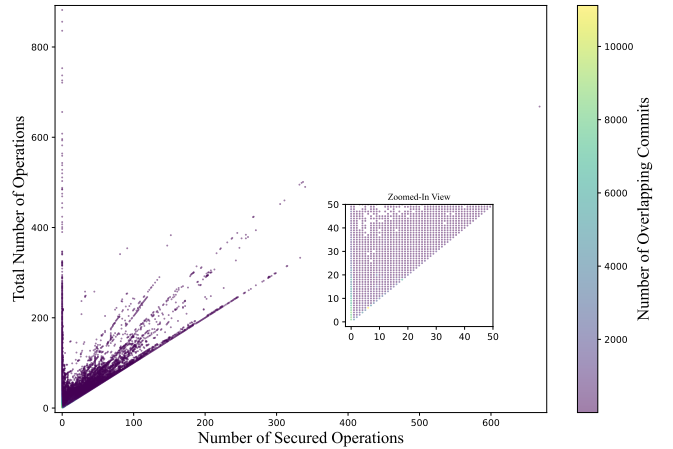
**Figure 4:** Granularity Level Probability by API Size

## VII. RQ3: SECURITY COVERAGE

### A. API Size-Security Coverage Correlation in Commits

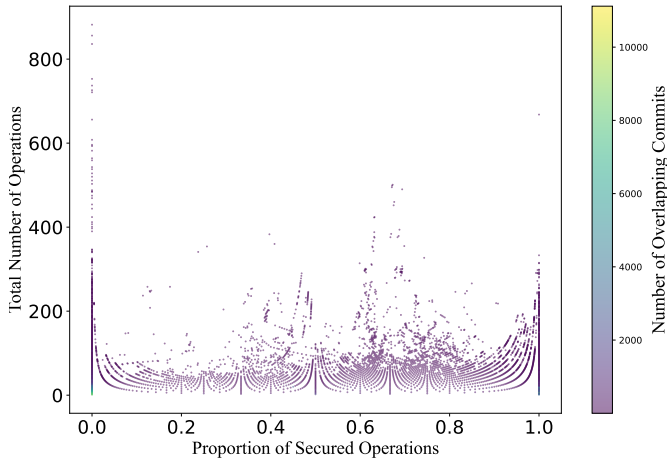
To measure the total number of secured operations  $SO$  and the proportion of covered endpoints  $SOC$  in a specific API commit and how they correlate with the API size (Total number of operations  $TO$ ) at that moment, we draw the density scatter plots in Figs. 5 and 6.

Fig. 5 shows that many specifications still maintain a relatively low number of secured operations  $SO$ , even as the total operation count increases. And, Fig. 6 reveals that the proportion of secured operations  $SOC$  varies widely between different API sizes, with no clear linear relationship between the proportion of secured endpoints and the total API size. This distribution shows clustered points at certain proportions, suggesting that security coverage is inconsistent and often independent of API size, with many specifications having either very low (in the extreme case, completely missing at 0%) or very high (and often fully covered at 100%) security coverage, regardless of the total number of operations  $TO$ .

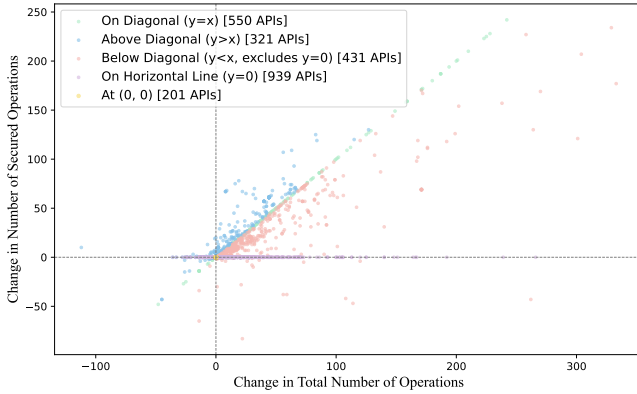


**Figure 5:** Density Scatter Plot: Number of Secured Operations vs. API Size (Total Number of Operations)

We include both plots, as the first (Fig. 5) shows that the number of secured operations is bound by the total number of operations in the API. In addition, three clusters of APIs stand out: those completely lacking security coverage aligned on the  $SO = 0$  vertical line, those fully covered aligned on the  $SO = TO$  identity line. The third group is found somewhere in the middle, with partial coverage. As APIs get smaller, the scatter plot gets crowded - we use a density plot to reveal that most of the specifications are located on the  $0 \leq TO \leq SO < 10$  region. Fig. 6 illustrates this in more detail this distribution by projecting the same scatter plot over the  $X = SO/TO$  and the same  $Y = TO$  axis. In it, the three clusters become more clearly distinct, with  $SOC = 0$ ,  $0 < SOC < 1$ , and  $SOC = 1$ . We also note that due to the large size of the dataset, there are artifacts of small sizes ( $TO < 50$ ) that cover the entire range of possible security coverage values.



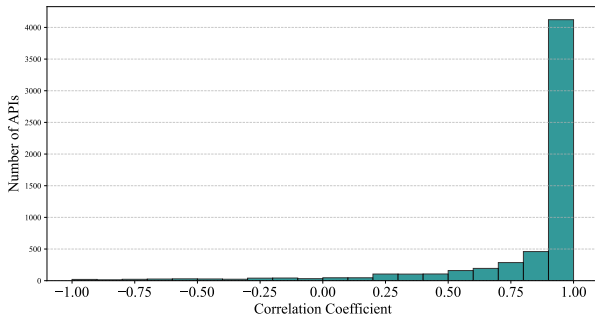
**Figure 6:** Security Coverage (Proportion of Secured Operations) vs. API Size (Total Number of Operations)



**Figure 7:** Secured Operation Change ( $\Delta SO$ ) depending on the API Size Change ( $\Delta TO$ ) across the whole API history for every API with more than 10 commits

### B. Size-Security Coverage changes across entire API history

We consider the changes across the entire history of the API, comparing the  $SO$  and  $TO$  metrics of the last commit against the first commit to study whether APIs which grow larger or shrink, also maintain, degrade, or improve their security coverage. In Fig. 7 we compare the  $\Delta TO$  (change of number



**Figure 8:** Distributions of the correlation between the number of operations  $TO$  and the number of secured operations  $SO$  in API commits excluding APIs where the  $(TO, SO)$  is constant during the whole history composed of at least 50 commits

operations) against the  $\Delta SO$  (change of number of secured operations) for each API with more than 10 commits. Overall, the tendency is towards positive growth, i.e., APIs tend to grow larger and also increase the number of security endpoints. We can distinguish different clusters: 1) Static APIs ( $x = 0, y = 0$ ) which never change their size or the number of secured operations; 2) APIs whose size variation is identically reflected in the number of operations with security documentation ( $y = x$ ); 3) APIs where the change in size is not reflected at all in the number of secured operations ( $y = 0$ ); 4) APIs which degrade the security coverage as they grow, since the number of secured operations does not grow as much as their size ( $y < x$ ); 5) APIs which improve the security coverage ( $y > x$ ). We show the number of APIs present in each cluster in the legend of Fig. 7. There are some very rare cases in which the API growth corresponds to a net decrease in the number of secured operations, or in which the API size decreases but the number of secured operations increases. We observe only 165 cases in which the API size remains the same ( $x = 0$ ) but the number of secured operations changes ( $y \neq 0$ ).

### C. Changes of security coverage over time

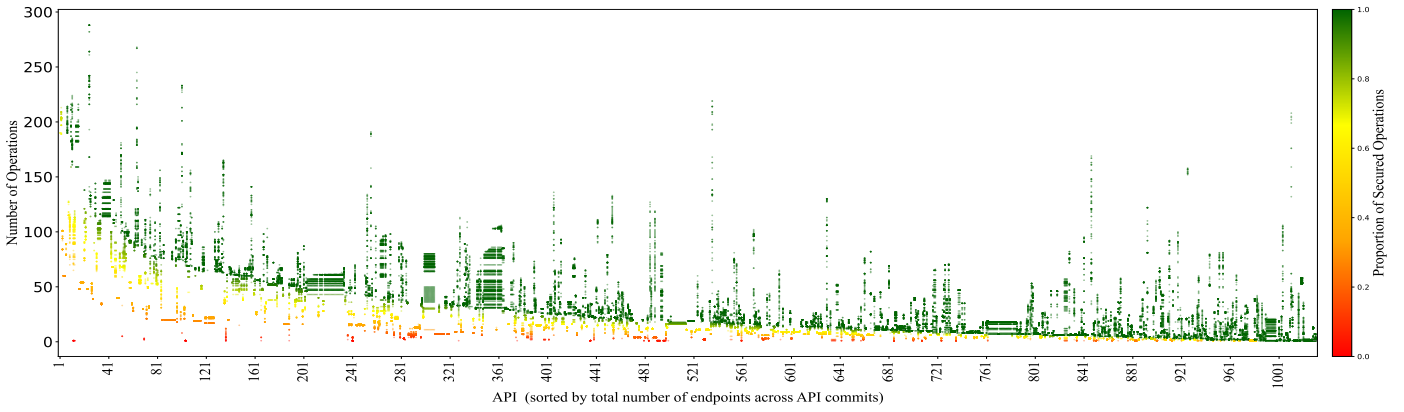
Tab. VI shows that APIs with more than 10 commits exhibit modest improvements in security coverage over time, with an average increase of 9% from the first to the last commit. While some APIs achieve full coverage, many start and remain unsecured, as reflected by the median initial and final security coverage of 0% and 34%, respectively. Incremental changes between commits are minimal, with a mean of just 1%, and the overall daily evolution is slow, as indicated by an average age of 445 days while a median of 0% in the average coverage change rate per day. This highlights a level of stability in API security coverage and a generally slow pace of improvement. We also observe rare cases where the API becomes fully secured (or fully unsecured) within the timeframe of one day.

When computing the correlation between  $SO$  and  $TO$  along the history of each API with at least 50 commits, we obtain the histogram of Fig. 8. Here we notice a strong correlation

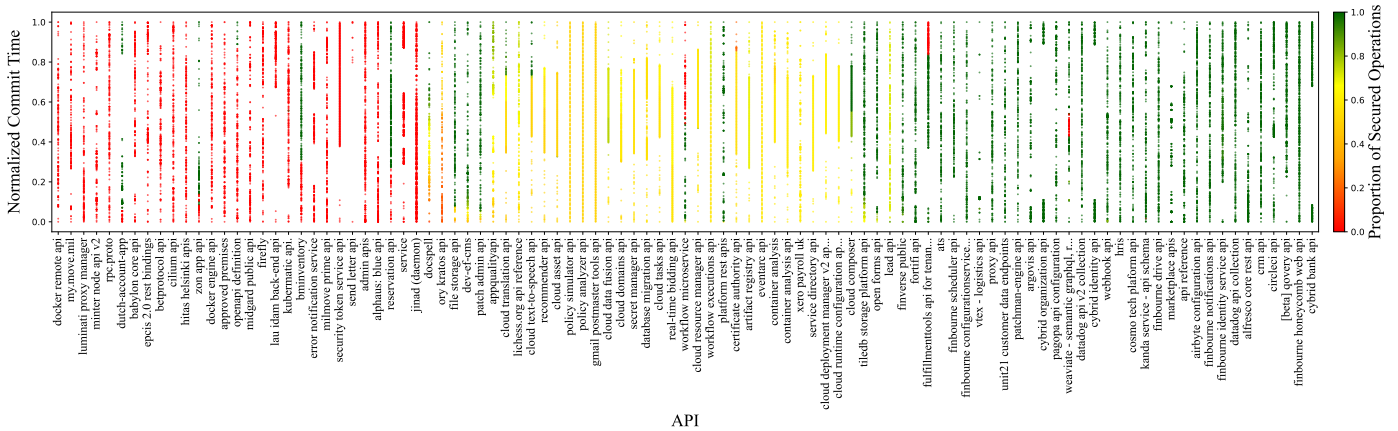
**Table VI:** Statistical Analysis of Total number of Operations ( $TO$ ), Number of Secured Operations ( $SO$ ), Operation Security Coverage ( $SOC$  (%)) and their Variations over the Evolution of APIs with more than 10 commits ( $N > 10$ )

Metric	Mean	Median	Std Dev	Min	Max
$TO$	581.61	234.00	1171.24	3.00	29111.00
$\Delta TO: TO(c_N) - TO(c_0)$	9.01	4.00	18.30	-644.00	346.00
$\delta TO: TO(c_{i+1}) - TO(c_i)$	0.44	0.20	0.95	-26.05	30.27
$SO$	302.45	30.00	924.42	0.00	23052.00
$SO(c_0)$	6.51	0.00	17.04	0.00	668.00
$SO(c_N)$	11.80	2.00	23.50	0.00	340.00
$\Delta SO: SO(c_N) - SO(c_0)$	5.29	0.00	16.14	-644.00	294.00
$\delta SO: SO(c_{i+1}) - SO(c_i)$	0.25	0.00	0.82	-21.47	22.62
$SOC$ (%)	47	36	47	0	1
$SOC(c_0)$ (%)	37	0	46	0	1
$SOC(c_N)$ (%)	46	34	46	0	1
$\Delta SOC: SOC(c_N) - SOC(c_0)$ (%)	9	0	32	-1	1
$\delta SOC: SOC(c_{i+1}) - SOC(c_i)$ (%)	1	0	2	-11	12
Lifetime (days)	444.40	298.00	460.45	0.00	2962.00
Daily Rate of Change of $SO$	0.11	0.00	0.94	-12.00	34.00
Daily Rate of Change of $SOC$ (%)	0	0	4	-1	1





**Figure 9:** Evolution of API Size (Number of Operations  $TO$ ) in each commit colored by security coverage ( $SOC$ ). APIs sorted by the total number of operations across all API commits



**Figure 10:** Evolution of security coverage during the lifetime of APIs with more than 100 commits, sorted by the security coverage in the first commit

between API size and its security coverage for the vast majority of APIs. We purposefully exclude from the histogram the APIs which do not change  $SO$  and  $TO$  during their history, since they would also contribute to the strong correlation.

We illustrate more in detail this relationship with Fig. 9 representing a side-by-side comparison of the history of a sample of more than 1000 APIs selected due to having more than 50 commits in their history. Each point along the y-axis corresponds to a commit in the API history, vertically placed according to the API size ( $TO$ ) and colored according to the security coverage ( $SOC$ ). While the previous scatter plots (Figs. 5 and 6) mix together every commit of every API in the dataset, here we study the evolution of each API separately, but also visually compare the relationship between size and security coverage across APIs with different histories with an appropriate ordering of the APIs on the X axis.

What stands out from the visualization of Fig. 9 is that when analyzing the history of each API separately, the security coverage appears to increase with the API size – as witnessed by the strong correlation of Fig. 8. The visualization highlights the lack of security coverage in commits of decreasingly small sizes, while above a certain size threshold, the APIs are fully covered with security documentation. This visualization, however, does not consider the order of commits in time, thus

it is not possible to observe whether security documentation is added as APIs grow larger, or removed as they grow smaller.

The time dimension is introduced on the Y axis of Fig 10, where again different API commit histories are plotted side by side. To reduce clutter, we filter 172 APIs that have more than 100 commits in their histories and sort them by the  $SOC(c_0)$  amount of security coverage measured in the first commit. We see APIs that initially have high security coverage on the right, and APIs initially lacking security documentation on the left. The visualization confirms the tendency towards stability with many APIs carrying security documentation or consistently lacking security documentation throughout their entire lifecycle. However, we also see examples of APIs in which security documentation is added after the first commit, or removed at some point, and even added again.

## VIII. DISCUSSION

The results presented in the previous sections give an in depth view over how two versions of the OpenAPI standard API description language are used to document security information in a large collection of artifacts mined from open source repositories. We classified individual commits as well as compared how different API evolve w.r.t. how security aspects are documented along their history.

RQ1 Being an optional validity requirement of an OAS document, security components are detected in only 38.6% of APIs (104,606 APIs), comprising 433,288 commits. APIs documented with OpenAPI 3.0 (OAS 3.0) demonstrate a higher prevalence of security components, with 32% of commits (288,517) including security compared to 16% (144,771) in Swagger 2.0 (OAS 2.0). However, 61.6% of APIs never included a security component throughout their history. The lack of security documentation does not imply that the API implementation itself does not actually make use of any authorization and authentication mechanisms, but only that these have not been explicitly described in the corresponding API specification.

The higher percentage of specifications with security documentation in OAS 3.0 compared to OAS 2.0 can be attributed to the enhancements introduced in OAS3.0. These improvements include increased modularity in documentation and better support for defining and organizing security schemes, such as “OAuth2”, as detailed in Section II.

RQ2 The Security granularity level classification reveals that while 32% of commits follow a simpler strategy by applying security globally (● GSec), 27% annotate every endpoint with local security (● FLSec), allowing developers to implement custom configurations to address particular requirements or unique use cases even to address fine-grained access control. Hybrid strategies, such as the combination of global and partial local security (● GPLSec or ● GFLSec), are less frequent but increasingly used in recent years. Partial coverage is more frequent in OAS 2.0 (30%) than in OAS 3.0 (24%). Granularity levels remain stable across most of API histories, with cases of changes occurring primarily between commits lacking security and those adopting it (in both directions).

RQ3 When studying individual artifacts, the API size does not correlate with security coverage, as there are clearly three clusters of commits, the ones with no security coverage at all, the ones with full coverage, and the intermediate ones. When analyzing the history of each API, however, it appears that APIs tend to include security documentation as they grow larger. The high correlation of a large subset of APIs between their size and security coverage maintained over a large number of commits can be explained by the impact of the global security definition, which – when introduced – brings the two metrics to the same value. We also observe a general stability trend when considering the security coverage over time: when adding or removing endpoints from an API, the proportion of operations covered with security documentation remains similar.

## IX. THREATS TO VALIDITY

1) *Construct Validity*.: The presence of security-related documentation is detected with standard OpenAPI constructs. The paper focuses on studying the decision on how and where to introduce security schemes in the API specification, both qualitatively and quantitatively [20]. Other approaches to detect and study how security is documented in OpenAPI may be possible, e.g., based on natural language processing

of textual descriptions [14] or the deployment context of the API by analyzing API gateway security settings.

2) *Internal Validity*.: Building the classifications required a systematic analysis to detect the presence or absence of the security scheme implemented under the Open API standard in each commit. While the detectors have been extensively tested and applied to a very large number of artifacts, they are assumed to run against standard compliant OpenAPI specifications. While we exclude by construction the presence of false positives, it may be possible that the results underestimate the APIs classified to feature security-related documentation due to true negatives. As mentioned above, the APIs classified as lacking security documentation may contain such information with a non-standard representation or do so using natural language descriptions.

3) *External Validity*.: The study conducted for a wide selection of APIs hosted in public GitHub repositories may consider APIs that are no longer actively maintained or have descriptions that are not yet fully developed and released. This concern affects the generalization of the study results.

## X. CONCLUSION AND FUTURE WORK

This study analyzed 915 988 OpenAPI descriptions from 270 564 distinct APIs, focusing on detecting security documentation practices and their evolution. Although a substantial portion of APIs (62%) lack any security documentation, those that do include security components tend to favor global security schemes, applying protection uniformly across all endpoints. OpenAPI 3.0 specifications show a greater adoption of security documentation than in Swagger 2.0. However, even among APIs with extensive histories, security coverage improvements are modest, with many APIs starting and remaining largely without security documentation.

Thanks to the large dataset analyzed, we could systematically establish and empirically validate a detailed and complete classification of the granularity levels of adoption of security schemes for each commit, path, and operation. Assessing adoption trends in API documentation provided a broader perspective on how designers are applying the OpenAPI standard to document their security-related design decisions.

Given some results observed (e.g., the widespread lack of security documentation for critical operations making use of HTTP methods such as POST or DELETE), this paper motivates further work in further developing API model analysis tools that can not only highlight but also correct such gaps in security documentation as they may indicate potential vulnerabilities of the corresponding API. We are also exploring how documented security scheme classification levels relate to the purpose of the endpoint in the API, whether the endpoint exposes access to sensitive data or critical actions, encompassing both business or system management endpoints.

## ACKNOWLEDGEMENTS

This work was supported by the SNF with the API-ACE project 184692.

## REFERENCES

- [1] OpenAPI Initiative. <https://www.openapis.org/>.
- [2] Esi Adeborna and Kenneth K Fletcher. An empirical study of web API quality formulation. In *Proc. 17th International Conference on Services Computing (SCC)*, pages 145–153. Springer, 2020.
- [3] Sultan S Alqahtani, Ellis E Eghan, and Juergen Rilling. Recovering semantic traceability links between apis and security vulnerabilities: An ontological modeling approach. In *Proc. International Conference on Software Testing, Verification and Validation (ICST)*, pages 80–91. IEEE, 2017.
- [4] David Bermbach and Erik Wittern. Benchmarking web api quality-revisited. *Journal of Web Engineering*, 19(5-6):603–646, 2020.
- [5] Carmen Cheh and Binbin Chen. Analyzing openapi specifications for security design issues. In *Proc. Secure Development Conference (SecDev)*, pages 15–22. IEEE, 2021.
- [6] Hsiao-Jung Chen, Shang-Pin Ma, and Hsueh-Cheng Lu. Collaborative security annotation and online testing for web apis. In *Proc. International Conference on e-Business Engineering (ICEBE)*, pages 9–15. IEEE, 2021.
- [7] Mengsu Chen, Felix Fischer, Na Meng, Xiaoyin Wang, and Jens Grossklags. How reliable is the crowdsourced knowledge of security implementation? In *Proc. 41st International Conference on Software Engineering (ICSE)*, pages 536–547. IEEE/ACM, 2019.
- [8] Josué Alejandro Díaz-Rojas, Jorge Octavio Ocharán-Hernández, Juan Carlos Pérez-Arriaga, and Xavier Limón. Web api security vulnerabilities and mitigation mechanisms: A systematic mapping study. In *Proc. 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 207–218. IEEE, 2021.
- [9] Mario Dudjak and Goran Martinović. An api-first methodology for designing a microservice-based backend as a service platform. *Information Technology and Control*, 49(2):206–223, 2020.
- [10] Daniel Fett, Ralf Küsters, and Guido Schmitz. A comprehensive formal security analysis of oauth 2.0. In *Proc. ACM SIGSAC conference on computer and communications security*, pages 1204–1215, 2016.
- [11] Daniel Fett, Ralf Küsters, and Guido Schmitz. The web sso standard openid connect: In-depth formal security analysis and security guidelines. In *Proc. 30th Computer Security Foundations Symposium (CSF)*, pages 189–202. IEEE, 2017.
- [12] Pascal Gadiant, Mohammad Ghafari, Marc-Andrea Tarnutzer, and Oscar Nierstrasz. Web apis in android through the lens of security. In *Proc. 27th international conference on software analysis, evolution and reengineering (SANER)*, pages 13–22. IEEE, 2020.
- [13] Patric Genfer, Souhaila Serbout, Georg Simhandl, Uwe Zdun, and Cesare Pautasso. Understanding security tactics in microservice apis using annotated software architecture decomposition models – a controlled experiment. *Empirical Software Engineering*, 2025. (to appear).
- [14] César González-Mora, Cristina Barros, Irene Garrigós, Jose Zubcoff, Elena Lloret, and Jose-Norberto Mazón. Improving open data web API documentation through interactivity and natural language generation. *Computer Standards & Interfaces*, 83:103657, 2023.
- [15] Peter Leo Gorski, Sebastian Möller, Stephan Wiefing, and Luigi Lo Iacono. “I just looked for the solution!” on integrating security-relevant information in non-security api documentation to support secure coding practices. *IEEE Transactions on Software Engineering*, 48(9):3467–3484, 2021.
- [16] Matthew Green and Matthew Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security & Privacy*, 14(5):40–46, 2016.
- [17] Fatima Hussain, Rasheed Hussain, Brett Noye, and Salah Sharieh. Enterprise API security and GDPR compliance: Design and implementation perspective. *IT professional*, 22(5):81–89, 2020.
- [18] IETF OAuth Working Group. OAuth 2.0. <https://oauth.net/2/>.
- [19] Stefan Karlsson, Adnan Caušević, and Daniel Sundmark. Quickrest: Property-based test generation of openapi-described restful apis. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 131–141. IEEE, 2020.
- [20] Basel Katt and Nishu Prasher. Quantitative security assurance metrics: REST API case studies. In *Companion Proc. 12th European Conference on Software Architecture (ECSA)*, pages 1–7, 2018.
- [21] István Koren and Ralf Klamma. The exploitation of openapi documentation for the generation of web frontends. In *Companion proceedings of the the web conference 2018*, pages 781–787, 2018.
- [22] Alexander Lercher, Johann Glock, Christian Macho, and Martin Pinzger. Microservice API evolution in practice: A study on strategies and challenges. *Journal of Systems and Software*, 215:112110, 2024.
- [23] Hongqian Karen Lu. Keeping your API keys in a safe. In *Proc. 7th International Conference on Cloud Computing*, pages 962–965. IEEE, 2014.
- [24] Neil Madden. *API security in action*. Simon and Schuster, 2020.
- [25] Maria Maleshkova, Carlos Pedrinaci, and John Domingue. Investigating web APIs on the world wide web. In *Proc. 8th European Conference on Web Services (ECOWS)*, pages 107–114. IEEE, 2010.
- [26] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. Test coverage criteria for RESTful web APIs. In *Proc. 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation (A-TEST)*, pages 15–21, 2019.
- [27] Chris Parnin and Christoph Treude. Measuring API documentation on the web. In *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, pages 25–30, 2011.
- [28] Souhaila Serbout and Cesare Pautasso. APiStic: A large collection of OpenAPI metrics. In *Proc. 21st International Conference on Mining Software Repositories (MSR)*, Lisbon, Portugal, April 2024. IEEE/ACM.
- [29] Souhaila Serbout and Cesare Pautasso. How are web apis versioned in practice? a large-scale empirical study. *Journal of Web Engineering*, 23:465–506, August 2024.
- [30] Souhaila Serbout, Fabio Di Lauro, and Cesare Pautasso. Web APIs structures and data models analysis. In *Companion Proc. 19th International Conference on Software Architecture (ICSA)*, pages 84–91. IEEE, 2022.
- [31] Prabhath Siriwardena. Mutual authentication with TLS. In *Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE*, pages 47–58. Springer, 2014.
- [32] Peter E Snyder. *Improving Web Privacy And Security with a Cost-Benefit Analysis of the Web API*. PhD thesis, University of Illinois at Chicago, 2018.
- [33] Pei Wang, Julian Bangert, and Christoph Kern. If it’s not secure, it should not compile: Preventing DOM-based XSS in large-scale web development with API hardening. In *Proc. 43rd International Conference on Software Engineering (ICSE)*, pages 1360–1372. IEEE, 2021.
- [34] Ying Zhang, Md Mahir Asef Kabir, Ya Xiao, Danfeng Yao, and Na Meng. Automatic detection of Java cryptographic API misuses: Are we there yet? *IEEE Transactions on Software*

Engineering, 49(1):288–303, 2022.

- [35] Mingyi Zhao, Aron Laszka, and Jens Grossklags. Devising effective policies for bug-bounty platforms and security vulnerability discovery. Journal of Information Policy, 7:372–418, 2017.
- [36] Olaf Zimmermann, Mirko Stocker, Daniel Lubke, Uwe Zdun, and Cesare Pautasso. Patterns for API design: simplifying integration with loosely coupled message exchanges. Addison-Wesley Professional, 2022.