On The Structure and Authorization Management of RESTful Web Services

Bojan Suzic
Graz University of Technology
Institute for Applied Information
Processing and Communications
Graz, Austria
bojan.suzic@iaik.tugraz.at

Bernd Prünster
Graz University of Technology
Institute for Applied Information
Processing and Communications
Graz, Austria
bernd.pruenster@iaik.tugraz.at

Dominik Ziegler
Know-Center
Graz, Austria
dominik.ziegler@tugraz.at

ABSTRACT

A broad range of emerging business models relies on the continual exchange of data that flow among different services to generate additional value and derive knowledge in many domains. The magnitude of resource sharing that form the basis of these interactions raises new challenges concerning the effectivity of existing security and privacy management instruments in environments of such complexity.

In this work, we examine the practical application of authorization management mechanisms employed over RESTful Web APIs, which today serve as a major approach to expose service interfaces on the web. For this purpose, we have examined the integration of security mechanisms in n=523 public Web APIs. Our findings reveal alarming integration patterns that demonstrate a rudimentary data security and privacy protection in cross-service resource sharing. Our analysis traces the cause back to the (1) shallow models and security capabilities offered by service providers, and (2) design deficiencies of dominantly applied OAuth 2.0 web authorization framework that restrict capabilities and lower the interoperability of underlying management functions. Following the initial discussion, we summarize potential solutions and establish an outline of the future work.

CCS CONCEPTS

• **Security and privacy → Authorization**; *Distributed systems security*; *Web protocol security*; • **Information systems → RESTful web services**; Service discovery and interfaces;

KEYWORDS

Web services; Service security; Service integration;

ACM Reference Format:
Bojan Suzic, Bernd Prünster, and Dominik Ziegler. 2018. On The Structure and Authorization Management of RESTful Web Services. In *SAC 2018: SAC 2018: Symposium on Applied Computing , April 9–13, 2018, Pau, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3167132.3167315

1 INTRODUCTION

The World Wide Web has created unprecedented possibilities to share, alter and process information without the regard to the geographic or organizational affiliation of manipulated resources, their host systems, or involved persons. Service-oriented architecture (SOA) has emerged as a paradigm to systematically and transparently support integrations in such distributed environment.

Presently, many businesses expose services using the structured web interfaces, enabling their users and other entities to access exposed resources programmatically, in a form that allows easier integration and reuse on a global scale. Many novel service models emerged following this possibility [27], establishing diverse integration scenarios that increasingly rely on cloud infrastructure [18] to provide data and service integration. While different technologies exist to support such integration, including traditional web services stack [5] and more recent Web APIs based on RESTful architectural style [11, 24], the previous research reveals the faster adoption of RESTful paradigm in the recent years [3, 21, 28].

Considering the current popularity and the pervasive use of Web APIs, in this work we investigate how the popular platforms integrate security mechanisms in their publicly exposed service interfaces. We focus on *authorization*, a process of specifying access privileges to users or processes [17] with the purpose to enforce *access control* [32] over resources in a way that ensures a high degree of overall security and privacy. Complementary to the work of authors that investigate capabilities of service description languages to express security mechanisms applied in Web APIs [22], in this work we go a step further by examining the real application of these mechanisms in a broad range of actively consumed Web APIs. Our work derives key usage characteristics and reveals key deficiencies both in terms of how service providers apply existing authorization controls, and what these controls allow concerning the interactions in cross-organizational context.

While the integration middleware and processes in traditional environments consisting of single enterprises were able to rely on proprietary and closed authorization models, the increasing dependence on data and service integration across multiple organizations requires more advanced mechanisms to address the complexity and security requirements in an open world. This is especially important if we want to achieve fine-grained security that restricts the permissions to the minimally applicable set for the particular access context, as mandated by the *least privilege principle* [33].

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167315

2 BACKGROUND AND MOTIVATION

Recent significant growth in the number of publicly available Web APIs may be witnessed both by inspecting popular API catalogues, such as ProgrammableWeb [28], and from the more insightful assesments carried out by the research community [3, 30]. As Web APIs have become a backbone of the Web, cloud, and mobile applications [38], increasing number of sectors recognizes their potential to facilitate innovation, increase productivity and generate new revenue streams [7].

Although their conformance to RESTful architectural style can be a subject of further discussions [12, 29], it is commonly agreed that the simplicity and consumability of underlying interfaces have contributed to the growing adoption of Web APIs [3, 38].

Despite their widespread use, diversity and inconsistency in implementations characterise Web APIs in practice. As oposed to traditional Web service stack [24], majority of Web APIs have evolved autonomously, with the interfaces, structure, and documentation being determined based on techological understanding, business case and the general discretion of implementing parties. The initiatives such as OpenAPI[1] aim to consolidate such fragmented ecosystem by establishing structured API descriptions and providing the tools that support their easier integration in development and integration workflows. In this work we apply the OpenAPIs terminology. Thus, service providers define *endpoints (paths)* and expose *operations* over each path. Operations are determined by the HTTP method [10] applied over the path, so that each path may expose several operations.

API keys are one of common forms of credentials to restrict access to Web APIs, broadly adopted during the previous decade [8]. By issuing API keys, service providers can constrain access to portions or complete web interfaces to particular clients only. Less frequently, their additional purpose was to ensure the integrity of incoming Web API invocations [8]. However, with the emergence of new products and complex interactions, the management and issuance of these credentials was associated with additional complexities. One of primary functional concerns related to the use of API keys was the lack of standardized mechanisms to automatically issue tokens to third parties.

In practice, developers who wanted to integrate two systems had to generate or receive API key from the service provider and employ it in the local application that interacts with the service. Due to the manual execution of this process, done using out-of-the-band channels, API keys were not suitable to enable integrations with third-party applications on a larger scale. OAuth 2.0 [15], among others, aimed to address this issue by defining the protocol flows and grant types that enabled structured resource sharing across several separate entities. This way, *resource owners* were able to authorize resource sharing at *service provider* with external, *third-party applications* using consent-based flows. While API keys do not offer means to control the extent of authorizations, OAuth 2.0 establishes *scopes*, which are parameters that inherently encompass permissions specific for the managed service.

2.1 Motivational scenario

In the current ecosystem, OAuth 2.0 is practically the only broadly adopted mechanism that allows the scalable authorization management of multilateral data sharing and service integrations. However, as it will be shown in the subsequent sections, there are several issues concering its underlying capabilities and the practical application among service providers. To illustrate the problem of coarse-grained scopes, we briefly rely on a scenario based on Zapier[2], a cloud automation platform [27] that integrates and processes resources of a single *resource owner* hosted at different *service providers*. Consider that Zapier needs to periodically integrate Gmail and MailChimp services with the goal to retrieve emails from Google, extract email senders and add them as subscribers to MailChimp marketing list. This step is repeated periodically, typically every ten minutes, whereas only the recently received messages are relevant, and only a particular list of subscribers at MailChimp has to be updated. To obtain necessary permissions, in its setup process Zapier creates separate permission requests for both service providers using OAuth 2.0 authorization flows [15]. These permissions have to be *consented* by the resource owner, which *authorizes* service provider to issue long-lived access token to Zapier.

When requesting these permissions, Zapier has to rely on the scopes defined by GMail and MailChimp. Currently, the GMail scope that satisfies this scenario and includes the lowest range of permissions is `gmail.readonly`. It, however, provides the access to 12 resources and allows the execution of 24 operations over them[3]. Among others, these include the ability to list all emails, to read all the metadata and content of these emails, to check the complete user's history, as well as to inspect user's drafts, settings and labels. From the reference documentation we learn that the access restriction in a more granular and context-sensitive manner is not supported due to scope selection. While the scenario requires that Zapier retrieves *only the senders* from the emails arrived during the last *ten minutes*, it actually has access to all messaging content, history and beyond of that.

In the case of the second service, the situation is simpler as MailChimp does not define scopes. Hence, an accessing client would be able to *read, add* and *update* all data exposed over all MailChip's endpoints. Besides lack of the complete support for requirements such as privacy and data confidentiality, the second service fails to support the requirement of data integrity and potentially introduces weaknesses related to other security metrics.

While this scenario exemplifies the security and privacy risks[4] that involve centralized resource sharing and processing using a cloud integration platform [27], it should be stressed that similar issues ocurr in other cases that rely on resource sharing in multilateral environment. We identify authorization management in service integrations as a critical asset whose underlying capabilities determine data security and privacy properties of those processes. According to that, in the subsequent chapters we analyze and discuss the practical application of authorization management across a broader range of RESTful services.

[1] https://www.openapis.org

[2] https://zapier.com

[3] https://developers.google.com/gmail/api/v1/reference

[4] For the distinction between security and privacy in the cloud we refer to [25]

3 ASSESMENT OF APIS

3.1 Data set and methodology

To perform this analysis, we have gathered, preprocessed and evaluated n=523 descriptions of Web APIs published at *OpenAPI Directory* [1]. Maintained by the *APIs.guru* project [1], OpenAPI directory is an open source, a community-driven project dedicated to providing machine-readable descriptions of public RESTful APIs. The specification and categorization of APIs are based on a structured and open process that includes quality control and frequent updates. The data published on this platform include descriptions of diverse APIs both in terms of functionality and purpose.

To the best of our knowledge, OpenAPI Directory is the most comprehensive source for structured API information on the Internet. Other data sources, such as *ProgrammableWeb* [28], aim to catalogize and provide basic information about APIs. This information is often focused on describing extrinsic properties that include various meta-data, such as category, vendor, and purpose. Our analysis, on the other hand, requires access to information about intrinsic properties of APIs, which is typically not published in such catalogs.

In our workflow, we have gathered API descriptions available during the first week of September 2017 using a semi-automated process. We have then applied discovery and transformation mechanisms using a range of custom scripting tools to extract and process necessary information. In the subsequent steps, we have exported all gathered information to the format applicable for further statistical analysis. The resulting data set is available at [36] to support reuse and validation of our results.

In the remainder of this section, we provide basic descriptions of our data set. We then define specific measures and apply them to gather more insights into API authorization management.

3.2 API properties

Table 1 presents the summary of the dataset used in the subsequent analysis steps. Overall, there were 523 structured APIs obtained. Their interfaces define 11.664 endpoints and expose 14.991 operations to accessing agents.

Table 1: Aggregate summary of API descriptions

	Measure	Number	%
(1)	APIs	523	-
(2)	Endpoints	11.664	-
(3)	Operations	14.991	-
(4)	Protected APIs	365	69.79%
(5)	Unprotected APIs	158	30.21%

In Table 2 we summarize the properties using per API measures. Comparing the results for average and median numbers of endpoints, it can be observed that the most APIs tend to provide a lower number of endpoints than suggested by aggregate average value. This can be confirmed by looking at outliers in the dataset. In fact, 259 (49.5%) of APIs tend to provide ten or fewer endpoints, and 368 (70.3%) expose no more than twenty endpoints. On the other hand, 19 (3.6%) APIs expose more than 100 endpoints. This distribution is illustrated on Figure 1, with the vertical axis showing the number of APIs using a logarithmic scale, and the horizontal axis representing the number of endpoints exposed by each data segment.

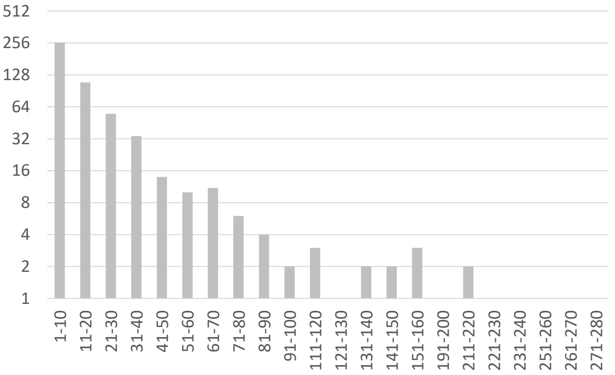

Fig. 1: Number of endpoints across APIs

What distinguishes operations from endpoints is their invocation using supported HTTP methods [10]. Thus an operation can be considered as an abstraction over an endpoint using a particular invocation method. In the analyzed dataset, operations exposed by APIs are accessible using one of seven HTTP methods, as presented in Table 3. Considering that the endpoint exposes a resource, and an operation provides access to a specific function for that resource, we can conclude that vendors tend to associate one method per endpoint. This can be observed from the average number of methods of 1.25 (Table 2) and the fact that 333 (63.6%) of APIs expose no more than this average of different methods per endpoint.

Table 2: Summary of API properties

	Measure	#
(1)	Average number of endpoints	22.30
(2)	Average number of operations	28.66
(3)	Average number of methods	1.25
(4)	Median number of endpoints	11
(5)	Median number of operations	14
(6)	Median number of methods	1

The distribution of HTTP methods applied over endpoints in all APIs is presented in Table 3. We show here the number of applied HTTP methods, followed by their percentual contribution, average and median measures. These represent aggregate counts derived from the whole data set.

Table 3: Distribution of HTTP methods

	Method	#	%	Avg	Med
(1)	HEAD	16	0.11	0.15	0.00
(2)	DELETE	1,213	8.09%	5.13	0.00
(3)	POST	5,373	35.84%	35.60	25.00
(4)	GET	6,726	44.87	52.25	50
(5)	OPTIONS	2	0.01	0.00	0.00
(6)	PUT	1,245	8.30	4.87	0.00
(7)	PATCH	416	2.77	2.0	0.00

The application of these methods in requests determines the type of operation that is expected to be executed on the server side. According to the semantics of HTTP methods [10], OPTIONS method is used to describe the communication options for the target resource. The methods GET and HEAD are applied to retrieve resource representation or its metadata, respectively. The use of POST initiates the request-specific processing of the request payload, while PUT and PATCH imply the update of state of the target resource or its parts. Removal of the association between the target resource and its functionality is instructed by applying DELETE method. For the more detailed specification these of methods we refer to [6, 10].

From Table 3 we can observe the dominant use of HTTP GET and POST methods. We can also notice that more than half of all exposed operations serve the purpose of modifying existing or creating new resources. These are referred to as *unsafe methods* that may produce side-effects on resources, as established by Fielding and Reschke [10]. For detailed explanation of side effects of HTTP methods and the overview of their use in surveyed APIs we refer to Section 3.6.

3.3 Access control in APIs

Depending on the use-case and practical integration concerns, many service providers need to restrict access to exposed endpoints and operations. This is especially important if their interfaces provide access to sensitive or critical data that should be protected. Several access control mechanisms that support this requirement have been developed and more or less adopted in practice. In this section, we examine the application of these mechanisms across a broad range of web services.

Based on the analysis of the provided data set, we have discovered that 30.21% of APIs do not declare access requirements for the external clients. Most APIs (365 or 69.79%) declare the use of some access control mechanism to restrict the access to their endpoints (Table 1). From this set, 347 APIs (95.07%) support one authorization mechanism, 16 APIs (4.38%) apply two, and only 2 APIs (<1%) require three or more authorization mechanisms.

In Table 4 we show the overview of supported security mechanisms in examined APIs. These numbers are derived from structured service specifications on an aggregate level, with the percentage ratios referring to the summarized count of different mechanisms across all services. This overview does not include 158 APIs that do not support any security mechanism. Included in the overview are 20 APIs that support two or more security mechanisms. From this data we can observe the broad and approximately similar use of two principal mechanisms - API keys [8] and OAuth [15].

Table 4: Application of security mechanisms

	Mechanism	APIs	%
(1)	HTTP	15	3.90%
(2)	API-Key	185	48.05%
(3)	OAuth	177	45.97%
(4)	Custom	8	2.08%

There are two key functional differences between API Keys and OAuth that are relevant for this work. They primarily concern the projected resource sharing scenario of a security mechanism and its capability to restrict access to the resources in a focused and granular manner.

In the first instance, API keys are credentials aimed at developers building applications that do not access more than a single user's data. This restricts the process of obtaining and distributing such credentials for an application to a manually performed activity. Although it does not represent a technical restriction, the number of API keys is in practice often limited to a couple of clients per service tenant. This corresponds to the envisaged use scenario, as these credentials are meant to support integration between two heterogeneous systems, whereas *one party* exhibits a (virtual) presence on both sides. OAuth, on the other hand, provides a mechanism to issue an access token in an automated way, enabling *resource owners* to share their data hosted at *service providers* with *third-party applications*. The process of obtaining such a token is based on an explicit *consent* of a resource owner to share its resource. OAuth hence involves multiple parties, implicitly providing a mechanism to manage the credentials issued to diverse connecting agents.

The second difference between API keys and OAuth reflects their ability to control the degree of given permissions. As API keys do not provide means to granularly restrict access to service resources and operations, they typically get applied in the form of an authentication mechanism. In comparison to that, the abstraction of *access scopes* in OAuth provides granular authorization capabilities that enable providers to associate token permissions with a subset of operations or resources. This is due to the underlying intention for API keys to be used in closed service integrations, whereas the developer acts both as access client and resource owner. While it does not explicitly specify underlying semantics and applications, the concept of scopes enables service providers to associate logical compartments to scopes and restrict the extent of permissions applicable to tokens provided under a particular scope.

Following these distinctions, we can conclude that nearly half of the services aim to provide their customers with means to share resources with third parties using a structured authorization mechanism.

Considering the limited capabilities and expressiveness of API keys, and OAuth with scopes as currently only applied concept that enables permissioning across the services, in the rest of this analysis we examine the application of OAuth authorization framework and its particular properties. For this purpose we have derived several metrics which we present in the subsequent sections.

3.4 Application of OAuth

The first dimension that describes the application of OAuth scopes relates to the degree of integration of scopes in specification and enforcement of access control of web services. In Table 5 we categorize the services based on the number of different scopes that were declared to be used. Here we can observe that nearly half of the services apply only one scope in their authorization models. Up to some extent this may contradict with OAuth's design intention, as one of its purposes was to enable the resource owner to restrict access permissions for third-party agents. By relying on only one scope, the resource owner practically allows the access to the complete range of resources and operations exposed by the service.

By analyzing the gathered data we have observed a particular discrepancy among the *declaration* of security mechanisms, and their practical *application* in service specification documents. Namely, of 177 services that declared the use of OAuth, 173 have specified authorization scopes. Furthermore, 130 services have actually used the specified scopes in API descriptions, and of them, 89 have employed two or more scopes. From this point, in the following sections we use these two data sets, refered as *standard* and *reduced* where appropriate, to perform the further analysis.

Table 5: Declared OAuth scopes

	Scopes	APIs	%
(1)	One	80	46.24
(2)	Two	29	16.76
(3)	Three	13	7.51
(4)	Four	14	8.09
(5)	Five+	37	21.39

3.5 Scope coverage and similarity

OAuth 2.0 defines the structure of the scope parameter as a list of space-delimited, case-sensitive strings, whose inherent permission extent is determined by the provider. Multiple of these strings (scopes) can be combined as a single parameter to express a combined range of access permissions.

Besides recommending that service providers should document their scopes, OAuth 2.0 specification does not provide any further guidance on the establishment of scopes, nor does impose any requirements that would allow automated dereferencing of scopes. Consequently, the scopes get applied as opaque strings with hard-wired logic, without reliance on a standardized approach to realize machine-based readability and an understanding of underlying permissions. Instead, the degree of permissions offered by scopes is typically explained in a service's documentation in a natural language, with the primary aim of supporting developers in implementations and service integrations. Due to the diversified approaches and expressive power of natural languages, these descriptions may vary across service providers or API versions, which increases the complexity and imposes additional overhead in the implementation and maintenance of service integrations.

As there is no machine-readable way to obtain scope permissions and relate them to running systems or resources, we have decided to take another approach and derive their extent by analyzing the practical application of scopes in APIs. With this aim we have constructed two metrics to support quantifying, expressing and comparing the permission properties of scopes in a single API.

Considering that each scope supports a subset of available operations, we define the *scope coverage* as a measure that describes the proportion of operations that a scope supports in an API. We apply the following equation to express the coverage of a scope s in an API α using the total number of operations in α (op^{α}), and the number of operations covered by the scope s (op_s^{α}):

$$cov_s^{\alpha} = \frac{|op_s^{\alpha}|}{|op^{\alpha}|}$$

Scope coverage is expressed with values in the range 0..1, with 1 implying that the scope s covers all operations in the API α. In other words, the API that relies on such scope allows the authorized agent to exploit all functionalities and access all resources under a single access token.

In addition to that, we derive an additional metric to express similarity between two scopes based on their common supported operations. *Scope similarity* is hence defined as a measure of operations shared among two scopes. Similarity of the scopes s_1 and s_2 under the API α is expressed as follows:

$$sim_{s1,s2}^{\alpha} = \frac{|op_{s1}^{\alpha} \cap op_{s2}^{\alpha}|}{|op_{s1}^{\alpha} \cup op_{s2}^{\alpha}|}$$

This measure takes values in the range 0..1, with 1 implying complete functional equivalence between the scopes s_1 and s_2. Compared to scope coverage, this measure goes beyond the quantitative degree of underlying permissions, enabling the examination of two scopes using a range of equivalent operations they cover. While there might be legitimate reasons to define scopes that are virtually the same, the existence of functionally equivalent scopes may serve as a starting point to assess the quality of an API design, its underlying authorization model, and potential security-related weaknesses.

We have analyzed scope coverage and similarity across the reduced data set to examine the application and distribution of underlying permissions in APIs. Both measures were derived from a reduced data set (n=89), as the analysis of granularity of permissions in APIs makes sense in environments relying on two or more scopes.

Both metrics for coverage and similarity grouped by the number of APIs corresponding to a range of average values are presented in Table 6. The average values in both cases have been calculated separately on the basis of each service and its applied scopes. In calculating scope similarities we have included only the unique scope pairs in each API and then derived their average values, as shown in Table 6 and Figure 3.

The distribution of scope coverages across services is visualized in Figure 2. The horizontal axis represents each service, while the vertical axis shows the coverage ratio. The value 1 indicates the coverage of 100% of operations available in an API. The vertical lines in the figure depict the range of a minimal (bottom) and a maximal coverage (top) found in an API . The average coverage of all scopes employed in an API is represented as a circle on the line.

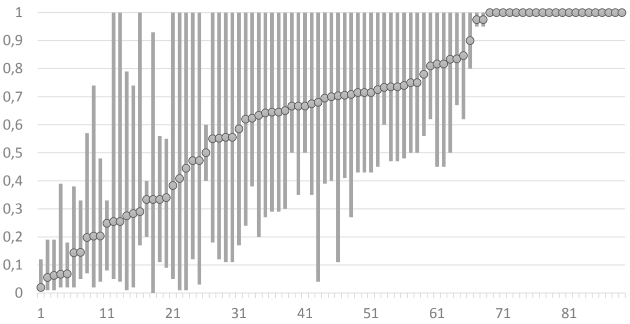

Fig. 2: Distribution of scope coverages

As observable from the respective table and figure, nearly a quarter of services use scopes with complete coverage (n=21, cov=1). Although their APIs employ several scopes, in practice all these scopes allow access to the same operations, rendering them functionally equivalent. Contributing to this aspect are the environments with a lower count of scopes whose extent is adapted from an immature development stage, potentially incepted during the initial API design. The second factor are *cross-API scopes* which are applied across several APIs of a single vendor. One of such examples is the Google Cloud Platform API, which defines two scopes, namely `datastore` and `cloud-platform`. While both of them support all of the operations exposed by the API, the latter can be obtained and applied for operations in other APIs of the same vendor, as well.

Table 6: Averages of scope coverages and similarities

	Range	Coverage # APIs	%	Similarity # APIs	%
(1)	0-0.1	5	5.62	15	16.85
(2)	0.1-0.2	3	3.37	6	6.74
(3)	0.2-0.3	8	8.99	5	5.62
(4)	0.3-0.4	5	5.62	9	10.11
(5)	0.4-0.5	4	4.49	9	10.11
(6)	0.5-0.6	6	6.74	4	4.49
(7)	0.6-0.7	13	14.61	12	13.48
(8)	0.7-0.8	15	16.85	5	5.62
(9)	0.8-0.9	6	6.74	1	1.12
(10)	0.9-1.0	24	26.97	23	25.84

Following the data from Table 6 we may further observe that more than 70% of APIs exhibit a scope coverage of more than 0.5. This suggests that, on average, most APIs employ scopes that allow a broad range of permissions. This can be confirmed by the visualization on Figure 2, which shows that more than 79% of APIs define at least one scope with a maximum coverage equal to 1, while a single scope with a minimal coverage greater than $\frac{1}{3}$ can be found in 56% of services. Only a fraction of services (n=10, 11.2%) uses a scope that covers no more than 50% of defined operations. Both of these metrics suggest that service providers tend to define scopes with relaxed and, on average, broader permissions.

The distribution of scope similarities across services is presented on Figure 3. The horizontal axis represents services, the vertical axis relative ratios in the scale 0..1. Vertical lines in the diagram depict minimum and maximum observed scope similarity (bottom and top values, respectively). The average similarity value derived from each API is represented as a circle on the vertical line.

In the figure, we can observe a similar permissioning trend as shown in the case of scope coverages. However, the data from Figure 3 and Table 6 shows a slightly different distribution of scope similarities. Here we can notice that roughly half of the services (n=45, 50.5%) exhibit an average scope similarity of more than 0.5, while a fraction of APIs contains a scope that exhibits a maximal similarity of $\frac{1}{3}$ (n=14, 15.7%). We have found that the majority of services with such a similarity (n=8) rely on three or more scopes.

Due to its dependence on semantically specific operations, we can consider scope similarity as a more strict measure of diversity in applied authorization and permission models. Looking at both measures, we can confirm the existence and significant degree of broad permissions in APIs.

In the following section, we extend this analysis by examining the relationship between authorization and interaction layers in service interfaces.

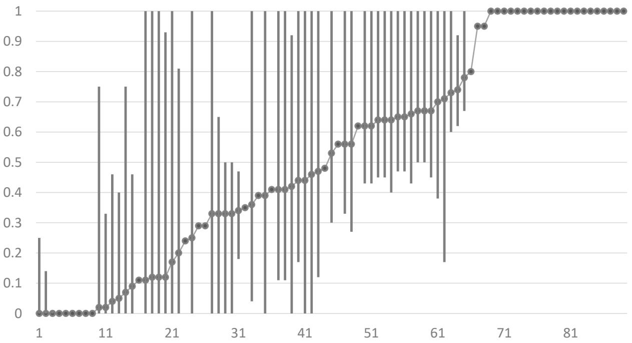

Fig. 3: Distribution of scope similarities

3.6 Application of HTTP methods in scopes

RFC7231 [10] distinguishes two categories of HTTP methods based on the degree of their side effects. Methods which are not intended to be used for actions other than retrieval are considered *safe methods*. This category includes GET and HEAD. Other methods, such as POST, PUT, and DELETE, are expected to induce effects when invoked and hence should be treated in a special way. Due to the possible effects on target resources, these are considered *unsafe methods*.

Scopes in APIs are usually applied to authorize one or more *operations*. In a typical scenario, an operation can be abstracted as an invocation of a specific HTTP method over a particular *resource* (endpoint). From this point, each scope can be related to one or more HTTP methods and resources.

In this analysis we have investigated how scopes from the dataset encompass the application of *safe* and *unsafe* methods. For this purpose we had separately summarized data for each API and calculated the degree of scopes that include *safe* and *unsafe* methods. Based on these results, we derived the proportion of the *overlapping* scopes, which are the ones that support both safe and unsafe methods. To support the visualization of the results, Figure 4 additionally includes this third category. The distribution of both categories is shown in a stacked form. The horizontal axis on the figure is used to represent APIs, while the vertical axis provides the values for each of three measures.

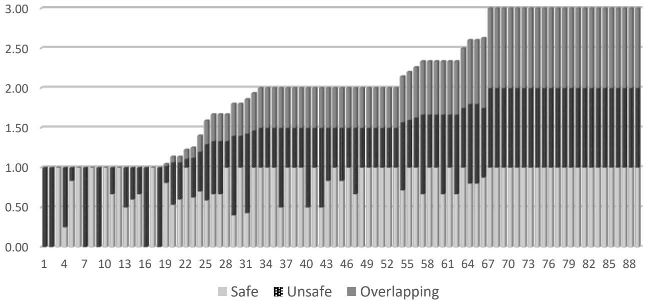

Fig. 4: Distribution of HTTP methods among scopes

All values presented in the figure are based on relative ratios applied to each category. Thus, the value of 0.85 for safe scopes implies that 85% of all scopes of an API use safe HTTP methods.

The measure of overlapping scopes provides information on the degree of scopes that use safe and unsafe methods simultaneously.

We have identified 18 (20.2%) APIs that specify scopes with no overlapping, which are shown in the left part of the diagram. The depicted sum of ratios of these scopes amounts no more than 1, as each scope relies only on one type of HTTP method. On the other side, 23 (25.8%) APIs apply scopes that exhibit full overlapping between those two categories of methods (right on the figure).
48 (54%) APIs, represented in the middle of the diagram, exhibit an overlap between safe and unsafe operations to some degree.

Based on the provided data we can observe a tendency of APIs to employ scopes whose reach combines safe and unsafe methods. In practice, 57 (64%) APIs demonstrate an overlapping of 0.5 or higher degree. We also observe a correlation between the use of unsafe methods and overall overlapping. Tested using Pearson's coefficient, this measure accounts 0.79 ($p<0.01$).

One of the contributing factors for such a distribution is the general tendency of API vendors to separate scopes to ones that support read-only, and others that support both read and modify operations. This can be illustrated with the example of GMail, which among others, establishes scopes `gmail.readonly` and `gmail.modify`, used to allow operations that perform retrieval and modification of emails, respectively. The latter scope practically encapsulates a broad range of operations, including the ones included in the scope `gmail.readonly`. Such an approach hinders users in fine-tuning application privileges. Not to mention that the second scope in this example supports a broad range of modification operations on different resources; it also does not allow resource owners to issue modify-only permissions.
As a result, resource owners and accessing agents are forced to rely on the permission package discretionary chosen by the service provider that might not optimally support their scenario and security requirements. This assumption may simplify the implementation of the authorization model at the service provider, but at the same time, it restricts the capability of other parties to manage resource sharing and use authorizations in a fine-grained manner.

4 DISCUSSION

In this section, we summarize and discuss the findings presented in this work. Our conclusions lead to the notion of interoperability of authorization management controls, which we see as an essential building block to advance security management from hard-wired controls with implicitly derived meanings, to the structured and machine-interpretable form that is applicable beyond a single use-case or the boundaries of a single organization.

4.1 Deriving authorization patterns

The findings presented both in sections 3.5 and 3.6 show the issues in determining access permissions for agents accessing web service APIs. We can identify several issues in the application of scopes. Analysis of the dataset provided the first insights on a broader scale on the degree of the application of scopes. While OAuth 2.0 enabled the service providers and clients to specify requested and provided extent of the authorization using the *scope* parameter, more than 60 % of evaluated APIs declare no more than two scopes. This practically restricts the choice for resource owners and accessing clients, as the requested and given permissions can be established on a coarse-grained level only. If we consider that an average API exposes some 22 endpoints or 28 operations, one or two scopes may enable controlling access to these resources in a limited manner only.

The presented findings further demonstrate the bias among providers towards coarse-grained permissioning that stems from the use of (1) low number of scopes, (2) their broad coverage and/or significant overlap of supported operations and (3) low distinction level among single scopes. In Section 3.6 we evidenced the construction of authorization mechanisms that (4) inconsistently map HTTP methods to scopes, in a way that encapsulates the invocation of different groups of methods under the same scope. All of these issues effectivelly lead to *overprivileging*, a well-known problem in information security.

While during the last decades many researchers invested significant efforts to produce a range of access control models to support efficient and effective authorization management in various scenarios (including [2, 16, 32]), it is surprising that not much concern has been raised when it comes to data and resource sharing across several entities and organizations using the scenarios discussed above. Here we primarily mean on the capability of underlying technologies to support the principle of least privilege and adapt it to conform to the new dynamic context characterized by the data exchange and service consumption across *different* organizational entities.

Due to the legal requirements, the sensitivity of private and business data and the scale of potential impacts, we see the increasing need to advance the capabilities of technologies that support the security management of cross-organizational data and service integrations. This is not important due to the related security and privacy risks, but also due to the potential of these technologies that can be unleashed only if they are considered as secure.

4.2 Maturity of current approaches

The overprivileging demonstrated in this work broadly disobeys the *principle of least privilege* [33], one of the key principles when it comes to protecting data confidentiality and integrity. Due to the current development level we see the current ecosystem for multilateral authorization management in early phases of maturity. For this we identify two contributing reasons.

While the initial version of OAuth 1.0a has been more of an effort produced by a dedicated community [14], the subsequent version in the form of OAuth 2.0 got a broader institutional attention and, after several refinements, got advanced as an IETF standard [15]. In this process, Eran Hammer resigned as a lead author and editor, citing reasons such as bad security and interoperability of the protocol and its bias towards enabling enterprises to solve use cases with the minimal effort [13]. While some of these statements may pose the ground for further debate, a range of discovered flaws [4, 9, 40] partially prove Hammer's expectations on the emergence of insecure implementations. In this work we have shown that service providers tend to employ authorization management to use a simpler set of capabilities, producing permission models that are inconsistent and less secure. While there is not enough evidence to draw a general conclusion, the issues summarized in this section may suggest that in some cases service providers invested suboptimal resources to integrate security capabilities provided by the OAuth 2.0 framework.

As a second reason, we may notice that the underlying technical mechanisms provide only limited capabilities to integrate and execute authorization management processes across several organizations. One of these immature capabilities is restricted *interoperability*, an issue which is separately mentioned in the original specification as well (Section 1.8 of [15]). The high level of a scope's abstractness, its unspecified underlying semantics and missing guidance on application and integration have left an open space for a broad range of implementation and integration approaches, with some of them representing suboptimal solutions both in terms of security and functionality. The underspecification of this element results in a lacking *vertical* and *horizontal* interoperability.

We consider the vertical dimension of interoperability as the capability of a scope to be related to other parts of a system in a machine-readable way. These include, for instance, referencing resources and operations of an API, as well as their constitutive parts or subsets. By relying on existing service descriptions incepted using available service description frameworks[5] the capabilities of scopes could be dynamically defined in a manner that allows fine-grained and contextually adaptable permissions. Instead of being opaque strings without dereferenceable structure, the scopes could reference a range of operations, resources or their segments in a way that is machine-interpretable, and hence, potentially subjected to more comprehensive security evaluation and enforcement.

For a further discussion and proposal that explores this direction, we refer to the work of Suzic et al. that deals with the cross-cloud aware specification of access scopes [37].

Under horizontal scope interoperability we understand the capability to relate and apply the scopes across several systems or organizations. Instead of being related to a single organization, scopes should be interpretable in the context going beyond of that. For the example of such interoperability level, we could take Google scopes mentioned in Section 3.6, which demonstrates the first step in this direction by being applicable beyond a single API. This would allow the cross-API reuse of scopes and definition of permissions that are applicable beyond a single environment. From this perspective, the horizontal interoperability of scopes could have the potential to unify and automate authorization management across services, leading to a higher level of security due to lower interoperability and integration obstacles.

We refer to our other work that relies on semantic vocabularies[6] to define and transparently enforce access controls among different cloud-based services [35]. We consider that work as an orthogonal contribution to [37] that aims to advance the authorization management by unifying service descriptions and security policies. Such approach allows the definition and enforcement of security policies across a range of supported web services in a granular, context-sensitive and user-centric manner.

5 RELATED WORK

To the best of our knowledge, there is no study that systematically and transparently examines the use of OAuth 2.0 authorization mechanisms across services and their Web APIs. Previous work of Suzic et al. [34] has investigated the security aspects of cross-domain service integrations, considering particularly OAuth 2.0 framework [15] and emerging UMA protocol [19, 20]. The identified deficiencies from this work motivated us to perform the analysis of the practical application of OAuth 2.0 in a larger context.

One of the initial studies of Web API descriptions has been carried out by Maleshkova et al. [21]. In this work, authors retrieved a set of 222 APIs from the ProgrammableWeb directory [28] and analyzed their metrics that include the distribution of API types and input parameters and formats employed in API invocations. The later studies with more detailed insights have been published by Bülthoff and Maleshkova [3], and Renzel et al. [29]. These surveys included descriptions of 45 and 20 Web APIs, respectively, selected by their popularity measure according to Alexa web ranking.

More recent findings on Web APIs are provided in the surveys of Rodriguez et al. [30] and Petrillo et al. [26]. In contrast to other works, the study by Rodriguez et al. investigated a data set derived from the mobile traffic of the largest Italian mobile internet provider. The motivating goals of these studies were the mapping of the API structures, as well as the assessment of their conformity to RESTful principles [23] and maturity models [31]. However, in terms of security properties, only the surveys of Bülthoff and Maleshkova [3] and Renzel et al. [29] derived particular insights on authentication methods employed in the APIs. However, no more details beyond the distribution of these mechanisms were provided.

Recent work of Nguyen et al. [22] reviews security-related capabilities of description languages for RESTful web services. While this contribution delivers a systematic overview of the most relevant and recent API specification frameworks, its scope focuses on identifying supported security mechanisms and protocols. No insights were given on the actual application of these security controls in practice, nor their capabilities were evaluated beyond expressing the service invocation requirements.

The coarse-grained permissions and over-privileging are present on other platforms. Fernandes et at. have analyzed the security of SmartThings, a Samsung's smart home appliance for management of a broad range of devices including motion sensors, alarms, and door locks [9]. SmartThings uses proprietary cloud environment to expose and control a large number of applications (SmartApps) and relies on OAuth protocol to protect third-party integrations. Both the privileges on the level of home appliance and OAuth scopes used to protect access to SmartApps establish coarse-grained capabilities that encompass operations with different risk classes. Fernandes et al. constructed four proof-of-concept attacks that demonstrate security impacts of the underlying design flaws [9].

Chen et al. have discovered the flaws in the integration of OAuth protocol with Android mobile applications. In their study [4], 59.% of 600 analyzed mobile applications were vulnerable due to developer misconceptions. Both Fernandes et al. and Chen et al. point to the protocol underspecification, vagueness and missing guidelines in the implementation of OAuth protocol flows.

[5] Such as OpenAPI semantic descriptions [39]

[6] DASP-Sec domain-specific vocabularies, available at http://daspsec.org

6 CONCLUSION

By exhibiting a pervasive and the significant growth during the recent years, RESTful Web APIs have been established as an essential infrastructural constituent of modern Web, cloud, and mobile applications. Service providers from diverse business sectors expose their interfaces using Web APIs to allow their users and other services to establish integrations and (re)use data and services in a broad range of scenarios. As many modern services rely on data sharing and service consumption that span across different organizational entities, the capability to establish and maintain the comprehensive security and privacy management of the underlying interactions requires additional attention.

In this work, we examined the application of authorization management mechanisms over a broad range of RESTful APIs currently exposed and used in many applications. We particularly focused on the capability of OAuth 2.0 web authorization framework, currently only standardized and broadly adopted approach for managing resource sharing on the web. We have demonstrated deficiencies both in the application of this framework, and its underlying capabilities. Our analysis points to overprivileging present at a large scale across many providers. To advance the security and privacy of cross-organizational data sharing and to support further prospects of emerging complex ecosystem we motivate the establishment of interoperability in cross-organizational authorization management on the level of security controls.

ACKNOWLEDGMENTS

This work has been partially supported by A-SIT Secure Information Technology Center Austria, and EU H2020 Programme under the SUNFISH project with grant № 644666.

REFERENCES

[1] APIs.guru. 2017. APIs.guru Wikipedia for WEB APIs . (2017). https://apis.guru/openapi-directory//

[2] Jasper Bogaerts, Maarten Decat, Bert Lagaisse, and Wouter Joosen. 2015. Entity-Based Access Control: supporting more expressive access control policies. In *Proceedings of the 31st Annual Computer Security Applications Conference*. ACM, 291–300.

[3] Frederik Bülthoff and Maria Maleshkova. 2014. RESTful or RESTless–Current state of today's top Web APIs. In *European Semantic Web Conference*. Springer, 64–74.

[4] Eric Y Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague. 2014. OAuth Demystified for Mobile Application Developers. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 892–903.

[5] Robert Daigneau. 2011. *Service Design Patterns: fundamental design solutions for SOAP/WSDL and RESTful Web Services.* Addison-Wesley.

[6] L Dusseault and J Snell. 2010. PATCH Method for HTTP, Internet Request for Comments, vol. RFC 5789 (Proposed Standard). (2010).

[7] Peter C Evans and Rahul C Basole. 2016. Revealing the API ecosystem and enterprise strategy via visual analytics. *Commun. ACM* 59, 2 (2016), 26–28.

[8] Stephen Farrell. 2009. API Keys to the Kingdom. *IEEE Internet Computing* 13, 5 (2009).

[9] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security analysis of emerging smart home applications. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 636–654.

[10] R Fielding and J Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, Internet Request for Comments, vol. RFC 7231 (Proposed Standard). (2014).

[11] Roy T Fielding and Richard N Taylor. 2002. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)* 2, 2 (2002), 115–150.

[12] Roy T Fielding, Richard N Taylor, Justin R Erenkrantz, Michael M Gorlick, Jim Whitehead, Rohit Khare, and Peyman Oreizy. 2017. Reflections on the REST architectural style and principled design of the modern web architecture (impact paper award). In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 4–14.

[13] Eran Hammer. 2012. OAuth 2.0 and the Road to Hell. (2012). https://hueniverse.com/oauth-2-0-and-the-road-to-hell-8eec45921529

[14] Eran Hammer-Lahav. 2010. The OAuth 1.0 Protocol, Internet Request for Comments, vol. RFC 5849 (Informational). (2010).

[15] Dick Hardt. 2012. The OAuth 2.0 authorization framework, Internet Request for Comments, vol. RFC 6749 (Proposed Standard). (2012).

[16] Xin Jin, Ram Krishnan, and Ravi S Sandhu. 2012. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. *DBSec* 12 (2012), 41–55.

[17] Audun Jøsang. 2017. A Consistent Definition of Authorization. In *International Workshop on Security and Trust Management*. Springer, 134–144.

[18] Michael Kleeberg, Christian Zirpins, and Holger Kirchner. 2014. *Information Systems Integration in the Cloud: Scenarios, Challenges and Technology Trends.* Springer International Publishing, Cham, 39–54.

[19] Maciej P Machulak, Eve L Maler, Domenico Catalano, and Aad Van Moorsel. 2010. User-managed access to web resources. In *Proceedings of the 6th ACM workshop on Digital identity management*. ACM, 35–44.

[20] Eve Maler, Maciej Machulak, and Justin Richer. 2017. User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization. *Kantara Initiative, Draft Recommendation* (2017).

[21] Maria Maleshkova, Carlos Pedrinaci, and John Domingue. 2010. Investigating Web APIs on the World Wide Web. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*. IEEE, 107–114.

[22] Hoai Viet Nguyen, Jan Tolsdorf, and Luigi Lo Iacono. 2017. On the Security Expressiveness of REST-Based API Definition Languages. In *International Conference on Trust and Privacy in Digital Business*. Springer, 215–231.

[23] Cesare Pautasso. 2014. RESTful web services: principles, patterns, emerging technologies. In *Web Services Foundations*. Springer, 31–51.

[24] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. 2008. RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. In *Proceedings of the 17th international conference on World Wide Web*. ACM, 805–814.

[25] Siani Pearson. 2013. Privacy, security and trust in cloud computing. In *Privacy and Security for Cloud Computing*. Springer, 3–42.

[26] Fabio Petrillo, Philippe Merle, Naouel Moha, and Yann-Gaël Guéhéneuc. 2016. Are REST APIs for cloud computing well-designed? An exploratory study. In *International Conference on Service-Oriented Computing*. Springer, 157–170.

[27] M. Pezzini and B. J. Lheureux. 2011. Integration Platform as a Service: Moving Integration to the Cloud. *Gartner* (2011). http://www.gartner.com/id=1575414

[28] ProgrammableWeb. 2017. ProgrammableWeb - APIs, Mashups and the Web as Platform. (2017). https://www.programmableweb.com/

[29] Dominik Renzel, Patrick Schlebusch, and Ralf Klamma. 2012. *Today's Top "RESTful" Services and Why They Are Not RESTful.* Springer Berlin Heidelberg, Berlin, Heidelberg, 354–367.

[30] Carlos Rodríguez, Marcos Baez, Florian Daniel, Fabio Casati, Juan Carlos Trabucco, Luigi Canali, and Gianraffaele Percannella. 2016. REST APIs: a large-scale analysis of compliance with principles and best practices. In *International Conference on Web Engineering*. Springer, 21–39.

[31] Ivan Salvadori and Frank Siqueira. 2015. A Maturity Model for Semantic RESTful Web APIs. In *Web Services (ICWS), 2015 IEEE International Conference on*. IEEE, 703–710.

[32] Pierangela Samarati and Sabrina Capitani de Vimercati. 2000. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*. Springer, 137–196.

[33] Fred B Schneider. 2003. Least privilege and more [computer security]. *IEEE Security & Privacy* 99, 5 (2003), 55–59.

[34] Bojan Suzic. 2016. Securing integration of cloud services in cross-domain distributed environments. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 398–405.

[35] Bojan Suzic. 2016. User-centered security management of API-based data integration workflows. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 1233–1238.

[36] Bojan Suzic. 2017. Structure and authorization management of RESTful APIs: Accompanying data set for the paper published at ACM SAC 2018. (2017). http://www.daspsec.org/soap18

[37] Bojan Suzic, Andreas Reiter, and Alexander Marsalek. 2017. Structuring the Scope: Enabling Adaptive and Multilateral Authorization Management. In *Communications and Network Security (CNS), 2017 IEEE Conference on*. IEEE.

[38] Wei Tan, Yushun Fan, Ahmed Ghoneim, M Anwar Hossain, and Schahram Dustdar. 2016. From the Service-Oriented Architecture to the Web API Economy. *IEEE Internet Computing* 20, 4 (2016), 64–68.

[39] Ruben Verborgh, Andreas Harth, Maria Maleshkova, Steffen Stadtmüller, Thomas Steiner, Mohsen Taheriyan, and Rik Van de Walle. 2014. Survey of semantic description of REST APIs. In *REST: Advanced Research Topics and Practical Applications*. Springer, 69–89.

[40] Hui Wang, Yuanyuan Zhang, Juanru Li, and Dawu Gu. 2016. The Achilles heel of OAuth: a multi-platform study of OAuth-based authentication. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 167–176.