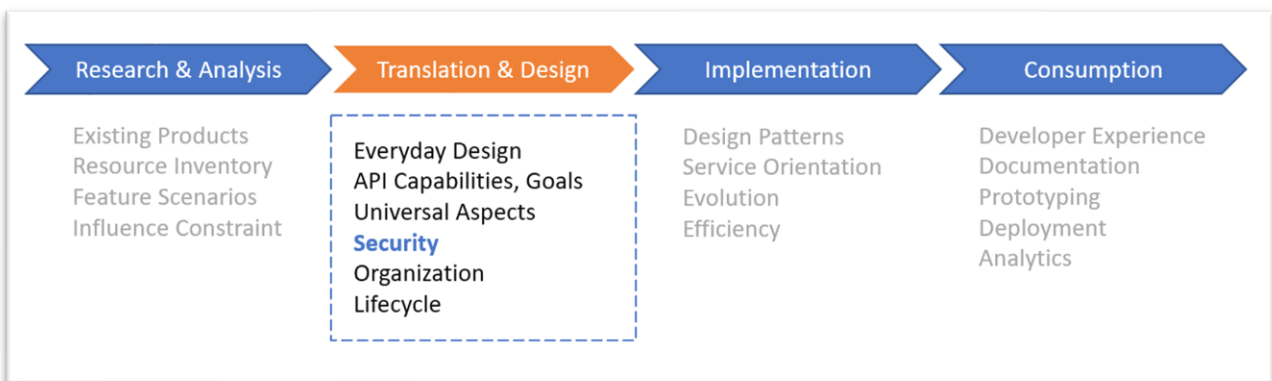


Process overview

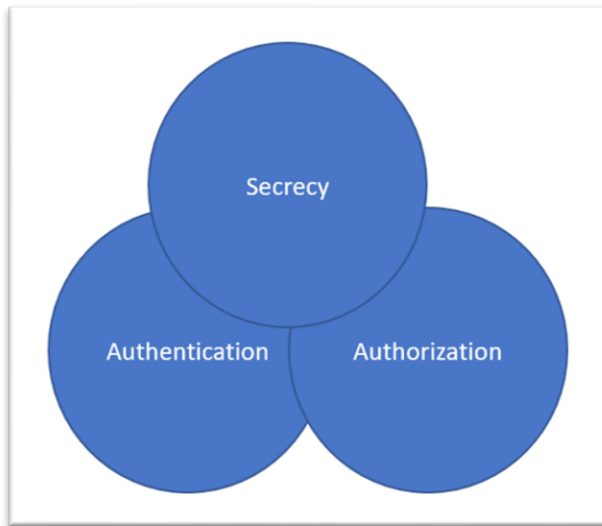
The approach to creating an enterprise-caliber API involves a set of well-defined process and roles. There are distinct steps that first identify desired capabilities and then progress to a complete solution. These phases require an in-depth understanding of the longer-term objectives, how best to establish a solid foundation, as well as foresight in allowing for growth, adaptability, and flexibility. There are general or overarching aspects that have a broad scope. This will include performance, security, utility, and technical conventions. There are also very nuanced and niche aspects that have a very limited focus. In this regard, there is a considerable impact on the later use, adoption, and lifecycle roadmap for future utility. Lastly, as expected with most software engineering or development projects, a formal iterative sequence will underly the delivery usually seen through a versioning methodology. In this breakout text, we will take look at the process drivers as whole but focus on the overarching design concern of what makes up API security.

The API development process is not detached from a standard software project approach. One key difference, however, is that the preliminary research and analysis may be centered around an existing offering or suite of solutions already in place. In many ways, this will greatly accelerate and support the initial efforts. Having an established reference point does introduce a special type of drawback and risk. The objectives and benefits of producing the API may require a great deal of translation from the current concept in order to fulfill the end-use goals. Security and related access requirements will undoubtedly align, but additional needs will present during the API design phase.



As it applies to security, we can look at the core considerations as three areas of interest: secrecy, authentication, and authorization. A more granular discussion of course yields a more comprehensive insight as it applies to the topic, but those three items are suitable to frame what will be most relevant in the API design. Secrecy is simply the ability to control information as it is exchanged, stored, or retrieved. For the most part, this concern is addressed through the use of encrypted internet connections via SSL (secured sockets layer) and does not require an extra-ordinary decomposition. A method that prevents eavesdropping is a sufficient model for the discussion purpose. Authentication and authorization are very much complimentary, however the two are quite distinct. Additionally, the granularity and *modality* of these aspects will define how the API security is addressed. Most simply put, authentication is the resolution of an identity. That identity may represent a user account, that identity may represent a more abstract entity. In either case, the result of authentication is a reference

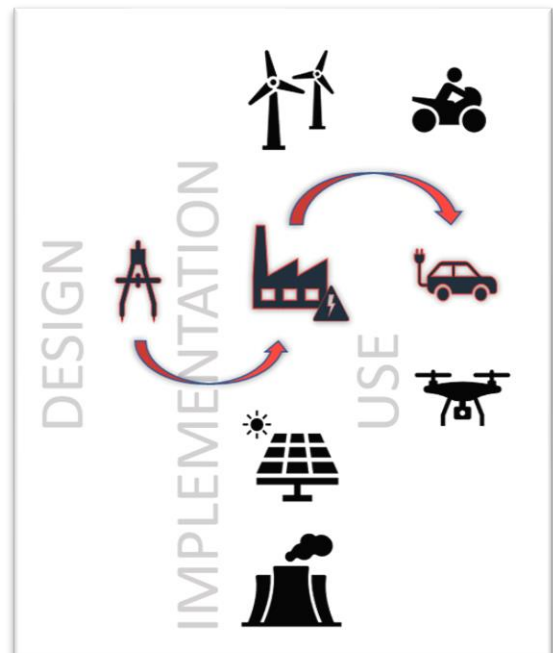
to a reliably determined identity that is later used in the grant or denial of resources. The subsequent grant or denial of resources for a particular identity then constitutes authorization.



It is important to indicate that secrecy, authentication, and authorization may each be achieved through a variety of methods, protocols, platforms, and conventions. In the same progression sequence, the intricacies associated with each are more pronounced as applicable to designing, implementing, and consuming an API. Excluding secrecy (since this can be covered by SSL internet connections), authentication and authorization will garner the majority of attention for the security implications.

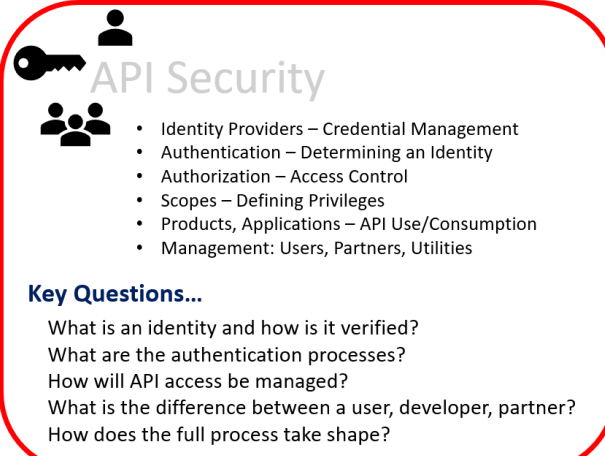
Before describing how deeply authentication and authorization influence the API as a whole, it is best to revisit how the process flows from initial research

to concluding usage. Keep in mind that the API objectives, denoted as “Goals”, will likely differ subtly or substantially from its counterpart in an existing offering/application. In using the OpenAPI standard (www.openapis.org) as the accepted basis for the API delivery, strong delineation occurs between design, implementation, and use. Furthermore, although practically all technical concerns have a touchpoint in the design, the implementation still must take place to provide the expected utility. With the OpenAPI standard, the prescribed design language, technical documentation formats, and convention governance truly form the blueprint dictating the later implementation. Lastly, it is also necessary to differentiate and clarify the separation between implementation and usage. End-use or usage can be interchange more generally as API (service) consumption. The API consumption is usually in the context of and application or utility that operates as part of a larger constituent. An API consumer relies on the design (blueprint), as an assurance of capabilities, routines, input|outputs, and conventions. The API implementation is the delivery of the design-defined capabilities “to-specification” such that **any** consumer would leverage the service(s) as described in the design. The OpenAPI standard puts formal emphasis on the design language, convention, and artifacts as the “glue” that binds the subsequent implementation to use scenarios.



Think of the completed/implemented API as an electric power utility provider. There are a set of standards, conventions, specifications that are mandated by the design. The design will dictate the “what” while the implementation has dominion over the “how”. The users of the electric power utility,

regardless of their varying scenarios/situations, are assured that the **implementation** will adhere to assertion made in the design. Regardless of the “how”, the specification of the “what” is guaranteed in that regard – the power outlet is the same although the source of the electricity may differ.

A diagram titled "API Security" enclosed in a red rounded rectangle. It features a key icon and a group of three people icons. Below the title is a bulleted list of API security topics. At the bottom, under the heading "Key Questions...", are five questions related to API security.

API Security

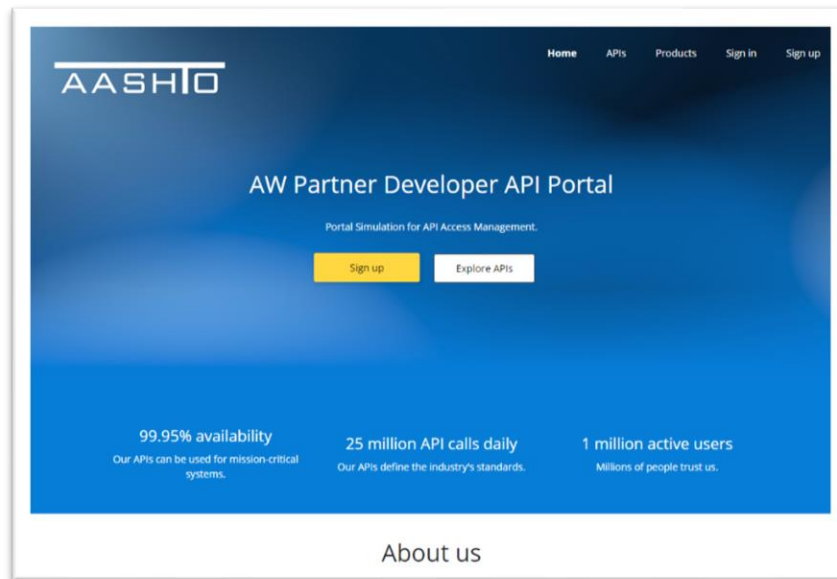
- Identity Providers – Credential Management
- Authentication – Determining an Identity
- Authorization – Access Control
- Scopes – Defining Privileges
- Products, Applications – API Use/Consumption
- Management: Users, Partners, Utilities

Key Questions...

- What is an identity and how is it verified?
- What are the authentication processes?
- How will API access be managed?
- What is the difference between a user, developer, partner?
- How does the full process take shape?

How does this apply to security? Is security incorporated into the design definition or is this an implementation concern? Is there any variation based on the API consumer? Can you limit API consumers? What if there is an existing authentication and authorization basis – how is that reflected with the API capabilities? There are number of pertinent questions around how security will take shape in an API. This will also span and influence the design, implementation, and use. Here are the starting points and examples to put in perspective. Security is applicable to design, implementation, and use –

there is no surprise that this is the case. Additionally, there are different technical responsibilities and considerations in each. Organization of capabilities, existing credentials’ managements, and intended consumption channels will all influence the API security profile. At the design tier, a mandate for authentication can be added as a requirement for implementation. For example, authentication by username & password (or alternate means) can be enforced as a prerequisite in using some or all of the API routines. However, the implementation must enforce this design aspect. Furthermore, the use/consumption must support the preconditions defined in the API design (e.g. collection of username and password before initiating an action). Authorization granularity will then become very much dependent on the feature scenarios. There is an expected partitioning of the lowest level granularity in data management that is just outside the scope of what would be defined in the API design. Consider this: a hypothetical API design can outline a generalized basis for online banking and simple financial transactions. An authentication requirement is included, a set of permissions to perform certain tasks are combined to establish a **role**, and then a formal registration process is created for application developer to use this API. Individual financial institution would have the implementation responsibility to adhere to the API design basis. The ability to “manage account profile” can be defined with the API as part of a permissions scope, implemented by the financial institution, and the consumed by a 3rd party “financial toolkit” app. The expectation is that once a username and password is provided, the application will allow simple updates to the account profile. The nuance and specific implementation component around security in this case is that **only** the profile of the corresponding **authenticated** user can be modified.



1- Auto Generated Developer/Partner Portal, API Consumption

As part of a later topic breakout, we will discuss the process and management of the aforementioned “3rd party” app vendor. This entity has the charter to create an application that **consumes** the API, as implemented, with the expected capabilities as enumerated in the design documentation. Applications that consume the API are usually, but not exclusively, produced by an entity outside of the implementing organization – often through a curated “partner” program. It is

important to note that, API access requires a supplemental layer of “gatekeeping” and restraints such that even the consuming applications are duly tracked. This is easily achieved through the creation of a partner/developer portal where any vendor application must first “register” through a process. The extends the security concepts discussed but it is a supplement outside of the design|implementation|use workflow.

API Design Technical Elements: Authentication, Authorization, Implementation

OpenAPI 2.0 vs 3.0

The Phase 2 engagement is based on the OpenAPI 3.0 standard which includes guidance, practices, and conventions to establish Authentication and Authorization specifications as part of the API design purview. As compared to the prior OpenAPI 2.0 standard, the OpenAPI 3.0 standard includes a greater breadth and depth of contexts, with particular interest to enterprise scenarios, that can now be fully addressed. These standardizations in extension of methodologies that can now described are reflective of new capabilities in other standards (OAuth, OpenID) in addition to large-scale adoption and simplification of use.

The primary OpenAPI capabilities in design descriptions for the 2.0 standard were limited to Authentication. Additionally, the authentication mechanisms then were still a subset of what are now considered commonplace. The subsequent definition of authorization as an access control descriptor that aligns to more contemporary usages is definitely one of the advancements of the 3.0 standard compared to its predecessor. For completely “public” API scenarios – neither the authentication nor authorization points are applicable. However, for even the most basic application that exposes functionality via OpenAPI, authentication and authorization should be addressed within a rudimentary design context. For enterprise applications, these aspects are fairly common universal design requirements.

Phase 2A Related Tasks

Within the Phase 2a design efforts, and specifically to deliverables within tasks 1,2,3: Authentication and Authorization specifications have touchpoints. The most relevant technical concerns are incorporated into the design definitions, while at the same time allowing for methods that may not be as applicable for example or training exercises. For example, referencing and including “basic” (user name & password) and enterprise SSO (OpenID) scenarios are ideal based on the AW product line, Phase 1 conclusions, and “future-focused” technical outlook. The API design basis will also allow, support, and provide guidance for authentication by x509 certificate or generated “secret keys” but the practical examples and training exercise demonstrations will focus on the more applicable username|password and MS Office 365 illustrations. Lastly, to again provided a real-world scenario for training, the authorization concept within the API design (and to the OpenAPI 3.0 standard), will be aligned to be familiar again constructs within AW products. This roughly maps to defining “roles” and securables for those roles.

Wherever possible – any/all of the conceptual implementation points that are in the AW API design, will be “recycled” as the example illustration for the training demos around the design|implementation|usage theme.

OpenAPI 3.0 authentication|authorization will entail:

- Basic Authentication
- Token/API-Key
- Bearer Authentication
- OpenID (OAuth – MS Ent SSO, Google, etc)
- Cookie Persisted

Context

The API design and reference materials will focus on the basic authentication and the OpenID (connect) modes as this will be most applicable for implementation and most beneficial for use. That singular design, sample implementation, and usage flow will be part of the training exercise(s) in the early sessions. This is a cross-cutting concerning that is fairly universal and not unique to a typical enterprise application.