Design Secure APIs

Technical specifications and tools from the API Italian Interoperability Framework

EuroPython 2021

Roberto Polli
API Ecosystem
Agenda

Enforce secure design practices and coherent interfaces for services provided by 20k+ agencies

➔ Digital Interfaces Challenges
➔ API Guidelines
➔ A roundup of useful standards on HTTP headers, content and authentication
➔ Open source Online Validator
Standardizing all public sector APIs

Guidelines can uniform APIs produced by thousands of service providers

60M People
+12k Public Agencies
+8k Cities
20 Regions
(∞ cultural heritage)
Secure and usable by design with API Guidelines

To achieve **reliable, secure** and **consistently designed services** Italy wrote API Guidelines and support tools

**Risks**

- **over-complexity**: bureaucratic, non-digital processes are mapped to convoluted APIs without a proper redesign
- **time-constrained engineering**: a restricted group of people addressing the above use-cases within a short deadline

**Mitigations**

- **Interface Description Languages**: a formal description of API interactions, eg: OpenAPI (HTTP) and WSDL (SOAP).
- **API Guidelines**: to uniform the design and security of REST and SOAP services between 12k agencies, together with **tools** to help agencies and their suppliers in checking their design. **Further tools** will be provided.
Security basics

Using OpenAPI3 simplifies a broad set of design checks, including some of the OWASP API Security top 10

- **HTTPS** - checks that all URLs in the spec use the https scheme
- **Authentication and authorization** - checks that every endpoint is properly protected
- **Use HTTP methods correctly** - for example checking that PATCH requests have a suitable media-type, eg. application/merge-patch [rfc7386](https://tools.ietf.org/html/rfc7386)
- **RateLimit (OWASP API4:2019)** - define and enforce a coherent ratelimit framework such as [draft-ietf-httpapi-ratelimit-headers](https://tools.ietf.org/html/draft-ietf-httpapi-ratelimit-headers)
HTTP Headers

Adding Content Security Policy / Strict Transport Security and removing Server information is not enough. You need to document relevant headers usage too!

- **Cache-Control** - clarify in the specification how do you use cache
- **Authorization** - describe authentication and authorization policies using secure flows of OAuth2 and OIDC
- **Limit header parsing complexities** - eg. RFC8941 defines a safe serialization model for header values. There's a python library too!

```python
from http_sfv import Dictionary as D

h = str(D({"a": 1, "b": "two", "c": b'\01\02\03'}))
print(h)  # 'a=1, b="two", c=:AQID:'

d = D()  # initialize a dictionary
d.parse(b'a=1, b="two", c=:AQID:' )
d['b'].value  # "two"
```
Use interoperable subsets of JSON and XML

json-schema and XSD can model complex data-types, support nested structures and implementations have many nuances.

Some JSON hints:

➔ utf-8 only - RFC 8259

➔ encode floats/bigint as strings - rfc7493#section-2.2

>>> json.dumps({'"1^1000": 1e1000', '"1^1000": Infinity'}

➔ beware of duplicate names - rfc7493#section-2.3

>>> json.loads({"x": 1, "x": 2})
{"x": 2}

➔ use strict parsers and don't truncate characters - consider that client and server will probably use different libraries. Custom parser may be less secure.

>>> custom_json_parser.loads({"u": "ro\ud888ot"})
{"u": "root"}

Read I-JSON specs RFC7493.

For XML, see the comprehensive OWASP XML Security Cheat Sheet
Limit item values and occurrences.

Limit numbers and strings - in practice, every schema is limited in size. Start with reasonable values, then raise up.

Limit array sizes - arrays should be limited too

**Uid:**
- type: integer
- minimum: 0
- maximum: 1000

**Title:**
- type: string
- minLength: 5
- maxLength: 128
- pattern: '^[a-zA-Z0-9 ]$'

**Titles:**
- type: array
- minItems: 1
- maxItems: 100
- uniqueItems: true
- items: {required: true, $ref: '#/Title'}

Constrain json-schema objects

- Limit numbers and strings - in practice, every schema is limited in size. Start with reasonable values, then raise up.

Object validation is very tolerant

Account:
- type: object
- properties:
  - user: {required: true, $ref: '#/Uid'}

Specify required properties - or objects will always validate.

Account:
- type: object
- required: [user]
- properties:
  - user: {required: true, $ref: '#/Uid'}

Disable additional properties - if you don't want unexpected fields

Account:
- type: object
- required: [user]
- properties:
  - user: {required: true, $ref: '#/Uid'}
  - titles: []
JWT and OAuth2

RFC8725 defines security best practices for JSON Web Tokens, and OAuth2 deprecated insecure flows.

JSON and XML flexibility increases their attack surface

JWT Best Current Practices (RFC8725):
- use utf-8 (again :)
- use and verify appropriate algorithms, avoid substitution attacks
- use / validate the audience, issuer and subject claims
- don’t trust received claims

More JWT hints:
- limit temporal validity with nbf and exp claims
- add a jti claim to identify replay attacks
- don’t use TLS certs for JWT to avoid cross-protocol attacks

OAuth2 hints:
- don’t use "implicit" and "resource owner password" flows
- use "authorization code with PKCE" and "client credentials" with jwt-bearer client_assertion_type (RFC7523)
- limit access token requests to specific resources using RFC8707
APIs interactions and data schemas must be formally defined in "specification files" using an Interface Description Language. We can validate those files using automatic tools like `italia/api-oas-checker`!

- **security**: avoid common errors in API design (under-defined schemas, insecure methods, ...)
- **standards**: verify that Internet Standards are used correctly
- **usability**: the design is consistent with respect to the API domain and other usability rules (eg. field names, methods, ...)

/echo:
  get:
    summary: Ritorna un timestamp in formato RFC3424.
    description: Ritorna un timestamp in formato RFC3424 prestando dal server attuale.
    operationId: get_echo
    tags:
      - public
    responses:
      *: common-responses
      200:
        description: The current timestamp is returned.
        headers:
          *: ratelimit-headers
      content:
        application/json:
          schema:
            type: object
            description: Un Timestamp in RFC5424
            required:
              - timestamp
            properties:
              timestamp:
                type: string
                format: date-time
                example: '2018-12-30T12:23:32Z'

components:
  securitySchemes:
    JWT:
      type: oauth2
      description: A brief description about JWT usage.
      flows:
        clientCredentials:
          tokensUrl: https://oauth2.example
  schemas:
    Problem:
      $ref: '#/components/schemas/Problem'
    X-Ratelimit-Limit: $ref: '#/components/schemas/X-Ratelimit-Limit'
    X-Ratelimit-Remaining:
The Framework

Guideline support tools

Coherent and secure by design, integrating checks in your IDE.
Challenges

➔ **Usability**: improve the web interface, which is the showcase of the API Guidelines

➔ **Security**: create a community around the identification and implementation of more security rules

➔ **Coherence**: improve the coverage of the Italian API Guidelines and evolve the project together with the framework

➔ **Community**: hand over the project to Developers to make it grow, contribute to underlying projects

Next Steps

Involve communities, countries and API experts in the project