API
Stylebook

OWASP Top 10

Export

Set of rules to enforce OWASP security guidelines.

To use these rules:

1. Go to your Stoplight workspace.
2. Create a style guide project OR edit a project that has an API.
3. Select **Manage Style Guides**.
4. Enable `OWASP Top 10` from list of public style guides.

You can then:

- Use the style guide as-is to automatically lint your API files
- Disable individual rules that do not follow your organization's standards
- Reuse and customize rules to meet your needs

If you have suggestions on how to improve the ruleset or find any bugs, you can open an issue here.

owasp:api1:2019-no-numeric-ids

Path parameters must use random IDs that cannot be guessed, such as UUIDs.

Invalid Example

In this example, the `{userId}` parameter has a type of `integer`.

```
paths:
  '/users/{userId}':
    parameters:
      - schema:
          type: integer
```

API
Stylebook

```
          name: userId
          in: path
          required: true
          description: Id of an existing us
```

Valid Example

In this example, the `{userId}` parameter has a type of `string` with a format of `uuid`.

```
paths:
  '/users/{userId}':
    parameters:
      - schema:
          type: string
          format: uuid
        name: userId
        in: path
        required: true
        description: Id of an existing us
```

owasp:api2:2019-auth-insecure-schemes

Security scheme must use a secure method.

`negotiate` and `auth2` are considered to be insecure security schemes.

Invalid Example

This example is invalid because `oauth` is considered an insecure scheme.

```
securitySchemes:
   Oauth1:
     type: http
     scheme: oauth
```

Valid Example

```
securitySchemes:
   Bearer:
     type: http
     scheme: bearer
```

API Stylebook

owasp:api2:2019-jwt-best-practices

Security scheme description must state that the implementation conforms with JSON Web Tokens RFC7519, the JSON Web Token standard.

Invalid Example

This example is invalid because RFC8726 is not included in the security scheme description.

```
JWTBearer:
      type: oauth2
      flows:
        authorizationCode:
          ...
          ...
          ...
          ...
      description: A bearer token in the
```

Valid Example

```
JWTBearer:
      type: oauth2
      flows:
        authorizationCode:
          ...
          ...
          ...
          ...
      description: A bearer token in the
```

owasp:api2:2019-no-api-keys-in-url

Security scheme must not contain API Keys in query parameters.

API Keys are (usually opaque) strings that can be eavesdropped, especially when they are passed as URL parameters.

Invalid Example

The `in:query` setting makes this example invalid.

```
securitySchemes:
```

API Stylebook

```yaml
    API Key:
      name: API Key
      type: apiKey
      in: query
```

Valid Example

The `in:header` makes this example valid.

```yaml
securitySchemes:
    API Key:
      name: API Key
      type: apiKey
      in: header
```

---

owasp:api2:2019-no-credentials-in-url

Path parameter must not contain credentials, such as API key, password, or secret.

Invalid Example

This example is invalid because the path parameter includes a string with the name `password`.

```yaml
paths:
  '/user/{password}':
    parameters:
      - schema:
          type: string
          format: password
        name: password
        in: path
        required: true
```

Valid Example

Remove the invalid path parameter.

```yaml
paths:
  '/user/
```

---

owasp:api2:2019-no-http-basic

Security scheme must not use basic auth. Use a more secure authentication method, such as OAuth

API Stylebook

2.0.

Invalid Example

```
securitySchemes:
    basicAuth:
      type: http
      scheme: basic
```

Valid Example

```
securitySchemes:
    OAuth2:
      type: oauth2
      flows:
        ...
          ...
          ...
          ...
          ...
```

owasp:api2:2019-protection-global-unsafe

POST. PUT, PATCH, and DELETE operations must be protected by a security scheme at either the global level or operation level.

Security rules are defined in the `securityScheme` section.

Valid Example: Global

```
securitySchemes:
    API Key:
      name: API Key
      type: apiKey
      in: header
security:
  - API Key: []
```

*Valid Example: Operation

```
paths:
  '/users/{userId}':
    patch:
      ...
```

API
Stylebook

```
      responses:
        ...
      security:
        - API Key: []
```

owasp:api4:2019-array-limit

Array size should be limited to mitigate resource exhaustion attacks. This can be done using `maxItems`. You should ensure that the subschema in `items` is constrained too.

owasp:api4:2019-integer-format

Integers should be limited to mitigate resource exhaustion attacks. Specifying whether int32 or int64 is expected via `format`.

owasp:api4:2019-integer-limit

Array size should be limited to mitigate resource exhaustion attacks. This can be done using `maxItems`. You should ensure that the subschema in `items` is constrained too.

owasp:api4:2019-integer-limit-legacy

Array size should be limited to mitigate resource exhaustion attacks. This can be done using `maxItems`. You should ensure that the subschema in `items` is constrained too.

owasp:api4:2019-rate-limit

Headers for 2xx and 4 xx responses must contain `RateLimit-Limit`, `RateLimit-Reset`, `X-RateLimit-Limit`, or `X-Rate-Limit-Limit`.

Proper rate limits avoid attackers overloading the API. There are many ways to implement rate-limiting, but most of them involve using HTTP headers, and there are two popular ways to do that:

IETF Draft HTTP RateLimit Headers:.
https://datatracker.ietf.org/doc/draft-ietf-httpapi-

ratelimit-headers/

Customer headers like X-Rate-Limit-Limit (Twitter: https://developer.twitter.com/en/docs/twitter-api/rate-limits) or X-RateLimit-Limit (GitHub: https://docs.github.com/en/rest/overview/resources-in-the-rest-api)

Invalid Example

The 200 response does not contain rate-limiting headers.

```
responses:
'200':
   description: User Not Found
```

Valid Example

The 200 response contains rate-limiting headers.

```
responses:
  '200':
    headers:
      RateLimit-Limit:
        description: The number of allowe
        schema:
          type: integer
      RateLimit-Reset:
        description: The number of second
        schema:
          type: integer
```

owasp:api4:2019-rate-limit-retry-after

Headers for 429 responses must contain `Retry-After`.

Invalid Example

```
'429':
      description: Too Many Requests
      headers:
        RateLimit-Limit:
          ...
        RateLimit-Reset:
```

API
Stylebook

```
          ...
```

Valid Example

```
'429':
   headers:
    RateLimit-Limit:
      ...
    RateLimit-Reset:
      ...
    Retry-After:
      description: The number of seconds
        schema:
          type: integer
```

---

owasp:api4:2019-string-limit

String size should be limited to mitigate resource exhaustion attacks. This can be done using `maxLength`.

---

owasp:api4:2019-string-restricted

To avoid unexpected values being sent or leaked, ensure that strings have either a format or a RegEx pattern. This can be done using `format` or `pattern`.

---

owasp:api3:2019-define-error-responses-401

Operation must have a 401 response defined.

Invalid Example

```
get:
  summary: Get User Info by User ID
  tags: []
  responses:
    '200':
       ...
    '400':
      ...
    '501':
      description: Bad Gateway
      headers:
        ...
```

API
Stylebook

Valid Example

```yaml
get:
  summary: Get User Info by User ID
  tags: []
  responses:
    '200':
       ...
    '400':
      ...
    '429':
        ...
    '401':
      description: Not Authenticated
      headers:
        ...
```

owasp:api3:2019-define-error-responses-500

Operation must have a response defined.

Invalid Example

```yaml
get:
  summary: Get User Info by User ID
  tags: []
  responses:
    '200':
       ...
    '400':
      ...
    '501':
      description: Bad Gateway
      headers:
        ...
```

Valid Example

```yaml
get:
  summary: Get User Info by User ID
  tags: []
  responses:
    '200':
       ...
    '400':
      ...
    '429':
```

API
Stylebook

```
          ...
    '500':
      description: Internal Server Error
      headers:
        ...
```

owasp:api3:2019-define-error-validation

Operation must have a 400, 422 or 4xx response defined.

Invalid Example

```
get:
  summary: Get User Info by User ID
  tags: []
  responses:
    '200':
      ...
    '404':
      description: User Not Found
      headers:
        ...
```

Valid Example

```
get:
  summary: Get User Info by User ID
  tags: []
  responses:
    '200':
      ...
    '400':
      description: Bad Request
      headers:
      ...
```

owasp:api4:2019-rate-limit-responses-429

Operation must have a 429 response defined.

Invalid Example

```
get:
  summary: Get User Info by User ID
  tags: []
  responses:
```

API
Stylebook

```
    '200':
      ...
    '400':
     ...
    '431':
      description: Request Header Fields
      headers:
        ...
```

Valid Example

```
get:
  summary: Get User Info by User ID
  tags: []
  responses:
    '200':
      ...
    '400':
     ...
    '429':
      description: Too Many Requests
      headers:
        ...
```

---

owasp:api6:2019-constrained-additionalProperties

By default JSON Schema allows additional properties, which can potentially lead to mass assignment issues, where unspecified fields are passed to the API without validation.

---

owasp:api6:2019-no-additionalProperties

Object should not allow for additional properties, which can allow unspecified fields passed to the API without validation.

Invalid Example

In this example, `additionalProperties` are allowed on the object.

```
schemas:
   User:
     type: object
```

API
Stylebook

```
      title: User
       additionalProperties: true
       properties:
         id:
           type: integer
         firstName:
           type: string
         lastName:
           type: string
```

Valid Example

In this example, `additionalProperties` are not allowed on the object.

```
schemas:
   User:
     type: object
     title: User
     description: ''
      additionalProperties: false
      properties:
        id:
          type: integer
        firstName:
          type: string
        lastName:
          type: string
```

owasp:api2:2019-protection-global-safe

GET and HEAD operations should be protected by a security scheme at either the global level or operation level.

Security rules are defined in the `securityScheme` section.

Valid Example: Global

```
securitySchemes:
    API Key:
      name: API Key
      type: apiKey
      in: header
security:
```

API Stylebook

```
  - API Key: []
```

*Valid Example: Operation

```
paths:
  '/users/{userId}':
    get:
      ...
      responses:
        ...
      security:
        - API Key: []
```

owasp:api2:2019-protection-global-unsafe-strict

POST, PATCH, DELETE, and PUT operations should be protected by a security scheme at either the global level or operation level.

Security rules are defined in the `securityScheme` section.

Invalid Example

The PATCH operation has an empty security value so it is not protected.

```
paths:
  '/users/{userId}':
    patch:
      ...
      responses:
        ...
      security:
      - []
```

Valid Example

The PATCH operation is protected by the API Key. As an alternative, remove the empty security setting at the operation level and use global security.

```
paths:
  '/users/{userId}':
    patch:
```

API
Stylebook

```
      ...
      responses:
        ...
      security:
        - API Key: []
```

ɟn