OpenAPI as a security tool, not just documentation

Dr. Philippe De Ryck

https://Pragmatic Web Security.com

OPENAPI

Swagger™

/api/restaurants/{id}/reviews

Path Parameters

`id` integer · int32 required

Signed 32-bit integers (commonly used integer type).

Responses

> 200 OK

GET /api/restaurants/{id}/reviews — Shell Curl

```
curl 'https://restograde.com/api/restaurants/{id}/reviews'
```

Test Request

200 — Show Schema


```
[
  {
    "id": 1,
    "restaurantId": 1,
    "title": "…",
    "content": "…",
    "authorId": 1
  }
]
```


OK

OpenAPI specs support documentation

Various tools make it trivial to generate OpenAPI definitions from your code, which in turn can be transformed into interactive developer documentation pages.

I am *Dr. Philippe De Ryck*

Pragmatic Web Security
Security for developers

Founder of Pragmatic Web Security

Google Developers Experts

Google Developer Expert

SecAppDev

SecAppDev organizer

I help developers with security

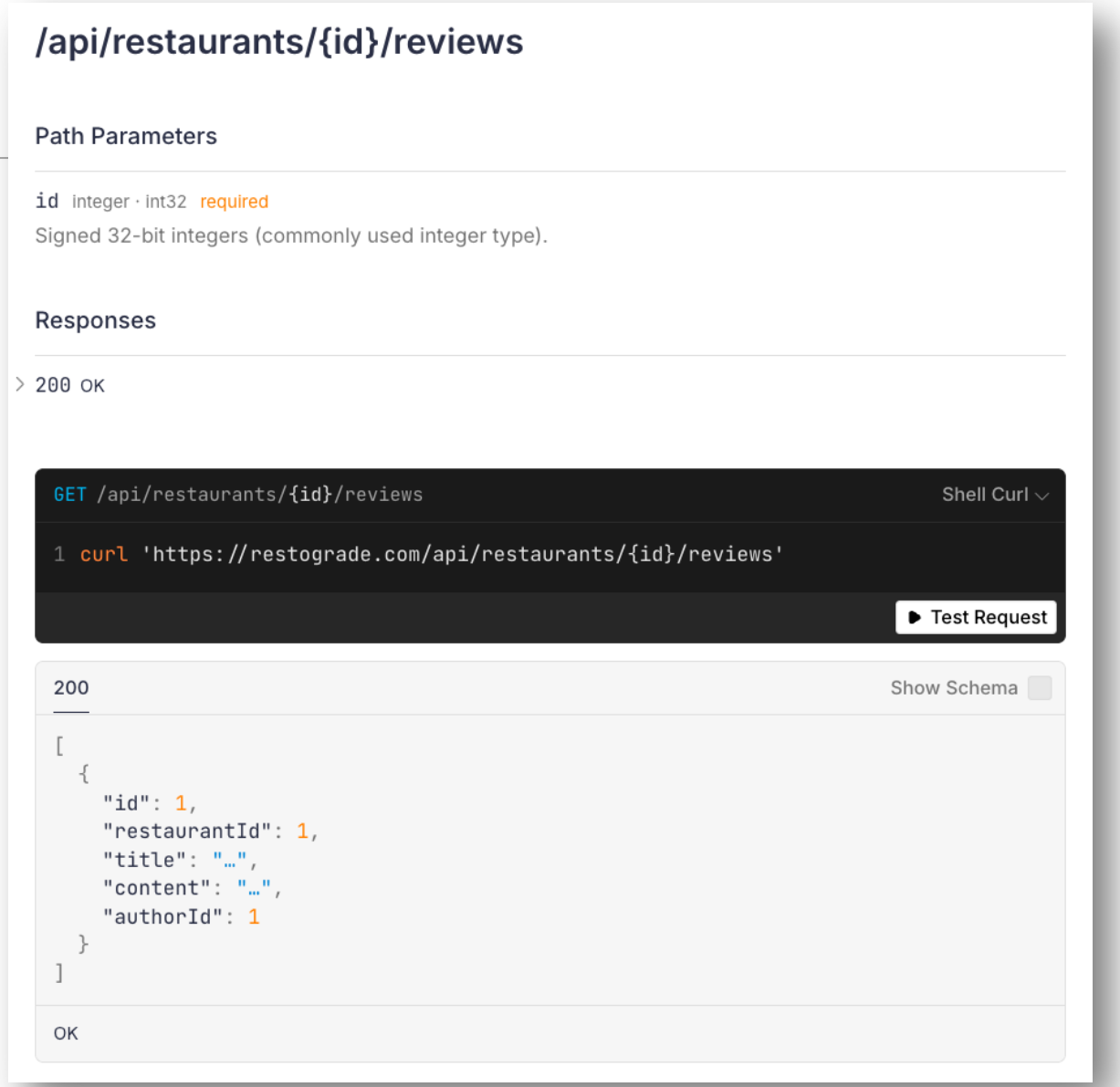
- **Hands-on in-depth security training**
- **Advanced online security courses**
- **Security advisory services**

https://pdr.online


RESTAURANT

An OpenAPI definition for a GET endpoint

```yaml
/api/restaurants/{id}/reviews:
  get:
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
          format: int32
    responses:
      "200":
        description: OK
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: "#/components/schemas/ResponseReview"
```



The endpoint definitions in code carry plenty of metadata that can be used to generate an OpenAPI specification

The Java Spring API endpoint

```java
@RequestMapping(
    path = "/api/restaurants/{id}/reviews",
    method = RequestMethod.GET,
    produces = "application/json")
public ResponseEntity<List<ResponseReview>> getReviewsForRestaurant(
    HttpServletRequest request,
    @PathVariable int id
) throws Exception {
    List<Review> reviews = DB.getReviewsForRestaurant(id);
    return new ResponseEntity<>(
        reviews.stream().map(ResponseReview::new).collect(Collectors.toList()),
        HttpStatus.OK
    );
}
```

An OpenAPI definition for a GET endpoint

```yaml
/api/restaurants/{id}/reviews:
  get:
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
          format: int32
    responses:
      "200":
        description: OK
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: "#/components/schemas/ResponseReview"
```

Augmenting your OpenAPI specifications

Enrich your OpenAPI specifications

By adding relevant data, you can enrich the OpenAPI specifications and transform them into valuable developer documents.

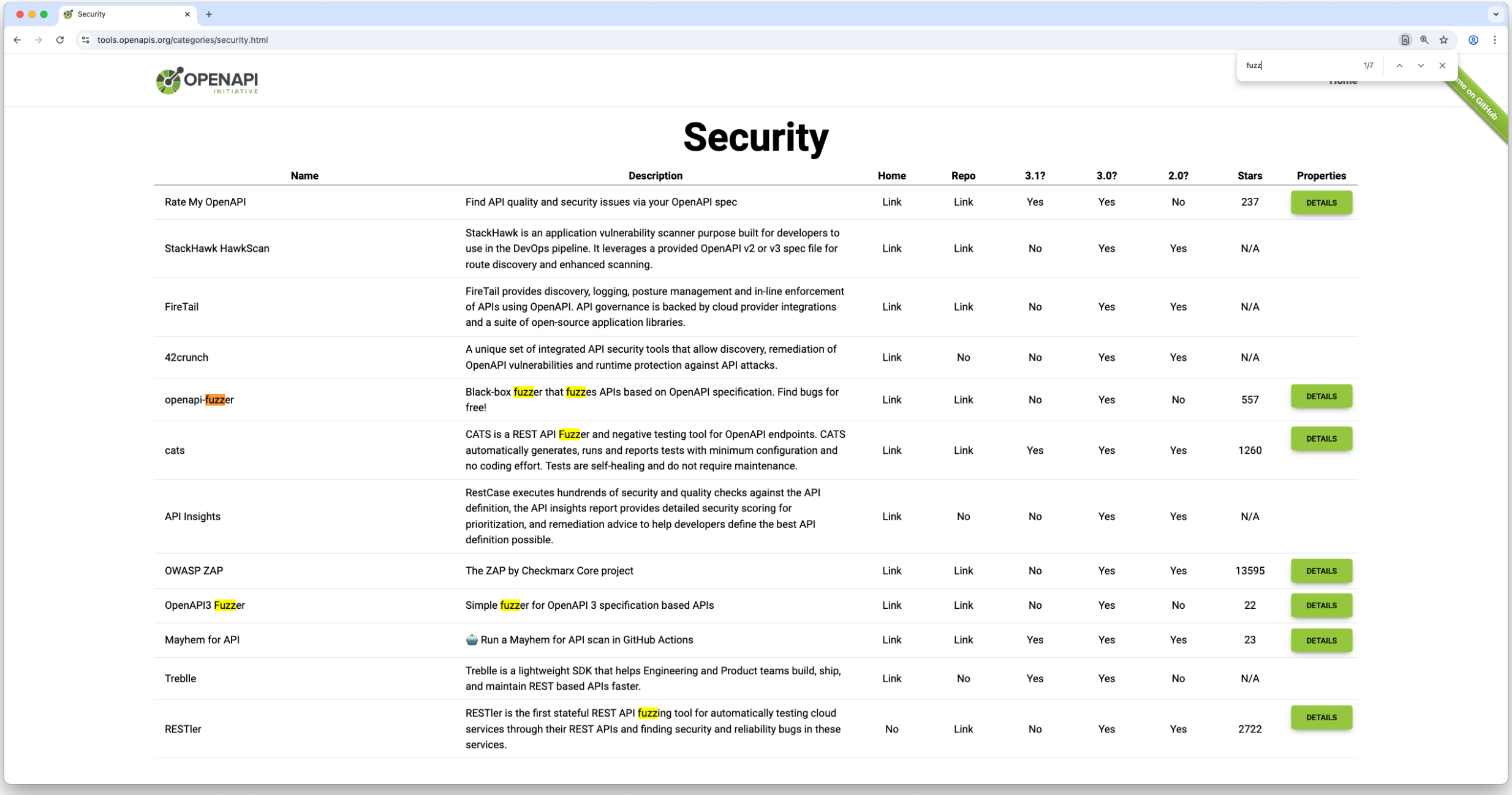
An OpenAPI definition for a GET endpoint

```yaml
/api/restaurants/{id}/reviews:
  get:
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
          format: int32
    responses:
      "200":
        description: OK
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: "#/components/schemas/ResponseReview"
```

The spec defines which HTTP methods are supported

The spec defines where parameters go and how they are formatted

The spec defines expected response codes and details



Security

Name	Description	Home	Repo	3.1?	3.0?	2.0?	Stars	Properties
Rate My OpenAPI	Find API quality and security issues via your OpenAPI spec	Link	Link	Yes	Yes	No	237	DETAILS
StackHawk HawkScan	StackHawk is an application vulnerability scanner purpose built for developers to use in the DevOps pipeline. It leverages a provided OpenAPI v2 or v3 spec file for route discovery and enhanced scanning.	Link	Link	No	Yes	Yes	N/A	
FireTail	FireTail provides discovery, logging, posture management and in-line enforcement of APIs using OpenAPI. API governance is backed by cloud provider integrations and a suite of open-source application libraries.	Link	Link	No	Yes	Yes	N/A	
42crunch	A unique set of integrated API security tools that allow discovery, remediation of OpenAPI vulnerabilities and runtime protection against API attacks.	Link	No	No	Yes	Yes	N/A	
openapi-fuzzer	Black-box fuzzer that fuzzes APIs based on OpenAPI specification. Find bugs for free!	Link	Link	No	Yes	No	557	DETAILS
cats	CATS is a REST API Fuzzer and negative testing tool for OpenAPI endpoints. CATS automatically generates, runs and reports tests with minimum configuration and no coding effort. Tests are self-healing and do not require maintenance.	Link	Link	Yes	Yes	Yes	1260	DETAILS
API Insights	RestCase executes hundrends of security and quality checks against the API definition, the API insights report provides detailed security scoring for prioritization, and remediation advice to help developers define the best API definition possible.	Link	No	No	Yes	Yes	N/A	
OWASP ZAP	The ZAP by Checkmarx Core project	Link	Link	No	Yes	Yes	13595	DETAILS
OpenAPI3 Fuzzer	Simple fuzzer for OpenAPI 3 specification based APIs	Link	Link	No	Yes	No	22	DETAILS
Mayhem for API	🤖 Run a Mayhem for API scan in GitHub Actions	Link	Link	Yes	Yes	Yes	23	DETAILS
Treblle	Treblle is a lightweight SDK that helps Engineering and Product teams build, ship, and maintain REST based APIs faster.	Link	No	Yes	Yes	No	N/A	
RESTler	RESTler is the first stateful REST API fuzzing tool for automatically testing cloud services through their REST APIs and finding security and reliability bugs in these services.	No	Link	No	Yes	Yes	2722	DETAILS

An OpenAPI definition for a GET endpoint

```
1  /api/restaurants/{id}/reviews:
2    get:
3      parameters:
4        - name: id
5          in: path
6          required: true
7          schema:
8            type: integer
9            format: int32
10     responses:
11       "200":
12         description: OK
13         content:
14           application/json:
15             schema:
16               type: array
17               items:
18                 $ref: "#/components/schemas/ResponseReview"
```

Fuzzers will try undocumented HTTP methods and ensure they are not accepted

Fuzzers will throw invalid data to the API to see what happens

Fuzzers will look at responses and report unexpected or undocumented response codes

Running fuzzing and scanning tools

Use OpenAPI specs as input for scanners/fuzzers

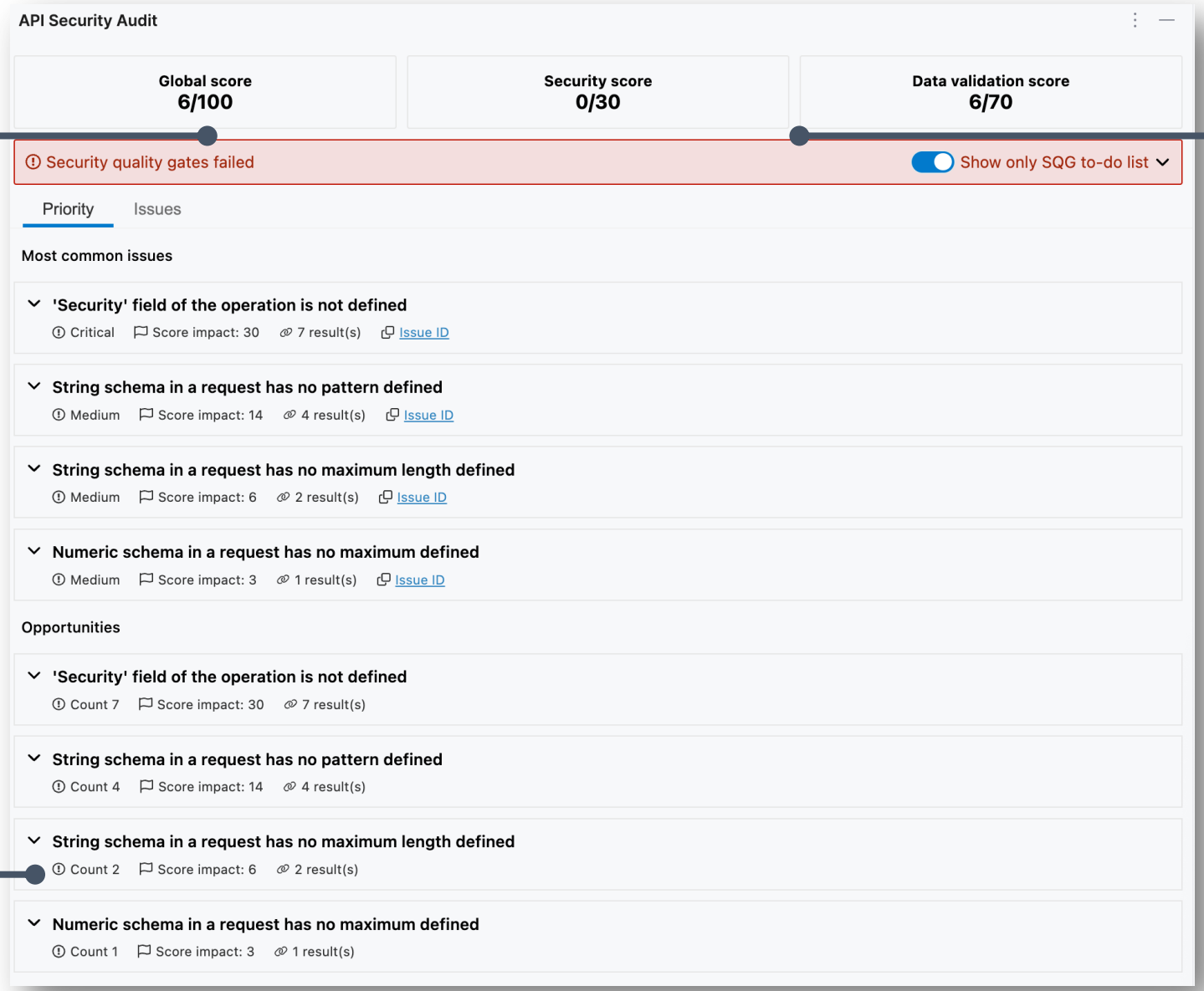
Giving scanners and fuzzers an OpenAPI spec helps them to increase specificity of tests as well as the coverage of the API.

These tools are highly useful to identify undocumented or unexpected features.

Fuzzers and scanners are vague and noisy, making them somewhat challenging to use

Auditing OpenAPI specs for security

The audit also complains about missing response codes (e.g., 406, 429, ...)



The OAS is audited for security and data validation properties

Schemas are severely underspecified in our OpenAPI spec

The 42Crunch API security platform is just an example. I am not affiliated with 42Crunch, nor do I benefit from showing this tool in any way.

A GET endpoint with data validation annotations (Java)

```java
@RequestMapping(
  path = "/api/restaurants/{id}/reviews",
  method = RequestMethod.GET,
  produces = "application/json"
)
public ResponseEntity<List<ResponseReview>> getReviewsForRestaurant(
  HttpServletRequest request,
  @PathVariable
    @Parameter(
      description = "The ID of the restaurant for which the reviews are retrieved",
      example = "1"
    )
    @Min(1)
    @Max(Integer.MAX_VALUE)
    int id
) throws Exception {
  ...
```

Framework-specific validation tools can be applied on request data, making these constraints also visible in the OpenAPI specification

A GET endpoint with data validation annotations

```java
@RequestMapping(
  path = "/api/restaurants/{id}/reviews",
  method = RequestMethod.GET,
  produces = "application/json"
)
public ResponseEntity<List<ResponseReview>> getReviewsForRestaurant(
  HttpServletRequest request,
  @PathVariable
    @Parameter(
      description = "The ID of th
      example = "1"
    )
    @Min(1)
    @Max(Integer.MAX_VALUE)
    int id
) throws Exception {
  ...
```

An OpenAPI definition for a GET endpoint

```yaml
/api/restaurants/{id}/reviews:
  get:
    parameters:
      - name: id
        in: path
        required: true
        description: The ID of the restaurant ...
        example: 1
        schema:
          type: integer
          format: int32
          minimum: 1
          maximum: 2147483647
```

The OpenAPI format supports detailed data validation properties for simple values and complex data types.

Java is cool and all,
but what about other languages?

A GET endpoint with data validation annotations (.NET)

```
1  [HttpGet("{id}/reviews")]
2  [Produces("application/json")]
3  public ActionResult<List<ResponseReview>> GetReviewsForRestaurant(
4    [FromRoute]
5    [Range(1, int.MaxValue)]
6    [SwaggerParameter(
7      Description = "The ID of the restaurant for which the reviews are retrieved",
8      Example = "1"
9    )]
10   int id)
11 {
12   ...
```

A GET endpoint with data validation annotations (Python Flask with Marshmallow)

```python
class PathParamSchema(Schema):
  id = fields.Int(
    required=True,
    validate=validate.Range(min=1, max=2147483647),
    metadata={
      "description": "The ID of the restaurant for which the reviews are retrieved",
      "example": 1
    }
  )

@blp.route('/<int:id>/reviews')
@blp.arguments(PathParamSchema, location="path")
def get_reviews_for_restaurant(args):
  ...
```

OpenAPI supports more than just data types. You can also specify error responses and authentication properties

A global error handler for generating and documenting 404 responses (Java)

```java
@ExceptionHandler(ResourceNotFoundError.class)
@ApiResponse(
  responseCode = "404",
  content = @Content(
    mediaType = MediaType.APPLICATION_JSON_VALUE,
    schema = @Schema(implementation = PublicApiError.class)
  )
)
public ResponseEntity<PublicApiError> handleNotFound(ResourceNotFoundError error) {
  return ResponseEntity.status(HttpStatus.NOT_FOUND)
          .body(new PublicApiError(error));
}
```

Being specific about which responses can be returned is a best practice.

OAS generation will pick up these annotations and document this behavior in the spec.

A global error handler for generating and documenting 404 responses (Java)

```java
@ExceptionHandler(ResourceNotFoundError.class)
@ApiResponse(
  responseCode = "404",
  content = @Content(
    mediaType = MediaType.APPLICATION_JSON_VALUE,
    schema = @Schema(implementation = PublicApiError.class)
  )
)
public ResponseEntity<PublicApiError> ha
  return ResponseEntity.status(HttpStatu
          .body(new PublicApiError(erro
}
```

Being specific about which responses can be returned is a best practice.

OAS generation will pick up these annotations and document this behavior in the spec.

The OpenAPI spec documents different responses for each endpoint

```yaml
/api/restaurants/{id}/reviews:
  get:
    ...
    responses:
      "200":
        description: OK
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: "#/components/schemas/ResponseReview"
      "404":
        description: Not Found
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/PublicApiError"
```

The OpenAPI spec documents each response code, along with the schema of its contents

Authentication

> **Note**
>
> OAS **3** This guide is for OpenAPI 3.0. If you use OpenAPI 2.0, see our OpenAPI 2.0 guide.

OpenAPI uses the term **security scheme** for authentication and authorization schemes.
OpenAPI 3.0 lets you describe APIs protected using the following security schemes:

- HTTP authentication schemes (they use the `Authorization` header):
  - Basic
  - Bearer
  - other HTTP schemes as defined by RFC 7235 and HTTP Authentication Scheme Registry
- API keys in headers, query string or cookies
  - Cookie authentication
- OAuth 2
- OpenID Connect Discovery

Follow the links above for the guides on specific security types, or continue reading to learn how to describe security in general.

OAS supports the definition of bearer tokens in the *Authorization* header, but also other mechanisms, such as session cookies

The definition of some of these mechanisms is a bit clunky and unelegant, but they add useful metadata to the OAS.

This data is used by developers and by automated security tools to understand how to authenticate requests.

https://swagger.io/docs/specification/v3_0/authentication/

Defining authentication rules in OpenAPI

AUDITING OAS IMPROVES YOUR SECURITY POSTURE

Auditing tools can use OpenAPI specs to identify lax or missing security rules, as well as missing features (e.g., 415 or 429 responses).

Advanced audits also look at security rules (e.g., authorization) defined in the OpenAPI contract.

OpenAPI Extensions

Extensions are prefixed with *x-* and are supported on various levels: the generic *info* section, for *security schemes*, for *paths* and *responses*, and for specific operation parameters.

Extensions can be used to describe specific mechanisms in more detail, beyond what the spec supports out of the box

> **Note**
>
> OAS **3** This guide is for OpenAPI 3.0. If you use OpenAPI 2.0, see our OpenAPI 2.0 guide.

Extensions (also referred to as *specification extensions* or *vendor extensions*) are custom properties that start with `x-`, such as `x-logo`. These are used to add extra information or functionality that the OpenAPI standard doesn't include by default. For example, many tools including Amazon API Gateway, ReDoc, APIMatic, and Fern use extensions to include details specific to their products.

https://swagger.io/docs/specification/v3_0/openapi-extensions/

Example

An API that uses Amazon API Gateway custom aut

to this:

```yaml
components:
  securitySchemes:
    APIGatewayAuthorizer:
      type: apiKey
      name: Authorization
      in: header
      x-amazon-apigateway-authtype:
      x-amazon-apigateway-authorizer
        type: token
        authorizerUri: arn:aws:apiga
        03-31/functions/arn:aws:lamb
        id:function:function-name/in
        authorizerCredentials: arn:a
        identityValidationExpression
        authorizerResultTtlInSeconds
```

x-amazon-apigateway-authorizer object

PDF RSS Focus mode

Defines a Lambda authorizer, Amazon Cognito user pool, or JWT authorizer to be applied for authorization of method invocations in API Gateway. This extension applies to the security definition in OpenAPI 2 and OpenAPI 3.

Property name	Type	Description
type	string	The type of the authorizer. This is a required property. For REST APIs, specify `token` for an authorizer with the caller identity parameters. Specify `cognito_user_pools` for an authorizer that use For HTTP APIs, specify `request` for a Lambda authorizer with the call
authorizerUri	string	The Uniform Resource Identifier (URI) of the authorizer Lambda functic `"arn:aws:apigateway:us-east-1:lambda:path/2015-03-31`
authorizerCredentials	string	The credentials required for invoking the authorizer, if any, in the form
authorizerPayloadFormatVersion	string	For HTTP APIs, specifies the format of the data that API Gateway sends version.
enableSimpleResponses	Boolean	For HTTP APIs, specifies whether a `request` authorizer returns a Bool enabled, the Lambda authorizer function returns a Boolean value. To le
identitySource	string	A comma-separated list of mapping expressions of the request parame
jwtConfiguration	Object	Specifies the issuer and audiences for a JWT authorizer. To learn more,
identityValidationExpression	string	A regular expression for validating the token as the incoming identity. I
authorizerResultTtlInSeconds	string	The number of seconds during which authorizer result is cached.

https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-swagger-extensions-authorizer.html

You can customize your OAS for your situation

Extending OpenAPI specifications can be useful when you have company-specific requirements.

The information defined in the extension is very useful to clarify certain details, such as detailed information about authorization tokens or credentials.

1	Broken object level authorization
2	Broken authentication
3	Broken object property-level authorization
4	Unrestricted resource consumption
5	Broken function level authorization
6	Unrestricted access to sensitive business flows
7	Server-side request forgery
8	Security misconfiguration
9	Improper inventory management
10	Unsafe consumption of APIs

API Security TOP 10

1	Broken object level authorization
2	Broken authentication
3	Broken object property-level authorization
4	Unrestricted resource consumption
5	Broken function level authorization
6	Unrestricted access to sensitive business flows
7	Server-side request forgery
8	Security misconfiguration
9	Improper inventory management
10	Unsafe consumption of APIs

API Security TOP 10

MASS ACCOUNT TAKEOVER IN THE YUNMAI SMART SCALE API

The server leaks the ‘accessToken’, and the ‘refreshToken’. As a result, we can impersonate the account.

https://fortbridge.co.uk/research/mass-account-takeover-yunmai/

Disclosure of Sensitive User Information via API

High **yogeshojha** published **GHSA-r3fp-xr9f-wv38** on Feb 1

Package	Affected versions	Patched versions
No package listed	<= 2.2.0	2.2.0

Severity

High

CVE ID

CVE-2025-24899

Weaknesses

▸ CWE-200

Description

Summary

A vulnerability was discovered in reNgine, where **an insider attacker with any role** (such as Auditor, Penetration Tester, or Sys Admin) **can extract sensitive information from other reNgine users.** After running a scan and obtaining vuln target, the attacker can obtain details such as `username`, `password`, `email`, `role`, `first name`, `last name` `activity information` by making a GET request to `/api/listVulnerability/`.

Details

After running a vulnerability scan on a random target and generating any result in the ***Vulnerabilities*** tab, an a with any role, regardless of whether it is Sys Admin, Penetration Tester or Auditor, can exploit this vulnerability

Create a query for the Endpoint `/api/listVulnerability/` using the `GET` method:

```
GET /api/listVulnerability/ HTTP/2
Host: myrengine
Cookie: sessionid=YOUR_SESSIONID
```

URL: https://RENGINE-IP/api/listVulnerability/

Note that the API response is leaking sensitive information from users who have already run a scan before. If a scan, information will only be leaked from the single user who performed the scan, but if more than one user p more than one user will have their information leaked in the response.

Request

Pretty Raw Hex

```
1 GET /api/listVulnerability/ HTTP/2
2 Host: myrengine
3 Cookie: sessionid=axmo82u[redacted]81b5s4
4
5
```

Response

Pretty Raw Hex Render


```
ection: {\r\n\r\n}\r\nscreenshot: {\r\n  'enable_h
ttp_crawl': true,\r\n  'intensity': 'normal',\r\n
 'timeout': 10,\r\n  'threads': 40\r\n}\r\n\r\n# c
ustom_headers: [\"Cookie: Test\"]",
"default_engine":true
},
"initiated_by":{
"id":1,
"password":
"pbkdf2_sha256$260000$PLie5m[redacted]
[redacted]jr1pVJTNbmcLJN3qONphg4=",
"last_login":"2025-01-15T13:36:42.112488Z",
"is_superuser":true,
"username":"raditzz",
"first_name":"",
"last_name":"",
"email":"r[redacted]@gmail.com",
"is_staff":true,
"is_active":true,
"date_joined":"2025-01-12T22:59:54.733417Z",
"groups":[
],
"user_permissions":[
]
},
"aborted_by":null,
"emails":[
],
"employees":[
],
```


Search 0 highlights

Search 0 highlights

Excessive data exposure in action

An OpenAPI definition for listing restaurants

```
1  /api/restaurants:
2    get:
3      responses:
4        "200":
5          description: OK
6          content:
7            application/json:
8            schema:
9              type: array
10             items:
11               $ref: "#/components/schemas/Restaurant"
```

The contents are defined in the *Restaurant* schema

An OpenAPI definition for listing restaurants

```yaml
/api/restaurants:
  get:
    responses:
      "200":
        description: OK
        content:
          application/json:
          schema:
            type: array
            items:
              $ref: "#/components/schemas/Restaurant"
```

An OpenAPI definition for the Restaurant schema

```yaml
Restaurant:
  type: object
  properties:
    id:
      type: integer
      format: int32
    owner:
      $ref: "#/components/schemas/User"
    name:
      type: string
    description:
      type: string
```

This OAS suffers from excessive data exposure by leaking internal user fields in the restaurant's owner property

An OpenAPI definition for the User schema

```yaml
User:
  type: object
  properties:
    id:
      type: integer
      format: int32
    username:
      type: string
    password:
      type: string
    role:
      type: string
```

Excessive data exposure can be solved by using response-specific DTOs or schemas

OAS HELPS TO IDENTIFY POTENTIAL VULNERABILITIES

Having a clearly-defined contract of what data the API returns helps in identifying potential data exposure vulnerabilities.

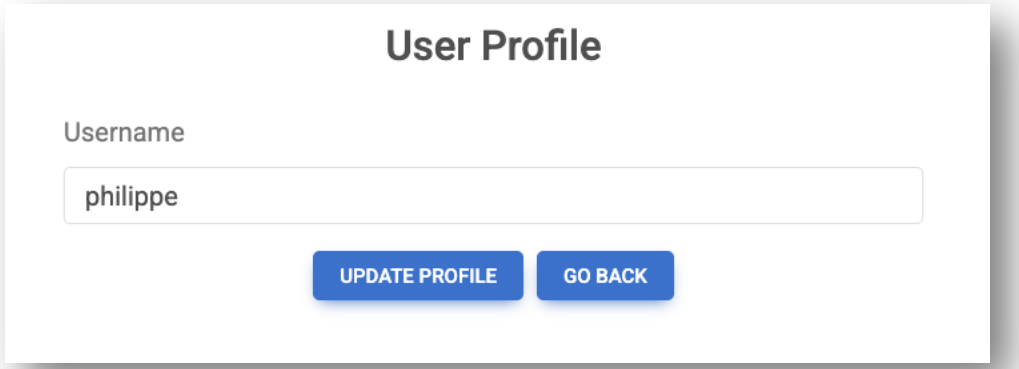
Note that identifying vulnerabilities still requires manual review of the generated OpenAPI spec.

1	Broken object level authorization
2	Broken authentication
3	Broken object property-level authorization
4	Unrestricted resource consumption
5	Broken function level authorization
6	Unrestricted access to sensitive business flows
7	Server-side request forgery
8	Security misconfiguration
9	Improper inventory management
10	Unsafe consumption of APIs

API Security TOP 10

In a *mass assignment* vulnerability, a user can write/update properties they should not be accessing

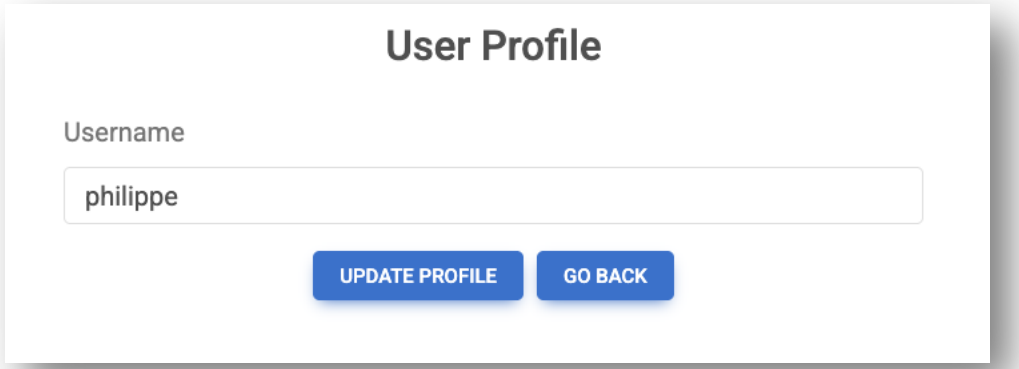
Mass assignment in action



The UI for updating user profiles supports the changing of the username, but the underlying API accepts full user objects as input

An OpenAPI definition for updating the user profile

```
patch:
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/User"
```



An OpenAPI definition for updating the user profile

```yaml
patch:
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/User"
```

An OpenAPI definition for the User schema

```yaml
User:
  type: object
  properties:
    id:
      type: integer
      format: int32
    username:
      type: string
    password:
      type: string
    role:
      type: string
```

This OAS suffers from mass assignment, as it allows a user to overwrite internal fields, leading to an escalation of privilege

Mass assignment can be solved by using request-specific DTOs or schemas

OAS HELPS TO IDENTIFY POTENTIAL VULNERABILITIES

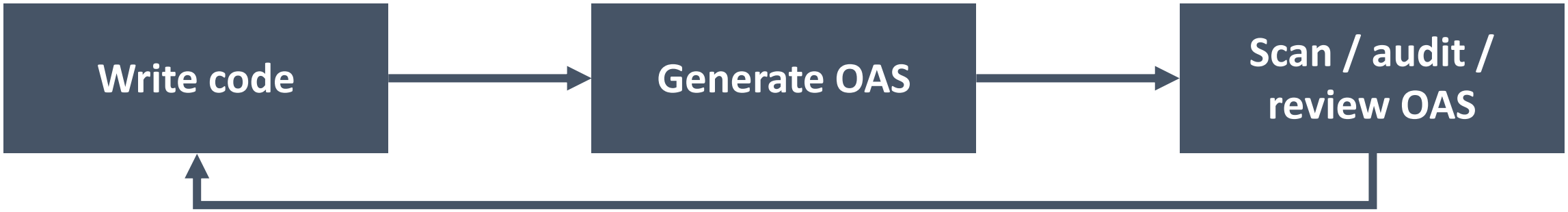
Having a clearly-defined contract of what data the API accepts helps in identifying potential mass assignment vulnerabilities. Mass assignment is harder to identify than data exposure vulnerabilities.

Note that identifying vulnerabilities still requires manual review of the generated OpenAPI spec.

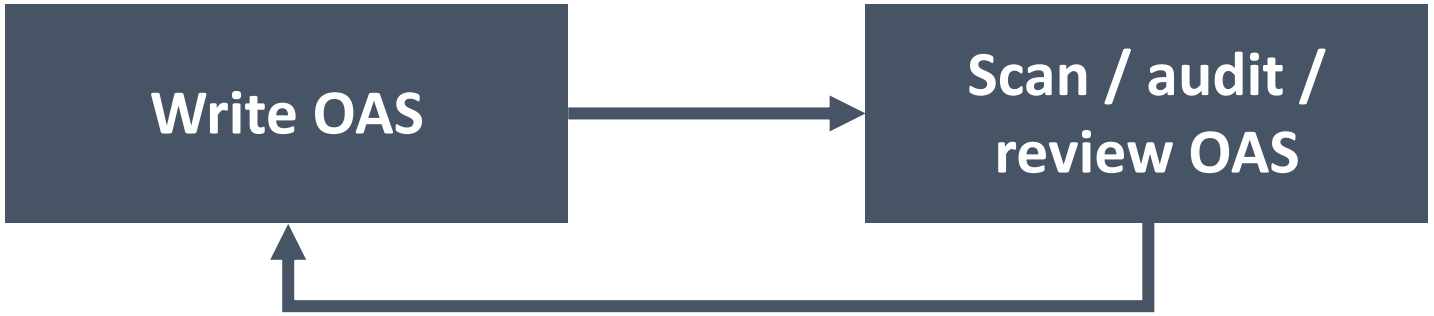
What if ...

What if … we flip the script?

The code-first approach to using OpenAPI specifications

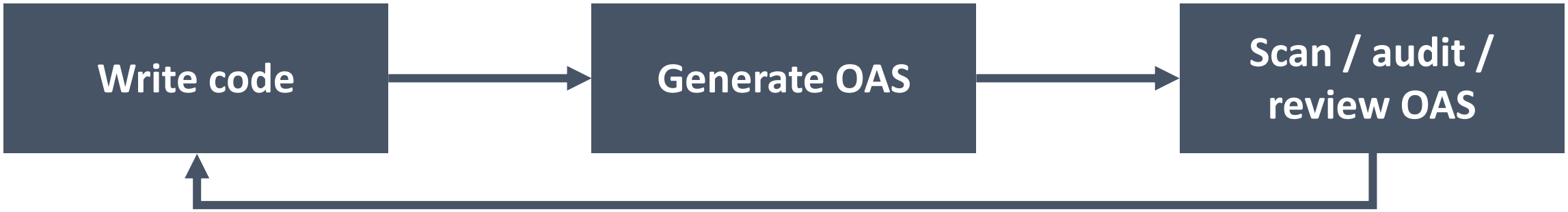


The API-first approach to using OpenAPI specifications

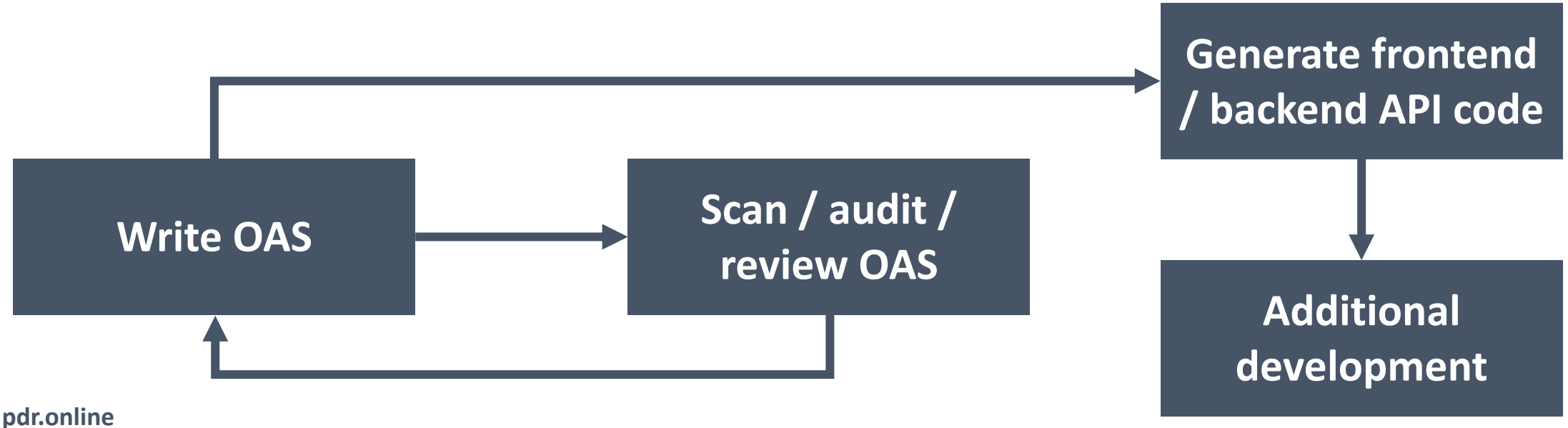


Writing the OAS for the *Update Profile* endpoint

The code-first approach to using OpenAPI specifications



The API-first approach to using OpenAPI specifications



The OAS is a great tool to drive both frontend and API development (code generation, testing, ...)

Having a strict and well-defined OAS combined with code generation is beneficial for the security of the API

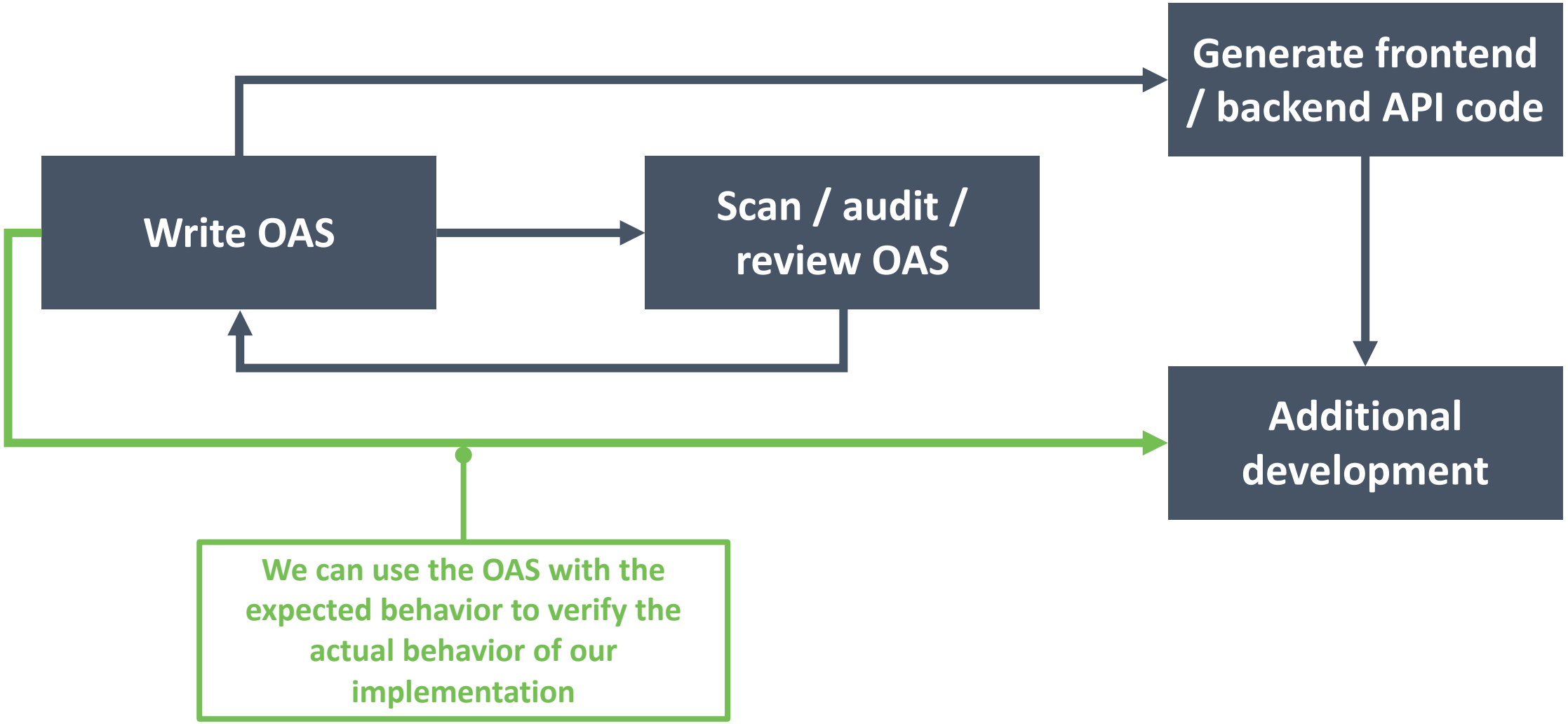
API-FIRST APPROACHES INSPIRE GREATER CARE

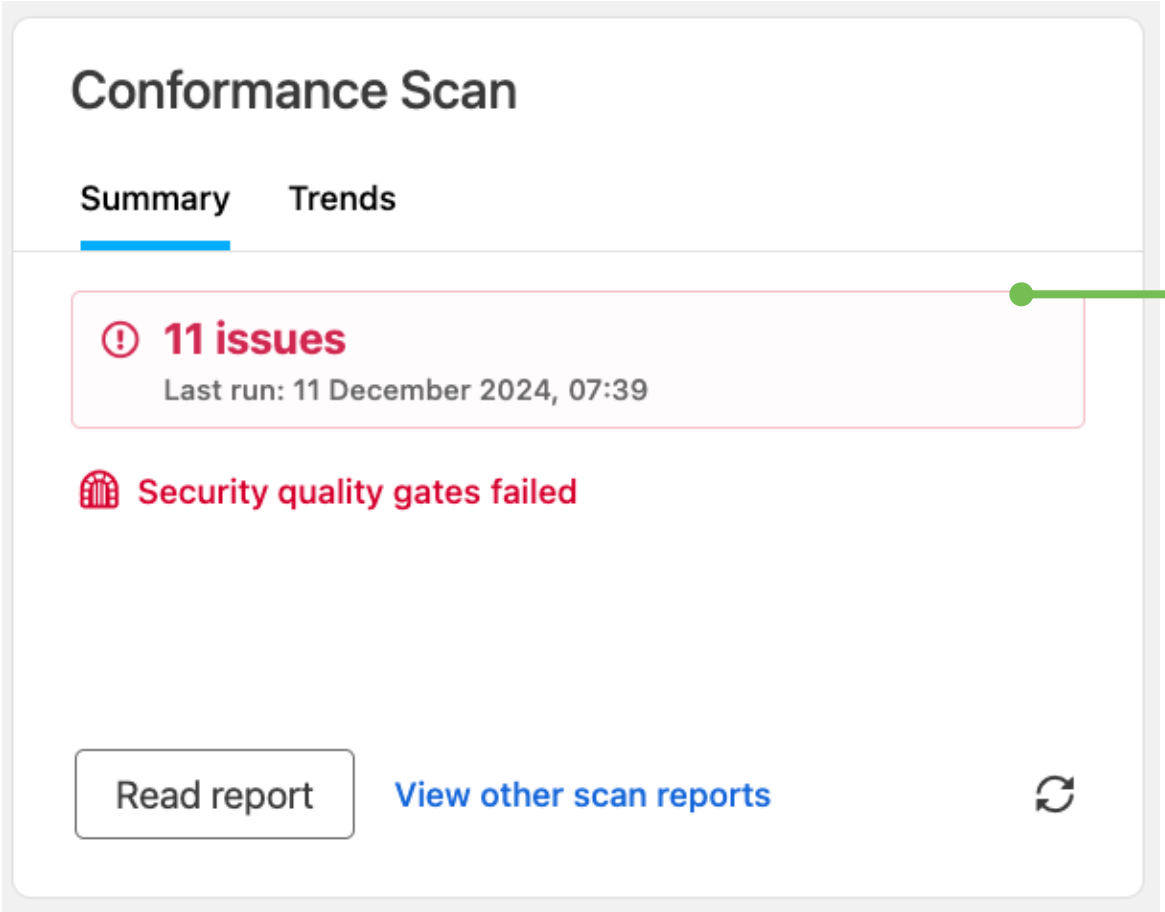
Writing out an OpenAPI spec by hand requires careful attention to the design of the API and the data handled in requests and responses.

Introducing broken object-level property authorization *vulnerabilities becomes a lot harder.*

The OAS can be defined, reviewed, improved, and then used to generate large chunks of code for both the API and the frontend.

But what if the code diverges from the spec?

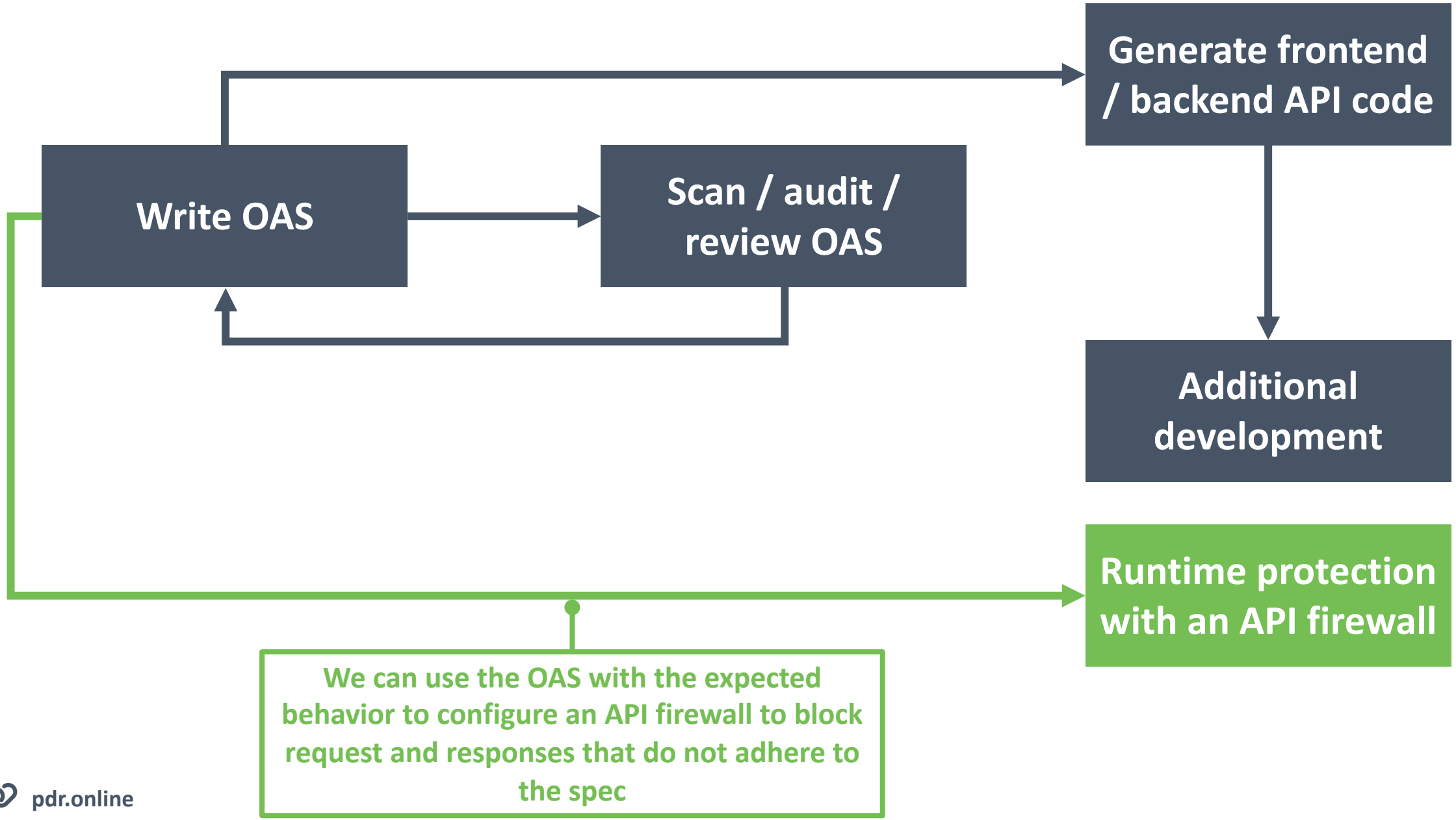

Generate frontend / backend API code
Write OAS
Scan / audit / review OAS
Additional development
We can use the OAS with the expected behavior to verify the actual behavior of our implementation



Running automated conformance scans in the CI/CD pipeline will help you detect when the implementation deviates from the specification

Automated conformance scans can also be extended with other techniques, such as fuzzing etc.

The 42Crunch API security platform is just an example. I am not affiliated with 42Crunch, nor do I benefit from showing this tool in any way.

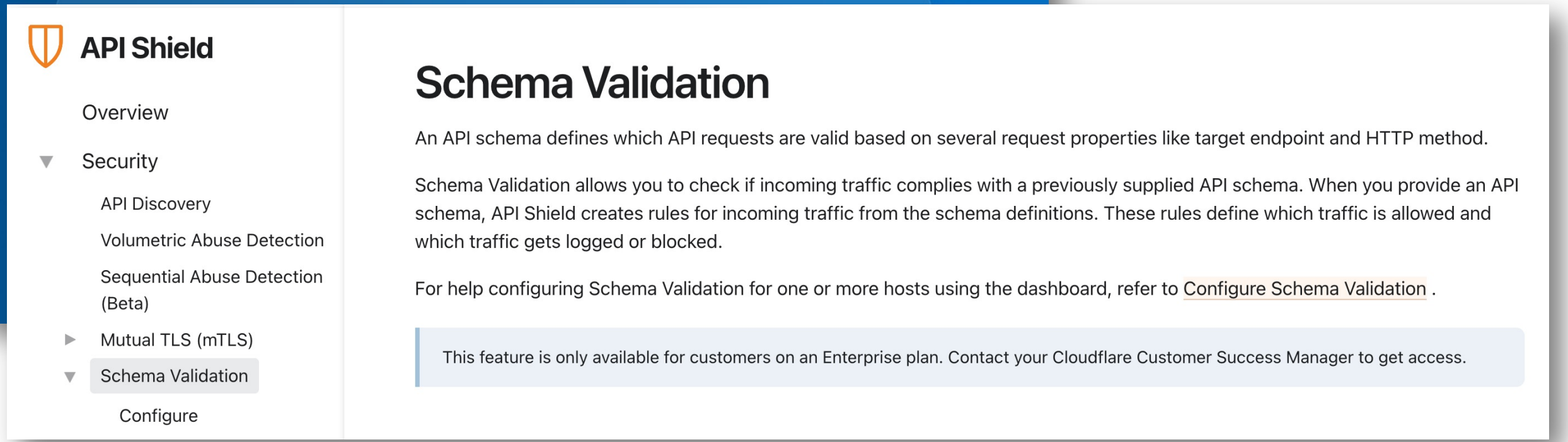

Generate frontend / backend API code
Write OAS
Scan / audit / review OAS
Additional development
Runtime protection with an API firewall
We can use the OAS with the expected behavior to configure an API firewall to block request and responses that do not adhere to the spec

Azure API Management

Unify, govern, and secure your APIs across cloud and on-premises environments with an Azure-native API management solution.

These tools offer request validation against OpenAPI specs to avoid unwanted or malformed requests.

This effectively protects against mass assignment attacks.



https://azure.microsoft.com/en-us/products/api-management
https://developers.cloudflare.com/api-shield/security/schema-validation/

Protection is automatically applied at deployment time

Finally, the API contract is used to **protect APIs using our micro API firewall**. The runtime is fully optimized to be deployed and run on any container orchestrator such as Docker, Kubernetes or Amazon ECS. It can protect North-South and East-West microservices traffic. With minimal latency and footprint, it can be deployed against hundreds of API endpoints with minimal impact.

- API Firewall is configured in one-click from API contract
- Contract becomes the allowlist for security
- No need to guess via AI which traffic is valid
- No policies to write

This tool offers request and response validation, effectively enforcing compliance of the impleemntation to the OAS.

This effectively protects against mass assignment and data exposure attacks.

https://42crunch.com/api-security-platform/

This space is the cutting edge of security, so you will find plenty of alternative tools as well!

API-FIRST APPROACHES RESULT IN BETTER SECURITY

API-first approaches help ensure that the implementation adheres to the specification.

Additionally, the OAS can be used to enable runtime protection tools to stop attacks that aim to abuse data exposure or mass assignment vulnerabilities.

At some point, someone has to think!

Automated tools are awesome, but they always require someone to carefully write or vet a contract.

Tools cannot (yet) fully automatically identify data exposure or mass assignment problems.

OpenAPI specifications are the future of API development and API security

Tool Types

We've organised everything into categories so you can jump to the section you're interested in.

- **Auto Generators:** Tools that will take your code and turn it into an OpenAPI Specification document
- **Converters:** Various tools to convert to and from OpenAPI and other API description formats.
- **Data Validators:** Check to see if API requests and responses are lining up with the API description.
- **Description Validators:** Check your API description to see if it is valid OpenAPI.
- **Documentation:** Render API Description as HTML (or maybe a PDF) so slightly less technical people can figure out how to work with the API.
- **DSL:** Writing YAML by hand is no fun, and maybe you don't want a GUI, so use a Domain Specific Language to write OpenAPI in your language of choice.
- **Gateways:** API Gateways and related tools that have integrated support for OpenAPI.
- **GUI Editors:** Visual editors help you design APIs without needing to memorize the entire OpenAPI specification.
- **Learning:** Whether you're trying to get documentation for a third party API based on traffic, or are trying to switch to design-first at an organization with no OpenAPI at all, learning can help you move your API spec forward and keep it up to date.
- **Miscellaneous:** Anything else that does stuff with OpenAPI but hasn't quite got enough to warrant its own category.
- **Mock Servers:** Fake servers that take description document as input, then route incoming HTTP requests to example responses or dynamically generates examples.
- **Monitoring:** Monitoring tools let you know what is going on in your API.
- **Parsers:** Loads and read OpenAPI descriptions, so you can work with them programmatically.
- **SDK Generators:** Generate code to give to consumers, to help them avoid interacting at a HTTP level.
- **Security:** By poking around your OpenAPI description, some tools can look out for attack vectors you might not have noticed.
- **Server Implementations:** Easily create and implement resources and routes for your APIs.
- **Testing:** Quickly execute API requests and validate responses on the fly through command line or GUI interfaces.
- **Text Editors:** Text editors give you visual feedback whilst you write OpenAPI, so you can see what docs might look like.

Do not expose your OpenAPI specs for your APIs (unless they serve as documentation)

Key takeaways

1. Start using OpenAPI specifications to familiarize yourself
2. Adopt security tooling based on OpenAPI specs and improve
3. Adopt an API-first approach, which yields the best security results

Thank you!

Connect with me to stay in touch about security

https://pdr.online