



OpenAPI
the common language of APIs

ISABELLE MAUNY

Field CTO - 42Crunch / BGB Chair OpenAPI Initiative

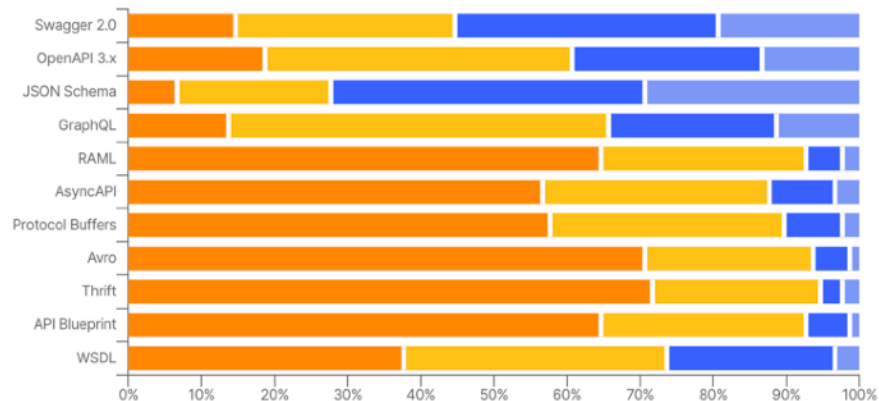
- A **open** standard for REST APIs
- Used by majority of developers
- OpenAPI 2.0 == Swagger 2.0

API Technologies

Specifications



We also asked folks which API specifications they use and love. JSON Schema was by far the most popular choice, used by 72% of respondents. The next most popular were Swagger 2.0 (55%) and OpenAPI 3.x (39%).



Supporting Members



Security Schemes

Parameters (format + constraints)

Data formats (i.e. JSON Schemas)

SLA

Security (JWT / Signatures / Encryption)

Workflows

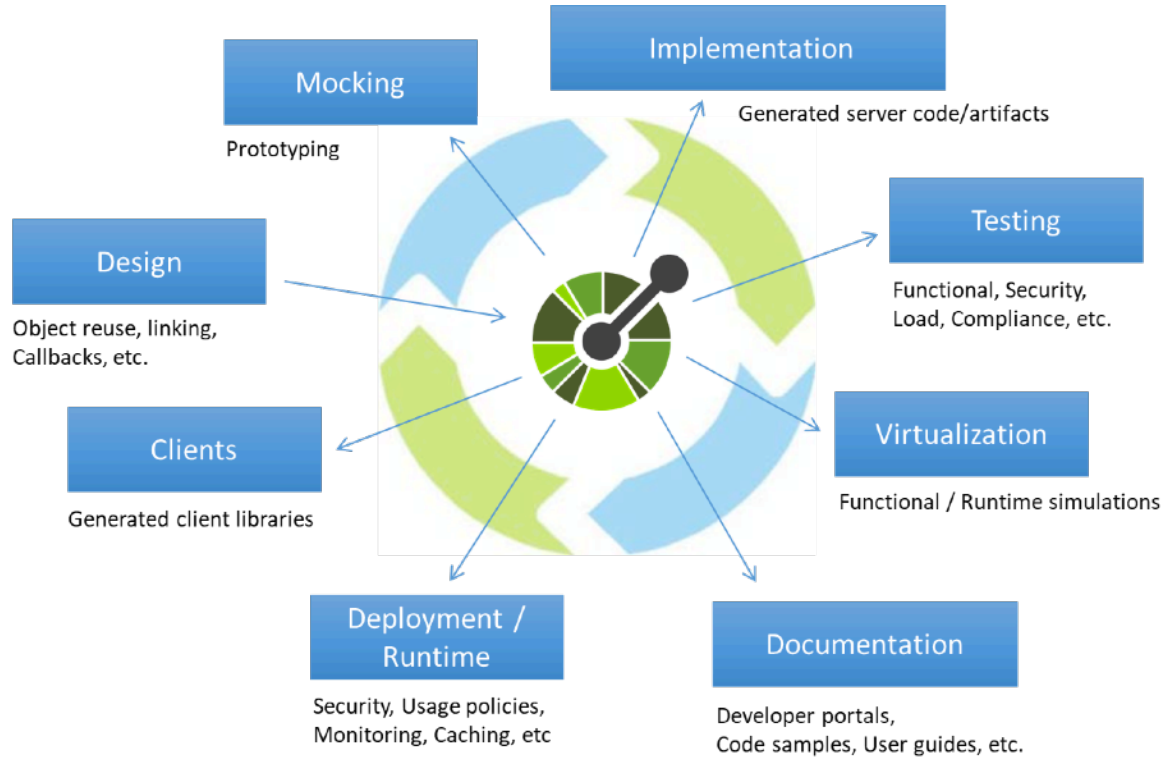




01001101001010
01011010110101
10101010101010
00101010110000
01001101001010
01011010110101
10101010101010
00101010110000

DEMONSTRATION OPEN WEATHER

Much more than documentation!



<https://swagger.io/blog/api-strategy/benefits-of-openapi-api-development/>

Categories

Select a category below to browse the available tooling

(You might find a tool is in more than one category...

and some categories overlap, but it'll be refined over time as we tweak it)

Parsers (430)

**Server Implementations
(410)**

Server (119)

SDK (117)

Testing (115)

Code Generators (84)

Documentation (81)

Data Validators (76)

Low-level Tooling (66)

Description Validators (64)

Unclassified (47)

Converters (41)

Mock (28)

GUI Editors (14)

Text Editors (11)

Security (9)

Learning (8)

User Interfaces (8)

OpenAPI Tooling

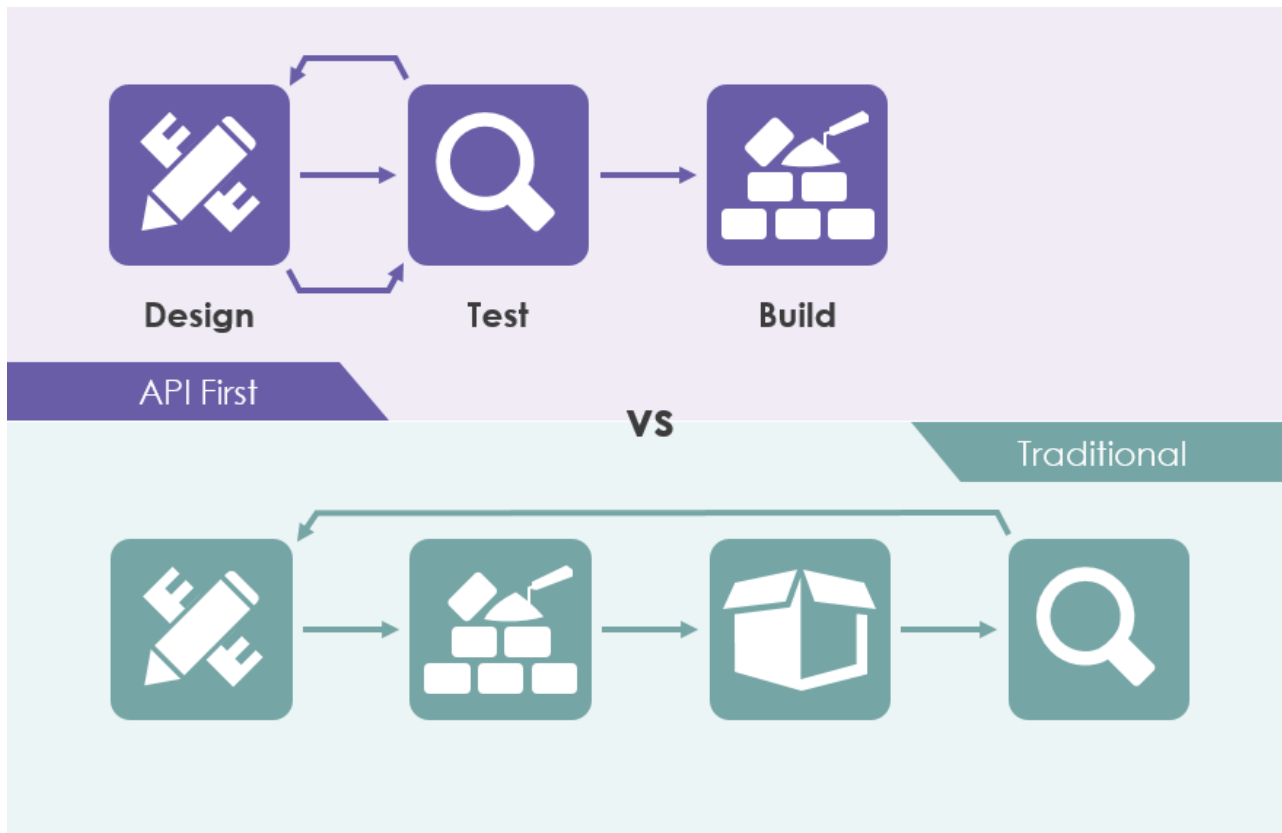
Registry of OpenAPI resources



Using OpenAPI for Design



API First vs Design First



From: <https://www.visual-paradigm.com/guide/development/code-first-vs-design-first/>

Few examples

- Writing OpenAI Plugins:
 - <https://github.com/openai/plugins-quickstart>
- OpenAPI for OpenAI 😊
 - <https://github.com/openai/openai-openapi/blob/master/openapi.yaml>
- Open Weather
- SpringBoot API



01001101001010
01011010110101
10101010101010
00101010110000
01001101001010
01011010110101
10101010101010
00101010110000

DEMONSTRATION

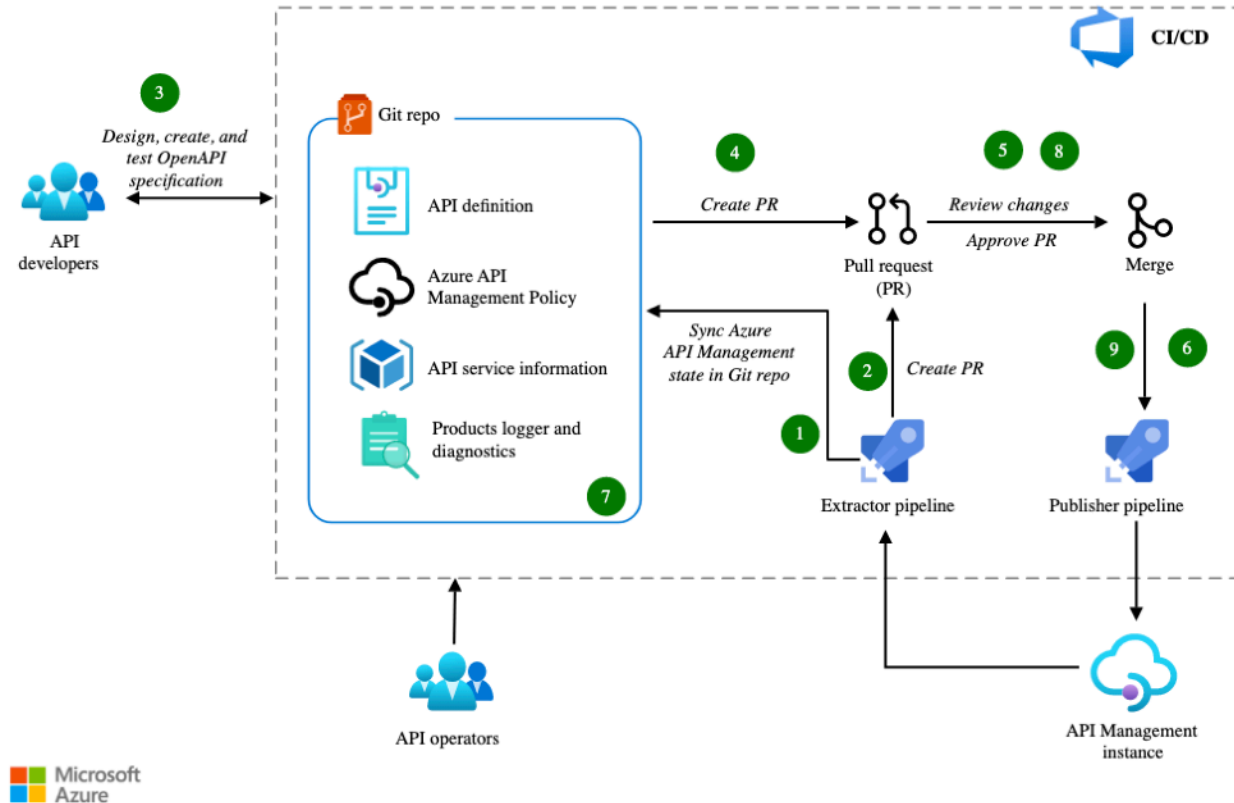
CODE/OPENAPI GENERATION



Automated Deployment and Configuration

Example: API Ops

- Most API Management platforms (if not all) can be automatically configured from an OpenAPI file.
- Avoids manual intervention (and crucial errors...)





01001101001010
01011010110101
10101010101010
00101010110000
01001101001010
01011010110101
10101010101010
00101010110000

DEMONSTRATION

AZURE API MANAGEMENT

OpenAPI is an extensible language

```
"securityDefinitions": {  
  "request_authorizer_header_query" : {  
    "type" : "apiKey",  
    "name" : "Unused",  
    "in" : "header",  
  },  
  "x-amazon-apigateway-authtype" : "custom",  
  "x-amazon-apigateway-authorizer" : {  
    "type" : "request",  
    "identitySource": "method.request.header.HeaderAuth1,method.request.querystring.QueryString1",  
    "authorizerCredentials": "arn:aws:iam::123456789012:role/AWSepIntegTest-CS-LambdaRole",  
    "authorizerUri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/XXX",  
  },  
}
```

Another example: <https://developer.apiture.com/docs/reference/>



01001101001010
01011010110101
10101010101010
00101010110000
01001101001010
01011010110101
10101010101010
00101010110000

DEMONSTRATION

Spectral



Leveraging OpenAPI for API Threat Protection

STARBUCKS (JUNE 2020)

[HTTPS://SAMCURRY.NET/HACKING-STARBUCKS/](https://samcurry.net/hacking-starbucks/)

- ▶ Hacker pokes around to find problems
 - ✓ Tests many (invalid) paths
- ▶ Finds valid calls from Starbucks website
 - ✓ Tries to get to the root of the API to navigate down
- ▶ Finds a path which tricks the Web Application Firewall

```
GET /bff/proxy/stream/v1/me/streamItems/web\..\..\..\
HTTP/1.1
Host: app.starbucks.com
```

- ▶ From there, he starts making calls to find the root of the data graph...



Hacking Starbucks
and Accessing Nearly
100 Million Customer
Records

HACKERS USE TRIAL AND ERROR

- ▶ Try all verbs (GET/POST/HEAD/OPTIONS...)
 - ▶ Try resources names (admin, users, profiles, teachers, accounts, search, ...)
 - ▶ Try various Content-Type values (remember [CVE-2017-5638](#) ?)
 - ▶ Inject data ([mass assignment](#))
 - ▶ Use answers to find info and guess further
-
- ▶ Check this enlightening video : <https://www.youtube.com/watch?v=qgmyAxfGV9c> !

Why are those issues still happening?

- Security is **considered too late in the API development lifecycle**
- APIs are developed really fast, **testing and security can't keep up**
- Too much trust in:
 - ➔ Internal APIs
 - ➔ Internal Apps
 - ➔ Internal Users



How can OpenAPI help?

API LOCKDOWN

- ▶ Describe every piece of data flowing through
 - ✓ Headers
 - ✓ Path params
 - ✓ Query params
 - ✓ Requests
 - ✓ Responses
- ▶ Want to test whether your API is hardened ?
 - ✓ Download one of 42Crunch IDE plugins
 - ✓ One e-mail and you're done.



DATA CONSTRAINTS


- ▶ Data is poorly constrained
 - ✓ Unbounded array sizes
 - ✓ Undefined strings
 - ✓ Unbounded numbers
- ▶ Why this matters ?
 - ✓ Overflow protection (**API4**)
 - ✓ Mass Assignment (**API6**)
 - ✓ Injection protection (**API8**)
- ▶ Base for Input/Output Validation !

```
“_id”: {  
  “type”: “number”,  
  “example”: 1  
},  
“pic”: {  
  “type”: “string”,  
  “example”: 1  
},  
“email”: {  
  “type”: “string”,  
  “example”: “email@email.com”  
},  
“password”: {  
  “type”: “string”,  
  “example”: “p@ssword1”  
},  
“name”: {  
  “type”: “string”,  
  “example”: “Johnny Appleseed”  
},
```




YOU KNOW THE DATA!

```
parameters:  
  - name: uuid  
    in: header  
    description: >-  
      A 128 bit universally unique identifier (UUID) that you generate  
      every request and is used for tracking. It is recommended to use  
      output from Java UUID class or an equivalent  
    type: string  
    required: true
```



```
"properties" : {  
  "accountGroup" : {  
    "type" : "string",  
    "example" : "CHECKING",  
    "description" : "Account group is a classification of accounts. Values include:  
    CHECKING and SAVINGS" ←  
  },  
}
```





SPECIFY THOSE AS CONSTRAINTS!

```
"name" : "uuid",  
"in" : "header",  
"description" : "128 bit random UUID generated uniquely for every request",  
"required" : true,  
"type" : "string",  
"maxLength" : 36,  
"minLength" : 36,  
"pattern" : "[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}"
```

```
"properties" : {  
  "accountGroup" : {  
    "type" : "string",  
    "example" : "CHECKING",  
    "enum" : [ "CHECKING", "SAVINGS" ]  
  },  
}
```



DESPITE ALL YOUR EFFORTS, APIS WILL FAIL !! (OH, AND THEY DO RETURN DATA TOO...)

```
"paths": {
  "/v1/creditCards/notifications": {
    "post": {
      "consumes": [
        "application/json"
      ],
      "produces": [
        "application/json"
      ],
      "parameters": [...],
      "responses": {
        "200": {
          "description": "Successful operation."
        }
      }
    }
  }
}
```



OTHER COMMON API ISSUES



Medium

⬇️ 0.2

415 response should be defined for operations receiving a body (POST, PUT, PATCH)



Medium

⬇️ 0.2

429 response should be defined for all operations



Medium

⬇️ 0.2

No default response defined for the operation



Medium

⬇️ 0.1

If operation has security defined, the 401 response should be defined



RESPONSES MATTER !

- ▶ Which responses are you going to return ?
 - ✓ **API3** : Data leakage and exception leakage
- ▶ Which error codes are valid ?
- ▶ Do you control them ?
 - ✓ Do you have tests that can trigger any of those codes ?
- ▶ Take ownership of your schemas!
 - ✓ Are they strict enough ?
 - ✓ Where do you validate against them ?
 - ✓ Are you **sure** you are doing that systematically ?



01001101001010
01011010110101
10101010101010
00101010110000
01001101001010
01011010110101
10101010101010
00101010110000

DEMONSTRATION

42Crunch

A top-down view of a person's hands typing on a keyboard in a dimly lit room. The scene is overlaid with a semi-transparent blue and purple circular graphic. The background is filled with a pattern of binary code (0s and 1s) in a light blue color. The person's hands are visible, wearing a watch and a ring. A laptop is on the left, and a mouse is on the right. The overall atmosphere is tech-oriented and focused on digital work.

Time to test!



OPENAPI IS GREAT FOR SECURITY TESTING!

- ▶ If you want to avoid being hacked, you need to push your API to the edge!
- ▶ Leveraging OpenAPI: gets the basic testing done **automatically**
 - ✓ Data Fuzzing
 - ✓ IDOR (BOLA) Testing
 - ✓ Authentication testing
 - ✓ Authorization testing
 - ✓ Navigation testing
- ▶ <https://owasp.org/www-project-web-security-testing-guide/v42/>

MORE AND MORE API SECURITY SOLUTIONS CONSUME OPENAPI SPECS

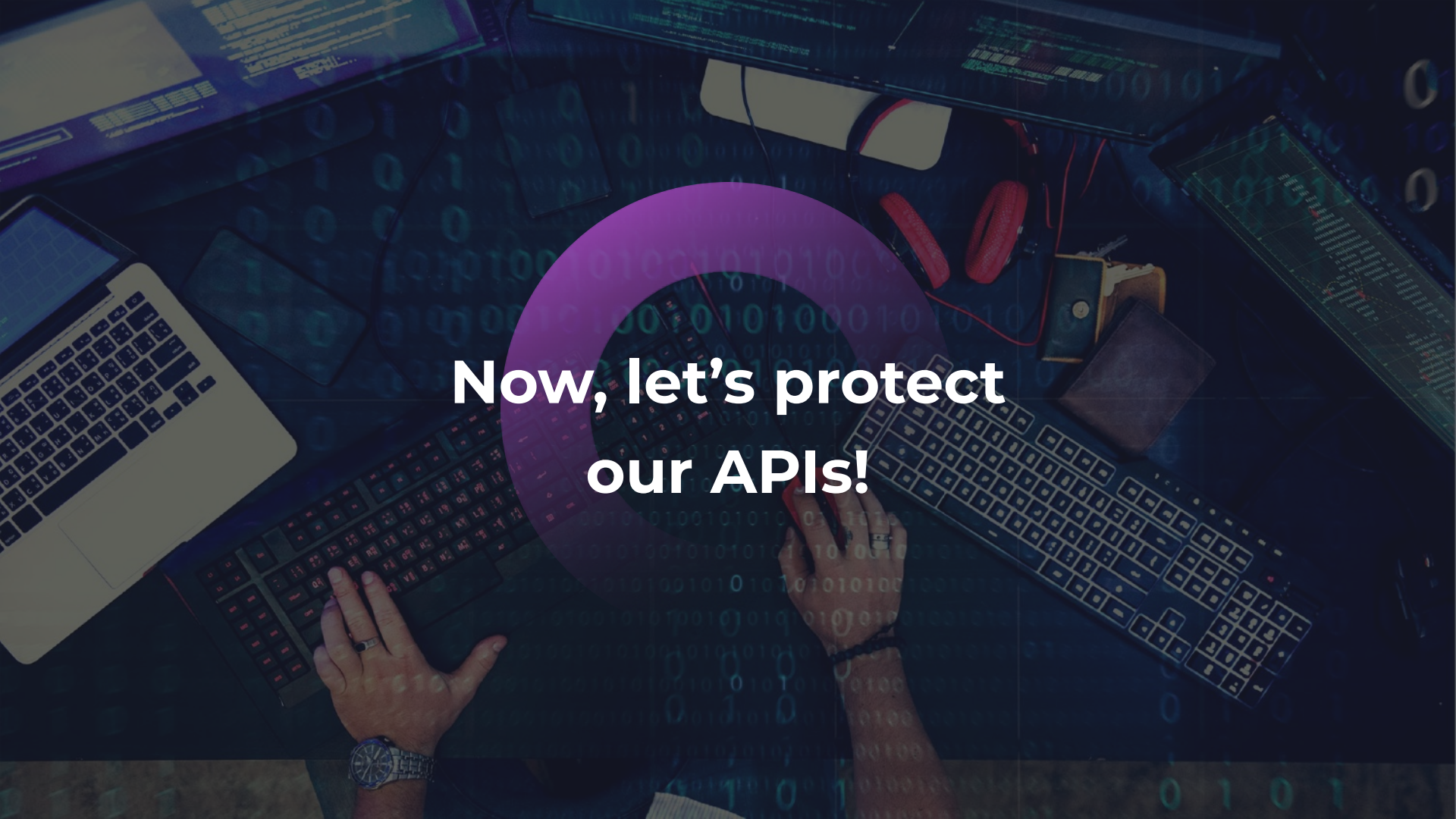
- ▶ OWASP ZAP (OWASP OpenSource Fuzzer)
 - ✓ <https://www.zaproxy.org/docs/desktop/addons/openapi-support/>
- ▶ Burp OpenAPI module
 - ✓ Check this is a guide : <https://blog.secureideas.com/2020/04/getting-started-api-penetration-testing-with-insomnia.html>
- ▶ Schemathesis
 - ✓ <https://schemathesis.readthedocs.io/en/stable/introduction.html>
- ▶ 42Crunch
 - ✓ <https://42crunch.com/api-security-platform>
- ▶ And of course traditional security testing tools like DAST take (some) advantage of OpenAPI information.



01001101001010
01011010110101
10101010101010
00101010110000
01001101001010
01011010110101
10101010101010
00101010110000

DEMONSTRATION

Schemathesis



**Now, let's protect
our APIs!**



NEGATIVE SECURITY MODEL



Access **Allowed**
by default



Block access for
suspicious traffic



Threats centric



POSITIVE SECURITY MODEL



Access **Denied** by default



Allow Access only to **approved traffic**



Trust centric



WHY A POSITIVE MODEL ?

- ▶ Much stricter access control
- ▶ Limited false positives

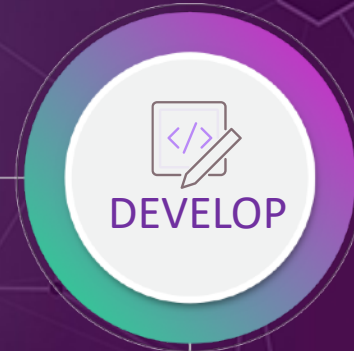
- ▶ You're protected even if new rules have not been created to detect the new threats.



OPENAPI ENABLES A POSITIVE MODEL

- ▶ All interactions are defined (operations/verbs)
 - ▶ Headers
 - ▶ Parameters
 - ▶ Data that will flow inbound/outbound is defined
-
- ▶ Leveraged by multiple security solutions including Akamai, F5, CloudFlare or 42Crunch

ENABLING A SECURITY VIRTUOUS LOOP



Thank
you!

Code and Slides available at: <https://github.com/isamauny/secappdev2023>

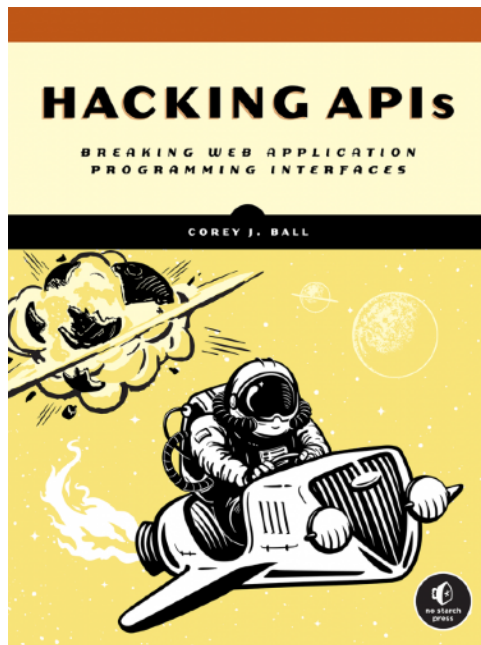
Learning more

APISecurity.io



<https://apisecurity.io/>

“Hacking APIs” - Corey Ball



<https://nostarch.com/hacking-apis>

Learning Application Security



[Buy the book](#)