

# Security Testing of Mass Assignment Vulnerabilities in RESTful APIs

**Mariano Ceccato**

mariano.ceccato@univr.it

**Davide Corradini, Michele Pasqua**



UNIVERSITÀ  
di **VERONA**

Dipartimento  
di **INFORMATICA**

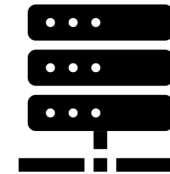
# What is a Web API or REST API?



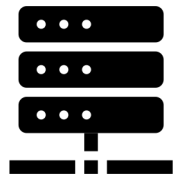
Create  
Read  
Update  
Delete

# Is Web APIs implementation correct?

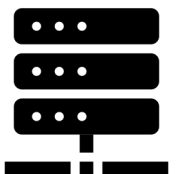
- **Technology:** HTTP network messages to test them
- **Interaction:** Need to mock other external services
- Very **abstract:** Not guided by a Graphical User Interface
- What **scenarios** to test: Intended designed Vs unintended erroneous
- What **data** to use?



facebook



Pay



# Presentation Topic

- **Security Testing of Mass Assignment Vulnerabilities in RESTful APIs**
- Problems:
  - Different programming languages/frameworks
  - No source code access
  - Dynamically deployed/undeployed component (microservices)
- Formal service definition as Open API Specification (OAS)

Our solution:

Black-box testing of Web APIs based on their interface





# Technical background



UNIVERSITÀ  
di **VERONA**

Dipartimento  
di **INFORMATICA**

# REST APIs

- API: **A**pplication **P**rogramming **I**nterface
  - Interface that offers services to other pieces of software
- REST: **RE**presentational **S**tate **T**ransfer
  - Architectural style for distributed hypermedia systems
  - An API must adhere to this architectural style in order to be considered a REST API



# REST API principles

1. Uniform Interface
  - Identification of resources
  - Manipulation of resources through representations
  - Self-descriptive messages
2. Client-server
3. Stateless
  - Requests must provide all the necessary information for processing the corresponding responses
4. Cacheable
5. Layered system
  - Allows a layered/hierarchical architecture
6. Code on demand (optional)
  - Allows to download code in the form of applets or scripts



# The HTTP protocol

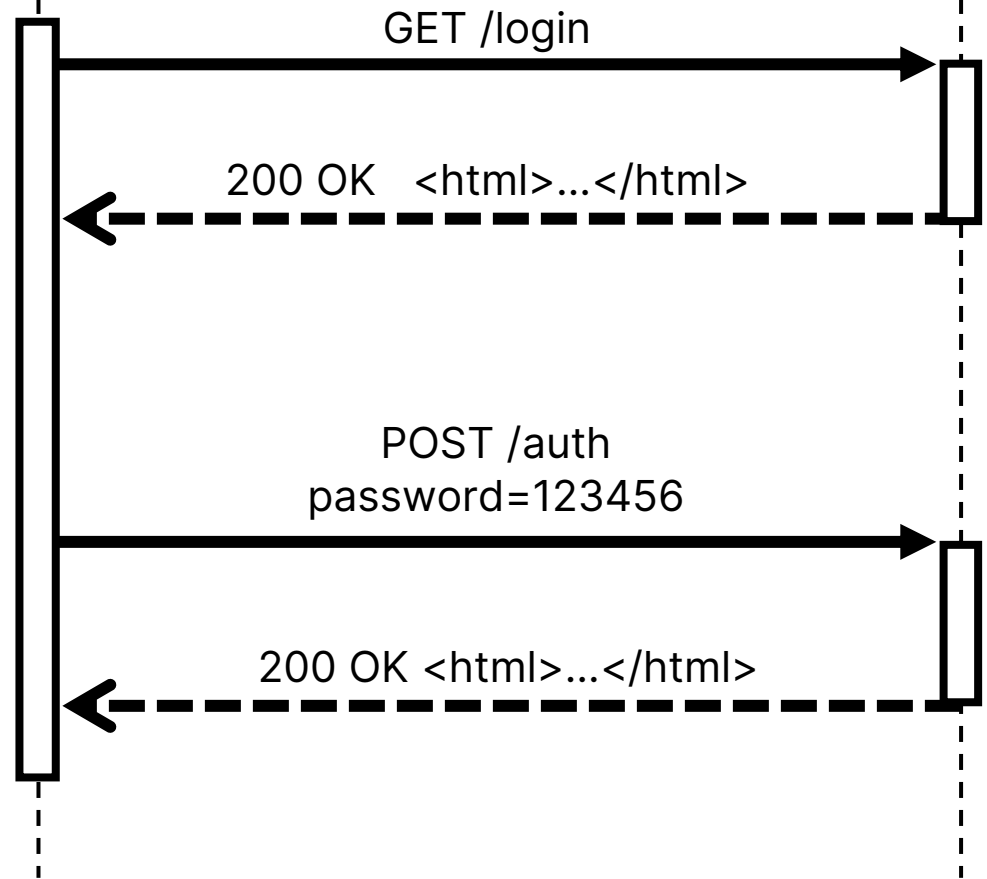
- Hypertext Transfer Protocol
  - Used in the web to transfer web pages, images, and files
- Application layer protocol
- Request-response protocol in the client-server model
- Stateless



# A typical HTTP web interaction

Client (browser)

Web server



# HTTP request

	Method/Verb	Path	Protocol version
	POST	/authentication	HTTP/1.1
Headers	Host: univr.it Accept: application/json Content-Type: application/x-www-form-urlencoded Content-Length: 15		
Body	password=123456		



# HTTP methods

- GET: requests a representation of the specified resource
- POST: submits an entity to the specified resource, often causing a change in state or side effects on the server
- PUT: replaces all current representations of the target resource with the request payload
- DELETE: deletes the specified resource
  
- Other, less used, methods: PATCH, HEAD, TRACE, OPTION, CONNECT



# HTTP response

Protocol version `HTTP/1.1` `200` `OK`

Headers

```
Date: Thu, 11 Nov 2021 17:13:27 GMT
Content-Type: text/html
Content-Length: 16
Connection: keep-alive
```

Body `<html> ... </html>`



# HTTP response status code

1XX Informational	
100	Continue
101	Switching Protocols
102	Processing

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirection	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

# JSON

- JavaScript Object Notation (it is not tied to JavaScript)
- Lightweight data-interchange format
- Supports objects, arrays, strings, numbers, booleans and nulls.
- Example: university employee

```
{  
  "firstName": "Mariano",  
  "lastName": "Ceccato",  
  "occupation": "Professor",  
  "courses": [  
    "Software engineering",  
    "IoT security"  
  ]  
}
```



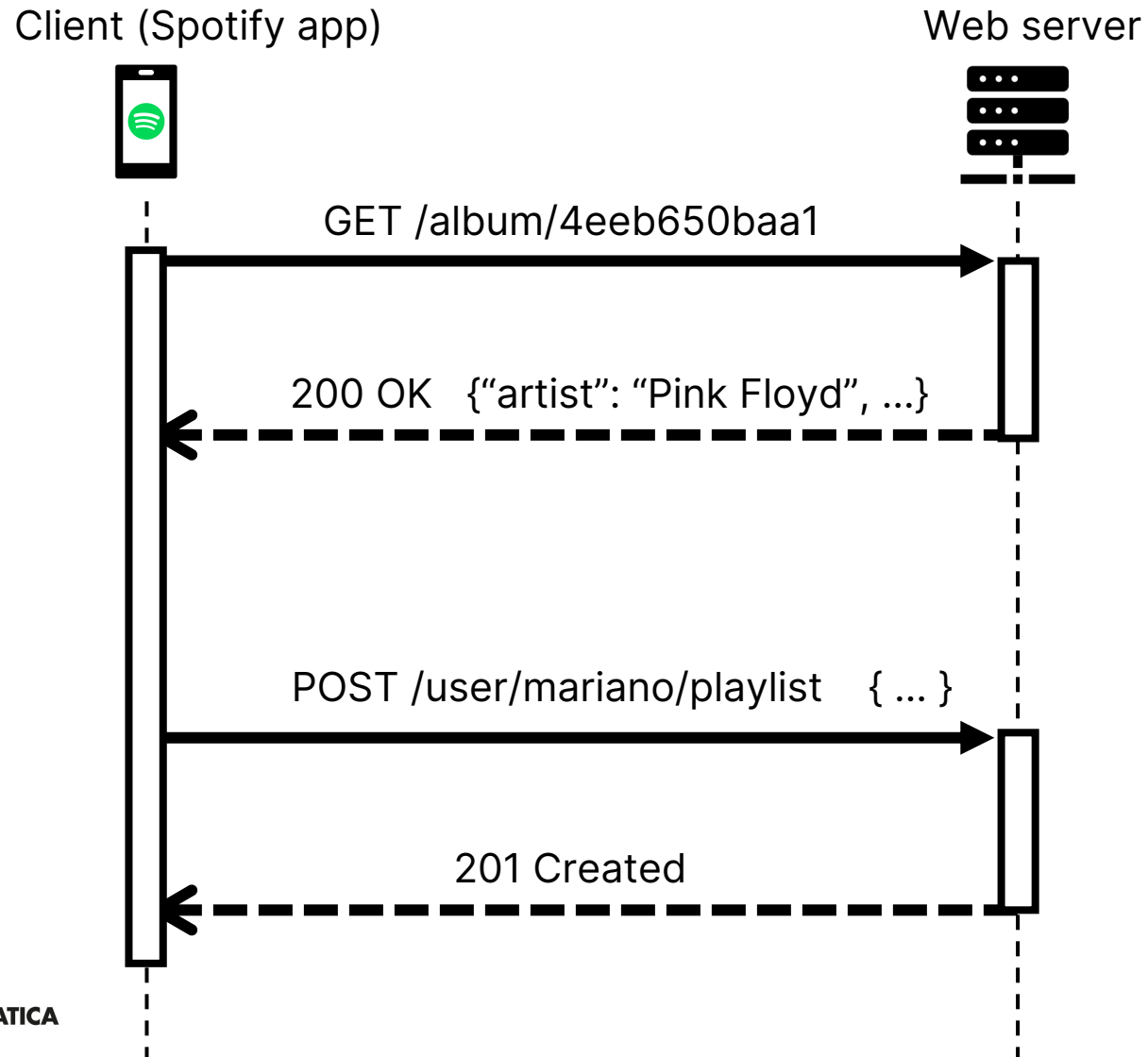
# The Spotify REST API



<https://developer.spotify.com/documentation/web-api/>



# A REST HTTP interaction

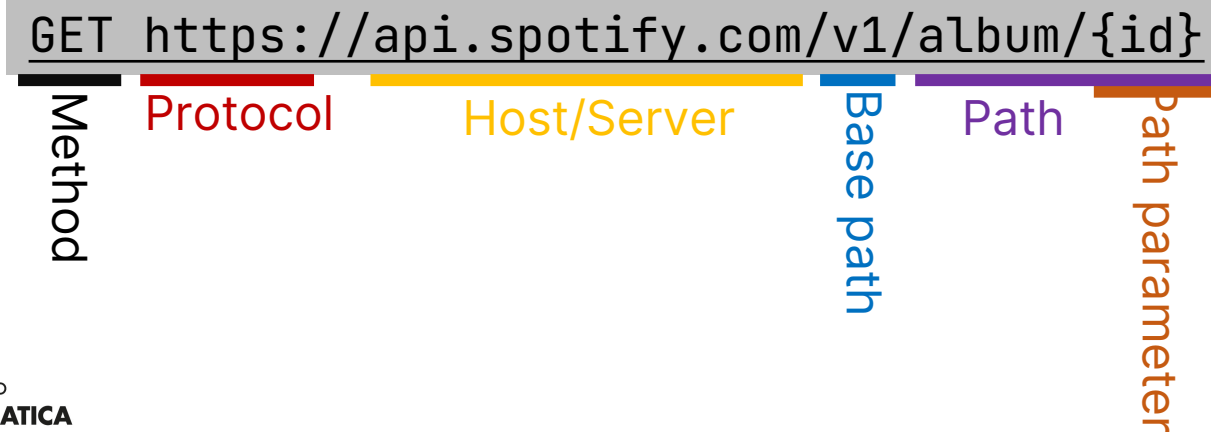


# Some operations of the Spotify REST API

- Search



- Get album info



# The OpenAPI specification

- Formal definition of the REST API
- JSON or YAML format
- It describes:
  - Information about the REST service (servers, maintainers, etc...)
  - Available paths/endpoints and accepted HTTP methods
    - The pair <HTTP method, Path> is known as *Operation*
  - Accepted parameters (types, bounds, example values, etc.)
  - Response formats in multiple scenarios (e.g., successful response, error response)
- Enables OpenAPI based applications, e.g.,:
  - Editors (<https://editor.swagger.io/>)
  - Swagger UI: produces an interactive GUI with the documentation (e.g., <https://petstore.swagger.io/>)
  - Server and client generation
  - RestTestGen: automated test case generation



# The Spotify OpenAPI specification

```
{  
  "openapi": "3.0.1",  
  "servers": [{"url": "https://api.spotify.com/v1"}],  
  "info": {  
    "title": "Spotify Web API",  
    "version": "2021.8.15",  
  },  
  "externalDocs": {  
    "description": "Find more info ...",  
    "url": "https://developer.spotify.com/document..."  
  },  
  ...  
}
```

Server exposing the REST API

Location of the documentation

# The Spotify OpenAPI specification

```
"paths": {  
  "/search": { The path  
    "get": { The HTTP method  
      "description": "Search...",  
      "operationId": "endpoint-search",  
      "parameters": [{  
        "in": "query",  
        "name": "q",  
        "required": true,  
        "schema": {"type": "string"}  
      }],  
      "responses": {  
        "200": {  
          "content": {  
            "application/json": {  
              "schema": {  
                "$ref": "#/components/schemas/SearchResponseObject"  
              }  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

Parameters, including their location, name, schema and if they are mandatory

Format of the response  
\$ref is a reference to a schema definition





# Postman

- Tool for API testing
  - Manual writing of test cases / requests
  - Can automate test execution
- Available as desktop app or web app
  - Web app does not support requests to local networks



# Postman

Method

Input parameters

Response

Send request button

URL

KEY	VALUE	DESCRIPTION
Key	Value	Description

```
1 {
2   {
3     "id": 1,
4     "title": "Software Engineering",
5     "author": "Mariano Ceccato",
6     "price": 10.0
7   }
8 }
```



# RestTestGen



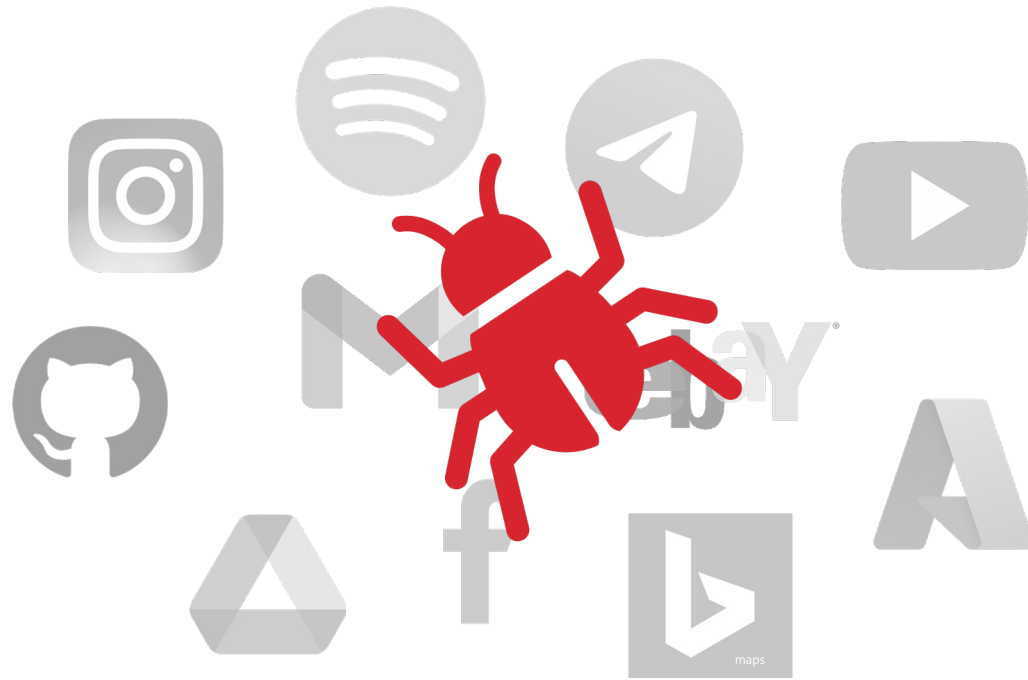
UNIVERSITÀ  
di **VERONA**

Dipartimento  
di **INFORMATICA**

# REST APIs are everywhere

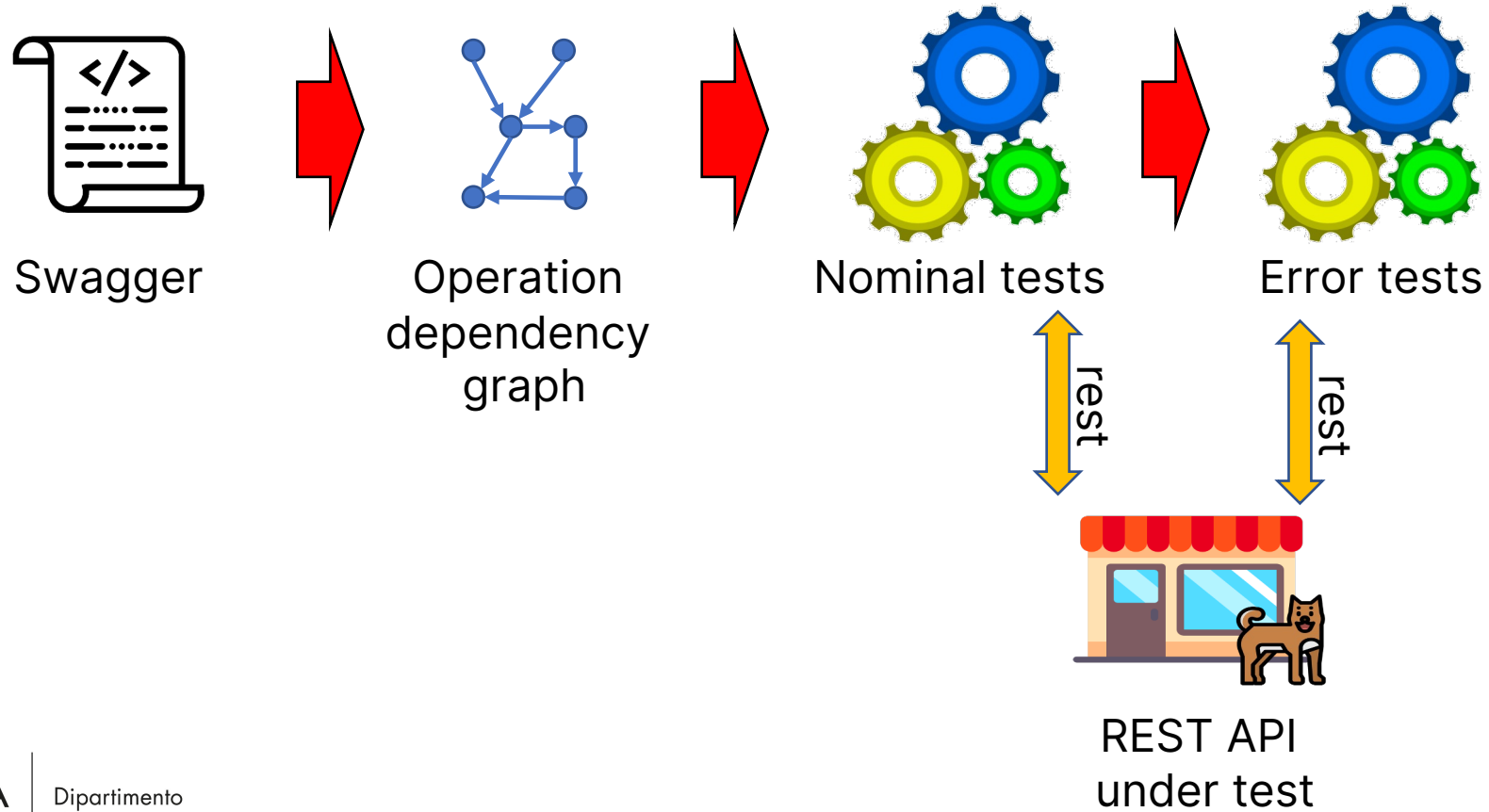


# REST API automated testing



- Several approaches
  - Evolutionary algorithms
  - Model based
  - Ontology
  - Deep learning
- Research tools implemented from scratch
  - Very limited code reuse

# Approach overview



# Operation Dependency

```

/pets:
  get:
    summary: List all pets
    operationId: getPets
    tags:
      - pets
    responses:
      '200':
        description: PetIds
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  petId:
                    type: integer
  
```

output

```

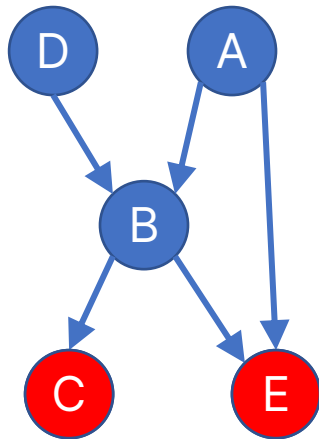
/pets/{petId}:
  get:
    summary: Info for a specific pet
    operationId: getPetById
    tags:
      - pets
    parameters:
      - name: petId
        in: path
        required: true
        schema:
          type: string
  
```

input

Case mismatch  
 petID, petid, petId  
 Id completion  
 /getPet  $\Rightarrow$  Pet  
 pet.id  $\Rightarrow$  petId  
 Stemming  
 pets  $\Rightarrow$  pet



# Operation Testing Order



- Leaf nodes are selected (no outgoing edges)
  - No input
  - Input is not available on operations output
- To maximize the likelihood of a successful test, resources might require to be in a certain status
- Leaf nodes are order based on the CRUD semantics

1. head

2. post

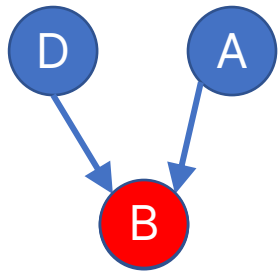
3. get

4. put/patch

5. delete



# Operation Testing Order



- Tested operations are removed from the graph
- New operations become leaf nodes and can now be tested

The order in which operations are tested can not be precomputed, because it depends on what operations we succeed in testing

# Input Value Generation

- Based on response dictionary
  - Map (name→values) of data observed at testing time, while testing previous operations
    - Exact name match `petId` ✓ `petId`
    - Concatenation of object + field `pet.id` ✓ `petId`
    - Name edit distance < threshold `petsId` ✓ `petId`
    - Key is a substring `myPetId` ✓ `petId`
- Based on swagger definition
  - Enum, example, default values
  - Random values (compatible with constraints)



# HTTP Status Code Oracle

- 2xx means correct execution
  - 200: ok
  - 201: successful resource creation
- 4xx means error that is correctly handled
  - 400: bad request
  - 404: not found
- 5xx means error
  - 500: server crash




# Schema Validation Oracle

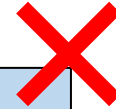
```
responses:  
  '200':  
    description: Expected response to a valid request  
    content:  
      application/json:  
        schema:  
          $ref: "#/components/schemas/Pet"
```

```
components:  
  schemas:  
    Pet:  
      type: object  
      required:  
        - id  
        - name  
      properties:  
        id:  
          type: integer  
          format: int64  
        name:  
          type: string  
        tag:  
          type: string
```

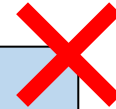
```
{  
  "id": 1,  
  "name": "doggy",  
  "tag": "dog"  
}
```



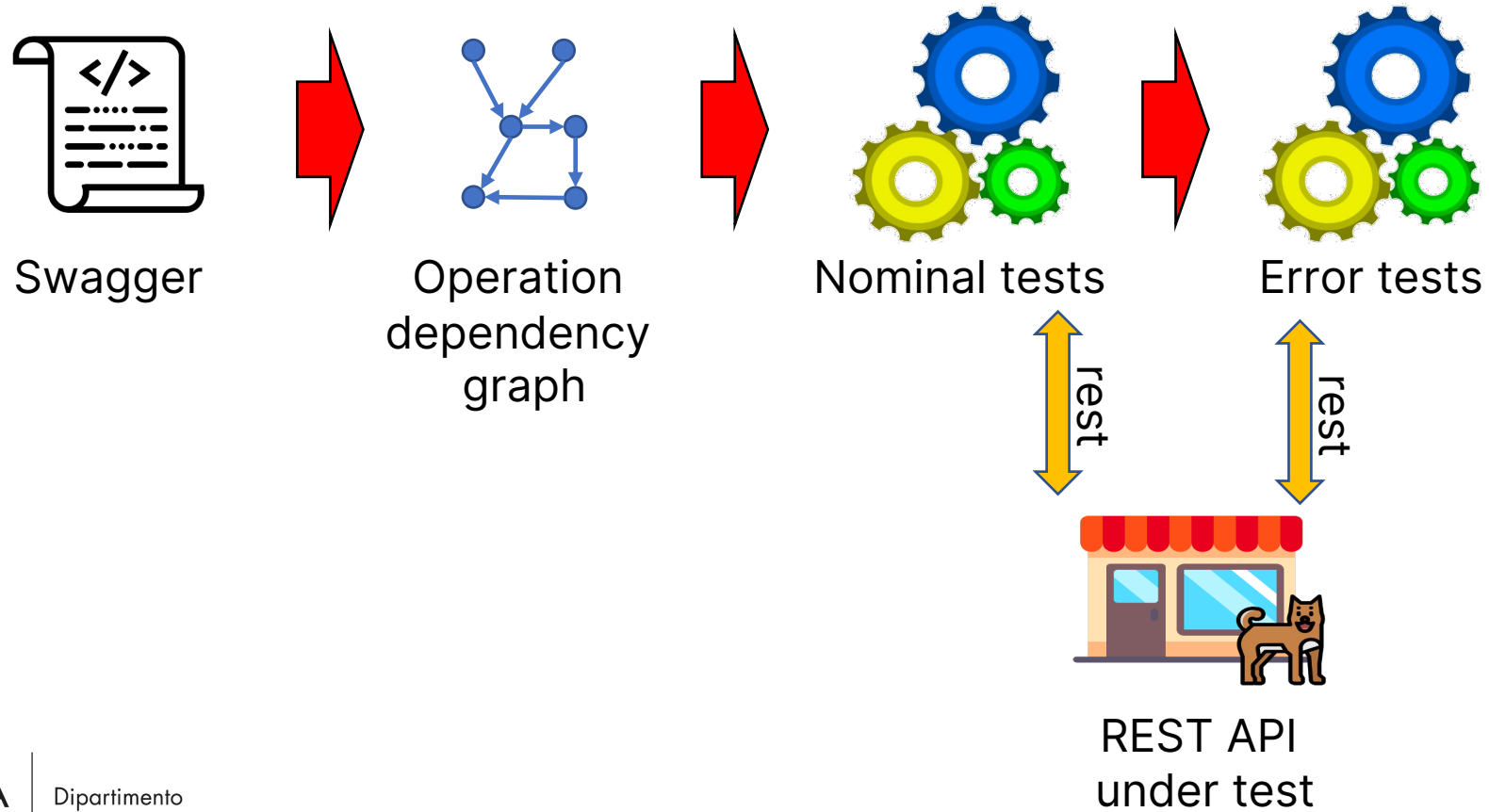
```
{  
  "id": 1,  
  "name": "doggy"  
}
```



```
{  
  "id": 1,  
  "name": "doggy",  
  "tag": 5  
}
```

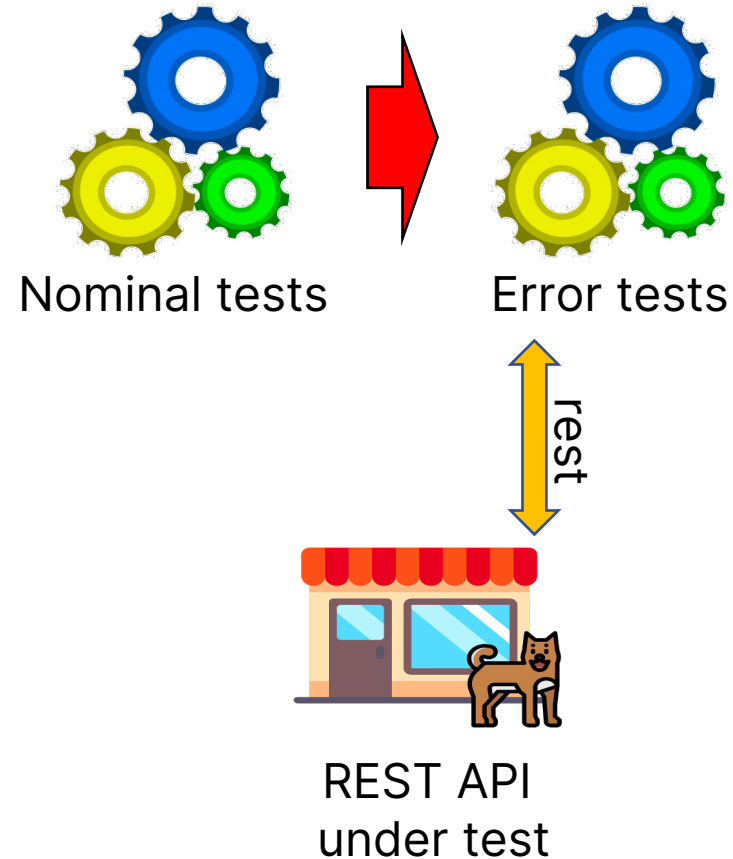


# Approach overview



# Testing of Error Cases

- Analyses how an API behaves when it is given wrong input data
- Mutation operators
  - Remove a `required` input field
  - Change field type
  - Change field value



# HTTP Status Code Oracle

- 2xx means correct execution
  - 200: ok
  - 201: successful resource creation
- 4xx means error that is correctly handled
  - 400: bad request
  - 404: not found
- 5xx means error
  - 500: server crash



# Experimental Validation

## Research questions

- Is the Nominal Tester module effective in generating test cases?
- Is the Error Tester module effective in generating test cases?

## Case studies

- 87 REST APIs listed in the website API.guru (2.6k operations)
  - Filtering out those with authentication or not responding

## Procedure

- Nominal tester for 10 minutes per REST API
- Test cases with 2xx status code are mutated
- Error tester for 10 minutes per REST API
  - $N_{\text{fuzz}}=5$
  - Response dictionary threshold=1





# Results

	APIs	Operations
<b>Total</b>	<b>87</b>	<b>2,612</b>
Status code 2xx	62	625
Status code 5xx	20	151
Validation error	66	1,733

Mutation operator	Mutants	Status code 2xx	Status code 5xx
Missing required	459	283	7
Wrong input type	707	511	16
Constraint violation	119	68	11
<b>Total</b>	<b>1,285</b>	<b>864</b>	<b>23</b>



# Discover RestTestGen

- Reach me during after this presentation
- GitHub → <https://github.com/SeUnivr/RestTestGen>
- Contact me → [mariano.ceccato@univr.it](mailto:mariano.ceccato@univr.it)



# Test coverage



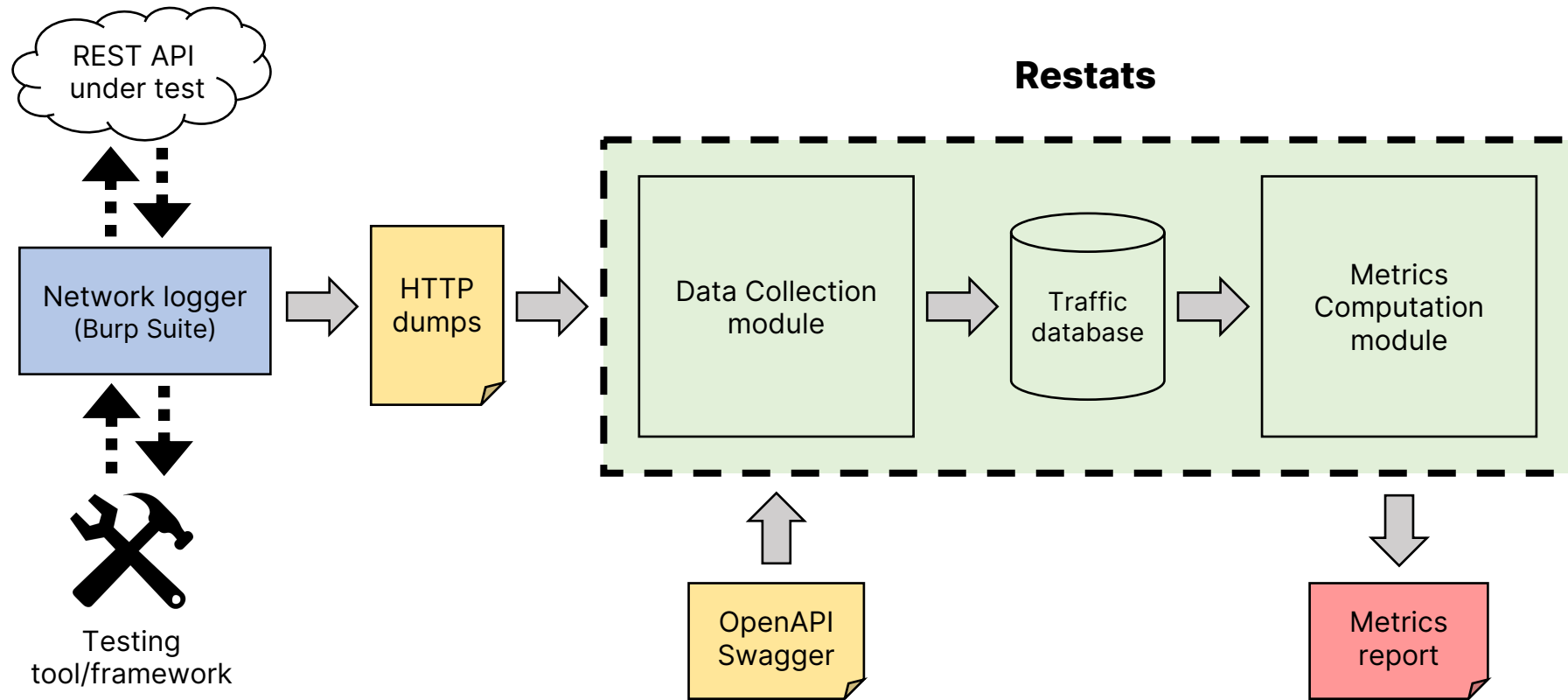
# Problem definition

- REST APIs are developed with different languages, frameworks, and closed-source libraries
  - White-box testing approaches difficult to apply
- Approaches are available to test REST API with a black-box viewpoint

Black-box coverage metrics for  
REST APIs



# Architecture of Restats



# Metrics Computation module

## Input coverage metrics

- Path coverage
- Operation coverage
- Parameter coverage
- Parameter value coverage
- Request content-type coverage

## Output coverage metrics

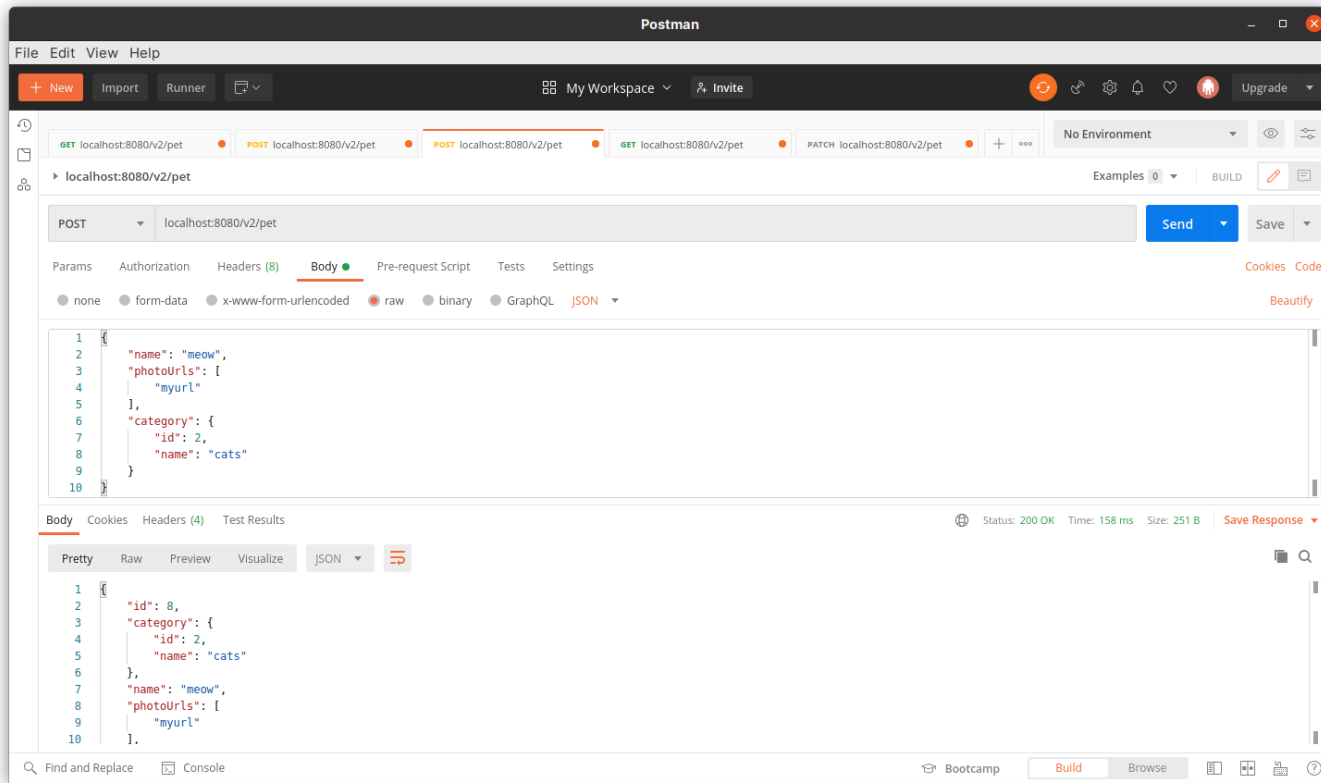
- Status code class coverage
- Status code coverage
- Response content-type coverage

Metrics are computed as defined by Martin-Lopez et al. [12], with adaptations in some cases to make them operative.

[12] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, “Test coverage criteria for RESTful web APIs,” in Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation, 2019, pp. 15–21.



# 1. Execute test cases



- Manually (e.g., browser)
- Using testing tools (e.g., Postman)
- Using advanced tools that automatically generates test cases (e.g., RestTestGen)

# 2. Record the network traffic

The screenshot shows the Burp Suite interface with a list of intercepted HTTP requests. The selected request (number 3) is a POST to /v2/pet with a status of 200 and a JSON response. The response details show a 200 status and a JSON body containing information about a cat named 'meow'.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS
1	http://127.0.0.1:8081	GET	/v2/pet			200	1350	JSON				127.0.0.
2	http://127.0.0.1:8081	POST	/v2/pet		✓	500	5730	JSON				127.0.0.
3	http://127.0.0.1:8081	POST	/v2/pet		✓	200	249	JSON				127.0.0.
4	http://127.0.0.1:8081	PATCH	/v2/pet			405	5066	JSON				127.0.0.

```
Request
1 POST /v2/pet HTTP/1.1
2 Accept: application/json
3 Content-Type: application/json
4 User-Agent: PostmanRuntime/7.26.8
5 Host: localhost:8080
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Content-Length: 129
9
10 {
11   "name": "meow",
12   "photoUrls": [
13     "myurl"
14   ],
15   "category": {
16     "id": 2,
17     "name": "cats"
18   }
19 }
```

```
Response
1 HTTP/1.1 200
2 Content-Type: application/json; charset=UTF-8
3 Date: Tue, 14 Sep 2021 11:37:46 GMT
4 Connection: close
5 Content-Length: 109
6
7 {
8   "id": 8,
9   "category": {
10     "id": 2,
11     "name": "cats"
12   },
13   "name": "meow",
14   "photoUrls": [
15     "myurl"
16   ],
17   "tags": [
18   ],
19   "status": "available"
20 }
```

The screenshot shows a file explorer window with a directory structure. The files are organized into pairs of request and response files, numbered 1 through 5.

- 1-request.txt, 1-response.txt
- 2-request.txt, 2-response.txt
- 3-request.txt, 3-response.txt
- 4-request.txt, 4-response.txt
- 5-request.txt, 5-response.txt





# Restats can help



## Developers

Development  
Testing



## Stakeholders

Evaluation



## Researchers

Evaluation  
Comparison

GitHub repository:

<https://github.com/SeUniVr/restats>



# Object tools

## RestTestGen [5]

- Operation Dependency Graph

## RESTler [6]

- Full enumeration of sequences

## bBOXRT [7]

- Large collection of mutation operators

## REStest [8]

- Inter-parameter dependencies

[5] E. Viglianisi, M. Dallago, and M. Ceccato, "RESTTESTGEN: Automated black-box testing of RESTful APIs," in 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, pp. 142–152

[6] V. Atlidakis, P. Godefroid, and M. Polishchuk, "RESTler: Stateful REST API fuzzing," in Proceedings of the 41st International Conference on Software Engineering, ser. ICSE '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 748–758.

[7] N. Laranjeiro, J. Agnelo, and J. Bernardino, "A black box tool for robustness testing of REST services," IEEE Access, vol. 9, pp. 24 738–24 754, 2021.

[8] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "REStest: Black-box constraint-based testing of RESTful web APIs," in Service-Oriented Computing - 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14-17, 2020, Proceedings, ser. Lecture Notes in Computer Science, E. Kafeza, B. Benatallah, F. Martinelli, H. Hacid, A. Bouguettaya, and H. Motahari, Eds., vol. 12571. Springer, 2020, pp. 459–475.



# REST APIs case studies

- REST APIs whose state can be reset after each test session
- REST APIs that comes with an OpenAPI specification
- REST APIs that are representative of real-world REST APIs




Case Study	Language	Framework	Endpoints	Operations	# of lines
01-Slim	PHP	Slim	9	18	8,566
02-Airline	Java	Spring Boot	12	30	3,859
03-Streaming	Java	Spring Boot	5	5	1,780
04-Petclinic	Java	Spring Boot	17	47	8,550
05-Toggle	ASP.NET	.NET Core	8	16	2,363
06-Problems	Java	Spring Boot	5	9	2,174
07-Products	Java	Spring Boot	6	14	3,451
08-Widgets	Go	-	4	14	1,370
09-Safrs	Python	Flask	6	18	2,787
10-Realworld	PHP	Laravel	11	19	5,278
11-Crud	Node.js	Express	1	4	5,106
12-Order	PHP	Laravel	2	3	3,359
13-Users	TypeScript	Express	2	5	805
14-Scheduler	Node.js	Express	26	40	24,044

# Research questions

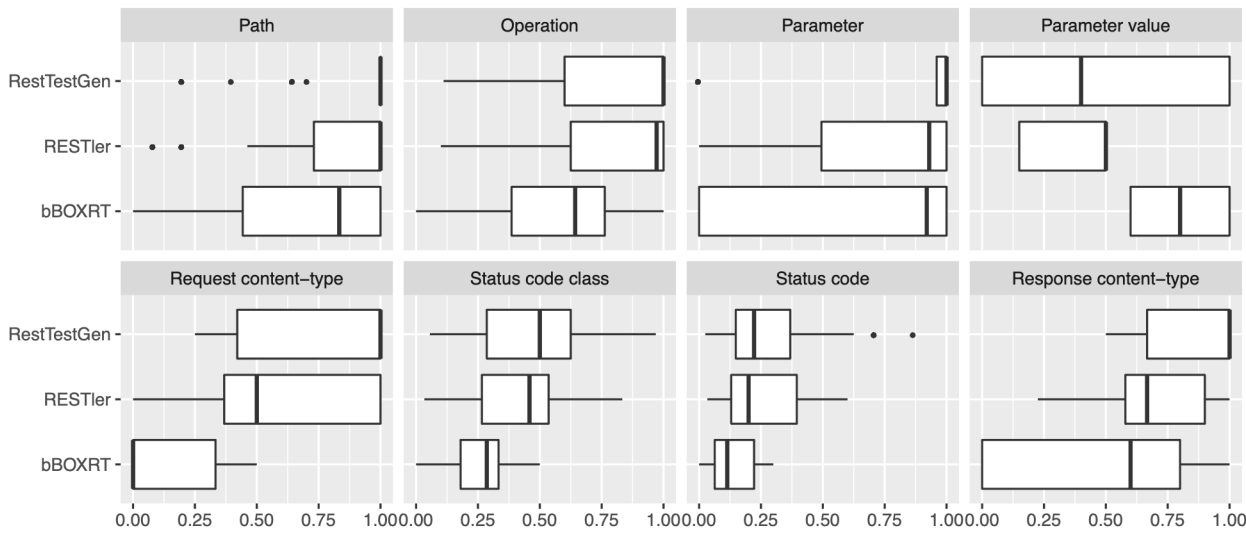
1. How *robust* are automated RESTful APIs test-case generation tools?
2. What is the *coverage* of the test suites emitted by automated RESTful APIs test-case generation tools?



# Results: robustness

Case study	RestTestGen 	RESTler 	bBOXRT 	REStest
01-Slim	✓	✓	✗	✓
02-Airline	✗	✓	✗	✗
03-Streaming	✗	✓	✗	✗
04-Petclinic	✓	✓	✗	✗
05-Toggle	✓	✓	✓	✗
06-Problems	✓	✓	✗	✗
07-Products	✓	✓	✓	✗
08-Widgets	✓	✓	✓	✓
09-Safrs	✓	✓	✓	✗
10-Realworld	✓	✓	✓	✗
11-Crud	✓	✓	✓	✗
12-Order	✓	✓	✓	✗
13-Users	✓	✓	✓	✗
14-Scheduler	✗	✓	✗	✗
<b>Total</b>	<b>11</b>	<b>14</b>	<b>8</b>	<b>2</b>

# Results: coverage



Coverage metric	RestTestGen	RESTler	bBOXRT	Draw
Path	1	0	0	7
Operation	1	3	0	4
Parameter	1	0	0	4
Parameter value	1	0	2	0
Req. content-type	2	1	0	4
Status code class	4	4	0	0
Status code	5	3	0	0
Resp. content-type	3	2	0	2

Number case studies for which a tool performed better than the others.



# Considerations

- One sequence Vs many sequences
  - RestTestGen when time and resources are limited
  - RESTler when a lot of time and resources are available
- bBOXRT is great for fault detection
  - High score for parameter value metric
- RESTest is still not mature for real-world REST APIs
  - Inter-parameter dependencies can be helpful in testing



# Testing Mass Assignment Vulnerabilities





# Auto-binding

## User

- email : String
- username : String
- password : String
- admin : boolean

```
POST /user/register  
Content-Type: application/json
```

```
{  
  "email": "mariano@univr.it",  
  "username": "mariano",  
  "password": "123456"  
}
```

HTTP request

## User

```
email String = "mariano@univr.it"  
- username : String = "mariano"  
- password : String = "123456"  
- admin : boolean = false
```

Object automatically instantiated  
from HTTP request



# Mass assignment vulnerability

```
POST /user/register
Host: example.com

{
  "email": "mariano@univr.it",
  "username": "mariano",
  "password": "123456",
  "admin": true
}
```

HTTP request

## User

- email : String = "mariano@univr.it"
- username : String = "mariano"
- password : String = "123456"
- **admin : boolean = true**

Object automatically instantiated from HTTP request

# Problem definition

- Mass assignment vulnerability is common in REST APIs<sup>1</sup>

## Our solution:

- ✓ Automatic testing of REST API for mass assignment vulnerabilities
  - Black-box perspective



# Approach



Static analysis of specification: identification of read-only fields



Automated generation of security test cases



Security testing oracle: vulnerability exploitation detection

# 1. Identification of read-only fields

POST /user/register <span style="float: right;">C</span>	
Input data	email username password
Output data	-

GET /user/{username} <span style="float: right;">R</span>	
Input data	username
Output data	email username password admin

POST /book	
Input data	title author
Output data	id title author

$R = \{e\}$   
 $W = \{e\}$   
 $RO = R$

# Notation

- **C** create: POST
- **R** read: GET
- **U** update: PUT
- **D** delete: DELETE



# Notation

- $C_T$  create a resource of type T
- $R_T$  read a resource of type T
- $U_T$  update a resource of type T
- $D_T$  delete a resource of type T



# Notation

- $C^{+f}$  create a resource adding the read-only input  $f$
- $U^{+f}$  update a resource adding the read-only input  $f$





# Notation

- $(D,R)?$  These operations are optional



# 2. Test case generation

Abstract test templates


$$\langle C_{\tau}^{+f}, R_{\tau}, (D_{\tau}, R_{\tau})^? \rangle$$

$$\langle C_{\tau}, R_{\tau}, U_{\tau}^{+f}, R_{\tau}, (D_{\tau}, R_{\tau})^? \rangle$$



# 2. Test case generation

Identification of resource-id parameters

$$\langle C_{\tau}^{+f}, R_{\tau}, (D_{\tau}, R_{\tau}) \rangle$$


bookId → 

username → 

password → 

# 3. Vulnerability exploitation detection

$$\langle C_{\tau}^{+f}, R_{\tau}, (D_{\tau}, R_{\tau}) \rangle$$

```
POST /user/register
Host: example.com

{
  "email": "mariano@univr.it",
  "username": "mariano",
  "password": "123456",
  "admin": true
}
```

HTTP request

```
HTTP/1.1 200 OK

{
  "email": "mariano@univr.it",
  "username": "mariano",
  "password": "123456",
  "admin": true
}
```



# Evaluation

- **RQ1:** What is the accuracy of the automated identification of operations CRUD semantics, resource types, and resource-id parameters?
- **RQ2:** What is the accuracy in revealing mass assignment vulnerabilities in REST APIs?
- **RQ3:** Does the proposed approach to detect mass assignment vulnerabilities scale to large REST APIs?



# Benchmark APIs

- Open-source
- Not read-only
- With OpenAPI specification

API	Prog. Lang.	REST framework	No. Of Operations	No. Of Vulnerabilities
<b>VAmPI</b>	Python	Flask	12	1
<b>OWASP</b>	Java	Spring	10	4
<b>Toggle</b>	ASP.NET	.NET Code	16	2
<b>Bookstore</b>	Java	Spring	5	1
<b>CRUD</b>	JavaScript	Express	4	2



# Results: accuracy of CRUD extraction, clustering, and resource-id identification

Case study	CRUD	Clustering	Resource-id
VAmPI	100%	100%	67%
OWASP	100%	80%	100%
Toggle	88%	88%	100%
Bookstore	100%	100%	100%
CRUD	100%	100%	100%



# Results: accuracy of vulnerability detection

Case study	<i>Safe</i>		<i>Vulnerable</i>					
	Tests	FP	Tests	TP	FP	FN	Pr	Re
VAmPI	4.0	0.0	4.0	1.0	0.0	0.0	100%	100%
OWASP	8.0	0.0	7.4	3.6	0.0	0.4	100%	90%
Toggle	2.0	0.0	2.0	2.0	0.0	0.0	100%	100%
Bookstore	2.0	0.0	2.0	1.0	0.0	0.0	100%	100%
CRUD	2.0	0.0	2.0	2.0	0.0	0.0	100%	100%



# Results: scalability of the approach

Case study	# Ops.	Time (s)	# Read-only fields
Gmail	68	3.0	23
Analytics	88	5.0	166
Calendar	37	2.0	11
Classroom	61	5.0	15
Custom Search	2	1.0	66
Drive	48	3.0	49
Fitness	13	1.4	4
My Business	50	7.4	527
Search Console	11	1.0	10
YouTube	76	8.4	110
<b>Total</b>	<b>454</b>	<b>37.2</b>	<b>981</b>

