42crunch

16 February 2023

# Are Your APIs Rugged?

**Colin Domoney**
Chief Technology Evangelist

@colindomoney

# About the speaker

**Colin Domoney**

*Chief Technology Evangelist*

DevSecOps specialist and evangelist, lifelong learner/hacker and latent developer

- **VP of AppSec at Deutsche Bank**
  - 20k developers, 6k app
  - Fixed over 3 million high severity flaws
  - Built global AppSec program
- **Innovation manager/DevRel/Solution Architect at Veracode Inc.**
  - Frequent speaker and blogger
  - Advised Fortune 100 on DevSecOps implementations
  - Advisor to Product Management team
- **Independent DevSecOps consultant**
- **Industry analyst and advisor**

# The need for rugged software

The Rugged Manifesto

*I recognize that software has become a **foundation of our modern world.***
*I recognize the **awesome responsibility** that comes with this foundational role.*

*I recognize that my code will be used in ways **I cannot anticipate**, in ways **it was not designed,** and for longer than **it was ever intended.***

*I recognize that my code will be attacked by **talented and persistent adversaries** who threaten our physical, economic, and national security.*

*https://ruggedsoftware.org/*

# Beyond secure … becoming rugged

## Secure

- Using transport security
- Authenticate users via standard methods
- Authorize access to:
  - Functions
  - Objects
- Validate input data
- Use standard methods for token exchange
- Use API gateways
- Eliminate common coding vulnerabilities

**“What you do to satisfy your regulators”**

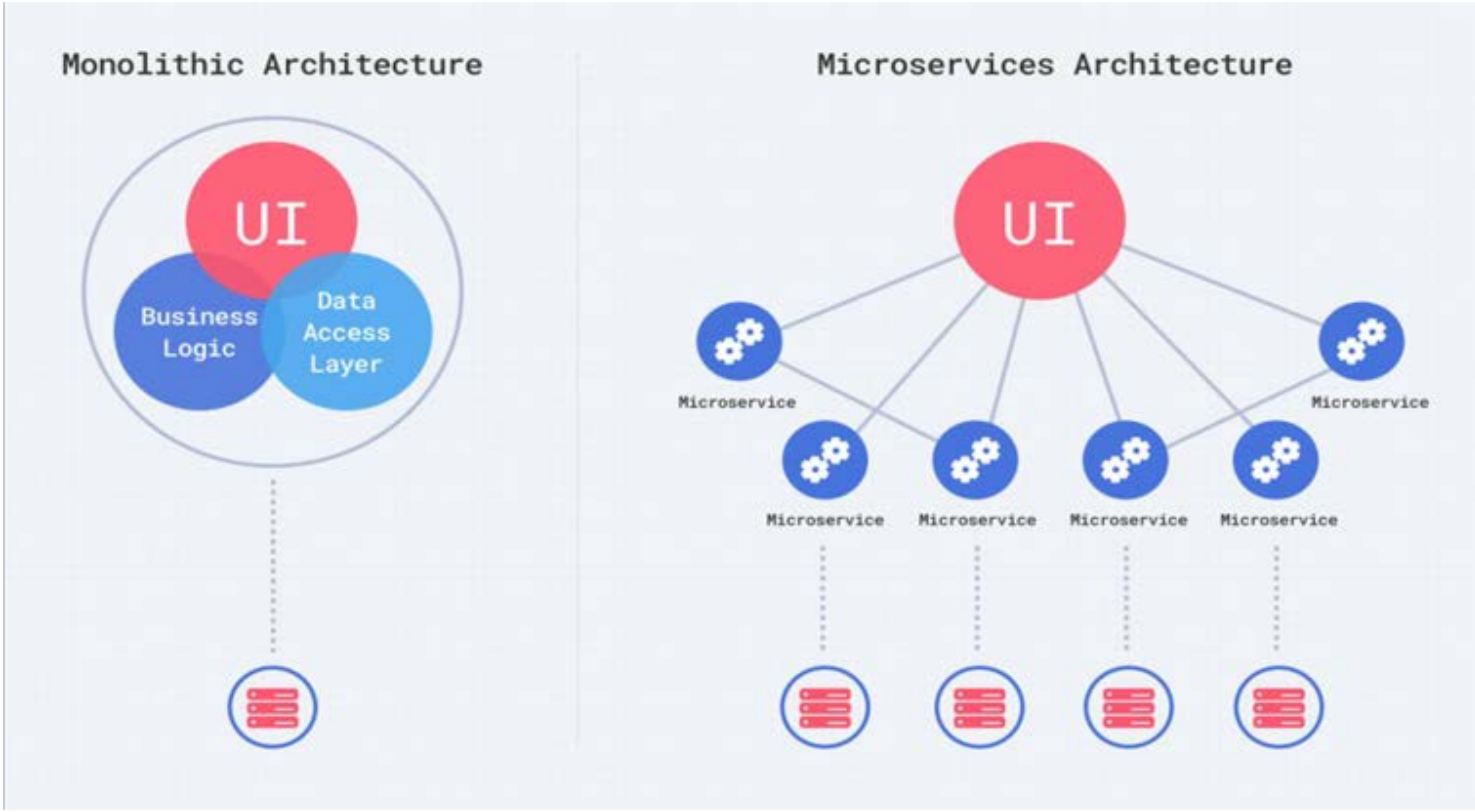
## Rugged

- Everything included by being **secure** !
- Use standard libraries and components
- Use defense-in-depth
- Manage your API inventory
- Manage access and abuse cases via:
  - Rate limiting
  - Quotas
- Restrict access via risk factors:
  - Known bad IP addresses
  - Common attack methods
- Attack your own APIs

**“What you do to delight your customers”**

# The need for API security

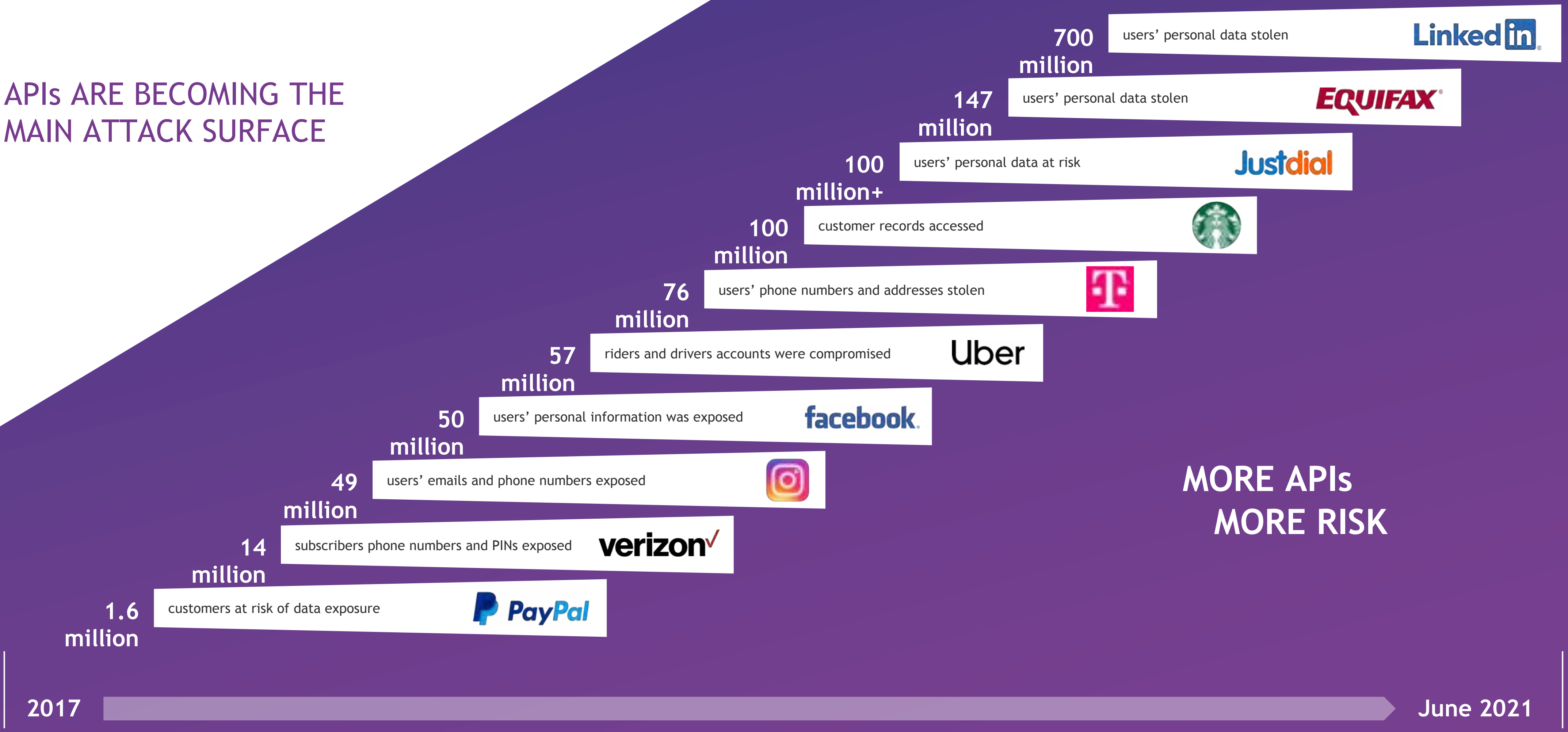
# From the monolith to microservices



# It’s all about the supply chain

# Growing Number and Scale of Attacks

APIs ARE BECOMING THE
MAIN ATTACK SURFACE



700 million users’ personal data stolen LinkedIn

147 million users’ personal data stolen EQUIFAX

100 million+ users’ personal data at risk Justdial

100 million customer records accessed

76 million users’ phone numbers and addresses stolen

57 million riders and drivers accounts were compromised Uber

50 million users’ personal information was exposed facebook

49 million users’ emails and phone numbers exposed

14 million subscribers phone numbers and PINs exposed verizon

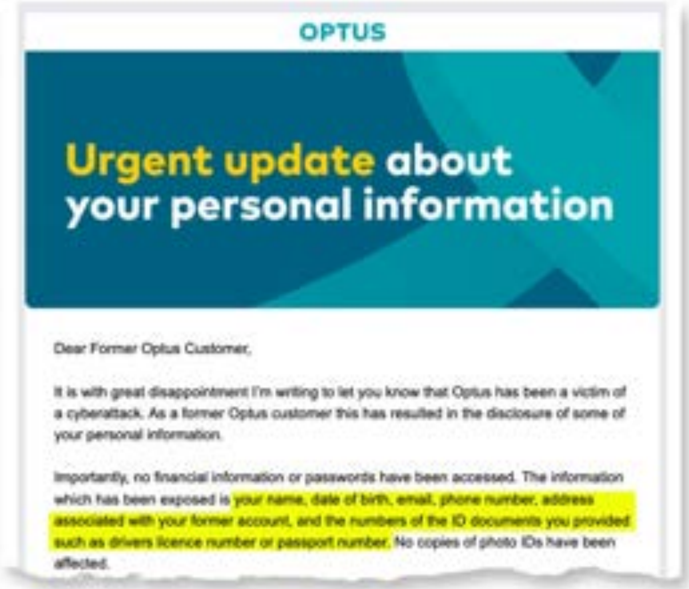
1.6 million customers at risk of data exposure PayPal

2017

June 2021

MORE APIs
MORE RISK

# The age of the API mega-breach



OPTUS

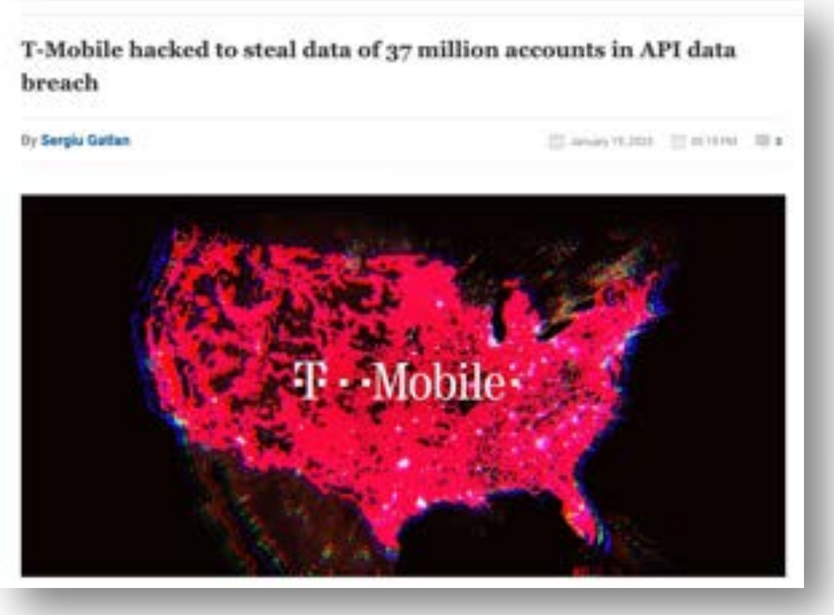
**Urgent update about your personal information**

Dear Former Optus Customer,

It is with great disappointment I'm writing to let you know that Optus has been a victim of a cyberattack. As a former Optus customer this has resulted in the disclosure of some of your personal information.

Importantly, no financial information or passwords have been accessed. The information which has been exposed is your name, date of birth, email, phone number, address associated with your former account, and the numbers of the ID documents you provided such as drivers licence number or passport number. No copies of photo IDs have been affected.

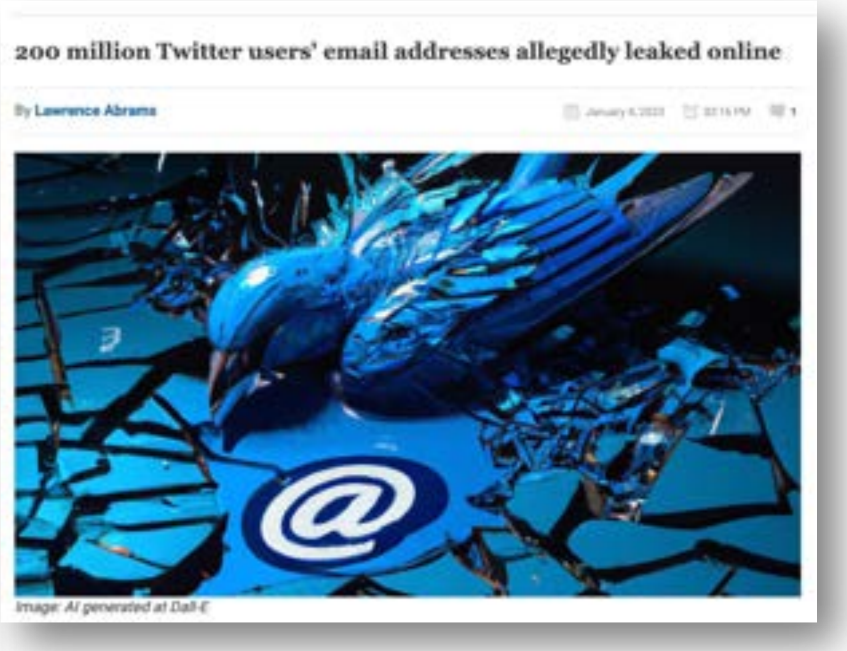
https://www.optus.com.au/about/media-centre/media-releases/2022/09/optus-notifies-customers-of-cyberattack



T-Mobile hacked to steal data of 37 million accounts in API data breach

By Sergiu Gatlan January 19, 2023 [illegible]

https://www.bleepingcomputer.com/news/security/t-mobile-hacked-to-steal-data-of-37-million-accounts-in-api-data-breach/



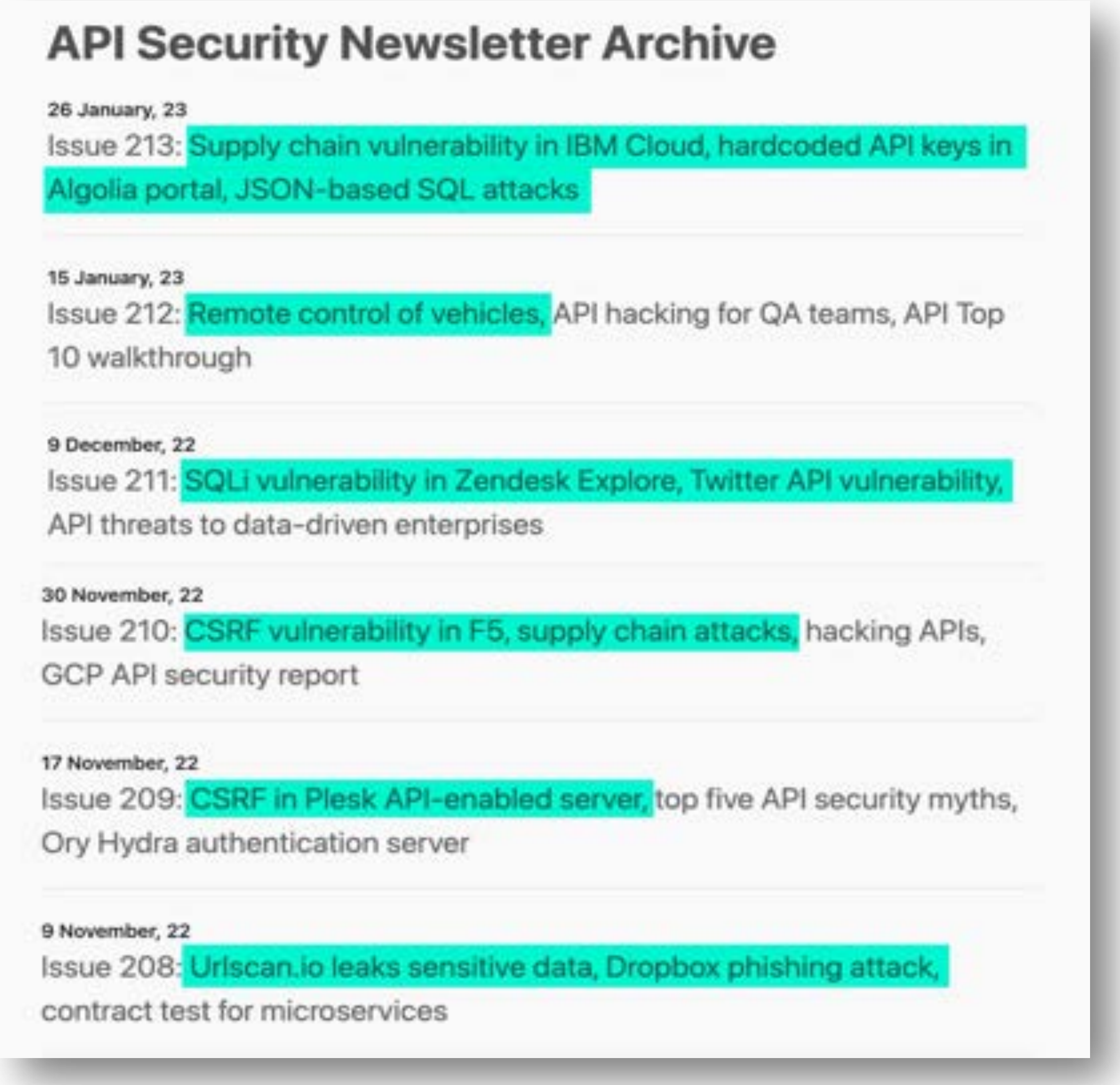
200 million Twitter users' email addresses allegedly leaked online

By Lawrence Abrams January 4, 2023 [illegible] 1

Image: AI generated at Dall-E

https://www.bleepingcomputer.com/news/security/200-million-twitter-users-email-addresses-allegedly-leaked-online/

# Every week brings a new story



## API Security Newsletter Archive

26 January, 23

Issue 213: Supply chain vulnerability in IBM Cloud, hardcoded API keys in Algolia portal, JSON-based SQL attacks

15 January, 23

Issue 212: Remote control of vehicles, API hacking for QA teams, API Top 10 walkthrough

9 December, 22

Issue 211: SQLi vulnerability in Zendesk Explore, Twitter API vulnerability, API threats to data-driven enterprises

30 November, 22

Issue 210: CSRF vulnerability in F5, supply chain attacks, hacking APIs, GCP API security report

17 November, 22

Issue 209: CSRF in Plesk API-enabled server, top five API security myths, Ory Hydra authentication server

9 November, 22

Issue 208: Urlscan.io leaks sensitive data, Dropbox phishing attack, contract test for microservices

https://apisecurity.io/

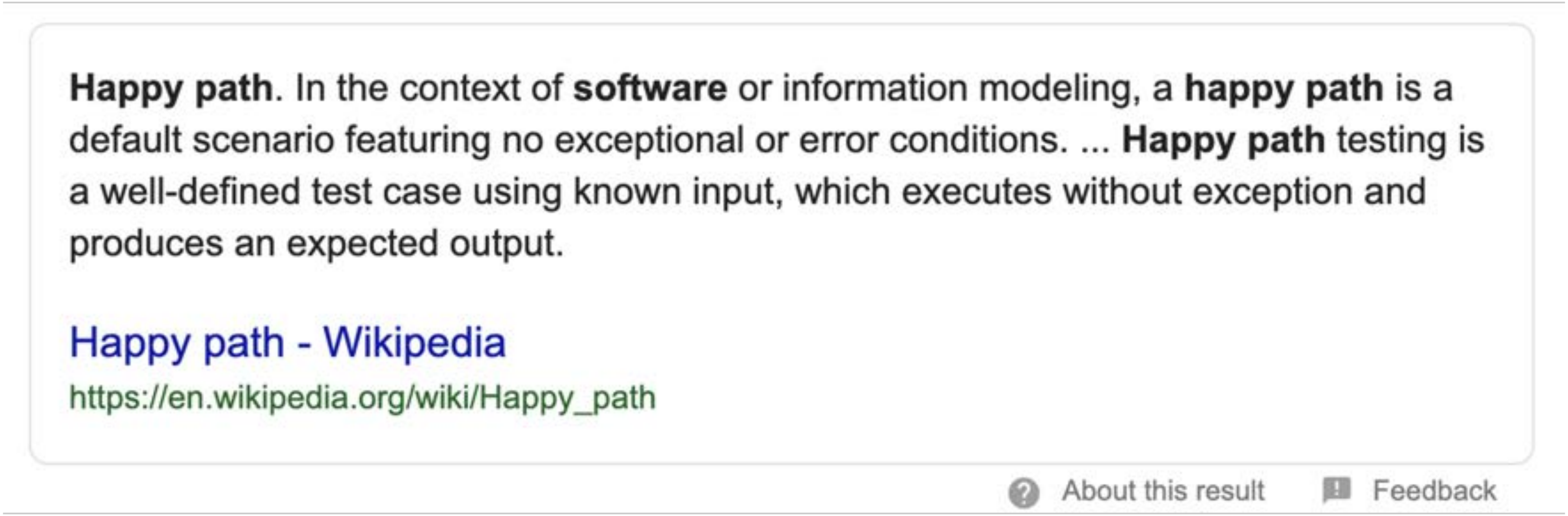
# The root cause of API vulnerabilities …

## Human error is the root cause of API vulnerabilities !

- Logic errors
- Poor design
- Coding errors
- Misuse of components/libraries
- Misconfiguration of servers
- Shortcuts
- Assumptions
- Insecure defaults
- Misunderstanding attack vectors
- Vulnerable dependencies

# Why does insecure software exist?

# Fail #1: Over optimism

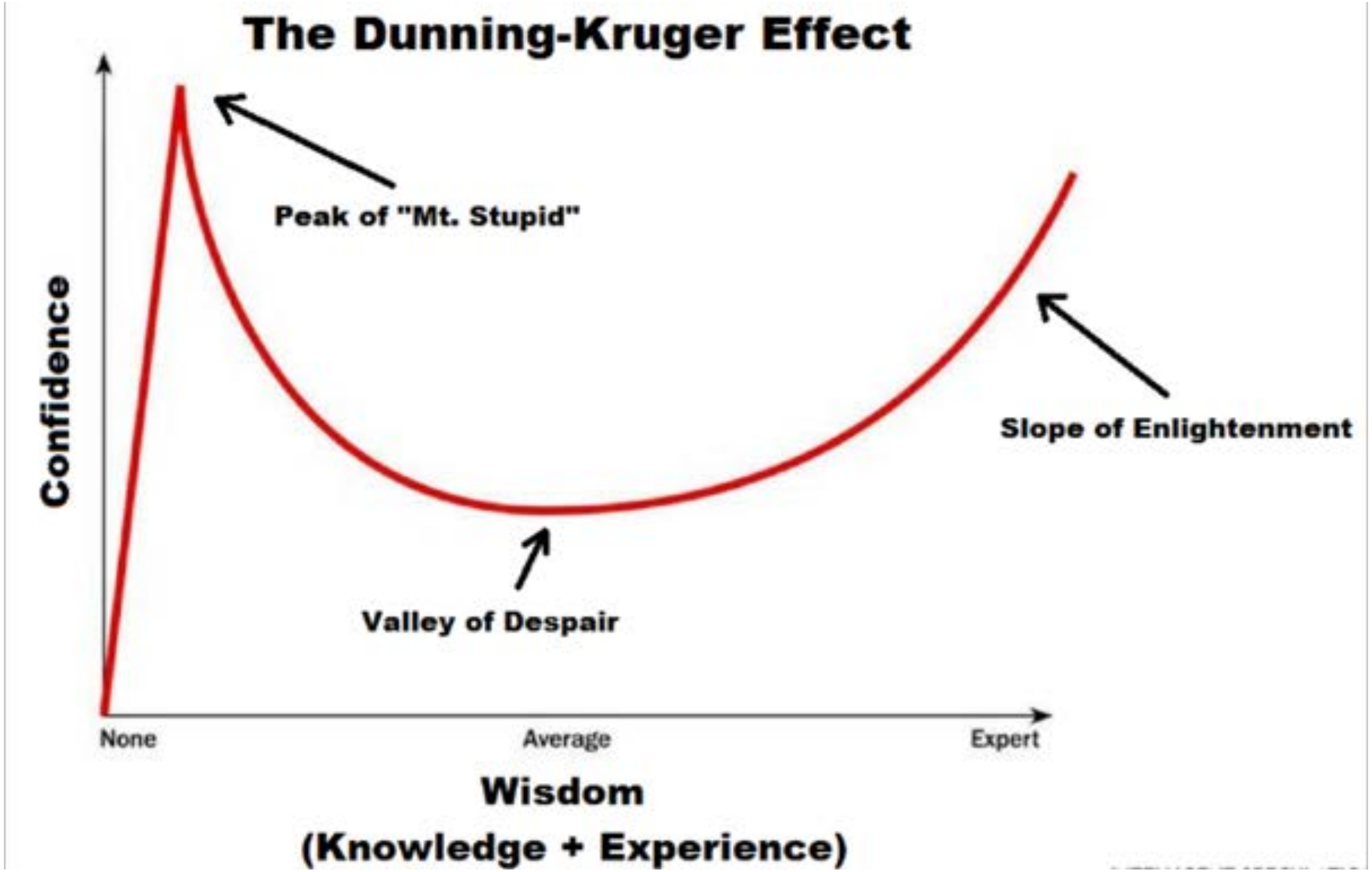


**Happy path**. In the context of **software** or information modeling, a **happy path** is a default scenario featuring no exceptional or error conditions. ... **Happy path** testing is a well-defined test case using known input, which executes without exception and produces an expected output.

Happy path - Wikipedia

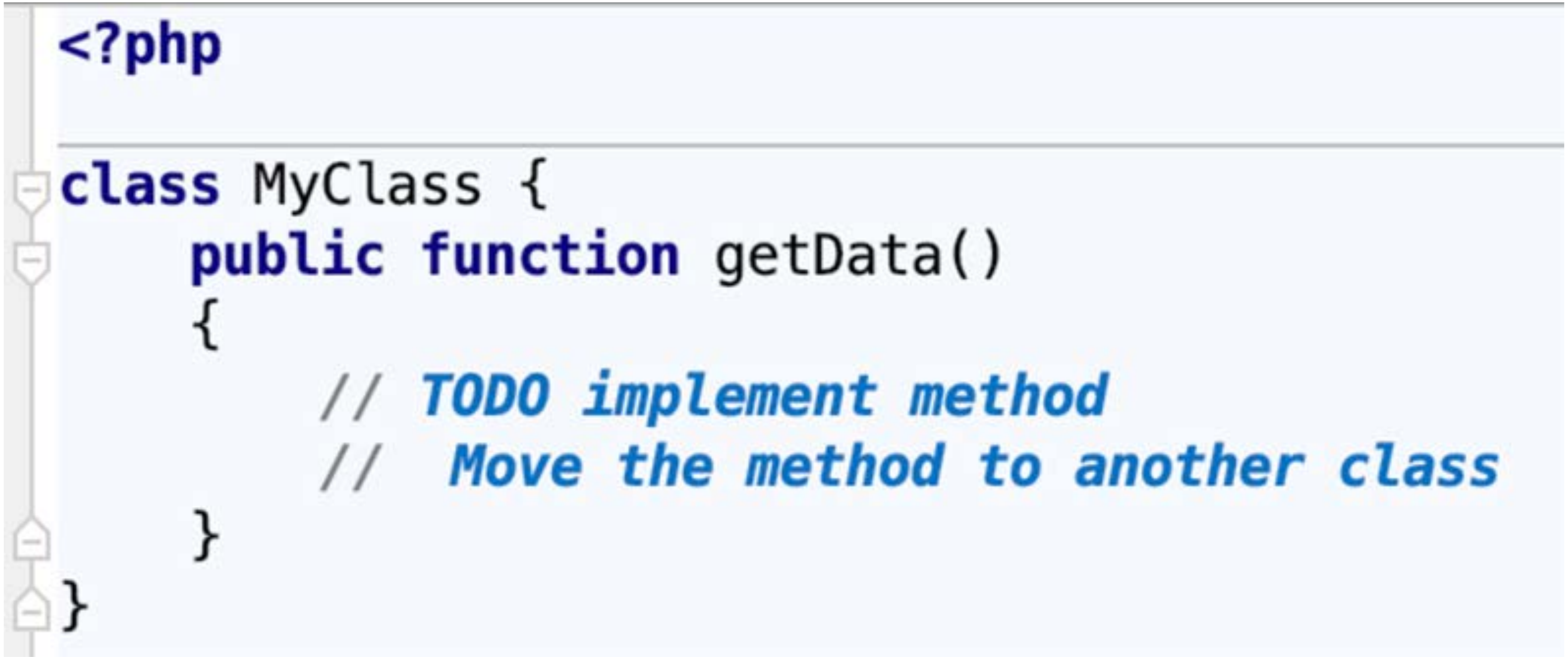
https://en.wikipedia.org/wiki/Happy_path

# Fail #2: Over confidence



# Fail #3: Bad things happen to others

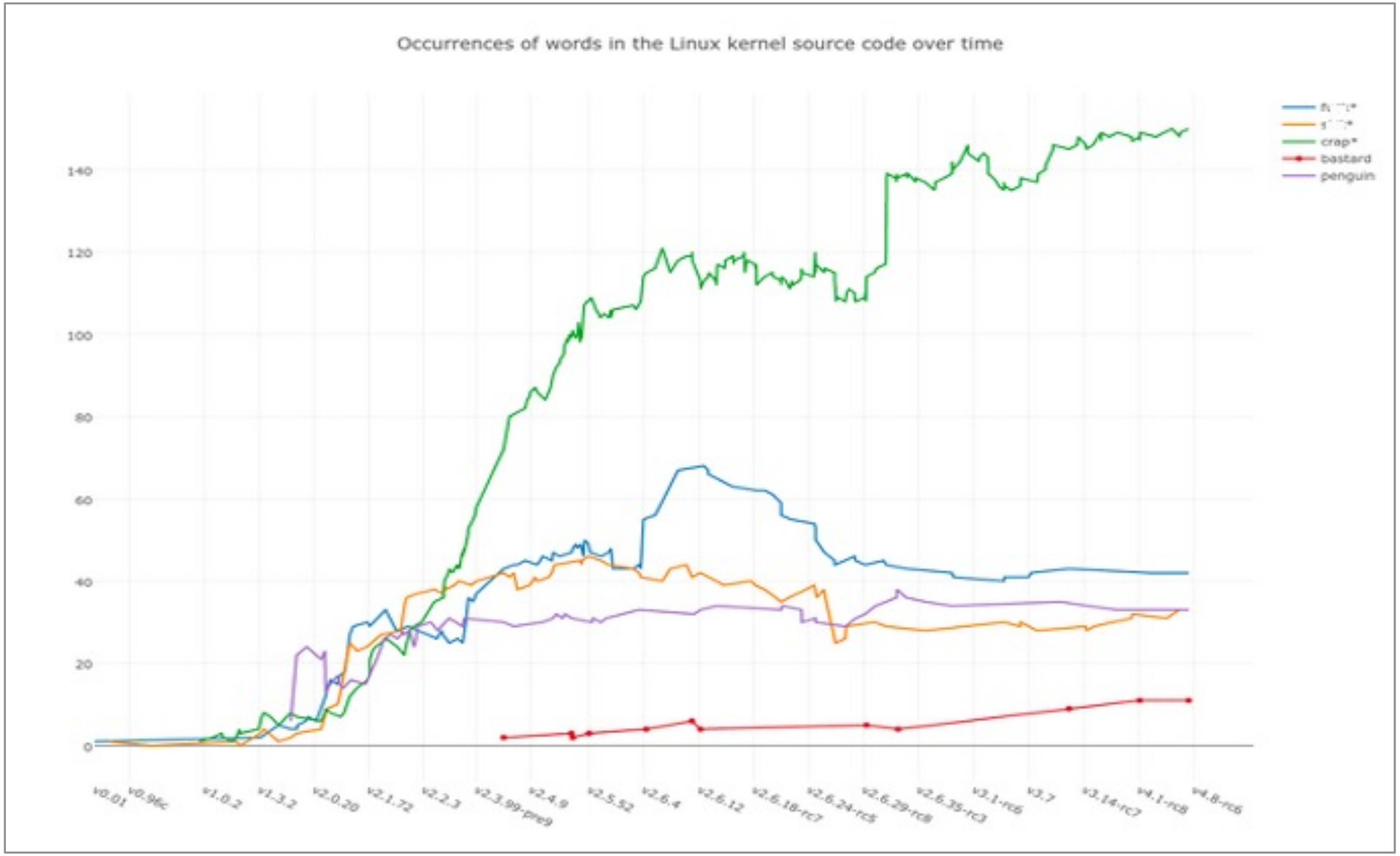
# Fail #4: Taking a shortcut



```php
<?php

class MyClass {
    public function getData()
    {
        // TODO implement method
        //  Move the method to another class
    }
}
```

# Fail #5: This stuff is difficult !



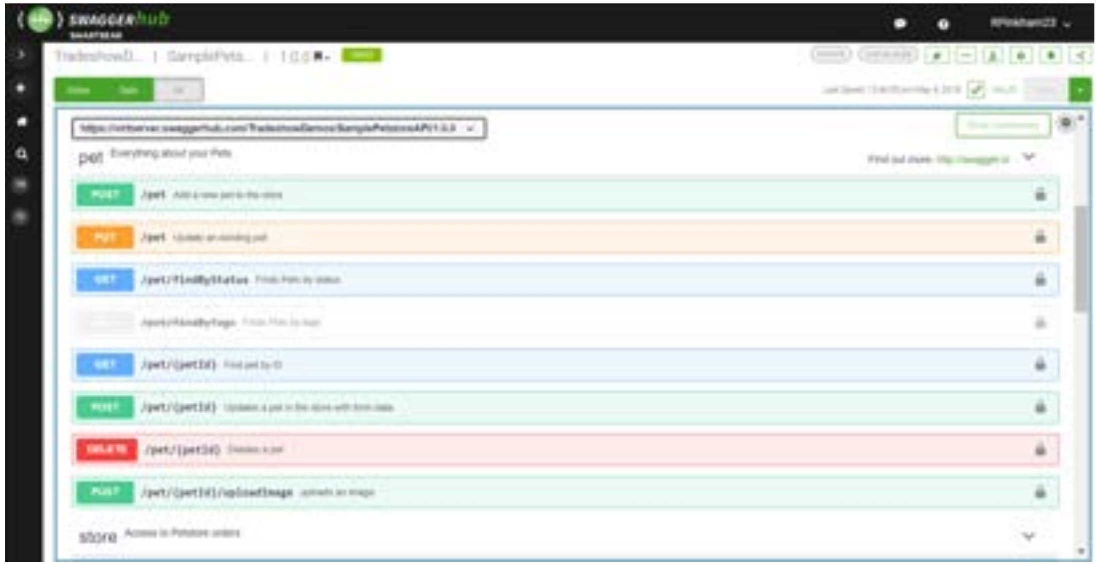
# Why API security is hard ?

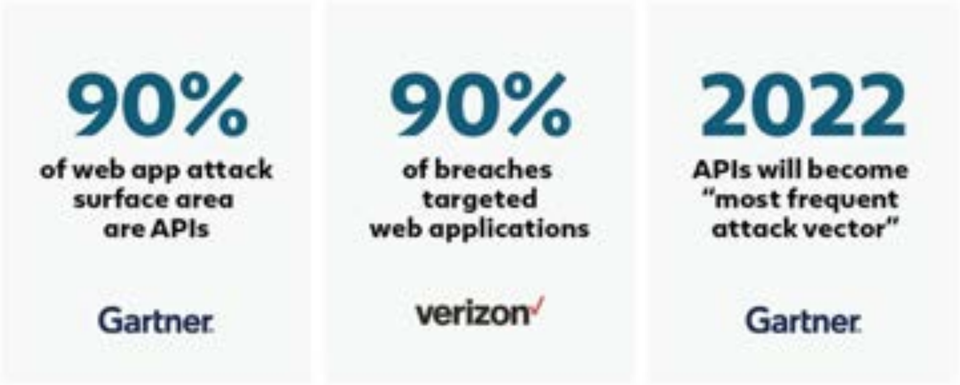
# API security is different to web security

| OWASP API Security Top 10 | OWASP Top 10 |
|---|---|
| API1:2019 Broken Object Level Authorization | A01:2021-Broken Access Control |
| API2:2019 Broken User Authentication | A02:2021-Cryptographic Failures |
| API3:2019 Excessive Data Exposure | A03:2021-Injection |
| API4:2019 Lack of Resources & Rate Limiting | A04:2021-Insecure Design |
| API5:2019 Broken Function Level Authorization | A05:2021-Security Misconfiguration |
| API6:2019 Mass Assignment | A06:2021-Vulnerable and Outdated Components |
| API7:2019 Security Misconfiguration | A07:2021-Identification and Authentication Failures |
| API8:2019 Injection | A08:2021-Software and Data Integrity Failures |
| API9:2019 Improper Assets Management | A09:2021-Security Logging and Monitoring Failures |
| API10:2019 Insufficient Logging & Monitoring | A10:2021-Server-Side Request Forgery |

# APIs are an easy target to attack

- They are easily **discoverable**
- They are well **documented**
- Attacks can be easily **automated**
- Excellent tools exist to automated attacks

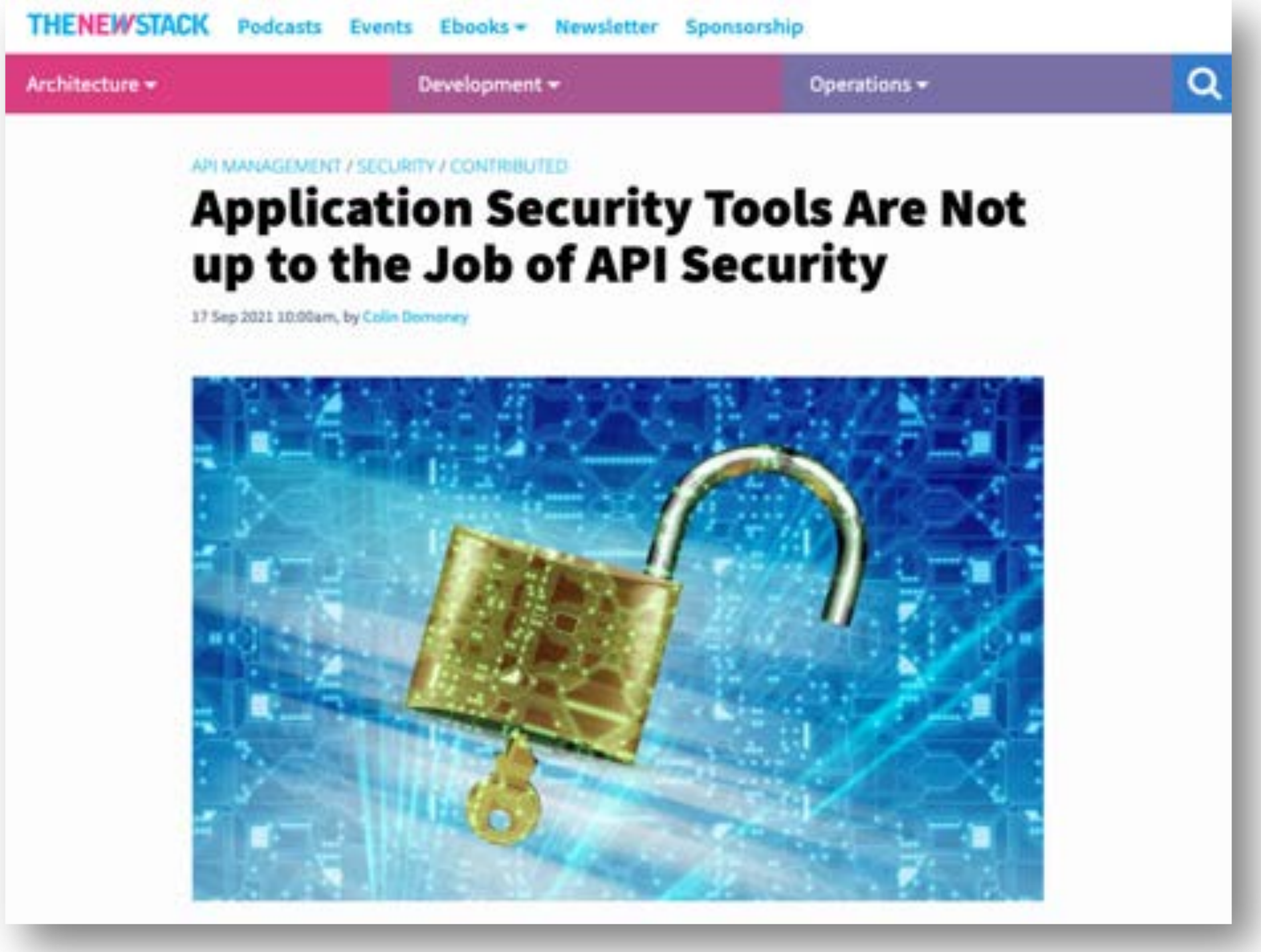



90%
of web app attack surface area are APIs
Gartner
90%
of breaches targeted web applications
verizon
2022
APIs will become "most frequent attack vector"
Gartner

https://outpost24.com/blog/what-is-api-security-and-how-to-protect-them

# Your existing tools probably don’t work well for APIs

- **SAST** — wasn’t designed for API-centric applications. Complex data flow paths or unsupported frameworks reduce the accuracy of a SAST analysis since the model may be incomplete or inaccurate.
- **DAST** — lacks context of APIs. DAST tools can’t provide an intelligent assessment of API security.
- **SCA** - useful but not sufficient
- **IAST** - complex to install and use



API MANAGEMENT / SECURITY / CONTRIBUTED

**Application Security Tools Are Not up to the Job of API Security**

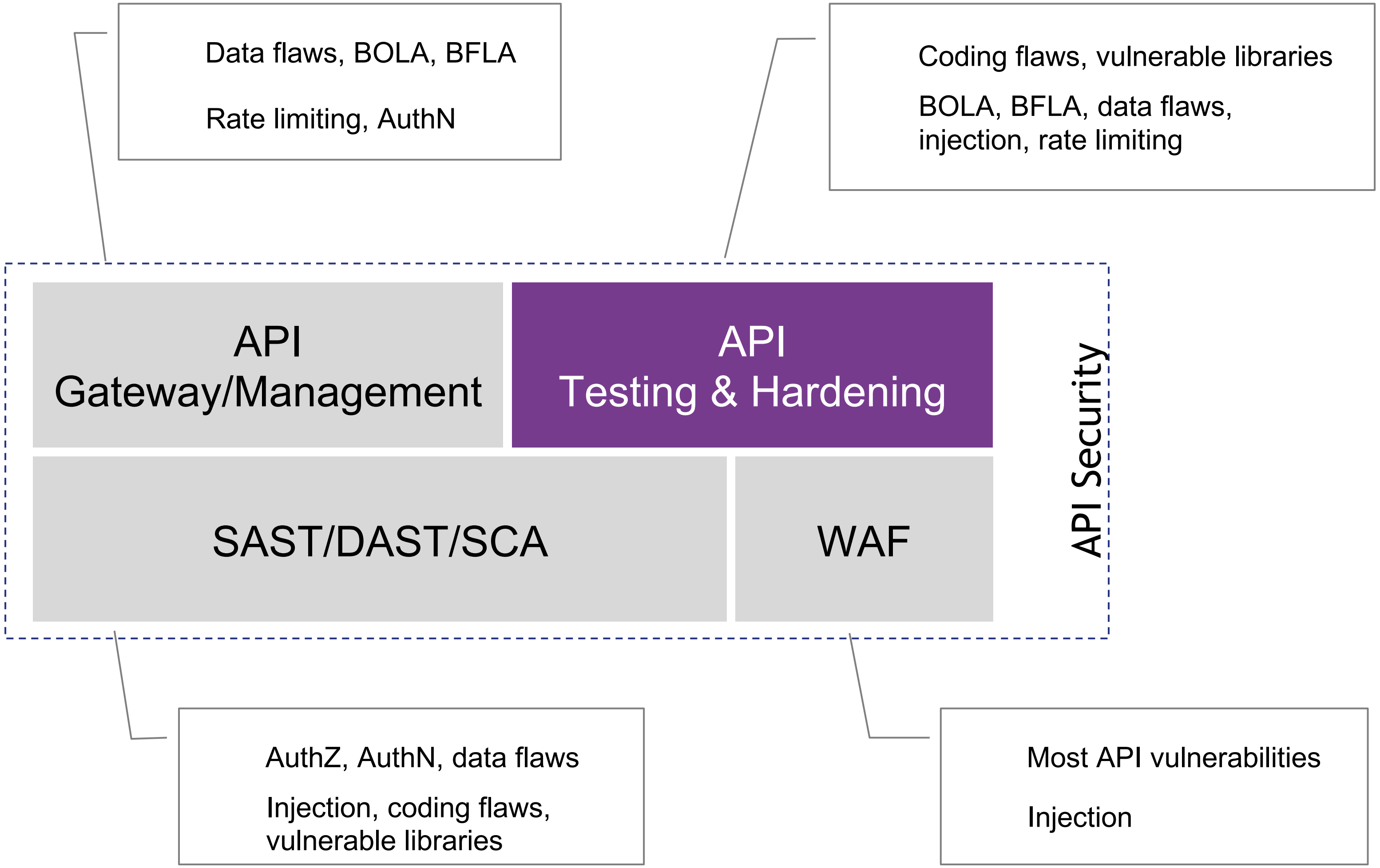
17 Sep 2021 10:00am, by Colin Domoney

*https://thenewstack.io/application-security-tools-are-not-up-to-the-job-of-api-security/*

# Your security architectures are built to protect this …

# With APIs, security measures need to protect this!

# Where does API security fit?

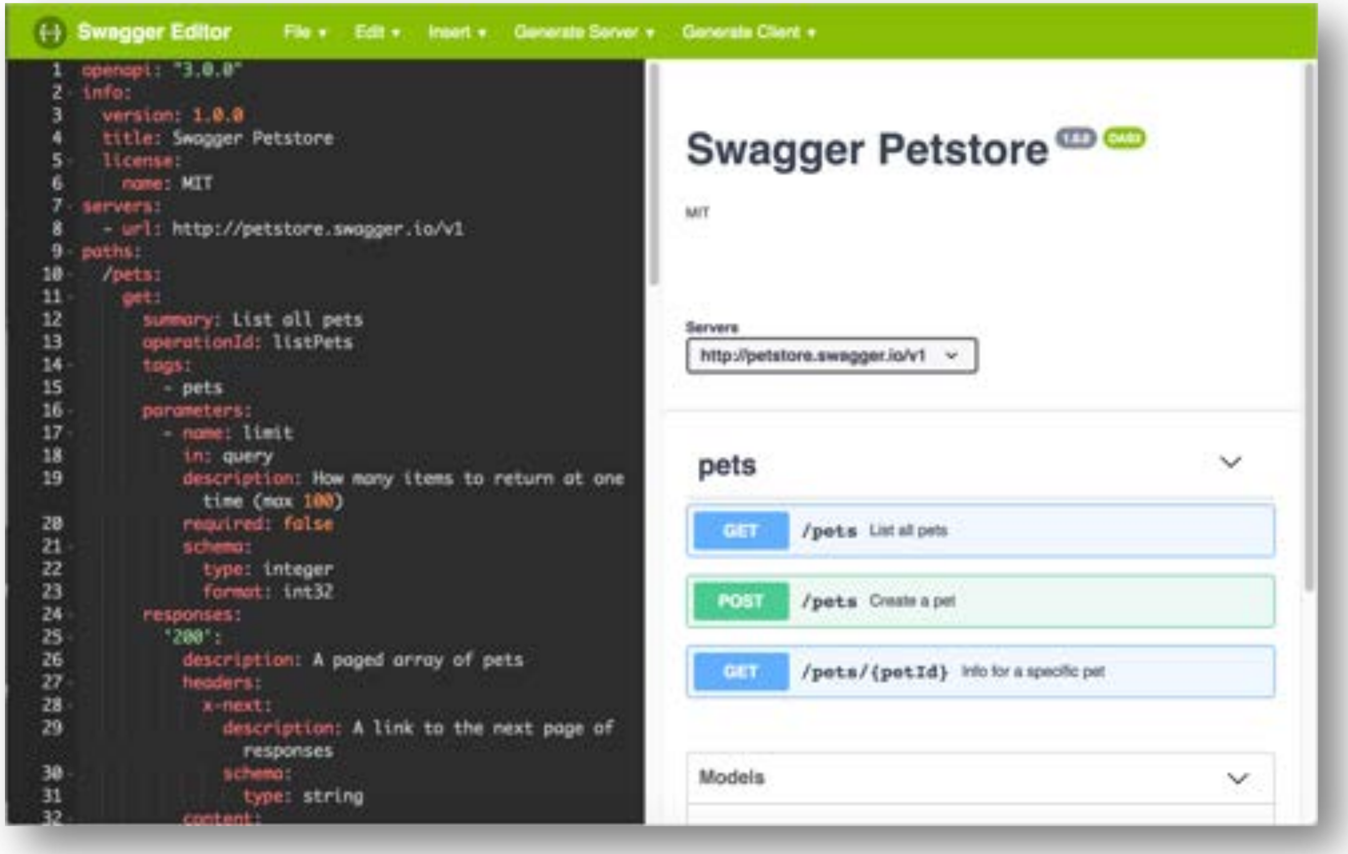


| # | OWASP API Top 10 Vulnerabilities |
|---|---|
| 1 | Broken Object Level Authorization |
| 2 | Broken User Authentication |
| 3 | Excessive Data Exposure |
| 4 | Lack of Resources & Rate Limiting |
| 5 | Broken Function Level Authorization |
| 6 | Mass Assignment |
| 7 | Security Misconfiguration |
| 8 | Injection |
| 9 | Improper Assets Management |
| 10 | Insufficient Logging & Monitoring |

# The unique opportunities for API security

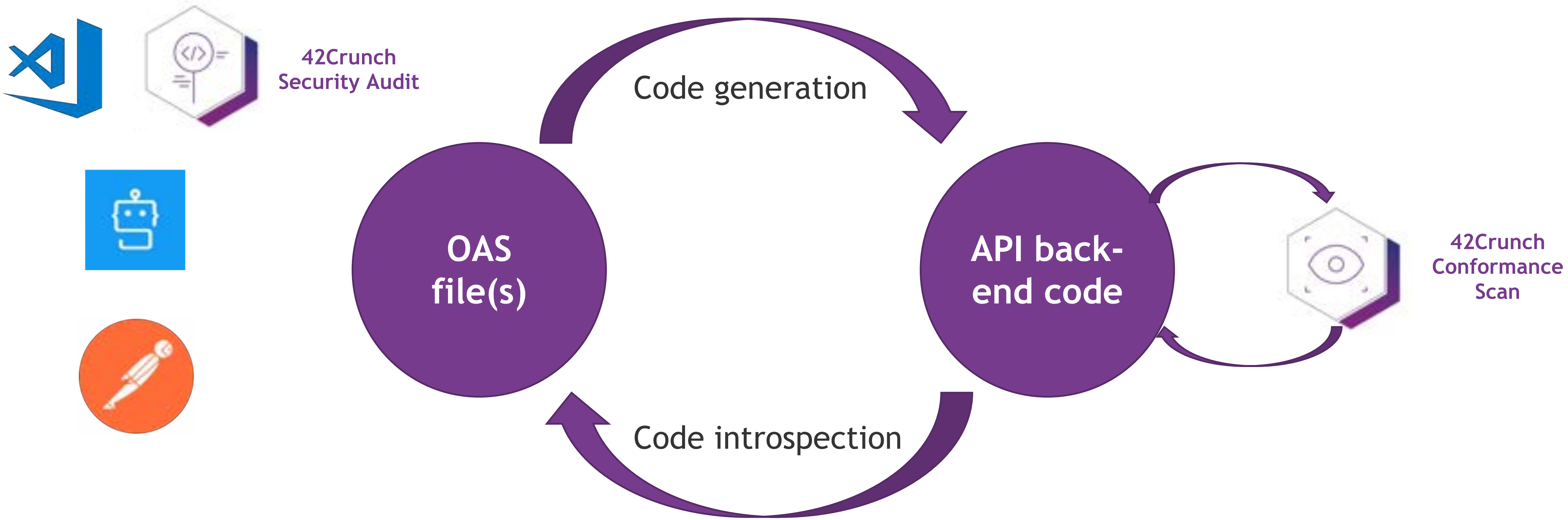
# OpenAPI Specification at the heart

- OAS forms a definitive contract for all downstream development
- OAS allows for a precise definition of request and response data types
- OAS allows operations to be tightly specified
- OAS allows security primitives to be specified
- Extensions allow for additional primitives to be included



```yaml
openapi: "3.0.0"
info:
  version: 1.0.0
  title: Swagger Petstore
  license:
    name: MIT
servers:
  - url: http://petstore.swagger.io/v1
paths:
  /pets:
    get:
      summary: List all pets
      operationId: listPets
      tags:
        - pets
      parameters:
        - name: limit
          in: query
          description: How many items to return at one
            time (max 100)
          required: false
          schema:
            type: integer
            format: int32
      responses:
        '200':
          description: A paged array of pets
          headers:
            x-next:
              description: A link to the next page of
                responses
              schema:
                type: string
          content:
```

# Use OAS as the core of a ‘shift left’ process



42Crunch Security Audit

Code generation

OAS file(s)

API back-end code

42Crunch Conformance Scan

Code introspection

# The benefits of a positive security model

## Allowlist vs. Blocklist

**Allowlist**

- Allowed data types strong defined and enforce in OAS mode
- Data format can be precisely defined
- Operations can be fully specified too

- **Only allow data conforming to specification — anything else is an error**
- **Only allows “known good”**

**Blocklist**

- Attempts to interpret data based on the runtime context i.e., Javascript, HTML
- Attempt to block what shouldn’t be present in a given context
- Can easily be subverted with encoding, etc.

- **Attempts to block “known bad”**

# Building guardrails – trust, but verify

- Provide the security **tooling and solutions**
- Provide the **guidance** on usage
- Set the **policies and standards**
- Implement **governance**

- Give the developers the **freedom** to implement their solutions

# Getting rugged with your APIs

# Rugged by design

- **Threat model** your APIs
- Think like an **attacker**
- Understand your **data**
  - Privacy
  - Revocation
- Understand your **environments**
  - Secret storage
  - Device integrity

thaddeus e. grugq
@thegrugq

Your threat model is not my threat model.

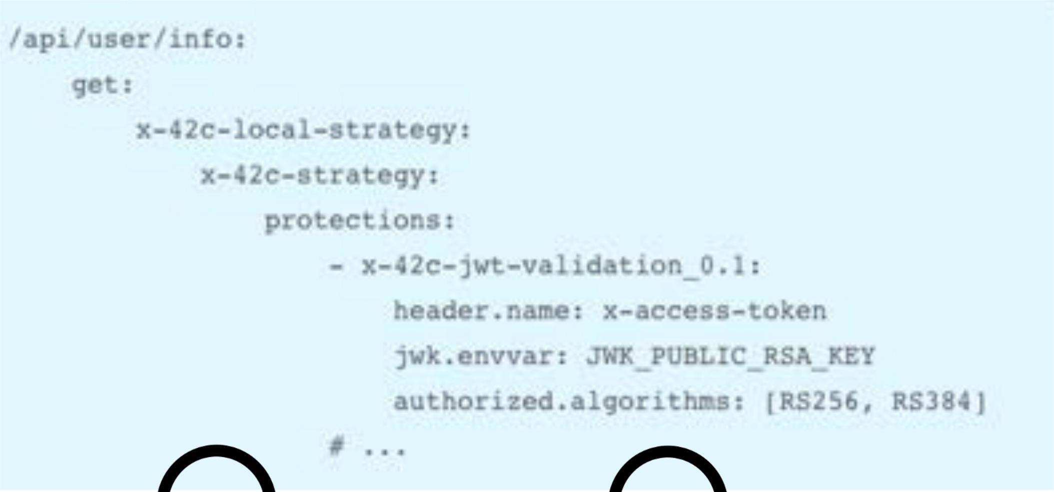
8:42 AM - May 15, 2017 - Tweetbot for iOS

703 Retweets 1.3K Likes

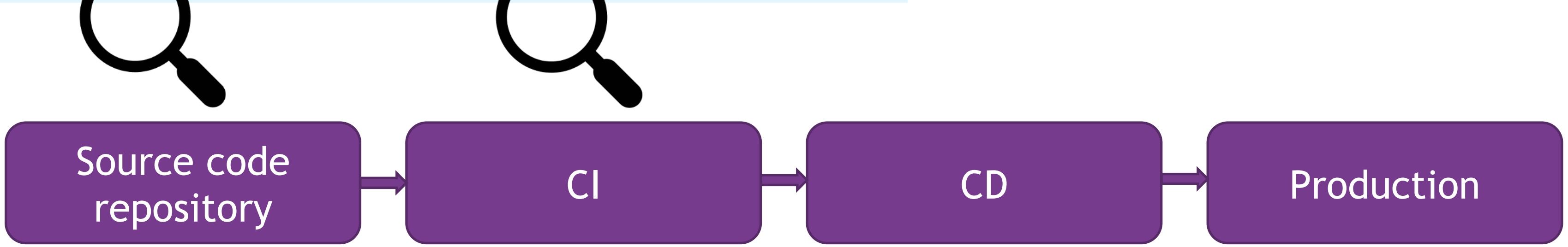
# Rugged by implementation

- Use standard **protocols**
- Use standard **libraries/components**
  - AuthN
  - AuthZ
  - Crypto
- Validate all **data**
- Understand your **framework/middleware**
  - Disable unused paths/methods
  - Beware of defaults

# Verify by inspection



```
/api/user/info:
    get:
        x-42c-local-strategy:
            x-42c-strategy:
                protections:
                    - x-42c-jwt-validation_0.1:
                        header.name: x-access-token
                        jwk.envvar: JWK_PUBLIC_RSA_KEY
                        authorized.algorithms: [RS256, RS384]
                    # ...
```



# Rugged at runtime

- Use your **API gateway**
  - Rate limiting
  - Token validation
- Integrate with **SIEM/SOC**
  - Threat detection
  - Bot detection
- Use **Cloud protections**
  - DDoS
  - Firewalls

Azure Sentinel

apigee WSO2

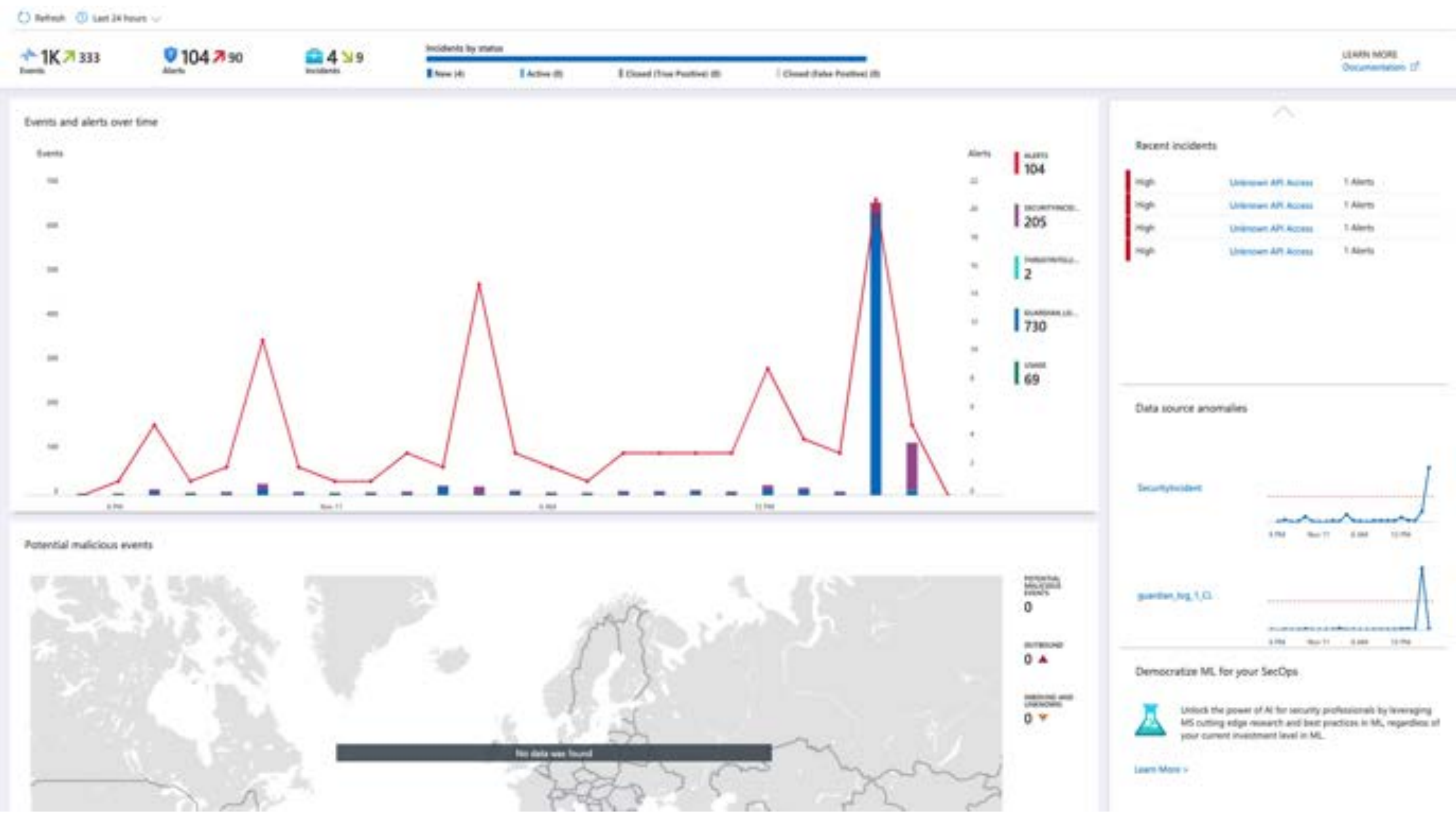
# Input data validation

- **No Trust** (even for internal APIs and for East-West traffic
- Validation can happen client side, but it must happen server-side!
- Do not blindly update data from input structure
  - Apply caution when using frameworks that map directly database records to JSON objects
- Do not use the same data structures for **GET** and **POST/PUT**
- **Validate Inputs**
  - Only accept information specified in JSON schema (contract-based, allowlist approach) Reject all others.
  - Also validate Headers
- How to test
  - Send bad verbs, bad data, bad formats, out of bounds, etc.

# Output data validation

- Never rely on client apps to filter data; instead, create various APIs depending on consumer, with just the data they need
- Take control of your JSON schemas !
  - Describe the data thoroughly and enforce the format at runtime
- Review and approve data returned by APIs
- Never expose tokens/sensitive/exploitable data in API responses
- Properly design error messages - make sure they are not too verbose!
- Beware of GraphQL queries!
  - Validate fields accessed via query

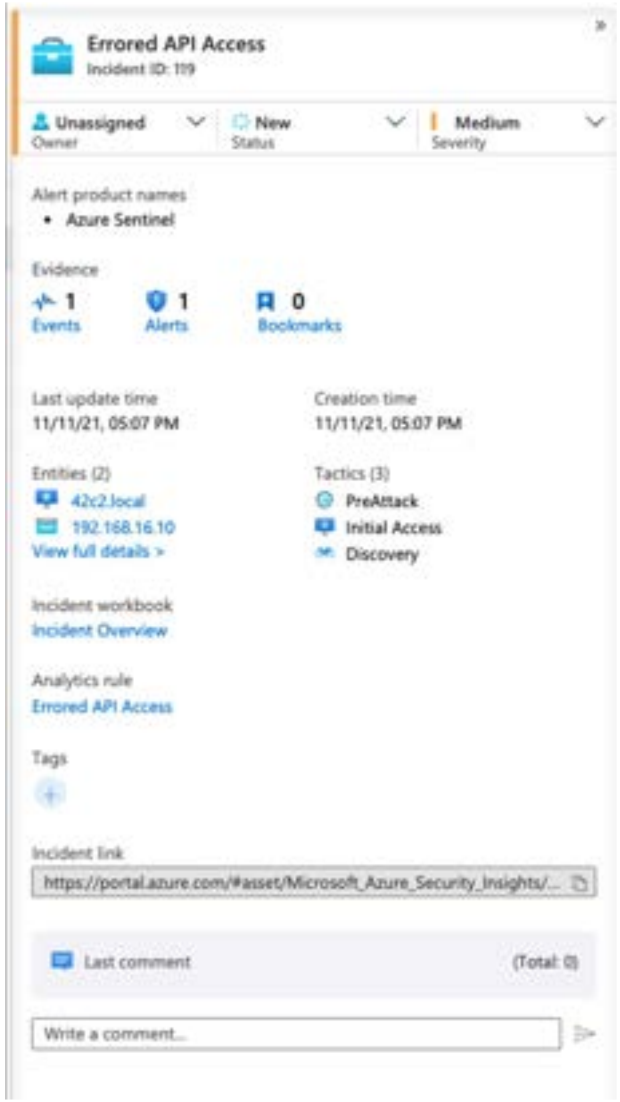
# Continuous protection – integration with Azure Sentinel



Recent incidents

| | | |
|---|---|---|
| High | Unknown API Access | 1 Alerts |
| High | Unknown API Access | 1 Alerts |
| High | Unknown API Access | 1 Alerts |
| High | Unknown API Access | 1 Alerts |

Unlock the power of AI for security professionals by leveraging MS cutting edge research and best practices in ML, regardless of your current investment level in ML.



Errored API Access
Incident ID: 119

Unassigned
Owner

New
Status

Medium
Severity

Alert product names
- Azure Sentinel

Evidence
1 Events
1 Alerts
0 Bookmarks

Last update time
11/11/21, 05:07 PM

Creation time
11/11/21, 05:07 PM

Entities (2)
42c2.local
192.168.16.10
View full details >

Tactics (3)
PreAttack
Initial Access
Discovery

Incident workbook
Incident Overview

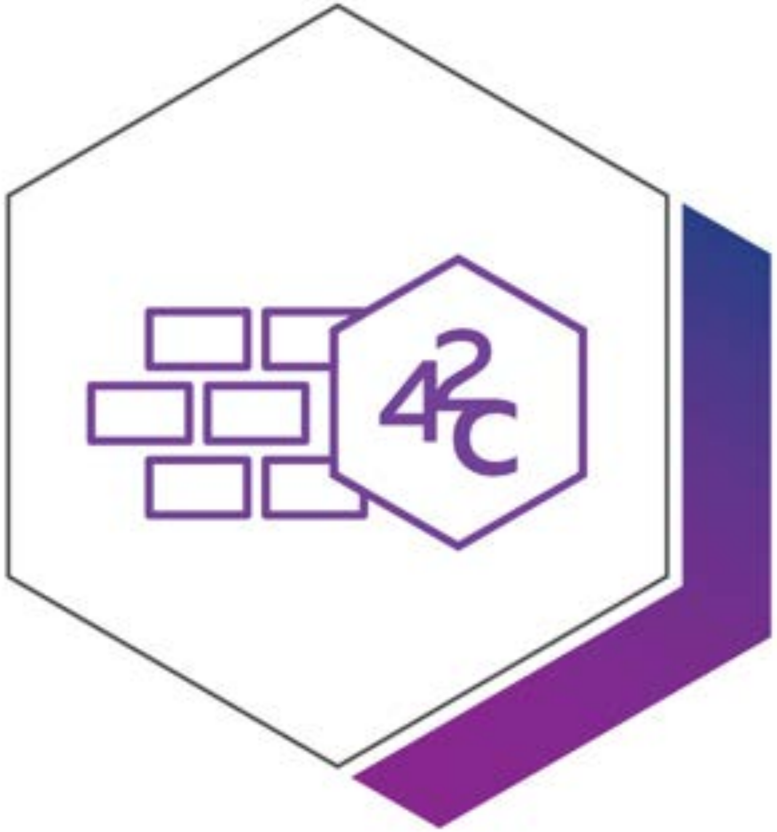
Analytics rule
Errored API Access

Tags

Incident link
https://portal.azure.com/#asset/Microsoft_Azure_Security_Insights/...

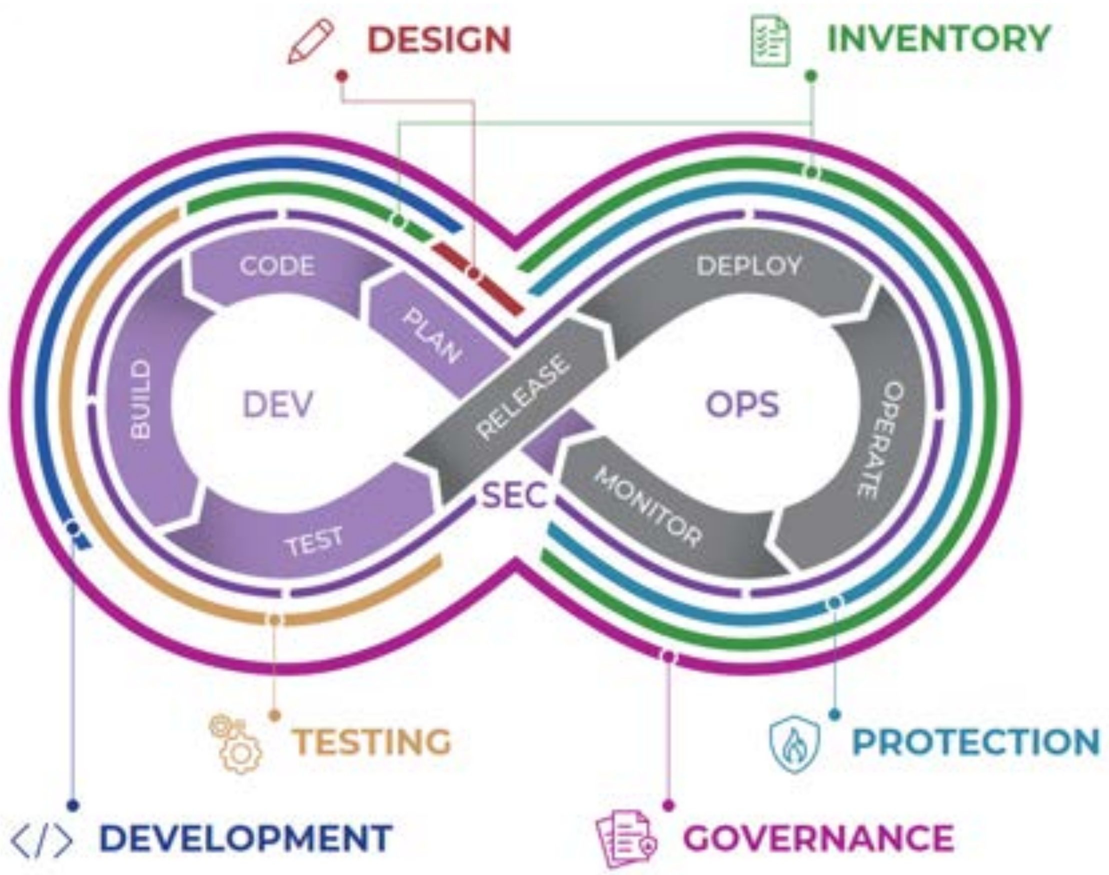
Last comment (Total: 0)

Write a comment...



# Our approach to API security

# The six pillars of API security



### API INVENTORY

Do you understand what APIs you own? Do you track shadow and zombie APIs?

### API DESIGN

Are you doing API-design-first? Do you incorporate security into the design phase?

### API DEVELOPMENT

Are your developers trained to code securely? Do they understand API security threats and risks?

### API TESTING

Are you doing automated API testing? Are you considering security in your test strategy?

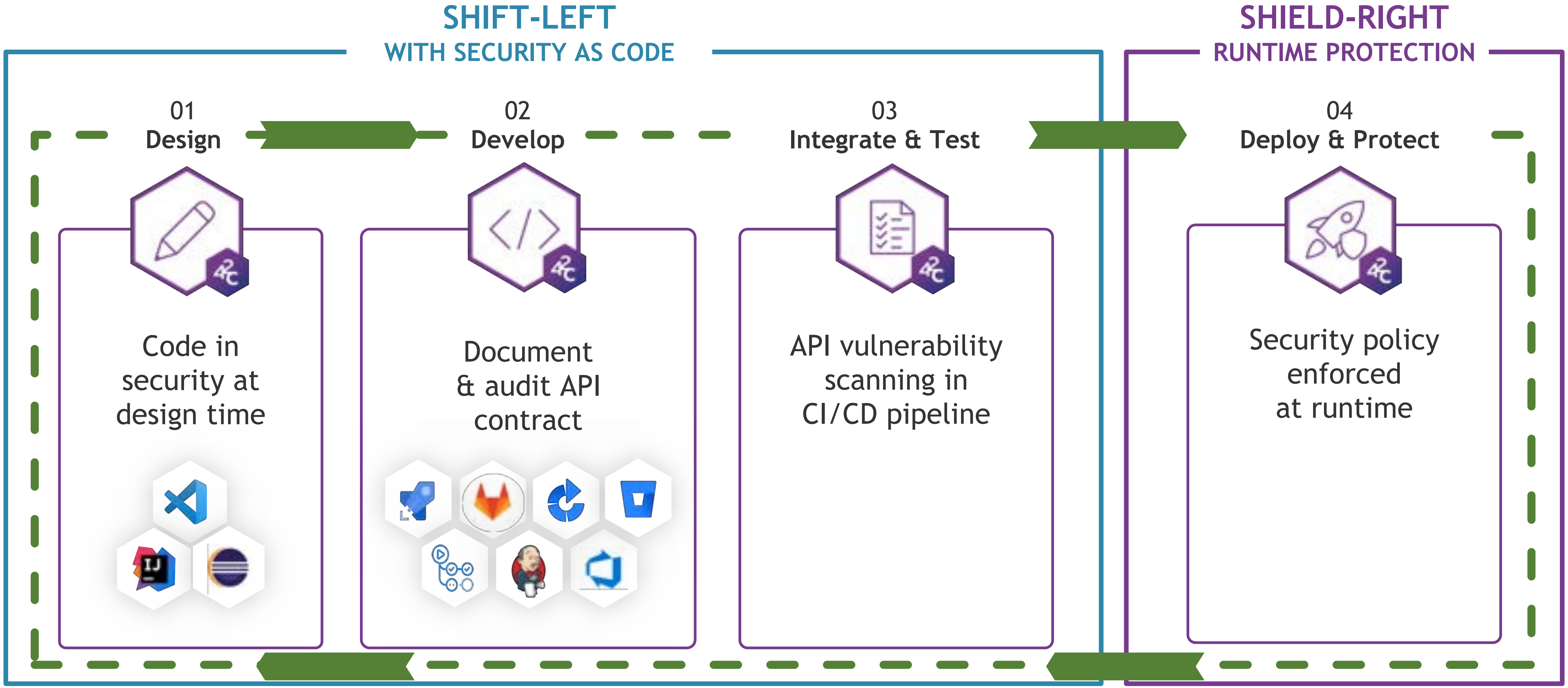
### API PROTECTION

Are you using API protection technology (WAFs, WAAPs, API gateways) in your deployments?

### API GOVERNANCE

Do you control and actively monitor your API estate and environments?

# Shift-Left, Shield-Right



SHIFT-LEFT
WITH SECURITY AS CODE

SHIELD-RIGHT
RUNTIME PROTECTION

01
Design

Code in security at design time

02
Develop

Document & audit API contract

03
Integrate & Test

API vulnerability scanning in CI/CD pipeline

04
Deploy & Protect

Security policy enforced at runtime

# The developer first API security platform

## SECURITY MANAGEMENT & GOVERNANCE

Visibility & control of security policy enforcement throughout API lifecycle for security teams.

## API AUDIT

Lock down your API's definitions to reduce the attack surface and remove potential security gaps.

## API SCAN

Dynamic runtime testing of your API to ensure compliance with API Contracts.

## API PROTECT

Protect each API with an API micro-firewall to distinguish legitimate traffic from malicious API attacks.

## INTEGRATED ACROSS API LIFECYCLE

Continuous security enforcement across IDE, CI/CD and at runtime.

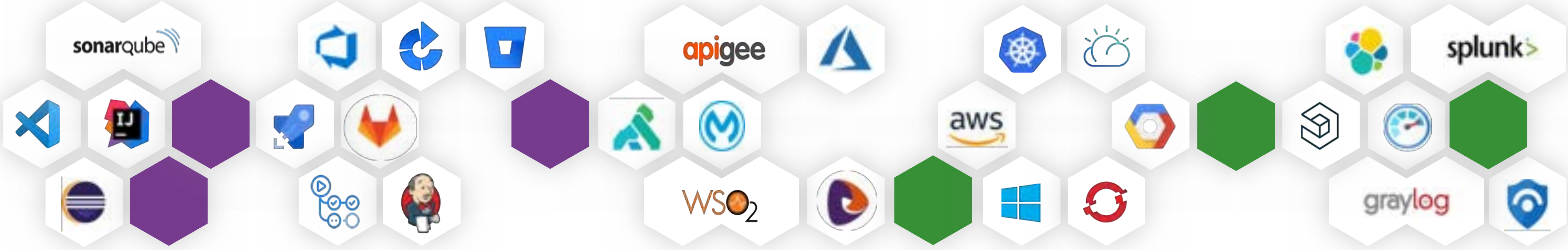
**BENEFITS:** **AUTOMATION - COMPLIANCE - COST SAVINGS - TIME TO MARKET**

# Leverage power of the ecosystem

Complete and continuous protection for APIs throughout the SDLC.
Secure by design and at runtime.

**SHIFT LEFT**
with 42Crunch AUDIT & SCAN
(Design & Develop)

**SHIELD RIGHT**
with 42Crunch FIREWALL
(Deployment Runtime)



IDE

CI/CD

API MANAGEMENT

CLOUD PLATFORM

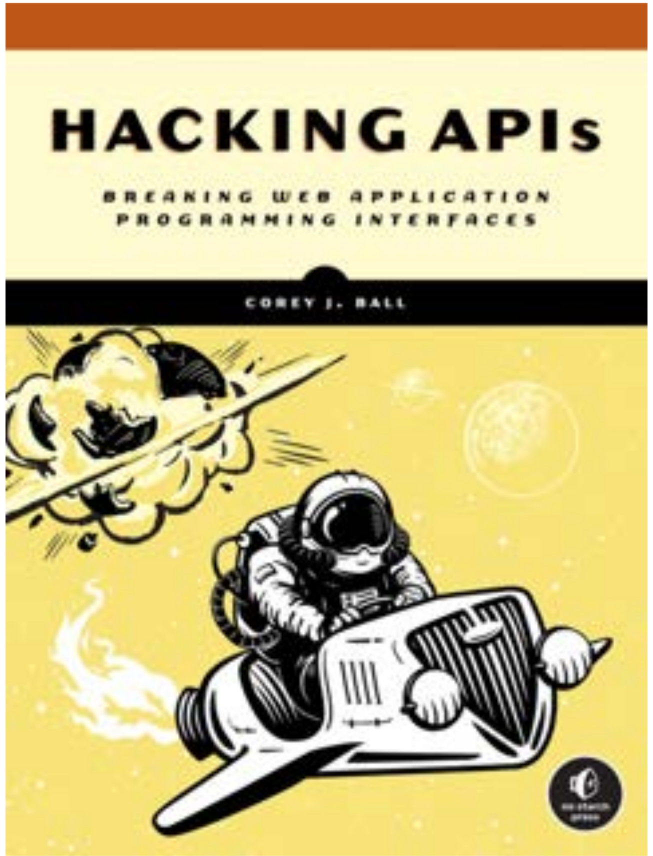
MONITORING SIEM

# Learning more

## APISecurity.io

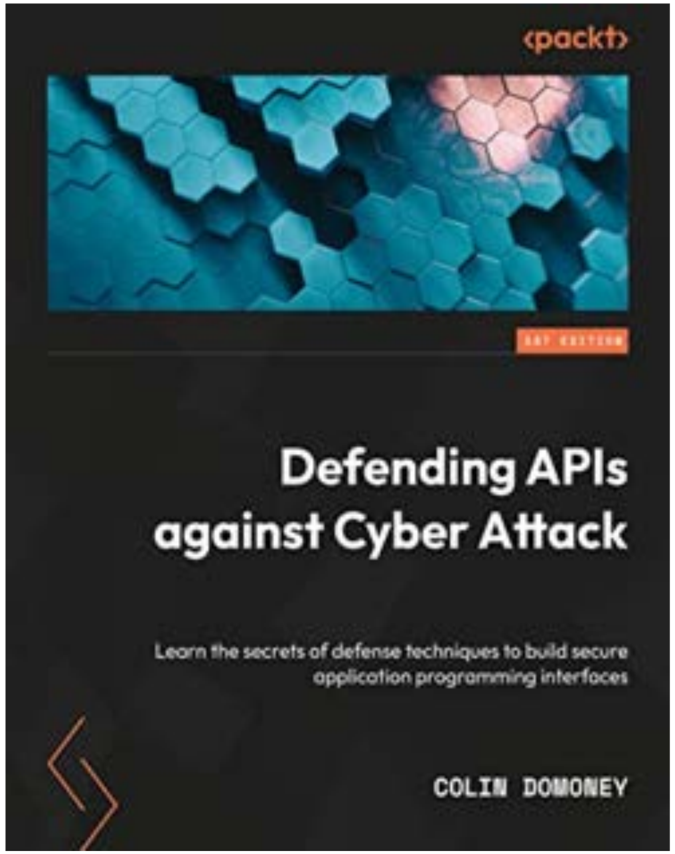
[https://apisecurity.io/](https://apisecurity.io/)

## "Hacking APIs" – Corey Ball



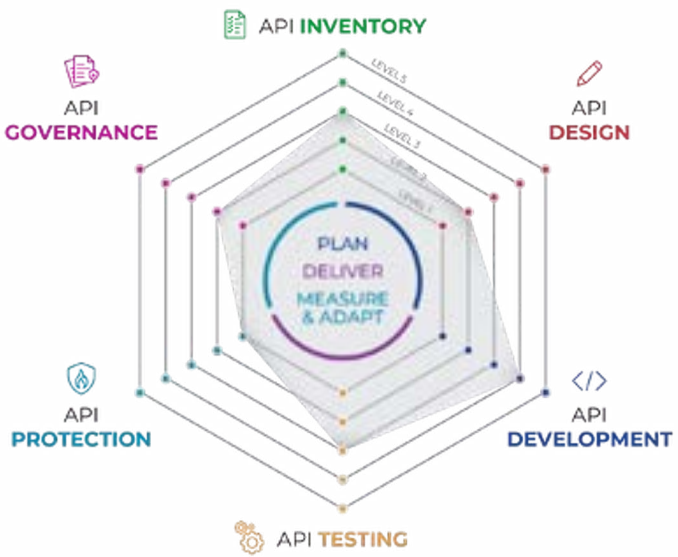
[https://nostarch.com/hacking-apis](https://nostarch.com/hacking-apis)

## "Defending APIs against Cyber Attack" – Colin Domoney


<packt>
1ST EDITION
Defending APIs against Cyber Attack
Learn the secrets of defense techniques to build secure application programming interfaces
COLIN DOMONEY

[https://amzn.to/3fHp8Mz](https://amzn.to/3fHp8Mz) eBook

# API Security: A Blueprint for Success



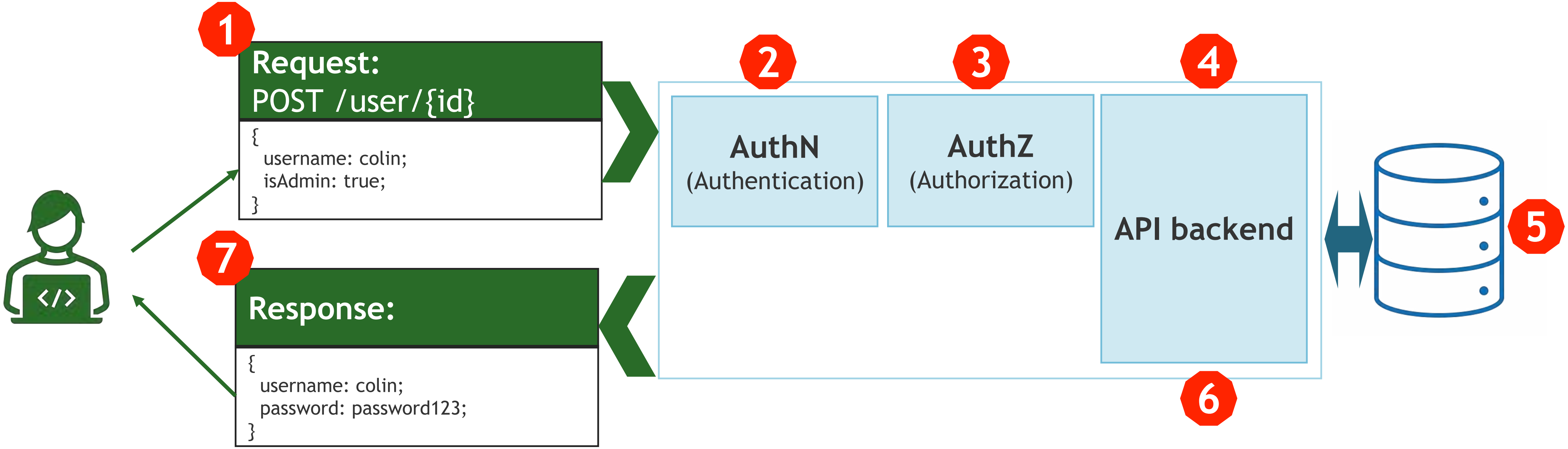
- *Practical Guide on an API Security program.*
- *Map your enterprise's API security posture against 6 key domains.*
- *Champion the case for API Security.*

**Download here:** https://42crunch.com/ebook-api-security-blueprint/

**APIsecurity.io Community Newsletter:** https://apisecurity.io/

# OWASP API Security Top 10

# How does an API work?



1 **Request:**
POST /user/{id}

```
{
  username: colin;
  isAdmin: true;
}
```

2 AuthN
(Authentication)

3 AuthZ
(Authorization)

4 API backend

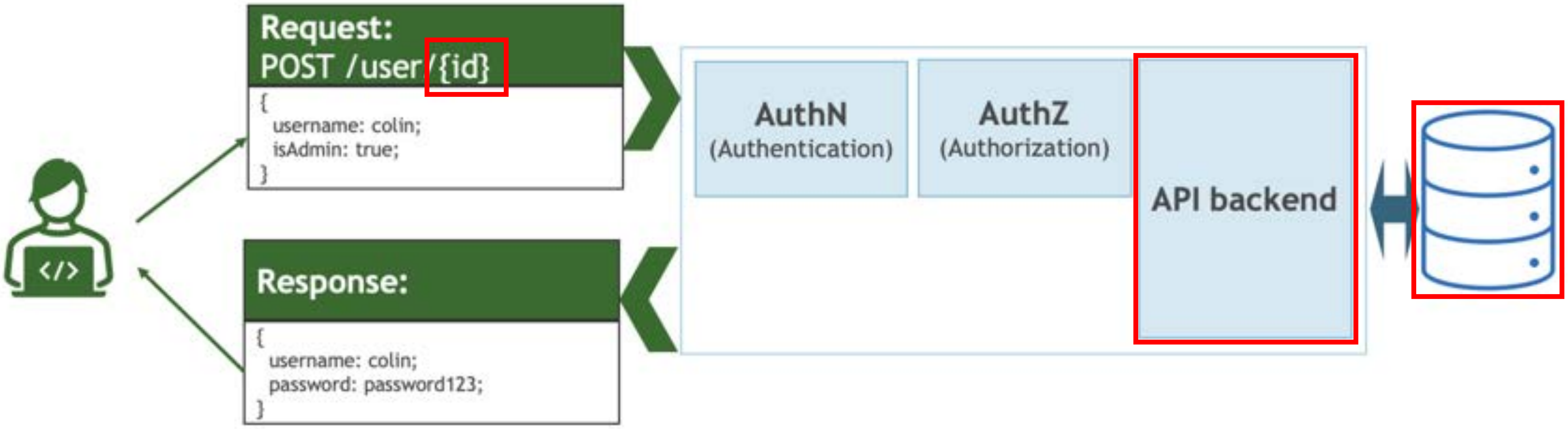
5

6

7 **Response:**

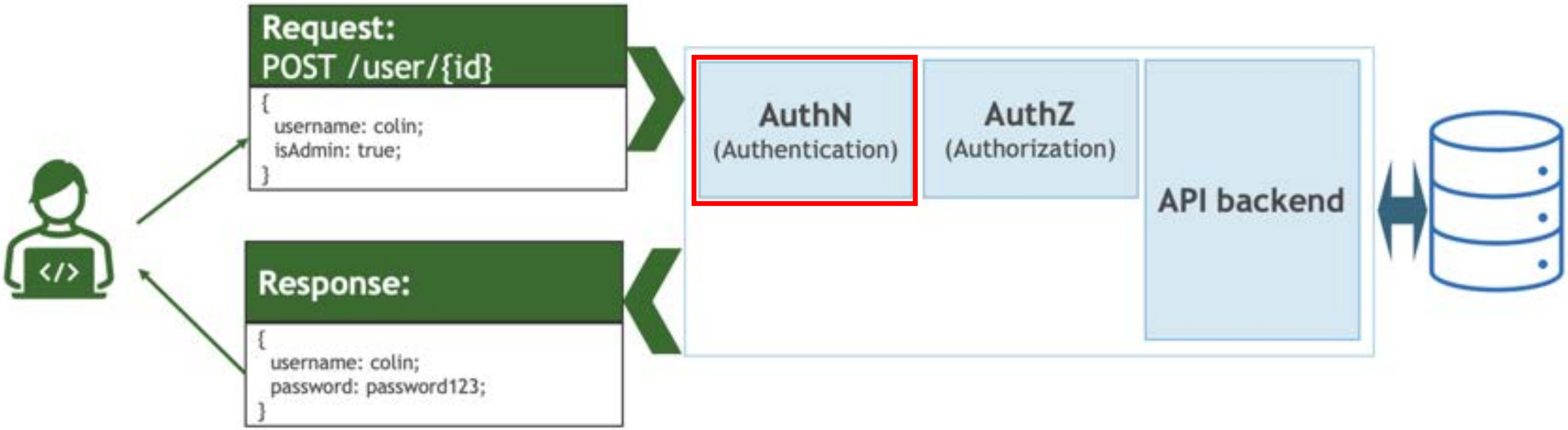
```
{
  username: colin;
  password: password123;
}
```

# API1 — BROKEN OBJECT LEVEL AUTHORIZATION



https://apisecurity.io/encyclopedia/content/owasp/api1-broken-object-level-authorization

# API 2: BROKEN AUTHENTICATION



Request:
POST /user/{id}

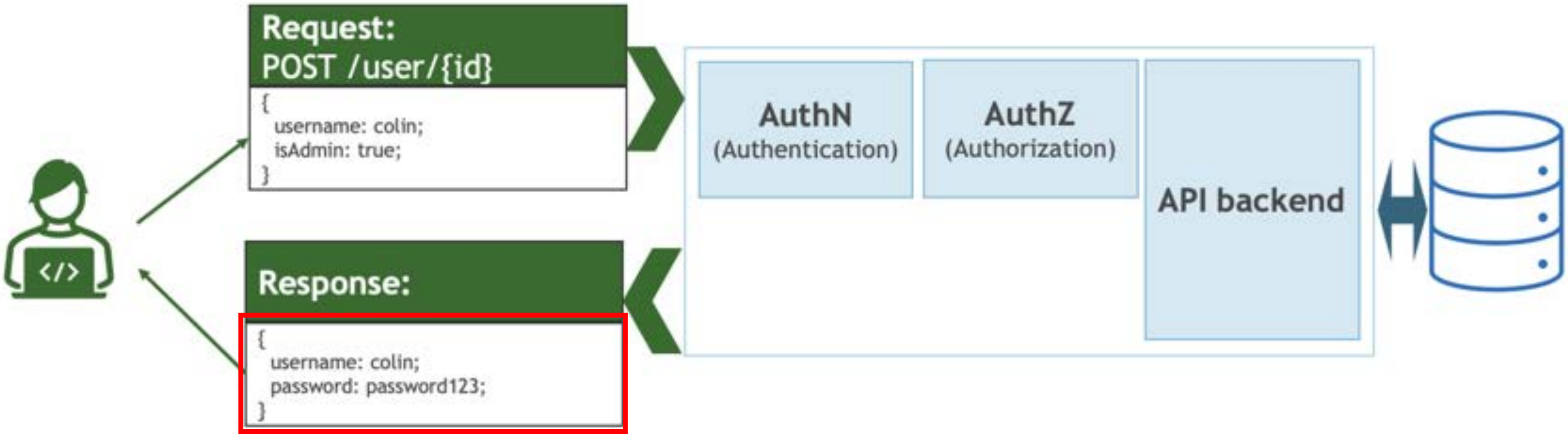
```
{
 username: colin;
 isAdmin: true;
}
```

Response:

```
{
 username: colin;
 password: password123;
}
```

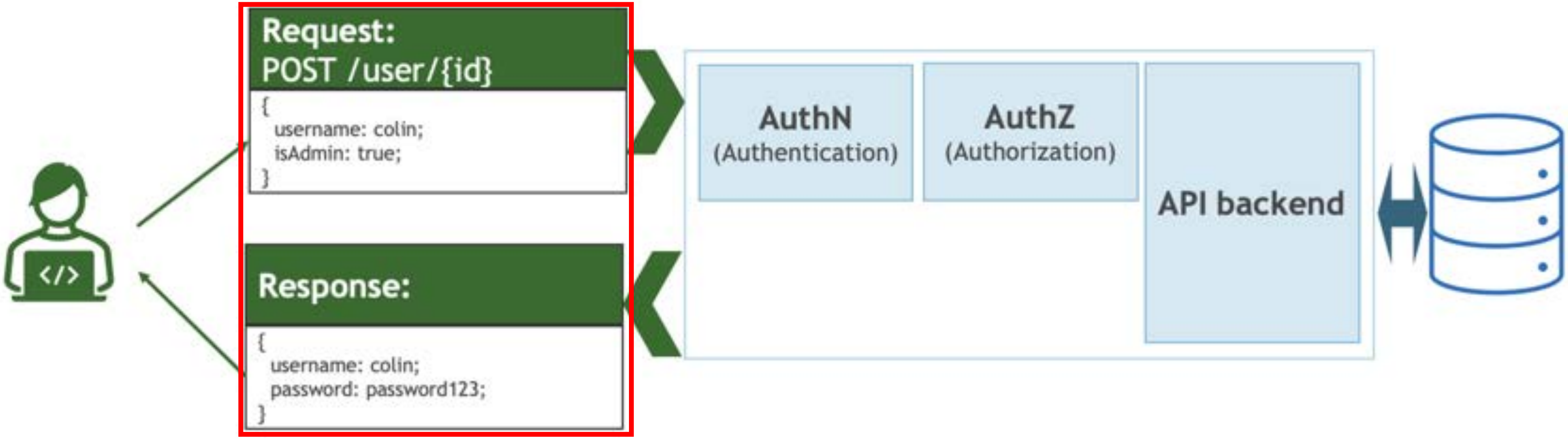
https://apisecurity.io/encyclopedia/content/owasp/api2-broken-authentication

# API 3: DATA/EXCEPTION LEAKAGE



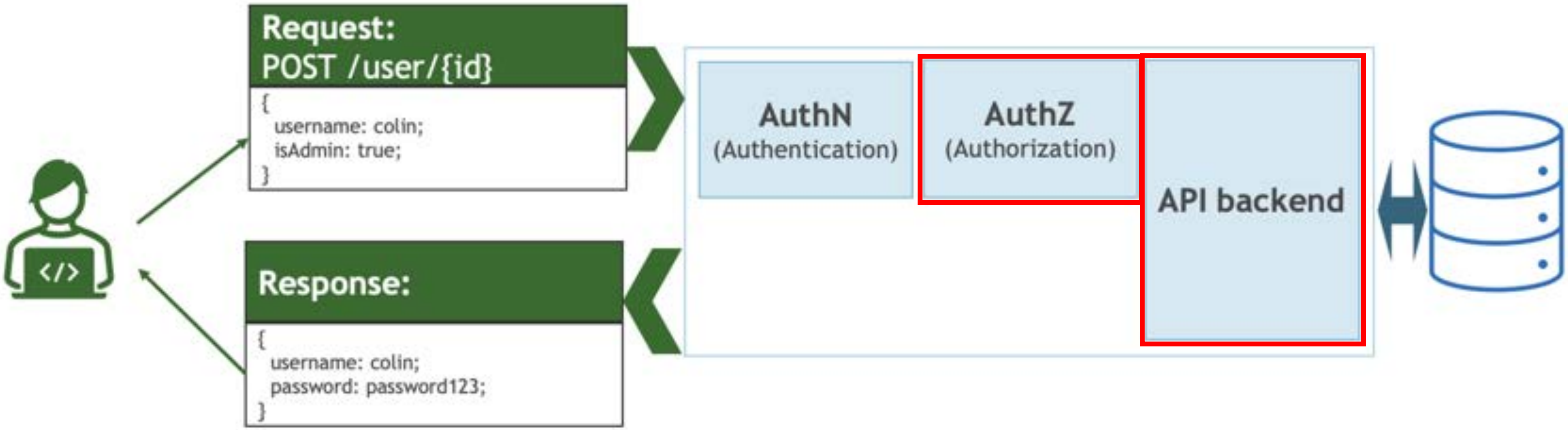
https://apisecurity.io/encyclopedia/content/owasp/api3-excessive-data-exposure

# API 4: RESOURCES PROTECTION/RATE LIMITING



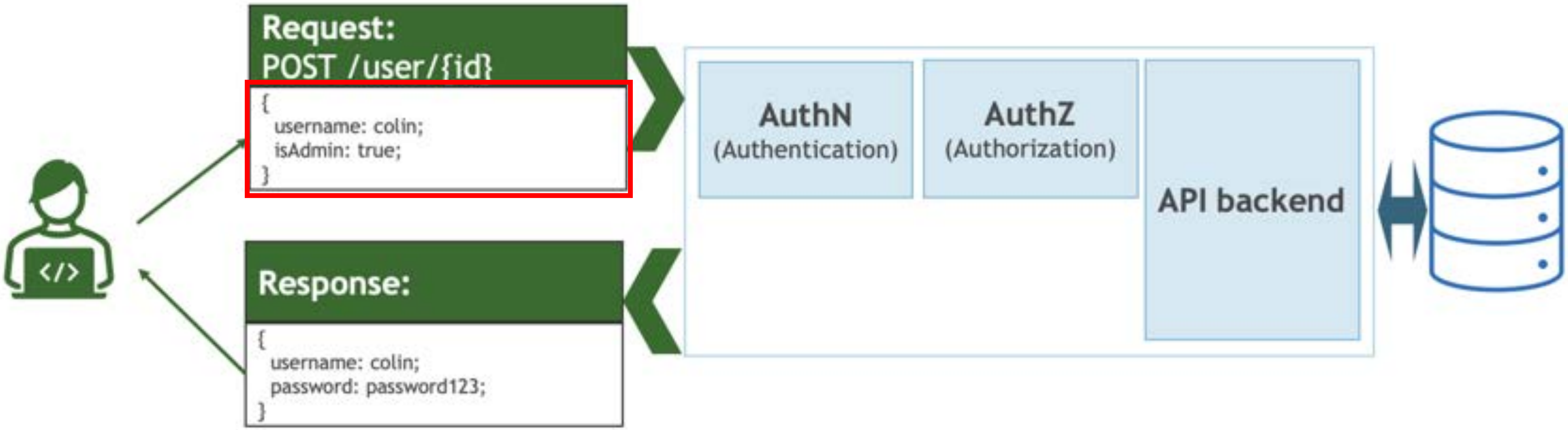
https://apisecurity.io/encyclopedia/content/owasp/api4-lack-of-resources-and-rate-limiting

# API 5: BROKEN FUNCTION LEVEL AUTH



https://apisecurity.io/encyclopedia/content/owasp/api5-broken-function-level-authorization

# API 6: MASS ASSIGNMENT



Request:
POST /user/{id}

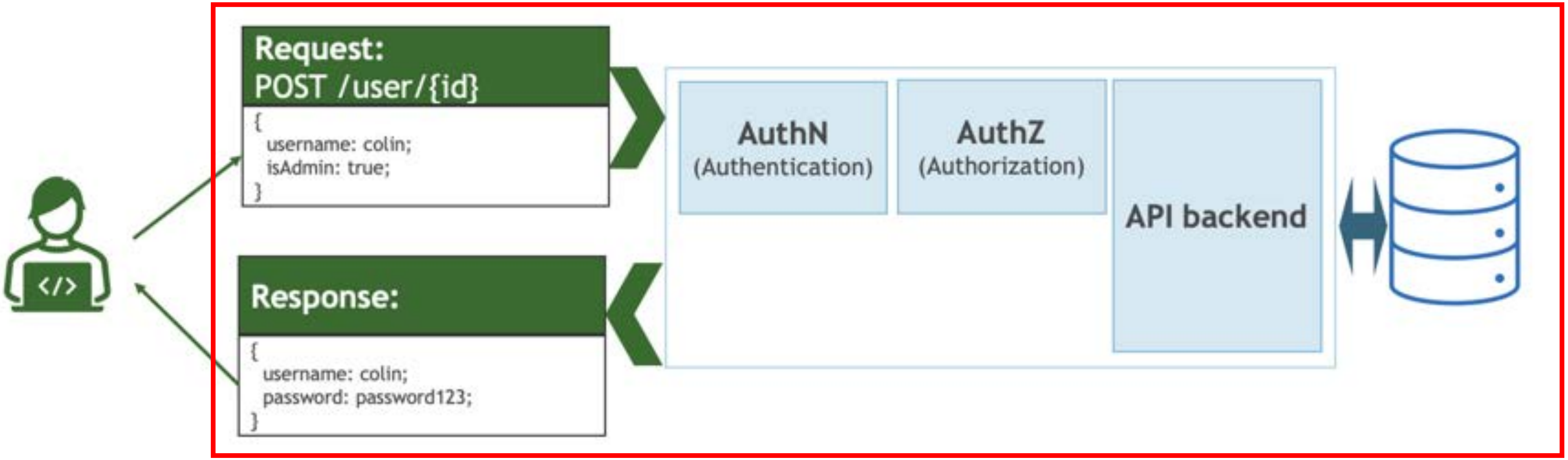
```
{
  username: colin;
  isAdmin: true;
}
```

Response:

```
{
  username: colin;
  password: password123;
}
```

*https://apisecurity.io/encyclopedia/content/owasp/api6-mass-assignment*

# API 7: SECURITY MISCONFIGURATION

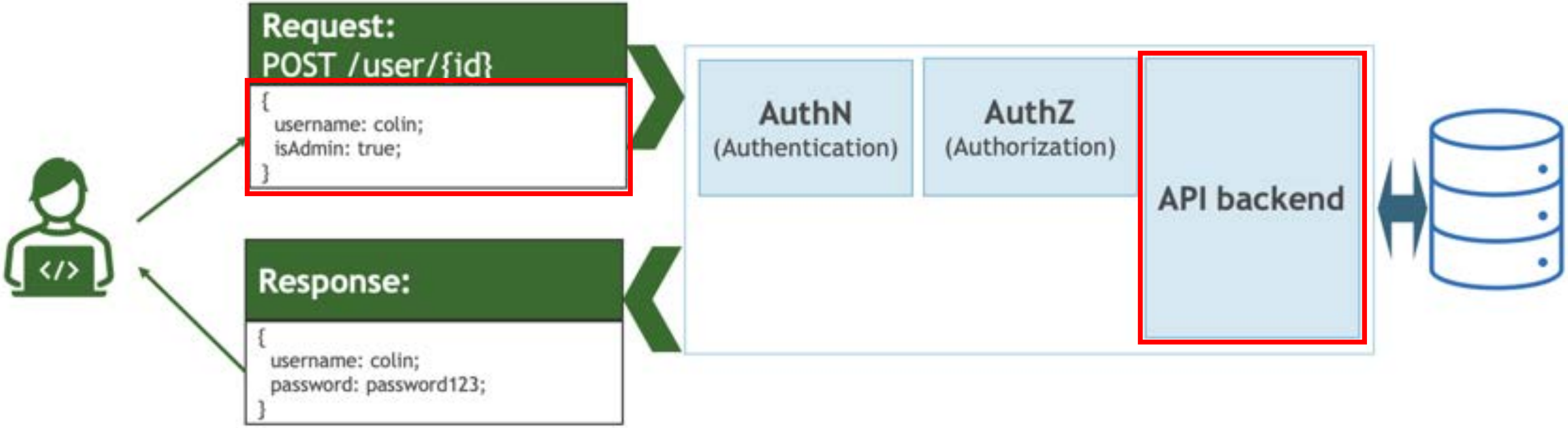


```
{
  username: colin;
  isAdmin: true;
}
```

```
{
  username: colin;
  password: password123;
}
```

*https://apisecurity.io/encyclopedia/content/owasp/api7-security-misconfiguration*

# API 8: INJECTION

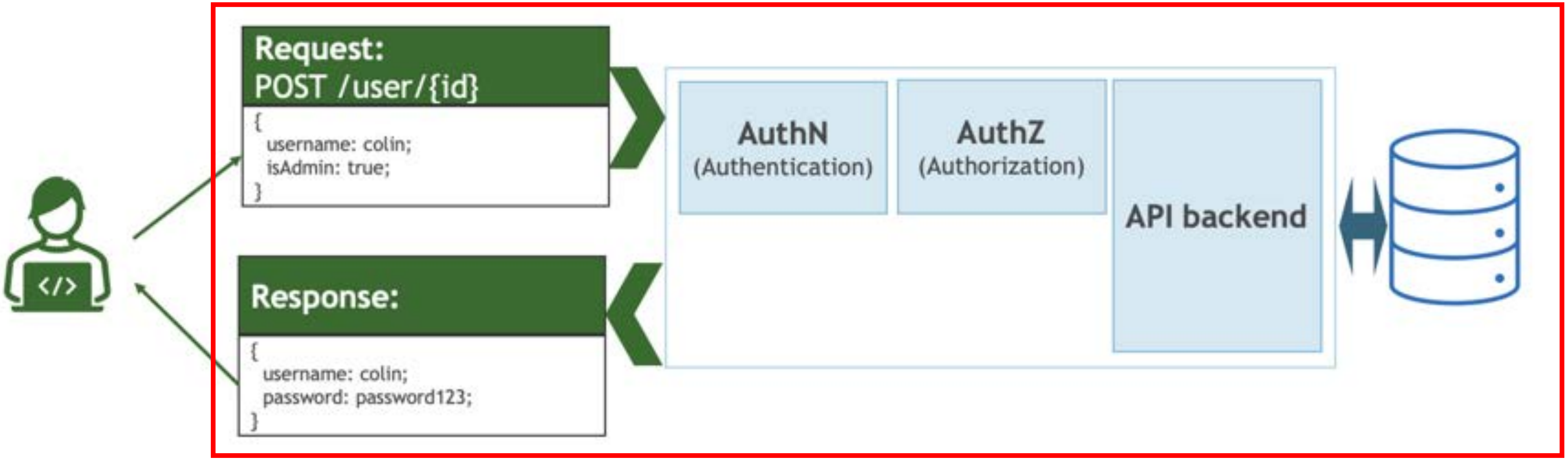


```
{
  username: colin;
  isAdmin: true;
}
```

```
{
  username: colin;
  password: password123;
}
```

https://apisecurity.io/encyclopedia/content/owasp/api8-injection

# API 9: IMPROPER ASSETS MANAGEMENT



Request:
POST /user/{id}

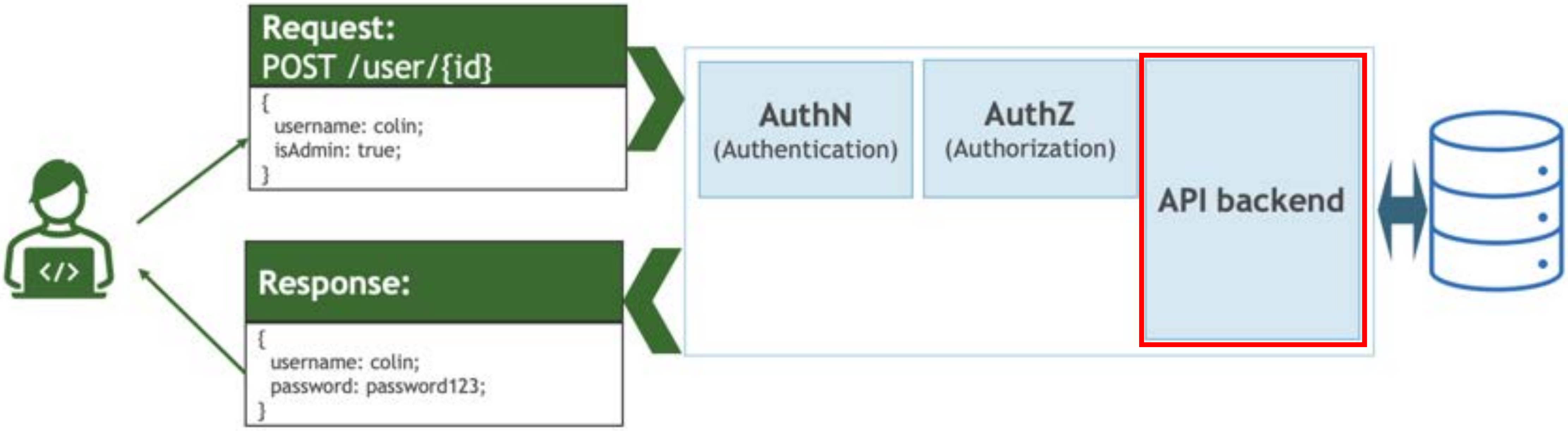
```
{
 username: colin;
 isAdmin: true;
}
```

Response:

```
{
 username: colin;
 password: password123;
}
```

*[https://apisecurity.io/encyclopedia/content/owasp/api9-improper-assets-management](https://apisecurity.io/encyclopedia/content/owasp/api9-improper-assets-management)*

# API 10: LOGGING /MONITORING



https://apisecurity.io/encyclopedia/content/owasp/api10-insufficient-logging-and-monitoring