



# API Security

# API Owner

Escuela de APIs

{api  addicts}



# Isabelle MAUNY

- Field CTO and co-founder @42Crunch
- 42Crunch delivers an API Security Platform
- We are the company behind [apisecurity.io](https://apisecurity.io)
- Working with APIs and integration since 2005!
- French native, in Spain since 2003

# APIs connect the world





Fintech

Stripe, Paypal , Google Pay are APIs



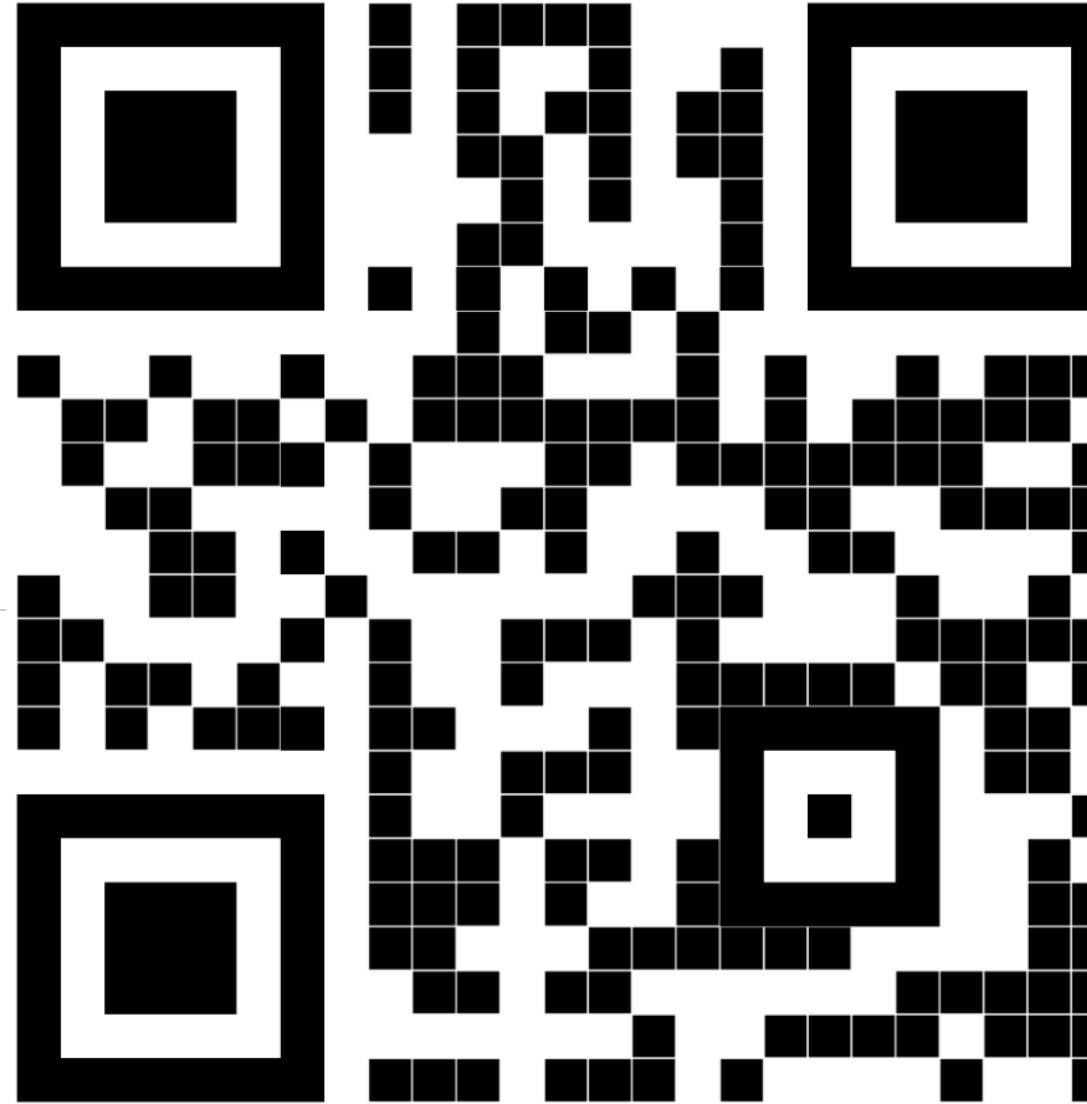
Healthcare

Huge impact during COVID time



IoT

Take control of your smart objects!



<https://apisecurity.io>

18 October, 23

Issue 231: API authentication bypass in Ivanti Sentry, Docker images expose API and keys

5 October, 23

Issue 230: OpenSea API breach, flaw in Atlas VPN, using API fuzzing

21 September, 23

Issue 229: Incidents with DuoLingo and JumpCloud, FastAPI for APIs, and five best practices

7 September, 23

Issue 228: 3rd party API security, OAuth2 step-up deep-dive, shadow and zombie APIs

25 August, 23

Issue 227: GhostToken on Google Cloud, Gartner on zero trust, API authentication

10 August, 23

Issue 226 : Jetpack WordPress plugin has API vulnerability, how to address API security in 2023

26 July, 23

Issue 225 : API security needs a reset, vAPI walkthrough, five stages to attain API security

6 July, 23

Issue 224 : API security is critical in 2023, API contract testing, and Fencer security testing tool

# Nothing New !

Says the wise ChatGPT

when was the first input validation cheat sheet published by OWASP ?

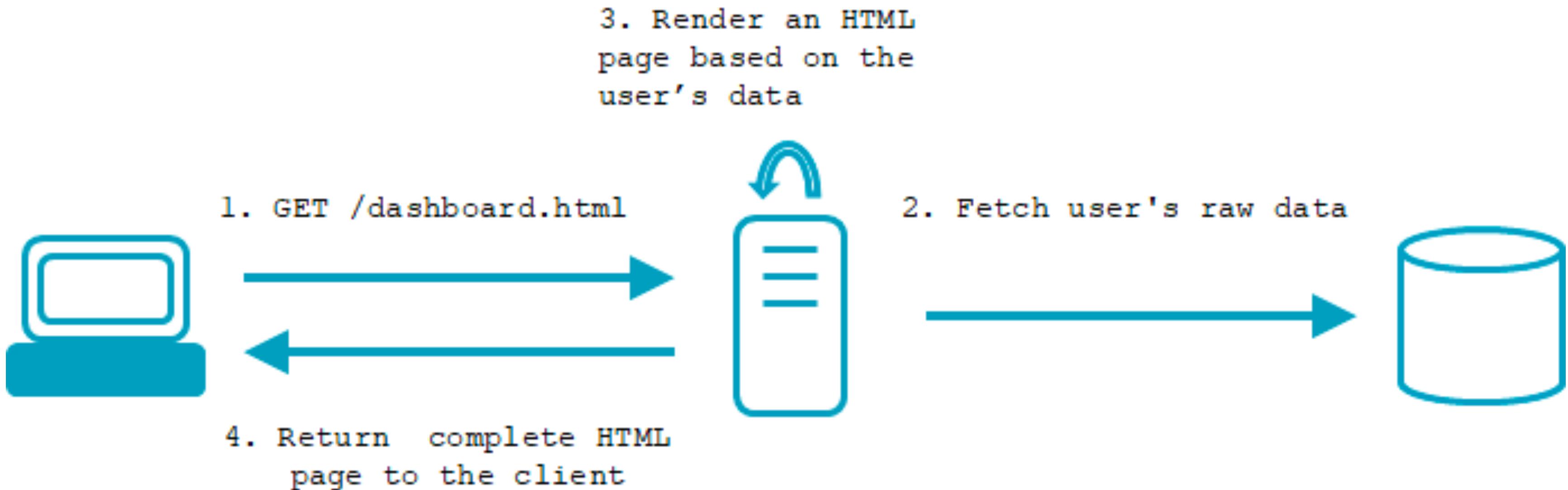
The Open Web Application Security Project (OWASP) is an organization that focuses on improving the security of software. Their "cheat sheets" are concise collections of information about specific security topics, providing guidance on best practices.

As of my last update in January 2022, the OWASP Input Validation Cheat Sheet was first published in 2009. However, OWASP content can be updated or changed over time, so for the most up-to-date information or specific publication dates after 2022, I would recommend checking OWASP's official website or their repository directly.

**WHY ARE APIs  
SUCH A PROBLEM?**

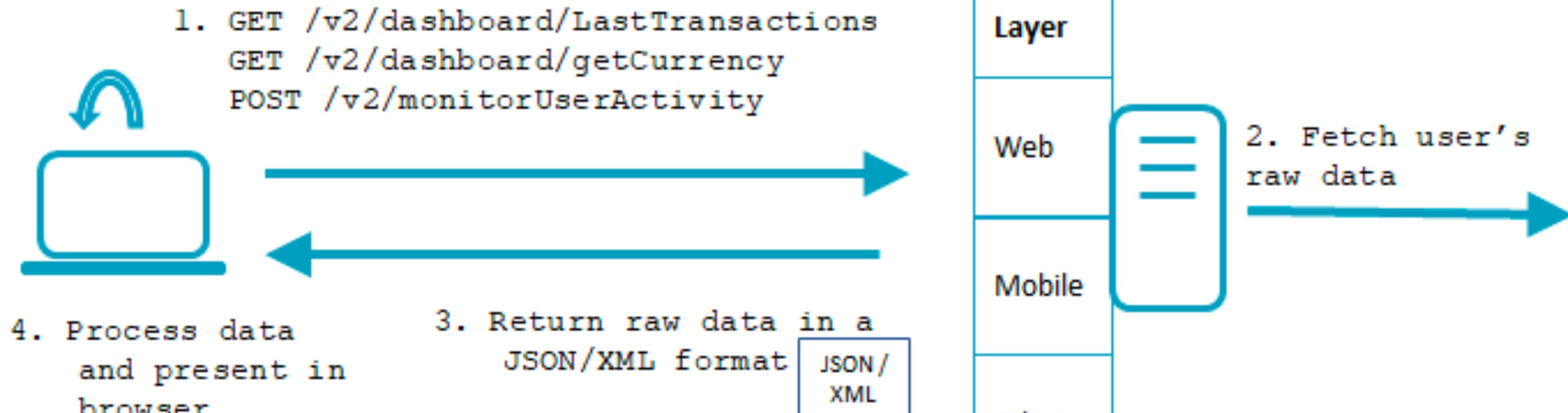


# “Traditional” web architecture



<https://apisecurity.io/encyclopedia/content/owasp/owasp-api-security-top-10.htm>

# API-based API architectures



<https://apisecurity.io/encyclopedia/content/owasp/owasp-api-security-top-10.htm>



APIs are the heart of application development and integration

“APIs are popping up like mushrooms in my data centers”

*From a desperate Fortune 500 CISO*



CommitStrip.com

# Security is still an afterthought!

Delivery pressure leads to untested APIs into the wild.

## Design flaws are critical

---

APIs suffer from **many design flaws**, which are hard, sometimes impossible to fix without a redesign.



# Humans Errors

---

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





# A BIT OF VOCABULARY

# Security Goals Overview

INTEGRITY

Message has not been tampered with

CONFIDENTIALITY

Message can only be seen by target audience

AVAILABILITY

Resistance to attacks, such as Denial-of-service (DOS)

AUTHENTICATION

Identity of the caller is known.

AUTHORIZATION

We can guarantee the caller has proper permissions to access a resource

AUDIT

System has non-perishable trace of all machine/human interactions.

NON-REPUDIATION

There is (legal) proof that the action has taken place.

# SECURITY PRINCIPLES



# API Security is risk based

- Threat Model
- Data Classification
- Actors Identification



# You can't protect what you don't know!

- API Catalog
- API Governance
- API Security Status



# No Trust



# An API on the Internet is a public API



# GUIDING FRAMEWORK



# OWASP API Security Top 10 2023

- API1:2023 Broken Object Level Access Control
- API2:2023 Broken Authentication
- API3:2023 Broken Object Property Level Authorization (**Updated**)
- API4:2023 Unrestricted Resources Consumption
- API5:2023 Broken Function Level Authorization
- API6:2023 Unrestricted Access to Sensitive Business Flows (**New**)
- API7:2023 Server Side Request Forgery (**New**)
- API8:2023 Security Misconfiguration
- API9:2023 Improper Inventory Management
- API10:2023 Unsafe Consumption of APIs (**New**)



BAD PROBLEMS USUALLY OCCUR WHEN MULTIPLE OF THESE ARE COMBINED

# Testing Pixi

- Github project: <https://github.com/isamauny/apisecurity-training>
  - Local build with docker compose
  - Postman Collection
  - API runs at <http://localhost:8090>

# AUTHORIZATION



# Authorization Levels

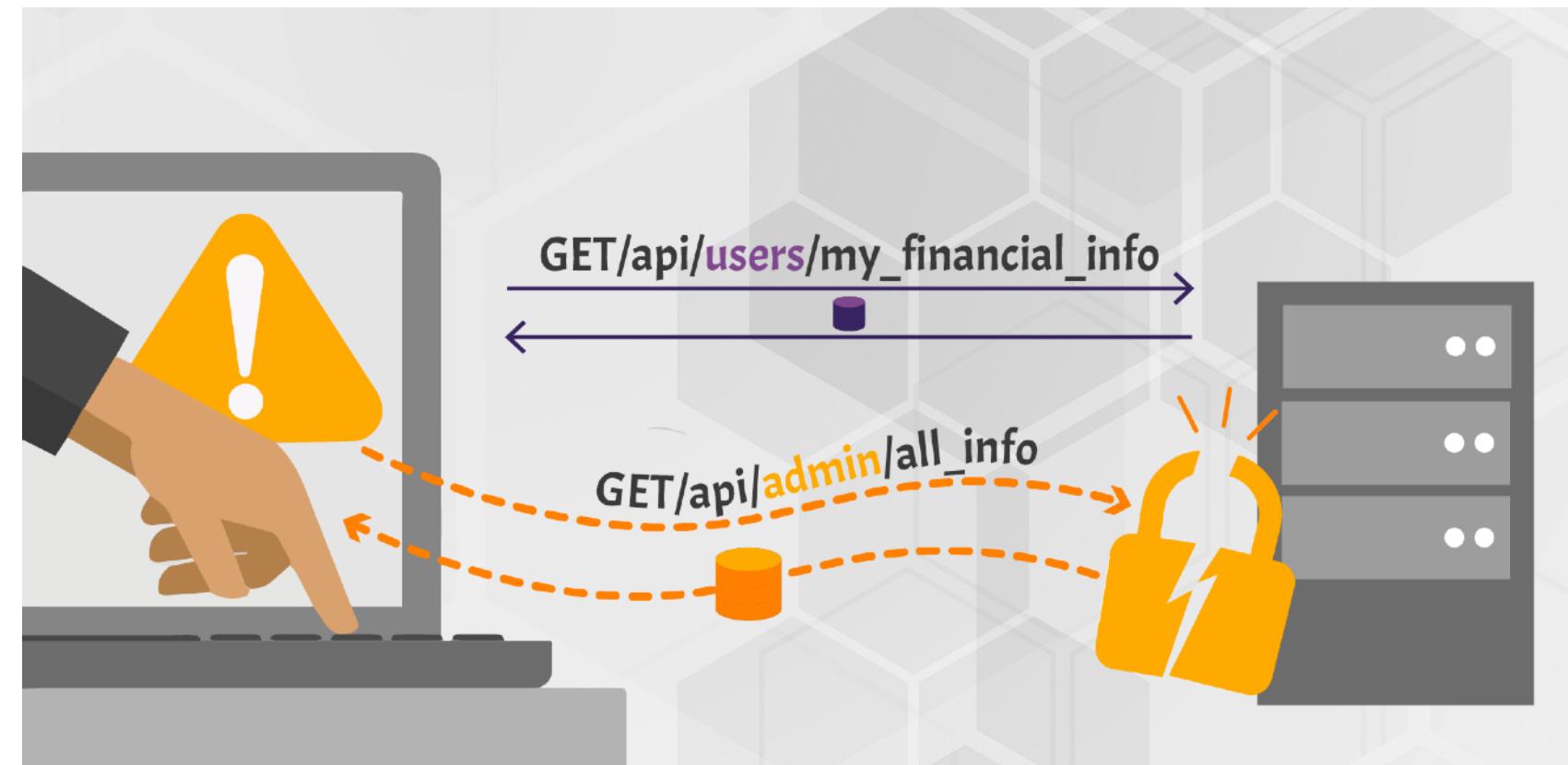
```
"/picture/{id}": { ← API5: BFLA
  "delete": {
    Scan | Try it | Audit ← API1: BOLA
    "parameters": [
      {
        "schema": {
          "type": "string",
          "format": "uuid",
          "pattern": "^[0-9A-Fa-f]{8}-[0-9A-Fa-f]{4}-[0-9A-Fa-f]{4}-[0-9A-Fa-f]{4}-[0-9A-Fa-f]{12}$",
          "minLength": 36,
          "maxLength": 36
        },
        "name": "id",
        "in": "path",
        "required": true
      }
    ],
    "responses": {
      "200": {
        "description": "Picture was deleted successfully",
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "message": {
                  "type": "string",
                  "maxLength": 100
                }
              },
              "required": [
                "message"
              ]
            }
          }
        }
      }
    }
  }
},
```

The diagram illustrates the structure of a nested API definition. The outermost part defines a resource at `/picture/{id}` with a `delete` method. This method includes a parameter named `id` which is required and has a specific schema (a UUID). The response to a successful deletion (`200`) is a JSON object containing a single message field with a maximum length of 100 characters. Annotations are present: a yellow arrow labeled **API5: BFLA** points to the path `/picture/{id}`; a red arrow labeled **API1: BOLA** points to the `delete` method; and a purple arrow labeled **API 3: BOPLA** points to the `application/json` schema within the `200` response.

# API5: Broken Function Level Authorization aka *BFLA*

Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws.

By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.



# Voting App

- The Attack
  - Data breach using leaked admin userid/passwords
- Core Issues
  - Admin endpoint left open with no authentication
  - Allows to retrieve all systems users, including the password (in clear).
  - **Endpoint was hardcoded in application source code.**

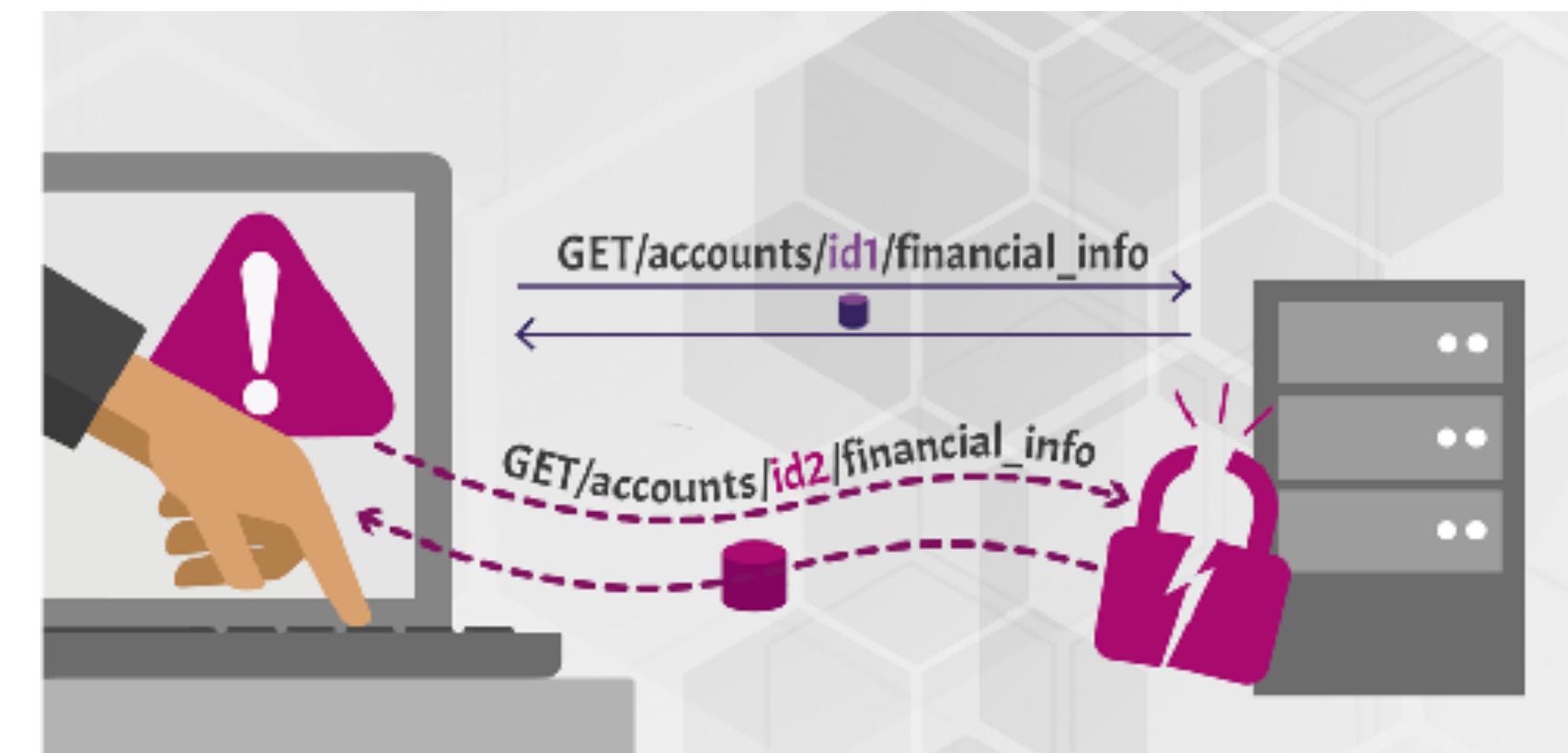


```
curl -X GET SERVER_URL+"/get-admin-users?"
```

```
90 "usersCount": 100,
91 "adminId": null,
92 "allowExcelUpload": false,
93 "active": true,
94 "dataEncrypted": false
95 },
96 {
97 "oid": 86,
98 "name": "הילכוד לכתבת",
99 "password": "1234567890",
100 "email": "",
101 "campaignName": "כנתת 23 ליכת",
102 "maxUsers": 10000,
103 "businessId": "",
104 "businessName": "",
105 "expirationDate": "19-01-2024",
106 "mainOfficeAddress": "",
107 "cost": null,
108 "purchasedSms": 710000,
109 "remainingSms": 340788,
110 "phone": "052-1234567",
111 "creationTime": "2020-01-19",
112 "usersCount": 100,
113 "adminId": null,
114 "allowExcelUpload": false,
115 "active": true,
116 "dataEncrypted": false
117 }
118 }
```

# API1: Broken Object Level Access Control aka *BOLA*

APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.



# Large Telco

- The Attack
  - Attackers have been able to gather details from millions of users through their phone number.
- Core Issues
  - No rate limiting
  - Guessable/known IDs (phone numbers)
  - No authorization validation



# DEMO TIME



# Addressing BFLA Issues

- Design properly your authorization policies and test them !
  - OAuth scopes can be used to restrict the access to operations.
- Avoid mixing admin and non-admin operations in the same API
  - Easy to discover via dictionary attacks
- Restrict access to admin APIs, for example:
  - by Mutual TLS
  - by (validated!) IP Range
  - **Do not rely on the client to do that!**

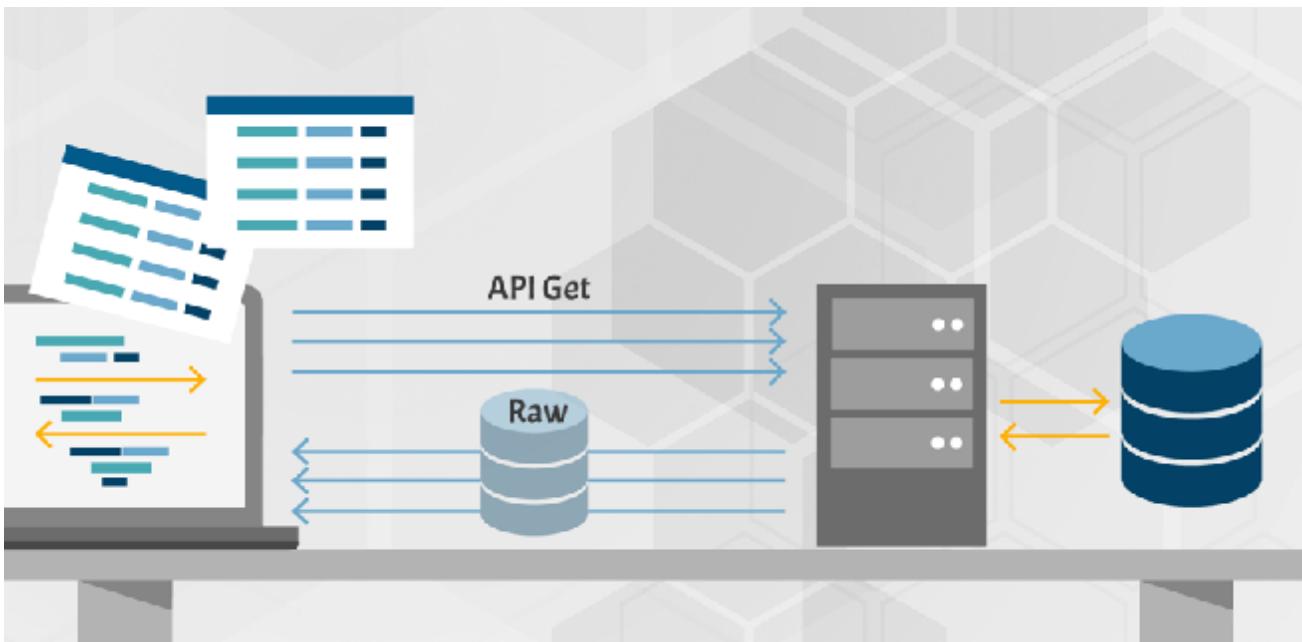
# Addressing BOLA Issues

- **True fix : Fine-grained authorisation to resources in every controller layer**
  - Decision engine is required, preferably externally to the code.
  - Need to be able to enforce something like : *Jean is allowed to access account 123, but cannot edit or delete it.*
- Potential *mitigation* approaches
  - Avoid using IDs when possible: GET <https://myapis.com/phone/me>
  - Avoid iterative IDs (123, 124, 125...)
  - Avoid exposing internal IDs through your APIs (for example internal database IDs)
  - Mitigate potential data scrapping by putting **rate limiting** in place (and alerting!)
- **OAuth scopes are not helping here, as they limit access to a function and not to a resource.**

# DATA AUTHORIZATION

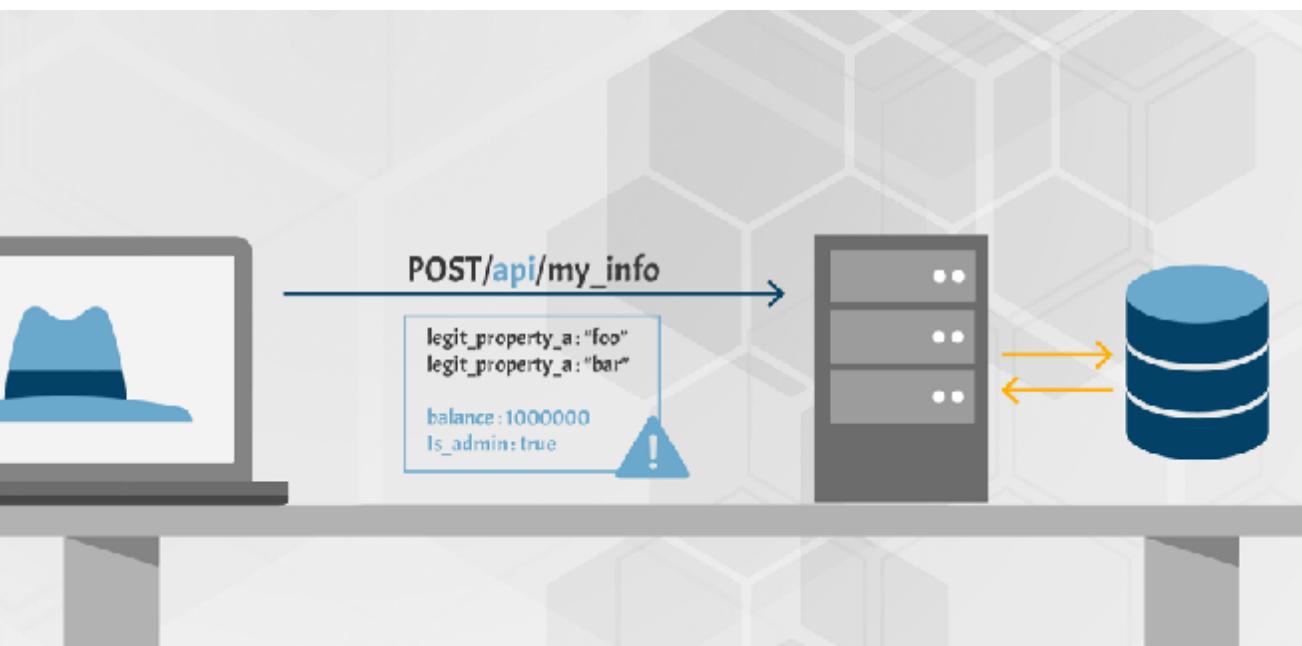


# API3: Broken Object Property Level Authorization



## Input Data

Binding client provided data (e.g., JSON) to data models, without proper properties filtering based on an allowlist, usually leads to Mass Assignment. Either guessing objects properties, exploring other API endpoints, reading the documentation, or providing additional object properties in request payloads, allows attackers to modify object properties they are not supposed to.



## Output Data

Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user.

# 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, whitelist 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**
  - Constraint the amount of data returned by the API
- 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

# DEMO TIME



# JWTs transport data too!

- Can leak data
- Can be prone to injections (example: [kid sql injection](#))
- Recommended best practice:
  - \* Use opaque tokens for external consumption
  - \* Use JWTs for internal consumption

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7Il9pZCI6ODUsImVtYWlsIjoiY3VzdG9tZXJAcGl4aS5jb20iLCJwYXNzd29yZCI6ImhlbGxvcGl4aSIsIm5hbWUiOiJjb3N0ZXhwbgFuYXRpb24iLCJwaWMiOiJodHRwczovL3MzMzLmFtYXpvbmF3cy5jb20vdWlmYWNlcyc9mYWNlcyc90d210dGVyL3NoYW5lSXhELzEyOC5qcGciLCJhY2NvdW50X2JhbGFuY2Ui0jEwMDAsImlzX2FkbWluIjpmYWxzZSwiYWxsX3BpY3R1cmVzIjpbXX0sIm1hdCI6MTYwMzIxNjIwOX0.DjgTBCev5Kq_DpvBwfKva3K3rLCs4r9hN17S-hh6qMI
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT: DATA

```
{  
  "user": {  
    "_id": 85,  
    "email": "customer@pixi.com",  
    "password": "hellopixi",  
    "name": "costexplanation",  
    "pic":  
      "https://s3.amazonaws.com/uifaces/faces/twitter/shaneIx  
      D/128.jpg",  
    "account_balance": 1000,  
    "is_admin": false,  
    "all_pictures": []  
  },  
  "iat": 1603216209  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

# AUTHENTICATION



# API2: Broken Authentication

Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising system's ability to identify the client/user, compromises API security overall.



# Social Network(s)

- Several examples of:
  - No Authentication at all
  - Badly protected auth endpoints (password reset especially)
  - No proper rate limiting



# Choosing an Authentication Method

- Build a Threat model to decide what is best for your API
- Choose authentication method balancing **risk and ease of use**:
  - Who are the consumers ?
  - What is the data/operations sensitivity ?
- **Do not use API keys or bearer tokens as proof of identity**
  - Remember a Bearer token is like a hotel key, once you have it, you can open the door. There is no way to validate who you are.

# Which options are available ?

Type	Format	Potential Issues	Advantages
Basic Auth	b64 encoded User/password	<ul style="list-style-type: none"><li>• Goes in clear (encoded) over the network</li><li>• Deprecated.</li></ul>	<ul style="list-style-type: none"><li>• Backward compatibility</li></ul>
API Key	Long string of characters	<ul style="list-style-type: none"><li>• Travels over network</li><li>• Prone to leaks (Github, hardcoded in apps etc.)</li><li>• Expiration management</li></ul>	<ul style="list-style-type: none"><li>• Simple to use</li><li>• Opaque</li></ul>
OAuth Access Token	<ul style="list-style-type: none"><li>• Long string of characters</li><li>• JWT</li></ul>	<ul style="list-style-type: none"><li>• Used for authentication</li><li>• Complexity</li><li>• Carries potentially sensitive data</li></ul>	<ul style="list-style-type: none"><li>• Delegated authorization</li><li>• Reduced security risks</li><li>• Expires</li></ul>
OpenID Connect	<ul style="list-style-type: none"><li>• JWT</li></ul>	<ul style="list-style-type: none"><li>• Complexity</li></ul>	<ul style="list-style-type: none"><li>• True Authentication</li><li>• Session/Logout management</li><li>• Reduced Security Risks</li></ul>
Signed Requests	Digital signature in header/body	Complex for API consumers	<ul style="list-style-type: none"><li>• Integrity of information</li><li>• API/Signing keys are not travelling over network.</li></ul>

Your most sensitive endpoints are  
**authentication and password reset**  
endpoints.

# RESOURCES ABUSE



# Rate/Resources Limiting (API 4)

Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user. Not only can this impact the API server performance, leading to Denial of Service (DoS), but also leaves the door open to authentication flaws such as brute force.



# Rate Limiting Recommendations

- Adjust rate limiting to risk : by API Key/access token/user identity/fingerprint
- IPs are very easy to create/forge
- If you are going to use IP, validate it first (reputation, TOR, etc.)
- **Bad example:** 200 attempts per minute per IP for password reset

*"In a real attack scenario, the attacker needs 5000 IPs to hack an account. It sounds big but that's actually easy if you use a cloud service provider like Amazon or Google.*

*It would cost around 150 dollars to perform the complete attack of one million codes"*

# Further DOS Protection recommendations

- Constrain the data (positive security model) : max value, max size for arrays, string patterns, max size for a file
- Additionally:
  - Set limits in container based environments (memory, CPU, file descriptors, etc.)
  - Good for DOS protection AND for the bill at the end of the month....

**Weak Authentication**

**+**

**Bad Rate Limiting**

**=**

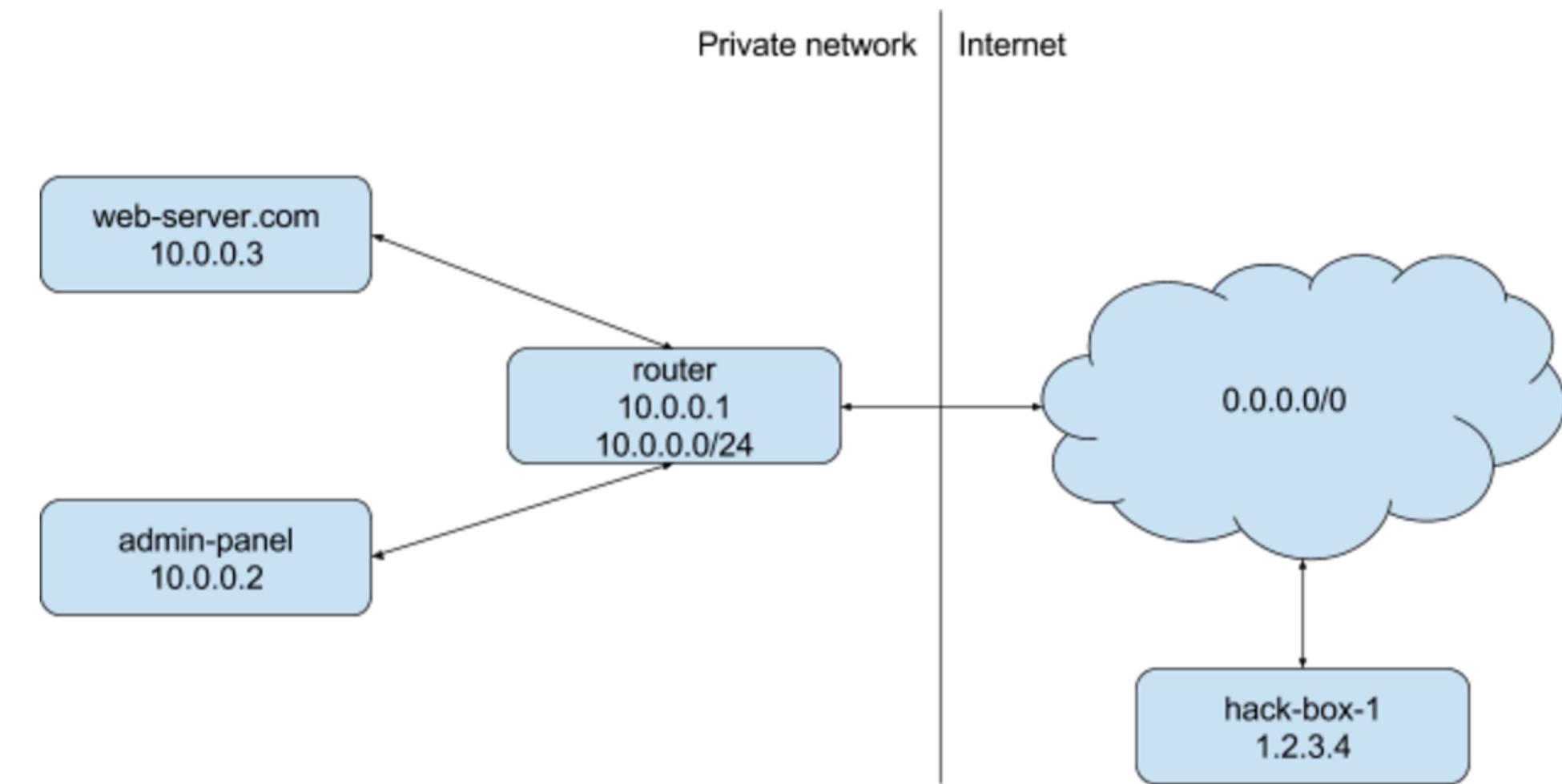




# SSRF

# SSRF

Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to **induce the server-side application to make requests to an unintended location.**



# How to protect yourself from SSRF!

- Getting Started
  - AllowList of websites you allow access to
  - Network policies
  - DenyList of protocols (ftp:// , file:// )
  - Disable/control redirection
- Did you know ?
  - Blacklisting <http://localhost> or <http://127.0.0.1> is not enough to prevent access to local machine
  - Try : ping 127.0 or ping 2130706433 on your machine or create a DNS entry pointing to localhost..
- Check references for more data on how attackers can circumvent prevention measures!

# API6:2023 - Unrestricted Access to Sensitive Business Flows

- Using an API in a way in which it was not designed or abusing the underlying business flow or logic
- In a single word - BOTS
- Becoming a very common attack vector



# How to mitigate bots ?

First a bit of Threat Modelling:

- Identify the business flows that might harm the business if they are excessively used.

Mitigation Approaches:

- **Device fingerprinting:** denying service to unexpected client devices (e.g. headless browsers) tends to make threat actors use more sophisticated solutions, thus more costly for them
- **Human detection:** using either a captcha or more advanced biometric solutions (e.g. typing patterns)
- **Non-human patterns:** analyse the user flow to detect non-human patterns (e.g. the user accessed the "add to cart" and "complete purchase" functions in less than one second)
- Consider blocking IP addresses of Tor exit nodes and well-known proxies

# API10:2023 - Unsafe Consumption of APIs

- Developers tend to trust data received from third-party APIs more than user input
- APIs are part of a supply chain and need to be secured at every point
  - When evaluating service providers, assess their API security posture
  - Ensure all API interactions happen over a secure communication channel (TLS)
  - Always validate and properly sanitize data received from integrated APIs before using it.



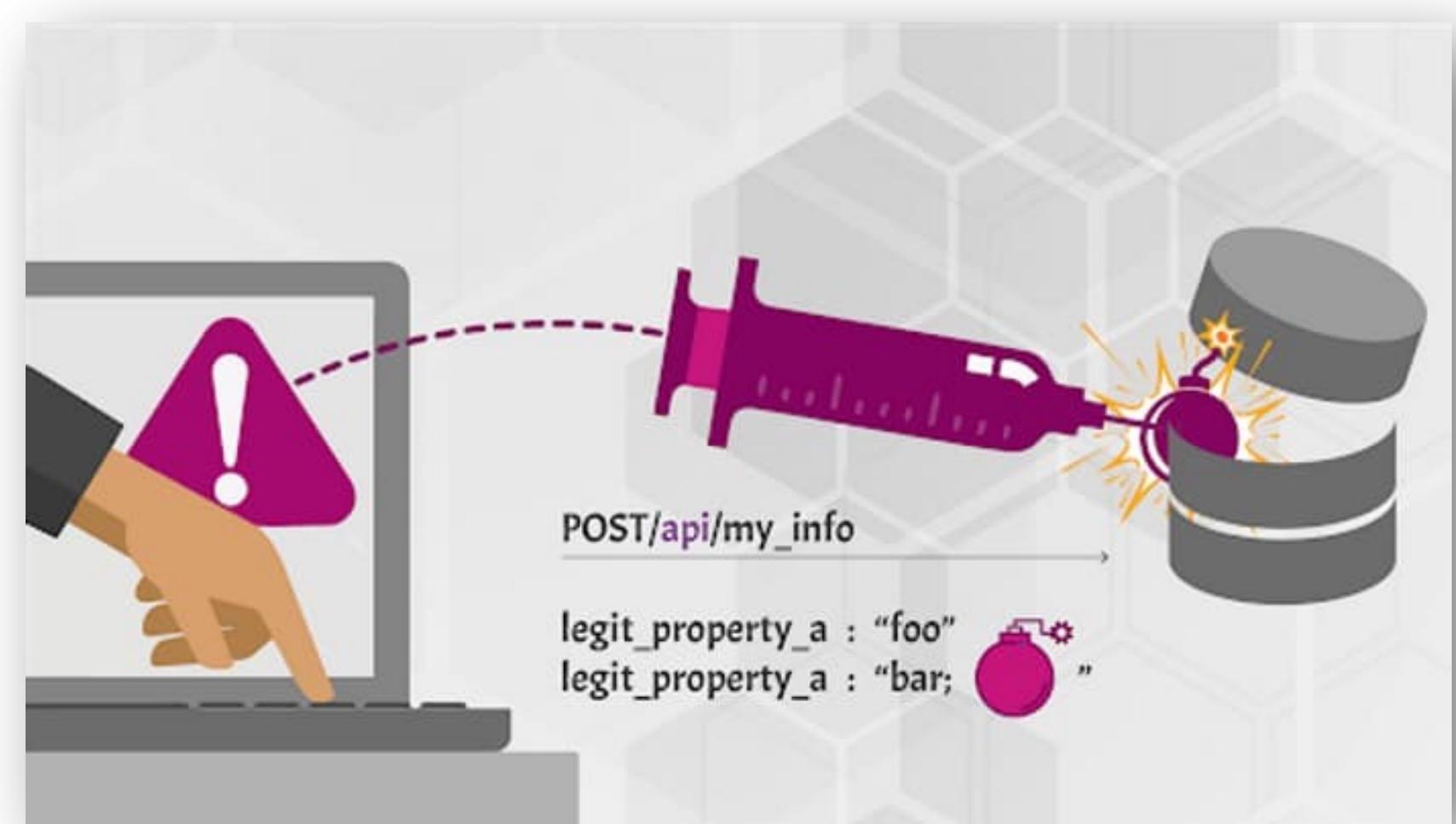
# What about injections?

- Injection attacks affect all software systems, not only APIs
- Injection attacks are still **very prevalent** and affect APIs frequently
- Remediation advice as per OWASP Top 10 still applies (nothing API specific)
- 

Lack of injection in 2023 API10 #86

Open cyn8 opened this issue on Mar 8 · 19 comments

<https://github.com/OWASP/API-Security/issues/86>



<https://apisecurity.io/owasp-api-security-top-10/api8-injection/>

# LOGGING



# Secure Logging

- **Goals:**
  - Forensics
  - Non-repudiation
- **Need event logs** for anything unusual
  - Rejected requests (auth issues, authorization issues, data validation, application errors).
  - Critical information needs to be logged at the lowest logging level (i.e. not the debug level)
- **Need to record:** what happened, when, who was the caller, where (app/api details, machine name, pod name, etc.)
- **Recommendations:**
  - Log early - Adding logs once code is written is a nightmare...
  - Invest in a shared framework / custom library that everyone uses and which implements those best practices - will make logging easier and coherent

# Secure Logging

- Careful with the data we log:
  - No PII
  - No tokens/API Keys
  - No Encryption keys
  - Anything sensitive for your business
- Sensitive data can be :
  - **Redacted** (replaced with xxxxx for example)
  - **Hashed** : very useful for tokens/IPs for traceability
  - **Encrypted**
- Logs file may need to be signed for non-repudiation purposes

More at : [https://cheatsheetseries.owasp.org/cheatsheets/Logging\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html)

# Tokens/Keys passed as query param!

- Anything in query param end logs in a log somewhere
- ...Then pushed to a central log manager...
- ...Now it's visible in dashboards!
- Always use headers to pass sensitive information (or body in a POST)

More at: <https://www.fullcontact.com/blog/2016/04/29/never-put-secrets-urls-query-parameters/>



# IMPLEMENT PROACTIVE SECURITY

An API **must never blindly trust**  
anything it receives from the client.

And that includes

---

- Request payloads
- Headers
- JSON Web Tokens
- IP addresses (X-Forwarded-For)
- And everything else :)



**STAY  
PARANOID  
AND  
TRUST  
NO ONE**

## Follow the “Hacky Path”

---

For each happy path test, you should have **10 Hacky Path** tests

And if you automate them, even better!

The way to protect yourself from Human Errors

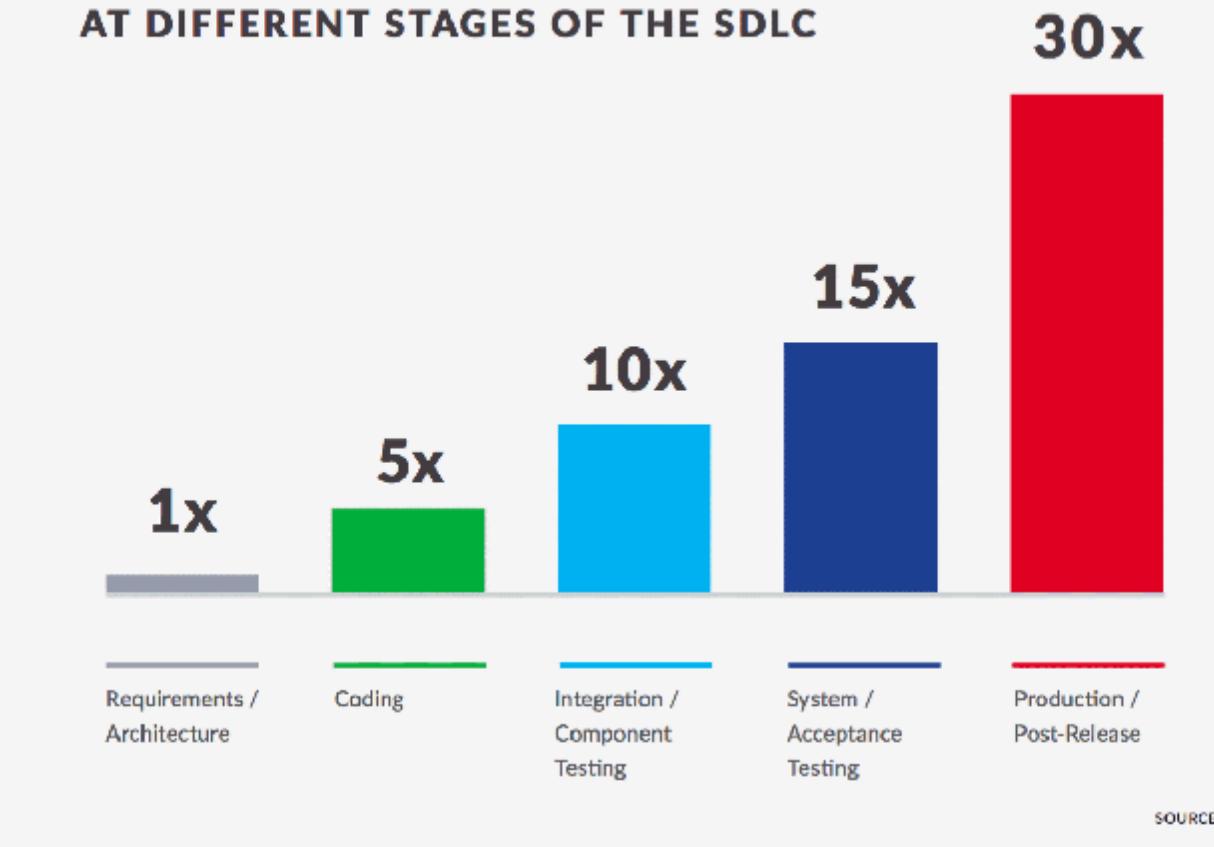


# CALL TO ACTION!

- ▶ Use API Top 10 as framework for design and testing
- ▶ Start worrying about API Security at design time
  - ✓ A vulnerability discovered at production time costs up to 30x more to solve
- ▶ Hack yourselves leveraging API contracts
  - ✓ For each functional test, create 10 negative tests
  - ✓ Hammer your APIs with bad data, bad tokens, bad users
- ▶ Automate Security
  - ✓ Inject Security into DevOps practices and don't rely on manual testing of APIs.
  - ✓ Only solution to scale and have avoid human errors

<https://www.helpnetsecurity.com/2020/05/20/devops-software-development-teams/>

THE RELATIVE COST OF FIXING A FLAW  
AT DIFFERENT STAGES OF THE SDLC

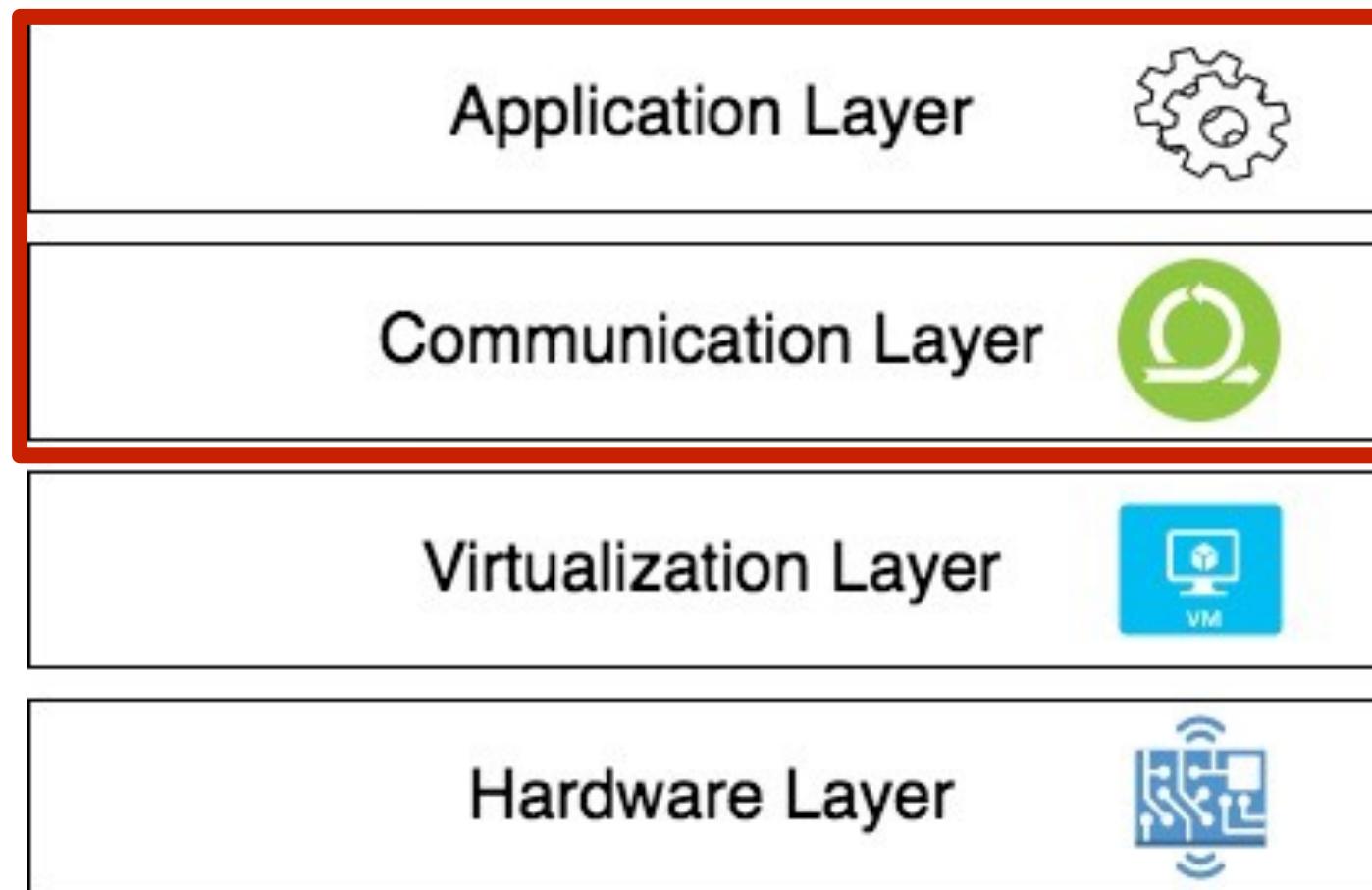


*"I think security, in most cases, is not a single person's specialization. Security must be a practice of every member of the team from the frontend developer to the system administrator (also non tech roles)."*

From: [Gitlab DevSecOps report](#) - 2021

# ARCHITECTURE AND TOOLS

# API Security Applies at Multiple Levels



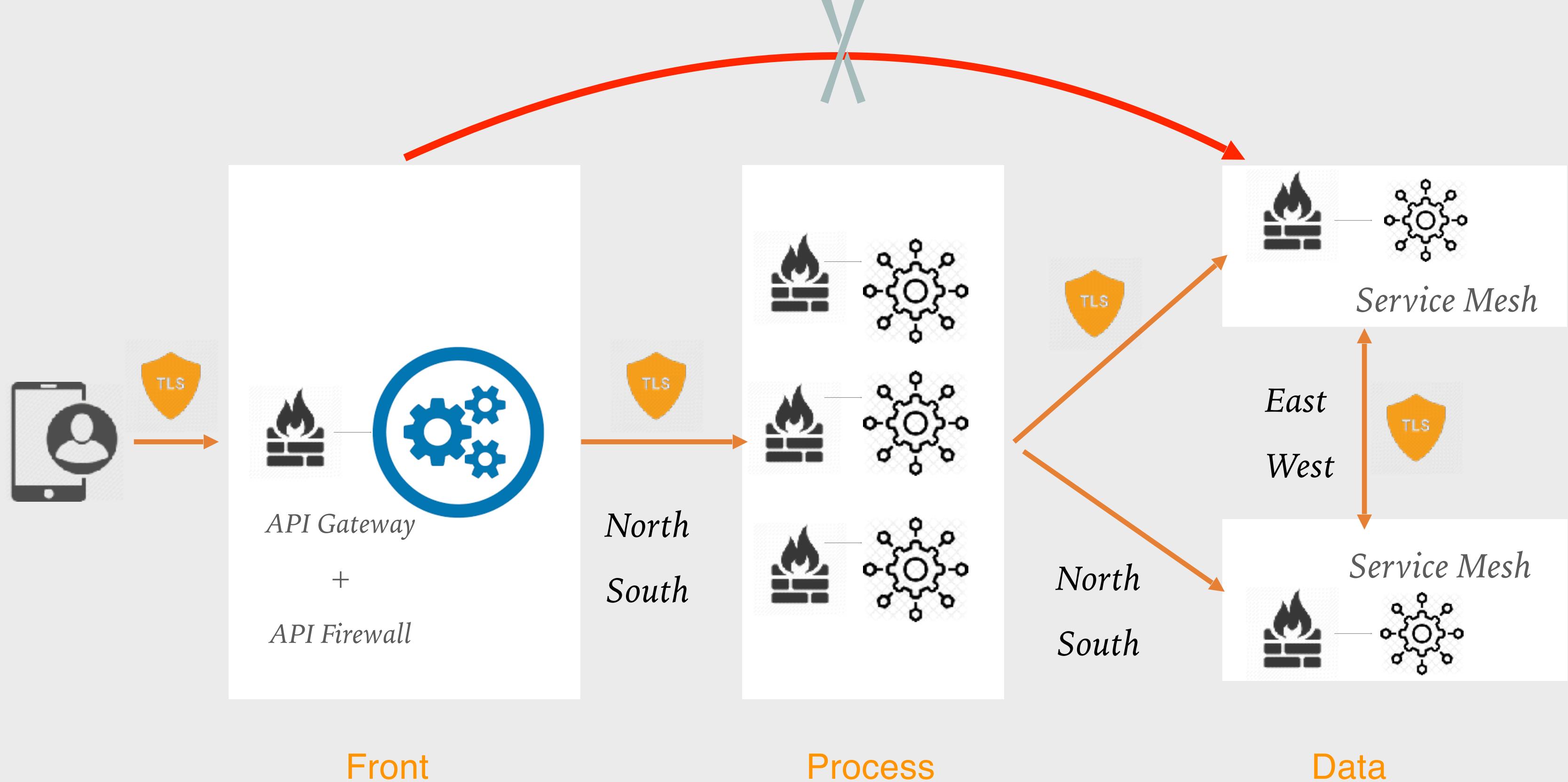
*App level security (Auth, Azn, libs, code, images, data)*

*Intra-services communication (Auth, Azn, TLS)*

*Hypervisor, Kubernetes*

*OS / Network / Physical Access*

# DEPLOYMENT ARCHITECTURE



# References

- **Input Validation**
  - [https://cheatsheetseries.owasp.org/cheatsheets/Input\\_Validation\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html)
- **OWASP REST Security Cheat Sheet**
  - [https://www.owasp.org/index.php/REST\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/REST_Security_Cheat_Sheet)
- **Transport Layer Security Cheat Sheet**
  - [https://www.owasp.org/index.php/HTTP\\_Strict\\_Transport\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet)
  - [https://www.owasp.org/index.php/Transport\\_Layer\\_Protection\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet)
- **HTML5 Security Cheat Sheet**
  - [https://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet#Local\\_Storage](https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet#Local_Storage)
- **Mitigate OWASP Top10 with API Management**
  - <https://learn.microsoft.com/en-us/azure/api-management/mitigate-owasp-api-threats>
- **42Crunch webinars with Jim Manico on SSRF/CSRF :**
  - <https://42crunch.com/defending-apis-with-jim-manico-episode-1/>

# DATOS DE CONTACTO

+34 91 764 79 82

[contacta@apiaddicts.org](mailto:contacta@apiaddicts.org)

[www.apiaddicts.org](http://www.apiaddicts.org)



Facebook  
[ApiAddicts](#)



Linkedin  
[API Addicts](#)



Twitter  
[@APIAddicts](#)



Meet Up  
[API Addicts](#)



Youtube  
[API Addicts](#)