API Security
API Owner

Escuela de APIs

{api addicts}

Isabelle MAUNY

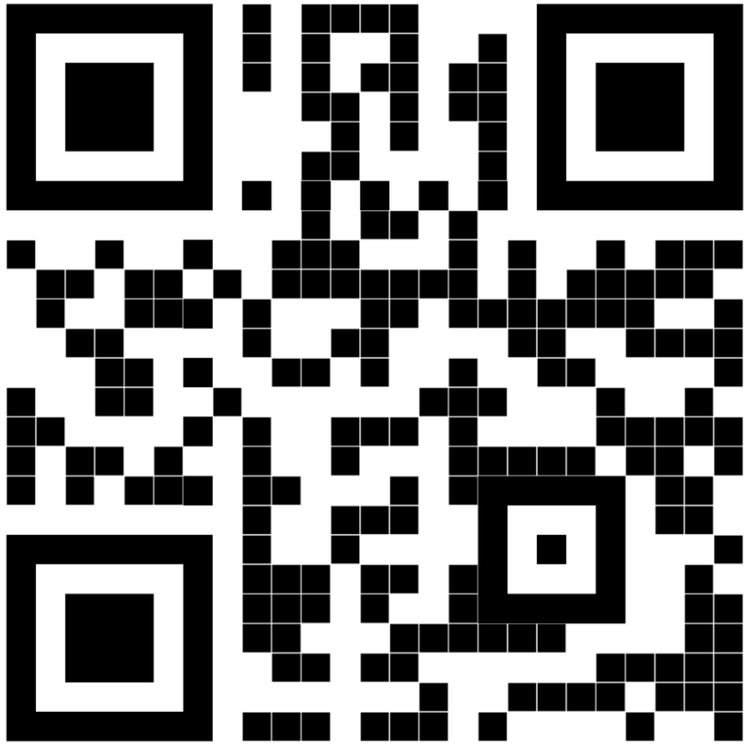
- ➔ *Field CTO @WSO2*
- ➔ *Founder CTO @42Crunch*
- ➔ *Working with APIs and integration since 2005!*
- ➔ *French native, in Spain since 2003*

➔ *isabelle@apis.coach*

{api addicts}

What are our customers saying?

API Security is a universal issue.



https://apisecurity.io

21 November, 24

Issue 259: API flaw exposes 4 million WordPress sites, API error handling bugs, a case for API First

7 November, 24

Issue 258: API governance at Vodafone, OpenAPI updates, APIs with OWASP vulnerabilities

24 October, 24

Issue 257: Internet Archive under attack, API Gateways insecure by default, OWASP injection attacks

10 October, 24

Issue 256: Privilege escalation bugs in Kia vehicles, Cisco and Gov APIs, NIST's new rules for password security

26 September, 24

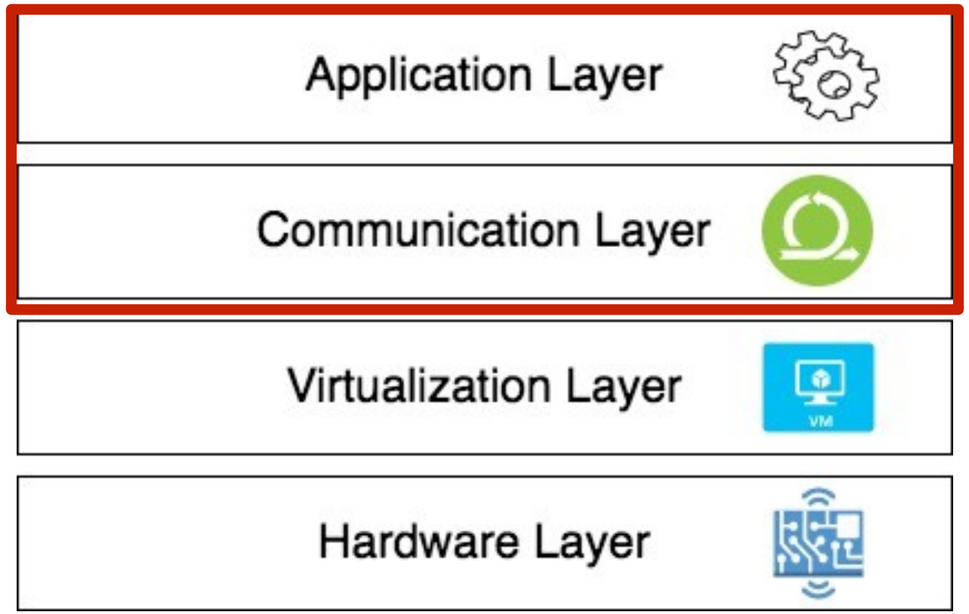
Issue 255: Versa Director API flaw, Feeld BOLA vulnerabilities, logic flaw risks aircraft disaster

12 September, 24

Issue 254: WhatsApp and IBM WebMethods vulnerabilities, 3rd-party API and LLM risks, API access controls

SO, WHY ARE APIS SUCH A PROBLEM?

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

Nothing New ?

Says the wise ChatGPT

why is API Security different from web security

API security and web security differ because they address distinct technologies and threats, even though they may seem closely related. Here are key distinctions:

1. Communication Methods:

- **API Security:** APIs (Application Programming Interfaces) allow different software systems to communicate with each other, often using protocols like HTTP, but they exchange structured data like JSON or XML. APIs are typically used for machine-to-machine communication.
- **Web Security:** Web security is focused on protecting user interactions with web applications. These applications typically involve browsers (human interaction) that send HTTP requests and receive HTML responses.

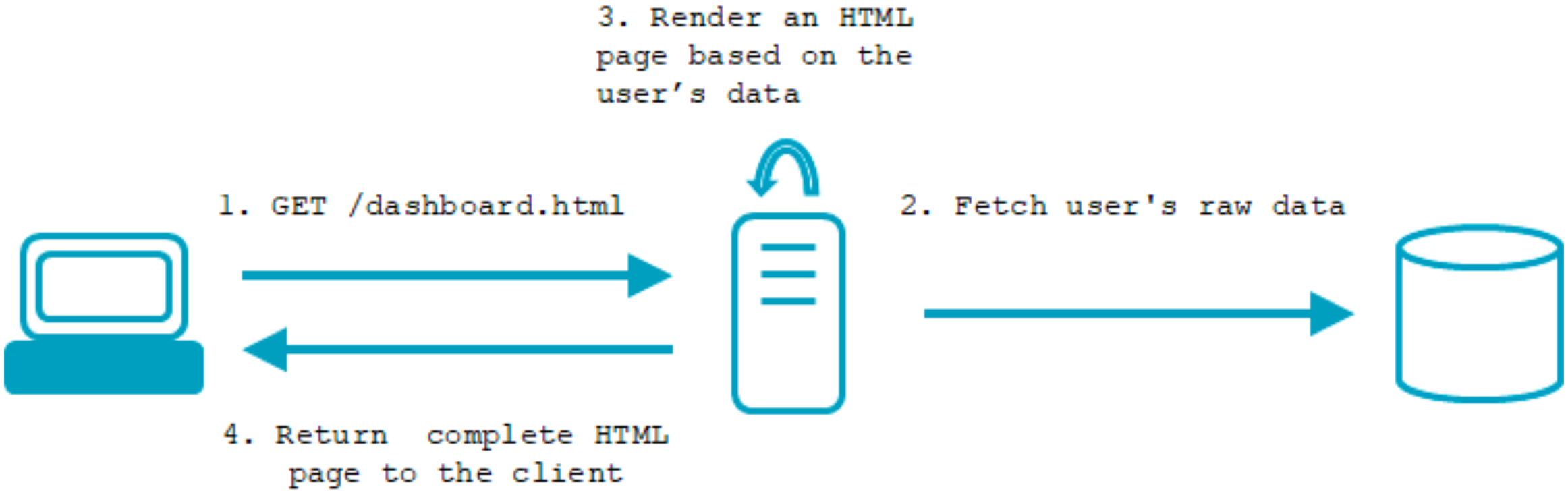
2. Access Control and Authentication:

- **API Security:** API calls often require authentication through tokens (e.g., OAuth, JWT), and APIs may expose sensitive functionalities or data directly. API access must be carefully controlled by rate-limiting, monitoring, and proper permission management.
- **Web Security:** Web security often focuses on session management, CSRF tokens, and cookie-based authentication. Access control in web apps is more user-centric, ensuring that users access only the pages and actions they are authorized to.

3. Data Exposure:

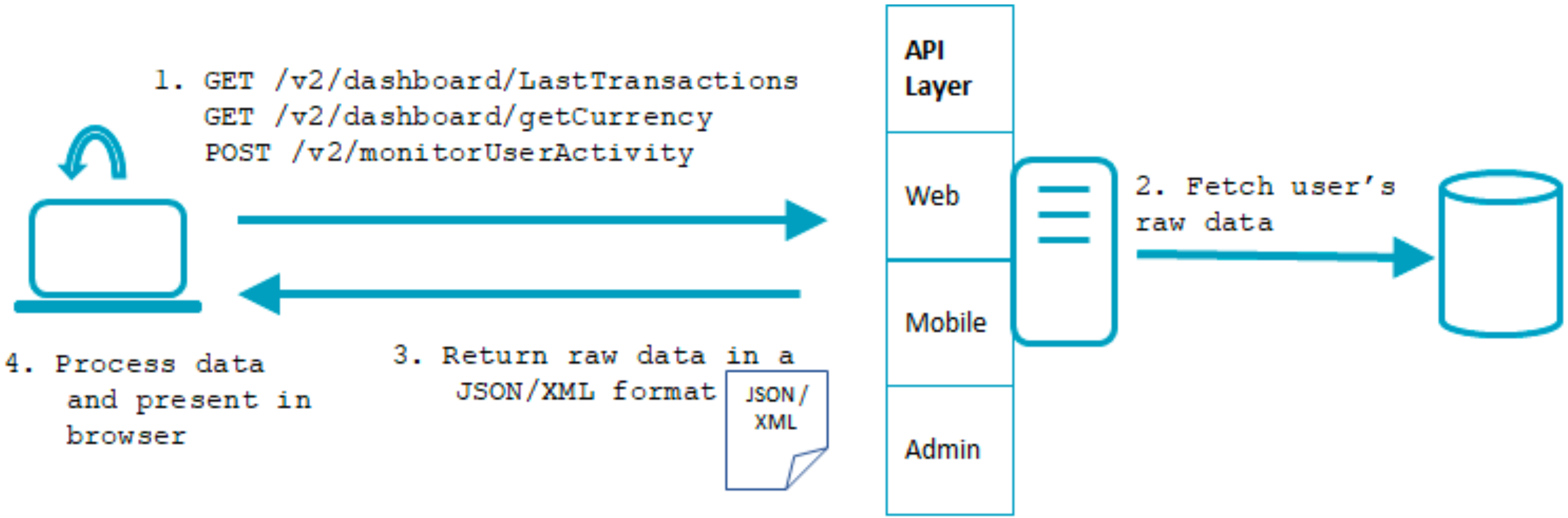
- **API Security:** APIs are more prone to exposing sensitive data since they can deliver structured data directly to other systems. This makes controlling access to data (e.g., limiting excessive data exposure) a major concern.
- **Web Security:** Web security risks may involve direct exposure through web pages but are more likely focused on vulnerabilities like improper input validation (leading to XSS, SQLi) or session hijacking.

“Traditional” web architecture



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

API-based API architecture

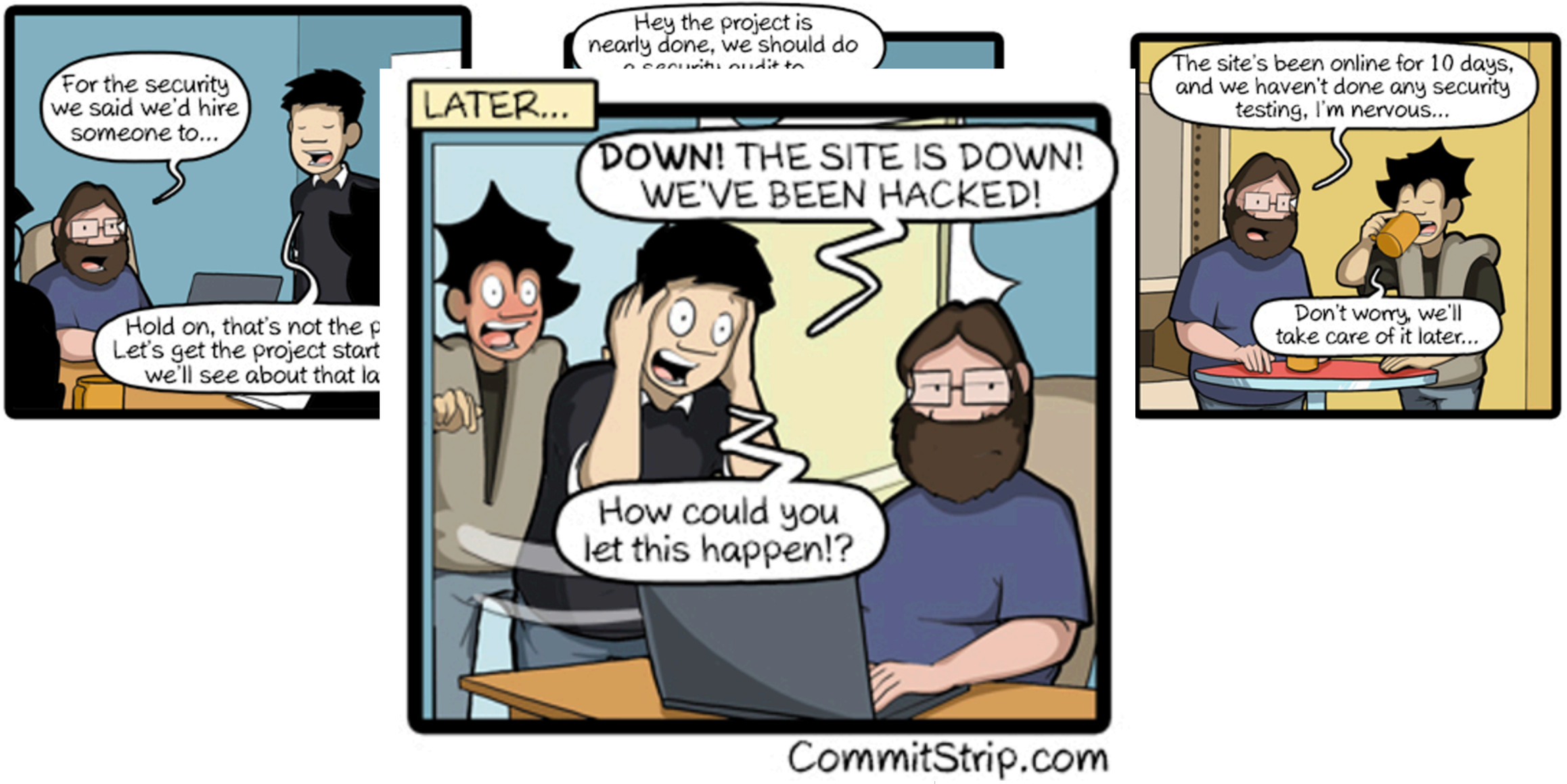


https://apisecurity.io/encyclopedia/content/owasp/owasp-api-security-top-10.htm

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



Security is still an afterthought!

Start testing security scenarios as early as possible

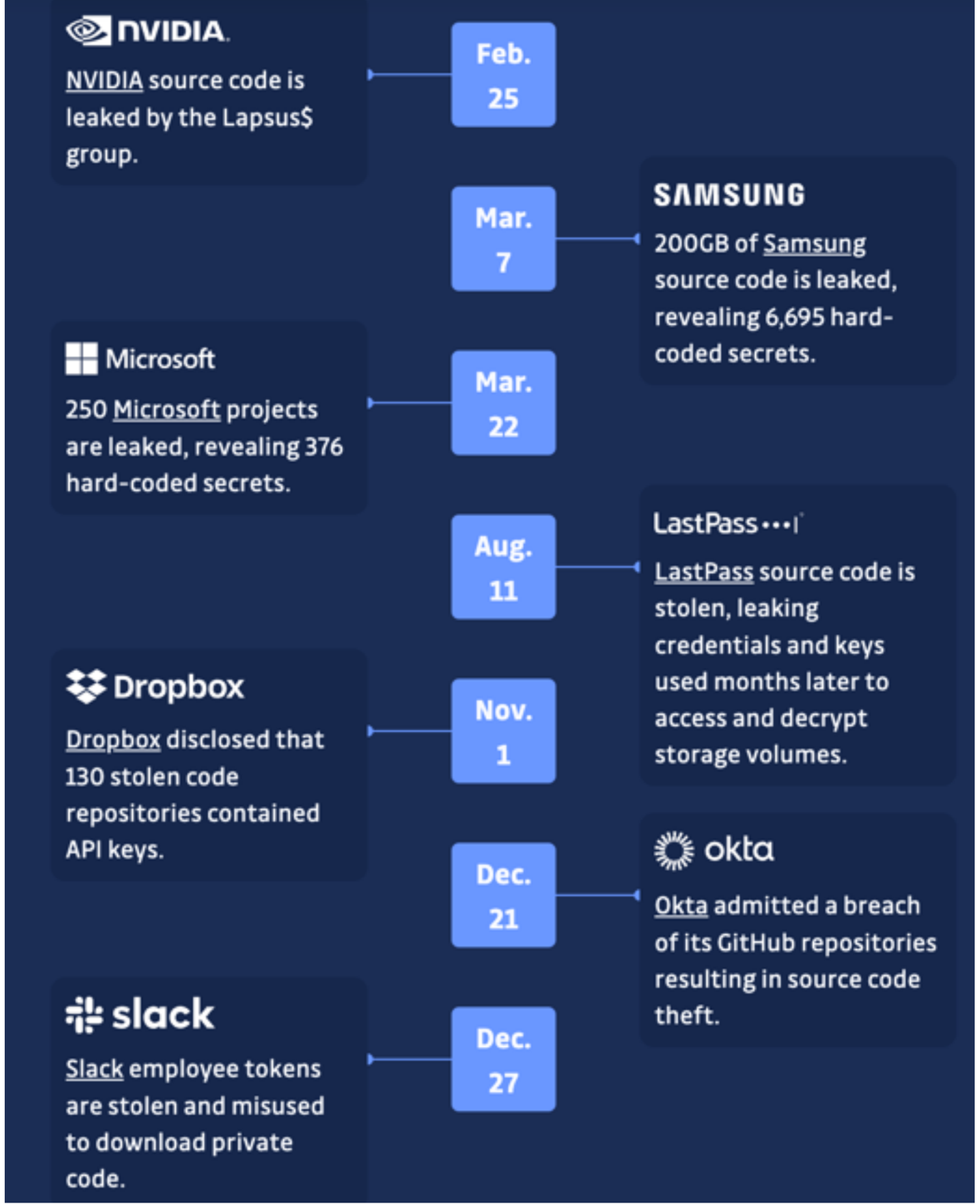
Bad design!

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

Loosing the keys

- Hardcoded tokens and keys
- Committed in Github
- Leaked in Paste.bin
- Leaked in Docker images

Secret scanning tools are your friend!
- Gitguardian
- GitLeaks (OSS)
- Github Secret Scanning



Graphic from: https://www.gitguardian.com/state-of-secrets-sprawl-report-2023

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 iteractions.
NON-REPUDIATION	There is (legal) proof that the action has taken place.

SECURITY PRINCIPLES

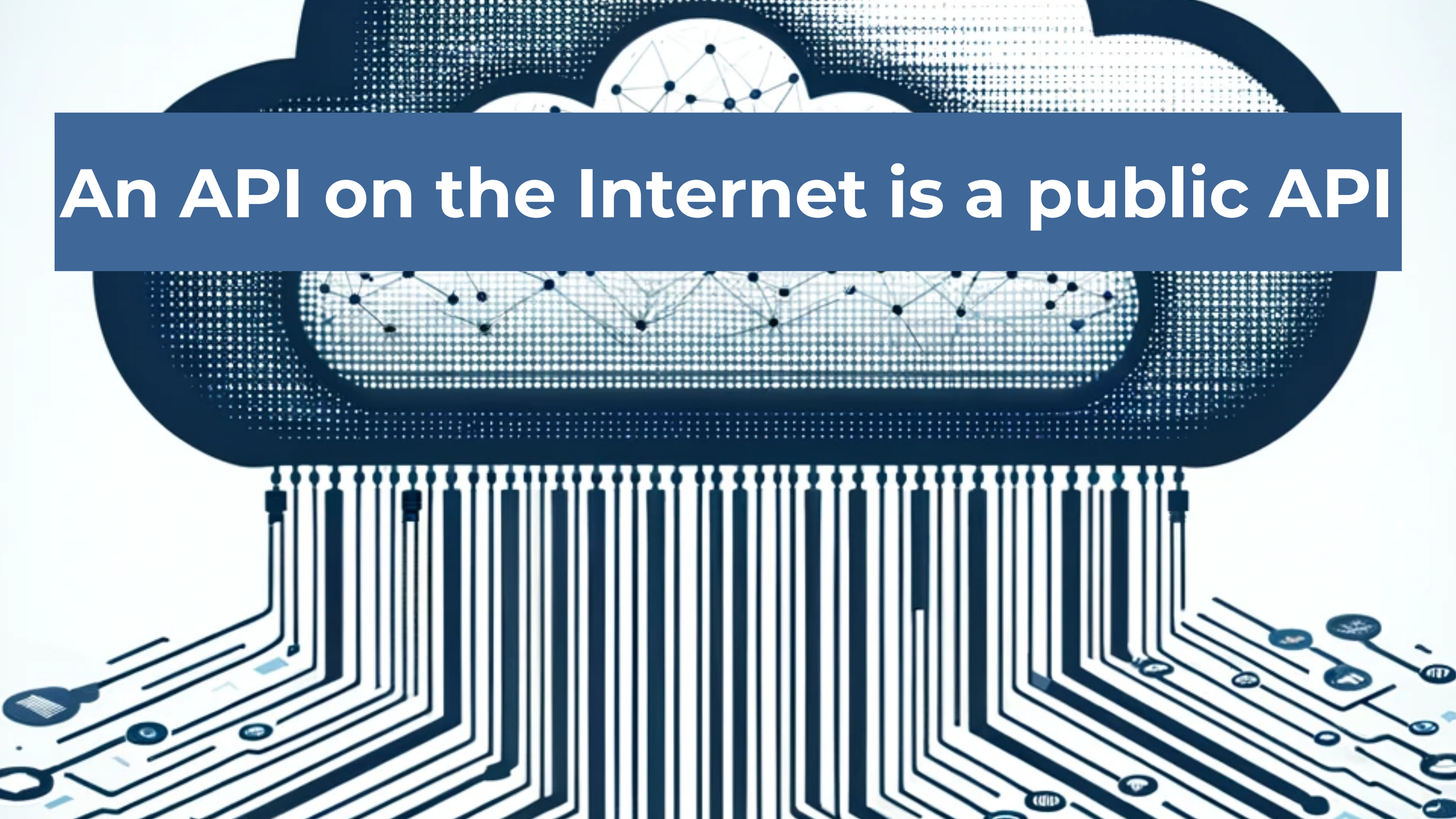
API Security is risk based

- ☑ Identify Threat Actors
- ☑ Know Data Sensitivity

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

- 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
- Cloud version: https://pixiapp.uksouth.cloudapp.azure.com

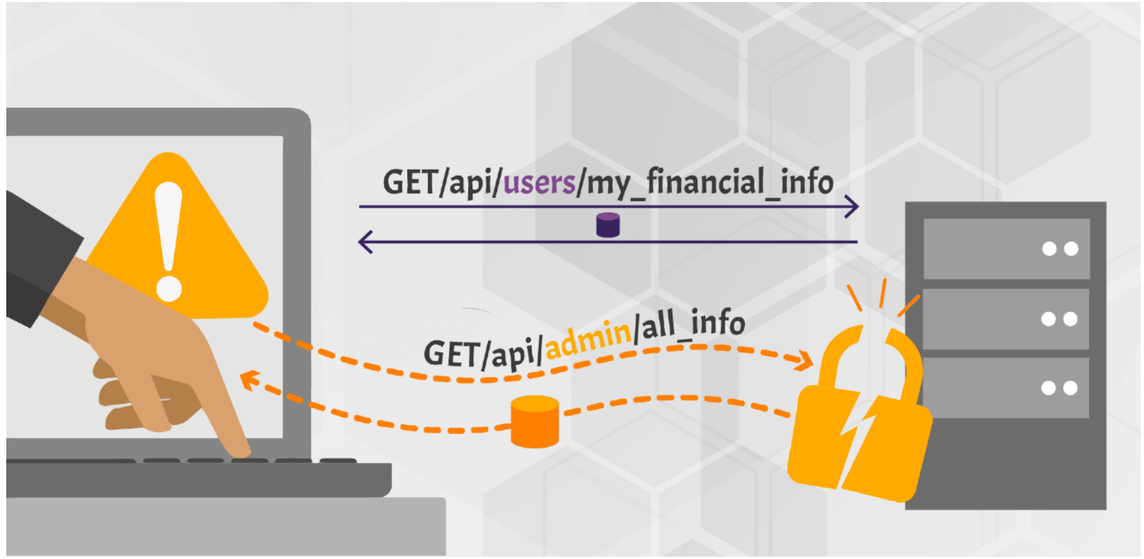
AUTHORIZATION

Authorization Levels

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

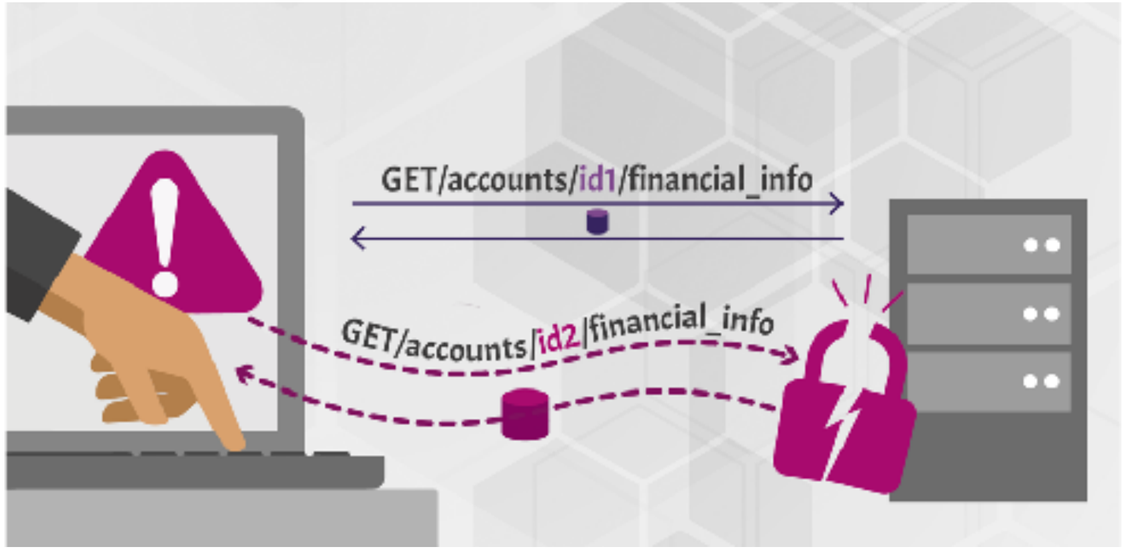
- The Attack
  - Data breach using leaked admin userid/passwords

- 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**.

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
    - GET mytelco.com/user_profile/1883456789
  - 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

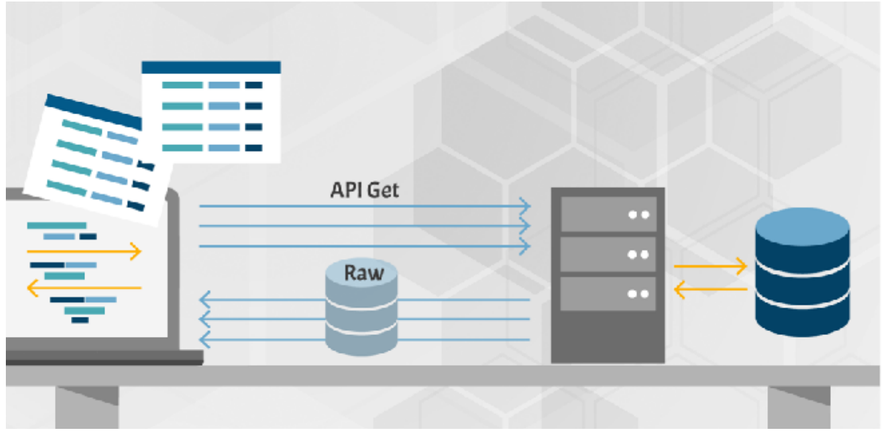
- **True fix : Fine-grained authorisation to resources in every controller layer**
  - Decision engine is required, preferably externally to the code.
  - Need to be able to enforce something like :
    - *Jean is allowed to access account 123, but cannot edit or delete it.*
    - *Peter is not allowed to access account 123*

- 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.**

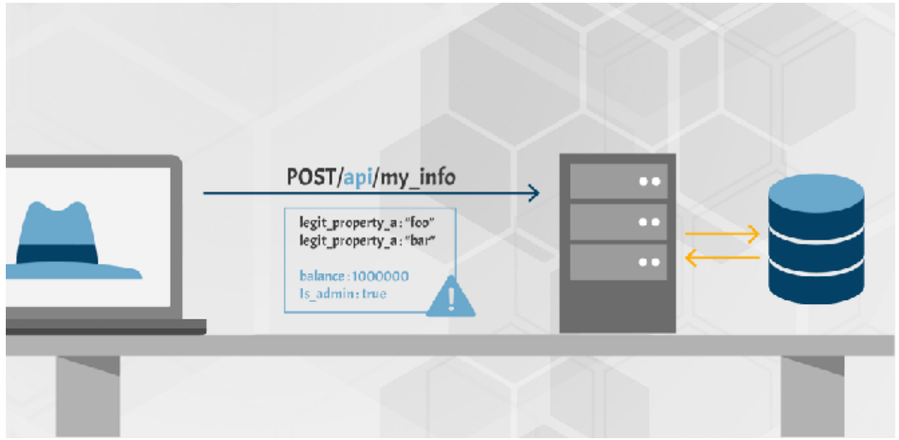
DATA AUTHORIZATION

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.


DEMO TIME

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

JWTs transport data too!

- JWT are only **encoded**, they 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.ey
J1c2VyIjp7Il9pZCI6ODUsImVtYWlsIjoiY3Vzd
G9tZXJAcGl4aS5jb20iLCJwYXNzd29yZCI6Imhl
bGxvcGl4aSIsIm5hbWUiOiJjb3N0ZXhwbGFuYXR
pb24iLCJwaWMiOiJodHRwczovL3MzLmFtYXpvbm
F3cy5jb20vdWlmYWNlcy9mYWNlcy90d2l0dGVyL
3NoYW5lSXhELzEyOC5qcGciLCJhY2NvdW50X2Jh
bGFuY2UiOjEwMDAsImlzX2FkbWluIjpmYWxzZSw
iYWxsX3BpY3R1cmVzIjpbXX0sImlhdCI6MTYwMz
IxNjIwOX0.DjgTBCev5Kq_DpvBwfKva3K3rLCs4
r9hN17S-hh6qMI
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


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


PAYLOAD: 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
```


{api addicts}

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

- 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	• Goes in clear (encoded) over the network • Deprecated.	• Backward compatibilty
API Key	Long string of characters	• Travels over network • Prone to leaks (Github, hardcoded in apps etc.) • Expiration management	• Simple to use • Opaque
OAuth Access Token	• Long string of characters • JWT	• Used for authentication • Complexity • Carries potentially sensitive data	• Delegated authorization • Reduced security risks • Expires
OpenID Connect	• JWT	• Complexity	• True Authentication • Session/Logout management • Reduced Security Risks
Signed Requests	Digital signature in header/body	Complex for API consumers	• Integrity of information • API/Signing keys are not travelling over network.

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….

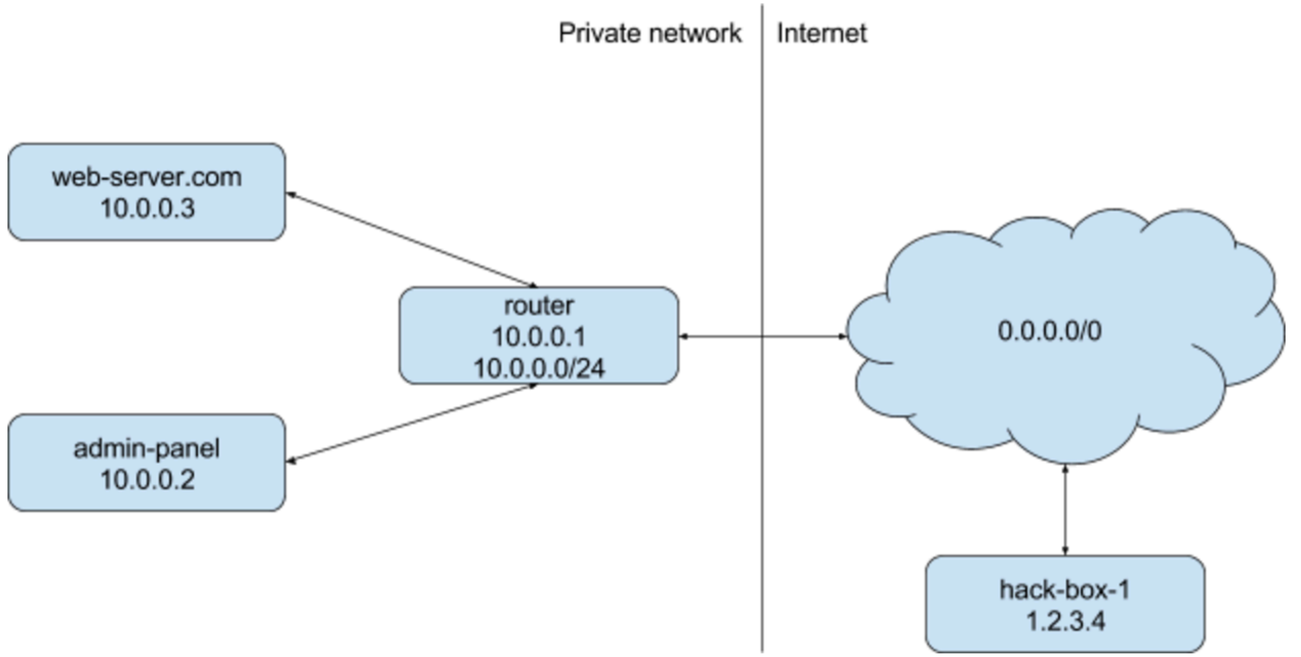
Weak Authentication
+
Bad Rate Limiting
=

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.**

https://portswigger.net/web-security/ssrf



From: https://www.hackerone.com/application-security/how-server-side-request-forgery-ssrf

How to protect yourself from SSRF!

- Getting Started
  - **AllowList** of websites you want to allow access to (vs. Deny List)
  - 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!

ADDITIONAL THREATS

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 other words : 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:

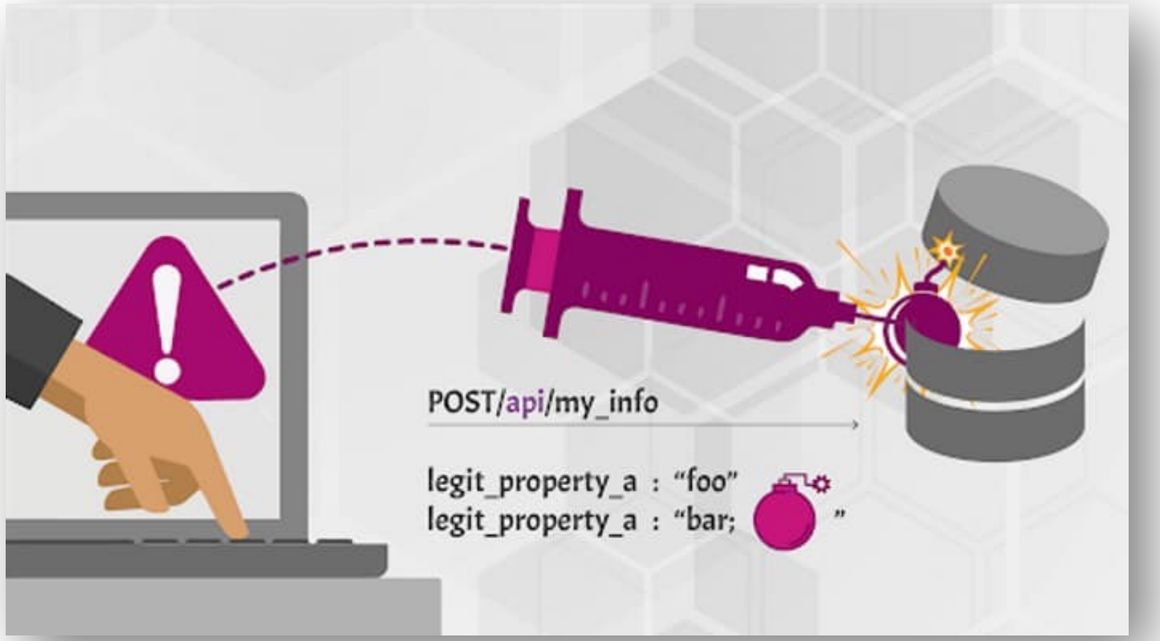
- **Device fingerprinting**: denying service to unexpected client devices (e.g. headless browsers) tends to make threat actors use more sophisticated solutions, thus more costly for them
- **Human detection**: using either a captcha or more advanced biometric solutions (e.g. typing patterns)
- **Non-human 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)



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

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

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

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

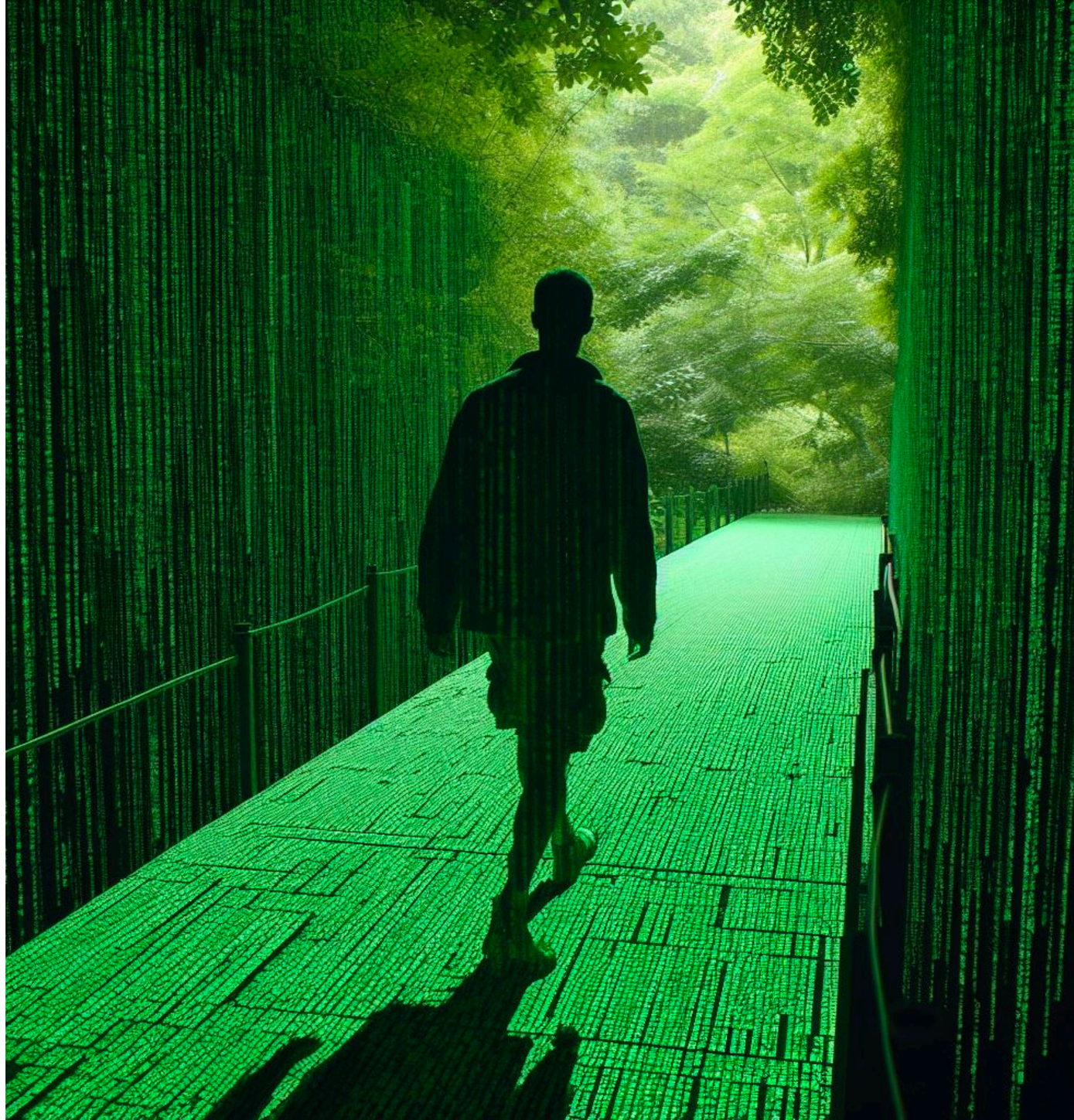
- Request payloads
- Headers
- JSON Web Tokens
- IP addresses (X-Forwarded-For)
- And everything else :)

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

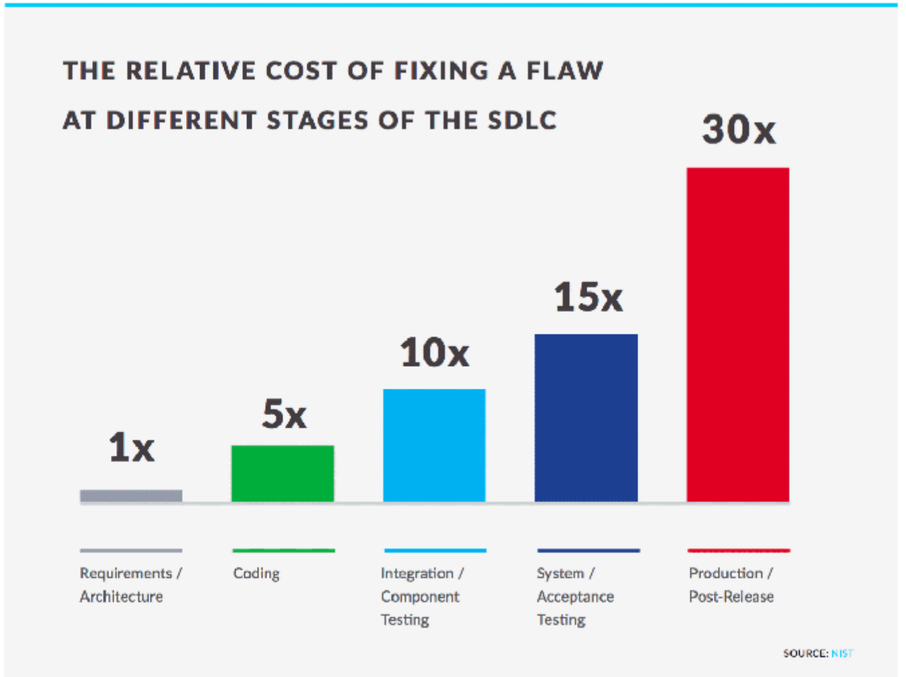
Hack yourselves leveraging API contracts

- ✓ 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/

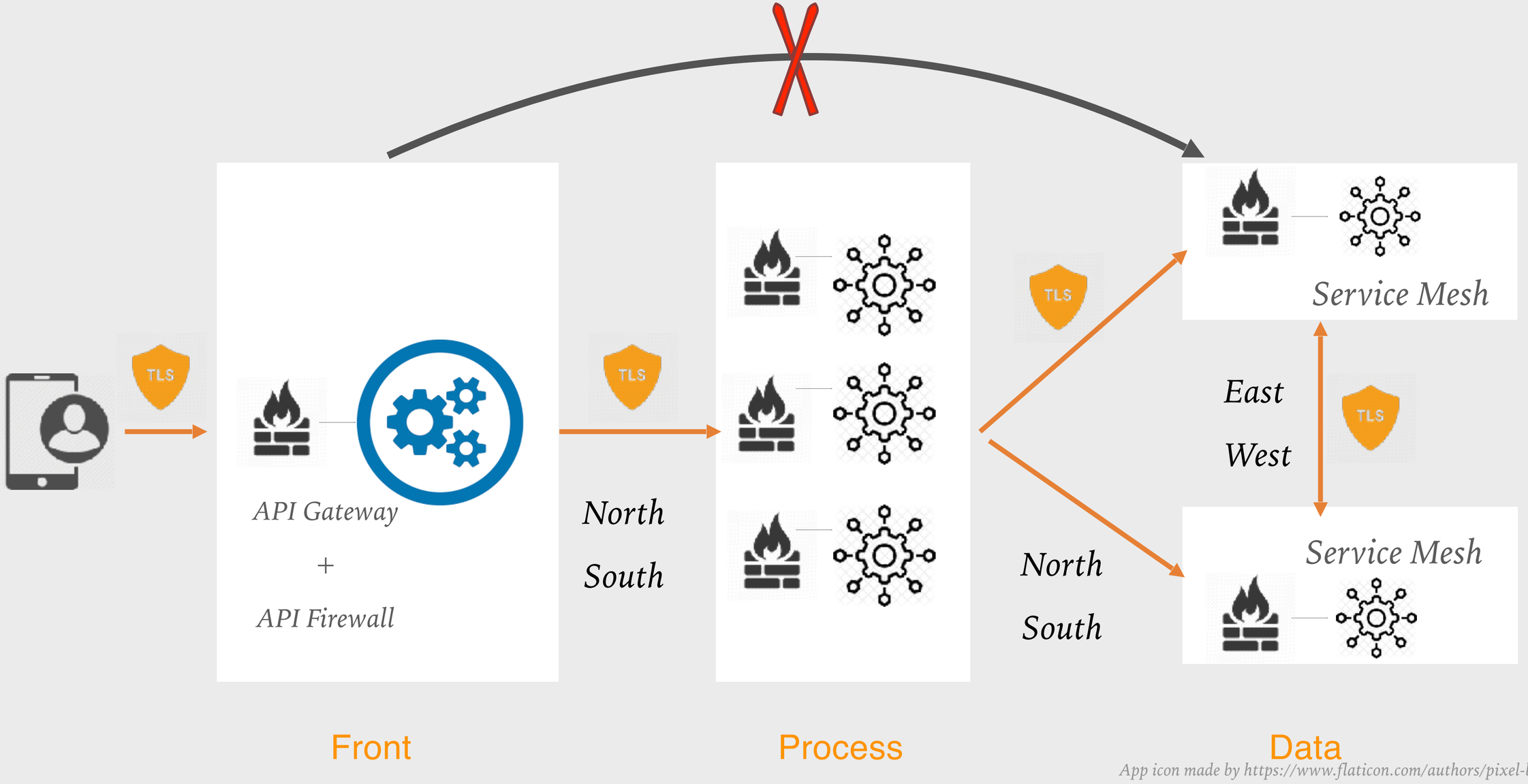


"I think security, in most cases, is not a single person's specialization. Security must be a practice of every member of the team from the frontend developer to the system administrator (also non tech roles)."

From: Gitlab DevSecOps report - 2021

ARCHITECTURE AND TOOLS

DEPLOYMENT ARCHITECTURE



App icon made by https://www.flaticon.com/authors/pixel-buddha

{api addicts}

Further learning

- **42Crunch webinars with Jim Manico on SSRF/CSRF**
  - https://42crunch.com/defending-apis-with-jim-manico-episode-1/
- **42Crunch webinars with Philippe De Ryck on OWASP Top 10 :**
  - https://42crunch.com/owasp-api-security-top-10-webinar-series-recordings-3/
  - https://42crunch.com/webinar-top-things-you-need-to-know-about-api-security/
- **42Crunch webinar on JWTs**
  - https://42crunch.com/webinar-json-web-tokens-api-security/
- **OWASP Input Validation Cheat Sheet**
  - 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
- **OWASP Transport Layer 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
- **OWASP HTML5 Security Cheat Sheet**
  - 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

DATOS DE CONTACTO

+34 **91 764 79 82**

contacta@apiaddicts.org

www.apiaddicts.org

Facebook

ApiAddicts

Linkedin

API Addicts

Twitter

@APIAddicts

Meet Up

API Addicts

Youtube

API Addicts

{api addicts}