CONFERENCE
MADRID 2023

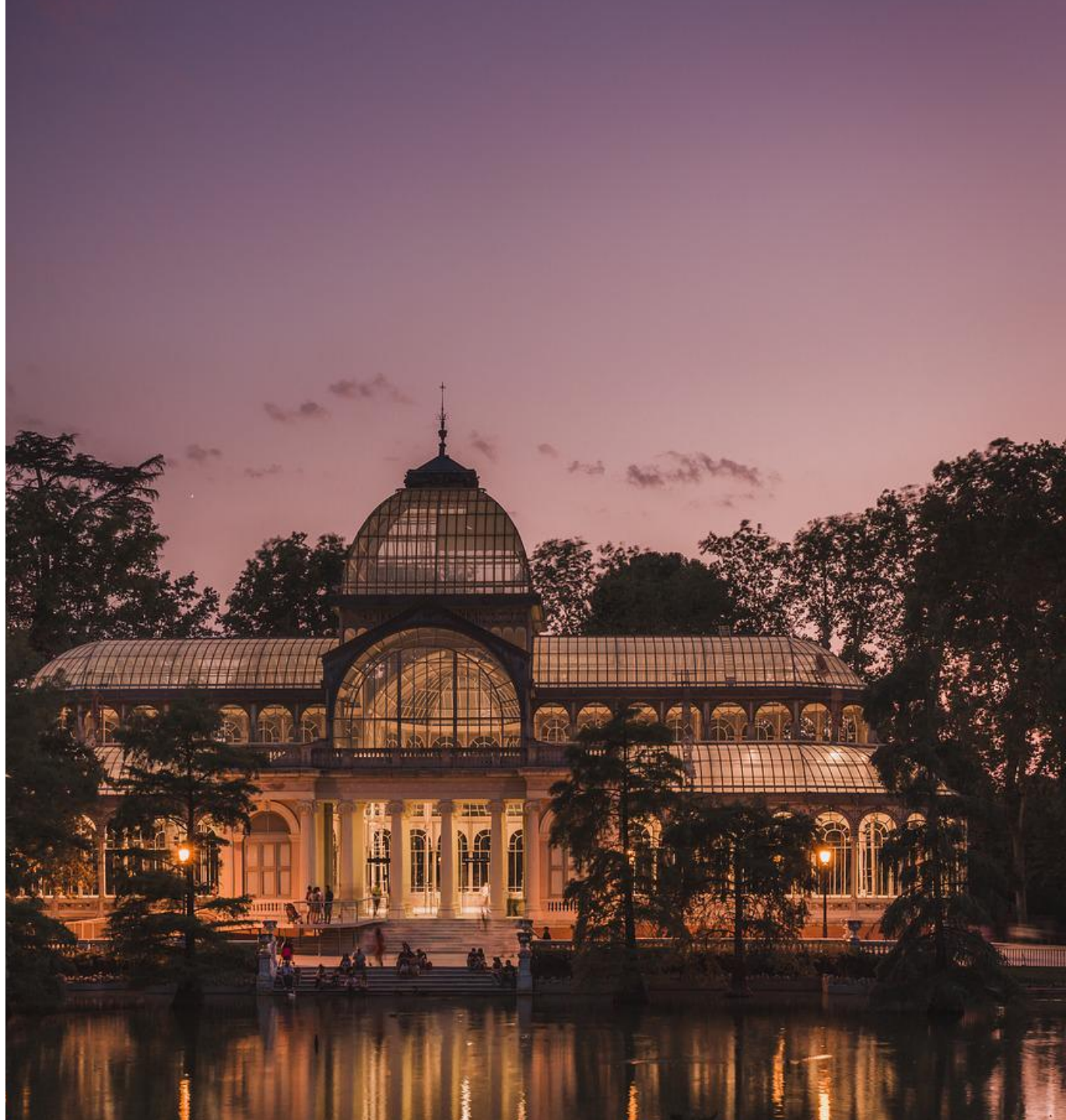
Building Rugged APIs
And surviving in the attempt

Isabelle MAUNY | @isamauny

Field CTO & Co-Founder - 42Crunch

Glad to be here!

- Field CTO / Founder of 42Crunch and apisecurity.io
- French National, living in Spain for past 20 years
- Most of career in the integration world, pioneering what would become API Management



https://apisecurity.io

28 April, 23

Issue 217: Wordle API exposes answers, Twitter API breach updates, AWS exposed dangerous API

27 March, 23

Issue 216: Hacking a .Net application, state of API security report, myths of API security

5 March, 23

Issue 215: API flaws in Lego marketplace, API style guides, 42Crunch joins MISA

9 February, 23

Issue 214: Google Cloud's four pillars of API security, Cerbos for API permissions, attacking predictable GUIDs

26 January, 23

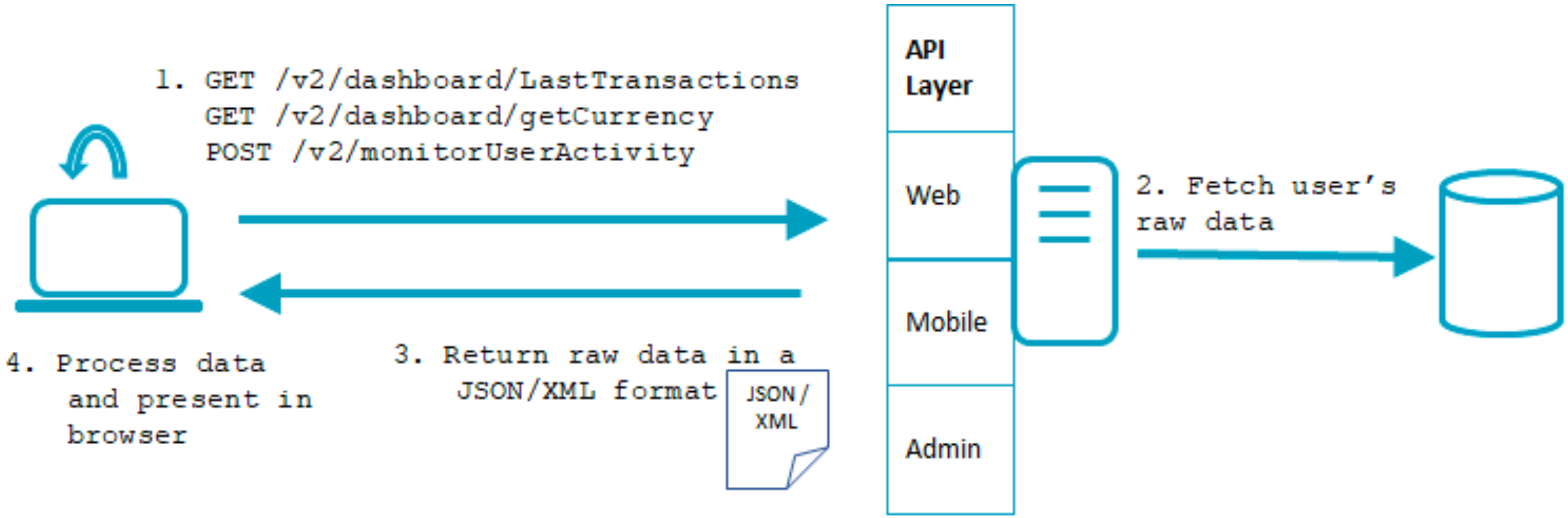
Issue 213: Supply chain vulnerability in IBM Cloud, hardcoded API keys in Algolia portal, JSON-based SQL attacks

15 January, 23

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

WHY ARE APIS SUCH A PROBLEM?

We lost the server-side controller layer



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

Security is still an afterthought!

Start testing security scenarios **as early as possible**

Design Flaws

APIs suffer from many design flaws, which are hard, including impossible to fix after the fact.

Before doing an API. **Stop and Think.**

OWASP API Security Top 10 2023 (RC)

- 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 Server Side Request Forgery (**New**)
- API7:2023 Security Misconfiguration
- API8:2023 Lack of Protection from Automated Threats (**New**)
- API9:2023 Improper Assets Management
- API10:2023 Unsafe Consumption of APIs (**New**)

BAD PROBLEMS USUALLY OCCUR WHEN MULTIPLE OF THESE ARE COMBINED

*Meaning of **rugged** in English*

rugged

adjective

UK /ˈrʌg.ɪd/ **US** /ˈrʌg.ɪd/

rugged ***adjective*** **(NOT EVEN)**

Add to word list

(of land) wild and not even; not easy to travel over:

- *rugged landscape/terrain/hills/cliffs*

+ SMART Vocabulary: related words and phrases

rugged ***adjective*** **(STRONG)**

strong and simple; not delicate:

- *Jeeps are rugged vehicles, designed for rough conditions.*

An API **must not 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 :)

AUTHORIZATION

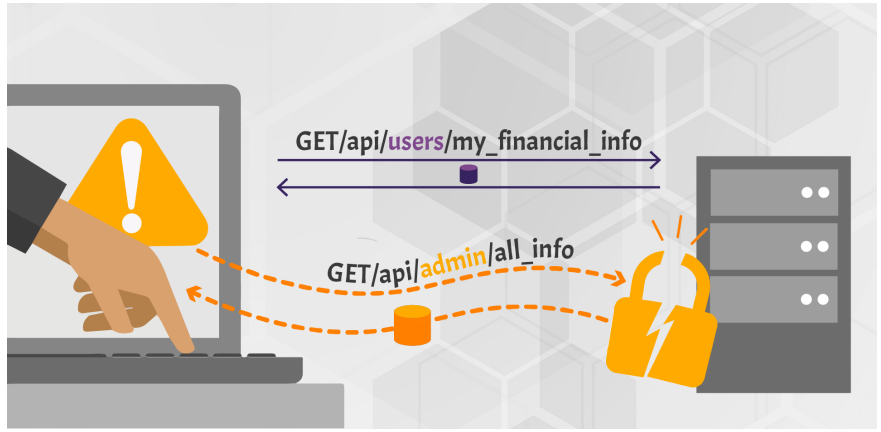
Authorization Levels

API 5 : BFLA

API 1: BOLA

```
"/user/{userid}": {
  "put": {
    Scan | Try it
    "tags": [
      "users"
    ],
    "summary": "edit user information",
    "description": "user supplies valid token and updates all user info",
    "operationId": "edituserinfo",
    "parameters": [⋯
    ],
    "requestBody": {
      "description": "userobject",
      "content": {
        "application/json": {
          "schema": {
            "$ref": "#/components/schemas/UserUpdateData"
          }
        }
      },
      "required": true
    },
```

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.

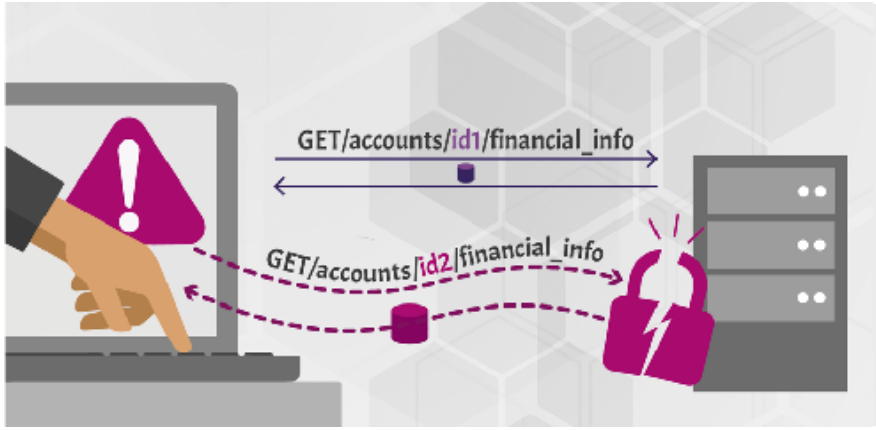
Likud Party

Likud Voting App

https://www.zdnet.com/article/netanyahus-party-exposes-data-on-over-6-4-million-israelis/

- The Attack
  - Data breach using leaked admin userid/passwords
- The Breach
  - Unknown. Potentially could have leaked the details of 6.4 million Israelis.
  - Citizens have since then sued the Likud party
- 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.

Uber

New Relic

Google Cloud

Online Registration System
Ministry of Electronics & Information Technology
Goverment of India

SONICWALL

Smart Scale

Full Vulnerability Report by Fortbridge

- The Attack
  - Researchers have been able to do a complete take over of user accounts, including resetting passwords.
- The Breach
  - Unknown. Took the company 7 months to fix the issue :(
- Core Issues
  - Limits enforced at client level (number of users per family)
  - IDs for users can be forged once you understand the pattern
  - No validation of user's role: once you have a token, you can delete or create users from other people's accounts.


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 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 *<u>are not</u>* helping here, as they limit access to a function and not to a resource.**

-

AUTHENTICATION

Broken Authentication (API 2)

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.

facebook

Auth0

SIEMENS

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 for authentication**
  - 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.

Addressing Broken Authentication

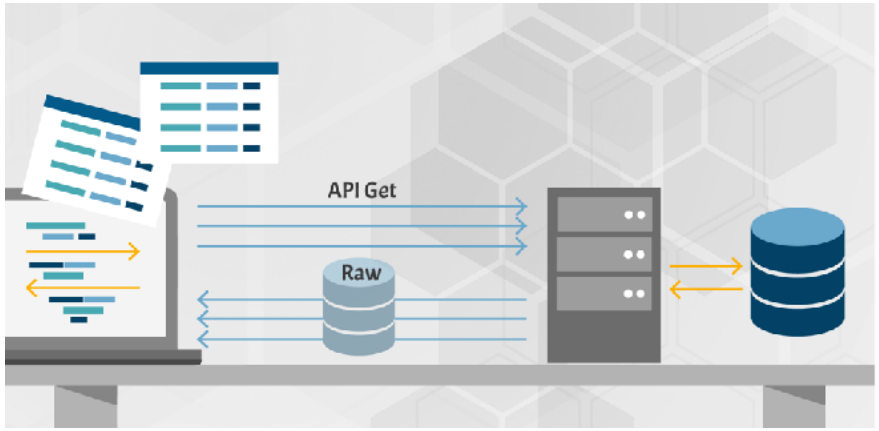
- Leverage OpenID Connect best practices as per the FAPI (Financial-grade APIs) profiles
  - Built as part of OpenBanking initiative
  - https://openid.net/specs/openid-financial-api-part-1-1_0-final.html
  - https://openid.net/specs/openid-financial-api-part-2-1_0.html
- Use multi-factor authentication
- Special attention to **authentication and password resets endpoints**
  - Specific rate limiting and brute force prevention
  - Same format message for all potential authentication issues (bad user or bad password)

- **Test authentication resilience with all kind of combinations!**
- **Learn more: https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication**

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

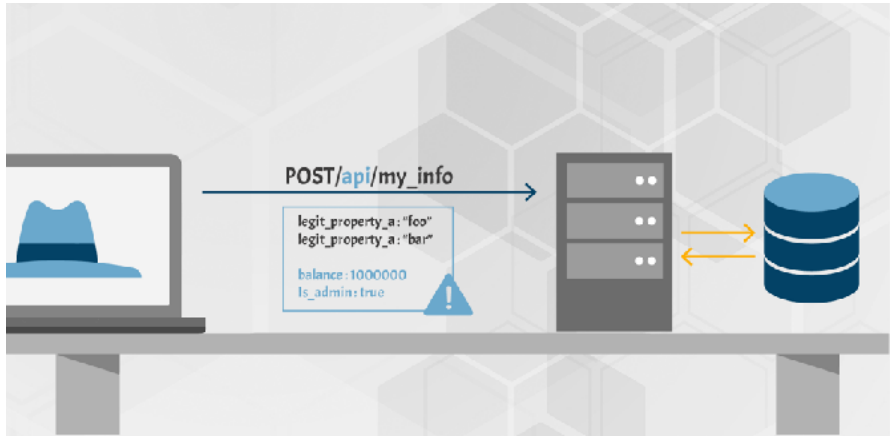
DATA AUTHORIZATION

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

JWTs transport data too!

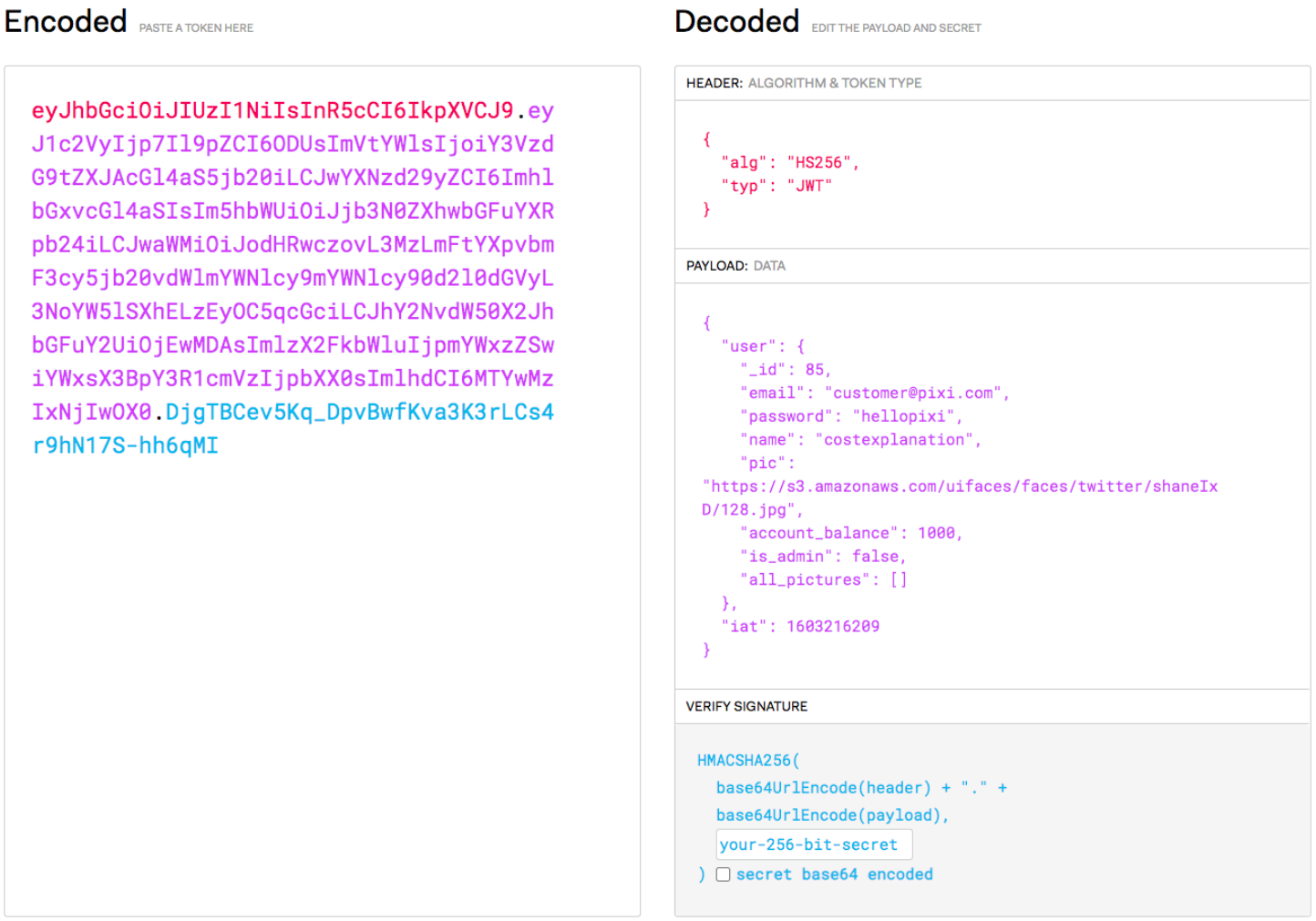
- Can leak data

- Can be prone to injections (example: kid sql injection)

Recommended best practice:

* Use opaque tokens for external consumption

* Use JWTs for internal consumption




DEMO TIME

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

- Protect all authentication endpoints from abuse (login, password reset, OAuth endpoints)
  - Smart rate limiting : by API Key/access token/user identity/fingerprint
  - Short timespan
- 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"

DOS Protection recommendations

- Constraint 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

+

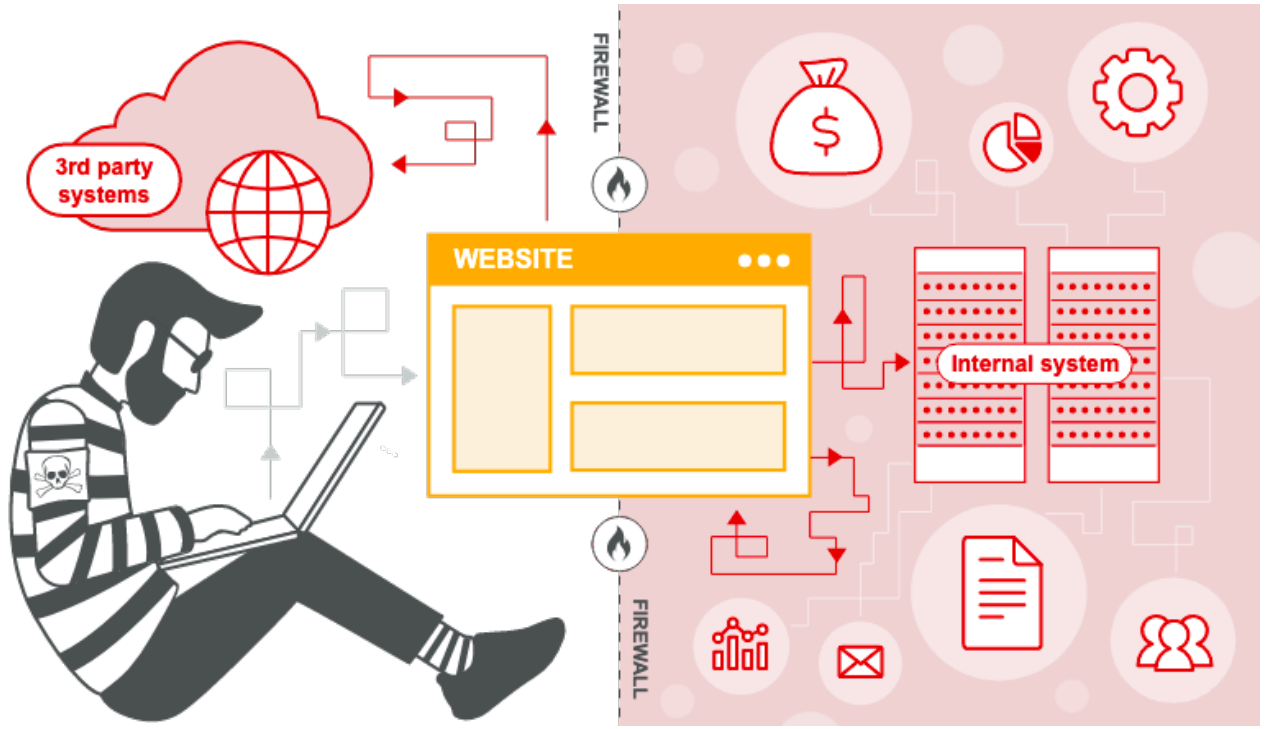
Bad Rate Limiting

=

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!

- **Goal**: prevent connection to any system that you're not supposed to connect to !
- Getting Started
  - Whitelist of websites (3rd party APIs for example)
  - Network policies
  - Blacklist of protocols (ftp:// , file://)
  - Disable 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 or create a DNS entry pointing to localhost..
  - Microsoft API Management just suffered this !
    - https://ermetic.com/blog/azure/when-good-apis-go-bad-uncovering-3-azure-api-management-vulnerabilities/

- Check references for more on attackers can circumvent prevention measures.


WE NEED
PROACTIVE
SECURITY

CALL TO ACTION!

Use OWASP 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

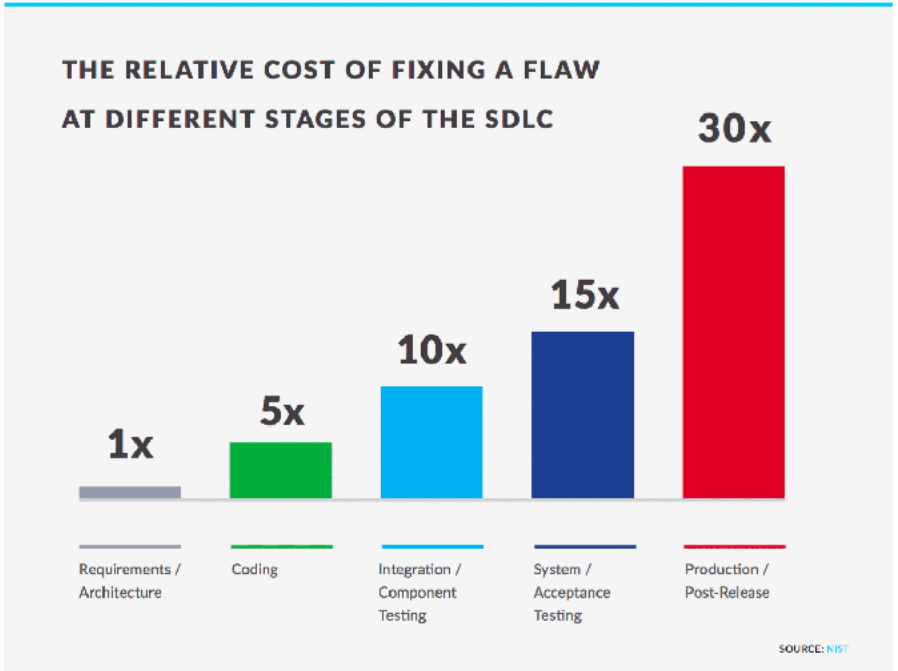
Hack yourselves leveraging API contracts

- ✓ 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

References

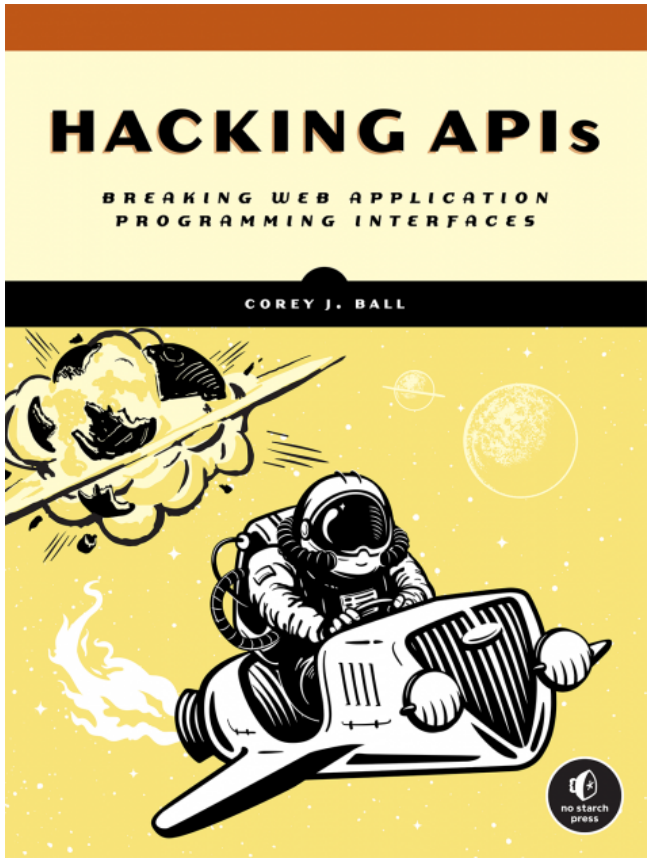
- **Evolution of OWASP API Top 10** @APIDays Paris: https://www.youtube.com/watch?v=ARIZNLzKwJI
- **BOLA 101**: https://inonst.medium.com/a-deep-dive-on-the-most-critical-api-vulnerability-bola-1342224ec3f2
- **OAuth Playground**: https://www.oauth.com/playground/
- **OAuth / OIDC Free Course**: https://pragmaticwebsecurity.com/courses/introduction-oauth-oidc.html
- **SSRF:**
  - **Intro @APIDays Paris**: https://www.youtube.com/watch?v=vG4n4ivsFnk
  - Overview and Labs: https://portswigger.net/web-security/ssrf
  - https://danaepp.com/exploiting-ssrf-in-an-api
    - And in general, Dana's blog!

Learning more

APISecurity.io

https://apisecurity.io/

“Hacking APIs” - Corey Ball



https://nostarch.com/hacking-apis

Learning Application Security

Buy the book