{ API:WORLD™ }

OCT 24–26 Santa Clara, CA OCT 31–NOV 2 Live Online

OWASP Top 10 2023 Deep Dive

Isabelle MAUNY | @isamauny

Field CTO & Co-Founder - 42Crunch

Glad to be here!

- Field CTO / Founder of 42Crunch and apisecurity.io
- French National, I have lived in Spain for the past 20 years
- Most of career in the integration world, pioneering in 2004 what would become API Management
- This presentation and sample code at: https://github.com/isamauny/apiworld-2023

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

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

AUTHORIZATION

Authorization Levels

```
"/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
    },
```

API 5 : BFLA

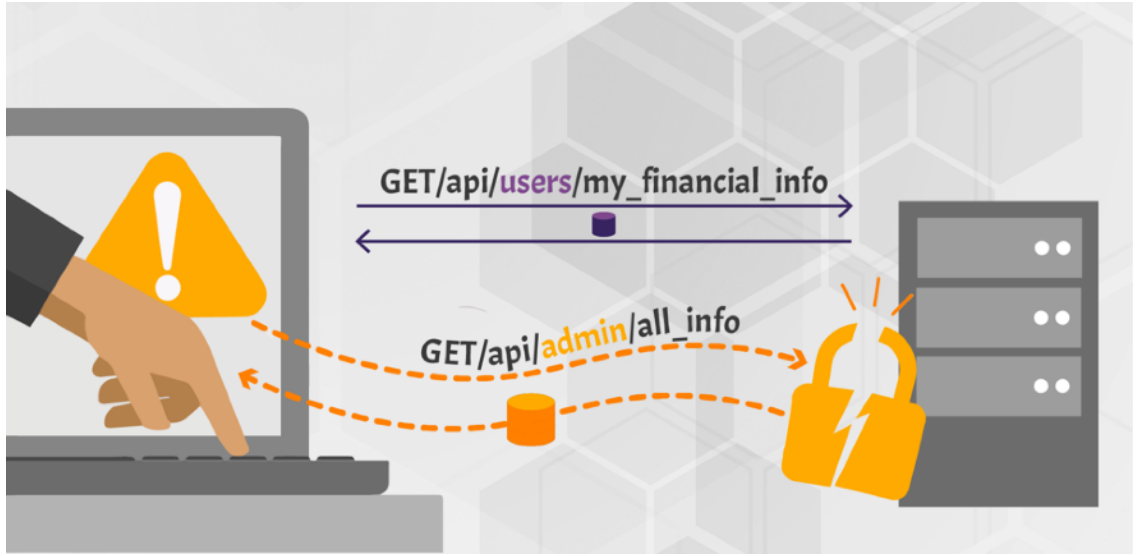
API 1: BOLA

API 3: BOPLA

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.

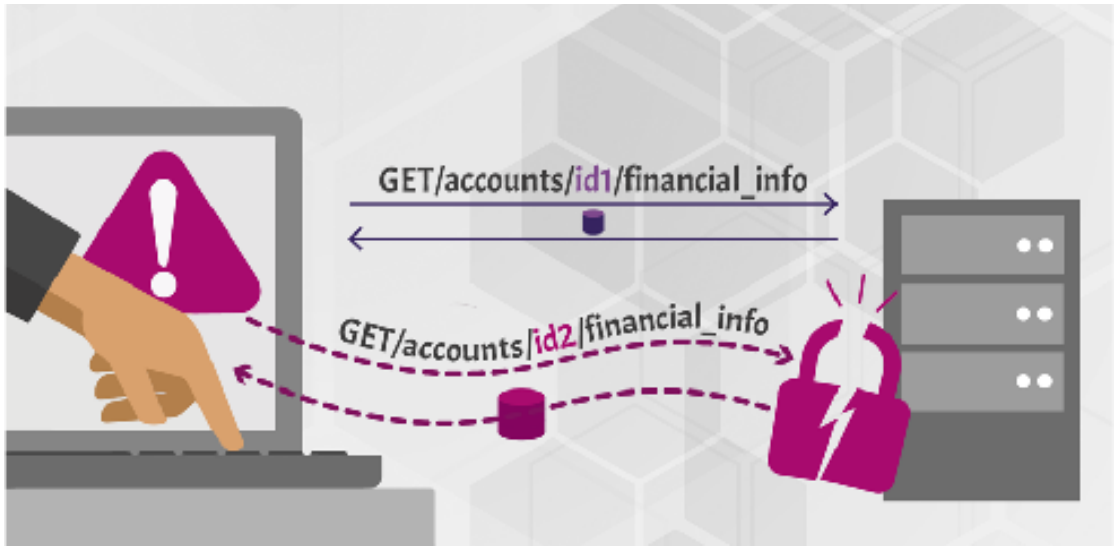


Voting App

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

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.

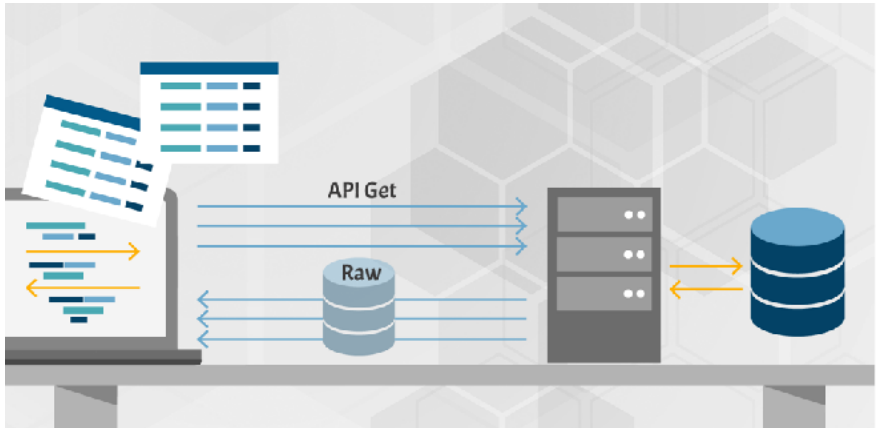
Which options are available ?

Type	Format	Potential Issues	Advantages
Basic Auth	b64 encoded User/password	• 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.

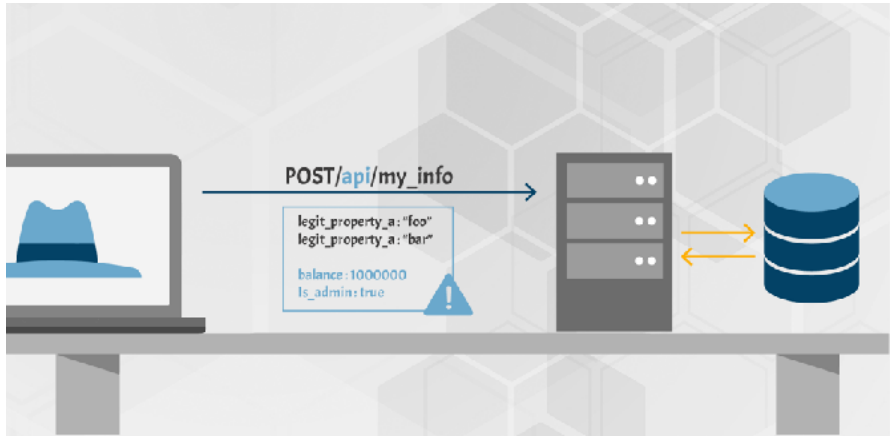
DATA AUTHORIZATION

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!

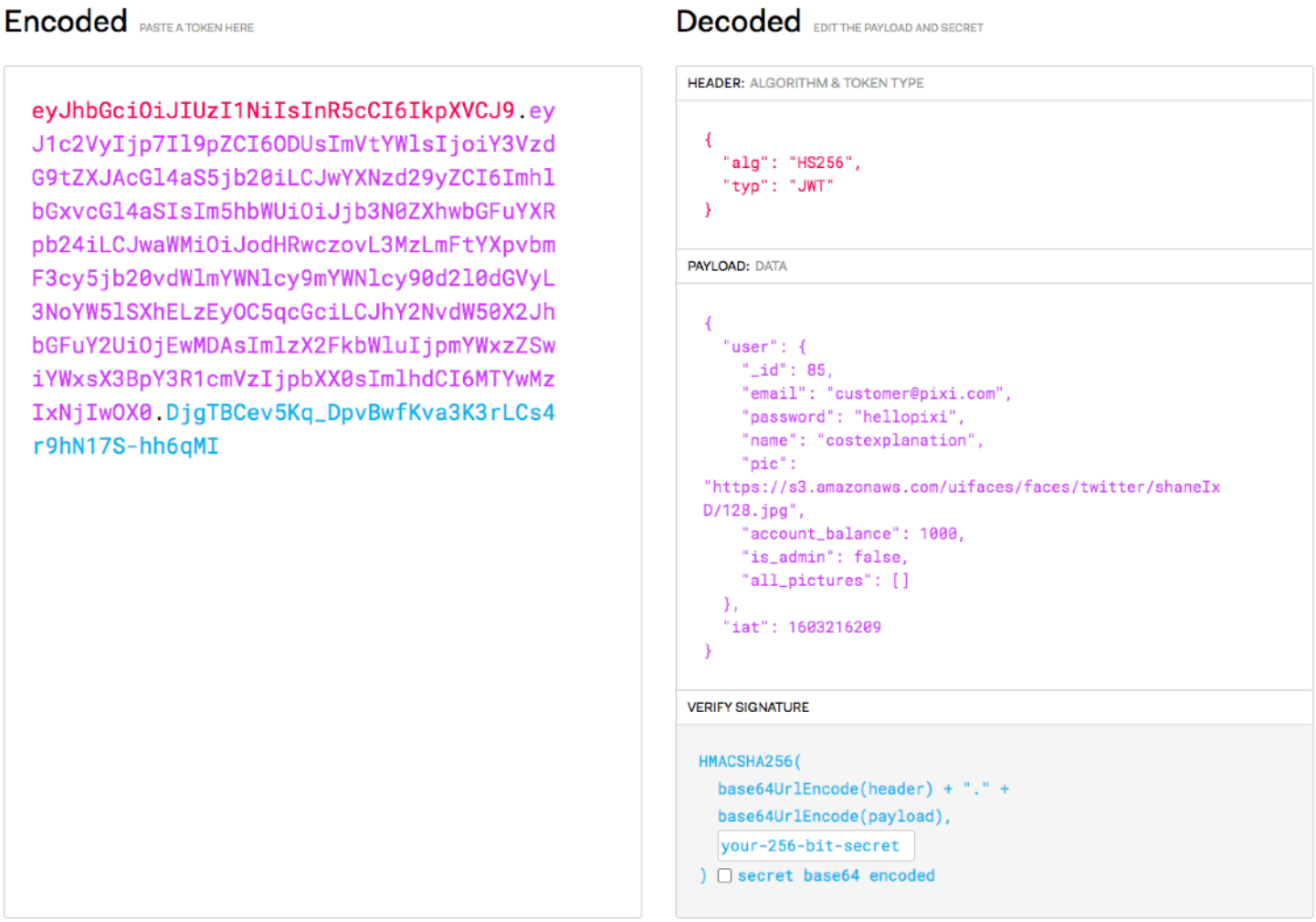
- Can leak data

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

Recommended best practice:

  * Use opaque tokens for external consumption

  * Use JWTs for internal consumption



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

- 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

+

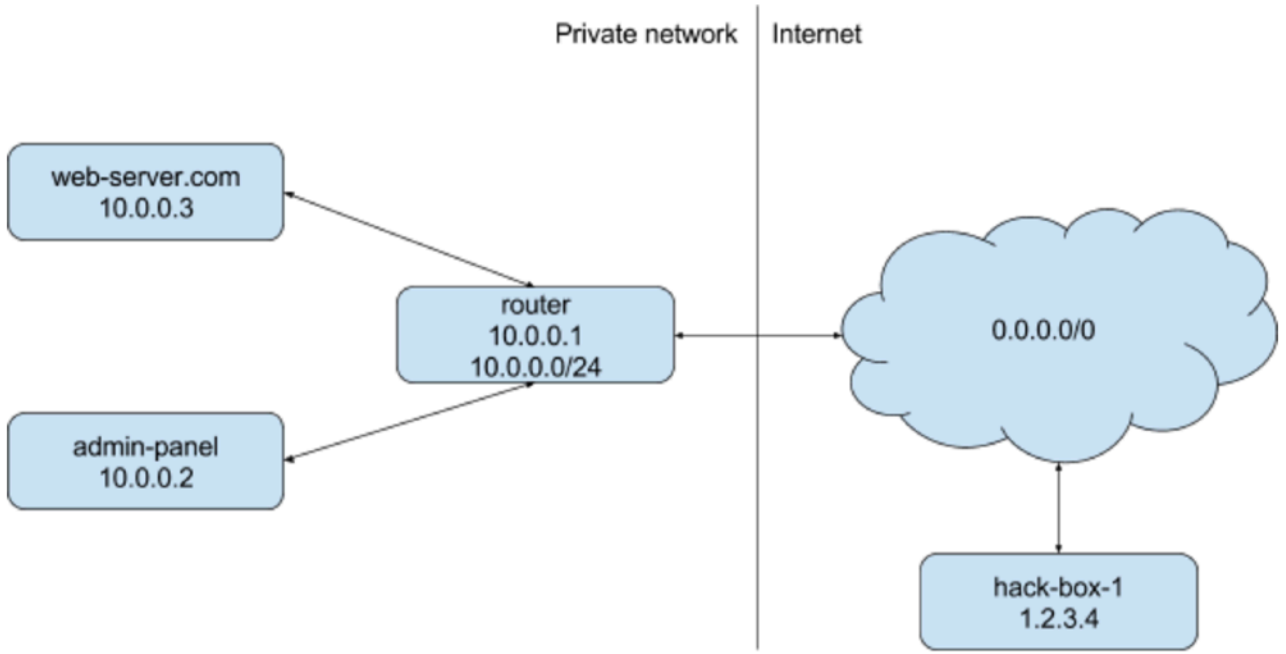
Bad Rate Limiting

=

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.**



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 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:

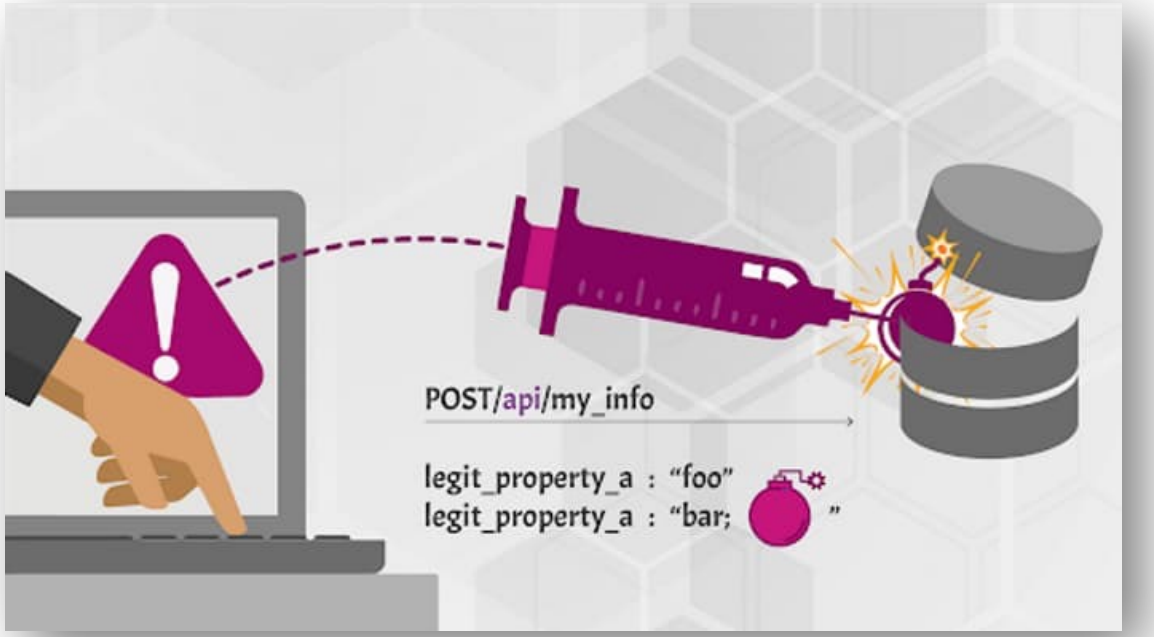
- **Device fingerprinting**: denying service to unexpected client devices (e.g. headless browsers) tends to make threat actors use more sophisticated solutions, thus more costly for them
- **Human detection**: using either a captcha or more advanced biometric solutions (e.g. typing patterns)
- **Non-human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

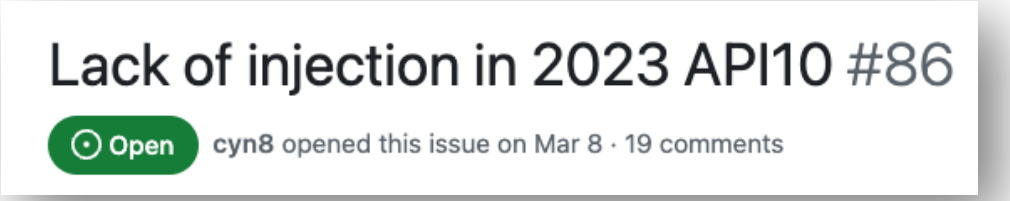
- Developers tend to trust data received from third-party APIs more than user input
- APIs are part of a supply chain and need to be secured at every point
  - When evaluating service providers, assess their API security posture
  - Ensure all API interactions happen over a secure communication channel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



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



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


IMPLEMENT
PROACTIVE
SECURITY

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

And that includes

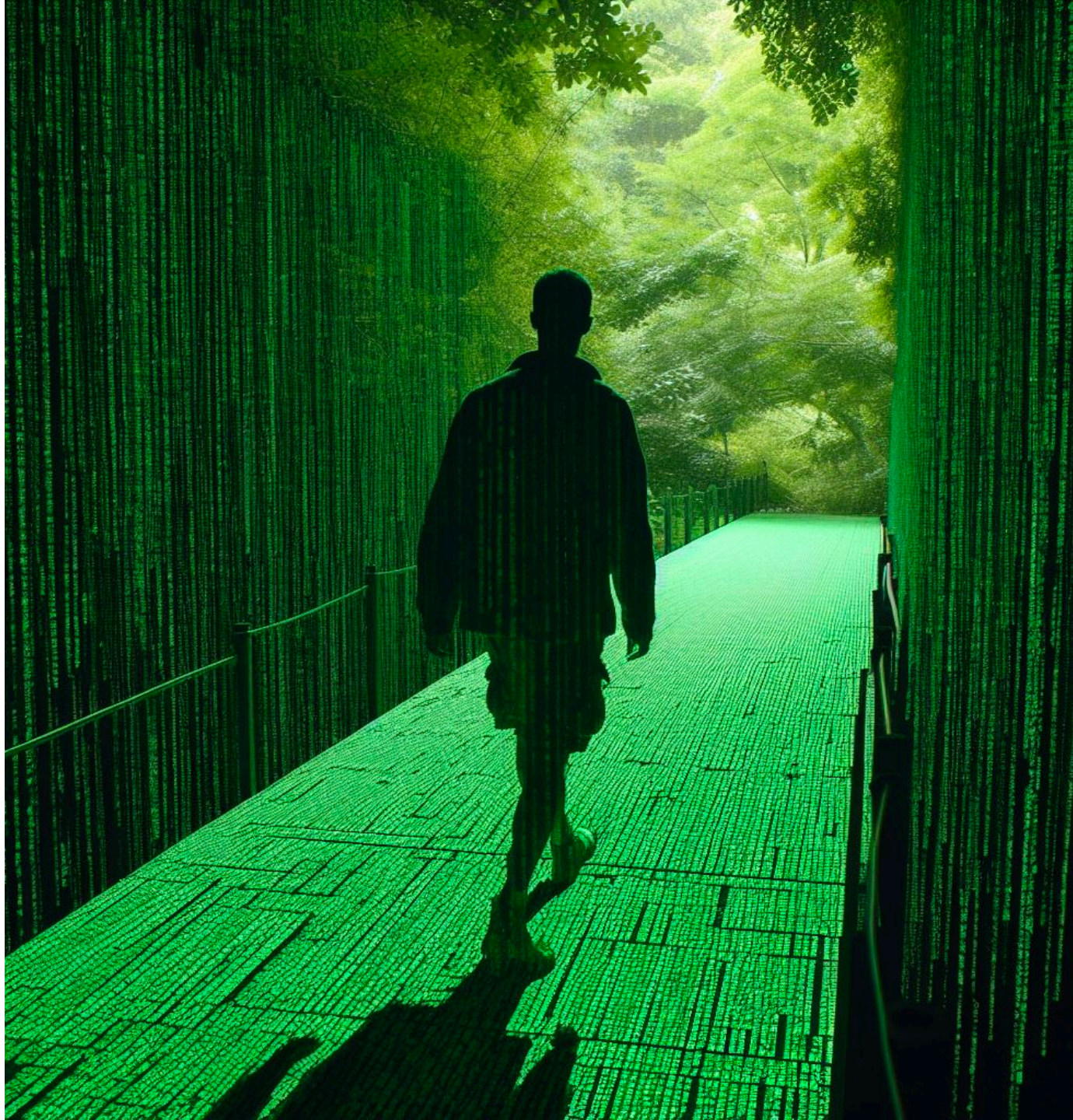
- Request payloads
- Headers
- JSON Web Tokens
- IP addresses (X-Forwarded-For)
- And everything else :)

Follow the “Hacky Path”

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

And if you automate them, even better!

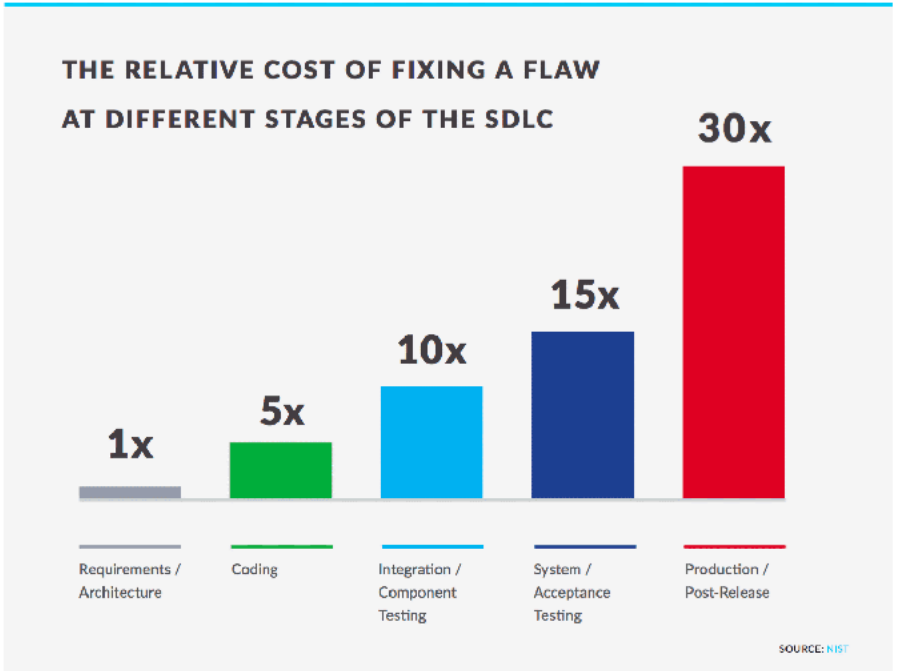
The way to protect yourself from Human Errors



CALL TO ACTION!

- Use OWASP API Top 10 as framework for design and testing
- Start with your critical APIs
- 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)."

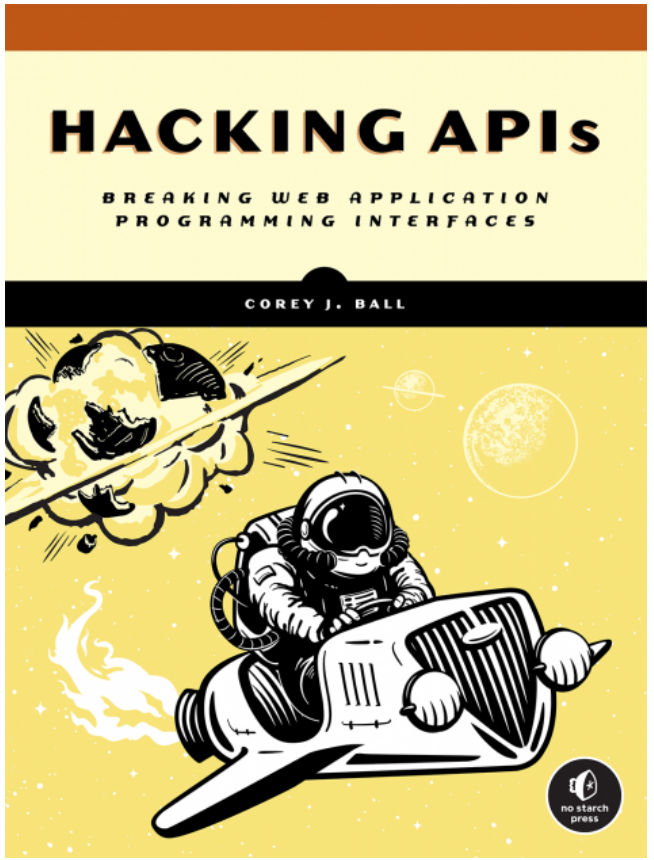
From: Gitlab DevSecOps report - 2021

Learning more

APISecurity.io

https://apisecurity.io/

“Hacking APIs” – Corey Ball



https://nostarch.com/hacking-apis

Learning Application Security

Buy the book

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!