🔄 OWASP API Top 10 – #6: Unrestricted Access to Sensitive Business Flows

📖 Definition

APIs that reveal valid business operations, also known as business flows, without sufficient controls are vulnerable to abuse or automation of those flows, frequently due to logical errors. Though ❌ can be used in unexpected ways, the business flow functions as planned.

🧠 Key Concept: Business Logic Abuse

"Happy path":

The recommended course of action for the user (e.g., reset password)

"Exploit path":

What an attacker manages to accomplish (e.g., brute-force the reset code)

📉 Real-World Case Study: Instagram Account Takeover

Situation:

Instagram used a 6-digit verification code with one million possible combinations to reset passwords.

200 guesses per IP is the rate limit.

Ten minutes for the timeout

What happened:

A hacker found that the 200 guess limit was reset when the IP was changed.

Within ten minutes, they brute-forced every six-digit code they could think of using rotating IPs and automation.

Effect:

Brute-forcing the reset code could potentially take over any account, including those of famous people like Taylor Swift.

🔓 Why This Happens

Unlike traditional security flaws, business logic is frequently not tested.

Developers concentrate on making things function rather than how they might be abused.

Security tools such as WAFs and scanners are unable to identify patterns of logic abuse.

✅ The Best Ways to Stop Flow Abuse

1. Consider Yourself an Attacker

"What could I abuse if I could automate this?"

Create threat models for flows of high value.

2. Critical Operations with Rate Limits

Set rate limits for each user, IP, and device.

Safeguard sensitive API calls, signups, referrals, promo codes, and password resets.

3. Rapidly invalidate reset tokens

Make use of short-lived, one-time tokens (e.g., 5 min max).

After three to five unsuccessful attempts, lock out codes

4. Steer clear of predictable identifiers

Avoid using numeric or incremental codes (e.g., 000001 → 999999).

Make use of secure random generators or high-entropy UUIDs.

5. Abuse Testing: A Component of Quality Assurance
   Model edge cases for logic and automation.
   Your API test suites should incorporate business flow testing.

🌐 OWASP API Top 10 – #7: Server Side Request Forgery (SSRF)

📖 Definition

An attacker can use API endpoints that accept user-supplied URLs or destinations to trick your server into sending unauthorized requests to internal or external systems. This is known as server side request forgery, or SSRF.

🧨 These queries are frequently designed to gain access to cloud storage, metadata endpoints, internal services, or even malicious websites, thereby converting your server into a proxy.

🔥Why It's Risky

Network boundaries (firewalls, NATs) can be circumvented by SSRF.
Server privileges (such as AWS IAM roles) may be greater than those of the attacker.
permits access to resources that are only available internally.
could result in internal reconnaissance, credential theft, or data exfiltration.

📉 Real-World Case Study: Capital One Breach (2019)

Exploit: A Web Application Firewall (WAF) that was improperly configured allowed an attacker to take advantage of SSRF.
Attack vector: Using its own elevated AWS permissions, the WAF relayed untrusted requests.
Target: S3 buckets and internal AWS metadata services
Effect:
  About 30 GB of stolen information
  More than 100 million credit applications
  More than 100,000 SSNs were exposed.
  Regulatory fine of $190 million
Key flaw: The main weakness was that the server (WAF) was deceived into using AWS internal services, and its IAM permissions were abused.

🛑 What SSRF Can Lead To

  Internal port scanning
  Reading files on internal systems (e.g., /etc/passwd)
  Accessing cloud instance metadata (http://169.254.169.254 in AWS)
  Accessing internal admin APIs
  Bypassing firewall rules or NAT isolation

✅The Best Ways to Avoid SSRF 🔒

1. Use the Least Privilege Principle
   Avoid giving your WAFs or API server general IAM roles.
   If not specifically needed, block server-to-server requests.
2. Prevent Internal Requests
   Reject all requests made to:
   localhost

127.0.0.1
Internal IP ranges, such as 10.0.0.0/8
IP addresses for cloud metadata (169.254.169.254)

3. Strict Validation of URLs
Only permit domains that are on the whitelist, like linkedin.com.
Reject any input that
includes local file URIs (file://).
has both private and internal IPs.
Refers to unidentified or distorted URLs

4. During testing, simulate SSRF
Make use of resources such as
(With Collaborator) Burp Suite
SSRFmap
Personalized curl scripts
Test endpoints that generate outgoing calls in response to user input to look for SSRF.

5. Turn off DNS rebinding and redirects
Make sure that when you perform user-controlled fetches, your server does not automatically follow redirects.
Keep an eye out for and prevent DNS rebinding attacks.

🏗️ OWASP API Top 10 – #8: Security Misconfiguration

📖 Definition

Any incorrect setup or absence of security controls across your API infrastructure, including servers, headers, frameworks, and permissions, is referred to as security misconfiguration. APIs are left vulnerable by the "everything bucket" of configuration and operational errors.

📉 Real-World Example: Experian API Exposure

Organization: Experian, a consumer credit reporting agency
Use: A private API that only reliable loan partners can access
What Happened:
A partner used the private API of Experian to create a public application.
Users were able to obtain complete credit scores through the API by entering:
Given name
Take note of
Date of Born
A vulnerability
Not requiring authentication or rate limits
The entered date of birth didn't even have to be accurate.
Almost anybody's credit report could be pulled by someone with little information.
Impact
Millions of consumer credit profiles could be made public.
Serious privacy and compliance abuses

⚠️The Reasons Behind Security Misconfigurations:

assumed that auth is not required for "internal" APIs

Infrastructure expansion outpaces security evaluations.
Defaults are never altered.
Without revalidation, APIs are reused across environments.
Inconsistent or unmonitored security procedures.

✅ Top Tips for Avoiding Security Misconfigurations :

1. Hardening of the server
   Turn off any unused ports and services.
   Modify the default credentials
   In production, disable debug/info logging.
2. Implement Secure Headers and HTTPS
   CORS: Specify permitted domains
   HSTS: Make sure that HTTPS is used strictly.
   Content Security Policy (CSP): Restrict the sources of scripts
   Rate-limiting headers: Let clients know how much they can request.
3. All Endpoint Access Control
   Without appropriate authentication, no API should be made available to the general public.
   Verify the backend and frontend access rules.
4. Make Use of Security Templates and Baselines
   Use IaC tools (such as Terraform and Ansible) to automate baseline configurations.
   Throughout deployments, continuously validate them.
5. Automate Testing for Misconfigurations
   Make use of resources such as
   ZAP OWASP
   Burp Suite Pro
   ProjectDiscovery's nuclei
   Lynis and Nikto
   Your CI/CD pipeline should incorporate misconfig scans.
6. Clear Error Management
   Avoid disclosing infrastructure information or stack traces in error messages.
   Provide generic error codes (like 403, 500) with as little information as possible.

🧠Developer Mindset Shift:

Give up thinking, "It's working, ship it."
Begin by asking yourself, "Is it safe if someone pokes at it from all angles?"

🗂️ OWASP API Top 10 – #9: Improper Inventory Management:

📖 Definition

When a company does not keep accurate visibility and control over all of its APIs, it is exhibiting improper inventory management. This includes:
APIs that are known versus unknown (shadow/zombie)
Versioning of APIs (V1, V2, V3, etc.)
Private versus public access
Patch status, retirement, ownership, and documentation

"What you don't know exists cannot be secured."

🚨 Two Fundamental Issues:

Visibility

Are you even aware of which APIs are active in your environment?

Command

Are those APIs secure, required, current, and appropriately decommissioned?

🧟 Common Risks in Inventory Mismanagement:

Shadow APIs -- APIs deployed outside official process, undocumented
Zombie APIs -- Deprecated APIs still running and accessible
Version Drift -- Old API versions (e.g., /v1/) left online and unpatched
Lack of Ownership -- No clear owner for API maintenance/security
Lack of Gateway -- APIs bypass central enforcement (rate limits, auth, logging).

📉 Real-World Breach: Optus (Australia):

Summary of the Breach:

A private API for subscriber data was developed by an internal engineer.
It was made available to the general public online.
Wide access to PII without authentication:
Home addresses, phone numbers, and names
Birthdates and driver's license numbers.

Other Errors:

used incremental IDs to enable automated scraping
Without any security awareness, the API stayed online.

Effect:

9.8 million records were exposed.
An attempt at extortion
examination at the national level (even the prime minister participated).

🧰 Inventory Management Solutions

🧠 1. Governance First

Create cross-functional guidelines for the API lifecycle:

Dev
Safety
Operations
Observance
Danger

🔍 2. Discovery Methods

Approach	Description
Traffic-Based --	Use WAF/API Gateway logs to detect API calls
Code Scanning --	Analyze repositories for exposed endpoints
App Crawling --	Navigate app UIs to surface linked APIs
Dictionary Attacks --	Brute-force common API paths
Ask Engineering --	Establish communication with dev leads—often the fastest route.

3. Use a Centralized API Gateway:
Enforce consistent:
  Authentication
  Rate limiting
  Logging
  Version tracking
4. Version & Retire Responsibly
  Maintain a versioning policy
  Ensure clients migrate to new versions
  Decommission and remove deprecated endpoints.
📋 5. Access & Audit Reviews
Regularly verify:
  Which users/orgs have access to which APIs
  Whether those accesses are still justified.

🧠 Mindset Shift for Teams
Security teams often don't know all APIs—but
engineering teams usually do.
🗣️ Start by talking to your dev leads. Collaboration will dramatically improve visibility and accountability.

🌐 OWASP API Top 10 – #10: Unsafe Consumption of APIs
📖 Definition
When a company uses third-party APIs (from partners, vendors, or cloud platforms, for example) without adequate validation, security review, or trust boundaries, it is introducing risks known as "unsafe consumption of APIs."
🔁 It is the opposite of Inventory Management (OWASP #9) in that you are blindly relying on someone else's APIs rather than paying attention to your own.

🤝The Reasons It's a Serious Risk:
  The majority of contemporary apps rely significantly on external APIs.
  Not every third-party API:
  Are safe
  Verify the input.
  Maintain appropriate access controls
  Misconfigurations or misuse can turn even reliable vendors into attack points.

📉 Real-World Example: Companies House – Script Injection
  Agency: Companies House, UK Government; Situation:
  A user registered a company name that included JavaScript (<script src=...>).
  This carried out XSS payloads on the website of Companies House.
  Even worse, any additional API users who used this data without sanitization were put at risk.
Outcome:
  demonstrated how numerous external consumers can be impacted by inadequate sanitization

at the source.
The person had to rename the business, which is now amusingly called:
The business whose name was used to include HTML Script Tags LTD.

✅ Safe API Consumption Best Practices

1. Keep Track of and Catalog All Third-Party APIs
 Maintain a record of every external API that is being used.
 Incorporate:
 Version of the API
 Vendor/owner
 Make contact
 Auth technique applied
2. Use Third-Party APIs as If They Were Your Own
 Verify every input received from external APIs.
 To stop injection and XSS, enforce output encoding.
 Even for upstream responses, use rate limits and access controls.
3. Examine Third-Party APIs
 When permitted by law, pen test third-party API integrations.
Seek out:
 Vectors of injection
 Issues with authorization
 Leaks of data
4. Request Security Records
 Request from vendors:
 Reports on penetration tests
 Audits of compliance
 Security guidelines for APIs
5. Examine Your Integration Logic
 Make sure you:
 Avoid displaying unescaped third-party data in user interfaces.
 Prevent SSRF and logic injection by preventing external responses from initiating internal actions.

💬 Real Talk: Third-Party ≠ Trusted

🔐 "Just because you didn't build it doesn't mean you can't get burned by it."
Whether it's a CRM, payments API, or government data feed, apply equal scrutiny to how you consume and implement them.