CS-066 · API Security

REST API Security

Protecting HTTP-based application interfaces from abuse, data exposure, and broken access control.

Executive Summary

REST API security is the practice of protecting HTTP-based application interfaces so that only authorized clients can perform allowed actions on the data they are entitled to see. Because APIs expose business logic and data directly, they are a primary target for attackers. Strong REST security combines transport encryption, verified identity, enforced authorization on every request, and careful input and output handling.

What It Is

A REST API is a set of HTTP endpoints that let clients read and change resources using standard methods such as GET, POST, PUT, PATCH, and DELETE. REST API security is the set of controls that keep those endpoints from being abused. It answers three questions on every request: is the connection private, who is the caller, and is that caller allowed to do exactly this to exactly this resource. Unlike a traditional web page, an API returns structured data and trusts the client less, so each request must stand on its own and be independently verified. The server should never assume that because a client reached one endpoint it may reach another, or that an identifier in the request belongs to the caller.

Why It Matters

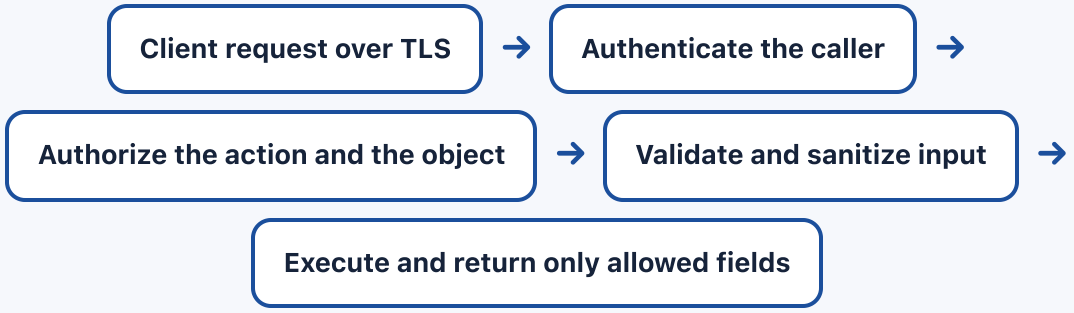
APIs now carry most of the traffic between mobile apps, single-page front ends, partners, and back-end services, which makes them one of the largest parts of an organization's attack surface. A weak API can leak customer records, let one user read or change another user's data, or allow bulk extraction of an entire database through a single overlooked endpoint. These failures are often invisible from the outside until they are exploited, and they frequently pass

functional testing because the feature still works for the intended user. For engineers and security professionals, API security is a high-demand skill because a single broken authorization check can undo every other control.

How It Works

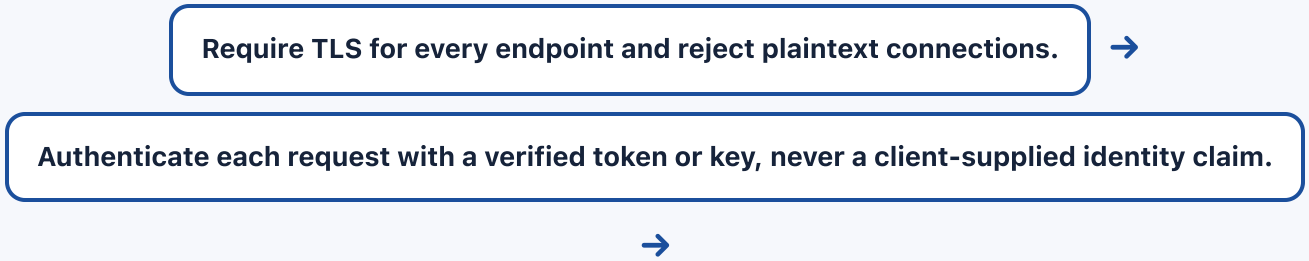
REST security is layered across the request path. The transport layer uses TLS so credentials and data cannot be read in transit. Authentication establishes who the caller is, usually through a token presented on each request. Authorization then decides, per request, whether that caller may perform the action on the specific resource, checking both the type of action (function level) and ownership of the object (object level). Input validation rejects malformed or malicious data before it reaches business logic or a database, and output controls ensure responses return only the fields the caller should see. Rate limiting and monitoring sit alongside these to blunt brute force, scraping, and abuse. Because APIs are stateless, none of these checks can rely on a prior request; each one is re-evaluated every time.

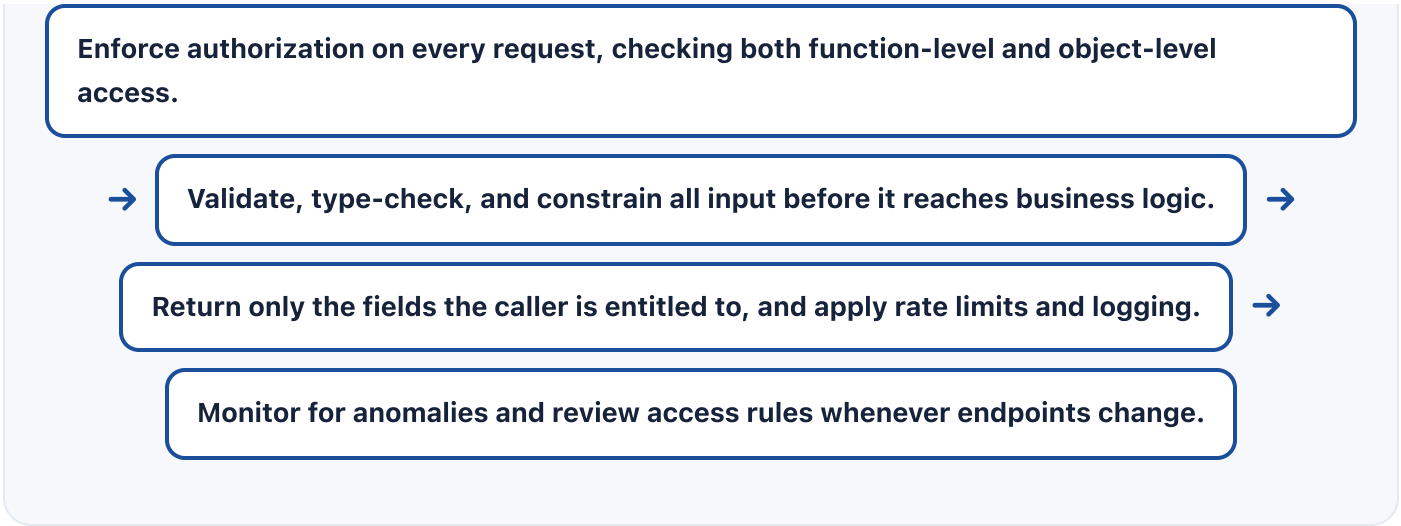
Architecture Diagram



Every REST request passes through independent gates before it can touch data, and each gate is re-checked on every call.

Visual Workflow





Common Attacks

- Broken object level authorization, where changing an ID in the request exposes another user's data
- Broken function level authorization, where a normal user reaches an admin-only action
- Injection through unvalidated input reaching a database or command
- Excessive data exposure, where responses include sensitive fields the client filters but the server sent
- Mass assignment, where a request sets fields the client should not control
- Credential stuffing and brute force against weakly protected authentication endpoints

Common Mistakes

- Checking that a user is logged in but not whether they own the specific record
- Trusting an object identifier in the request without verifying ownership
- Relying on the front end to hide fields or buttons instead of enforcing rules on the server
- Returning full database objects and letting the client decide what to display
- Leaving verbose error messages or stack traces enabled in production
- Exposing internal or debug endpoints alongside public ones

Best Practices

- Enforce TLS everywhere and disable weak protocols and ciphers

- Authenticate with short-lived tokens and validate them on every request
- Apply least-privilege, object-level authorization checks on every endpoint
- Validate input against a strict schema and reject anything unexpected
- Return only necessary fields and define responses explicitly rather than dumping objects
- Apply rate limiting, log security events, and follow the OWASP API Security Top 10 as a baseline

Quick Checklist

- [x] TLS required on all endpoints, plaintext rejected
- [x] Every request authenticated with a validated token
- [x] Object-level and function-level authorization enforced server-side
- [x] Input validated against an explicit schema
- [x] Responses limited to allowed fields, no full-object dumps
- [x] Rate limiting, logging, and error handling reviewed before release

Recommended Tools

API gateway
Centralizes authentication, rate limiting, and routing in front of services

Web application firewall (WAF)
Filters malicious requests before they reach the API

API security testing tool
Scans endpoints for broken authorization, injection, and exposure

Schema validation library
Enforces strict input and output shapes at the code level

Industry Standards

OWASP API Security Top 10
The definitive list of the most critical API security risks

OWASP Application Security Verification Standard (ASVS)
Verification requirements that apply to API endpoints

NIST SP 800-95
Guidance on securing web services and interfaces

Career Relevance

REST API security is core to application security engineers, API security engineers, and backend engineers, and it is increasingly expected of any developer who ships endpoints. Security engineers and penetration testers spend much of their time probing APIs, and GRC and AI governance professionals need to understand these risks because so many systems, including AI services, are exposed as REST APIs.

Interview Questions

- What is broken object level authorization, and how do you prevent it?
- Why must authorization be checked on the server for every request rather than in the client?
- How do you prevent mass assignment and excessive data exposure in API responses?
- What is the difference between authentication and authorization in an API context?
- How would you protect an authentication endpoint from credential stuffing?

Related Certifications

OWASP-aligned application security training (GWAPT)

GIAC Web Application Penetration Tester

CompTIA Security+ (foundational)

Further Reading

- OWASP API Security Project
- OWASP Application Security Verification Standard
- NIST Publications

Key Takeaways

- REST security verifies transport, identity, and authorization on every single request.

- Broken object-level and function-level authorization are the most damaging and common failures.
- Never trust the client to enforce rules or filter sensitive fields.
- Validate input strictly and return only the fields the caller is entitled to.
- The OWASP API Security Top 10 is the practical baseline for securing REST APIs.

FAQ

Is HTTPS enough to secure a REST API?

Why is broken object level authorization so common?

Where should a team start with API security?

Related Careers

RELATED ROLES

Application Security Engineer

Api Security Engineer

Backend Engineer

Security Engineer

RELATED CERTIFICATIONS

OWASP-aligned application security training

GIAC Web Application Penetration Tester (GWAPT)

CompTIA Security+ (foundational)

CURRENT OPENINGS

Live roles are on the job board.

Browse all jobs

GUIDES & SALARY

Career guides

Role roadmaps

Salary data

Career tools

SUGGESTED LEARNING PATH

1 Ground the basics with CS-001 Cybersecurity

2. Study this sheet: **REST API Security**
3. Go deeper: JSON Web Tokens (JWT)
4. Go deeper: OAuth 2.0
5. Validate it: work toward **OWASP-aligned application security training**
6. Find the role: browse current openings

RELATED SHEETS

CS-067 JSON Web Tokens (JWT) | CS-068 OAuth 2.0 | CS-070 API Rate Limiting | CS-062 Authentication | CS-063 Authorization