CODELIVLY

Hacking Your First APIs

The Complete Guide to REST, GraphQL, JWT, OAuth 2.0, API Pentesting, the OWASP API Security Top 10, and Secure API Development.

Ethical Hackers | Penetration Testers | Bug Bounty Hunters | API Developers | Security Engineers | DevSecOps | Beginner → Advanced

Codelivly

www.codelivly.com

Hacking Your First APIs

The Complete Guide to REST, GraphQL, JWT, OAuth 2.0, API Pentesting, the OWASP API Security Top 10, and Secure API Development.

First Edition

Author: Codelivly
Website: www.codelivly.com

© Codelivly. All Rights Reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in reviews and certain other non-commercial uses permitted by copyright law.

All trademarks, product names, tool names, and company names mentioned in this book are the property of their respective owners and are used for identification and educational purposes only.

> **LEGAL & ETHICAL DISCLAIMER.** This book is intended solely for educational purposes and for authorized security testing. The techniques described must only be used against systems you own or for which you have explicit, written authorization to test. Unauthorized access to computer systems and APIs is illegal in most jurisdictions and may carry severe civil and criminal penalties. The author and Codelivly accept no liability for misuse of the material, for any damage caused, or for any laws broken by readers. Always operate within the boundaries of a signed scope, engagement letter, or bug bounty program policy.

Codelivly • www.codelivly.com

PREFACE

So You Want to Hack APIs

Here's a secret the tech industry doesn't advertise: the app on your phone is a liar. The pretty buttons, the greyed-out fields, the "you don't have permission" messages — none of that is real security. It's theatre performed by the client. Behind it sits the *real* system: the API. And the API will happily talk to anyone who knows how to ask. That gap — between what the app pretends and what the API actually allows — is where hackers live.

This book teaches you to live there. You will learn to ignore the app entirely, speak to the API directly, and make it do things its builders never intended: hand you other users' data, promote you to admin, fetch secrets from inside the cloud, and cough up records it should have guarded. You'll do it with the same tools and techniques used by professional penetration testers and the best bug bounty hunters on the planet — and you'll understand *why* each attack works, so you can find the next one nobody has documented yet.

It starts from zero. You do not need to be a programmer or a security expert. You need curiosity, a willingness to poke at things until they break, and the discipline to do it only where you're allowed. By the end you'll think like an attacker by reflex — reading an API and instantly seeing the weak seam, the missing check, the door left unlocked.

How this book attacks the subject

We build fast. First the fundamentals — what an API is, how HTTP carries it, how authentication and authorization are *supposed* to work (so you can spot where they don't). Then the main event: the ten flaws behind most real breaches, each with a hands-on lab where *you* pull off the attack. From there it only gets sharper — GraphQL, JWT and OAuth abuse, a full attacker's methodology, chained real-world exploits, then the specialist surfaces: cloud, AI/LLM, mobile, and more. You also learn to defend, because the best attackers understand defense cold — and because that's how you get paid to do this.

Read it like a hacker

Don't just read — *do*. Every time you hit a *Hands-on Lab*, stop and run it. Appendix I gets you a free, legal hacking lab running in one sitting; break everything in it. Reading teaches you the theory; your hands teach you the reflex. The distance between "I understand BOLA" and "I find BOLA in ninety seconds" is measured in repetitions, not pages.

The only rule that matters

These skills are real, and so are the consequences. Use them **only** on systems you own or have explicit written permission to test — your lab, or an in-scope bug bounty program. That single line is the difference between a career and a criminal record. Hack ethically, and this becomes one of the most rewarding skills you'll ever own. Now let's go break something.

Codelivly — www.codelivly.com

Contents

PART I — API FUNDAMENTALS

1 · The World of APIs 9

2 · HTTP, HTTPS & the Web API Stack 16

3 · API Styles: REST, SOAP, GraphQL, gRPC & WebSockets 20

4 · Authentication, Authorization & the API Lifecycle 23

PART II — REST API SECURITY

5 · Designing & Securing RESTful Resources 28

6 · Validation, Errors, Pagination & Versioning 31

PART III — AUTHENTICATION

7 · From Basic Auth to Bearer Tokens 36

8 · Sessions, Cookies, MFA & Token Lifecycles 39

PART IV — OWASP API SECURITY TOP 10

9 · The OWASP API Security Top 10 (2023) 43

PART V — GRAPHQL SECURITY

10 · Attacking & Defending GraphQL 65

PART VI — JWT SECURITY

11 · JSON Web Tokens In Depth 71

PART VII — OAUTH 2.0 SECURITY

12 · OAuth 2.0 & OpenID Connect 77

PART VIII — API PENTESTING METHODOLOGY

13 · A Repeatable API Pentest Methodology 82

PART IX — SECURE API DEVELOPMENT

14 · Building APIs That Resist Attack 90

PART X — CLOUD-NATIVE API SECURITY

15 · Microservices, Gateways & Zero Trust 95

PART XI — PROFESSIONAL REPORTING

16 · Risk Rating, Evidence & Report Writing 99

PART XII — ADVANCED AUTHENTICATION & IDENTITY

17 · Request Signing, HMAC & Machine Identity 103

18 · SSO, SAML, Federation & Passkeys 106

PART XIII — INJECTION ATTACKS IN DEPTH

19 · SQL & NoSQL Injection in APIs 110

20 · Command Injection, SSTI, XXE & Deserialization 114

PART XIV — DENIAL OF SERVICE & RESILIENCE

21 · Resource Exhaustion & Rate Limiting 118

PART XV — WEBHOOKS & API SUPPLY CHAIN

22 · Webhooks, Integrations & Third-Party Trust 123

PART XVI — MOBILE & IOT API SECURITY

23 · APIs Behind Mobile Apps & Devices 127

PART XVII — AUTOMATION & DEVSECOPS

24 · Securing APIs in the CI/CD Pipeline 131

PART XVIII — CASE STUDIES & PATTERNS

25 · Learning From Real-World API Breaches 135

PART XIX — API THREAT MODELING

26 · Thinking Like an Attacker, by Design 139

PART XX — PROGRAM & GOVERNANCE

27 · Building an API Security Program 144

PART XXI — EMERGING & SPECIALIZED SURFACES

28 · Securing AI & LLM APIs 149

29 · gRPC, WebSockets & Event-Driven APIs 153

PART XXII — THE PRACTITIONER'S TOOLKIT

30 · Intercepting Proxies: Burp Suite & mitmproxy 158

31 · Discovery, Clients & Automation 161

PART XXIII — COMPLIANCE & STANDARDS

32 · APIs, Regulations & Audit 165

PART XXIV — API INCIDENT RESPONSE

33 · When an API Is Breached 169

PART XXV — FRAMEWORK-SPECIFIC HARDENING

34 · Hardening Node, Spring, Django & .NET 174

PART XXVI — THE HACKER'S PLAYBOOK

35 · The Attacker's Mindset & the API Kill Chain 179

36 · Chaining Bugs: A Real-World Attack Walkthrough 182

37 · Bypasses & Evasion: Getting Past the Defenses 186

PART XXVII — ADVANCED OFFENSIVE TECHNIQUES

38 · Breaking Access Control, Deeply 190

39 · Breaking Authentication: JWT & OAuth Attack Chains 193

40 · Injection & SSRF: From Input to Impact 196

APPENDICES

A–J · Cheat Sheets, Checklists, Glossary, Sample Report, Lab & Careers 200

PART I

API Fundamentals

Before you can attack or defend an API, you must understand what an API really is, how HTTP carries it, and the vocabulary — methods, status codes, headers, content types, authentication, authorization — that every later chapter assumes. This part builds that foundation from zero.

CHAPTER 1

The World of APIs

> *An API is a contract between two pieces of software — and like every contract, it's only as good as what actually gets enforced. Understanding that contract (what it promises, what it hides, and, crucially, where it blindly trusts) is the root of every attack you'll ever land and every defense you'll ever build. Learn to read the contract and you learn to see where it can be broken.*

1.1 What Is an API?

API stands for *Application Programming Interface*. It is a defined way for one program to request something from another program without knowing how that other program works internally. When your phone's weather app shows the forecast, it did not compute the weather. It sent a request over the internet to a weather company's server and received a structured answer. That request-and-answer channel is an API.

Three ideas make the concept click:

- **Interface (the WHAT).** An API exposes a menu of things you are allowed to ask for — "give me user 42", "create an order", "list my messages" — while hiding the kitchen. You order from the menu; you never walk into the kitchen.
- **Contract (the PROMISE).** The API promises that if you send a request in the agreed format, you will get back an answer in an agreed format. Both sides depend on that promise.
- **Abstraction (the HIDING).** The provider can rewrite the entire kitchen — change databases, languages, servers — and as long as the menu stays the same, your app keeps working. This is powerful, and it is also where security assumptions hide.

WHY THIS MATTERS FOR SECURITY

Every API is a doorway into a business. The menu tells attackers exactly what the business can do. Unlike a website designed for humans, an API is designed for machines, which means it is predictable, scriptable, and often *under-protected* because developers assume "only our app will call it." That assumption is the single most exploited belief in API security.

1.2 The Restaurant Analogy (WHY it works this way)

Imagine a restaurant. You (the **client**) sit at a table. You cannot enter the kitchen (the **server / backend**). The **waiter** is the API: they take your order in a language both sides understand, carry it to the kitchen, and bring back a plate. The **menu** is the API documentation. The kitchen can hire new chefs or buy a new oven — you never notice, because the menu (the contract) is unchanged.

Now translate every part into web terms:

Table 1.1 — The restaurant analogy mapped to web APIs

Restaurant	Web API	Security relevance
Customer	Client (browser, mobile app, script)	Can be impersonated or automated by an attacker
Waiter	The API endpoint over HTTP	The attack surface — every request passes here
Menu	API documentation / Swagger / OpenAPI	A roadmap for attackers if exposed
Order ticket	HTTP request (method, path, headers, body)	Every field is attacker-controllable input
Kitchen	Backend logic + database	The prize; must never trust the ticket blindly
The plate	HTTP response (status + body)	May leak more data than intended

1.3 A Real Request, End to End

Let us make it concrete. Here is what actually travels across the wire when an app asks for a user profile. This is the atomic unit of everything in this book — learn to read it fluently.

HTTP REQUEST

```
GET /api/v3/users/42 HTTP/1.1
Host: api.shop.example
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Accept: application/json
User-Agent: ShopApp/2.1 (iOS)
```

HTTP RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
```

```
{
  "id": 42,
  "name": "Aditi Sharma",
  "email": "aditi@example.com",
  "role": "customer"
}
```

Read it top to bottom. The client used the `GET` method to ask for the resource at path `/api/v3/users/42`. It proved who it was with an `Authorization` header carrying a token. The server answered `200 OK` and returned a JSON object. Four things a security tester immediately asks about this exchange:

- **The `42` in the URL.** What happens if I change it to `43`? (Chapter 9, BOLA.)
- **The token.** Is it validated properly? What is inside it? (Chapter 11, JWT.)
- **The response fields.** Does a customer need to see `role`? Could there be hidden fields like `isAdmin`? (Chapter 9, BOPLA.)
- **Rate.** Can I send this a million times? (Chapter 9, Unrestricted Consumption.)

> **HABIT TO BUILD NOW**
>
> For every request you ever see, silently ask: "*What is the user-controlled input, and what does the server trust about it?*" Ninety percent of API vulnerabilities live in the gap between those two questions.

1.4 Where APIs Live: The Modern Architecture

A single mobile tap may trigger a dozen internal API calls. The diagram below shows the typical path of a request through a modern system — the same path an attacker's request travels.

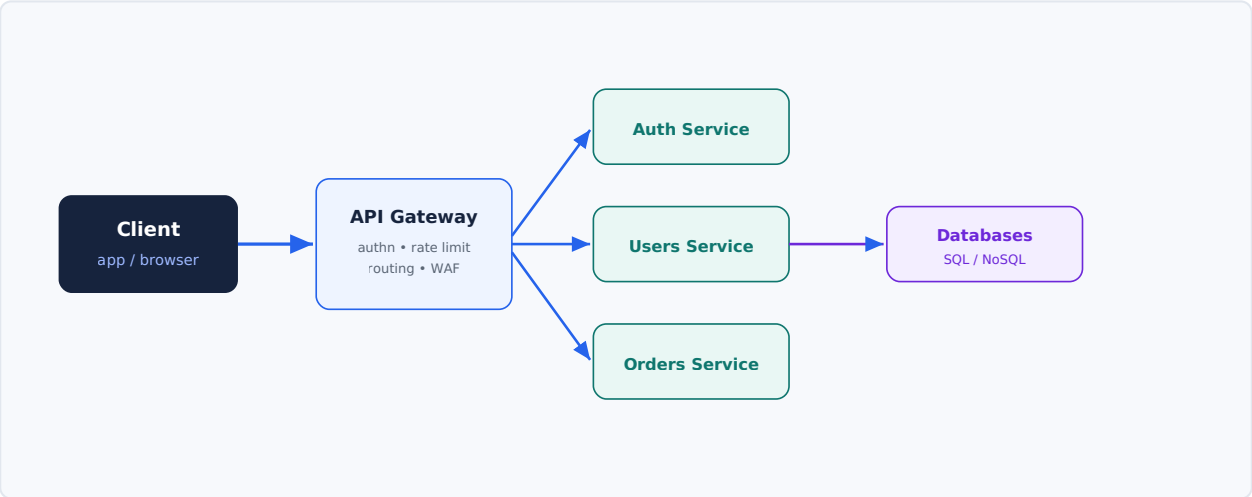


Figure 1.1 — A request flows from client → gateway → microservices → data stores. Each hop is a trust boundary and therefore a place a security control can be enforced or forgotten.

Two internal **trust boundaries** matter enormously. The first is the gateway: the last place the public internet touches before the request enters your network. The second is the boundary *between* services. Many teams protect the first boundary heavily and assume everything past it is trusted — so a request that slips past the gateway, or originates inside the network (SSRF, Chapter 9), can call internal services with no further checks.

1.5 Talk to an API Yourself (Your First Requests)

Reading about requests is one thing; sending them is another. The moment you fire your own request by hand, the mystery evaporates — you realize an API is just text you send and text you get back, and *you* control every character you send. Here is the exact same profile request, sent three ways a hacker actually uses.

CURL — THE UNIVERSAL TOOL

```
curl -X GET https://api.shop.example/api/v3/users/42 \
     -H "Authorization: Bearer <token>" \
     -H "Accept: application/json"
```

HTTPIE — FRIENDLIER SYNTAX

```
http GET api.shop.example/api/v3/users/42 \
     "Authorization: Bearer <token>"
```

Or point-and-click it in **Postman**: set the method to `GET`, paste the URL, add the `Authorization` header, hit Send. Same request, prettier buttons. The tool does not matter — what matters is the realization underneath it:

> **THE REALIZATION THAT MAKES YOU A HACKER**
>
> Every part of that request — the method, the path, the `42`, every header, every field in the body — is **yours to change**. The app normally sends "nice" values, but nothing physically stops you from sending different ones. The server has to *decide* what to do with whatever you send. Security is entirely about those decisions — and this whole book is about the decisions developers get wrong.

1.6 The Same Request, an Attacker's Eyes

A normal user sees "my profile loaded." A developer sees "the users service returned a record." A hacker sees a menu of experiments. Let's re-read `GET /api/v3/users/42` the way you'll learn to read every request in this book:

Table 1.2 — One request, six attacker questions

What you see	What you try	If it works…
`/users/42`	Change 42 → 43, 1, 9999	You read other people's profiles (BOLA)
The `Bearer` token	Remove it; tamper it; decode it	Broken auth / JWT forgery
`"role":"customer"`	Look for hidden fields; try to set them	Property-level abuse (BOPLA)
The endpoint path	Try `/users/42/orders`, `/admin/users`	Hidden functionality (BFLA)
The whole request	Send it 10,000 times fast	Resource abuse / brute force
Any URL-like field	Point it at an internal address	Server-side request forgery (SSRF)

Notice that *every* line is just "what did the developer assume, and what if I don't cooperate?" That single reflex, applied relentlessly, is 90% of API hacking. The remaining 10% is knowing the specific techniques — which is the rest of this book.

1.7 JSON in Sixty Seconds

APIs mostly speak **JSON** (JavaScript Object Notation) — a simple, human-readable way to represent data. If you can read this, you can read JSON:

JSON BASICS

```
{
  "id": 42,                      // a number
  "name": "Aditi",               // a string (in quotes)
  "verified": true,              // a boolean: true / false
  "roles": ["customer"],         // an array (list) in [ ]
  "address": { "city": "Pune" }  // a nested object in { }
}
```

That's the whole language: objects in `{ }`, lists in `[ ]`, and values that are strings, numbers, booleans, or `null`. Why does a hacker care about the shape? Because **types are a trust boundary**. When code expects the string `"secret"` and you send the object `{"$ne": null}` instead, you can break authentication (Chapter 19). The first thing you'll learn to do is send data in a shape the developer never expected.

1.8 Your First API Hack

Enough theory — let's land one. This is the entire loop of API hacking in its simplest form, the exact move that has earned real money on bug-bounty platforms

thousands of times. You will do this for real in the lab (Appendix I); here, walk through the logic.

WALKTHROUGH · BOLA

The setup. You register on a site and open your own profile. Your proxy shows the app quietly making this call:

STEP 1 — OBSERVE YOUR OWN REQUEST

```
GET /api/v1/users/1337/profile HTTP/1.1
Authorization: Bearer <your_token>

200 OK  →  { "id":1337, "name":"You", "email":"you@mail.com" }
```

The idea. That `1337` is *your* user ID. The request basically says "give me user 1337's profile." A dangerous question forms: what if I ask for a *different* user?

STEP 2 — CHANGE ONE NUMBER

```
GET /api/v1/users/1336/profile HTTP/1.1
Authorization: Bearer <your_token>

200 OK  →  { "id":1336, "name":"Someone Else", "email":"victim@mail.com" }
```

The win. The server returned a stranger's profile to *you*. It checked that you were logged in, but never checked that user 1336 was actually *your* account. Now loop from 1 upward and you can read the entire user base. That is **BOLA** — Broken Object Level Authorization — the single most common serious API bug in the world (Chapter 9), and you just understood it on page one of your journey.

> **THE LINE YOU NEVER CROSS**
>
> That move is trivial to perform and devastating in impact — which is exactly why doing it without permission is a crime, not a hobby. Practice it in your own lab or on a program that invites it. The skill is the same either way; only the authorization separates a bug-bounty payout from a police report. Every hacker in this book stays on the right side of that line.

If that gave you a small jolt of "wait, it's *that* easy?" — good. That feeling is why people fall in love with this craft. The rest of the book turns that one trick into a hundred, and teaches you to fix them all too.

REVIEW QUESTIONS

1. In one sentence, what is the difference between the *interface* and the *implementation* of an API?
2. Using the restaurant analogy, which component is the attack surface and why?
3. List the four questions a tester asks when reading the request in §1.3.
4. What is a "trust boundary," and why is the boundary *between* internal services often weaker than the gateway?
5. Which parts of an HTTP request can an attacker change, and why does that matter?
6. In your own words, what is BOLA, and what single check would have stopped the hack in §1.8?
7. Why is the *type* of a JSON value (string vs object) a security concern?