Securing and Designing APIs

Three days on APIs that describe themselves and defend themselves. Design and security are taught together because they fail together — and because the newest consumer of your API is an agent that will do exactly what the interface permits, at machine speed.

Why this, and why now

APIs are the leading breach vector, and the reasons are structural rather than careless. There are thousands of endpoints where there used to be a handful. They are machine-consumable by design, and their own descriptions make them self-documenting attack surfaces. Most of them authenticate; far fewer authorize. And the protections that guarded the web application in front of them frequently do not apply to the API behind it.

Into that landscape has arrived a new kind of client. An agent does not read the wiki, cannot ask the team what an endpoint is *really* for, and does not get tired. It reads the description it is given and acts on it. That makes two old virtues newly urgent:

A self-describing API is one an agent can use correctly. Clear resource identity, honest HTTP semantics, typed representations and links that say what can happen next were always good design. They are now the difference between an agent that is following the interface and one that is guessing at it. APIs and self-describing data are more useful than they have ever been.

The security mechanisms are how you connect an agent safely. An agent holding a user's long-lived bearer token can do anything that user could ever do, for as long as the token lives, on the say-so of whatever text it read last. The mechanisms in this course — short-lived tokens scoped to a task, tokens bound to the client that holds them, grants that carry the exact transaction, credentials that can be narrowed but never widened — were built for other reasons. They turn out to be precisely the tools for handing an agent enough authority to be useful and no more.

Three days, six half-day sessions. Day one is design, day two is security, and day three is operations and agents. Each day is independently deliverable, and the sequence is the recommended one: it is difficult to secure an interface whose resources were never clearly named.

One API, carried through all three days

Every exercise runs against a service we supply: small, readable, installed with one command and needing no external accounts. It ships with an identity provider, a gateway and a set of clients, and it is seeded with design decisions and defects chosen to produce specific realizations at specific moments. Nothing in it is labeled as an exercise.

Bringing a real API fails in predictable ways — it cannot be run in a classroom, it cannot be attacked in a classroom, and what is found in it cannot be discussed. A planted authorization flaw produces the same realization in every room. Participants carry the same review back to their own interfaces between sessions, and a checklist for doing so is part of the handout.

Day 1 · Design

DAY 1 | TWO HALF-DAYS | WORKING HTTP KNOWLEDGE ASSUMED

For engineers and architects who build or consume HTTP APIs and have inherited more conventions than reasons. The day supplies the reasons, so that the next design argument can be settled on something other than taste.

THE SESSIONS

1 · Resources, Names and the Uniform Interface. What the Web's architecture actually asks for, read through the Richardson maturity levels: identity first, then interaction, then representation. Naming resources so the names survive a reorganization. HTTP semantics as a contract with every intermediary between client and server — safety and idempotency, conditional requests, caching — and what breaks when an API quietly violates them. Errors a client can act on. Pagination, filtering, long-running operations and retries that do not double-charge anybody.

Exercises: Name the Resources — a supplied feature brief, modeled twice · The Retry That Charged Twice.

2 · Representations, Description and Change. Content negotiation and the separation of a resource from its representations. Hypermedia as the client being told what it may do next rather than being compiled to assume it. Description formats — OpenAPI, JSON Schema, JSON-LD — and what each can and cannot promise to a consumer that is a program rather than a person. Evolving an API without a version number in the URL, and knowing when you need one anyway. When REST is the wrong answer: GraphQL, gRPC and events, chosen on their merits rather than their novelty. The API as a product with an inventory, because **you cannot secure what you cannot catalog**.

Exercises: Break the Client — ship a change and see which consumers survive · Describe It for a Stranger — the description is handed to an agent, and the room watches what it does with it.

Day 2 · Security

DAY 2 | TWO HALF-DAYS | NO SECURITY BACKGROUND ASSUMED

For developers, who own more of this than their organizations usually admit. Security drifted away from development teams for historical reasons, and most API vulnerabilities are still written, and best fixed, in application code. The day is built around the question that finds the bugs: *where, exactly, is the check?*

THE SESSIONS

3 · Threats, Transport and Identity. Why APIs get breached, from the incident record rather than from a list: broken object-level authorization, authentication without authorization, misconfiguration, secrets in mobile apps and single-page applications. Architectural risk assessment as a business-level decision tool. Defense in depth and least privilege as design constraints rather than slogans. TLS as it is now deployed — 1.3, forward secrecy, the hybrid post-quantum key exchange already on the wire — and how to find out what a connection is actually using. Identification, authentication and authorization, which are three words and three problems. OAuth 2.0 as it was published, what was removed on the way to 2.1 and why, PKCE and what it does not protect against, and redirect handling. OpenID Connect as the identity layer OAuth needed: the access token against the ID token, the claims that matter, and why you never key an account on an email address.

Exercises: Threat-Model the Supplied API · Walk the Flow — an authorization code exchange traced by hand, then broken on purpose.

4 · Tokens, and How They Go Wrong. Hashes, MACs and why `hash(secret + message)` is not one. JWTs and their four recurring failures — `alg: none`, algorithm confusion, weak symmetric secrets, and the claims nobody checked — with a real incident in which a signing key validated tokens it should never have been trusted for. The bearer problem: whoever holds the token *is* the user. Sender-constrained tokens through mutual TLS and DPoP. HTTP message signatures, and what signing the request buys that transport security does not. PASETO as the opinionated alternative when you control both ends. Macaroons, where anyone holding a credential can narrow it and nobody can widen it. Rich Authorization Requests, which let a grant carry the exact amount, payee and action instead of a scope the user has to trust. Passkeys and WebAuthn, why phishing cannot work against them, and why your recovery path is your real security floor. Sessions, refresh tokens, and when a cookie is still the right answer.

Exercises: Spot the Bug — token verification code that reads as correct and is not · JWT Inspector · Build the OIDC Verifier.

Day 3 · Operations and Agents

DAY 3 | TWO HALF-DAYS | DAY 2 OR EQUIVALENT

For the engineers and leads who will be asked to connect an agent to something that matters. The morning covers what keeps an API secure after launch. The afternoon spends everything from the first five sessions on the question most teams are now facing without a method.

THE SESSIONS

5 · Service-to-Service, Authorization and Operations. The question nobody asks early enough: once a request passes the gateway, who does each service believe it is talking to? Workload identity with mutual TLS and SPIFFE. Carrying the user's identity across service boundaries without handing every service the user's full authority — token exchange, audience restriction, and the confused deputy. Authorization beyond authentication: where role checks are fine, where the model breaks, and externalizing the decision to policy that can be tested and audited. Then the operational half, because cryptography has a lifecycle: where keys live, envelope encryption, rotation designed in rather than bolted on, certificate automation as lifetimes shrink, revocation and why it disappoints, and crypto-agility — algorithms as data, not code — ahead of the post-quantum migration.

Exercises: Where Is the Check? — trace one request through four services · A Leaked Signing Key — a tabletop incident, from the first message through the two weeks of work that follow.

6 · Connecting Agents Safely. What is genuinely different when the client acts on its own: instructions arriving inside data, goal manipulation, excessive agency, and the confused deputy at hyperscale. The organizing rule — **trust the architecture, not the model.** A tool that cannot be called does not need to be refused; a credential that cannot be reached does not need to be guarded; an action that cannot commit does not need to be undone. Then the mechanisms from days one and two, applied:

- **Capability scoping.** Each tool call carries a short-lived token bound to this user, this task, this resource and this time window, issued by token exchange with the intent captured. A compromise yields what the agent *currently holds*, not what the user *could ever hold*.
- **Transactional grants.** Rich Authorization Requests, so that a human approves the specific action rather than a category of actions.
- **Attenuation.** Credentials an agent can narrow before delegating to a sub-agent, and that nothing downstream can widen.
- **Sender constraint.** Tokens bound to the holder, so that one leaked through a prompt, a log or a transcript is useless elsewhere.
- **Affordance equals authorization.** A tool list shaped by the agent's capability, so that what it is not permitted to do it cannot see. The self-describing API from day one is what makes this possible: the description *is* the tool surface.

Approval gates matched to reversibility, and the fatigue that erodes every one of them. Traces, kill switches and replay, because every ring of defense will fail and the plan has to include what happens next.

Exercises: Scope the Agent — the supplied API is connected to an agent with a user's session token, and then re-connected properly; the difference in blast radius is measured rather than asserted · Design Review — a proposed agent integration, reviewed against the five rings.

> **What this course is not.** It is not penetration-testing training and it is not a product course for a gateway or an identity provider. The standards and the reasoning are taught because they are the portable part; the supplied identity provider is there to be inspected, not endorsed.

Who should come

Software engineers, technical leads and architects who build, review or consume HTTP APIs. Working familiarity with HTTP and JSON is assumed. No security background is assumed, and no cryptographic mathematics is required — the material is about choosing, configuring and verifying, not about implementing primitives. Security engineers who want the developer's view of the same ground are well served by days two and three.

Practical requirements

What participants need. A machine they can install and run code on, a command line, and an HTTP client they are comfortable with. The supplied service and its identity provider run locally, with no external accounts and no credentials. Day three uses whichever agent tooling the participant's organization already permits.

What is supplied. The exercise service and its clients; a participant handout for each session; an API review checklist covering design, authentication, authorization and agent access, for use on the participant's own interfaces; and a facilitator key for each session.

Format. Online or in a room. Exercises are solo, with discussion in chat or open floor; each half-day runs about four hours including two breaks.

Adapting it. The syllabus is a starting point rather than a catalog item. It can be run as a two-day security course, weighted toward design for a platform team, or paired with *Linked Data in the Time of AI*, the other half of the same argument.

Next steps. This syllabus is a starting point. Courses are assembled per client — the emphasis, the examples and the length all move to fit the team in the room. To request more detail, or to schedule a discussion about running this course, get in touch at bosatsu.net/contact.html.

Brian Sletten · Bosatsu Consulting, Inc.
API Design & Strategy · REST · Security · OAuth · OIDC · zero trust · Encryption · agents