— CHAPTER 17 · SMART ON FHIR 2.2 · ZERO-TRUST RBAC

Authentication is not authorization. A FHIR server has to prove both.

We built and seeded the server. Now the hardest question - who may read this patient's blood pressure, and who may not. Authentication asks who you are. Authorization asks what you may touch. Confuse the two and you leak protected health information. SMART on FHIR answers both - and it is live regulatory pressure, with the TEFCA FAST security deadline landing in January 2026.

What we will cover

- App launch & the OAuth2 / OIDC flow
- The version-two scope grammar
- The zero-trust enforcement chain

— CONCEPT · SMART APP LAUNCH 2.2.0 - OAUTH2 + OIDC, PROFILED FOR HEALTHCARE

Four players. The FHIR server never sees a password - only a token.

Ava & Marcus

SMART 2.2.0 · STU on R4

OAuth2 plus OpenID Connect, profiled for healthcare. Version 2.2.0 is the current trial-use release - it carries the granular v2 scopes we are about to meet.

The four parties

SMART is OAuth2, with healthcare roles

4 **User**
Clinician or patient - grants consent

— DIAGRAM · AUTHORIZATION-CODE FLOW - PKCE MANDATORY FOR EVERY CLIENT

How the app earns its token - code first, then exchange.

Ava & Marcus

● PKCE · S256

Because in 2026 PKCE is mandatory for all clients - even confidential ones. It defends the code if it is ever intercepted.

The code-then-exchange dance

Two round-trips to the authorization server

4 **Access token (+ ID token)**
openid fhirUser → identity layer

— CONCEPT · LEAST PRIVILEGE, WRITTEN AS A STRING

A v2 scope reads like a sentence - context, resource, permissions.

— THE GRAMMAR

`{context}/{Resource}.{cruds}?params`

cruds spells out five permission letters - c r u d s - create, read, update, delete, search. The context says whose data, the params narrow it further.

- patient/Observation.rs - read + search this patient's observations
- user/*.cruds - full access to everything the user can reach
- patient/Observation.rs?category=vital-signs - vitals only

— THE C-R-U-D-S LETTERS

c create · **r** read · **u** update · **d** delete · **s** search. Combine only the letters an app actually needs - that is least privilege by construction.

— THE THREE CONTEXTS

patient/ just this patient · **user/** everything the logged-in user may see · **system/** backend services, no user in the loop.

— COMPARISON · V1 COARSE VS V2 GRANULAR - AND THE RULE THAT BITES

Read-everything became read exactly what you need.

Version-one clinical scopes were coarse - read-all or write-all. Version two replaces them with per-letter cruds, so you grant exactly what an app needs and nothing more. A server may negotiate or downgrade - but never mix the two grammars in a single request. They are distinct.

— V1 · DEPRECATED

`patient/Observation.read`

`patient/Observation.write`

Coarse - read everything or write everything. HTI-1 and US Core 7 deprecate dot-read and dot-write for ONC-certified systems. Non-compliance is a certification failure.

— V2 · LEAST PRIVILEGE

`patient/Observation.rs`

Read + search, narrowed with

`?category=vital-signs`

Per-cruds letters grant exactly the access an app needs. In 2026 version two is not a nicety - it is the requirement.

- Never mix v1 and v2 grammars in one authorization request
- v2 is least-privilege by construction

— CONCEPT · HOW THE APP LEARNS WHICH PATIENT IS ON SCREEN

Same scopes, two doorways - EHR launch and standalone.

When the app opens, how does it know which patient is in front of the clinician? That is launch context, and there are two ways in. The difference is whether the host EHR hands the app a session, or the app starts cold and asks for what it needs.

— EHR LAUNCH

Launched from inside the record

The app receives a `launch` parameter tied to the EHR session and requests `launch/patient`. The authorization server resolves context and returns the patient and encounter ids alongside the token - plus an array of typed context references, an intent, and a styling url so the app matches its host.

— STANDALONE LAUNCH

The app starts cold

No `launch` parameter. The app requests the context scopes it needs directly, and the user picks the patient during login.

- fhirContext = typed context references, not a single named field
- Same scope grammar, different entry point

— WALKTHROUGH · FRAMEWORK ANCHOR - WHERE AUTHORIZATION LANDS

The scope string literally becomes a guard rule.

Ava & Marcus

PRE_HANDLED seam · Ch 10

The resource server. In HAPI an authorization interceptor - configured with a rule-builder pattern. Conceptually you write rules like allow read of Observation inside this patient's compartment, and the scope string drives which rules apply.

The enforcement chain

Decision happens before the FHIR op runs

4 **Access decision at the seam**
Allow or deny - before the op

— CONCEPT · ZERO TRUST - NO SESSION YOU CAN RIDE

Every request re-validates the token. A valid signature is not permission.

Authorization lands on every single request - that is zero trust. No implicit trust, no session to coast on. Each call re-validates the bearer JWT through four checks, and even a flawless token can still earn a 403 if its scope does not cover the resource.

CHECK 1

Signature

Verified against the authorization server's JWKS public keys.

CHECK 2

exp

The token has not expired.

CHECK 3

aud

Minted for this resource server.

CHECK 4

iss

From the authorization server we trust.

— THE PART PEOPLE MISS

A valid signature proves who, not what. Scopes still gate the operation - a perfectly signed token still gets a `403` when its scope does not cover that resource. An expired or replayed token dies right here, at the seam.

— DIAGRAM · SYNTHESIS - ONE APP SESSION, EVERY GATE LABELLED

The whole chapter on one canvas.

Get the token

Launch through token exchange

4 **Access token**
+ scopes + patient context

Use the token

Validate, then enforce the scope

8 **Match → data · Miss → 403**
Returned or refused

launch · PKCE · scopes · context · validate · enforce - one picture

— RECAP · TRY IT YOURSELF - THEN FORWARD TO CHAPTER 18

Decode a token, write three scopes, map each to one check.

— TRY IT · DECODE A TOKEN

Grab a sample SMART access token and decode it at `jwt.io`. Read the claims - `iss`, `aud`, `exp`, the `scope` list, and `patient`.

— TRY IT · WRITE THREE SCOPES

For a vitals-dashboard app: `patient/Observation.rs?category=vital-signs`, `launch/patient`, and `openid fhirUser`. Map each to the single interceptor check it becomes.

- App launch & OAuth
- PKCE
- v2 scope grammar
- Launch contexts
- Enforcement chain
- Zero-trust JWT

- Next: Chapter 18 - FHIR over MCP for LLM agents · same gate, system/ scopes, no human in the loop