**STRIDE Threat Model**

|  |  |  |  |
| --- | --- | --- | --- |
| **Threat** | **Property Violated** | **Example / Focus** | **Mitigation Principle** |
| **S – Spoofing** | Authentication | Fake identity, forged credentials | MFA, Authenticator Pattern |
| **T – Tampering** | Integrity | Modifying code/data | Hashing, Digital Signatures |
| **R – Repudiation** | Non-repudiation / Accountability | Delete or deny logs | Immutable Logs, Audit Trails |
| **I – Information Disclosure** | Confidentiality | Unauthorized read | Encryption, Access Control |
| **D – Denial of Service** | Availability | Overload / Disable | Rate Limiting, Redundancy |
| **E – Elevation of Privilege** | Authorization | Bypass roles | Least Privilege, RBAC/ABAC |

**LINDDUN Privacy Threat Categories**

|  |  |  |
| --- | --- | --- |
| **Letter** | **Meaning** | **Typical Violation** |
| **L** – Linkability | Linking records across contexts | Pseudonymization |
| **I** – Identifiability | Re-identifying subjects | Anonymization |
| **N** – Non-repudiation | Can’t deny actions → privacy breach | Policy separation |
| **D** – Detectability | Existence of data visible | Obfuscation, padding |
| **D** – Disclosure of Information | Unauthorized exposure | Encryption |
| **U** – Unawareness | No user knowledge/consent | Notice, Transparency |
| **N** – Non-compliance | Breach of law/policy (e.g., GDPR) | Data Minimization, Purpose Limitation |

**Security Patterns**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pattern** | **Type / Scope** | **Problem Solved** | **Key Design Concept** |
| **Authenticator** | *Design* | Verify identity centrally | Central Auth Service validates credentials → issues token/assertion |
| **Multi-Factor Auth (MFA)** | *Mechanism* | Strengthen user verification | Combine ≥ 2 factors: Knowledge (pwd), Possession (token/HOTP), Inherence (biometric) |
| **Gatekeeper / Security Proxy** | *Structural / Implementation* | Central enforcement before business logic | Intercepts requests → validates token → applies policy → logs access |
| **RBAC (Role-Based Access Control)** | *Architectural / Design* | Manage authz by roles | Map user → role → permission matrix |
| **ABAC (Attribute-Based Access Control)** | *Architectural / Design* | Contextual decisions | Evaluate subject/object/environment attributes |
| **Secure Channel (mTLS)** | *Implementation* | Confidentiality + authenticity in transport | Mutual certificate verification; encrypt data |
| **Input Validation** | *Code-level* | Prevent injection/XSS | Sanitize inputs before processing |

**Access & Authentication Concepts**

* **Authentication vs Authorization**  
  AuthN = who you are | AuthZ = what you can do
* **Tokens / Assertions**
  + JWT: iss, sub, aud, exp, scope, role
  + SAML / OIDC: signed XML or JWT assertions
* **Lifetimes & Scopes**
  + Short-lived tokens (≤ 15 min) → refresh via Authenticator
  + Scoped permissions (e.g., billing.read, mission.write)
* **Service-to-Service Auth**
  + Use client certs (mTLS) + service JWTs (aud = target API)
  + No password sharing between microservices
* **Auditability**
  + Immutable logs + time-stamped token IDs + user context

**Principles for Secure Architecture**

* **Least Privilege:** Grant minimal required access.
* **Defense in Depth:** Layer controls (Gatekeeper + RBAC + mTLS).
* **Complete Mediation:** Check every access, every time.
* **Fail Secure:** If auth fails, deny access safely.
* **Separation of Duties:** No single entity can perform destructive actions alone.
* **Accountability:** Trace every action to an identity.
* **Secure by Default:** Opt-in for risk, not for protection.