FPT Education
FPT UNIVERSITY

# *Session 3*
# Mitigations & Security Design Patterns


Presenter: **Mr. Ngo Tung Son (Ph.D.)**

- Head of Information Assurance Department, FPT University, Hanoi, Vietnam
- Director of RISCS Laboratory, FPT University, Hanoi, Vietnam

FPT Education
FPT UNIVERSITY

# Recap: Threat Modeling

CONCEPT & MEANING

- **Adversarial perspective**: stop thinking like builders; see **code + components + data flows** across **trust boundaries**.
  Góc nhìn đối kháng: nhìn hệ thống như mã + thành phần + luồng dữ liệu qua ranh giới tin cậy.
- **Focus on assets & attack surface**: prioritize high-value data/services/privileges; list entry points & interfaces.

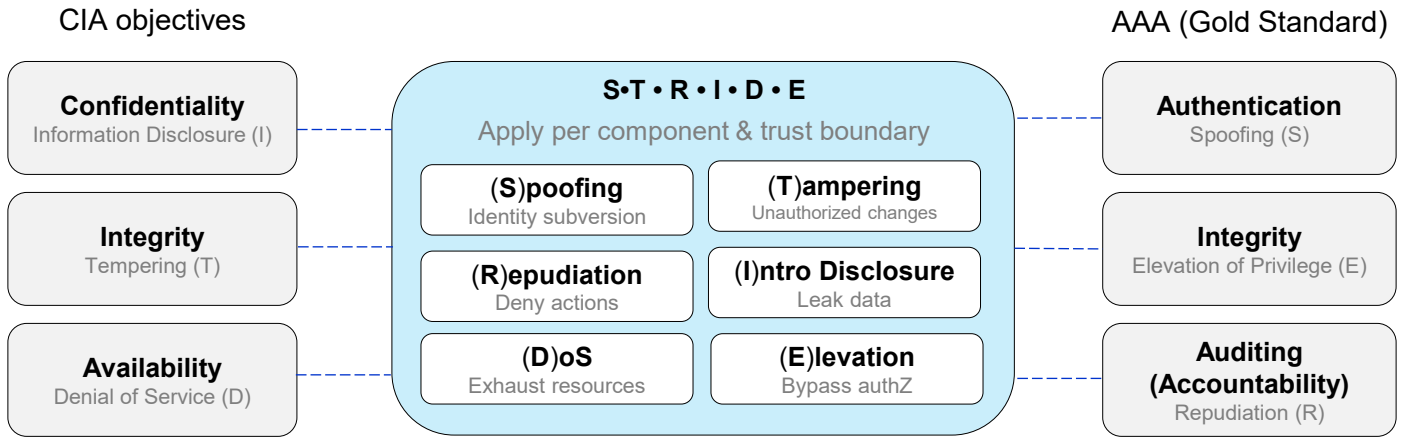
Tập trung tài sản & bề mặt tấn công: ưu tiên dữ liệu/dịch vụ/quyền giá trị cao; liệt kê điểm vào & giao diện.

- **Four Questions (Q1→Q4)**
- **STRIDE for Threat Classification**
  - ✓ **Per entry point**: ask 6 questions (S/T/R/I/D/E).
    Mỗi điểm vào: hỏi đủ 6 câu STRIDE.
- **Misuse/Abuse cases:** write reverse stories + detection criteria; map to STRIDE & owners.
  Tình huống lạm dụng: story đảo + tiêu chí phát hiện; ánh xạ STRIDE & gán phụ trách.

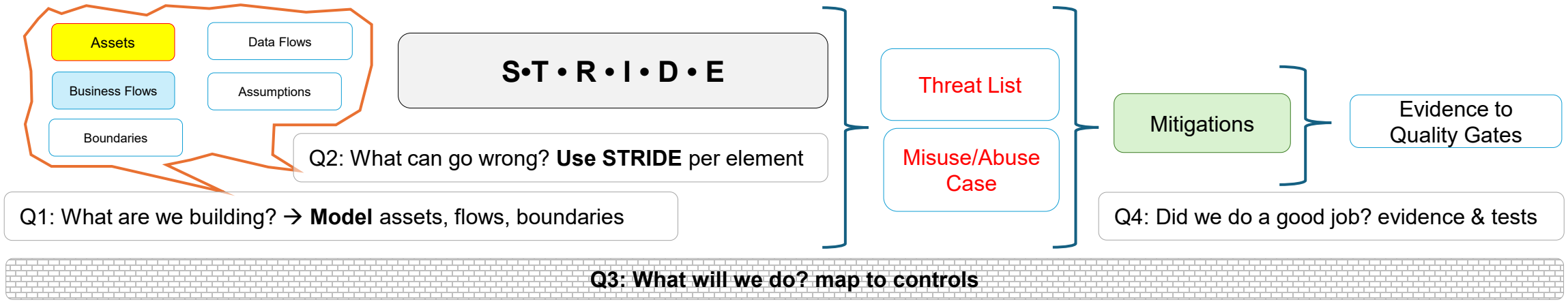
ILLUSTRATION

## STRIDE ↔ CIA & AAA

Three map to CIA; three to AAA (Gold Standard).
Ba hạng mục ánh xạ CIA; ba hạng mục ánh xạ AAA.




## Where STRIDE fits (Matching with 4 Questions)



FPT Education
FPT UNIVERSITY

# Objectives

- State how mitigations make attacks less likely, harder, less harmful, or more detectable, and commit to carrying them through SDLC quality gates with evidence
  Xác định mitigation làm tấn công ít xảy ra, khó hơn, ít gây hại, hoặc dễ phát hiện, và gắn chúng vào các cổng chất lượng SDLC kèm bằng chứng.
- Prioritize structural mitigations: minimize attack surface and data exposure, narrow windows of vulnerability, and protect interfaces/communications/storage.
  Ưu tiên biện pháp mang tính kiến trúc: giảm bề mặt tấn công & phơi bày dữ liệu, thu hẹp "cửa sổ" dễ tổn thương, và bảo vệ giao diện/kết nối/lưu trữ.
- Embed patterns like Least Privilege, Defense in Depth, Least Information, Complete Mediation, Fail Securely, and Allowlists-over-Blocklists in everyday decisions.
  Áp dụng các mẫu như Đặc quyền tối thiểu, Phòng thủ nhiều lớp, Thông tin tối thiểu, Trung gian hoàn chỉnh, Fail-secure và Danh sách cho phép trong quyết định thiết kế & triển khai.
- Recognize recurring risky choices—Confused Deputy, Backflow of Trust, Third-Party Hooks, Unpatchable Components—and replace them before they harden into debt.
  Nhận diện và thay thế sớm các phản mẫu—Confused Deputy, Dòng tin cậy chảy ngược, Hook bên thứ ba, Thành phần không thể vá—trước khi thành nợ kỹ thuật.
- For every selected pattern/anti-pattern decision, produce test results, logs, and configuration proofs to pass SDLC gates.
  Với mỗi quyết định về mẫu/phản mẫu, xuất trình kết quả kiểm thử, log và cấu hình để vượt qua các

# Part 1
# Mitigation Framework & SDLC Tracks

# Mitigation — Definition & Design Maxims

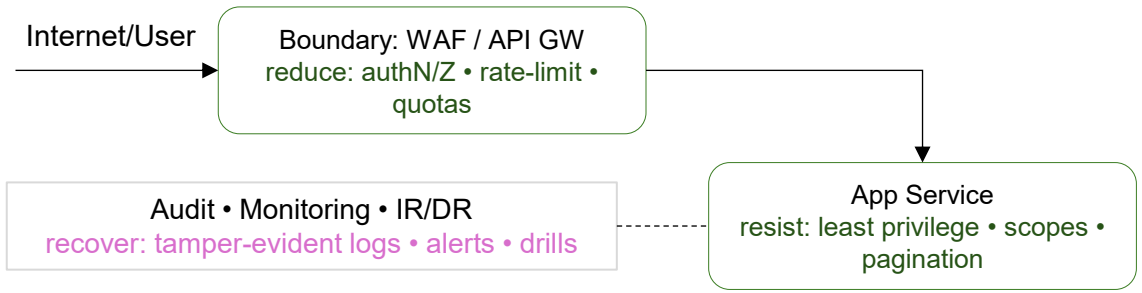
DEFINITION

## Proactive response to risk

- Make attacks **less likely**, **harder**, **less harmful**, or **more detectable** (faster response).
  Làm tấn công ít xảy ra, khó hơn, ít gây hại, hoặc dễ phát hiện (phản ứng nhanh).
- Not elimination: removing a risk is *removal*, not mitigation.
  Không phải loại bỏ: xoá hẳn rủi ro là *removal*, không phải mitigation.

**Design maxims:** *reduce* • *resist* • *recover* — shrink surfaces, add independent layers, prepare for rapid recovery.
thu nhỏ bề mặt, phòng thủ nhiều lớp, chuẩn bị phục hồi nhanh.

ILLUSTRATION – REDUCE • RESIST • RECOVER




Three complementary levers across the boundary and services to evidence & response.
Ba đòn bẩy bổ trợ qua ranh giới và dịch vụ, gắn với bằng chứng & phản ứng.

TEMPLATE

## Describe the mitigation thoroughly

- **Target threat** & **risk** (Likelihood × Impact): name the STRIDE type + concrete scenario; state *L*×*I* (e.g., High×High) and short rationale.
  Đe doạ mục tiêu & rủi ro (Khả năng × Tác động): Đe doạ & rủi ro: nêu kiểu STRIDE + tình huống cụ thể; ghi *L*×*I* và lý do ngắn gọn.
- **Asset** / **boundary** / **flow** where it applies: reference DFD nodes/edges and trust boundary IDs (e.g., D1→P2 via TB-API).
  Tài sản / ranh giới / luồng áp dụng: tham chiếu nút/cạnh DFD và mã ranh giới tin cậy.
- **Mechanism**: named pattern/control + key configuration: use a named pattern/control with key params (e.g., *API GW rate-limit 60 req/min/token*).
  Cơ chế: nêu pattern/kiểm soát có tên và tham số chính
- **Injection point** (code/ infra/ pipeline): where it lives (service/repo/path, Terraform/WAF policy, pipeline step).
  Điểm chèn (mã/hạ tầng/pipeline): vị trí triển khai (service/repo/đường dẫn, Terraform/WAF, bước pipeline).
- **Evidence:** tests, logs, reviews, signatures: list tests (unit/integration), policy-as-code, log schema + sample, alert rule.
  Bằng chứng: liệt kê test, policy-as-code, schema log + ví dụ, rule cảnh báo
- **Residual risk & assumptions;** human factors & usability: what remains out of scope; dependencies; human/usability trade-offs.
  Rủi ro còn lại & giả định: phần chưa bao phủ; phụ thuộc; đánh đổi về khả dụng.

# Mitigation to SDLC & Gates (Spec/Architecture → SDR/FSR)

SDLC MAPPING

## From candidate mitigations to implemented controls

- **Spec/Architecture**: create a catalog of *candidate mitigations*; attach to key *flows* & *entry-points* by boundary/asset.
  Đặc tả/kiến trúc: lập danh mục *biện pháp ứng viên*; gắn vào *luồng & điểm vào* then chốt theo ranh giới/tài sản.
- **SDR (Security Design Review)**: review TM vN and architectural placements; confirm owners/dates; approve to build.
  SDR: rà Threat Model vN và điểm đặt biện pháp trong kiến trúc; chốt chủ sở hữu/ngày; phê duyệt để chuyển sang xây dựng.
- **Construction & Testing**: move from description to *code/config/infra*; accumulate verifiable *evidence* via tests/logs/reviews.
  Xây dựng & Kiểm thử: chuyển từ mô tả sang *mã/cấu hình/hạ tầng*; gom *bằng chứng* có thể kiểm chứng (test/log/review).
- **Pre-FSR**: show *updated Threat Model + test/config evidence + SBOM/CVE posture + Bug Bar* met.
  Trước FSR: chứng minh *Threat Model cập nhật + bằng chứng test/cấu hình + SBOM/CVE + Bug Bar* đạt.

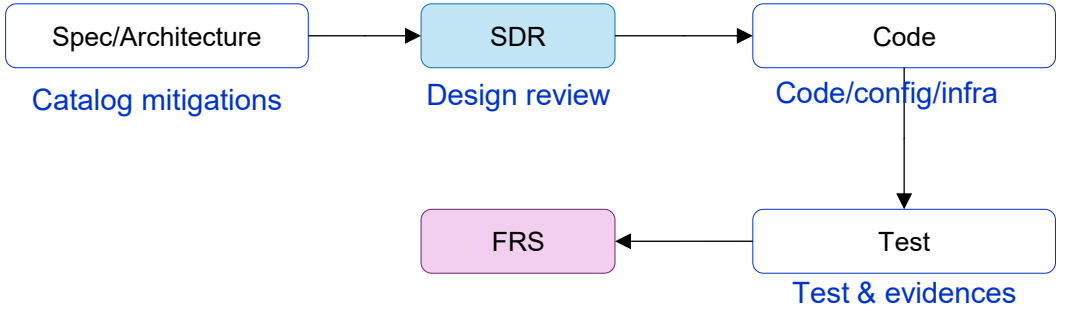
**Claim → Argument → Evidence**: each control states a claim, backs it with a clear argument (threat/placement/mechanism), and provides verifiable evidence (tests, logs, reviews).

Claim → Argument → Evidence: mỗi kiểm soát nêu tuyên bố, lập luận (đe doạ/vị trí/cơ chế), và bằng chứng kiểm chứng (test, log, review).

SDLC MAPPING

## From candidate mitigations to implemented controls

- **SDR (Security Design Review)**: completeness of design docs & DFD; top risks mitigated at the right architectural points; owners/dates set.
  SDR: đầy đủ thiết kế & DFD; rủi ro lớn được xử lý đúng điểm kiến trúc; có chủ sở hữu/ngày hoàn thành.
- **Bug Bar**: Critical/High = 0 before ship; Medium has mitigation & scheduled fix; Low has an owner.
  Bug Bar: Critical/High = 0 trước khi phát hành; Medium có biện pháp & lịch sửa; Low có chủ sở hữu.
- **FSR (Final Security Review)**: decision packet shows TM-vN, test results, configs, SBOM/CVE status; Go/No-Go/Risk-Accepted recorded.
  FSR: hồ sơ quyết định gồm TM-vN, kết quả kiểm thử, cấu hình, SBOM/CVE; ghi nhận Go/No-Go/Risk-Accepted.




Controls progress left→right; gates require evidence packets (TM, tests, configs, SBOM/CVE, Bug Bar).
Kiểm soát tiến hoá trái→phải; cổng yêu cầu hồ sơ bằng chứng (TM, test, cấu hình, SBOM/CVE, Bug Bar).

# Structural Mitigation Strategy — Reduce Attack Surface

DEFINITION & WHY IT MATTERS

**Attack surface = all entry points an attacker can interact with**

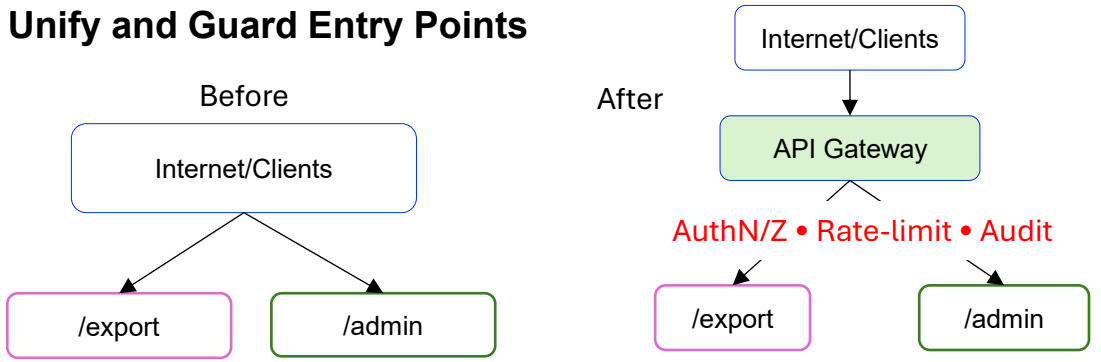
- After identifying surfaces, harden the "outer shell" and reduce how much code/data each entry point touches.
  Khi đã nhận diện bề mặt, gia cố "vỏ ngoài" và giảm lượng code/dữ liệu bị chạm tới từ mỗi điểm vào.
- Unify duplicate interfaces so there's less code that can harbor bugs.
  Hợp nhất các giao diện trùng lặp để giảm mã có thể chứa lỗi.

**CONCRETE MOVES**

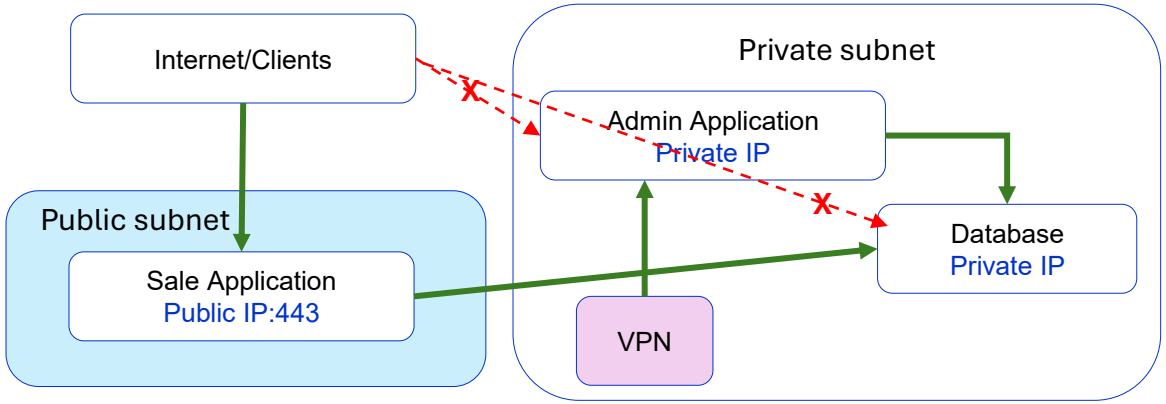
- Move from *anonymous/public* API to **authenticated** API; enable traceability & rate-limits.
  Chuyển từ API *mặc định mở* sang API yêu cầu nhận dạng/xác thực; hỗ trợ truy vết & giới hạn tần suất.
- Disable rarely used *remote admin*; prefer on-prem/wired access when needed.
  Vô hiệu hoá cấu hình *quản trị từ xa* ít dùng; ưu tiên truy cập có dây khi cần.
- Consolidate enforcement at a **gateway/guard** (PEP) — authN/Z, input contracts, quotas, audit.
  Tập trung cưỡng chế tại gateway/guard — xác thực/uỷ quyền, hợp đồng đầu vào, hạn mức, ghi vết.
- Reduce exposure in deployment: put what you can *behind firewalls*; minimize open ports/services.
  Giảm phơi bày khi triển khai: đưa tối đa dịch vụ *vào sau tường lửa*; tối thiểu cổng/dịch vụ mở.

EXAMPLES

**Unify and Guard Entry Points**




**Network Segmentation**




**Don't over-trust intranets**: treat them as *lower-risk surfaces*, not fully trusted. For kiosks/devices, the exterior (screen/buttons) is a *physical attack surface* — define safe behavior in untrusted environments.

Đừng tin tuyệt đối intranet: coi như *bề mặt rủi ro thấp*, không phải "an toàn tuyệt đối". Với thiết bị/kiosk, phần bên ngoài (màn hình/nút bấm) là *mặt phẳng tấn công vật lý* — cần định nghĩa rõ hành vi an toàn trong môi trường không tin cậy.

# Structural Mitigation Strategy — Narrow Windows of Vulnerability

PRINCIPLE

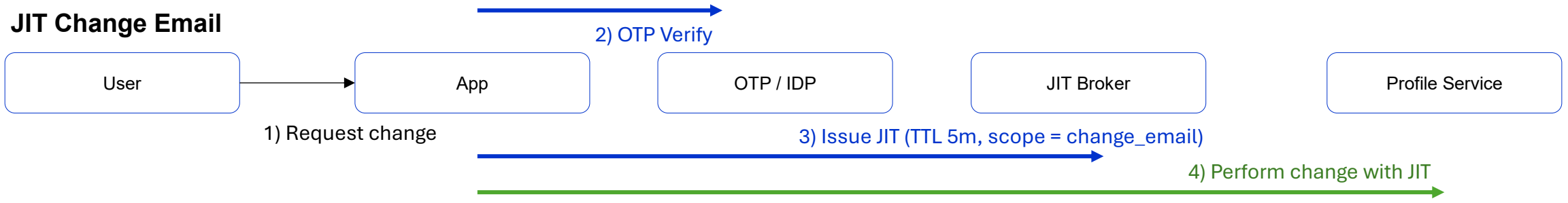
**Minimize the *time* and *scope* during which high privilege or risky exposure exists**

- **Just-in-time (JIT) elevation**: assert privilege at the last responsible moment; *revert* immediately after use.
  Nâng quyền đúng lúc (JIT): chỉ nâng đặc quyền ngay trước khi cần; *hạ* ngay khi xong.
- **Short-lived credentials & sessions**: tokens/leases with tight TTL; frequent rotation; avoid long-lived admin sessions.
  Thông tin xác thực ngắn hạn: TTL chặt; xoay khoá thường xuyên; tránh phiên quản trị kéo dài.
- **Small critical sections**: split sensitive flows; check *pre-conditions* (MFA, risk/context) before entering; release resources on exit.
  Đoạn quan trọng ngắn: tách phần nhạy cảm; kiểm tra *tiền điều kiện* (MFA, rủi ro/ngữ cảnh); giải phóng tài nguyên khi thoát.
- **Context limits**: daily caps, business hours, geo/tenant/currency constraints to cap worst-case impact.
  Giới hạn ngữ cảnh: hạn mức/ngày, khung giờ làm việc, khu vực/tenant/loại tiền… để giới hạn thiệt hại tối đa.

EXAMPLES

**Pre-assigned URL**


**JIT Change Email**



# Structural Mitigation Strategy — Minimize Data Exposure

PRINCIPLE

## Collect, process, and reveal the *minimum necessary* data

- **Least Information**: request/access only what the task needs; do not pass extra fields around by habit.
  Ít thông tin nhất: chỉ yêu cầu/truy cập đúng phần dữ liệu cần; tránh truyền thêm trường thừa theo thói quen.
- **Short residency**: avoid keeping sensitive values (passwords, tokens, PII) in memory longer than needed; wipe local caches after use.
  Thời gian tồn tại ngắn: tránh giữ lâu thông tin nhạy cảm trong bộ nhớ; xoá cache cục bộ sau khi dùng.
- **Storage lifecycle**: explicit retention + secure deletion (shred) upon expiry; encrypt raw logs/objects; rotate keys regularly.
  Vòng đời lưu trữ: đặt thời hạn lưu rõ ràng + xoá an toàn khi hết hạn; mã hoá dữ liệu gốc; xoay khoá định kỳ.
- **Observability with privacy**: default to *filtered* logs; unfiltered access only with approval; logs are append-only and auditable.
  Quan sát tôn trọng riêng tư: mặc định log *đã lọc*; chỉ mở log *đầy đủ* khi có phê duyệt; log chỉ-bổ-sung và có thể audit.

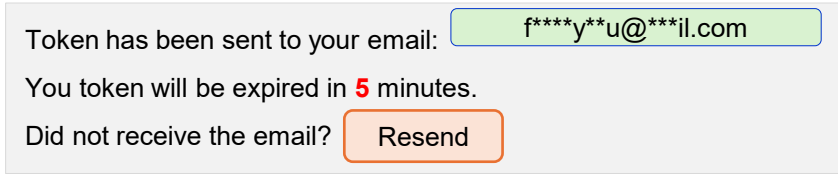
EXAMPLES

- **API design**: RequestCustomerData(id, items=["name","zip"]) → {name, zip} (not the full record).
  Thiết kế API: chỉ trả về các trường yêu cầu thay vì bản ghi đầy đủ.
- **Logs**: store filtered views (e.g., country instead of IP); keep raw encrypted; grant temporary unfiltered view by tokenized approval.
  Log: lưu bản đã lọc (vd: quốc gia thay vì IP); bản gốc mã hoá; mở tạm bản đầy đủ qua phê duyệt có token.

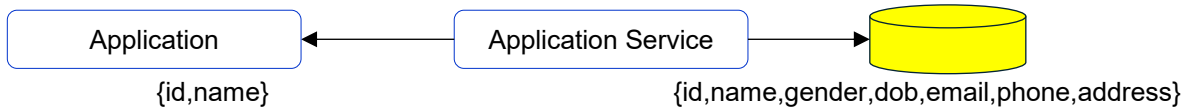
EXAMPLES

## Least Information

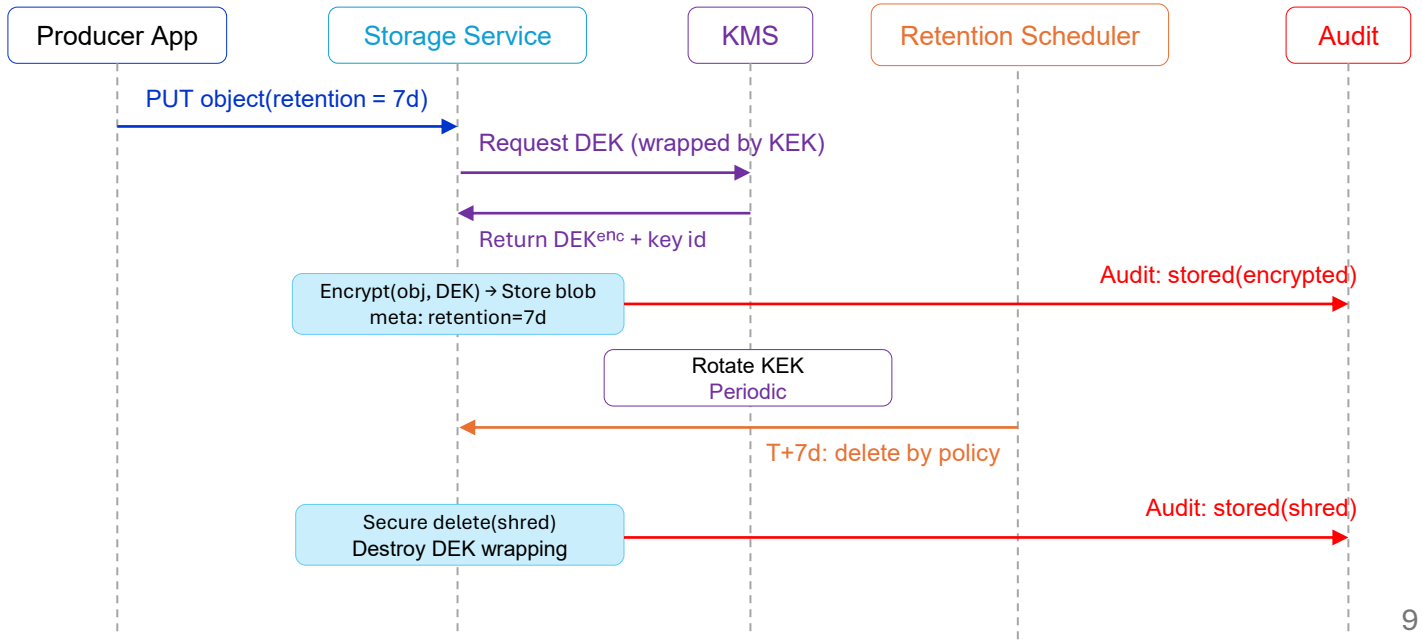
- Hide PII




- API Design for query data




## Storage Lifecycle



# Access Policy & Controls

**Why policy matters.** OS file permissions are all-or-nothing and tied to process owners, not modern web/microservice principals. Define an *ideal access policy* (who/ what /when/where/how much) independent of platform quirks; then enforce via guards.
Quyền hệ điều hành kiểu tất-cả-hoặc-không gắn với tiến trình, không phù hợp web/microservice hiện đại. Hãy định nghĩa *chính sách truy cập lý tưởng* (ai/cái gì/khi nào/ở đâu/bao nhiêu) độc lập nền tảng; rồi cưỡng chế qua điểm chặn.

CONCEPTS

**From OS ACLs → Fine-Grained Access Policy**

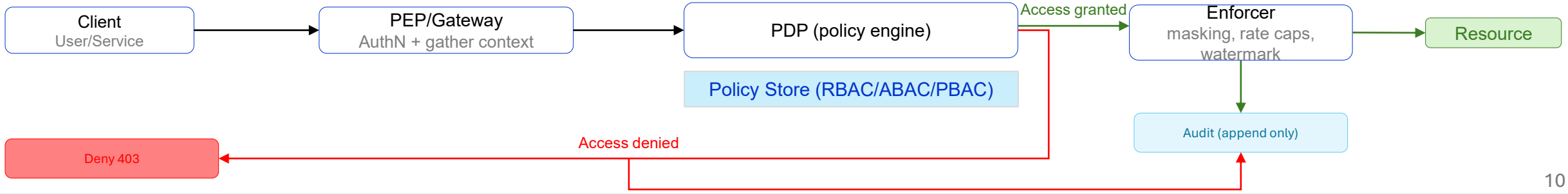
- **Problem.** One process often serves all users → needs access to *all data* all the time; any vuln risks everything.
  Vấn đề. Một tiến trình phục vụ mọi người dùng → cần quyền tới *mọi dữ liệu* mọi lúc; lỗ hổng đe doạ toàn bộ.
- **Policy before controls.** Start from principals/needs; express proper access precisely (resource, action, conditions, *limits*).
  Đặt chính sách trước cơ chế. Xuất phát từ chủ thể/nhu cầu; mô tả đúng quyền truy cập (tài nguyên, hành động, điều kiện, *giới hạn*).
- **Dimensions to use.** Per-minute/hour/day caps; data volume limits; business hours; peer/historical anomaly thresholds.
  Chiều đo hạn chế. Trần theo phút/giờ/ngày; giới hạn dung lượng; giờ làm việc; ngưỡng bất thường so với đồng nghiệp/lịch sử.
- **Iterate.** Start lenient for observability → tighten as you learn.
  Lặp cải tiến. Bắt đầu nới để quan sát → siết dần theo thực tế sử dụng.

MODELS *BASED ACCESS CONTROL

**RBAC vs ABAC vs PBAC**

| Model | Idea | Pros | Cons | Example |
|---|---|---|---|---|
| **R(Role)BAC** | Grant by *role* | Simple; auditable | Coarse; poor context. | role == 'Agent' → read:Customer |
| **A(Attribute)BAC** | Decide by *attributes* of subject/object/env. | Flexible; context-aware. | Policy complexity. | dept == 'Care' & time in business_hours |
| **P(Policy)BAC** | Central policy (XACML/OPA); obligations & limits. | Uniform; testable; decoupled. | Infra overhead | allow if risk<=low; mask PII; cap 100/day |

DECOUPLING POLICY FROM ENFORCEMENT



# Interfaces - Trust Boundaries & Security Focus

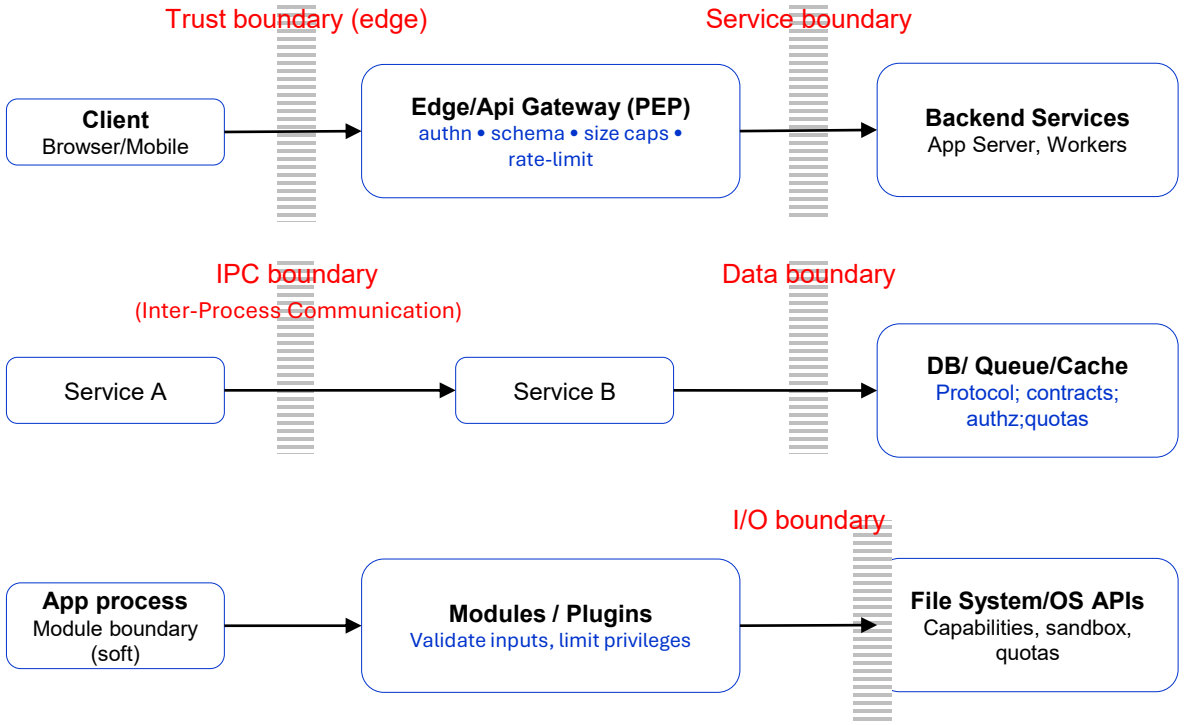
**Why boundaries matter.** Components connect via interfaces (network, process, in-process). Interfaces expose flows and create clear places to add mitigations—especially where lower-trust calls into higher-trust.
Vì sao ranh giới quan trọng. Thành phần nối nhau qua giao diện (mạng, tiến trình, trong tiến trình). Giao diện lộ luồng và tạo điểm để chèn kiểm soát—đặc biệt khi miền tin cậy thấp gọi vào miền cao.

CONCEPTS

**Interfaces (Application Programming Interface/ User Interface)**

- **Normalize early.** Parse & canonicalize inputs; enforce *strict schemas*; reject unknowns.
  Chuẩn hoá sớm. Phân tích & chuẩn hoá; áp schema nghiêm; từ chối trường lạ.
- **Constrain format & size.** Allowlists for types, MIME, file size; back-pressure with 413.
  Giới hạn định dạng & kích thước. Danh sách trắng kiểu, MIME, cỡ tệp; phản hồi 413 khi quá tải.
- **Rate-limit at the edge.** Per IP/app/user; token bucket; idempotency keys for retries.
  Giới hạn tốc độ ở edge. Theo IP/ứng dụng/người dùng; bucket; khoá bất biến cho retry.
- **Embed control.** CSP/allowlist for images/scripts/fonts; block mixed content.
  Kiểm soát nhúng. CSP/danh sách trắng cho ảnh/script/font; chặn nội dung pha trộn.

INTERFACES LANDSCAPE- WHERE TRUST BOUNDARIES USUALLY ARE

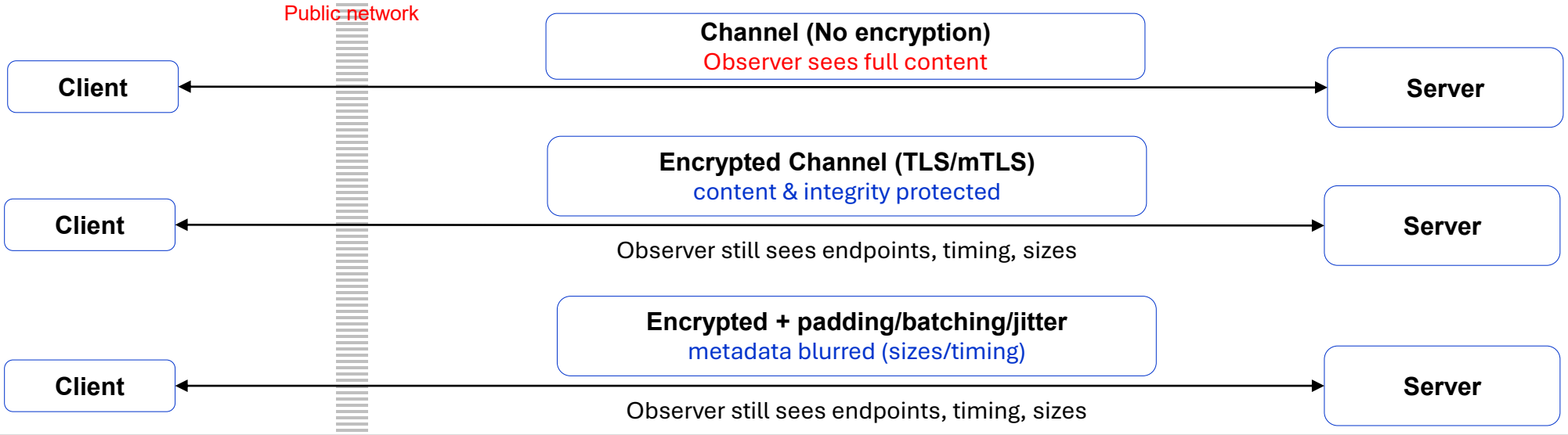


# Communication — Channels • Encryptions

Modern systems are almost always networked (internet, private nets, Bluetooth/USB, etc.). To protect communications, either **physically secure the channel** or **encrypt to ensure confidentiality & integrity**. Physical security alone is fragile; encryption overhead is usually acceptable.

Hệ thống hiện đại hầu như luôn có mạng (internet, mạng nội bộ, Bluetooth/USB, v.v.). Để bảo vệ truyền thông, hoặc bảo vệ kênh vật lý hoặc mã hoá để đảm bảo bí mật & toàn vẹn. Chỉ dựa vào vật lý thì mong manh; chi phí mã hoá thường chấp nhận được.

INTERFACES LANDSCAPE- WHERE TRUST BOUNDARIES USUALLY ARE




## What encryption gives you

- **Confidentiality** of payloads over untrusted links.
  Bí mật của dữ liệu qua kênh không tin cậy.
- **Integrity & authenticity** (TLS MAC, certificates/mTLS).
  Toàn vẹn & xác thực (MAC của TLS, chứng thư/mTLS).
- **Perfect Forward Secrecy** with ephemeral keys.
  Bí mật chuyển tiếp hoàn hảo với khoá tạm thời.

## What encryption does not hide

- **Fact of communication**: who talks to whom, when.
  Việc giao tiếp diễn ra: ai nói với ai, khi nào.
- **Traffic metadata**: byte counts, direction, timing side-channels.
  Số liệu siêu dữ liệu: số byte, hướng, kênh phụ thời gian.
- **Availability**: attackers can delay/drop packets (DoS).
  Tính sẵn sàng: kẻ tấn công có thể trì hoãn/chặn (DoS).

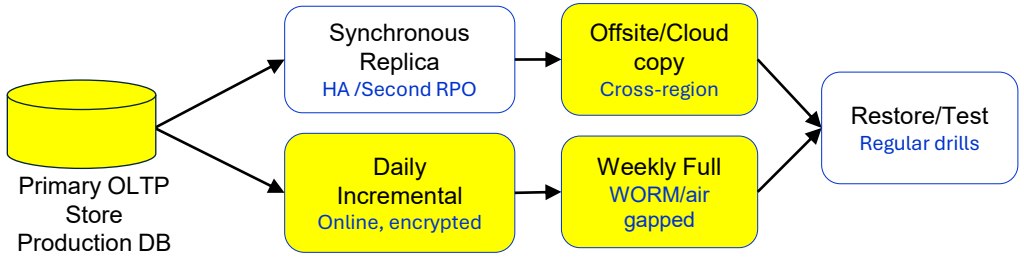
# Storage – Data at Rest

## CONCEPTS

- Data at rest faces threats of **disclosure**, **tampering**, and **loss**. Mitigate with **physical security** or **encryption**, and ensure **availability** via backups/replication tested for restore.
  Dữ liệu khi lưu đối mặt với rò rỉ, bị sửa đổi và mất mát. Giảm thiểu bằng bảo vệ vật lý hoặc mã hoá, và đảm bảo sẵn sàng qua sao lưu/sao chép có kiểm tra phục hồi.
- Communication protects bytes in transit; storage protects bytes at rest (to future self).
  Truyền thông bảo vệ dữ liệu khi đi qua mạng; lưu trữ bảo vệ dữ liệu khi nằm trên phương tiện (cho chính ta trong tương lai).

## BACKUPS & REPLICATION

- **Continuity vs Recovery:** Synchronous replica = continuity (near-zero **RPO**) but **not a backup**.
  Liên tục vs Khôi phục: Bản sao đồng bộ = liên tục (RPO gần 0) nhưng không phải backup..
- **Prove it:** Regular **restore/tests** to validate **RTO/RPO**, keys/escrow, and immutability.
  Chứng minh: Khôi phục/diễn tập định kỳ để kiểm chứng RTO/RPO, khoá/ký gửi và tính bất biến.

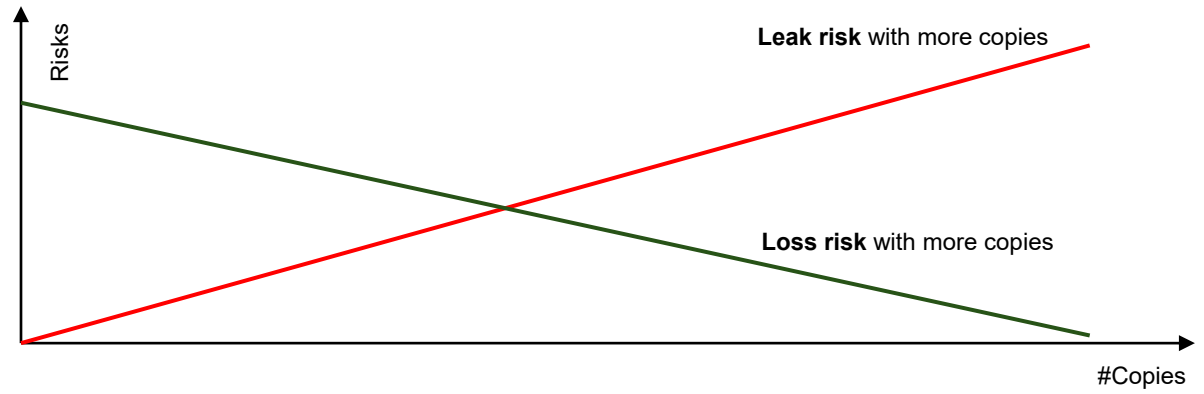



## BACKUP STRATEGIES

- Backup requirements must be designed from the ground up, balancing cos - performance when choosing full or incremental, defining frequency/latency, and storing independently; cloud can replicate near real-time but at a cost.
  Yêu cầu sao lưu phải thiết kế từ đầu, cân bằng chi phí–hiệu năng khi chọn full hay incremental, quy định tần suất/độ trễ và lưu trữ độc lập; cloud có thể nhân bản gần thời gian thực nhưng tốn chi phí.

| Strategy | Pros | Cons | Use when |
|---|---|---|---|
| **Full** | Simple restores; single artifact | Large storage/time | Small datasets; weekly baselines |
| **Incremental** | Efficient daily copies | Restore chains; more steps | Frequent changes; narrow RPO |
| **Differential** | Restore faster than incremental | Grows over time until next full | Balanced restore complexity |

## COPY X RISKS (TRADE OFF)



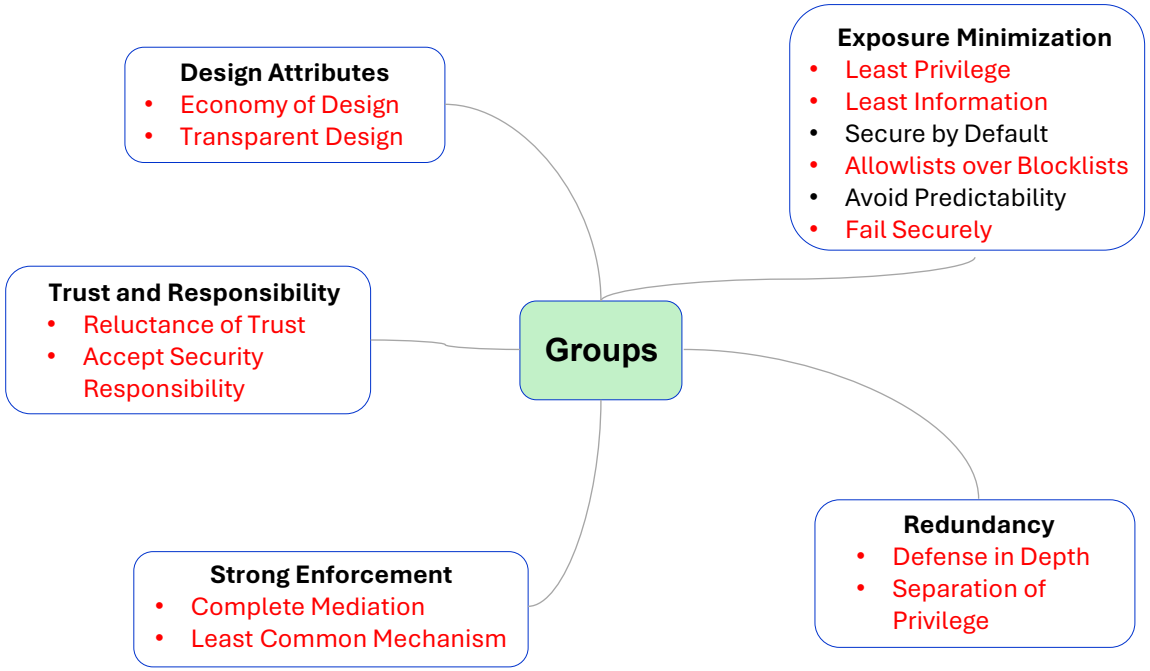
# Part 2
# Security Patterns: Overview & Usage

# Security Design Patterns

OVERVIEW

## Reusable, structural solutions to recurring security

- **What is a pattern?** A named, reusable *design solution* capturing trade-offs and evidence.
  Pattern là gì? Mẫu *thiết kế tái sử dụng* có tên, nêu rõ đánh đổi và bằng chứng hiệu lực.
- **How to use?** Tie pattern → *threat* → *control point* → *evidence* (tests/config/logs).
  Cách dùng? Gắn pattern với *mối đe doạ* → *điểm kiểm soát* → *bằng chứng* (kiểm thử/cấu hình/log).




- **There are five well-known design pattern groups:**
  - ✓ **Design Attributes**: patterns describe at a high level what secure design looks like: simple and transparent. These derive from the adages “keep it simple” and “you should have nothing to hide.”
    đơn giản và minh bạch. Những điều này bắt nguồn từ câu châm ngôn “hãy giữ mọi thứ đơn giản” và “bạn không nên che giấu điều gì”.
  - ✓ **Exposure Minimization:** set of design & operational measures to reduce the amount of sensitive data collected, processed, stored, and displayed; shorten the time it is visible; and by default, hide/limit visibility.
    tập hợp các biện pháp thiết kế & vận hành nhằm giảm lượng dữ liệu nhạy cảm được thu thập, xử lý, lưu trữ và hiển thị; rút ngắn thời gian hiện diện; và mặc định che giấu/giới hạn tầm nhìn.
  - ✓ **Strong Enforcement:** an architectural stance that forces every access through an authoritative decision point and removes shared mutable mechanisms that could bypass or dilute policy.
    tư thế thiết kế buộc mọi truy cập đi qua điểm phán quyết tin cậy và loại bỏ cơ chế dùng chung dễ làm loãng chính sách.
  - ✓ **Redundancy**: layering independent, differently-scoped safeguards so one failure does not cascade, favoring diversity and separation along critical paths.
    phòng thủ nhiều lớp độc lập để một lỗi không kéo sập toàn bộ; ưu tiên đa dạng hoá và tách biệt trên các luồng then chốt.
  - ✓ **Trust & Responsibility**: limiting implicit trust via explicit verification of identities /data and assigning clear ownership for security decisions and follow-through in operations.
    giới hạn tin cậy ngầm, xác minh rõ ràng, và phân định trách nhiệm ra quyết định & vận hành an ninh.

# Design Attributes — Economy & Transparent Design

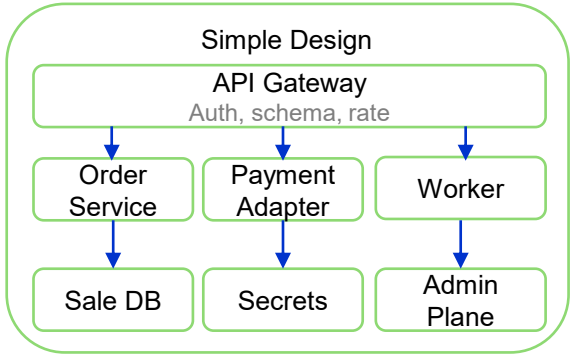
ECONOMY OF DESIGN

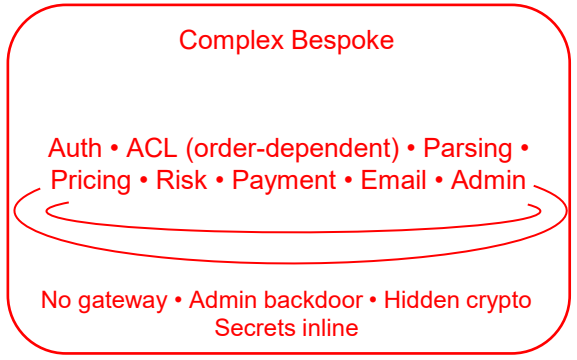
## Keep it simple

- Prefer the **simplest design** that meets requirements; fewer parts → fewer bugs.
  Ưu tiên thiết kế đơn giản nhất đáp ứng yêu cầu; ít thành phần → ít lỗi.
- Compose systems from **small, reusable blocks** instead of bespoke complex pieces.
  Ghép hệ thống từ khối nhỏ, tái sử dụng thay vì các phần phức tạp đặc thù.
- Only embrace complexity when it adds **significant value**.
  Chỉ chấp nhận phức tạp khi mang lại giá trị đáng kể.

EXAMPLES

- **Web services**: simple frontend + small subservices (parse, search, rank) vs one giant program.
  Dịch vụ web: frontend đơn giản + các dịch vụ nhỏ (parse, search, rank) thay vì 1 chương trình khổng lồ.



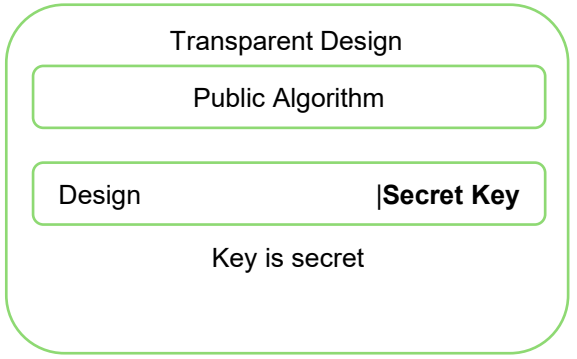

TRANSPARENT DESIGN

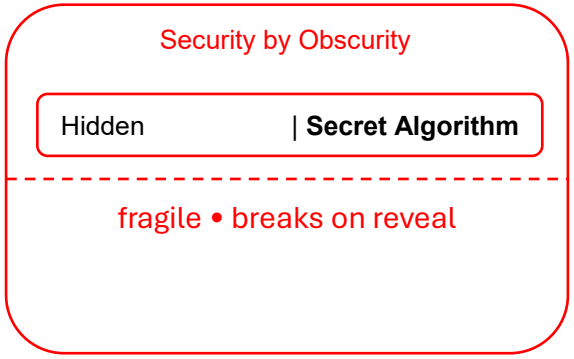
## No Security by Obscurity

- Strong protection should **not rely on secrecy of the design**.
  Bảo vệ mạnh không phụ thuộc vào việc giữ bí mật thiết kế.
- Publishable designs; keep **keys/identities** secret, not algorithms.
  Thiết kế có thể công khai; giữ bí mật là khóa/danh tính, không phải thuật toán.
- Fix weak designs (e.g., legacy DES), don't hide them.
  Sửa thiết kế yếu (vd DES cũ), đừng che giấu

EXAMPLES

- **Crypto**: public algorithm + secret key (Kerckhoffs' principle) vs home-grown hidden cipher.
  Mật mã: thuật toán công khai + khóa bí mật (nguyên lý Kerckhoffs) so với "mã hoá tự chế" bị giấu kín.



# Strong Enforcement — Complete Mediation

DEFINITION

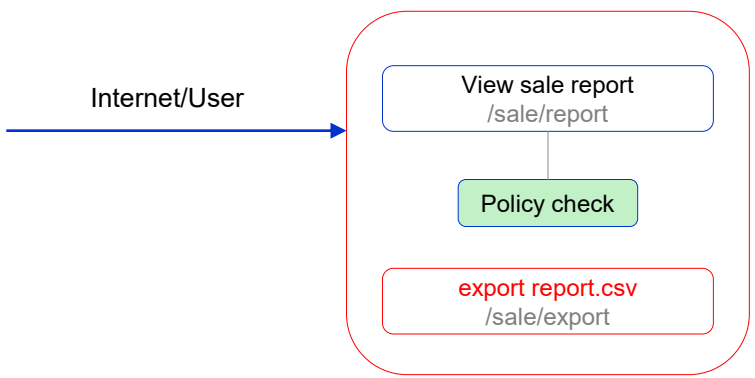
**Protect *all* access paths consistently — no side doors, no weaker policies.**

- Prefer a **single bottleneck/guard** where the authorization decision happens.
  Ưu tiên một điểm gác tập trung nơi ra quyết định uỷ quyền.
- If multiple paths are unavoidable, **the same check** (ideally **identical code/policy**) must run on all.
  Nếu buộc phải có nhiều đường, cùng một kiểm tra (tốt nhất chung mã/chính sách) phải áp dụng cho tất cả.

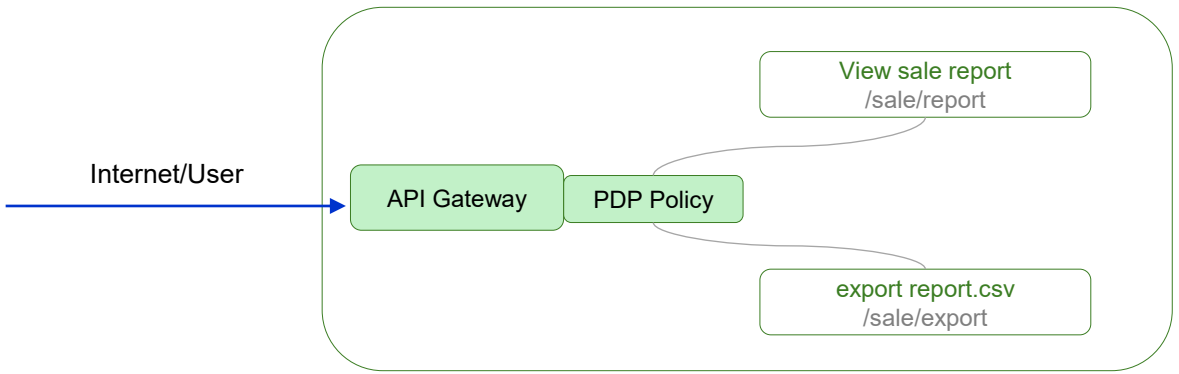
**Compliance Levels**

- **High** — one common routine/guard mediates every access.
  Cao — một hàm/điểm gác chung trung gian mọi truy cập.
- **Medium** — multiple guards, but functionally identical and reused.
  Trung bình — nhiều điểm gác nhưng *đồng nhất chức năng* và tái sử dụng.
- **Low** — multiple paths with inconsistent checks (incomplete mediation).
  Thấp — nhiều đường với kiểm tra *không nhất quán* (trung gian không đầy đủ).

EXAMPLE – WEB APPLICATION (INCOMPLETE MEDIATION VS COMPLETE MEDIATION)




Non-compliant (Incomplete Mediation)




Compliant (Single Guard / Identical Checks)

# Strong Enforcement — Least Common Mechanism

DEFINITION

## Maintain isolation by *minimizing shared mechanisms*

- Shared/internal mechanisms (caches, globals, pools, queues) can become **bridges** between otherwise isolated users/processes.
  Các cơ chế dùng chung (cache, biến toàn cục, pool, queue) có thể trở thành **cầu nối** giữa các người dùng/tiến trình vốn tách biệt.
- Prefer designs where functionality lives *inside the natural isolation boundary* (per-process/user/tenant) instead of in a common hub.
  Ưu tiên đặt chức năng *bên trong ranh giới cách ly tự nhiên* (mỗi tiến trình/người dùng/tenant) thay vì một trung tâm dùng chung.
- Classic intuition: the OS kernel isolates processes; avoid kernel-wide shared state that could leak/influence across processes.
  Trực giác kinh điển: kernel HĐH cách ly tiến trình; tránh trạng thái dùng chung toàn kernel có thể rò rỉ/ảnh hưởng chéo.

ANALOGY

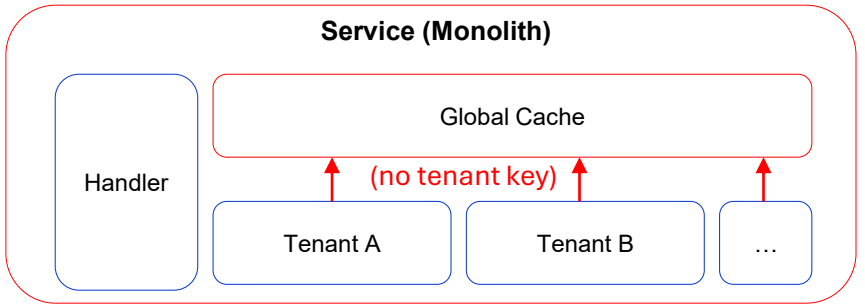
- Messy accountant desk = common workspace → mix-ups & privacy leaks; tidy per-client file = isolation.
  Bàn làm việc lộn xộn = không gian chung → lẫn lộn & lộ thông tin; hồ sơ riêng cho từng khách = cách ly.

**Maxim**: If a can run per-user/per-tenant/per-process, do it there — not in a featureshared singleton.

Châm ngôn: Nếu tính năng có thể chạy theo từng người dùng/tenant/tiến trình, hãy đặt ở đó — đừng gom vào singleton chung.

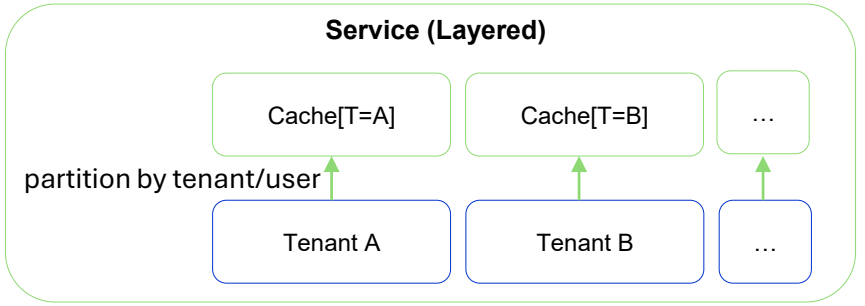
BRIDGE RISK FROM A SHARED MECHANISM

Before – Shared Cache




Bridge risk: leakage/influence across tenants

After – Isolated Mechanism




Move optional logic to userland (per-process)

# Exposure Minimization — Least Privilege

WHY THESE PATTERNS

**Grant only the access and data that are strictly necessary**

- **Least Privilege**: separate a *minimal privileged part* from the normal flow; elevate only when needed and *drop back* promptly.
  Đặc quyền tối thiểu: tách phần *đặc quyền tối thiểu* khỏi luồng bình thường; chỉ nâng khi cần và *hạ quyền* ngay.
- **Scoping Privilege**

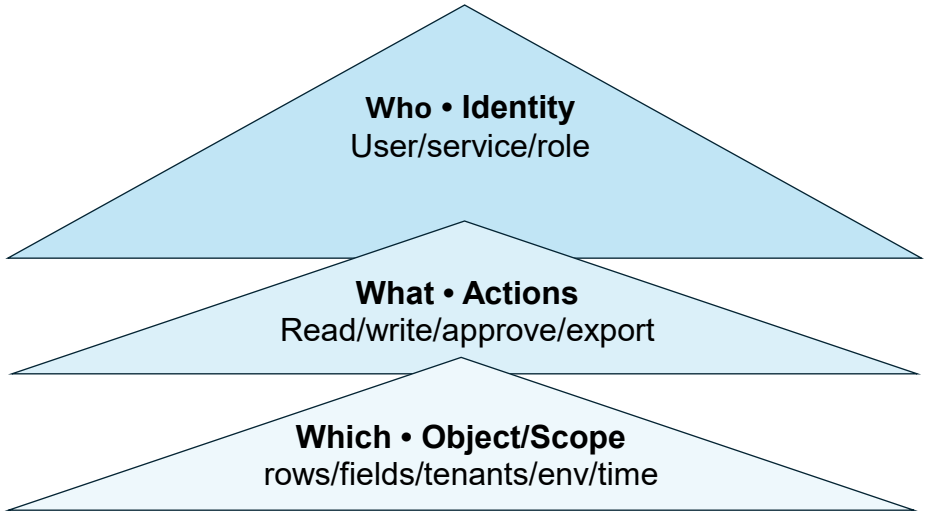
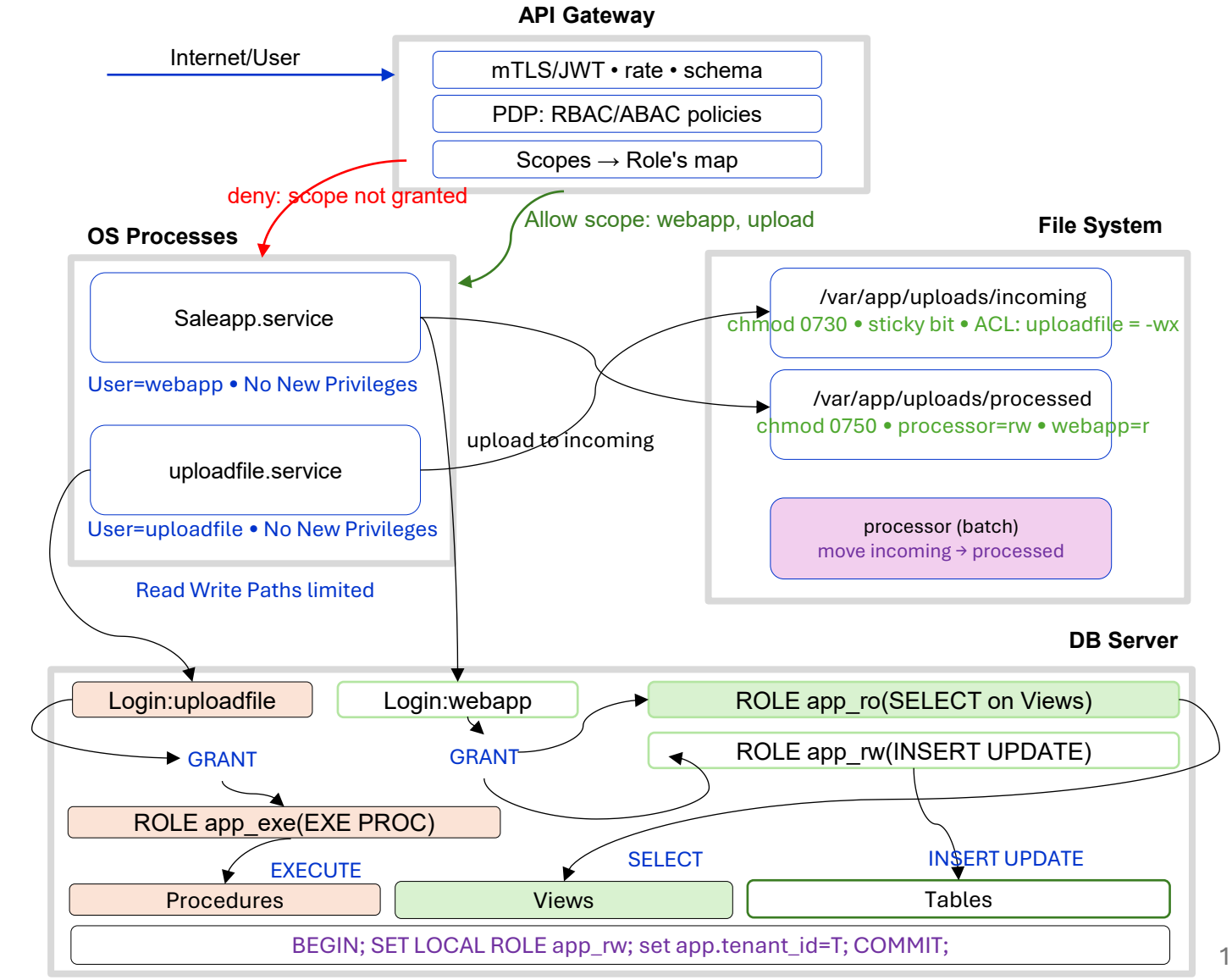



ILLUSTRATION OF LEAST PRIVILEGE



# Exposure Minimization: Least Information

**Definition.** *Least Information* means every interface returns the minimal fields, precision, and timing needed for its purpose. Limit at three stages: **Collect → Store → Disclose.**

Định nghĩa. *Ít thông tin nhất* là mọi giao diện chỉ trả về tối thiểu trường dữ liệu, độ chính xác và tín hiệu thời gian cần thiết. Giới hạn ở ba giai đoạn: Thu thập → Lưu trữ → Tiết lộ.

## Why exposure is dangerous

- **Enumeration & inference.** Extra fields/precision reveal total users, sales, or keys.**Liệt kê & suy diễn.**
  Trường/độ chính xác thừa làm lộ tổng số, doanh số, khoá.
- **Pivoting.** Errors/metadata provide internal IDs, table names, stack traces.**Điểm tựa tấn công.**
  Lỗi/metadata làm lộ ID nội bộ, tên bảng, stack trace.
- **Privacy leaks.** PII in responses, logs, URLs, or search suggests identity/behavior.
  Rò rỉ riêng tư. PII trong phản hồi, log, URL, hay tìm kiếm gợi ý danh tính/hành vi.

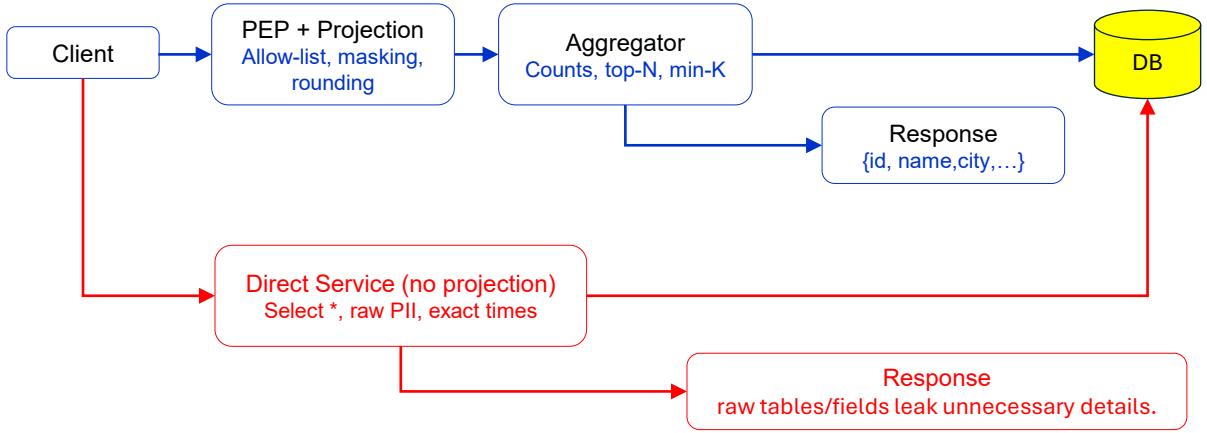
## Design Maxims

- **Purpose-binding.** Tie fields to a stated purpose; disallow out-of-purpose use.
  Ràng buộc mục đích. Gắn trường dữ liệu với mục đích; cấm dùng sai mục đích.
- **Projection layer.** Whitelist explicit fields; deny.
  Lớp chiếu dữ liệu. Cho phép danh sách trắng trường cụ thể; cấm.
- **Reduce precision. Bucketize/round timestamps, amounts, and locations.**
  Giảm độ chi tiết. Gom nhóm/làm tròn thời gian, số tiền, vị trí.
- **Short retention.** TTL & deletion by default; archive aggregates, not raw.
  Giữ ngắn hạn. TTL & xoá mặc định; lưu tổng hợp thay vì dữ liệu thô..

## Exposure surface → Controls

| Surface | Risk/Example | Control |
|---|---|---|
| API responses | PII/IDs/volumes leak via rich payloads | Projection allow-list; masking; paginated counts (no totals) |
| Errors | Stack traces, SQL, internal hosts | Neutral messages; error IDs; server-side correlation |
| Logs/metrics | PII in logs; raw queries | Redaction; hashing/tokenization; field-level encryption |
| URLs/IDs | Guessable IDs; PII in path/query | Opaque IDs; no PII in URLs; short-lived presigned links |
| Search/reporting | Exact matches expose rare records | K-anonymity buckets; thresholds (min-k); privacy budgets |

## Shape, Mask, Good vs. Bad



# Exposure Minimization — Allowlists over Blocklists

DEFINITION

**Prefer *allowlists* (finite set of safe cases) over *blocklists* (attempt to list all unsafe cases).**

- **Allowlists** draw the line from the "safe side" — omissions are conservative.
  Allowlist kẻ ranh từ phía "an toàn" — thiếu sót vẫn an toàn hơn.
- **Blocklists** are open-ended — omissions become exploitable holes.
  Blocklist là tập mở — mọi thiếu sót biến thành lỗ hổng.

ANALOGY (NON-SOFTWARE)

- It is always safer to list what is allowed than what is prohibited because it is difficult to list all the bad cases.
  - Nêu hoạt động được phép luôn an toàn hơn nêu bị cấm vì khó liệt kê toàn bộ trường hợp xấu.

ACCEPTANCE IN REVIEW

- Controls are **deny-by-default**; policies enumerate allowed items; exceptions are temporary & auditable.
  Kiểm soát ở trạng thái từ chối theo mặc định; chính sách liệt kê mục được phép; ngoại lệ tạm thời & có audit.

EXAMPLES

- **Input validation**: allowlist characters/format (e.g., ^[A-Z0-9_-]{1,32}$) instead of "disallow <script>, …".
  Kiểm tra đầu vào: allowlist ký tự/định dạng (ví dụ regex ở trên) thay vì "cấm <script>, …".
- **File uploads**: accept only MIME/types & extensions explicitly listed (.pdf,.jpg,.png) and verify content magic bytes.
  chỉ chấp nhận loại tệp cho phép và xác minh magic bytes nội dung.
- **Outbound network**: egress rules allow specific hosts/ports; default drop for all else.
  Mạng ra ngoài: chỉ cho phép danh sách máy chủ/cổng cụ thể; mặc định chặn.
- **Mobile intents/deeplinks**: handle a narrow set of schemes/hosts; ignore others.
  Mobile intents/deeplinks: chỉ xử lý tập scheme/host hẹp; bỏ qua phần còn lại.
- **Authorization**: RBAC/ABAC policies enumerate allowed actions per role/attribute; deny unknowns.
  Uỷ quyền: RBAC/ABAC liệt kê hành động được phép theo vai trò/thuộc tính; từ chối phần không xác định.
- **HTML sanitization**: allowlist safe tags/attrs; everything else stripped.
  Lọc HTML: allowlist thẻ/thuộc tính an toàn; loại bỏ phần còn lại.

# Redundancy — Separation of Privilege (a.k.a. Separation of Duty)

DEFINITION

## Two independent parties/controls are required for a critical action.

- Two locks with different keys → stronger than one; keys held by different people/teams.
  Hai ổ khoá với *hai chìa khác nhau* → mạnh hơn một; chìa do *hai người/nhóm* giữ.
- Minimizes insider mistakes/malice; makes subversion slower & more detectable.
  Giảm rủi ro nhầm lẫn/ác ý nội bộ; khiến tấn công khó hơn, dễ bị phát hiện hơn.

WHEN TO APPLY

- Overlapping responsibilities around a protected resource (e.g., datacenter ops vs. physical access).
  Khi có trách nhiệm chồng lấn quanh tài nguyên cần bảo vệ (vd: vận hành máy chủ vs. kiểm soát vật lý).
- Critical operations with catastrophic impact (e.g., bulk-data export; destructive admin; nuclear/financial triggers).
  Tác vụ trọng yếu có tác động nghiêm trọng (vd: xuất dữ liệu hàng loạt; lệnh quản trị huỷ dữ liệu; kích hoạt hệ thống).

KEY PRINCIPLES

- Different chains of command (reporting to different executives) to reduce collusion risk.
  Chuỗi báo cáo độc lập (khác cấp quản lý) để giảm rủi ro thông đồng.
- People on the floor do not have admin credentials; remote admins have no physical access.
  Người tại chỗ không có thông tin đăng nhập admin; admin từ xa không có quyền vào vật lý.
- Critical maintenance requires both presence (badge) and approval (JIT token via gateway).
  Bảo trì trọng yếu cần cả hiện diện (thẻ) và phê duyệt (token JIT qua gateway).

IMPLEMENTATION GUIDE

## Designing for Separation of Privilege

- **Dual approvals** for critical actions; enforce *concurrency window* (e.g., ≤ 60s) and *independent factors* (MFA + separate role).
  Hai phê duyệt cho tác vụ trọng yếu; cưỡng chế *cửa sổ đồng thời* (vd ≤ 60s) và *yếu tố độc lập* (MFA + vai trò khác).
- **Split key custody**: different KMS keyrings/owners; avoid one team holding all keys.
  Chia quyền giữ khoá: keyring/KMS khác nhau và người sở hữu khác; tránh một nhóm giữ hết.
- **Separate org lines**: controls owned by teams reporting to different executives.
  Tách tuyến quản trị: kiểm soát thuộc các nhóm báo cáo lên lãnh đạo khác nhau.
- **Gateway logging** for admin access; consider *brokered SSH* with full session capture.
  Ghi log tại cổng cho truy cập admin; cân nhắc *SSH qua broker* ghi lại toàn bộ phiên.
- **Break-glass** path is *fail-closed (*automatically shuts down or blocks access to prevent further operation or unauthorized entry*)*, noisy (alerts), and requires post-mortem.
  Đường "đập kính" phải *fail-closed (tự ngắt hệ thống)*, tạo ồn (cảnh báo), và bắt buộc hậu kiểm.

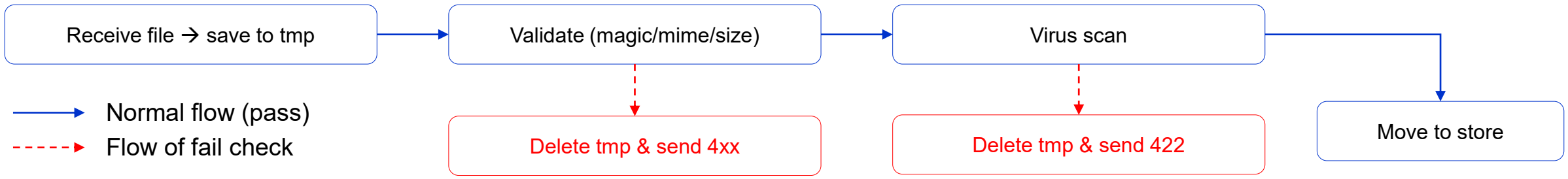
# Exposure Minimization — Fail Securely

DEFINITION

**If a problem occurs, the system must end up in a secure state (deny-by-default, no dangerous leftovers).**

- Deny-by-default on error: timeouts, missing proofs, partial failures ⇒ reject (fail-closed).
  Mặc định từ chối khi lỗi: timeout/thiếu bằng chứng/lỗi một phần ⇒ từ chối.
- Validate early, purge on failure: e.g., image upload — if malformed, immediately delete temp & return 4xx.
  Kiểm tra sớm, xoá khi hỏng: ví dụ upload ảnh — sai định dạng thì xoá tệp tạm và trả 4xx.
- Atomicity & rollback: all-or-nothing writes; use temp+rename or transactions to avoid partial state.
  ghi tất cả hoặc không; dùng tệp tạm+đổi tên hoặc giao dịch.
- Least disclosure on errors: consistent error codes, no stack traces/keys; log details to server only.
  Tiết lộ tối thiểu: mã lỗi nhất quán, không lộ stack trace/khoá; chi tiết chỉ ghi log máy chủ.
- Safe retries: idempotency keys; circuit breakers degrade to safe mode instead of risky bypass.
  Thử lại an toàn: khoá idempotency; circuit breaker chuyển chế độ an toàn thay vì đi tắt rủi ro.
- Compound errors: when multiple faults combine, the result remains secure (still closed/clean).
  Lỗi chồng: nhiều lỗi cùng lúc vẫn phải an toàn (đóng/sạch).

EXAMPLE – FILE UPLOAD PIPELINE (fail-closed)



# Trust & Responsibility — Reluctance to Trust · Accept Security Responsibility

DEFINITION

**Reluctance to Trust**

- Grant *trust* only as an explicit choice backed by strong *evidence*. Apply this to verifying the authenticity of code/config before installing, requiring *strong authentication* before *authorization*, and never assuming client-supplied data is truthful.
  Chỉ trao *trust* khi có *bằng chứng* đủ mạnh. Áp dụng: xác minh tính chính danh của mã/cấu hình trước khi cài; yêu cầu xác thực mạnh trước khi cấp quyền; không mặc định tin dữ liệu từ client.

Example:

- **Ignore client totals/prices.** Server recomputes line prices and grand total from its own price list and rules.
  Không tin tổng tiền/đơn giá do client gửi. Server tự tính theo bảng giá & chính sách của hệ thống.
- **Validate coupon/discount server-side.** Enforce eligibility, expiry, stacking rules, and per-user limits.
  Coupon/discount kiểm tra phía server. Áp điều kiện hợp lệ, hết hạn, cộng dồn, giới hạn theo người dùng.
- **Require an idempotency key.** Prevent duplicate orders on retries.
  Bắt buộc idempotency key. Tránh tạo đơn trùng khi người dùng bấm lại.
- **Never trust inventory or credit flags from the client.** Check stock and credit limit on the server.
  Không tin cờ tồn kho hoặc tín dụng từ client. Kiểm tra tồn & hạn mức tín dụng phía server.

DEFINITION

**Accept Security Responsibility**

- Every role (designer/dev/ops) actively owns security duties in their scope, expressed via requirements, interface contracts, and verifiable *evidence*.
  Mọi vai trò (thiết kế/phát triển/vận hành) chủ động nhận trách nhiệm bảo mật, thể hiện bằng yêu cầu, hợp đồng giao tiếp và *bằng chứng* xác minh.

Example:

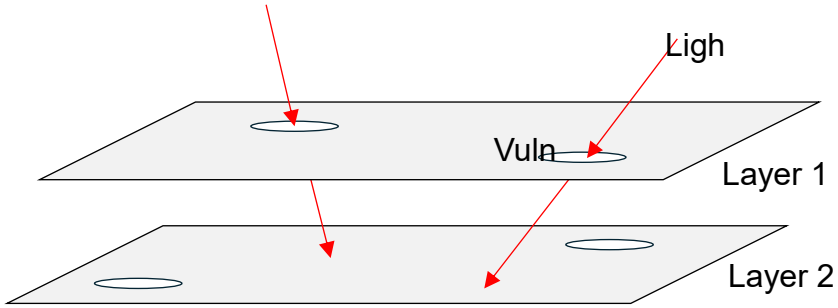
- **POS/Web (Component A) guarantees:** schema-valid payload (customer_id,items[{sku, qty}],coupon?), idempotency-key, no price fields; client-side checks are best-effort only.
  POS/Web (Thành phần A) đảm bảo: payload đúng schema, có idempotency-key, *không* gửi trường giá; kiểm tra phía client chỉ hỗ trợ.
- **Order Service (Component B) guarantees:** recompute prices, validate coupon, check stock & credit, enforce authZ, persist atomically, return canonical order.
  Order Service (Thành phần B) đảm bảo: tính lại giá, kiểm tra coupon, tồn kho & tín dụng, kiểm soát phân quyền, ghi nhận giao dịch nguyên tử, trả kết quả chuẩn.
- Both: audit logs, PII redaction in errors, metrics/alerts.
  Cả hai: log kiểm toán, ẩn PII trong lỗi, số đo & cảnh báo.

# Redundancy — Defense-in-Depth : Layered Protections

PRINCIPLE

**Critical decisions must not rely on a single barrier.**

- **Independent layers**: strong *authentication* + policy-based *authorization* + guarded *interfaces* + *input validation* + *sandboxed execution* + *monitoring & accountability*.
  Nhiều lớp độc lập: *xác thực* mạnh + *uỷ quyền theo chính sách* + *rào chắn giao diện* + *kiểm tra đầu vào* + *thực thi trong sandbox* + *giám sát/quy trách nhiệm*.
- **Guard / bottleneck**: route all entry paths through a *choke point* so controls are consistent.
  Điểm nghẽn (guard): dồn mọi đường vào qua *điểm kiểm soát* để áp chính sách đồng nhất.
- **Useful redundancy**: multiple layers catch faults; if one is bypassed, others remain.
  Dự phòng hữu dụng: nhiều lớp cùng bắt lỗi; một lớp hỏng, lớp khác vẫn đứng vững.
- **Avoid false confidence**: layers should be *independent* and address *different risk aspects*.
  Tránh an toàn giả: các lớp phải *độc lập* và kiểm soát *khía cạnh rủi ro khác nhau*.




EXAMPLE — HIGH-RISK ACTION: “REFUND PAYMENT”

1. **AuthN**: user must be *MFA-confirmed* recently.
   Xác thực: bắt buộc *MFA* gần đây.
2. **AuthZ**: *PDP* evaluates policy can_refund(amount, tenant) with context (role, risk, geo, business hours).
   Uỷ quyền: *PDP* quyết theo policy và ngữ cảnh (vai trò, rủi ro, vùng, khung giờ).
3. **Guarded API**: request goes through *API Gateway* (rate-limit, schema contract, anti-automation).
   API có rào chắn: qua *Gateway* (giới hạn tần suất, ràng buộc schema, chống tự động hoá).
4. **Input validation**: server-side checks (amount ≥ 0, currency allowed, id format, CSRF token).
   Kiểm tra đầu vào: kiểm tra phía server (số tiền ≥ 0, loại tiền hợp lệ, định dạng id, CSRF token).
5. **Sandboxed execution**: isolate refund plugin/script in a restricted sandbox (no network except allowlist).
   Sandbox: cô lập script hoàn tiền trong sandbox hạn chế (không mạng trừ allowlist).
6. **Data access guard**: DB via prepared statements + row-level policy; secrets via broker; immutable ledger append.
   Canh gác dữ liệu: DB dùng prepared statement + policy mức dòng; bí mật qua broker; ghi sổ bất biến (append-only).
7. **Monitoring & accountability**: structured audit (who/what/when/why), anomaly alerts, and *compensating holds* for large refunds.
   Giám sát & trách nhiệm: audit có cấu trúc (ai/cái gì/khi nào/tại sao), cảnh báo bất thường, và *giữ bù trừ* với khoản lớn.

# Exposure Minimization — Avoid Predictability

DEFINITION & RISK

**Any predictable data or behavior leaks information**

- Attackers can infer, enumerate, or pre-claim resources if IDs/tokens follow patterns (counters, timestamps, PII-derived).
  Kẻ tấn công có thể suy ra, liệt kê, hoặc chiếm trước tài nguyên nếu ID/token theo quy luật (bộ đếm, dấu thời gian, lấy từ PII).
  Example:

COMMON PITFALLS

- Auto-increment IDs (1,2,3…) expose total records and enable future-ID guessing.
  ID của bản ghi tăng dần (1,2,3…) lộ tổng số và cho phép đoán ID tiếp theo.
- IDs derived from PII (name initials, ZIP/postcode) leak identity clues.
  ID tạo từ PII (chữ cái tên, mã bưu chính) rò rỉ thông tin nhận dạng.
- Predictable tokens (reset codes, invite codes, order numbers from timestamps) allow guessing/bruteforce.
  Token dễ đoán (mã đặt lại, mã mời, số đơn dựa timestamp) dễ bị đoán/bruteforce.

DESIGN PLAYBOOK

- **Use CSPRNG-based opaque IDs** for anything public (customer, order, reset tokens).
  Target ≥128 bits entropy.Dùng ID mờ dựa CSPRNG cho định danh công khai (khách hàng, đơn hàng, token). Nhắm ≥128 bit entropy.
- **Separate internal PKs from public IDs.** Keep auto-increment PKs private; expose a random.
  Tách PK nội bộ khỏi ID công khai. Giữ PK tăng dần ở nội bộ; công khai
- **No PII in IDs.** Never encode names, location, or account metadata into identifiers.
  Không đưa PII vào ID. Tránh mã hoá tên, địa điểm, hay metadata tài khoản.
- **Harden enumeration.** Use rate-limit, CAPTCHA (where appropriate), and 404/403 that don’t reveal existence.
  Chống liệt kê. Giới hạn tần suất, CAPTCHA (khi hợp lý), và 404/403 không tiết lộ tồn tại.
- **Be mindful of length leakage.** Length reveals keyspace size; usually OK, but avoid tiny spaces.
  Lưu ý rò rỉ độ dài. Độ dài tiết lộ không gian khoá; thường chấp nhận, nhưng tránh không gian nhỏ.

# Anti-Patterns — What Not to Build

DEFINITION & RISK

## What not to build

- **Anti-patterns** are recurring design choices that *increase risk* even when they are not immediate vulnerabilities. Recognize and replace them early.
  Phản mẫu là các lựa chọn thiết kế lặp lại *làm tăng rủi ro* dù chưa là lỗ hổng ngay lập tức. Hãy nhận diện và thay thế sớm.

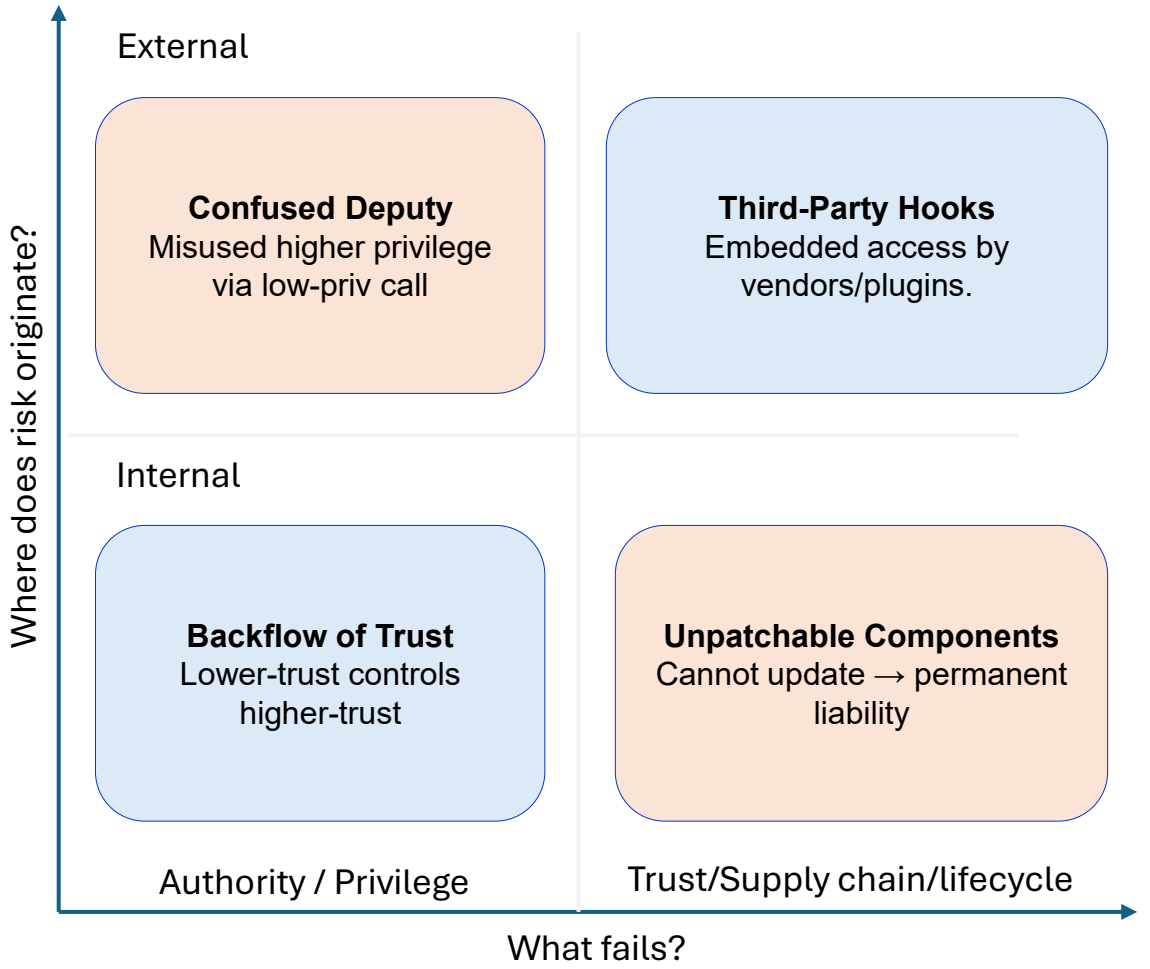
**Rule of thumb.** If trust/authority flows the wrong way, or components cannot be maintained, you are likely in an anti-pattern.
Gợi ý. Nếu luồng tin cậy/đặc quyền chảy ngược, hoặc thành phần không thể bảo trì, rất có thể bạn đang gặp phản mẫu.

## Why it matters

- Vì sao quan trọngAnti-patterns amplify threats and obscure accountability.
  Phản mẫu khuếch đại đe doạ và làm mờ trách nhiệm.
- They create brittle trust boundaries and unfixable debt.
  Tạo ranh giới tin cậy mong manh và nợ kỹ thuật khó sửa.

DEFINITION & RISK

## Map anti-patterns by the kind of failure and where risk originates.



# Anti Pattern — Confused Deputy

DEFINITION

## What not to build

- **Confused Deputy** happens when a *higher-privilege callee* (the deputy) performs actions for a *lower-privilege caller* without carrying caller identity/intent, thereby misusing its own authority.
  Confused Deputy xảy ra khi *thành phần đặc quyền cao* thực thi thay *bên gọi đặc quyền thấp* mà không mang danh tính/mục đích của bên gọi, dẫn tới lạm dụng thẩm quyền.

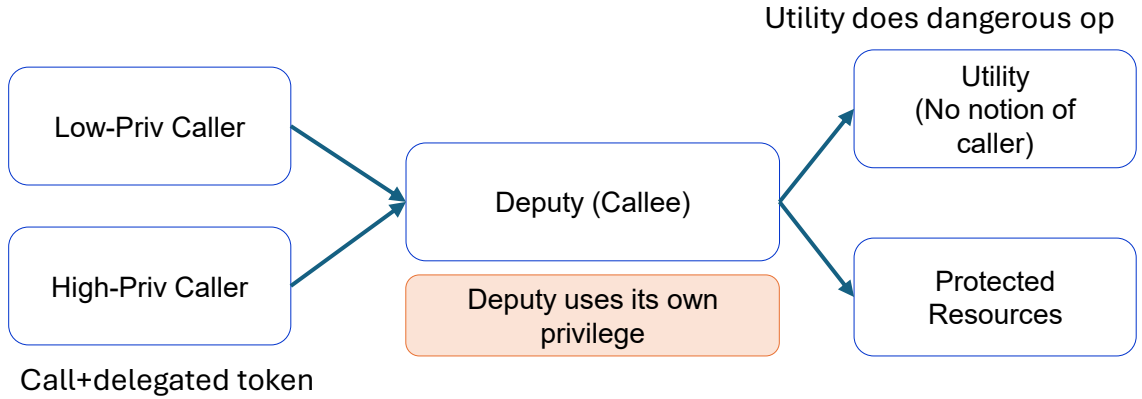
**Intention & Malice**

- Trustworthiness needs both honesty and competence. Confused Deputy often exploits *competence gaps* (lost context, weak validation) in otherwise honest code; users may also be tricked into trusting malicious software.
  Đáng tin đòi hỏi cả trung thực và năng lực. Confused Deputy thường khai thác *thiếu năng lực* (mất ngữ cảnh, kiểm tra yếu) trong mã trung thực; người dùng cũng có thể bị lừa tin phần mềm độc hại.

**Typical root causes**

- Nguyên nhân gốc điển hìnhCaller context lost across layers (identity, intent, resource ownership).
  Mất ngữ cảnh bên gọi qua các lớp (danh tính, mục đích, quyền sở hữu tài nguyên).
- Shared code paths used by both high/low privilege without guardrails.
  Đường mã dùng chung cho cao/thấp mà không rào chắn.
- Implicit trust in side-effects (e.g., logging, cache warms) writable by deputy.
  Tin tưởng ngầm vào tác dụng phụ (log, cache) do deputy ghi.

VISUALIZE – CONFUSED DEPUTY




EXAMPLES

- **CSRF on Admin Action.** Browser auto-sends cookies; admin visits attacker page → backend (deputy) performs admin action w/ admin's privilege.
  CSRF ở hành động quản trị. Trình duyệt tự gửi cookie; admin ghé trang kẻ tấn công → backend (deputy) thực thi hành động quản trị bằng đặc quyền của admin.
- **Log wiping via side-effect.** Requester supplies malformed string; deputy writes it to log → corrupts or injects control sequences to render logs unreadable.
  Xoá log qua tác dụng phụ. Yêu cầu đưa chuỗi lỗi định dạng; deputy ghi vào log → hỏng log hoặc chèn ký tự điều khiển làm log không đọc được.
- **Microservice refund.** API Gateway calls RefundService using its own broad token; user identity not propagated → refunds arbitrary orders.
  Hoàn tiền vi dịch vụ. Gateway gọi RefundService bằng token rộng của chính nó; không truyền danh tính người dùng → hoàn tiền sai đơn.

# Anti Pattern — Backflow of Trust · 3rd Party Hooks · Unpatchable Components

## Backflow of Trust

- **Definition.** A lower-trust component *controls* or *hosts* higher-tru st operations (structural elevation-of-privilege waiting to happen).
  Định nghĩa. Thành phần tin cậy thấp *điều khiển/lưu trữ* thao tác tin cậy cao (một dạng nâng quyền về cấu trúc chực chờ xảy ra).

**Examples:**

- Unmanaged personal phone → admin mobile app. Admin approves refunds from a personal phone not enrolled in MDM/Compliance; the low-trust device controls high-trust admin APIs.
  Điện thoại cá nhân không quản lý → ứng dụng admin. Admin duyệt hoàn tiền từ điện thoại không MDM/tuân thủ; thiết bị tin cậy thấp điều khiển API quản trị tin cậy cao.
- Admin web console with unvetted extensions. Admin uses a browser with untrusted extensions to access the admin dashboard; extensions (low trust) can trigger privileged actions.
  Bảng điều khiển web với extension không tin cậy. Admin dùng trình duyệt có extension không rõ nguồn gốc; extension (tin cậy thấp) có thể kích hoạt hành động đặc quyền.

## Third-Party Hooks

- **Definition.** Embedded hooks/tunnels give vendors *undue access* into sensitive internals, bypassing enterprise oversight.
  Định nghĩa. Móc nối/đường hầm nhúng trao *quyền truy cập quá mức* cho nhà cung cấp vào nội bộ nhạy cảm, vượt ngoài giám sát doanh nghiệp.

**Examples:**

- Live chat widget on admin pages. A chat bubble (3rd-party JS) loads on the admin dashboard and can read what the admin sees/types.
  Widget chat trên trang quản trị. Bong bóng chat (JS bên thứ ba) tải trên bảng điều khiển quản trị và có thể đọc nội dung mà admin xem/nhập.
- “Buy Now” button script. A vendor drop-in script controls checkout; if changed at the vendor, it could skim card/customer data.
  Script nút “Mua ngay”. Script nhúng của nhà cung cấp điều khiển thanh toán; nếu bị thay đổi phía vendor có thể hút dữ liệu thẻ/khách hàng.

## Unpatchable Components

- **Definition.** Any component you cannot update promptly becomes a *permanent liability* once vulnerabilities surface.
  Định nghĩa. Bất kỳ thành phần nào không thể cập nhật kịp thời sẽ trở thành *gánh nặng vĩnh viễn* khi lộ lỗ hổng.
- In short, "Unpatchable Components" is usually not a programming error on your part, but rather a dependency on a library, framework, or runtime that the vendor has "abandoned" and no longer supports security.
  "Unpatchable Components" thường không phải là lỗi lập trình của bạn, mà là lỗi do sự phụ thuộc (dependency) của bạn vào một thư viện, framework, hoặc runtime mà nhà cung cấp đã "bỏ rơi" và không còn hỗ trợ bảo mật.

**Examples:**

- Problem: AngularJS 1.x is EOL in July 2021.Scenario: If a new Cross-Site Scripting (XSS) vulnerability is found in the core of the AngularJS 1.x library, Google (the creator) will not release a patch. Any website still using the library will be permanently vulnerable until they upgrade or completely rewrite their frontend.
  Nếu một lỗ hổng Cross-Site Scripting (XSS) mới được tìm thấy trong *lõi* của thư viện AngularJS 1.x, Google (đơn vị tạo ra nó) sẽ không phát hành bản vá. Bất kỳ trang web nào vẫn còn sử dụng thư viện này sẽ vĩnh viễn tồn tại lỗ hổng đó cho đến khi họ nâng cấp hoặc viết lại hoàn toàn frontend của mình.

# From Threats to Controls

EXAMPLE

## STRIDE → Misuse → Vulnerability

**Context:** Refund flow: User → API GW → Refund Svc → Payment/DB.
**Misuse:**

1. Attacker submits refund for someone else's orderId.
2. Gateway authenticates, but Refund Svc uses a service token—no caller identity.
3. Predictable orderId → unauthorized access/refund.
    1. Kẻ tấn công gửi refund cho *orderId* của người khác.
    2. Gateway xác thực nhưng Refund Svc dùng *service-token*, không mang danh tính caller.
    3. orderId đoán được → truy xuất & hoàn sai.

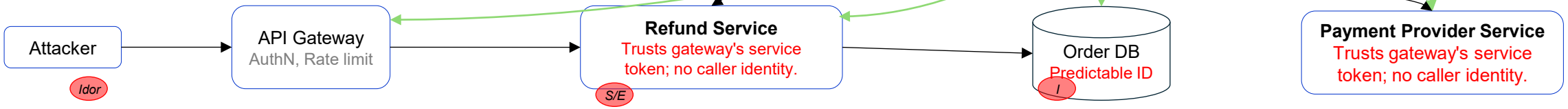
**STRIDE:** Spoofing/EoP | Information Disclosure | Tampering

**Vulnerabilities:**

- Missing complete mediation at critical boundary; scattered checks.
  Không *Complete Mediation* ở biên quan trọng; kiểm tra rải rác.
- Caller identity not propagated → Confused Deputy.
  Không truyền danh tính caller → *Confused Deputy*.
- Guessable orderId; missing record-level authorization.
  *orderId* dễ đoán; thiếu kiểm tra quyền truy cập bản ghi.
- 3P webhook lacks allowlist/signing.
  Webhook 3P không *allowlist*/kí payload.

## Selected Patterns

- Complete Mediation (API GW as a single choke point)
  Hoàn kiểm trung gian (API GW là single choke-point)
- Propagate caller identity (delegation tokens)
  Truyền danh tính caller (delegation tokens)
- Strong authorization (PDP/PEP; record-level ABAC/RBAC)
  Uỷ quyền mạnh (PDP/PEP; ABAC/RBAC cấp bản ghi)
- Opaque, high-entropy IDs (≥128-bit)
  Định danh mờ/khó đoán (opaque IDs ≥128-bit)
- Least information (mask PII; non-revealing errors)
  Tiết lộ tối thiểu (mask PII; lỗi không tiết lộ)
- Webhook allowlist + signed payload/cert pinning
  Allowlist webhook + ký payload/cert pinning
- Bulkheads/circuit breakers + rate limiting
  Ngăn khoang/cầu chì + rate-limit
- Tamper-evident audit & decision logs
  Audit chống giả mạo & nhật ký quyết định



# Mitigations & Security Design Patterns

## Part 3
## Labs

# Mitigations & Security Design Patterns

## SUMMARY

FPT Education
FPT UNIVERSITY

# Example of Smell/Anti-Patterns Catalog

## Smells/anti-patterns with brief examples and likely STRIDE categories

| SMELL / ANTI-PATTERN | DESCRIPTION & EXAMPLE | LIKELY STRIDE |
|---|---|---|
| **Confused Deputy** | Service uses another service's authority instead of the callers. *Example:* Refund Service trusts API GW service token; no caller identity. | SER |
| **Backflow of Trust** | Unverified third-party input is treated as trusted inside the boundary. *Example:* Gateway-forwarded claims not re-validated. | STI |
| **IDOR / Predictable IDs** | Guessable identifiers enable unauthorized access or enumeration. *Example:* Sequential *orderId* exposed publicly. | ISE |
| **Verbose Errors / Overexposed Responses** | Errors or responses reveal sensitive internals/PII. *Example:* stack traces and full objects returned by default. | IR |
| **Unbounded Webhook / 3P Plugin** | Callbacks to arbitrary domains with free-form payloads and no provenance. *Example:* Payment confirmation webhook can be spoofed. | TSI |
| **Global Shared Cache/Mutex (LCM Violation)** | Cross-tenant shared state enables leakage and privilege bleed. *Example:* Shared cache keyspace across tenants. | IE |
| **No Rate Limiting / Admission Control** | Traffic spikes/floods overwhelm components; no prioritization or shedding. *Example:* Bot flood cripples refund API. | D |
| **No Idempotency** | Retries duplicate side-effects. *Example:* Duplicate invoices/refunds under transient failures. | TR |
| **Weak Auditability** | Actions cannot be attributed or verified. *Example:* Missing decision logs; tamperable audit trail. | RI |
| **Overprivileged Service Account** | Tokens/roles grant far more permissions than needed. *Example:* Service can perform admin-level refunds. | ER |
| **Incomplete/Scattered Authorization Checks** | Enforcement is inconsistent across boundaries; gaps emerge. *Example:* Some endpoints enforce policy only at UI. | SIE |

FPT Education
FPT UNIVERSITY

# STRIDE x Patterns – Matrix

| STRIDE | DESIGN ATTRIBUTES | EXPOSURE MINIMIZATION | STRONG ENFORCEMENT | REDUNDANCY | TRUST & RESPONSIBILITY |
|---|---|---|---|---|---|
| **Spoofing** | •Simplicity & transparency<br>•Least common mechanism | •Rate-limit / Backpressure<br>•Secure by default | •MFA, mTLS/OIDC<br>•Complete Mediation | •Step-up re-auth<br>•Defense-in-Depth | •Propagate caller identity<br>•Clear trust boundaries |
| **Tampering** | •Immutability for config/logs<br>•Economy of mechanism | •Allowlist I/O<br>•Integrity tags/hashes | •Strict input contracts<br>•Signed requests / HMAC | •Idempotency keys<br>•Write-through audit | •SoD for sensitive changes<br>•Change approvals |
| **Repudiation** | •Transparent design<br>•Action identity tags | •Least disclosure on error<br>•Minimize PII in logs | •Policy-decision logs (PDP/PEP)<br>•AuthN + signed logs | •WORM/append-only<br>•Tamper-evident seals | •Time sync (NTP)<br>•Investigation playbooks |
| **Information Disclosure** | •Privacy by design<br>•Data classification | •Least information<br>•Mask/omit PII | •Complete Mediation (record-level ABAC/RBAC)<br>•TLS/mTLS, token scope | •Tokenization/FPE<br>•Repeat checks at boundaries | •Data-handling policies<br>•Security training |
| **Denial of Service** | •Fail securely<br>•Graceful degradation | •Rate-limit / Backpressure<br>•Quota / token bucket | •Admission control<br>•Human traffic priority | •Bulkheads & Circuit breakers<br>•Timeouts / load shedding | •Egress controls to avoid amplification<br>•Incident runbooks |
| **Elevation of Privilege** | •Least common mechanism<br>•Simplicity | •Least privilege<br>•Separation of Duties (SoD) | •Strong authorization (ABAC/RBAC)<br>•Complete Mediation | •Two-person rule<br>•Defense-in-Depth | •Break-glass controls & audit<br>•Clear role boundaries |

FPT Education
FPT UNIVERSITY

# Example of Residual Risk and Bug Bar

Residual Risk ≈ Inherent Risk × (1 − Control Effectiveness) + Control Side-effects

Decision: P0 – Block at Gate , P1- block at FSR (temporarily accept at SDR) , P2 - accept with conditions

| THREAT | PRIMARY MITIGATIONS | RESIDUAL RISK | BUG BAR @ SDR (DESIGN GATE) | BUG BAR @ FSR (PRE-SHIP GATE) |
|---|---|---|---|---|
| Confused Deputy (Refund) | •Complete Mediation (PEP at Gateway)<br>•Propagate caller identity (delegation tokens)<br>•Strong authorization (record-level PDP/PEP) | Some internal hops may still drop the *end-user principal* ⇒ tenant-scoped mis-refunds if policy lags. | P1 acceptable at SDR<br>•Architecture specifies choke point & delegation across all hops. | P0 blocker if missing<br>•Block if any path lacks caller identity; Accept when 100% go via PEP + decision logs + daily cap. |
| IDOR / Predictable IDs | •Opaque IDs (≥128-bit)<br>•Record-level authorization<br>•Least information | Side-channel enumeration (timing/latency) may remain, limited to metadata. | P2 at SDR<br>•Design decouples publicId/internal pk; entropy evidenced. | P0 if cross-record access<br>•Accept if anti-enumeration RL active & response shaping correct. |
| Unbounded Webhook / 3P Plugin | •Allowlist domain/IP + signed payloads<br>•Strict input contracts<br>•Cert pinning & replay window | Compromised 3P may send syntactically valid yet harmful payloads. | P1 at SDR<br>•Require allowlist + signatures + schema versioning. | P0 if wildcard/unsigned<br>•Accept if signature 100%, schema fail-closed, replay ≤5 minutes. |
| DoS (No Rate-limit / Backpressure) | •Rate-limit & backpressure<br>•Bulkheads / Circuit breakers<br>•Admission control & fail securely | Controlled degradation for non-core; core SLO preserved. | P2 at SDR<br>•Layered RL architecture & pool isolation (bulkheads). | P0 if core SLO breaks<br>•Accept if 95p/99p within limits under stress; shed only tier-3. |
| Weak Auditability / Repudiation | •Tamper-evident audit (WORM)<br>•Policy-decision logs (PDP/PEP)<br>•Time synchronization | Log latency (≤5′) hampers realtime view but preserves traceability. | P2 at SDR<br>•Append-only + hash chain; map trace-id ↔ actor. | P0 if gaps/editable logs<br>•Accept if completeness = 100% for sensitive paths + tamper checks pass. |

# Summary & Integration Roadmap into the SDLC

## SUMMARY

Key idea. Mitigations connect *Threat Modeling* with *Secure Design*: focus on structural levers (attack surface, windows of exposure, data exposure), implement via security *patterns*, avoid *anti-patterns*, and enforce through *guards/bottlenecks* to achieve Complete Mediation.
Ý chính. Mitigation nối *Threat Modeling (Ch.2)* với *Thiết kế bảo mật*: ưu tiên đòn bẩy cấu trúc (bề mặt tấn công, cửa sổ phơi bày, lộ dữ liệu), triển khai bằng *pattern*, tránh *anti-pattern*, và ràng buộc qua *guard/bottleneck* để đạt Complete Mediation.

### Structural Mitigations to Prioritize

- **Attack surface** ↓: remove entry points, centralize via gateway/PEP, least privilege, network segmentation.
  Giảm bề mặt tấn công: loại điểm vào, gom qua gateway/PEP, tối thiểu đặc quyền, phân đoạn mạng.
- **Windows of exposure** ↓: patch cadence, staged rollout, feature flags/kill-switch, rapid revoke.
  Thu hẹp cửa sổ phơi bày: nhịp vá, phát hành từng bước, cờ tính năng/tắt khẩn, thu hồi nhanh.
- **Data exposure** ↓: data minimization, field-level encryption/tokenization, mTLS, privacy budgets.
  Giảm lộ dữ liệu: tối thiểu dữ liệu, mã hoá/đổi token theo trường, mTLS, ngân sách quyền riêng tư.
- **Patterns**: Defense in Depth, Complete Mediation, Fail Securely, Secure by Default, Reluctance to Trust.
  Pattern: Phòng thủ theo lớp, Kiểm soát toàn diện, Fail Securely, Secure by Default, Thận trọng với tin cậy.
- **Avoid anti-patterns**: Backflow of Trust, Third-Party Hooks, Unpatchable Components, Confused Deputy.
  Tránh anti-pattern: Dòng tin cậy chảy ngược, Móc nối bên thứ ba, Thành phần không vá được, Phó nhầm việc.
- **Guard/Bottleneck**: API Gateway + centralized AuthZ (PDP/PEP), service mesh policy, egress filters.
  Điểm chặn/điểm cổ chai: Gateway + uỷ quyền tập trung (PDP/PEP), policy mesh, lọc egress.