Pertemuan 14 — *Data Security*

Mengamankan platform data dari *ingest* sampai *serving*

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. Ancaman nyata pada platform data
2. Prinsip: CIA, *least privilege*, *defense in depth*, *zero trust*
3. *Encryption in transit*: TLS dan mTLS
4. *Encryption at rest*: SSE, *envelope encryption*, KMS
5. Manajemen *secret*: Vault
6. *Access control*: IAM, RBAC, ABAC, *row/column-level security*
7. Data pribadi: *masking*, *tokenization*, pseudonimisasi, anonimisasi
8. Audit log, *crypto-shredding*, dan *security* Spark
9. *When to use* dan *best practice*
10. Praktikum: mengamankan *pipeline* bank syariah

Sasaran pertemuan ini

Anda dapat menunjukkan, untuk setiap *layer pipeline*, **siapa boleh membaca apa**, **bagaimana data dilindungi** saat dikirim dan disimpan, dan **bukti** bahwa kontrol itu benar-benar bekerja.

Ancaman nyata pada platform data

Platform data adalah sasaran yang menggiurkan

Database aplikasi hanya menyimpan data satu layanan. Platform data **mengumpulkan semuanya di satu tempat**: identitas nasabah, transaksi, pembiayaan, perilaku di aplikasi.

Konsekuensinya

Satu kredensial *data lake* yang bocor bisa membuka data yang jauh lebih banyak daripada satu *database* produksi. Nilai kumpulan data yang sudah bersih dan tergabung justru lebih tinggi bagi penyerang.

Siapa yang membaca *lakehouse* bank?

- *Data engineer* (menulis *pipeline*)
- *Analyst* dan *data scientist*
- Akun layanan: Airflow, Spark, dbt, *dashboard*
- *Vendor* dan konsultan
- Auditor dan divisi kepatuhan

Tiap pembaca adalah pintu. Makin banyak pintu tanpa kunci yang tepat, makin besar *attack surface*.

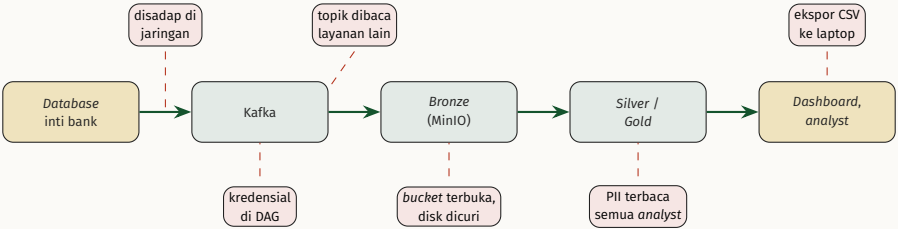
Lima pola *data breach* yang paling sering

Pola	Contoh	Kontrol utama
***Bucket* publik**	*Bucket* berisi ekspor nasabah dibuka "sementara" untuk *vendor*, lalu lupa ditutup	*Block public access*, *bucket policy*, pemindaian *configuration*
Kredensial di kode	*Access key* MinIO tertulis di DAG Airflow lalu masuk repositori Git	Vault, *secret scanning*, kredensial berumur pendek
Akses berlebih	Semua *analyst* memakai satu akun *superuser* ke *warehouse*	RBAC, *least privilege*, akun per orang
Orang dalam	Pegawai menyalin data nasabah sebelum keluar dari perusahaan	*Masking*, *tokenization*, audit log, deteksi anomali akses
Rantai pasok	Paket Python atau *image* Docker yang disusupi kode jahat masuk ke *pipeline*	*Pin* versi, pemindaian dependensi, *image* bertanda tangan

Ringkasan

Hampir semua pola di atas berasal dari **kesalahan *configuration* dan kebiasaan**, sehingga sebagian besar bisa dicegah dengan rancangan yang disiplin.

Memetakan ancaman pada *pipeline* kita



Latihan cepat: untuk tiap kotak merah, sebutkan satu kontrol yang mencegah dan satu yang **mendeteksi**. Mencegah saja tidak cukup, kita juga harus tahu bila kontrol jebol.

Prinsip keamanan

CIA, dan apa artinya untuk data

Confidentiality

Hanya yang berhak dapat membaca. Pada bank: NIK, saldo, dan riwayat pembiayaan tidak boleh terbaca orang yang tidak membutuhkannya.

Alat: *encryption*, *access control*, *masking*.

Integrity

Data tidak diubah tanpa izin dan perubahan dapat ditelusuri. Angka di laporan regulator harus sama dengan transaksi sumbernya.

Alat: *checksum*, tanda tangan, *lineage*, *time travel*, audit log.

Availability

Data tersedia saat dibutuhkan. Sistem *fraud* yang tidak bisa membaca skor juga sebuah insiden keamanan.

Alat: *replication*, *backup*, DR, perlindungan dari *ransomware*.

Ringkasan

Keamanan data mencakup integritas dan *availability* selain kerahasiaan. *Pipeline* yang diam-diam mengubah angka (integritas) atau berhenti saat dibutuhkan (*availability*) sama berbahayanya.

Tiga prinsip rancangan

Least privilege

Setiap orang dan akun layanan hanya mendapat hak minimum untuk tugasnya, selama waktu yang dibutuhkan.

Contoh: akun dbt boleh menulis `silver.*` dan `gold.*`, tetapi tidak boleh menghapus `bronze.*`.

Defense in depth

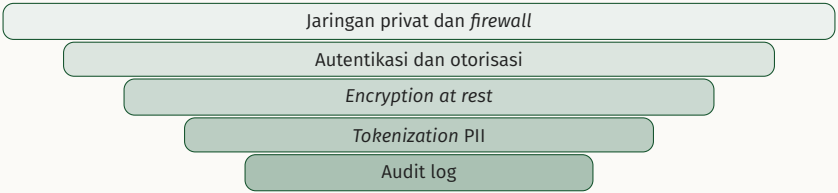
Berlapis: bila satu kontrol gagal, *layer* lain masih menahan.

Contoh: *bucket* privat **dan** terenkripsi **dan** kolom NIK ditokenisasi **dan** setiap pembacaan tercatat.

Zero trust

Jangan percaya karena "sudah di dalam jaringan kantor". Setiap permintaan diautentikasi, diotorisasi, dan dienkripsi.

Contoh: Spark ke Kafka tetap memakai mTLS walau satu *cluster*.

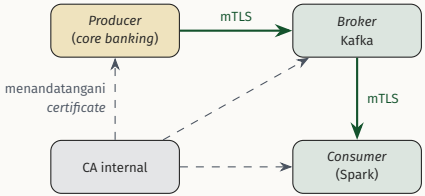


Encryption in transit

TLS dan mTLS di dalam platform data

Cara kerja TLS (*handshake*, *certificate* X.509, rantai kepercayaan) sudah dibedah di Sistem Terdistribusi P3. Di sini kita **memasangnya** pada komponen data.

- **TLS**: klien memverifikasi server (“saya benar bicara dengan Kafka bank”).
- **mTLS**: server juga memverifikasi klien (“yang menulis ke topik ini benar layanan *core banking*”).
- Jalur yang wajib terenkripsi: aplikasi → Kafka, Kafka ↔ Kafka (*replication*), Spark → MinIO, *dashboard* → *warehouse*.



Jebakan umum

Certificate kedaluwarsa menghentikan *pipeline* tengah malam. Otomatiskan penerbitan dan pembaruan (cert-manager, Vault PKI) dan pantau tanggal kedaluwarsanya.

Configuration TLS pada klien Kafka

```
from confluent_kafka import Producer

p = Producer({
  "bootstrap.servers": "kafka:9093",
  "security.protocol": "SSL",
  "ssl.ca.location": "/certs/ca.crt",
  # mTLS: identitas klien
  "ssl.certificate.location":
      "/certs/corebanking.crt",
  "ssl.key.location":
      "/certs/corebanking.key",
  "enable.idempotence": True,
})
p.produce("transaksi", key=b"REK-001",
          value=payload)
```

- *Port* 9093 khusus SSL; *port plaintext* dimatikan di produksi.
- Kunci privat klien **tidak** disimpan di repositori, melainkan dipasang dari Vault atau *secret* Kubernetes.
- Pada *broker*, identitas *certificate* klien dipetakan ke *principal*, lalu ACL Kafka menentukan topik mana yang boleh ditulis atau dibaca.
- Spark membaca Kafka dengan opsi serupa: `kafka.security.protocol=SSL` dan lokasi *truststore*.

Sumber: Dokumentasi Apache Kafka (*Security*) dan confluent-kafka-python.

Encryption at rest dan pengelolaan kunci

Tiga tingkat *encryption at rest*

Tingkat	Melindungi dari	Tidak melindungi dari
Disk (*encryption* volume)	Disk atau server fisik dicuri	Siapa pun yang punya akses baca ke *file system*
Object storage (SSE)	Akses langsung ke media penyimpanan di pusat data penyedia	Akun dengan izin baca *bucket* (data didekripsi otomatis untuknya)
Aplikasi / kolom	Admin penyimpanan, *analyst* tanpa kunci, salinan *backup* yang bocor	Aplikasi yang memegang kunci; butuh pengelolaan kunci yang lebih rumit

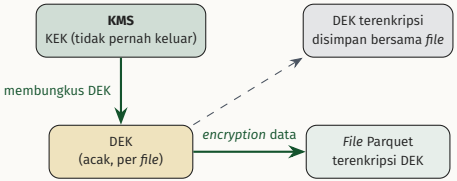
Varian SSE pada S3 dan MinIO

SSE-S3 (kunci dikelola layanan), SSE-KMS (kunci di KMS, pemakaian tercatat), SSE-C (kunci diberikan klien setiap permintaan).

Ringkasan

SSE itu wajib, tetapi **bukan** *access control*. Data paling sensitif (NIK, nomor kartu) butuh perlindungan di tingkat kolom.

Envelope encryption dan KMS



KEK = *key encryption key*; DEK = *data encryption key*.

- Data dienkripsi dengan DEK yang cepat dan lokal; hanya DEK kecil yang dikirim ke KMS untuk dibungkus atau dibuka.
- **Rotasi kunci**: KEK diganti berkala; cukup membungkus ulang DEK, tidak perlu mengenkripsi ulang petabyte data.
- Setiap pemakaian KEK tercatat di KMS: *audit trail* gratis.
- **BYOK** / HYOK: bank memegang kunci induknya sendiri, sehingga penyedia *cloud* tidak dapat membuka data tanpa izin bank.

Manajemen *secret*: berhenti menulis kata sandi di kode

Anti-pola

`MINIO_SECRET="rahasia123"` di DAG, di `docker-compose.yml` yang di-*commit*, atau di *notebook* yang dibagikan. Sekali masuk riwayat Git, menghapus barisnya tidak menghapus kebocorannya.

HashiCorp Vault (atau *secret manager cloud*)

- *Secret* disimpan terenkripsi dan diakses lewat identitas, bukan disalin
- ***Dynamic secret***: kredensial *database* dibuat saat diminta, kedaluwarsa otomatis
- Setiap pembacaan *secret* tercatat

Kebiasaan yang kita tegakkan

- `.env` masuk `.gitignore`; contoh nilai di `.env.example`
- *Secret scanning* di CI (misalnya gitleaks)
- Kredensial per layanan, bukan satu akun bersama
- Kredensial yang terlanjur bocor **dirotasi**; menghapusnya dari kode saja tidak cukup

Sumber: Dokumentasi HashiCorp Vault (KV, *database secrets engine*).

Access control

IAM, RBAC, dan ABAC

IAM

Identity and Access Management: siapa Anda (autentikasi) dan apa yang boleh Anda lakukan (otorisasi). Idealnya satu sumber identitas (SSO, misalnya Keycloak atau Entra ID) untuk semua alat data.

RBAC

Hak melekat pada **peran**: `analyst_cabang`, `data_engineer`, `auditor`. Mudah dipahami dan diaudit, tetapi jumlah peran bisa meledak.

ABAC

Hak ditentukan **atribut**: wilayah pengguna, *classification* kolom, tujuan pemrosesan. "*Analyst* hanya melihat baris cabangnya sendiri, dan kolom berlabel PII selalu ter-*mask*."

Peran	`bronze`	`silver` (PII ditokenisasi)	`gold`	Kunci detokenisasi
Data engineer	tulis	tulis	tulis	—
Analyst	—	baca	baca	—
Tim anti-*fraud*	—	baca	baca	terbatas, tercatat
Auditor	baca (rekaman)	baca	baca	—

Row-level dan *column-level security* di PostgreSQL

```
-- Row-level: analyst hanya melihat cabangnya
ALTER TABLE gold.transaksi
  ENABLE ROW LEVEL SECURITY;

CREATE POLICY cabang_sendiri
  ON gold.transaksi FOR SELECT
  TO analyst_cabang
  USING (kode_cabang =
         current_setting('app.cabang'));

-- Column-level: NIK tidak bisa dibaca
REVOKE SELECT ON gold.nasabah
  FROM analyst_cabang;
GRANT SELECT (id_nasabah, segmen, kota)
  ON gold.nasabah TO analyst_cabang;
```

- **RLS**: *policy* ditambahkan otomatis ke setiap *query*, sehingga *analyst* tidak bisa "lupa" memfilter.
- ***Column grant***: kolom sensitif tidak dapat dipilih sama sekali.
- Di *lakehouse*, padanannya diberikan oleh *layer* kebijakan: Apache Ranger (Trino, Spark, Hive), AWS Lake Formation, atau Unity Catalog.
- Kebijakan sebaiknya ditulis **sebagai kode** dan di-*review* seperti kode lain.

Sumber: Dokumentasi PostgreSQL: *Row Security Policies*, GRANT.

Melindungi data pribadi

Menemukan PII dulu, baru melindunginya

Tidak bisa melindungi yang tidak diketahui. Langkah pertama adalah ***classification***: kolom mana berisi data pribadi.

- **Terstruktur**: kolom `nik`, `nama`, `no_hp`, `alamat`, `tgl_lahir`.
- **Tersembunyi**: kolom bebas seperti `keterangan_transfer` (“bayar kontrakan Bu Siti 0812…”), log aplikasi, *file* KTP hasil unggah.
- **Kuasi-identifier**: kombinasi kode pos, tanggal lahir, jenis kelamin dapat mengenali seseorang walau tanpa nama.

Microsoft Presidio

Pustaka terbuka untuk mendeteksi dan menganonimkan PII dalam teks: gabungan *regex*, *checksum*, dan model NER. *Recognizer* khusus dapat ditambahkan, misalnya pola NIK 16 digit atau nomor rekening bank.

Ingat

Deteksi otomatis tidak sempurna: ada yang lolos (*false negative*) dan yang salah tandai (*false positive*). Ukur dengan contoh berlabel sebelum mempercayainya.

Masking, *tokenization*, pseudonimisasi, anonimisasi

Teknik	Contoh untuk NIK 3201…0001	Dapat dikembalikan?	Dipakai untuk
***Masking* statis**	`3201********0001` disimpan di salinan	Tidak	*Test data*, salinan untuk *vendor*
***Masking* dinamis**	Asli di tabel, ter-*mask* saat dibaca peran tertentu	Ya, bagi peran berhak	*Serving* ke banyak peran
Tokenization	`tok_8f3a...` ; pemetaan di *vault* terpisah	Ya, lewat layanan *token*	*Join* antar tabel tanpa membuka NIK
***Hashing* berkunci** (HMAC)	`hmac(kunci, nik)`	Tidak langsung; tetap konsisten	Kunci *join*, deduplikasi
Generalization	Umur 34 → “30–39”	Tidak	Statistik, publikasi

Pseudonimisasi

Identitas diganti pengganti, tetapi masih dapat dikaitkan kembali dengan informasi tambahan (kunci atau tabel *token*). **Tetap data pribadi**.

Anonimisasi

Tidak dapat dikaitkan kembali dengan upaya yang wajar. Jauh lebih sulit dicapai daripada kelihatannya.

Mengapa “hapus kolom nama” bukan anonimisasi

Kode pos	Umur	JK	Pembiayaan
16810	30–39	P	Murabahah
16810	30–39	P	Ijarah
16810	30–39	P	Murabahah
16911	40–49	L	Musyarakah
16911	40–49	L	Murabahah

Data ilustrasi. Kelompok pertama berukuran 3, kelompok kedua berukuran 2: tabel ini **2-anonymous**.

- **k-anonymity**: setiap kombinasi kuasi-identifier muncul minimal k kali, sehingga seseorang “bersembunyi” di antara k orang.
- Kelemahannya: bila semua anggota kelompok punya nilai sensitif sama, nilainya tetap bocor (motivasi *l-diversity*).
- Penggabungan dengan data lain (*linkage attack*) sering mengalahkan penghapusan kolom nama.
- ***Differential privacy***: menambahkan *noise* terukur pada hasil agregat, sehingga ada atau tidaknya satu orang hampir tidak mengubah jawaban. Parameter ε (dibaca “epsilon”; *privacy budget*; makin kecil makin privat) mengatur *trade-off* privasi dan *accuracy*.

Sumber: Sweeney (2002) tentang *k-anonymity*; Dwork dan Roth (2014) tentang *differential privacy*; Jarmul, *Practical Data Privacy* (2023).

Di *layer* mana PII dilindungi?



- *Tokenization* **sedini mungkin**: makin sedikit salinan PII asli, makin kecil kerusakan bila bocor. *Bronze* tetap menyimpan asli untuk pemrosesan ulang, tetapi dikunci paling ketat dan masa simpannya dibatasi.
- *Analyst* dan *data scientist* bekerja di *silver* dan *gold*; model *fraud* cukup memakai *token* yang konsisten sebagai kunci.
- Detokenisasi hanya untuk kebutuhan sah (misalnya tim anti-*fraud* menghubungi nasabah) dan setiap permintaan tercatat.

Audit log dan *crypto-shredding*

Audit log: bukti bahwa kontrol bekerja

Yang dicatat

- Siapa (identitas, bukan akun bersama), kapan, dari mana
- Objek apa: tabel, kolom, *bucket*, *key*
- Tindakan: baca, tulis, hapus, ubah izin, detokenisasi
- Hasil: diizinkan atau **ditolak**

Sumber log

Audit log MinIO/S3, `pgaudit` di PostgreSQL, log akses Trino atau Ranger, ACL Kafka, log KMS dan Vault.

Supaya log berguna

- Dikirim ke tempat terpisah yang tidak bisa diubah pelaku (*append-only*, *object lock*)
- Disimpan sesuai kebijakan retensi audit
- Dipantau: lonjakan pembacaan tabel nasabah tengah malam, banyak penolakan akses beruntun, ekspor besar ke luar jaringan

Hati-hati

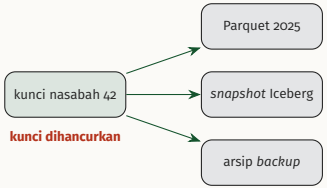
Log juga bisa berisi PII (misalnya isi *query*). Log perlu dilindungi seperti data.

Crypto-shredding: menghapus di *tier* dingin dan *backup*

Nasabah berhak meminta datanya dihapus, tetapi datanya tersebar di *file* Parquet lama, *snapshot* Iceberg, arsip Glacier, *backup*, dan *immutable storage* (WORM, *ledger*; P7) yang memang tidak bisa diubah.

Gagasan

Encryption data setiap nasabah (atau kolom PII-nya) dengan **kunci per subjek**. Untuk "menghapus", hancurkan kuncinya. Salinan di mana pun tinggal *ciphertext* yang tidak bisa dibuka lagi.



- Perlu KMS yang mampu mengelola banyak kunci
- Benturan dengan kewajiban retensi: data yang **wajib** disimpan tidak boleh ikut dihancurkan, dibahas di P15
- Di *ledger* atau *blockchain*: simpan hanya *hash*, data aslinya *off-chain* terenkripsi (P7)

Security di *pipeline* Spark tugas besar UAS

```
spark = (SparkSession.builder
  .appName("uas-silver")
  # autentikasi antar komponen Spark
  .config("spark.authenticate", "true")
  # enkripsi RPC driver <-> executor
  .config("spark.network.crypto.enabled", "true")
  # enkripsi shuffle/spill di disk lokal
  .config("spark.io.encryption.enabled", "true")
  # MinIO lewat HTTPS, kredensial dari env
  .config("spark.hadoop.fs.s3a.connection.ssl.enabled",
          "true")
  .getOrCreate())
```

- ***Tokenization* di langkah *bronze* → *silver***: jalankan fungsi `token()` sebagai Pandas UDF (berbasis Arrow, P9) sehingga PII asli tidak pernah sampai ke *silver* dan *gold*.
- ***Secret***: kunci HMAC dan kredensial MinIO dari Vault atau *environment*, tidak di kode atau *notebook*.
- ***Least privilege***: akun *job* Spark hanya boleh membaca `bronze` dan menulis `silver`; *dashboard* hanya membaca `gold`.
- **Spark UI**: jangan dibuka ke publik; pasang autentikasi atau akses lewat *tunnel*.
- **Kafka → Spark**: TLS/mTLS seperti frame sebelumnya (P11).

Sumber: Dokumentasi Apache Spark, *Security*; nama opsi dapat berbeda antarversi.

When to use dan *best practice*

When to use: alat keamanan data

Alat	Pakai bila	Pertimbangkan alternatif bila
HashiCorp Vault	*Secret*, *dynamic credential*, PKI, layanan *encryption* di lingkungan campuran atau *on-premise*	Seluruh platform di satu *cloud* (*secret manager* bawaan lebih ringan)
KMS *cloud*	Kunci *encryption* untuk S3, *warehouse*, *envelope encryption*	Kebijakan mengharuskan kunci di HSM milik sendiri
Apache Ranger	Kebijakan akses terpusat untuk Trino, Hive, Spark, Kafka di *on-premise*	Platform terkelola dengan katalog bawaan (Lake Formation, Unity Catalog)
PostgreSQL RLS	*Serving* data teragregasi ke banyak peran	Ribuan kebijakan dinamis lintas mesin *query*
Presidio	Mendeteksi PII di teks bebas dan kolom, bisa ditambah pola lokal	Data sangat besar tanpa kemampuan *batch* (jalankan di Spark atau sampel)

Best practice keamanan data di industri

1. **Kumpulkan seperlunya.** Kolom yang tidak dibutuhkan tidak diambil sejak *ingest*.
2. ***Tokenization* di batas pertama.** PII asli tidak menyebar ke *silver* dan *gold*.
3. **Privat secara bawaan.** *Bucket*, topik, dan tabel baru tertutup sampai diberi izin.
4. **Identitas per orang dan per layanan.** Tidak ada akun bersama.
5. **Kredensial berumur pendek.** *Dynamic secret*, rotasi otomatis.
6. **Semua jalur terenkripsi**, termasuk di dalam *cluster*.
7. **Kebijakan sebagai kode.** IAM, *bucket policy*, dan RLS di Git dan di-*review*.
8. ***Test data* sintetis.** Lingkungan dev tidak memakai salinan produksi.
9. **Log yang tidak bisa diubah**, dan benar-benar dipantau.
10. **Uji kontrolnya.** Buktikan akses yang seharusnya ditolak memang ditolak.

Ringkasan

Kontrol yang belum pernah diuji belum terbukti bekerja. Bagian penilaian praktikum hari ini adalah **bukti penolakan**: akses yang seharusnya ditolak benar-benar ditolak.

Praktikum

Praktikum 1: deteksi PII dan *tokenization* di *layer silver*

```
import hmac, hashlib, os
from presidio_analyzer import (AnalyzerEngine,
    PatternRecognizer, Pattern)
nik = PatternRecognizer(supported_entity="NIK",
    patterns=[Pattern("nik", r"\b\d{16}\b", 0.6)])
an = AnalyzerEngine()
an.registry.add_recognizer(nik)
KUNCI = os.environ["TOKEN_KEY"].encode()  # Vault

def token(v: str) -> str:
    return "tok_" + hmac.new(KUNCI, v.encode(),
        hashlib.sha256).hexdigest()[:20]

def bersihkan(t: str) -> str:
    hs = an.analyze(t, language="en",
                    entities=["NIK", "PHONE_NUMBER"])
    for r in sorted(hs, key=lambda r: -r.start):
        t = t[:r.start] + f"<{r.entity_type}>" + t[r.end:]
    return t
```

- Kolom terstruktur (`nik`, `no_hp`) diganti `token()`: konsisten, sehingga *join* dan deduplikasi tetap berjalan.
- Kolom bebas (`keterangan_transfer`) dibersihkan dengan `bersihkan()`.
- Kunci HMAC diambil dari Vault, tidak ditulis di kode.
- Jalankan sebagai UDF di Spark atau langkah Python di DAG *bronze* → *silver*.
- Ukur: dari 200 baris berlabel, berapa PII yang lolos?

Sumber: Model bahasa Presidio bawaan berbahasa Inggris; pola lokal ditambahkan sebagai *recognizer*.

Praktikum 2: *bucket policy* per peran dan bukti penolakan

```
# kebijakan baca silver dan gold saja
cat > analyst.json <<'EOF'
{"Version": "2012-10-17",
 "Statement": [{"Effect": "Allow",
   "Action": ["s3:GetObject","s3:ListBucket"],
   "Resource": ["arn:aws:s3:::silver",
                "arn:aws:s3:::silver/*",
                "arn:aws:s3:::gold",
                "arn:aws:s3:::gold/*"]}]}
EOF
mc admin policy create lab analyst analyst.json
mc admin user add lab ani PasswordLab-1
mc admin policy attach lab analyst --user ani

# uji: harus DITOLAK
mc cp lab-ani/bronze/nasabah.parquet .
```

1. Buat peran `analyst` (baca *silver*, *gold*) dan `engineer` (tulis semua *layer*).
2. Coba akses *bronze* dengan akun `ani`; tangkap *error message* `Access Denied`.
3. Aktifkan audit log MinIO dan temukan entri penolakan itu.
4. Aktifkan TLS di Kafka praktikum, lalu buktikan klien tanpa *certificate* gagal terhubung.
5. Laporkan: tabel kontrol, perintah uji, dan tangkapan bukti.

Sumber: Dokumentasi MinIO Client (`mc admin policy`); nama perintah dapat berbeda antar versi `mc`.

Tugas untuk pertemuan berikutnya

1. Kumpulkan laporan praktikum: tabel kontrol per *layer* (*bronze*, *silver*, *gold*, Kafka), kode *tokenization*, hasil pengukuran deteksi PII, dan bukti penolakan dari audit log.
2. Terapkan minimal tiga kontrol *security* dari hari ini pada *pipeline* Spark tugas besar UAS, termasuk *tokenization* PII di layer *silver*.
3. Baca ringkasan UU No. 27 Tahun 2022 tentang Pelindungan Data Pribadi, terutama bagian hak subjek data, kewajiban pengendali, dan kegagalan pelindungan data pribadi.
4. Baca DAMA-DMBOK Bab 1 (*Data Management*) dan Bab 3 (*Data Governance*).

Pertemuan depan

Data governance, GRC, dan *observability*: *catalog* dan *lineage*, *data contract*, *risk register*, kewajiban UU PDP dan ISO/IEC 27001, lalu memantau *pipeline* tugas besar, sekaligus presentasi progres UAS.

Terima kasih

Pertanyaan?