Contents

[10.9] System Coupling Failures

1. Operational Definition: A state where tightly coupled systems (where a failure in one immediately and inevitably affects the other) lack the resilience to handle security-related disruptions, leading to cascading failures.

2. Main Metric & Algorithm:

- **Metric:** Coupling Failure Impact (CFI). Formula: `CFI = (Downtime_of_Dependent_System / Downtime_of_Primary_System) * Data_Criticality_Weight`.
- **Pseudocode:**

  python

```
def calculate_cfi(primary_system_incident):
    # Get primary system downtime
    primary_downtime = primary_system_incident.downtime_minutes

    # Get most affected coupled system and its downtime from CMDB relations
    coupled_systems = get_directly_dependent_systems(primary_system_incident.affected_syst
    max_cfi = 0

    for system in coupled_systems:
        # Find incidents for this coupled system during the primary incident
        coupled_incident = find_related_incident(system, primary_system_incident.time)
        if coupled_incident:
            coupled_downtime = coupled_incident.downtime_minutes
            # Get criticality weight from CMDB (e.g., 1 for dev, 5 for critical prod)
            criticality = get_system_criticality(system)

            # Calculate CFI for this coupled system
            if primary_downtime > 0:
                cfi = (coupled_downtime / primary_downtime) * criticality
            else:
                cfi = coupled_downtime * criticality
            if cfi > max_cfi:
                max_cfi = cfi
    return max_cfi
```

- **Alert Threshold:** `CFI > 1.0` (The failure of a system causes a *more* severe outage in a dependent system, especially if it's critical).

3. Digital Data Sources (Algorithm Input):

- **CMDB:** For `depends_on` relationships and `system_criticality` ratings.
- **Incident Management Platform:** For `system_downtime` per incident.

4. Human-to-Human Audit Protocol: During architecture reviews, specifically challenge teams on system coupling. “If this database goes down for 10 minutes, what breaks? Does it break immediately, or is there a buffer? Show me the failure mode.” This exposes tight coupling.

5. Recommended Mitigation Actions:

- **Technical/Digital Mitigation:** Introduce loose coupling where possible: use queues (e.g., Kafka, SQS) to buffer between systems, implement retries with exponential backoff, and design for graceful degradation.
- **Human/Organizational Mitigation:** Train architects and engineers on the security and resilience implications of tight vs. loose coupling.
- **Process Mitigation:** Mandate that business continuity and disaster recovery tests specifically include scenarios where a tightly coupled system is compromised to test the resilience of its dependencies.