Diseño: Sistema de Consenso P2P y Reputación de Nodos

Versión: 1.0

Fecha: 25 de enero de 2026

1. Visión General

Objetivo

Implementar un sistema de **Consenso Distribuido** que detecte si un ataque de corrupción de checkpoints es:

- **LOCAL** (malware en el disco duro del usuario)
- **DE RED** (seed comprometido o CDN atacado)

Principio

> "No confíes en un solo reporte. Solo actúa cuando múltiples nodos independientes confirmen el problema."

2. Componentes Principales

2.1 Sistema de Reputación de Nodos

```
Cada nodo mantiene un registro de reputación para otros nodos:

Nodo A (reputation = 0.95) ← Muy confiable, pocos reportes falsos
Nodo B (reputation = 0.70) ← Moderado, algunos reportes falsos
Nodo C (reputation = 0.20) ← Sospechoso, muchos reportes falsos
Nodo D (reputation = -0.50) ← Malicioso/Bannado
```

Estructura de Datos

```cpp
struct NodeReputation {
  std::string node_id;              // Identificador único del nodo
  float score = 0.5f;               // 0.0 (malicioso) a 1.0 (confiable)
  uint64_t total_reports = 0;       // Reportes totales de este nodo
  uint64_t confirmed_reports = 0;   // Reportes confirmados por consenso
  uint64_t false_reports = 0;       // Reportes rechazados
  std::time_t last_updated = 0;     // Última actualización
```

```
  float calculate_reputation() {
    if (total_reports == 0) return 0.5f;
    return (confirmed_reports / (float)total_reports) * 0.9f + 0.1f;
  }
};
```

Algoritmo de Cálculo

```
Reputación = (Reportes_Confirmados / Total_Reportes) × 0.9 + 0.1

Ejemplo:
- Nodo A: 95 confirmados de 100 → 0.95 × 0.9 + 0.1 = 0.955 (EXCELENTE)
- Nodo B: 7 confirmados de 10 → 0.70 × 0.9 + 0.1 = 0.730 (BUENO)
- Nodo C: 2 confirmados de 10 → 0.20 × 0.9 + 0.1 = 0.280 (SOSPECHOSO)

Umbral mínimo para confiar: 0.40
```

2.2 Query de Seguridad (P2P)

Cuando un nodo detecta un problema, **pregunta a otros peers**:

```
struct SecurityQuery {
  std::string query_id;             // ID único del query
  uint64_t height;                  // Altura del checkpoint
  std::string expected_hash;        // Hash que esperamos
  std::string reported_hash;        // Hash que recibimos (incorrecto)
  std::string source;               // De dónde vino el hash falso
  std::string attack_type;          // "invalid_format", "replay",
"seed_mismatch"
  std::time_t timestamp;            // Cuándo lo detectamos
};

struct SecurityResponse {
  std::string query_id;             // ID del query original
  std::string responder_node_id;    // Quién responde
  bool also_detected = false;       // "¿Tú también lo ves?"
  std::string responder_hash;       // Qué hash tiene el respondedor
  std::time_t response_time;         // Cuándo respondió
  float responder_reputation;       // Su reputación en nuestros registros
};
```

Flujo de Consenso

```
1. Nodo A detecta hash inválido en height 30

2. Nodo A envía SecurityQuery a peers:
   "¿Ustedes también tienen problemas con height 30?"

3. Peers responden:
   - Nodo B: "Sí, tengo el mismo problema"
   - Nodo C: "Sí, detecté lo mismo"
   - Nodo D: "No, mis datos están bien"

4. Nodo A cuenta respuestas:
   - Confirmaciones: 2 (B, C)
   - Negaciones: 1 (D)
   - Consenso: 2/3 = 66.6% → ATAQUE CONFIRMADO

5. Nodo A actúa:
   ☑ SÍ reportar y aplicar protecciones
```

2.3 Consenso Mínimo

```
Regla de decisión:

confirmed_nodes = (respuestas afirmativas)
total_responses = (respuestas recibidas)

if (confirmed_nodes >= 2 AND confirmed_nodes / total_responses >= 0.66) {
  ATTACK_CONFIRMED = true
  BROADCAST_ALERT = true
  UPDATE_REPUTATION = true
} else {
  ATTACK_LOCAL = true
  LOG_LOCALLY = true
  NO_BROADCAST = true
}
```

Umbrales por Escenario

```
Escenario 1: Preguntamos a 3 nodos
  - 0 dicen "sí" → LOCAL (ignora)
  - 1 dice "sí"  → LOCAL (6 nodos totales) o INVESTIGAR (3 nodos totales)
  - 2 dicen "sí" → CONFIRMADO ☑
  - 3 dicen "sí" → FUERTEMENTE CONFIRMADO ☑☑

Escenario 2: Preguntamos a 5 nodos
  - 2 dicen "sí" → MARGINAL (investigar)
```

```
- 3 dicen "sí" → CONFIRMADO ☑
- 4-5 dicen "sí" → FUERTEMENTE CONFIRMADO ☑☑
```

2.4 Manejo de Reportes Recibidos

Cuando **otro nodo nos reporta** un ataque:

```
void on_receive_security_alert(const SecurityAlert& alert) {
  // 1. Verificar firma digital
  if (!verify_signature(alert)) {
    LOG_ERROR("Invalid signature - IGNORING");
    return;  // Rechazar alerta falsa
  }

  // 2. Verificar reputación del nodo que reporta
  float reputation = get_node_reputation(alert.reporting_node_id);
  if (reputation < 0.40) {
    LOG_WARN("Low-reputation node reporting - IGNORING");
    update_reputation(alert.reporting_node_id, false);  // Penalizar
    return;
  }

  // 3. Validar si NOSOTROS también vemos el problema
  bool we_see_it = verify_alert_locally(alert);
  if (!we_see_it) {
    LOG_WARN("Alert not confirmed locally - IGNORING");
    update_reputation(alert.reporting_node_id, false);  // Penalizar
    return;
  }

  // 4. Solo entonces actuar
  LOG_ERROR("Alert CONFIRMED - applying protections");
  add_source_to_blacklist(alert.source);
  update_reputation(alert.reporting_node_id, true);  // Recompensar
}
```

2.5 Detección de Nodo Bajo Ataque y Quarantine Temporal

Problema Identificado

Un nodo legítimo puede ser atacado selectivamente (malware en su PC, BGP hijacking, etc.) haciéndolo recibir checkpoints corruptos. Si el nodo reporta esto, podría perder reputación injustamente por estar haciendo reportes "falsos" (que otros no confirman porque NO están siendo atacados).

Solución: Quarantine Temporal

Indicadores de Ataque Selectivo

```
Un nodo está siendo atacado si:
├─ Hace 5+ reportes en <1 hora
├─ Todos reportan la MISMA FUENTE (mismo seed)
├─ Reportan el MISMO HASH INCORRECTO
├─ PERO otros nodos pueden conectar a esa fuente sin problemas
├─ Y obtienen datos VÁLIDOS de la misma fuente
└─ Patrón consistente y repetitivo
```

Estructura de Datos

```
struct QuarantinedNode {
  std::string node_id;
  std::time_t quarantine_start;
  std::time_t quarantine_end;
  std::string attack_source;           // Fuente que lo está atacando
  uint64_t attack_count;               // Reportes durante el ataque
  float reputation_before_quarantine;  // Guardar reputación anterior
  bool under_attack = true;
};

// En checkpoints.h
std::map<std::string, QuarantinedNode> m_quarantined_nodes;
std::time_t m_quarantine_duration = 3600;  // 1 hora por defecto (configurable
1-6h)
```

Lógica de Detección

```
bool is_node_under_attack(const std::string& node_id) {
  // Contar reportes en última hora
  uint64_t recent_reports = count_reports_last_hour(node_id);

  if (recent_reports < 5) return false;

  // Todos vienen de la misma fuente?
  std::string attack_source = get_most_reported_source(node_id);
  float same_source_ratio = calculate_same_source_ratio(node_id,
attack_source);

  if (same_source_ratio < 0.80) return false;  // 80% mínimo

  // Otros nodos pueden alcanzar esa fuente SIN problemas?
  bool others_can_reach = can_other_peers_reach(attack_source);

  if (!others_can_reach) return false;

  // Todos los reportes tienen el mismo hash incorrecto?
```

```
  std::string bad_hash = get_most_reported_hash(node_id, attack_source);
  float same_hash_ratio = calculate_same_hash_ratio(node_id, bad_hash);

  if (same_hash_ratio < 0.80) return false;  // 80% mínimo

  // CONCLUSIÓN: Nodo A está siendo atacado selectivamente
  return true;
}
```

Activación de Quarantine

```
void activate_quarantine(const std::string& node_id,
                         const std::string& attack_source) {
  QuarantinedNode qnode;
  qnode.node_id = node_id;
  qnode.quarantine_start = get_current_time();
  qnode.quarantine_end = qnode.quarantine_start + m_quarantine_duration;
  qnode.attack_source = attack_source;
  qnode.attack_count = count_reports_last_hour(node_id);
  qnode.reputation_before_quarantine = get_peer_reputation(node_id);

  m_quarantined_nodes[node_id] = qnode;

  // LOG local
  MERROR("[QUARANTINE] Nodo " << node_id << " puesto en cuarentena");
  MERROR("[QUARANTINE] Fuente de ataque: " << attack_source);
  MERROR("[QUARANTINE] Reportes detectados: " << qnode.attack_count);
  MERROR("[QUARANTINE] Duración: " << (m_quarantine_duration/3600) << "
horas");

  // Notificar al nodo si es local
  if (is_local_node(node_id)) {
    display_quarantine_warning(node_id, attack_source, qnode.quarantine_end);
  }
}
```

Aviso en Terminal (Nodo Afectado)

```
void display_quarantine_warning(const std::string& node_id,
                                const std::string& attack_source,
                                std::time_t release_time) {
  std::string time_remaining = format_seconds(release_time -
get_current_time());

  // Mostrar en terminal / LOG con formato destacado
  MERROR("\n");
  MERROR("╔═══════════════════════════════════════════════════════════╗");
```

```
  MERROR("╠            ⚠  ATTACK DETECTED - QUARANTINE ACTIVE  ⚠
╣");
  MERROR("╠═══════════════════════════════════════════════════════════════╣");
  MERROR("╠                                                               ╣");
  MERROR("╠  Your node is under SELECTIVE ATTACK                          ╣");
  MERROR("╠                                                               ╣");
  MERROR("╠  Attack source:     " << std::setw(45) << std::left <<
attack_source << "╣");
  MERROR("╠  Type:              Corrupted checkpoints (LOCAL)             ╣");
  MERROR("╠  Status:            TEMPORARY QUARANTINE MODE                 ╣");
  MERROR("╠  Duration:          " << std::setw(45) << std::left <<
time_remaining << "╣");
  MERROR("╠                                                               ╣");
  MERROR("╠  Automatic actions:                                           ╣");
  MERROR("╠  • Your reports will NOT affect your reputation              ╣");
  MERROR("╠  • Node will continue attempting to recover valid data       ╣");
  MERROR("╠  • When attack is confirmed, +reputation awarded             ╣");
  MERROR("╠                                                               ╣");
  MERROR("╠  Recommendations:                                             ╣");
  MERROR("╠  • Check your internet connection                            ╣");
  MERROR("╠  • Verify no malware is present on your system               ╣");
  MERROR("╠  • Consider using a VPN if on public network                 ╣");
  MERROR("╠                                                               ╣");
  MERROR("╚═══════════════════════════════════════════════════════════════╝");
  MERROR("\n");
}
```

Durante la Quarantine

```
void update_peer_reputation(const std::string& node_id, bool confirmed) {
  // Verificar si el nodo está en quarantine
  if (is_quarantined(node_id)) {
    // NO actualizar reputación (ni penalizar ni recompensar)
    MINFO("[QUARANTINE] " << node_id
          << " está en cuarentena - ignorando cambio de reputación");

    // Pero SÍ seguir contando reportes para análisis posterior
    m_quarantined_nodes[node_id].attack_count++;
    return;
  }

  // Lógica normal de reputación
  NodeReputation& rep = m_peer_reputation[node_id];
  if (confirmed) {
    rep.confirmed_reports++;
    rep.score += 0.1f;
  } else {
    rep.false_reports++;
    rep.score -= 0.05f;
```

```
  }
}
```

Salida de Quarantine

```
void check_quarantine_status() {
  std::time_t now = get_current_time();

  for (auto it = m_quarantined_nodes.begin(); it != m_quarantined_nodes.end();
) {
    if (now >= it->second.quarantine_end) {
      // Quarantine terminada
      std::string node_id = it->first;
      QuarantinedNode& qnode = it->second;

      MERROR("[QUARANTINE] Fin de cuarentena para " << node_id);
      MERROR("[QUARANTINE] Reputación restaurada a: "
             << qnode.reputation_before_quarantine);

      // Restaurar reputación a la que tenía antes
      m_peer_reputation[node_id].score = qnode.reputation_before_quarantine;

      // Si se confirmó el ataque en la red → premiar
      if (was_network_attack_confirmed(qnode.attack_source)) {
        m_peer_reputation[node_id].score += 0.2f;  // Bonificación
        MERROR("[QUARANTINE] Ataque confirmado por red - +0.2 reputación");
      }

      it = m_quarantined_nodes.erase(it);
    } else {
      ++it;
    }
  }
}
```

Configuración (ninacatcoin.conf)

```
# Quarantine temporal para nodos bajo ataque
quarantine-enabled = true
quarantine-min-duration = 3600           # 1 hora
quarantine-max-duration = 21600          # 6 horas
quarantine-attack-threshold = 5          # 5+ reportes
quarantine-time-window = 3600            # en <1 hora
quarantine-source-ratio = 0.80           # 80% misma fuente
quarantine-hash-ratio = 0.80             # 80% mismo hash
```

---

3. PAUSE MODE (SIN CAMBIOS)

El PAUSE MODE actual **se mantiene exactamente igual**:

```
- Detecta hash inválido
- Entra en PAUSE MODE indefinido
- Intenta seeds cada 30 segundos
- Se mantiene en pausa hasta obtener datos válidos

+ NUEVO: Mientras está en PAUSE MODE, puede:
  - Consultar a otros nodos sobre la reputación de los seeds
  - Agregar información a su blacklist basado en consenso
  - (Pero sigue intentando recuperarse con seeds válidos)
```

---

4. Arquitectura de Componentes

4.1 En checkpoints.h (Nuevas declaraciones)

```cpp
class checkpoints {
private:
  // Sistema de reputación
  std::map<std::string, NodeReputation> m_peer_reputation;
  float m_reputation_update_factor = 0.1f;
  float m_reputation_threshold = 0.40f;

  // Queries en progreso
  std::map<std::string, SecurityQuery> m_pending_queries;
  std::map<std::string, std::vector<SecurityResponse>> m_query_responses;

  // Blacklists dinámicos
  std::set<std::string> m_dynamic_blacklist;

public:
  // Nuevos métodos
  bool query_peers_for_consensus(const SecurityQuery& query);
  void update_peer_reputation(const std::string& node_id, bool confirmed);
  void process_security_response(const SecurityResponse& response);
  float get_peer_reputation(const std::string& node_id);
};
```

4.2 Nueva Herramienta: security_query_tool.hpp

Ubicación: `tools/security_query_tool.hpp`

Funcionalidades:

- Serializar/deserializar queries

- Validar firmas digitales
- Calcular consenso
- Gestionar timeouts

4.3 Nueva Herramienta: reputation_manager.hpp

Ubicación: `tools/reputation_manager.hpp`

Funcionalidades:

- Persistencia de reputación (guardada en disco)
- Cálculo de scores
- Decay temporal (olvido de reportes antiguos)
- Estadísticas

---

5. Flujo Completo de Ejemplo

Caso: Ataque LOCAL (Malware en PC)

```
TIEMPO 1: Nodo A detecta hash inválido
├─ LOG: "[HASH VALIDATION] Invalid hash format at height 30"
├─ ACCIÓN: Generar SecurityQuery
└─ Query enviado a 3 peers: B, C, D

TIEMPO 2: Peers responden
├─ Nodo B: "No tengo ese problema"
├─ Nodo C: "No tengo ese problema"
└─ Nodo D: "No tengo ese problema"

TIEMPO 3: Análisis de consenso
├─ Confirmaciones: 0/3 = 0%
├─ CONCLUSIÓN: ATAQUE LOCAL
└─ ACCIÓN:
   ✓ Guardar log localmente
   ✓ NO reportar a red
   ✓ NO penalizar a otros nodos

RESULTADO: El malware está aislado en Nodo A solamente
           Resto de la red sigue operando normalmente
```

Caso: Ataque de RED (Seed comprometido)

```
TIEMPO 1: Nodo A detecta hash inválido
├─ Fuente: seed1.ninacatcoin.es
├─ Hash incorrecto: AABBCCDD...
└─ Query enviado a peers
```

```
TIEMPO 2: Peers responden
├─ Nodo B: "¡YO TAMBIÉN! Mismo problema con seed1"
├─ Nodo C: "¡YO TAMBIÉN! Mismo hash incorrecto"
└─ Nodo D: "No, seed1 me responde bien"

TIEMPO 3: Análisis de consenso
├─ Confirmaciones: 2/3 = 66.6%
├─ CONCLUSIÓN: ATAQUE CONFIRMADO
└─ ACCIÓN:
   ✓ Generar SecurityAlert
   ✓ Broadcast a toda la red
   ✓ Agregar seed1 a blacklist dinámico
   ✓ Actualizar reputación (B, C +0.1 || D -0.05)

TIEMPO 4: Otros nodos reciben alerta
├─ Nodo E: Verifica firma de Nodo A
├─ Nodo E: Comprueba reputación de A (0.95) ✓ ALTA
├─ Nodo E: Valida que seed1 esté comprometido
└─ Nodo E: Agregadoseed1 a su blacklist

RESULTADO: Toda la red ahora evita seed1
           Atacante fracasa
```

---

6. Persistencia de Datos

6.1 Guardar Reputación en Disco

```
Archivo: ~/.ninacatcoin/testnet/peer_reputation.json

{
  "nodes": [
    {
      "node_id": "node_abc123",
      "score": 0.95,
      "confirmed_reports": 95,
      "false_reports": 5,
      "last_updated": 1769371978
    },
    {
      "node_id": "node_def456",
      "score": 0.20,
      "confirmed_reports": 2,
      "false_reports": 8,
      "last_updated": 1769371900
    }
  ],
  "updated_at": 1769371978,
  "version": "1.0"
}
```

6.2 Cargar al Iniciar

```
void reputation_manager::load_from_disk() {
  // Al iniciar el daemon, cargar reputación guardada
  // Aplicar decay temporal:
  //   - Reportes de hace >7 días: menos peso
  //   - Reportes de hace >30 días: se olvidan
}
```

---

7. Protecciones Contra Falsas Alarmas

7.1 Nodo Mentiroso

```
Nodo Malicioso intenta reportar falsamente:
├─ Envía: "seed1 está comprometido!"
├─ Realidad: seed1 está limpio
└─ Otros nodos responden: "No, seed1 responde bien"

Resultado:
├─ Consenso RECHAZA el reporte (0/3 = 0%)
├─ Nodo Malicioso pierde reputación
├─ Siguiente vez será ignorado (reputation < 0.40)
└─ Eventualmente bannado de la red
```

7.2 Nodo Legítimo Bajo Ataque (NUEVO)

```
Nodo Bueno siendo atacado selectivamente:
├─ Hace 5+ reportes de seed1 en <1 hora
├─ Otros nodos pueden conectar a seed1 sin problemas
├─ Sistema detecta ataque selectivo
└─ ACCIÓN:
   ✓ Nodo puesto en QUARANTINE (1-6h)
   ✓ Reputación NOT afectada
   ✓ Aviso en terminal: "⚠ BAJO ATAQUE - QUARANTINE"
   ✓ Cuando salga de quarantine → RECOMPENSADO si se confirma

Resultado:
├─ Nodo bueno protegido de penalización injusta
├─ La red aprende que seed1 ES atacante
├─ Nodo gana +0.2 reputación
└─ Atacante identificado y eliminado
```

7.3 Ataque Sybil (Múltiples identidades falsas)

```
Atacante crea 10 nodos falsos para reportar juntos:
├─ Todos tienen reputation = 0.5 (nuevos)
├─ Todos dicen: "seed1 está malo"
└─ PERO nodos legítimos dicen: "seed1 está bien"

Resultado:
├─ Los reportes del atacante son ignorados
├─ Los 10 nodos falsos pierden reputación rápidamente
├─ Después de 5 reportes falsos c/u → reputation < 0.40
└─ Bannados
```

---

8. Timeline de Implementación

Fase 1: Infraestructura Base (Week 1)

- [ ] Crear `security_query_tool.hpp`
- [ ] Crear `reputation_manager.hpp`
- [ ] Integrar en `checkpoints.h`
- [ ] Estructura de Quarantine

Fase 2: Lógica de Consenso (Week 2)

- [ ] Implementar `query_peers_for_consensus()`
- [ ] Implementar procesamiento de respuestas
- [ ] Detección de ataque selectivo
- [ ] Pruebas unitarias

Fase 3: Quarantine y Notificaciones (Week 3)

- [ ] Activación/desactivación de quarantine
- [ ] Aviso en terminal con formato
- [ ] Persistencia (guardar estado quarantine)
- [ ] Protección de reputación durante quarantine
- [ ] Tests de quarantine

Fase 4: Persistencia e Integración (Week 4)

- [ ] Guardar/cargar reputación
- [ ] Decay temporal
- [ ] Estadísticas
- [ ] Usar consenso mientras está en PAUSE MODE
- [ ] Pruebas de fin a fin
- [ ] Documentación

---

9. Configuración (ninacatcoin.conf)

```
# Consenso de seguridad
consensus-peer-count = 3          # Cuántos peers consultar
consensus-threshold = 0.66        # Porcentaje para confirmar (66%)
reputation-threshold = 0.40       # Reputación mínima para confiar
reputation-decay-days = 30        # Olvido de reportes antiguos
reputation-update-factor = 0.1    # Cuánto cambiar por reporte

# Quarantine temporal para nodos bajo ataque
quarantine-enabled = true
quarantine-min-duration = 3600           # 1 hora mínimo
quarantine-max-duration = 21600          # 6 horas máximo
quarantine-attack-threshold = 5          # 5+ reportes activa quarantine
quarantine-time-window = 3600            # Detectar ataque en <1 hora
quarantine-source-ratio = 0.80           # 80% misma fuente
quarantine-hash-ratio = 0.80             # 80% mismo hash incorrecto
```

---

10. Métricas y Monitoreo

10.1 Estadísticas a Registrar

```
- Total queries enviados
- Consensos alcanzados
- Falsos positivos detectados
- Nodos bannados
- Reputación promedio de la red
- Tiempo promedio de respuesta
```

10.2 Dashboard Local

```
~/.ninacatcoin/testnet/security/
├─ consensus_stats.json
├─ peer_reputation.json
├─ alert_history.json
└─ blacklist_dynamic.json
```

---

11. Seguridad Criptográfica

11.1 Firmas Digitales

```
// Cada alerta debe estar firmada
SecurityAlert alert = {...};
crypto::signature sig = sign_alert(alert, m_node_private_key);
```

```
// Al recibir, verificar firma
if (!verify_signature(alert, sig, sender_public_key)) {
  REJECT_ALERT();
}
```

11.2 Preveninción de Replay

```
struct SecurityAlert {
  std::string query_id;      // ID único
  uint64_t timestamp;        // Timestamp del evento
  uint64_t nonce;            // Nonce aleatorio
  // ...
};

// Verificar que no es un reporte duplicado
if (m_seen_alerts.count(alert.query_id)) {
  IGNORE("Duplicate alert");
}
```

---

12. FAQ / Preguntas Frecuentes

P: ¿Qué pasa si todos los peers están caídos?

R: Se usa el PAUSE MODE actual. El consenso solo aplica cuando hay peers disponibles.

P: ¿Un ataque local afecta la reputación del nodo?

R: No, si se detecta como "nodo bajo ataque". Sistema automáticamente lo pone en QUARANTINE y protege su reputación.

P: ¿Cómo sé si estoy bajo ataque?

R: El nodo mostrará un aviso destacado en terminal: "⚠ ATAQUE DETECTADO - QUARANTINE". Aparecerá un cuadro con información sobre la fuente de ataque y cuánto tiempo durará la cuarentena.

P: ¿Cuánto dura la cuarentena?

R: Entre 1-6 horas (configurable). Termina automáticamente o cuando se confirme el ataque a nivel de red.

P: Si me ponen en quarantine, ¿pierdo mis reportes?

R: No. Se cuentan pero NO afectan tu reputación. Cuando salgas y se confirme el ataque = +0.2 reputación como recompensa.

P: ¿Cómo se evita que un atacante innde la red con alerts?

R: Verificación de firma + baja reputación → los alerts falsos se ignoran automáticamente.

P: ¿Se mantiene PAUSE MODE sin cambios?

R: Sí, 100% sin cambios. El consenso es complementario, no reemplaza.

P: ¿Qué pasa si un nodo está en quarantine y se cae?

R: Se restaura la duración del quarantine en el próximo inicio. Datos persistidos en JSON.

P: ¿Puede un nodo quedarse en quarantine para siempre?

R: No. Máximo 6 horas. Después sale automáticamente y su reputación se restaura (o aumenta si se confirma).

FIN DEL DOCUMENTO