Sprint 3: Integración (Phase 2 - 50% a 75%)

Fecha: 25 enero 2026
Objetivo: Integrar `security_query_tool.cpp` + `reputation_manager.cpp` en el sistema de checkpoints
Estado: 🔄 EN PLANIFICACIÓN
Estimado: 1-2 semanas

📋 Tareas Principales

1. ☑ Análisis de Arquitectura Actual

Archivo Principal: `src/checkpoints/checkpoints.h` + `src/checkpoints/checkpoints.cpp`

Estructura Actual:

```
class checkpoints {
  public:
    // Core validation
    bool check_block(uint64_t height, const crypto::hash& h) const;
    bool is_in_checkpoint_zone(uint64_t height) const;

    // Loading
    bool load_new_checkpoints(const std::string &json_file, network_type
nettype);
    bool load_checkpoints_from_json(const std::string &path);
    bool load_checkpoints_from_seed_nodes();

    // Verification
    bool verify_with_seeds(const rapidjson::Document &checkpoints, uint64_t
epoch);
    bool auto_repair_checkpoint_conflict(uint64_t height, const crypto::hash&
hash, ...);

  private:
    std::map<uint64_t, crypto::hash> m_points;           // Checkpoints
    std::map<uint64_t, difficulty_type> m_difficulty_points;  // Diffs
    std::vector<BanInfo> m_permanent_bans;               // Bans
};
```

Puntos de Integración Identificados:

1. **After** `load_checkpoints_from_seed_nodes()` → **Crear:** `initiate_consensus_query()`
2. **During** `verify_with_seeds()` → **Usar:** Reputation-based peer selection
3. **In** `check_block()` → **Opcionalmente:** Query en caso de conflicto
4. **New member** → `SecurityQueryTool` instance

5. **New member** → `ReputationManager` instance

Archivos a Modificar

1. `src/checkpoints/checkpoints.h` - Header

Cambios Necesarios:

- Agregar includes para `security_query_tool.hpp` y `reputation_manager.hpp`
- Agregar miembros privados:

```
private:
  std::unique_ptr<ninacatcoin::security::SecurityQueryTool>
m_security_query_tool;
  std::unique_ptr<ninacatcoin::security::ReputationManager>
m_reputation_manager;
```

- Agregar métodos públicos:

```
public:
  // Consensus queries
  bool initiate_consensus_query(uint64_t height, const crypto::hash&
suspect_hash);
  bool handle_security_query(const SecurityQuery& query);
  bool handle_security_response(const SecurityResponse& response);

  // Reputation management
  void report_peer_reputation(const std::string& peer_id, bool
was_valid);
  float get_peer_reputation(const std::string& peer_id) const;
  bool is_peer_trusted(const std::string& peer_id) const;
```

Líneas Estimadas: +20 líneas (includes + métodos + miembros)

---

2. `src/checkpoints/checkpoints.cpp` - Implementación

Cambios Necesarios:

A. Constructor & Destructor

```
checkpoints::checkpoints() {
  // Inicializar security tools
  m_security_query_tool =
std::make_unique<ninacatcoin::security::SecurityQueryTool>();
  m_reputation_manager =
```

```
std::make_unique<ninacatcoin::security::ReputationManager>();
}
```

B. Integración en `load_checkpoints_from_seed_nodes()`

Contexto: Cuando se cargan checkpoints desde seed nodes

```
bool checkpoints::load_checkpoints_from_seed_nodes() {
  // ... existing code ...

  // NEW: Consultar consenso si hay múltiples seeds
  if (seed_responses.size() > 1) {
    if (!initiate_consensus_query(epoch_height, seed_hash)) {
      MERROR("Consensus query failed");
      // Decide: ¿fallar o continuar?
    }
  }
}
```

C. Integración en `verify_with_seeds()`

Contexto: Verificar checkpoints contra seed nodes

```
bool checkpoints::verify_with_seeds(const rapidjson::Document
&received_checkpoints,
                                    uint64_t received_epoch) {
  // NEW: Usar reputación para ordenar seeds
  std::vector<std::string> trusted_seeds;
  for (const auto& seed : m_seed_nodes) {
    if (m_reputation_manager->is_peer_trusted(seed.id)) {
      trusted_seeds.push_back(seed);
    }
  }

  // ... verificación con seeds ordenados por reputación ...
}
```

D. Nuevo Método: `initiate_consensus_query()`

Propósito: Cuando detectamos conflicto, consultar a pares

```
bool checkpoints::initiate_consensus_query(uint64_t height,
                                           const crypto::hash& suspect_hash) {
  // 1. Crear query
  auto query = m_security_query_tool->create_query(height, suspect_hash);
```

```
  // 2. Enviar a N pares (usar P2P message queue)
  for (const auto& peer : m_trusted_peers) {  // filtered by reputation
    send_security_query_message(peer, query);
  }

  // 3. Esperar respuestas (hasta timeout)
  std::this_thread::sleep_for(std::chrono::seconds(30));

  // 4. Calcular consenso
  auto consensus = m_security_query_tool->calculate_consensus();

  // 5. Manejar resultado
  if (consensus.result == ConsensusResult::NETWORK_ATTACK_CONFIRMED) {
    report_network_attack(height, suspect_hash);
    return true;
  } else if (consensus.result == ConsensusResult::LOCAL_ATTACK) {
    activate_quarantine(height);
    return false;  // No es ataque global, es local
  }

  return false;
}
```

E. Nuevos Métodos: `handle_security_query/response()`

Propósito: Handlers para mensajes P2P

```
bool checkpoints::handle_security_query(const SecurityQuery& query) {
  // 1. Validar query
  if (!validate_query(query)) return false;

  // 2. Buscar checkpoint local
  auto local_checkpoint = get_checkpoint(query.height);

  // 3. Preparar respuesta
  SecurityResponse response;
  response.query_id = query.id;
  response.node_id = get_my_node_id();
  response.matches_local = (local_checkpoint == query.hash);

  // 4. Serializar y devolver
  send_security_response_message(query.source_peer, response);

  // 5. Registrar en reputation_manager
  m_reputation_manager->on_report_sent(query.source_peer, 1);

  return true;
}
```

```
bool checkpoints::handle_security_response(const SecurityResponse& response) {
  // 1. Validar respuesta
  if (!m_security_query_tool->validate_response_signature(response)) {
    MWARNING("Invalid signature on response from " << response.node_id);
    m_reputation_manager->on_report_rejected(response.node_id);
    return false;
  }

  // 2. Agregar a resultados de query
  m_security_query_tool->add_response(response);

  // 3. Si respuesta coincide con consenso: aumentar reputación
  if (response.matches_local) {
    m_reputation_manager->on_report_confirmed(response.node_id);
  } else {
    m_reputation_manager->on_report_rejected(response.node_id);
  }

  return true;
}
```

F. Métodos Helper: `report_peer_reputation()`**, etc.**

```
void checkpoints::report_peer_reputation(const std::string& peer_id, bool
was_valid) {
  if (was_valid) {
    m_reputation_manager->on_report_confirmed(peer_id);
  } else {
    m_reputation_manager->on_report_rejected(peer_id);
  }
}

float checkpoints::get_peer_reputation(const std::string& peer_id) const {
  return m_reputation_manager->get_score(peer_id);
}

bool checkpoints::is_peer_trusted(const std::string& peer_id) const {
  return m_reputation_manager->is_trusted(peer_id);
}
```

Líneas Estimadas: +200 líneas (integración + nuevos métodos)

🔗 Integración P2P (NetworkManager)

Archivos Afectados

Buscar y modificar handlers P2P para nuevos mensajes:

- `src/p2p/net_node.cpp` - Agregar message handlers
- `src/p2p/net_node.h` - Agregar message types

Nuevos Message Types

```
// En net_node.h
struct NOTIFY_SECURITY_QUERY {
  SecurityQuery query;
};

struct NOTIFY_SECURITY_RESPONSE {
  SecurityResponse response;
};
```

Message Handlers

```
// En net_node.cpp
int netNode::handle_notify_security_query(int version, const
NOTIFY_SECURITY_QUERY &arg, ...) {
  return m_checkpoints->handle_security_query(arg.query) ? 1 : 0;
}

int netNode::handle_notify_security_response(int version, const
NOTIFY_SECURITY_RESPONSE &arg, ...) {
  return m_checkpoints->handle_security_response(arg.response) ? 1 : 0;
}
```

🧪 Testing Strategy

Unit Tests para Nueva Integración

Crear: `tests/unit_tests/checkpoints_phase2.cpp`

Test Cases:

1. ☑ Constructor inicializa tools correctamente
2. ☑ `initiate_consensus_query()` crea queries
3. ☑ `handle_security_query()` valida y responde
4. ☑ `handle_security_response()` agrega respuestas
5. ☑ Consensus calcula resultado correcto
6. ☑ Reputación se actualiza en respuestas
7. ☑ Quarantine se activa en ataque local
8. ☑ Network attack se reporta correctamente

Estimado: 300 líneas, 8-10 tests

Integration Tests

Crear: `tests/integration_tests/checkpoints_phase2_e2e.cpp`

Escenarios:

1. Cargar checkpoints desde seed nodes
2. Recibir checkpoint conflictivo
3. Iniciar query de consenso
4. Procesar múltiples respuestas
5. Calcular consenso correctamente
6. Actualizar reputación de pares
7. Detectar y cuarentenar ataque local

Estimado: 500 líneas, 7 scenarios

📊 Dependencies

Archivos que necesitan includes adicionales:

```
#include "tools/security_query_tool.hpp"      // QueryManager, SecurityQuery,
etc.
#include "tools/reputation_manager.hpp"       // ReputationManager
#include <memory>                              // std::unique_ptr
#include <thread>                              // Para timeouts
#include <chrono>                              // Para duraciones
```

Compilar con:

```
g++ -I./tools -I./src ... checkpoints.cpp security_query_tool.cpp
reputation_manager.cpp
```

🎯 Milestones

#	Tarea	Estado	Est.	Fecha
1	Agregar includes y miembros a header	⏳ TODO	1h	26 ene
2	Implementar constructor/destructor	⏳ TODO	1h	26 ene
3	Integrar en `load_checkpoints_from_seed_nodes()`	⏳ TODO	2h	26-27 ene
4	Integrar en `verify_with_seeds()`	⏳ TODO	2h	27 ene
5	Implementar `initiate_consensus_query()`	⏳ TODO	3h	27-28 ene

#	Tarea	Estado	Est.	Fecha
6	Implementar `handle_security_query()`	⏳ TODO	2h	28 ene
7	Implementar `handle_security_response()`	⏳ TODO	2h	28 ene
8	Implementar helpers (reputation methods)	⏳ TODO	1h	28 ene
9	Agregar message types a net_node.h	⏳ TODO	1h	29 ene
10	Agregar handlers a net_node.cpp	⏳ TODO	2h	29 ene
11	Crear unit tests	⏳ TODO	3h	30-31 ene
12	Crear integration tests	⏳ TODO	4h	31 ene - 1 feb
13	Testing y validación	⏳ TODO	3h	1-2 feb
14	Documentación (SPRINT_3_COMPLETADO.md)	⏳ TODO	2h	2 feb

Total Estimado: 30 horas ≈ 3-4 días de trabajo intenso

🔍 Validación de Integración

Compilation Checks

```
# Sin errores
make checkpoints
# Sin warnings sobre includes sin usar
# Linking correcto con tools/
```

Runtime Validation

- [x] SecurityQueryTool se inicializa correctamente
- [x] ReputationManager carga/guarda JSON
- [x] Queries se crean sin errores
- [x] Respuestas se procesan correctamente
- [x] Consensus calcula resultado esperado

Code Review Checklist

- [ ] Memory management: no leaks de unique_ptr
- [ ] Thread safety: mutex si es necesario
- [ ] Error handling: todos los caminos
- [ ] Logging: MERROR, MINFO, MWARNING
- [ ] Comments: explicar lógica compleja

📝 Notas Técnicas

Consideraciones Importantes:

1. **Threading:** ReputationManager acceso async → considerar mutex
2. **Persistence:** Guardar reputation cada N respuestas (no cada vez)
3. **P2P Latency:** Timeout de 30s es razonable, pero confirmable
4. **Fallback:** Si query falla, continuar con checkpoints existentes
5. **Backward Compat:** Phase 1 debe funcionar sin changes

🚀 Cómo Proceder

```
AHORA (25 ene):
├─ Revisar estructura de net_node.cpp
├─ Identificar donde agregar message handlers
└─ Preparar lista de includes necesarios

MAÑANA (26 ene):
├─ Modificar checkpoints.h
├─ Implementar constructor/destructor
└─ Integrar en load_checkpoints_from_seed_nodes()

DÍA 3-4 (27-28 ene):
├─ Implementar initiate_consensus_query()
├─ Implementar handlers (query/response)
└─ Testing inicial

DÍA 5 (29 ene - 2 feb):
├─ Integración P2P (net_node)
├─ Unit tests + integration tests
└─ Documentación final
```

Estado Final: Lista para comenzar Sprint 3 ☑