Ethernity MEV Detector - API e Guia de Implementação

1. Configuração Inicial

1.1 Dependências

toml

```toml
[dependencies]
ethernity-detector-mev = { path = "../ethernity-detector-mev" }
ethernity-core = { path = "../ethernity-core" }
ethernity-rpc = { path = "../ethernity-rpc" }
tokio = { version = "1.0", features = ["full"] }
ethereum-types = "0.14"
```

1.2 Inicialização Básica

rust

```rust
use ethernity_detector_mev::*;
use ethernity_rpc::{EthernityRpcClient, RpcConfig};
use std::sync::Arc;

#[tokio::main]
async fn main() -> Result<()> {
    // Configurar cliente RPC
    let rpc_config = RpcConfig {
        endpoint: "https://mainnet.infura.io/v3/YOUR_KEY".into(),
        ..Default::default()
    };
    let rpc = Arc::new(EthernityRpcClient::new(rpc_config).await?);

    // Inicializar componentes
    let tagger = TxNatureTagger::new(rpc.clone());
    let mut aggregator = TxAggregator::new();
    let detector = AttackDetector::new(1.0, 10);

    Ok(())
}
```

2. API Principal

2.1 TxNatureTagger

```rust
impl<P: RpcProvider> TxNatureTagger<P> {
    /// Cria nova instância do tagger
    pub fn new(provider: P) -> Self

    /// Analisa uma transação e retorna sua natureza
    pub async fn analyze(
        &self,
        to: Address,
        input: &[u8],
        tx_hash: TransactionHash,
    ) -> Result<TxNature>

    /// Processa stream de transações brutas
    pub async fn process_stream(
        &self,
        rx: Receiver<RawTx>,
        tx: Sender<AnnotatedTx>,
    )
}
```

Estruturas de Dados

```rust
pub struct TxNature {
    pub tx_hash: TransactionHash,
    pub tags: Vec<String>,                // ["swap-v2", "router-call"]
    pub token_paths: Vec<Address>,        // Tokens envolvidos
    pub targets: Vec<Address>,            // Contratos alvo
    pub confidence: f64,                  // 0.0 - 1.0
    pub confidence_components: ConfidenceComponents,
    pub extracted_fallback: bool,
    pub ambiguous_execution_path: bool,
    pub reachable_via_dispatcher: bool,
    pub path_inference_failed: bool,
}

pub struct ConfidenceComponents {
    pub abi_match: f64,     // Confiança na correspondência ABI
    pub structure: f64,     // Confiança na estrutura do bytecode
    pub path: f64,          // Confiança no caminho de tokens
}
```

2.2 TxAggregator

```rust
impl TxAggregator {
    /// Cria novo agregador
    pub fn new() -> Self

    /// Adiciona transação e retorna chave do grupo
    pub fn add_tx(&mut self, tx: AnnotatedTx) -> Option<H256>

    /// Adiciona transação e emite evento
    pub fn add_tx_event(&mut self, tx: AnnotatedTx) -> Option<AggregationEvent>

    /// Obtém grupos ativos
    pub fn groups(&self) -> &HashMap<H256, TxGroup>

    /// Define início da janela
    pub fn set_window_start(&mut self, start: u64)

    /// Finaliza grupos e retorna eventos
    pub fn finalize_events(&mut self, complete: bool) -> Vec<AggregationEvent>
}
```

Critérios de Agrupamento

```rust
// Transações são agrupadas por:
// 1. Mesmo conjunto de tokens (ordenado)
// 2. Mesmos contratos alvo
// 3. Tags compatíveis
// 4. Janela temporal (configurável)

// Grupo válido requer:
// - Mínimo 2 tokens no caminho
// - Pelo menos 1 alvo
// - Tag permitida (swap-v2, swap-v3, etc.)
```

2.3 StateSnapshotRepository

rust

```rust
impl<P: RpcProvider> StateSnapshotRepository<P> {
    /// Abre repositório (cria se não existir)
    pub fn open(provider: P, path: impl AsRef<Path>) -> Result<Self>

    /// Captura snapshots para grupos
    pub async fn snapshot_groups(
        &self,
        groups: &HashMap<H256, TxGroup>,
        block_number: u64,
        profile: SnapshotProfile,
    ) -> Result<()>

    /// Recupera snapshot armazenado
    pub fn get_state(
        &self,
        address: Address,
        block_number: u64,
        profile: SnapshotProfile,
    ) -> Option<StateSnapshot>

    /// Backup do banco de dados
    pub fn backup(&self, dest: impl AsRef<Path>) -> Result<()>

    /// Restaura de backup
    pub fn restore(
        provider: P,
        src: impl AsRef<Path>,
        dest: impl AsRef<Path>,
    ) -> Result<Self>
}
```

Perfis de Snapshot

rust

```rust
pub enum SnapshotProfile {
    Basic,    // Apenas reserves/slot0
    Extended, // Incluindo liquidez e metadados
    Deep,     // Análise completa com histórico
}
```

2.4 StateImpactEvaluator

rust

```rust
impl StateImpactEvaluator {
    /// Cria avaliador com parâmetros customizados
    pub fn new(params: ImpactModelParams) -> Self

    /// Avalia impacto de grupo (função estática)
    pub fn evaluate(
        group: &TxGroup,
        victims: &[VictimInput],
        snapshot: &StateSnapshot,
    ) -> GroupImpact
}
```

Configuração de Parâmetros

rust

```rust
pub struct ImpactModelParams {
    pub liquidity: f64,               // Multiplicador de liquidez
    pub slippage_curve: f64,          // Slippage base esperada
    pub convexity: f64,               // Limite de convexidade
    pub curve_model: Arc<dyn CurveModel>, // Modelo AMM
    pub lightweight_simulation: bool,  // Simular sequência
}

// Modelos de curva disponíveis
let constant_product = ConstantProductCurve { fee: 0.003 };
let uniswap_v3 = UniswapV3Curve::default();
```

2.5 AttackDetector

rust

```rust
impl AttackDetector {
    /// Cria detector com parâmetros
    pub fn new(
        base_fee: f64,
        entropy_tolerance_window: u64,
    ) -> Self

    /// Analisa grupo para detectar ataques
    pub fn analyze_group(&self, group: &TxGroup) -> Option<AttackVerdict>
}
```

Tipos de Ataque

rust

```rust
pub enum AttackType {
    Frontrun { justification: String },
    Sandwich { justification: String },
    Spoof { justification: String },
    Backrun { justification: String },
    CrossChain { justification: String },
    FlashLoan { justification: String },
    MultiToken { justification: String },
    Layer2 { justification: String },
}

pub struct AttackVerdict {
    pub group_key: H256,
    pub attack_types: Vec<AttackType>,
    pub confidence: f64,            // 0.0 - 1.0
    pub reconsiderable: bool,       // Se deve reavaliar
}
```

2.6 MempoolSupervisor

rust

```rust
impl<P: RpcProvider> MempoolSupervisor<P> {
    /// Cria supervisor
    pub fn new(
        provider: P,
        min_tx_count: usize,
        dt_max: Duration,
        max_active_groups: usize,
    ) -> Self

    /// Ingere nova transação
    pub fn ingest_tx(&mut self, tx: AnnotatedTx)

    /// Executa tick do supervisor
    pub async fn tick(&mut self) -> Result<Vec<GroupReady>>

    /// Processa stream de eventos
    pub async fn process_stream(
        self,
        rx: Receiver<SupervisorEvent>,
        tx: Sender<GroupReady>,
    )
}
```

3. Traits e Extensibilidade

3.1 StateProvider

rust

```
#[async_trait]
pub trait StateProvider: Send + Sync {
    /// Retorna reserves de um par AMM
    async fn reserves(&self, address: Address) -> Result<(U256, U256)>;

    /// Retorna slot0 (Uniswap V3)
    async fn slot0(&self, address: Address) -> Result<(U256, U256)>;
}
```

3.2 TransactionClassifier

rust

```
#[async_trait]
pub trait TransactionClassifier: Send + Sync {
    /// Classifica transação com predições
    async fn classify(
        &self,
        to: Address,
        input: &[u8],
        tx_hash: TransactionHash,
    ) -> Result<Vec<TagPrediction>>;
}
```

3.3 ImpactModel

rust

```
pub trait ImpactModel: Send + Sync {
    /// Avalia impacto econômico de grupo
    fn evaluate_group(
        &mut self,
        group: &TxGroup,
        victims: &[VictimInput],
        snapshot: &StateSnapshot,
    ) -> GroupImpact;
}
```

3.4 CurveModel

rust

```rust
pub trait CurveModel: Send + Sync {
    /// Calcula saída esperada
    fn expected_out(&self, amount_in: f64, snapshot: &StateSnapshot) -> f64;

    /// Aplica trade ao snapshot (mutável)
    fn apply_trade(&self, amount_in: f64, snapshot: &mut StateSnapshot);
}
```

4. Pipeline de Eventos

4.1 Configuração de Pipeline

rust

```rust
// Criar canais
let (tx_raw, rx_raw) = mpsc::channel(1024);
let (tx_annotated, rx_annotated) = mpsc::channel(512);
let (tx_grouped, rx_grouped) = mpsc::channel(256);
let (tx_snapshot, rx_snapshot) = mpsc::channel(128);
let (tx_impact, rx_impact) = mpsc::channel(128);
let (tx_threat, rx_threat) = mpsc::channel(64);

// Spawnar processadores
tokio::spawn(tagger.process_stream(rx_raw, tx_annotated));
tokio::spawn(aggregator.process_stream(rx_annotated, tx_grouped));
tokio::spawn(evaluator.process_stream(rx_snapshot, tx_impact));
tokio::spawn(detector.process_stream(rx_impact, tx_threat));
```

4.2 Eventos do Sistema

rust

```rust
pub enum SupervisorEvent {
    NewTxObserved(AnnotatedTx),
    BlockAdvanced(BlockMetadata),
    StateRefreshed(String),
    GroupFinalized(String),
}

pub struct GroupReady {
    pub group: TxGroup,
    pub metadata: SyncMetadata,
}

pub struct SyncMetadata {
    pub window_id: u64,
    pub block_shadowed: bool,
    pub evaluated_with_stale_state: bool,
    pub timestamp_drifted: bool,
    pub state_alignment_score: f64,
    pub timestamp_jitter_score: f64,
}
```

5. Padrões de Uso Avançados

5.1 Detector Customizado

rust

```rust
struct MyCustomDetector;

impl MyCustomDetector {
    fn detect_custom_pattern(&self, group: &TxGroup) -> Option<AttackType> {
        // Lógica customizada
        if group.txs.len() > 5 && self.has_circular_path(group) {
            Some(AttackType::CrossChain {
                justification: "circular token path detected".into()
            })
        } else {
            None
        }
    }

    fn has_circular_path(&self, group: &TxGroup) -> bool {
        // Implementação
        false
    }
}
```

5.2 Provider com Fallback

rust

```rust
let primary = EthernityRpcClient::new(primary_config).await?;
let fallback = EthernityRpcClient::new(fallback_config).await?;

let provider = RpcStateProvider::with_fallback(primary, fallback);
```

5.3 Modelo de Impacto Customizado

rust

```rust
struct MyImpactModel {
    base_model: StateImpactEvaluator,
    custom_factor: f64,
}

impl ImpactModel for MyImpactModel {
    fn evaluate_group(
        &mut self,
        group: &TxGroup,
        victims: &[VictimInput],
        snapshot: &StateSnapshot,
    ) -> GroupImpact {
        let mut impact = self.base_model.evaluate_group(group, victims, snapshot);

        // Aplicar lógica customizada
        if self.is_high_value_target(&group.targets[0]) {
            impact.opportunity_score *= self.custom_factor;
        }

        impact
    }
}
```

6. Tratamento de Erros

6.1 Erros Comuns

rust

```rust
use ethernity_core::error::{Error, Result};

// RPC failures
match provider.call(address, data).await {
    Ok(result) => process(result),
    Err(Error::RpcError(msg)) => {
        // Retry com backoff ou usar fallback
    }
    Err(e) => return Err(e),
}

// Decode failures
match serde_json::from_slice::<StateSnapshot>(&data) {
    Ok(snapshot) => snapshot,
    Err(_) => {
        return Err(Error::DecodeError("invalid snapshot".into()));
    }
}
```

6.2 Recuperação Graceful

rust

```rust
// Fork detection e re-fetch
if saved_snapshot.block_hash != current_hash {
    // Invalida cache e busca novamente
    self.invalidate_cache(block_number)?;
    self.fetch_fresh_state(address, block_number).await?;
}

// Modo degradado para alta carga
match self.operational_mode {
    OperationalMode::Burst => {
        // Reduz TTL, processa apenas high-priority
    }
    OperationalMode::Recovery => {
        // Aumenta buffers, relaxa validações
    }
    _ => {}
}
```