Smart Contract Analysis

Enter Contract ID

Enter Contract Name

Analyze

Contract code retrived successfully!

```
;;; UniswapV2Router02.sol

(use-trait ft-trait 'SP2AKWJYC7BNY18W1XXKPGP0YVEK63QJG4793Z2D4.sip-010-tr...
```

Analysis complete!

Comprehensive Report on UniswapV2Router02 Smart Contract

1. Contract Code: UniswapV2Router02.sol

```
(use-trait ft-trait 'SP2AKWJYC7BNY18W1XXKPGP0YVEK63QJG4793Z2D4.sip-010-trait-ft-st
(use-trait ft-plus-trait .ft-plus-trait.ft-plus-trait)
(use-trait share-fee-to-trait .univ2-share-fee-to-trait.share-fee-to-trait)

(define-constant err-router-preconditions  (err u200))
(define-constant err-router-postconditions (err u201))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;; add-liquidity
(define-read-only
  (add-liquidity-calc
    (id           uint)
    (amt0-desired uint)
    (amt1-desired uint)
    (amt0-min     uint)
    (amt1-min     uint))
  (let ((pool (contract-call? .univ2-core do-get-pool id))
        (r0   (get reserve0 pool))
        (r1   (get reserve1 pool)))
    (if (and (is-eq r0 u0) (is-eq r1 u0))
        (ok {amt0: amt0-desired, amt1: amt1-desired})
        (let ((amt1-optimal (try! (contract-call? .univ2-library quote amt0-desire
              (amt0-optimal (try! (contract-call? .univ2-library quote amt1-desire
            ;; Note we do not use optimal if > desired.
            (if (<= amt1-optimal amt1-desired)
                (begin
                  (asserts! (>= amt1-optimal amt1-min) err-router-preconditions)
                  (ok {amt0: amt0-desired, amt1: amt1-optimal}))
                (begin
                  (asserts!
                    (and
                      (<= amt0-optimal amt0-desired)
                      (>= amt0-optimal amt0-min))
                    err-router-preconditions)
                  (ok {amt0: amt0-optimal, amt1: amt1-desired})) )) )))

(define-public
  (add-liquidity
    (id uint)
    (token0       <ft-trait>)
    (token1       <ft-trait>)
    (lp-token     <ft-plus-trait>)
    (amt0-desired uint)
    (amt1-desired uint)
    (amt0-min     uint)
    (amt1-min     uint))

  (let ((amts (try! (add-liquidity-calc
                id amt0-desired amt1-desired amt0-min amt1-min))))

    (asserts!
     (and (<= amt0-min amt0-desired)
```

```
          (<= amt1-min amt1-desired)
          (>= amt0-min u0)
          (>= amt1-min u0)
          (>= amt0-desired u0)
          (>= amt1-desired u0))
      err-router-preconditions)

    (contract-call? .univ2-core mint
      id
      token0
      token1
      lp-token
      (get amt0 amts)
      (get amt1 amts)) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; remove-liquidity
(define-public
  (remove-liquidity
    (id        uint)
    (token0    <ft-trait>)
    (token1    <ft-trait>)
    (lp-token  <ft-plus-trait>)
    (liquidity uint)
    (amt0-min  uint)
    (amt1-min  uint))

  (let ((event (try! (contract-call? .univ2-core burn
                  id token0 token1 lp-token liquidity))))

    (asserts!
      (and (>= (get amt0 event) amt0-min)
           (>= (get amt1 event) amt1-min))
      err-router-postconditions)

    (ok event) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; swap
(define-public
  (swap-exact-tokens-for-tokens
    (id             uint)
    (token0         <ft-trait>)
    (token1         <ft-trait>)
```

```
    (token-in       <ft-trait>)
    (token-out      <ft-trait>)
    (share-fee-to   <share-fee-to-trait>)
    (amt-in      uint)
    (amt-out-min uint))

  (let ((pool      (contract-call? .univ2-core do-get-pool id))
        (is-token0 (is-eq (contract-of token0) (contract-of token-in)))
        (amt-out   (try! (contract-call? .univ2-library get-amount-out
          amt-in
          (if is-token0 (get reserve0 pool) (get reserve1 pool))
          (if is-token0 (get reserve1 pool) (get reserve0 pool))
          (get swap-fee pool) )))
       (event      (try! (contract-call? .univ2-core swap
          id
          token-in
          token-out
          share-fee-to
          amt-in
          amt-out))) )

    (asserts!
     (and (is-eq (get token0 pool) (contract-of token0))
          (is-eq (get token1 pool) (contract-of token1))
          (> amt-in      u0)
          (> amt-out-min u0) )
     err-router-preconditions)

    (asserts!
      (and (>= (get amt-out event) amt-out-min))
      err-router-postconditions)

    (ok event) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define-public
  (swap-tokens-for-exact-tokens
    (id           uint)
    (token0       <ft-trait>)
    (token1       <ft-trait>)
    (token-in     <ft-trait>)
    (token-out    <ft-trait>)
    (share-fee-to <share-fee-to-trait>)
    (amt-in-max   uint)
```

```
    (amt-out      uint))

  (let ((pool      (contract-call? .univ2-core do-get-pool id))
        (is-token0 (is-eq (contract-of token0) (contract-of token-in)))
        (amt-in    (try! (contract-call? .univ2-library get-amount-in
          amt-out
          (if is-token0 (get reserve0 pool) (get reserve1 pool))
          (if is-token0 (get reserve1 pool) (get reserve0 pool))
          (get swap-fee pool) )))
        (event     (try! (contract-call? .univ2-core swap
          id
          token-in
          token-out
          share-fee-to
          amt-in
          amt-out))) )

  (asserts!
   (and (is-eq (get token0 pool) (contract-of token0))
        (is-eq (get token1 pool) (contract-of token1))
        (> amt-in-max u0)
        (> amt-out    u0) )
   err-router-preconditions)

  (asserts!
    (and (<= (get amt-in event) amt-in-max))
    err-router-postconditions)

  (ok event) ))

;;; eof
```

2. Summary

The UniswapV2Router02.sol contract is designed to facilitate decentralized exchange functionalities similar to Uniswap V2 but implemented on the Stacks blockchain using the Clarity language. Key functionalities include adding liquidity, removing liquidity, and swapping tokens. The contract integrates multiple traits and defines constants for error handling. The add-liquidity functionality calculates and asserts the optimal token amounts before minting new liquidity tokens. The remove-liquidity function burns liquidity tokens and ensures the minimum amounts are met. The swap functionalities allow for

token swaps with checks on input and output amounts, utilizing external library calls for price calculations.

3. Functions Analysis

add-liquidity-calc

- **Purpose**: Calculate the optimal amounts of tokens to add to the liquidity pool.
- **Logic**:
  - Retrieves the pool's current reserves.
  - Determines if the reserves are initially zero.
  - Calculates optimal token amounts using a quoting function.
  - Asserts precondition checks to prevent adding less than the minimum amounts.

add-liquidity

- **Purpose**: Add liquidity to a pool with the specified token amounts.
- **Logic**:
  - Calls `add-liquidity-calc` to get optimal amounts.
  - Asserts preconditions for the provided token amounts.
  - Mints liquidity tokens using the core contract.

remove-liquidity

- **Purpose**: Remove liquidity from a pool.
- **Logic**:
  - Burns liquidity tokens.
  - Asserts postcondition checks to ensure minimum amounts are met.

swap-exact-tokens-for-tokens

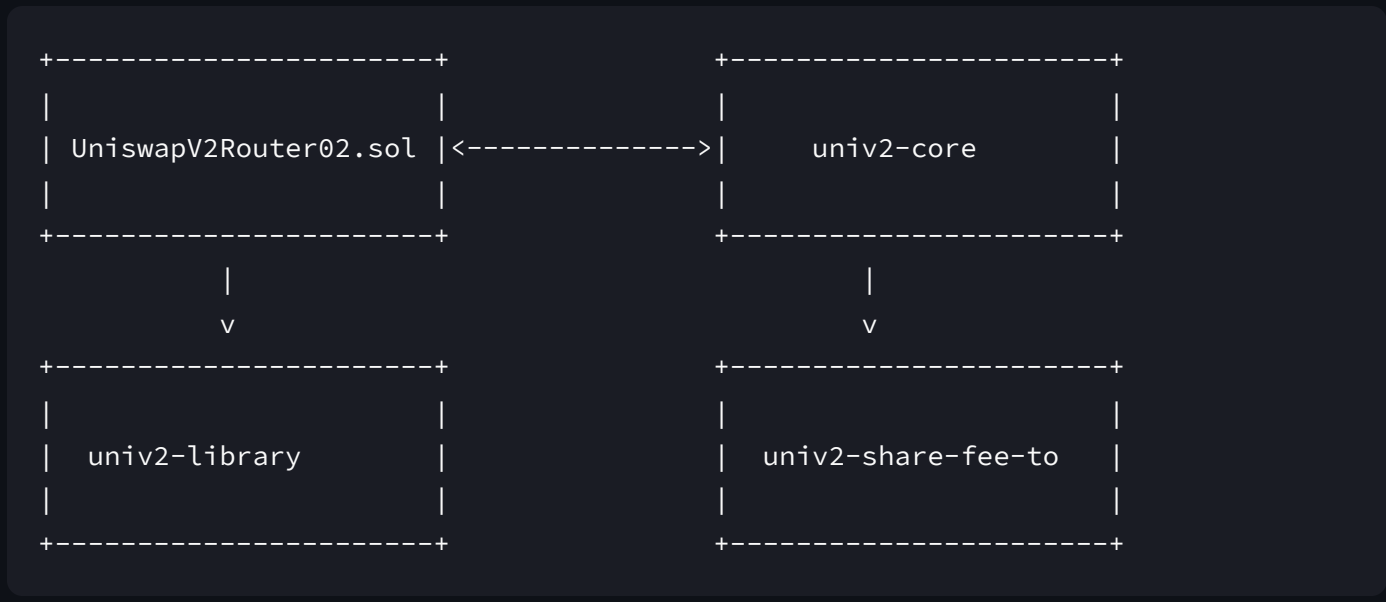
- **Purpose**: Perform a token swap where the input token amount is fixed.
- **Logic**:
  - Retrieves the pool's reserves.
  - Calculates the output token amount.
  - Asserts preconditions and postconditions.

- Executes the swap using the core contract.

swap-tokens-for-exact-tokens

- **Purpose**: Perform a token swap where the output token amount is fixed.
- **Logic**:
  - Retrieves the pool's reserves.
  - Calculates the input token amount.
  - Asserts preconditions and postconditions.
  - Executes the swap using the core contract.

4. Diagrams



5. Updateability

The contract's updateability depends on the mechanisms provided by the Stacks blockchain and the governance procedures of the deployed protocol. Generally, smart contracts in Clarity are immutable once deployed. However, upgrade patterns such as proxy contracts or governance-controlled contract redeployment can be explored.

6. Security

Reentrancy Attack

- **Description**: An attacker could exploit the `swap-exact-tokens-for-tokens` or `swap-tokens-for-exact-tokens` functions by recursively calling these functions before the initial transaction is completed, thereby manipulating the token balances.
- **Mitigation**:
  - Minimize external calls and ensure they are made only after internal state changes.
  - Use the Checks-Effects-Interactions pattern.

Front-Running

- **Description**: In the `add-liquidity` and `remove-liquidity` functions, an attacker could front-run a liquidity addition or removal to benefit from the price changes that occur as a result.
- **Mitigation**:
  - Implement Commit-Reveal schemes.
  - Introduce time delays or require multiple transactions for critical operations.

Price Manipulation

- **Description**: Functions like `add-liquidity-calc`, `swap-exact-tokens-for-tokens`, and `swap-tokens-for-exact-tokens` could be vulnerable if an attacker can manipulate the price oracle or perform large trades that swing the price.
- **Mitigation**:
  - Use time-weighted average prices (TWAP).
  - Ensure the use of decentralized and trustworthy oracles.

Arithmetic Overflows and Underflows

- **Description**: An attacker might exploit arithmetic overflows or underflows in functions like `add-liquidity-calc` or `get-amount-out` to manipulate the amount of tokens being swapped or added/removed from liquidity.
- **Mitigation**:
  - Use safe math libraries.
  - Validate input values to ensure they are within expected ranges.

Unauthorized Access

- **Description**: Functions like `mint` and `burn` in the core contracts could be exploited if they are not properly restricted, allowing unauthorized users to manipulate token supplies.

- **Mitigation**:
  - Implement proper access controls and authentication mechanisms.
  - Ensure sensitive functions are only callable by authorized contracts or addresses.

This comprehensive report integrates all analyses, including contract code, summary, functions analysis, diagrams, updateability, and security considerations, providing a thorough understanding of the UniswapV2Router02 smart contract on the Stacks blockchain.