Report on Uswap

Prepared By: Grim Reaper

Telegram: @grimreaper619

Disclaimer

This text is not a call to participate in the project and it’s only a description of the smart contract work at the specified address. Remember that you do all the financial actions only at your own risk.

Description

The project is an indirect clone of Uniswap V2 which is modified to suite for tron blockchain. It consists of two contracts, uswap, which creates a trc20 token named Uswap, deals with the token pair and a factory which creates these pairs. Another contract named uswap.route which deals with adding and removing liquidity of the token pairs. It also consists of a number of utility functions for frontend development.

Variables

Uswap

```
string public constant name = 'USwap';

string public constant symbol = 'USP';

uint8 public constant decimals = 18;

uint256 public totalSupply;

mapping(address => uint256) public balanceOf;
```

```
mapping(address => mapping(address => uint256)) public allowance;

uint256 public constant MINIMUM_LIQUIDITY = 1000;

uint112 private reserve0;

uint112 private reserve1;

uint32 private blockTimestampLast;

uint256 private unlocked = 1;

address public factory;

address public token0;

address public token1;

uint256 public price0CumulativeLast;

uint256 public price1CumulativeLast;

uint256 public kLast;

address public feeTo;

address public feeToSetter;

mapping(address => mapping(address => address)) public pairs;

address[] public allPairs;
```

Uswap.route

```
address public factory;

address public wtrx;
```

Functions

Uswap

Line120-125: _mint() – Mints “value” number of tokens and sends it to “to” address (Internal)

Line 127-132: _burn() – Burns “value” number of tokens from “from” address (Internal)

Line 134-138: _approve() – Approves “spender” to spend “value” amount of tokens from “owner” address (Private)

Line 140-145: _transfer() – Transfers “value” amount of token from address “from” to address “to” (Private)

Line 147-151: approve() – Caller function of _approve() (External)

Line 153-157: transfer() – Caller function of _transfer() (External)

Line 159-167: transferFrom() – Transfers “value” amount of token from one another account to “to” account according to the allowance (External)

Line 191-197: lock() – Modifier which sets access to other functions 199-201: constructor() – Sets factory as contract creator address (Public)

Line 203-206: _safeTransfer() – Calls an encoded function from an already deployed contract at address “token” (Private)

Line 208-224: _update() – Updates “price0CumulativeLast” and “price1CumulativeLast” as sum of corresponding variables with encoded value of “_reserve[x]” and high precision division is done on (“_reserve[x]” * “timeElasped”) (Private)

Line 226-246: _mintFee() – Sets “feeTo” address by calling “feeTo()” function from factory contract. Checks if “feeTo” not equal to address(0) and stores the Boolean value to “feeOn” (Private)

Line 248-253: initialize() – Initializes “token0” and “token1” for the pair (External)

Line 255-280: mint() – Get reserve values and token balances. Then “amount[x]” is updated as balance[x] - _reserve[x]. (External)

Line 282-307: burn() – Burns token from balance of contract address and updates the total balance. Transfers each token from the pair to the “to” address too (External)

Line 309-339: swap() – Function to exchange token according to the pair. Only allows exchange if liquidity is sufficient. (External)

Line 341-344: skim() – Transfers both tokens from the pair to the address “to” (External)

Line 346-348: sync() – Calls the “_update()” function which will update the balance of tokens in token pair (External)

Line 350-354: getReserves() – View function to get the reserve values and latest timestamp (Public)

Line 368-385: createPair() – Creates a trading pair between “tokenA” and “tokenB” and stores the data in “pair” mapping (External)

Line 387-391: setFeeTo() – Sets the “feeTo” address (External)

Line 393-397: setFeeToSetter() – Sets new “feeToSetter” address (External)

Line 399-401: getPair() – View function to see pair info of “tokenA” and “tokenB” (External)

Line 403-405: allPairsLength() – View function to retrieve total number of pairs created

Uswap.route

(Description of functions in standard libraries are skipped)

Line 189-216: _addLiquidity() – Function to add liquidity for a certain token pair. If the pair doesn’t exist, function automatically creates it.

Line 218-228: _swap() – Swap function which swaps tokens according to their availability in pool

Line 230-239: addLiquidity() – Caller function for _addLiquidity().

Line 241-254: addLiquidityTRX() – Adds TRX liquidity to the pool

Line 256-267: removeLiquidity() – Removes liquidity from the pool

Line 269-275: removeLiquididtyTRX() – Removes TRX from the pool

Line 277-285: swapExactTokensForTokens() – Swaps first token for receiving second in the pair

Line 287-295: swapTokensForExactTokens() – Swaps second token for receiving first in the pair

Line 297-309: swapExactTRXForTokens() – Swaps tron for token (Only accepts exact trx value)

Line 311-324: swapTRXForExactTokens() – Swaps tron for token (Only accepts exact token value)

Line 326-339: swapExactTokensForTRX() – Swaps token for trx (Only accepts exact token value)

Line 341-355: swapTokensForExactTRX() – Swaps token for trx (Only accepts exact trx value)

Line 357-359: getAmountsIn() – Returns input amount

Line 361-363: getAmountsOut() – Returns output amount

Line 365-370: calcPairLiquidity() – Calculates new pair liquidity

Line 372-377: calcPairSwap() – Calculates price impact on swapping

Line 379-388: getPair() – Returns pair info

Line 390-406: getPairs() – Returns a certain number of pairs info

Best Practices

The code has written according to all Secure Development Recommendations. The libraries used inside are all standard and secure. Code is properly organized and indented

Critical Severity

- Line 205 – Invalid eth address included inside a tron contract (uswap)
- Line 80 – Invalid eth address included inside tron contract (uswap.route)

Medium Severity

- Line 309 – No coping mechanism for possible stack too deep error (uswap)
- No domain separator used

Low Severity

- SafeMath operations are used unnecessarily. It can be avoided in places with no chance of overflows

Other Findings

- Line 354,385 – Storing token address in a different variable will help reduce gas fee
- No functions for supporting tokens which uses fee on transfer