CyberCrime Shield

cybercrimeshield.org

# Smart Contract Audit Report

# Uswap

https://tronscan.org/#/contract/TQ4F8Gr1qRKcMva64qYweAJNAVtgfj6ZJd/code

https://tronscan.org/#/contract/THPMGRqM7asytcp7nMu6MBR8X3dV3NiLa8/code

ID:2328118

20.11.2020

# INTRODUCTION

Blockchain platforms, such as Nakamoto's Bitcoin, enable the trade of crypto-currencies between mutually mistrusting parties.

To eliminate the need for trust, Nakomoto designed a peer-to-peer network that enables its peers to agree on the trading transactions.

Vitalik Buterin identified the applicability of decentralized computation beyond trading, and designed the Ethereum blockchain which supports the execution of programs, called smart contracts, written in Turing-complete languages.

Smart contracts have shown to be applicable in many domains including financial industry, public sector and cross-industry.

The increased adoption of smart contracts demands strong security guarantees. Unfortunately, it is challenging to create smart contracts that are free of security bugs.

As a consequence, critical vulnerabilities in smart contracts are discovered and exploited every few months.

In turn, these exploits have led to losses reaching billions worth of USD in the past few years.

It is apparent that effective security checks for smart contracts are strictly needed.

## SMART CONTRACT

**Factory**

https://tronscan.org/#/contract/TQ4F8Gr1qRKcMva64qYweAJNAVtgfj6ZJd/code

**Router**

https://tronscan.org/#/contract/THPMGRqM7asytcp7nMu6MBR8X3dV3NiLa8/code

## 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, it's possible issues and bugs. 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 PRACTICIES

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

No errors or vulnerabilities affecting the described functionality of the contract have been detected. No backdoors or overflows are present in the contract.

## MEDIUM SEVERITY

Line 309 – No coping mechanism for possible stack too deep error (uswap)

• No domain separator used following the time schedule

## LOW SEVERITY

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

## OTHER ANALYSIS

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

# AUDIT SUMMARY

- **The contract have been found to be free of critical security issues.**
- **Contract has well-formed structure.**
- **User input validation is performed.**
- **No overflows are present in the contract.**

## REFERENCES

[1] The DAO Attacked: Code Issue Leads to 60 Million Ether Theft.

[2] Etherdice. Available from: https://etherdice.io/.

[3] King of Ether. Available from: https://github.com/kieranelby/KingOfTheEtherThrone/blob/v0.4.0/contracts/KingOfTheEtherThrone.sol.

[4] King of Ether, Postmortem. Available from: https://www.

kingoftheether.com/postmortem.html.

[5] Reentrancy Woes in Smart Contracts.

[6] theDAO. Available from: https://etherscan.io/address/ 0xbb9bc244d798123fde783fcc1c72d3bb8c189413.

[7] Accidental bug may have frozen $280 million worth of digital coin ether in a cryptocurrency wallet. Available from: https://www.cnbc.com/2017/11/ 08/accidental-bug-may-have-frozen...

[8] Blockchain is empowering the future of insurance. blockchain-is-empowering-the-future-of-insurance/.

[9] ETHLance. Available from: http://ethlance.com/.

[10] An In-Depth Look at the Parity Multisig Bug.

[11] Northern Trust uses blockchain for private equity recordkeeping. Available from: http://www.reuters.com/article/nthern-trust-ibm-blockchain-idUSL1N1G61TX.

[12] Parity Ethereum Client. (2017). Available from: https://github.com/ paritytech/parity.

[13] Security Alert. (2017). Available from: https://paritytech.io/blog/ security-alert.html.

[14] Submarine Sends: IC3's Plan to Clamp Down on ICO Cheats. Available from: https://www.coindesk.com/ submarine-sends-inside-ic3s-plan-to-clamp-...

[15] Ethereum Smart Contract Security Best Practices.. Available from: https://consensys.github.io/smart-contract-best-practices/.

[16] Mythril. Available from: https://github.com/ConsenSys/mythril.

[17] Parity Wallet Library. Available from: https://github.com/ paritytech/parity/blob/4d08e7b0aec46443bf26547b17d10cb302672835/js/src/ contracts/snippets/enhanced-wallet.sol.

[18] Solidity, high-level language for writing smart contracts. Available from: https://solidity.readthedocs.io/en/develop/.

[19] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A Survey of Attacks on Ethereum Smart Contracts (SoK). In Principles of Security and Trust - 6th International Conference, POST. 164–186.

[20] Massimo Bartoletti, Salvatore Carta, Tiziana Cimoli, and Roberto Saia. Dissecting Ponzi schemes on Ethereum: identification, analysis, and impact. CoRR abs/1703.03779

[21] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. Formal Verification of Smart Contracts: Short Paper. In Proceedings of the ACM Workshop on Programming Languages and Analysis for Security (PLAS). 91–96.

[22] Vitalik Buterin. Ethereum: a next generation smart contract and decentralized application platform. Available from: https://github.com/ethereum/wiki/wiki/White-Paper.

[23] https://github.com/ethereum/solidity/issues/

[24] https://github.com/tronprotocol/java-tron/issues