verichains

SECURITY AUDIT OF

PANDEXCHANGE SMART CONTRACT

Public Report

Feb 27, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Feb 27, 2023. We would like to thank the PandExchange for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the PandExchange Smart Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified 3 vulnerable issues in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About PandExchange Smart Contract

PandExchange is a revolution Exchange which is built on BNB Chain.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the PandExchange Smart Contract.

It was conducted on commit `fea24a9821e9d1b566d737e95dc0053454747c56` from git repository *https://github.com/IARD-Solutions/PandExchange*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
`75dee85fac1924425e6e719e5d43b10ab629572945bb414ad99db4452bbf0c04`	`DCApp.sol`

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)

- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The PandExchange Smart Contract was written in `Solidity` language, with the required version to be `^0.8.16`. The source code was written based on OpenZeppelin's library.

2.1.1. The contract

By default, The contract owner is the contract deployer, but he can transfer ownership to another address at any time. The contract owner can `pause`/`unpause` contract, users can only `addNewDCAToUser()` and `executeSingleUserDCA()` when contract is not paused.

This contract has 3 main functions:

- `addNewDCAToUser()`: the user will transfer an amount of tokens to the contract and will perform the first swap. The contract owner will receive a fee corresponding to 0.5% of the total amount of tokens locked. Users can set `_fee5Decimals` to 0 if they do not want others to receive tokens when executing `executeSingleUserDCA()` function.
- `executeSingleUserDCA()`: After the user-set period (period) minus one half-day, the user may submit one DCA request. Others can do it for them and they will get back the amount of token fees that the user has set.
- `deleteUserDCA()`: stop requesting dca and withdraw all tokens (excluding fee for owner).

2.2. Findings

During the audit process, the audit team found 3 vulnerabilities in the given version of PandExchange Smart Contract.

2.2.1. The `tokenInLockedAmount` variable is miscalculated CRITICAL

Calculation `_amountPerOccurrence` = `_amountPerOccurrence` - `ownerFee/_totalOccurrences`, the value of `ownerFee/_totalOccurrences` will be rounded down, thus making the value of `_amountPerOccurrence` larger. Users will swap more than expected, the last swap will fail because there are not enough tokens in contract.

Example:

- user has 999,957 tokens available for 51 round DCA stake, `_totalOccurrences` = 51
- `_amountPerOccurrence` = 999,957 / 51 = 19,607 tokens
- `ownerFee` = 51 * 19,607 * 0.005 = 4,999 tokens
- `_amountPerOccurrence` = 19,607 - 4,999/51 = 19,509 tokens
- `tokenInLockedAmount` = 19,509 * 51 = 994,959

=> totalToken Input = 994,959 + 4,999 = 999,958 > 999,957

In the right way, `tokenInLockedAmount` should be calculated:

`tokenInLockedAmount` = `_totalOccurrences` * `_amountPerOccurrence` – `feeOwner`

= 51 * 19,607 - 4,999 = 994,958

```
function addNewDCAToUser(
        uint256 _period,
        uint256 _totalOccurrences,
        uint256 _amountPerOccurrence,
        address _tokenIn,
        address _tokenOut,
        uint256 _fee5Decimals,
        address[] memory _path,
        address _exchangeRouterAddress
    ) external payable notPaused returns (uint256) {
        ...

        uint256 ownerFee = ((_totalOccurrences *
            _amountPerOccurrence *
            100_000) * 500) / 10_000_000_000;

        _amountPerOccurrence =
            _amountPerOccurrence -
            ownerFee /
            _totalOccurrences;

        mapDCA[msg.sender][timestamp] = UserDCAData(
            _period,
            _totalOccurrences,
            0,
            _amountPerOccurrence,
            _tokenIn,
            _tokenOut,
            _totalOccurrences * _amountPerOccurrence,
            _path,
            _fee5Decimals,
            _exchangeRouterAddress
        );

        emit AddedNewDCA(
            timestamp,
            msg.sender,
            _totalOccurrences,
            _period,
            _amountPerOccurrence,
            _tokenIn,
            _tokenOut,
            _fee5Decimals
        );
```

```
        if (_tokenIn != address(0)) {
            //Require the user to approve the transfer beforehand
            bool success = IERC20(_tokenIn).transferFrom(
                msg.sender,
                address(this),
                _totalOccurrences * _amountPerOccurrence
            );
            require(success, "Error: TokenIn TransferFrom failed");
        } else {
            require(
                msg.value >= _totalOccurrences * _amountPerOccurrence,
                "Error: You did not sent enough BNB"
            );
        }

        //The first occurrence is executed immediately
        executeSingleUserDCA(msg.sender, timestamp);

        //Send the owner fee, a one time fee at the creation of the DCA
        sendFee(owner, _tokenIn, ownerFee);

        return timestamp;
    }
```

RECOMMENDATION

- Should calculate `tokenInLockedAmount` before `_amountPerOccurrence` minus `ownerFee`.
- Should swap all remaining tokens in the last swap.

UPDATES

- *Feb 25, 2023*: This issue has been acknowledged and fixed by the PandExchange team.

2.2.2. False control of slippage MEDIUM

The function `getAmountsOut()` in `executeSingleUserDCA()` calculates the amount to be executed at a block, with no slippage, allowing attackers to perform a sandwich attack to drain funds from the contract.

RECOMMENDATION

Add a slippage percentage variable to `addNewDCAToUser()` function and allow the user to modify it, so that it can be used in `executeSingleUserDCA()` function.

UPDATES

- *Feb 25, 2023*: The PandExchange team has acknowledged this issue.

2.2.3. Users may lose money if they deposit more than the required number of native tokens LOW

```
function addNewDCAToUser(
        uint256 _period,
        uint256 _totalOccurrences,
        uint256 _amountPerOccurrence,
        address _tokenIn,
        address _tokenOut,
        uint256 _fee5Decimals,
        address[] memory _path,
        address _exchangeRouterAddress
    ) external payable notPaused returns (uint256) {
       ...

        if (_tokenIn != address(0)) {
            //Require the user to approve the transfer beforehand
            bool success = IERC20(_tokenIn).transferFrom(
                msg.sender,
                address(this),
                _totalOccurrences * _amountPerOccurrence
            );
            require(success, "Error: TokenIn TransferFrom failed");
        } else {
            require(
                msg.value >= _totalOccurrences * _amountPerOccurrence,
                "Error: You did not sent enough BNB"
            );
        }

        //The first occurrence is executed immediately
        executeSingleUserDCA(msg.sender, timestamp);

        //Send the owner fee, a one time fee at the creation of the DCA
        sendFee(owner, _tokenIn, ownerFee);

        return timestamp;
    }
```

RECOMMENDATION

Change from `>=` to `==`.

UPDATES

- *Feb 25, 2023*: This issue has been acknowledged and fixed by the PandExchange team.

2.3. Additional notes and recommendations

2.3.1. Misspelling in comment INFORMATIVE

- Line: 451
- Wrong word: avoir

RECOMMENDATION

Change from avoir to avoid.

UPDATES

- *Feb 25, 2023*: This issue has been acknowledged and fixed by the PandExchange team.

APPENDIX

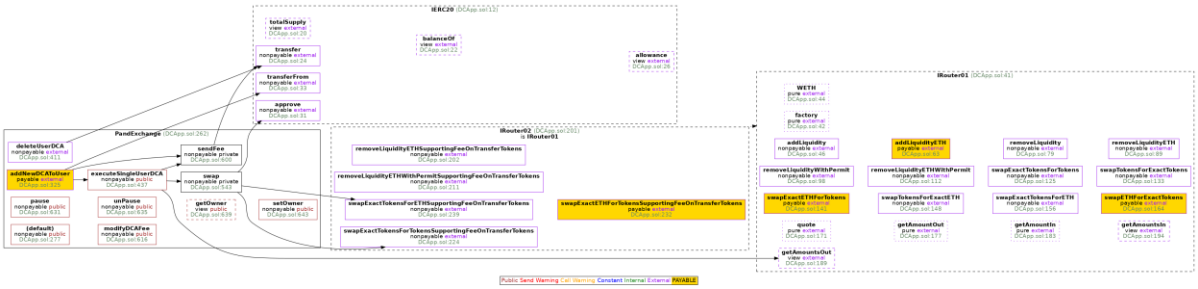

Image 1. PandExchange Smart Contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.1	*Feb 27, 2023*	Public Report	Verichains Lab

Table 2. Report versions history