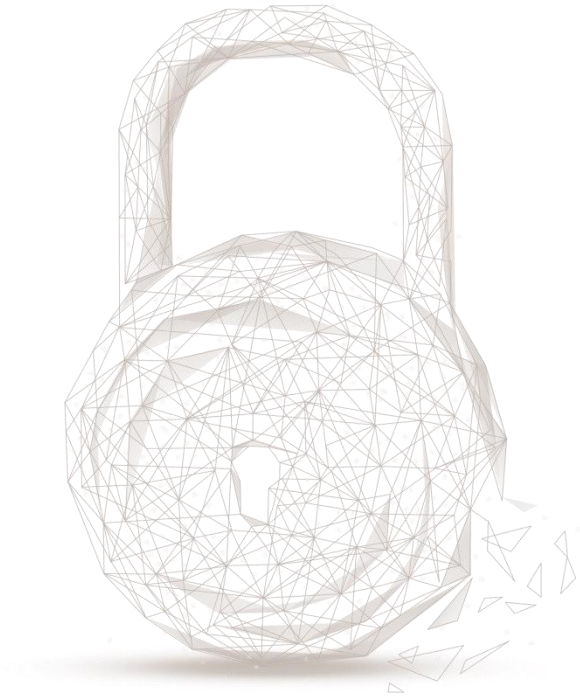
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202103251820

Smart Contract Name：

Orbit-Falco-Contract

Smart Contract Address：

0xfd24db627893fc1e2e515aef76b228ca697dc531

Smart Contract Address Link：

https://cn.etherscan.com/address/0xfd24db627893fc1e2e515aef76b228ca697dc531#code

Start Date：2021.01.28

Completion Date：2021.03.25

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass

		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts project

FundManageStorage, including Coding Standards, Security, and Business Logic. **The FundManageStorage project passed all audit items. The overall result is Pass.** The smart contract is able to function properly.

Audit Contents:

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract. In this project, the contract
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

(1) takePosition function

- Description: As shown in the figure below, The contract implements the *takePosition* function for users to call to manage contract investment to build positions. After the interaction with the management contract is completed, the user calls the position building function *takePosition* of the asset management contract to build the position. (If *lockUp*=false and *openUp*=false, the position can be built). Each address can only participate in opening a position once, and the ETH transferred from the address must be greater than the handling fee, and the handling fee cannot be zero. When entering a position, deduct the handling fee and pay it to the handling fee management contract (FUND_CONTRACT), record the total amount of ETH (excluding handling fee), and record the amount of the current creator's opening principal.

```
function takePosition(
    address _owner,
    uint _charge
) public payable isAdmin lockAllowed(_owner) returns (bool) {
    require(FUND_CONTRACT != address(0), "The address cannot be!");
    require(_charge > 0, "The fee must be greater than 0!");
    require(msg.value > _charge, "Building warehouse amount must be greater than poundage!");
    require(!_isTakePosition[_owner], "Non-repeatable operation!");

    _isTakePosition[_owner] = true;

    if (CostHand(FUND_CONTRACT).payableCharge{value:_charge}(_owner)) {
        uint _capital = msg.value.sub(_charge);
        ETHCapitalPool = ETHCapitalPool.add(_capital);

        _balance[_owner] = _balance[_owner].add(_capital);
    } else {
        revert();
    }
    return true;
}
```

Figure 1 source code of *takePosition*

- Related functions: *takePosition, payableCharge*
- Result: Pass

(2) coverToken function

- Description: As shown in the figure below, the contract implements the *coverToken* function for users to call the management contract to exchange tokens. After the user completes the interaction with the management contract, call the *coverToken* function to exchange the corresponding token. Exchange with the token exchange contract (FETCH_ADDRESS) based on the current contract ETH balance, recurring exchange and record (accumulate) the corresponding amount in the initial exchange fund pool *TokenCapitalPool*, call the callback method of the token exchange contract (FETCH_ADDRESS) to trigger the exchange action, The exchange will return the exchanged quantity, and compare the returned value with the externally obtained exchange rate. The error shall not exceed the set deviation value, the default is 10‰, and then verify whether the account is received. After the exchange is completed, the external exchange rate will be used to calculate the contract to obtain the The amount of USDC worth of WETH exchanged in this time is accumulated and recorded *capitalPool*.

```
function coverToken(string[] memory _tokenName, uint[] memory _ratio) external isAdmin returns (bool) {
    require(_tokenName.length == _ratio.length, "Token and proportional array sizes are inconsistent!");
    uint _capital = address(this).balance;

    uint _ethCoverTotal;
    for (uint i = 0; i < _tokenName.length; i++) {
        require(_ERC20Address[_tokenName[i]] != address(0), "Token address does not exist!");
        uint _coverNum = _capital.mul(_ratio[i]).div(100);
        _ethCoverTotal = _ethCoverTotal.add(_coverNum);
        uint _return_num = ETH2Token(_tokenName[i], _coverNum);
        TokenCapitalPool[_tokenName[i]] = TokenCapitalPool[_tokenName[i]].add(_return_num);
    }

    uint _cost = Exchange(_exchangeAddress).getAmountsOut(address(this).balance, _ERC20Address["WETH"], _ERC20Address["USDC"]);
    capitalPool = capitalPool.add(_cost);

    return true;
}
```

Figure 2 source code of *coverToken*

```
    function ETH2Token(
        string memory _return_symbol,
        uint _tokenNum
    ) internal returns (uint) {
        address _return_address = _ERC20Address[_return_symbol];
        require(_return_address != address(0), "The return token address does not exist!");

        uint _return_balance = getBalance(_return_symbol, address(this));

        uint _exchange = Exchange(_exchangeAddress).getAmountsOut(_tokenNum, _ERC20Address["WETH"], _return_address);

        uint _return_num = TransferCallback(FETCH_ADDRESS).fundETHCallback{value:_tokenNum}(_return_address);
        require(_return_num.add(_return_num.mul(_deviation).div(1000)) >= _exchange, "Excessive exchange rate misalignment!");
        require (_return_balance.add(_return_num) <= getBalance(_return_symbol, address(this)), "Not yet received the token!");
        return _return_num;
    }
}
```

Figure 3 source code of *ETH2Token*

- Related functions: *coverToken*, *ETH2Token*, *getAmountsOut*
- Result: Pass

(3) enableOpenUp function

- Description: As shown in the figure below, The contract implements the *enableOpenUp* function for users to call the management contract to calculate the final funds and pay commissions. After completing the interaction with the management contract, it is determined that the commission address is not a zero address, and the user can only open a position once and set *openUp* to true. If *_totalAsset* is passed in (The project party declares that the interactive contract obtains the *_totalAsset*, in a trusted manner at the opening of the position and then performs this action after being judged) is greater than the initial number of assets (*capitalPool*), then the number of tokens (*_currentSymbol*) of the value profit part (10% by default) will be deducted as a commission, and the commission Enter the commission management contract (BROKERAGE_CONTRACT), and finally record the number of *_currentSymbol* tokens that do not include the commission part for use in the calculation of the withdrawal operation.

```
function enableLockUp() public isAdmin returns (bool){
    require(!lockUp, "Non-repeatable operation!");
    lockUp = true;
    require(TokenManage(_tokenManageContract).closePositionRelease(), "Token release failed!");
    return true;
}

function enableOpenUp(uint _totalAsset) public isAdmin returns (bool) {
    require(BROKERAGE_CONTRACT != address(0), "The address cannot be 0!");
    require(!openUp, "Non-repeatable operation!");
    openUp = true;
    uint _currentBalance = getBalance(_currentSymbol, address(this));
    if (_totalAsset > capitalPool) {
        uint _rateUsd = _totalAsset.sub(capitalPool).mul(brokerageRate).div(100);
        uint _rateWbtc = _currentBalance.mul(_rateUsd).div(_totalAsset);
        require(_transfer(BROKERAGE_CONTRACT, _rateWbtc, _ERC20Address[_currentSymbol]), "Failed handling fee transfer out!");
        _currentBalance = _currentBalance.sub(_rateWbtc);
    }
    finalTokenNum = _currentBalance;
    return true;
}
```

Figure 4 source code of *enableOpenUp*

```
    function _transfer(
        address _to,
        uint _value,
        address _erc20Addr
    ) internal returns (bool) {
        require(IERC20(_erc20Addr).balanceOf(address(this)) >= _value, "Transfer out more than the maximum amount!");
        return IERC20(_erc20Addr).transfer(_to, _value);
    }
```

Figure 5 source code of *_transfer*

- Related functions: *enableOpenUp*, *_transfer*, *balanceOf*, *transfer*
- Result: Pass

(4) enableLockUp function

- Description: As shown in the figure below, the contract implements the *enableLockUp* function to lock the position and call the token management contract to release the governance currency. This function can only be called once by the admin. If the release of the governance currency fails, an exception will be thrown.

```
function enableLockUp() public isAdmin returns (bool){
    require(!lockUp, "Non-repeatable operation!");
    lockUp = true;
    require(TokenManage(_tokenManageContract).closePositionRelease(), "Token release failed!");
    return true;
}
```

Figure 6 source code of *enableLockUp*

```
interface TokenManage {
    function closePositionRelease() external returns (bool);
}
```

Figure 7 source code of *closePositionRelease*

- Related functions: *enableLockUp, closePositionRelease*
- Result: Pass

(5) withdrawToken function

- Description: As shown in the figure below, the contract implements the *withdrawToken* function for the user to call the management contract to withdraw. After the interaction with the management contract is completed, the number of tokens held by the address (VOUCHER_ADDRESS) and the total number of vouchers are issued, and the user address held by the user is destroyed For some voucher tokens, the destruction address is *_destroyAddress*. Calculate the proportion of the number of certificates held to the total number of certificate tokens issued on the entire network, and then withdraw cash to the _to address by calling _transfer. Withdrawal must be executed in the open position (*openUp*=true and *lockUp*=true). The project party declares: The generation of the voucher token is determined when the position is opened according to the number of positions opened. After the position is closed, the position cannot be opened. Therefore, there will be no new generation of this token after the position is closed or when the position is opened and withdrawn. The total amount of tokens will not change.

```
modifier openAllowed(address _addr) {
    require(openUp && lockUp, "Do not operate this item!");
    _;
}
```

Figure 8 source code of *openAllowed*(modifier)

```
function withdrawToken(
    address _to
) public openAllowed(_to) isAdmin returns (bool) {
    IERC20 _voucherContract = IERC20(VOUCHER_ADDRESS);
    uint haveTokenNum = _voucherContract.balanceOf(_to);
    require(haveTokenNum > 0, "No assets!");
    require(_voucherContract.transferFrom(_to, _destroyAddress, haveTokenNum), "Please approve!");

    uint totalTokenNum = _voucherContract.totalSupply();

    uint _returnNum = finalTokenNum.mul(haveTokenNum).div(totalTokenNum);
    if (_transfer(_to, _returnNum, _ERC20Address[_currentSymbol])) {
        return true;
    } else {
        revert();
    }
}
```

Figure 9 source code of *withdrawToken*

- Related functions: *withdrawToken, balanceOf, transferFrom, totalSupply, _transfer*
- Result: Pass

(6) fetchToken function

- Description: As shown in the figure below, the contract implements the *fetchToken* function for borrowing token exchange. Only the FETCH_ADDRESS address can call the *fetchToken* function, and the position is closed (*lockUp*=true) and not opened (*openUp*=false). After the interaction with the fund borrowing contract is completed, the amount of borrowed tokens is transferred to the borrowing contract, and the callback method of the fund borrowing contract (FETCH_ADDRESS) is called to trigger the exchange action. After the exchange is completed, the exchanged amount is returned based on the returned value and the exchange rate obtained from the outside. For comparison, the error should not exceed the set deviation value, the default is 10‰, and then verify whether it is received.

```
function fetchToken(
    string memory _symbol,
    string memory _return_symbol,
    uint _tokenNum,
    uint _queryId
) external fetchPermission returns (bool) {
    require(!openUp && lockUp, "Do not operate this item!");

    address _fetch_address = _ERC20Address[_symbol];
    address _return_address = _ERC20Address[_return_symbol];

    require(_fetch_address != address(0), "The extract token address does not exist!");
    require(_return_address != address(0), "The return token address does not exist!");
    require(getBalance(_symbol, address(this)) >= _tokenNum, "Insufficient account balance!");

    uint _return_balance = getBalance(_return_symbol, address(this));

    if (_transfer(msg.sender, _tokenNum, _fetch_address)) {
        uint _exchange = Exchange(_exchangeAddress).getAmountsOut(_tokenNum, _fetch_address, _return_address);
        uint _return_num = TransferCallback(msg.sender).fundCallback(_fetch_address, _return_address, _tokenNum, _queryId);
        require(_return_num.add(_return_num.mul(_deviation).div(1000)) >= _exchange, "Excessive exchange rate misalignment!");

        if (_return_balance.add(_return_num) <= getBalance(_return_symbol, address(this))) {
            return true;
        } else {
            revert();
        }
    }
    return false;
}
```

Figure 10 source code of *fetchToken*

- Related functions: *getBalance, getAmountsOut, safeTransfer*
- Result: Pass

(7) Contract parameter settings

- Description: The functions that only the administrator can call are the following functions *changeOwner*, *setAdmin*, *setBrokerageContract*, *setExchangeAddress*, *setDeviation*, *setTokenManageContract*, *setFundContract*, *setFetchAddress*, *setCurrentSymbol*, *setContractAddress* to configure the contract. The *changeOwner* function is used to transfer the administrator authority; the *setAdmin* function is used to add management contracts; the *setBrokerageContract* function is used to set the commission address; the *setDeviation* function is used to set the exchange rate; the *setExchangeAddress* function is used to set the exchange rate to obtain the contract address; *setTokenManageContract* function sets the Token management contract address; *setFundContract* sets the address of the fee contract; *setFetchAddress* function is used to set the address of the account that can borrow funds; *setCurrentSymbol* function is used to set the token symbol currently used.
- Related functions: *changeOwner*, *setAdmin*, *setBrokerageContract*, *setExchangeAddress*, *setDeviation*, *setTokenManageContract*, *setFundContract*, *setFetchAddress*, *setCurrentSymbol*, *setContractAddress*
- Result: Pass

(8) setBrokerageRate function

- Description: As shown in the figure below, This function is called by the management contract to set the commission ratio, and the commission is greater than zero.

```
function setBrokerageRate(uint _rate) external isAdmin {
    require(_rate > 0, "The interest rate has to be greater than 0!");
    brokerageRate = _rate;
}
```

Figure 11 source code of *setBrokerageRate*

- Related functions: *notifyRewardAmount*
- Safety advice: It is recommended to limit the upper limit of the commission ratio range.
- Modification results: The project party confirms that the value is controlled by the Admin contract and is ignored here.
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project FundManageStorage. The problems found by the audit team during the audit process have been notified to the project party and fixed, the overall audit result of the FundManageStorage project's smart contract is **Pass**. Note: The current contract has no unrepaired security issues caused by the contract's own code. However, there are contract interactions between the current contract and multiple external contracts, and user need to pay attention to the potential security risks of external contract interactions.

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com