BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number：202106041046

Report Query Name：Orbit-Swift-Contract

Audit Contract Source Code:

Orbit-Swift-Contract.sol

SHA256 HASH:

62EA5826AF166C80DA27F661AC68CB636A19EA2CE74030243A64A3D2EF03F4AD

Start Date：2021.05.31

Completion Date：2021.06.04

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

Disclaimer: This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project. 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 contract Orbit-Swift-Contract, including Coding Standards, Security, and Business Logic. **The Orbit-Swift-Contract contract passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

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.
- 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

3.1 Business analysis of contract Orbit-Swift-Contract

(1) open positions

- Description: The contract implements *takePosition* function for opening positions. This function can only be called by the address of the DISPATCH_ROLE role. When calling this function, caller need to transfer ETH to the contract, and the amount of transferred ETH cannot be 0, and then trigger the *PositionLog* event.

```
function takePosition(address _account, uint _usdcNum, bytes32 _id) external payable roleCheck(DISPATCH_ROLE) returns (bool) {
    require(msg.value > 0, "ETH_ZERO");
    uint _timestamp = block.timestamp;
    emit PositionLog(_id, _account, msg.value, _usdcNum, _timestamp);
    return true;
}
```

Figure 1 Source code of function *takePosition*

- Related functions: *takePosition*
- Result: Pass

(2) exit positions

- Description: The contract implements *outOfPosition* function for exiting positions. This function can only be called by the address of the DISPATCH_ROLE role. When calling this function, the caller can send the assets of multiple exiting users at the same time. During this process, the corresponding brokerage fee and charge fee will be charged, the brokerage fee will be sent to the brokerageAddr address, and the charge fee will be sent to the chargeAddr address.

```
function outOfPosition(
    address[] memory _addr,
    address tokenAddr,
    uint[] memory tokenNum,
    uint[] memory billTokenNum,
    uint[] memory tokenBrokerageNum,
    uint[] memory tokenChargeNum
) external roleCheck(DISPATCH_ROLE) returns (uint _tokenRes) {
    require(ERC20Address.contains(tokenAddr), "ERC20_NOT_FUND");
    require(chargeAddr != address(0), "CHARGE_ADDRESS_ZERO");

    uint _len = tokenNum.length;
    uint _charge;
    uint _brokerage;
    for (uint i = 0; i < _len; i++) {
        _outOfPosition(
            _addr[i],
            tokenAddr,
            billTokenNum[i],
            tokenNum[i],
            tokenChargeNum[i],
            tokenBrokerageNum[i]
        );
        _tokenRes = _tokenRes.add(tokenNum[i]);
        _charge = _charge.add(tokenChargeNum[i]);
        _brokerage = _brokerage.add(tokenBrokerageNum[i]);
    }
    chargeGrant(chargeAddr, tokenAddr, _charge);
    chargeGrant(brokerageAddr, tokenAddr, _brokerage);
    _tokenRes = _tokenRes.add(_charge).add(_brokerage);
}
```

Figure 2 Source code of function *outOfPosition*

- Related functions: *outOfPosition*
- Audit description: The *_transferToken* function calls the *transfer* function corresponding to ERC20 token. If the function corresponding to ERC20 token declares no return value (such as USDT contract), the transaction will throw an exception.

```
function _transferToken(
    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 3 Source code of function *_transferToken*

- Response from the project party: Currently, only WBTC and USDC tokens are supported, and the *transfer* functions of these two tokens both have return bool.

- Result: Pass

(3) Exchange ETH for tokens

- Description: The contract implements the *fetchETH2Token* function to convert the ETH of this contract to the specified token. This function can only be called by the EXCHANGE_ROLE role address (which is the FundExchange contract). When calling this function, the following operations will be performed in sequence: 1. Check whether the corresponding token is supported; 2. Call the *getAmountsOut* function of the UniswapV2Router02 contract to query the number of exchangeable tokens; 3. Call the *fundETH2TokenCallback* function of the external contract FundExchange, and carry it with the ETH to exchange, convert the ETH in this contract to the designated token; 4. Check whether the slippage between the number of exchanged tokens and the quantity queried in step 2 exceeds _deviation. If the slippage is too large, the transaction will be rolled back; 5. Check whether the actual number of tokens in the contract is not less than the number of tokens before the exchange plus the number of exchanged tokens (therefore, this contract does not support deflationary tokens).

```
function fetchETH2Token(
    address _return_address,
    uint _tokenNum,
    uint _queryId
) external roleCheck(EXCHANGE_ROLE) returns (uint) {
    require(ERC20Address.contains(_return_address), "RETURN_ADDRESS_NOT_FUND");

    require(address(this).balance >= _tokenNum, "INSUFFICIENT_BALANCE");
    IERC20 returnContract = IERC20(_return_address);
    uint _return_balance = returnContract.balanceOf(address(this));

    uint _exchange = Exchange(_exchangeAddress).getAmountsOut(_tokenNum, WETH, _return_address);
    uint _return_num = FundExchange(msg.sender).fundETH2TokenCallback{value:_tokenNum}(_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) <= returnContract.balanceOf(address(this))) {
        emit ExchangeLog(address(0), _return_address, _tokenNum, _return_num, block.timestamp);
        return _return_num;
    } else {
        revert();
    }
}
```

Figure 4 Source code of function *fetchETH2Token*

- Related functions: *fetchETH2Token, getAmountsOut, fundETH2TokenCallback*

- Result: Pass

(4) Exchange tokens for ETH

- Description: The contract implements the *fetchToken2ETH* function to convert the specified number of tokens into ETH in this contract. This function can only be called by the EXCHANGE_ROLE role address (which is the FundExchange contract). When calling this function, the following operations will be performed in sequence: 1. Check whether the corresponding token is supported; 2. Send the

corresponding amount of tokens to the FundExchange contract; 3. Call the *getAmountsOut* function of the UniswapV2Router02 contract to query the amount of convertible ETH; 4. Call the *fundToken2ETHCallback* function of the external contract FundExchange to convert the specified number of tokens into ETH; 5. Check whether the slippage between the amount of ETH exchanged and the amount queried in step 3 exceeds _deviation. If the slippage is too large, it will roll back the transaction; 6. Check whether the actual ETH quantity of the contract is not less than the ETH balance before the exchange plus the exchanged ETH quantity.

```
function fetchToken2ETH(
    address _fetch_address,
    uint _tokenNum,
    uint _queryId
) external roleCheck(EXCHANGE_ROLE) returns (uint) {
    require(ERC20Address.contains(_fetch_address), "FETCH_ADDRESS_NOT_FUND");

    IERC20 fetchContract = IERC20(_fetch_address);
    require(fetchContract.balanceOf(address(this)) >= _tokenNum, "INSUFFICIENT_BALANCE");

    if (_transferToken(msg.sender, _tokenNum, _fetch_address)) {
        uint _return_balance = address(this).balance;
        uint _exchange = Exchange(_exchangeAddress).getAmountsOut(_tokenNum, _fetch_address, WETH);
        uint _return_num = FundExchange(msg.sender).fundToken2ETHCallback(_fetch_address, _tokenNum, _queryId);
        require(_return_num.add(_return_num.mul(_deviation).div(1000)) >= _exchange, "Excessive exchange rate misalignment!");

        if (_return_balance.add(_return_num) <= address(this).balance) {
            emit ExchangeLog(_fetch_address, address(0), _tokenNum, _return_num, block.timestamp);
            return _return_num;
        }
    }
    revert();
}
```

Figure 5 Source code of *fetchToken2ETH*

- Related functions: *fetchToken2ETH, getAmountsOut, fundToken2ETHCallback*
- Result: Pass

(5) Exchange tokens for tokens

- Description: The contract implements the *fetchToken2Token* function to exchange the specified number of tokens in this contract into another token. This function can only be called by the EXCHANGE_ROLE role address (which is the FundExchange contract). When calling this function, the following operations will be performed in sequence: 1. Check whether the corresponding two tokens are supported; 2. Send the corresponding amount of tokens to the FundExchange contract; 3. Call the *getAmountsOut* function of the UniswapV2Router02 contract to query the exchangeable tokens quantity; 4. Call the *fundToken2TokenCallback* function of the external contract FundExchange to exchange the specified number of tokens for another type of token; 5. Check whether the slippage between the number of exchanged tokens and the number queried in step 3 exceeds _deviation, if slippage If the point is too large, the transaction will be rolled back; 6. Check whether the actual number of tokens in the contract is not less than the token balance before the exchange plus the number of exchanged tokens.

```
function fetchToken2Token(
    address _fetch_address,
    address _return_address,
    uint _tokenNum,
    uint _queryId
) external roleCheck(EXCHANGE_ROLE) returns (uint) {
    require(ERC20Address.contains(_fetch_address), "FETCH_ADDRESS_NOT_FUND");
    require(ERC20Address.contains(_return_address), "RETURN_ADDRESS_NOT_FUND");

    IERC20 fetchContract = IERC20(_fetch_address);
    require(fetchContract.balanceOf(address(this)) >= _tokenNum, "INSUFFICIENT_BALANCE");

    if (_transferToken(msg.sender, _tokenNum, _fetch_address)) {
        IERC20 returnContract = IERC20(_return_address);
        uint _return_balance = returnContract.balanceOf(address(this));

        uint _exchange = Exchange(_exchangeAddress).getAmountsOut(_tokenNum, _fetch_address, _return_address);
        uint _return_num = FundExchange(msg.sender).fundToken2TokenCallback(_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) <= returnContract.balanceOf(address(this))) {
            emit ExchangeLog(_fetch_address, _return_address, _tokenNum, _return_num, block.timestamp);
            return _return_num;
        }
    }
    revert();
}
```

Figure 6 Source code of *fetchToken2Token*

- Related functions: *fetchToken2Token, getAmountsOut, fundToken2TokenCallback*
- Result: Pass

(6) Permissions management

- Description: The contract has 4 permissions, DEFAULT_ADMIN_ROLE, ADMIN_ROLE, DISPATCH_ROLE, and EXCHANGE_ROLE. DEFAULT_ADMIN_ROLE is the default super administrator permission, which is used to manage all permissions; ADMIN_ROLE is the super administrator permission of the contract, used to configure contract parameters, including WETH contract address, USDC contract address, slippage ratio _deviation, _exchangeAddress contract address, brokerage fee address brokerageAddr and charge fee address chargeAddr; DISPATCH_ROLE is the asset scheduling permission, used to open positions and withdraw assets; EXCHANGE_ROLE is the fund exchange permission, used for ETH/Token exchange.

- Related functions: *grantRole, revokeRole, renounceRole, setWETH, setUSDC, setDeviation, setExchangeAddress, setBrokerageAddress, setChargeAddress, addFundERC20, removeFundERC20*

- Result: Pass

(7) Other audit notes

- Description: If an attack contract uses a flash loan to lend a large amount of ETH, and then converts ETH to USDC in the ETH-USDC pair, the price of USDC rises and the price of ETH falls. If the *fetchETH2Token* function of this contract is called and ETH is exchanged to USDC at this time, the price of USDC will rise further. And then, part of USDC will be exchanged for ETH to repay the flash

loan, and part of USDC will be left as profit. However, to perform this operation, the attack contract needs to have EXCHANGE_ROLE permission.

- Response from the project party: 1. The EXCHANGE_ROLE permission can only be granted by the super administrator. The *fetchETH2Token/fetchToken2ETH/fetchToken2Token* function called under the chain is executed in a trusted environment, and the executed private key is again protected by asymmetric encryption, which means that if attackers want to call in this method, first obtain the private key with this permission, and also need to decrypt the private key. 2. There is no internal evil in the corresponding operation.
- Related functions: *fetchETH2Token, fetchToken2ETH, fetchToken2Token*
- Result: Pass

4. Conclusion

Beosin(Chengdu LianAn) conducted a detailed audit on the design and code implementation of the smart contract Orbit-Swift-Contract. The contract Orbit-Swift-Contract passed all audit items, The overall audit result is **Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com