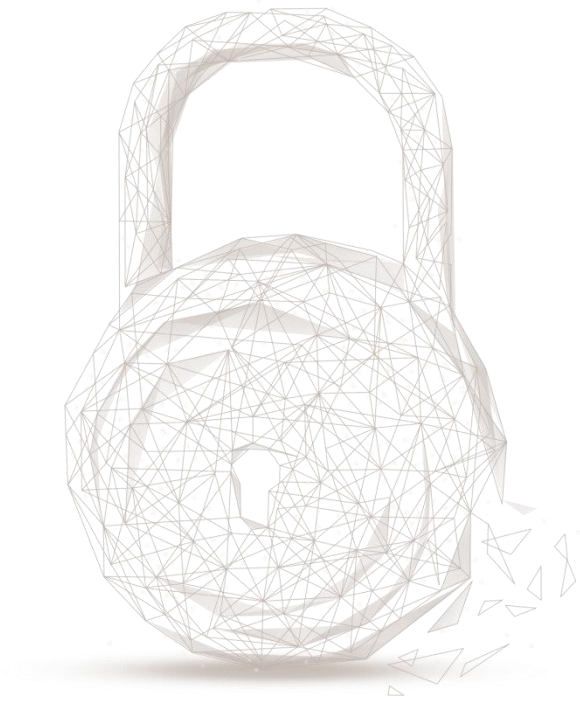
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202009062030

Project Name：

OneSwap

Project Link：

https://github.com/oneswap/oneswap_contract_ethereum

Commit Hash：

Start version: 49b5c8d0392e828b735445980e364d5ddc1c8542

Final version: 300261dabed260bf48cdffcc96f3c3c0293c5511

Start Date：2020.09.01

Completion Date：2020.09.06

Overall Result：Pass (Distinction)

Audit Team: Beosin 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 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 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 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 Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin. Due to the technical limitations of any organization, this report conducted by Beosin still has the possibility that the entire risk cannot be completely detected. Beosin disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin.

Audit Results Explained:

Beosin Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of OneSwap project, including Coding Standards, Security, and Business Logic. **The OneSwap project passed all audit items. The overall result is Pass (Distinction). The project 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 the weather 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

Check whether the business is secure.

3.1 Match engine and orderbook

- Description: The OneSwapPair contract implements orderbook and match engine, enabling automated match engine and on-chain match. The contract provides the OneSwapRouter contract with *addLimitOrder* for adding limit orders, and *addMarketOrder* for adding market order. Users can call *removeOrder* to cancel a single order, or *removeOrders* can be called to cancel multiple orders in orderbook. In addition, the contract provides a corresponding liquidity pool and orderbook status query interface for external application to query data on the chain.
- Related functions: *addLimitOrder, addMarketOrder, removeOrder, removeOrders, calcStockAndMoney, calcStockAndMoney, getOrderList, getPrices,* etc.
- Result: Pass

3.2 Router

- Description: The OneSwapRouter contract implements the initiation of limit orders, market orders, and the ability to add and remove liquidity. It also supports market orders across trading pools. User can call the *swapToken* method for single-trade pool or cross-trade pool and call *limitOrder* to initiate a limitOrder. User can call *addLiquidity* to add liquidity into the trading pool (or create the pair if it does not exist), or call *removeLiquidity* to remove the liquidity in the trading pool.
- Related functions: *limitOrder, swapToken, addLiquidity, removeLiquidity*
- Result: Pass

3.3 Token management

- Description:The token of contract can be transferred, batch transferred, approved, delegate transferred and destroyed, but cannot be mint. The contract has an owner address. The owner can set the blacklist and change the owner. The tokens cannot be transferred/delegate transferred into or out blacklist address.
- Related functions: *transfer, transferFrom, multiTransfer, approve, burn, burnFrom, addBlackLists, removeBlackLists, changeOwner, updateOwner*
- Result: Pass

3.4 OneSwapBuyback

- Description: The OneSwapBuyback contract implements a buyback. The OneSwap-Liquidity-Share tokens generated from all liquidity addition and withdrawals from the trading pool will be entered into the OneSwapBuyback contract. Anyone may use the *Removeliqudesert* method to convert the "oneswap-Liquidity-Share" token into the token in the corresponding transaction pool. In addition, the contract defines the mainToken array to hold the supported token type, and anyone can call the *swapForMainToken* method to convert the token in the contract to the supported token type in the corresponding transaction pool. Anyone can call the *swapForOnesAndBurn* method to convert the tokens supported in the contract to ones through the corresponding trade pool and destroy them. The owner of the OneSwapToken contract can change the type of token supported.
- Related functions: *addMainToken, removeMainToken, removeLiquidity, swapForMainToken, swapForOnesAndBurn*
- Result: Pass

3.5 Governance

- Description: The OneSwapGov contract implements the related functions. The owner of the OneSwapToken contract can call the functions *submitFundsProposal*, *submitParamProposal*, and *submitUpgradeProposal* to initiate the proposal of using governance ones balance, change transaction pool fee and change transaction logic address. Any address who holden over 1% of total supply can call the *submitTextProposal* function to submit text proposal, but it is required to do after one day of the end of last proposal. Each proposal lasts for three days, and only one proposal can exist at the same time. Any user can express "agree" or "object" to the proposal by calling the *vote* method and locking ones in OneSwapGov (votes can be changed before the proposal ends). As of the end of the voting, more options with ones are the results of the proposal. If the proposal fails, the proposal initiator will be deducted and destroyed. If the proposal passes, anyone can call the *tally* method to execute the proposal. Users can call *withdrawOnes* to withdraw the locked ones when they are not voting for the current proposal.

- Related functions: *submitFundsProposal, submitParamProposal, submitUpgradeProposal, submitTextProposal, vote, tally, withdrawOnes*

- Result: Pass

3.6 LockSend

- Description: The LockSend contract implements the lock-up function. Any user can call the *lockSend* method to send the caller's specified number of ERC20 tokens to the LockSend contract for lock-up. After the lock-up period expires, any user can call the *unlock* method to unlock the lock and send the locked-up tokens to the beneficiary specified at the time of lock-up.

- Related functions: *lockSend, unlock*

- Result: Pass

3.7 SupervisedSend

- Description: The SupervisedSend contract implements the C2C function, and realizes the legal currency transaction through the supervisor mechanism. Any user can call *supervisedSend* to lock ERC20 tokens to this contract and specify the recipient and supervisor. After the lock-up period expires, any user can call *supervisedUnlockSend* to unlock the lock, and send the tokens and rewards to the receiver and supervisor respectively. When the lock-up period has not expired, the supervisor can call *earlyUnlockBySupervisor* to send the locked token to the locker or receiver; the locker can call *earlyUnlockBySender* to send the locked token to the receiver in advance.

- Related function: *supervisedSend, supervisedUnlockSend, earlyUnlockBySupervisor, earlyUnlockBySender*

- Result: Pass

3.8 gas consumption

- Description: The OneSwap project uses a variety of technologies in the code to save gas consumption during the transaction, including data compression, code execution process optimization, etc. The following table uses the Uniswap project as a comparison, and its gas consumption is counted.

- Related functions: Whole project

- Result： Pass

OneSwap Gas Consumption Evaluation		
Operation	**Uniswap(Wei)**	**OneSwap(Wei)**
Create erc20 pair and add liquidity	2183256	497775
Only add liquidity in erc20 pair	142249	142249
Create eth pair and add liquidity	2175105	419071
Only add liquidity in eth pair	168226	115343
Remove liquidity in erc20 pair	123235	109890
Remove liquidity in eth pair	160722	97054
Swap erc20 pair once gas used	127682	121796
Swap erc20 pair fourth gas used	344991	224053

Table 1 OneSwap Gas Consumption Evaluation

4. Audit Detail

4.1 multiTransfer Error

Description

The *multiTransfer* function of contracts/OneSwapToken.sol is used for batch transfer. This function compresses each transfer information into a uint256 variable, as shown in the figure below. However, in the process of extracting the value from the parameter, the code performs an & operation between the transfer parameter and 0xffffffffffff. This operation only takes out the lowest 48 bits (0xffffffffffff is 2**48-1), which does not match the expected transfer amount.

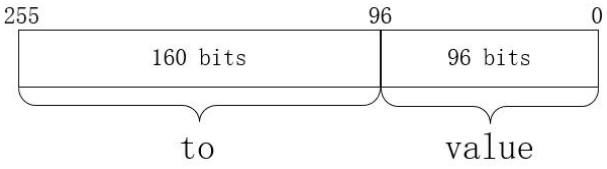


Figure 1 Transfer information

```
93    function multiTransfer(uint256[] calldata mixedAddrVal) public override returns (bool) {
94        for (uint i = 0; i < mixedAddrVal.length; i++) {
95            address to = address(mixedAddrVal[i]>>96);
96            uint256 value = mixedAddrVal[i]&0xffffffffffff;
97            _transfer(msg.sender,to,value);
98        }
99        return true;
100   }
```

Figure 2 related bug

Recommendation

Modify the corresponding code to correct this issue.

Result

Fixed. The fixed code is shown in the Figure 3 below.

```
92    function multiTransfer(uint256[] calldata mixedAddrVal) public override returns (bool) {
93        for (uint i = 0; i < mixedAddrVal.length; i++) {
94            address to = address(mixedAddrVal[i]>>96);
95            uint256 value = mixedAddrVal[i]&(2**96-1);
96            _transfer(msg.sender, to, value);
97        }
98        return true;
99    }
```

Figure 3 fixed code

4.2 calcStockAndMoney function does not add pure keyword

Description

The *calcStockAndMoney* function of contracts/OneSwapPair.sol is used to query calculate the 'stockAmount' and'moneyAmount' to be traded when given the 'amount' of stock and decimal floating point price'price32'. However, the function does not add the pure keyword, nor the view keyword, which causes a signature transaction to be initiated every time the function is called. Does not match the expected function of the function.

```
775     function calcStockAndMoney(uint64 amount, uint32 price32) external override returns (uint stockAmount, uint moneyAmount) {
776         uint[5] memory proxyData;
777         ProxyData.fill(proxyData, 4+32*(ProxyData.COUNT+2));
778         (stockAmount, moneyAmount, ) = _calcStockAndMoney(amount, price32, proxyData);
779     }
```

Figure 4 calcStockAndMoney source code

Recommendation

Add pure keyword to the function declaration. At the same time, you need to add the pure keyword to the *calcStockAndMoney* interface declaration in the contracts/interfaces/IOneSwapPair.sol contract and the *fill* function in contracts/libraries/ProxyData.sol

Result

Fixed. The fixed code is shown in the Figure 5 below.

```
774     function calcStockAndMoney(uint64 amount, uint32 price32) external pure override returns (uint stockAmount, uint moneyAmount) {
775         uint[5] memory proxyData;
776         ProxyData.fill(proxyData, 4+32*(ProxyData.COUNT+2));
777         (stockAmount, moneyAmount, ) = _calcStockAndMoney(amount, price32, proxyData);
778     }
```

Figure 5 Fixed calcStockAndMoney function source code

4.3 feeTo address is not set into governance contract

Description

The feeToSetter permission in the OneSwapFactory.sol contract is not included in the Gov contract. The feeToSetter permission can control which address (feeTo) the added liquidity currency (_mintFee) calls to. According to the design document, feeTo should be the address of the buyBack contract, but feeToSetter has the right to change it to another address. If it is changed to a normal wallet Address, then this liquid currency can be used to propose the currency in the pair.

Recommendation

Give feeToSetter authority to the governance contract, and the governance contract will manage feeTo to ensure that it is always a buyBack contract.

Result

Ignore. Keep this code unchanged, to prevent problems with the OneSwapBuyback contract, and later plan to change it to 0 address.

4.4 supervisedSend function code redundant

Description

In the supervisedSend function of the contracts/SupervisedSend.sol contract, the require check on line 34 requires that the values of the two variables info.amount and info.reward must be 0, otherwise the function execution will throw an exception. However, in the line 38 and line 39 of the code, the value of the local variable updateAmount is the sum of info.amount and the function parameter amount. Here, info.amount is 0, so there is no need to add the value of info.amount, and updateReward is the same.

```
30  function supervisedSend(address to, address supervisor, uint112 reward, uint112 amount, address token, uint32 unlockTime, uint256 serialNumber) public
    override {
31      bytes32 key = _getSupervisedSendKey(msg.sender, to, supervisor, token, unlockTime);
32      supervisedSendInfo memory info = supervisedSendInfos[key][serialNumber];
33      require(amount > reward, "SupervisedSend: TOO_MUCH_REWARDS");
34      // prevent duplicated send
35      require(info.amount == 0 && info.reward == 0, "SupervisedSend: INFO_ALREADY_EXISTS");
36      _safeTransferToMe(token, msg.sender, uint(amount).add(uint(reward)));
37      //todo: whether or not to allow serialNumber duplicated supervisedSend
38      uint updateAmount = uint(info.amount).add(amount);
39      uint updateReward = uint(info.reward).add(reward);
40      supervisedSendInfos[key][serialNumber]= supervisedSendInfo(uint112(updateAmount), uint112(updateReward));
41      emit SupervisedSend(msg.sender, to, supervisor, token, amount, reward, unlockTime);
42  }
```

Figure 6 redundant code

Recommendation

It is recommended to delete the redundant code.

Result

Fixed. The fixed code is shown in the Figure 7 below.

```
30  function supervisedSend(address to, address supervisor, uint112 reward, uint112 amount, address token, uint32 unlockTime, uint256 serialNumber) public
    override beforeUnlockTime(unlockTime){
31      bytes32 key = _getSupervisedSendKey(msg.sender, to, supervisor, token, unlockTime);
32      supervisedSendInfo memory info = supervisedSendInfos[key][serialNumber];
33      require(amount > reward, "SupervisedSend: TOO_MUCH_REWARDS");
34      // prevent duplicated send
35      require(info.amount == 0 && info.reward == 0, "SupervisedSend: INFO_ALREADY_EXISTS");
36      supervisedSendInfos[key][serialNumber]= supervisedSendInfo(amount, reward);
37      _safeTransferToMe(token, msg.sender, uint(amount).add(uint(reward)));
38      emit SupervisedSend(msg.sender, to, supervisor, token, amount, reward, unlockTime);
39  }
```

Figure 7 fixed code

4.5 Incorrect Comments

Description

In the *_emitNewLimitOrder* function of the contracts/OneSwapPair.sol contract, the comments on lines 383 and 384 are incorrect.

```
382  function _emitNewLimitOrder(
383      uint64 addressLow, /*255~193*/
384      uint64 totalStockAmount, /*192~128*/
385      uint64 remainedStockAmount, /*127~64*/
386      uint32 price, /*63~32*/
387      uint32 orderID, /*31~8*/
388      bool isBuy /*7~0*/) private {
389      uint data = uint(addressLow);
390      data = (data<<64) | uint(totalStockAmount);
391      data = (data<<64) | uint(remainedStockAmount);
392      data = (data<<32) | uint(price);
393      data = (data<<32) | uint(orderID<<8);
394      if(isBuy) {
395          data = data | 1;
396      }
397      emit NewLimitOrder(data);
398  }
```

Figure 8 Incorrect Comments

Recommendation

Modify the comment on the line 383 to /*255-192*/, and modify the comment on the line 384 to /*191~128*/.

Result

Fixed. The fixed code is shown in the Figure 9 below.

```
382     function _emitNewLimitOrder(
383         uint64 addressLow, /*255-192*/
384         uint64 totalStockAmount, /*191-128*/
385         uint64 remainedStockAmount, /*127~64*/
386         uint32 price, /*63~32*/
387         uint32 orderID, /*31~8*/
388         bool isBuy /*7~0*/) private {
389         uint data = uint(addressLow);
390         data = (data<<64) | uint(totalStockAmount);
391         data = (data<<64) | uint(remainedStockAmount);
392         data = (data<<32) | uint(price);
393         data = (data<<32) | uint(orderID<<8);
394         if(isBuy) {
395             data = data | 1;
396         }
397         emit NewLimitOrder(data);
398     }
```

Figure 9 fixed code

5 Conclusion

Beosin (Chengdu LianAn) conducted a detailed audit on the design and code implementation of the OneSwap project. All the problems found in the audit process were notified to the project party, and got quick feedback and repair from the project party. Beosin (Chengdu LianAn) confirms that all the problems found have been properly fixed or have reached an agreement with the project party has on how to deal with it. The overall result of this OneSwap audit is pass (Distinction).

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com