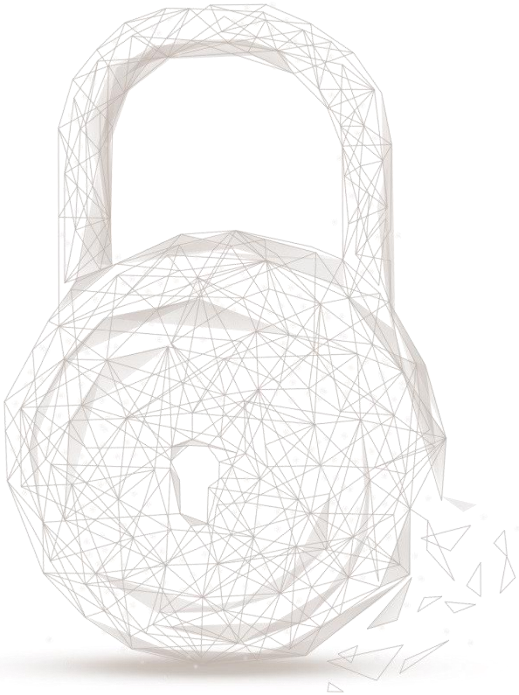
成都链安
BEOSIN

Smart contract security audit report



Audit Number：202101161611

Project Name：Gra

Contract address：

GraSwapGov	0x598C30ABb336948F1B8131e2468C8BCbbC7de9ae
GraSwapPair	0x0BaECBB404828F5eF2169D480D70C40BDD3D1449
PairFeeDistribution	0xc705612B0CA8c026854f38E866Bce2701828486a
GraSwapFactory	0x6b74e5c4137AfD6a7Dbd9d43c9d7af2A258e57A7
GraSwapRouter	0x345A688369E60999f0f2567DC948dca620B9a492
GraSwapBuyback	0x56871674195F91c77c1b4FF1A345d3614169E058

Start Date：2021.01.11

Completion Date：2021.01.16

Overall Result：Pass (Distinction)

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	Pass

		(PRNG)	
		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 Gra project, including Coding Standards, Security, and Business Logic. **The Gra 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 the whether 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.

The Gra project implements a decentralized exchange through AMM and order book mechanism, through which users can freely trade. In addition, the project also implements functions such as governance function, LP token mining function, and fee repurchase.

3.1 AMM and order book

- Description: The *GraSwapPair_deploy* contract implements the functions of a market-making engine and an order book, and supports automated market-making and on-chain matching. The contract provides *addLimitOrder* to the *GraSwapPair_deploy* contract to add limit orders, and *addMarketOrder* to add market orders. Users can call *removeOrder* to cancel a single unfinished order, or call *removeOrders* to cancel batches of unfinished orders. In addition, this contract also provides a corresponding liquidity pool and order status query interface for external systems to obtain data on the chain.

- Related functions: *addLimitOrder*, *addMarketOrder*, *removeOrder*, *removeOrders*, *getPrices*, *calcStockAndMoney*, *getOrderList*

- Result: Pass

3.2 Trading pair management

- Description: The *GraSwapFactory_deploy_main* contract stores some key parameters and implements the creation of a pool. *GraSwapFactory_deploy_main* stores the transaction fee ratio *FeeBPS* of the pool, the beneficiary addresses *feeTo_1*, *feeTo_2*, etc. of the tokens from the pool, and the logic address *pairLogic* of the pool. *FeeBPS* and *pairLogic* can only be changed by the governance contract address, and *feeTo* is set at initialization The *feeToSetter* address is modified. In addition, anyone can call the *createPair* to create a pool proxy contract *GraSwapPairProxy*.

- Related functions: *createPair*, *setFeeTo*, *setFeeToSetter*, *setPairLogic*, *setFeeBPS*

- Result: Pass

3.3 Repurchase

- Description: The *GraSwapBuyback_deploy* contract implements repurchase. Liquidity tokens (LP Token) generated by the adding and removing of liquidity in all pools can be transferred into the *GraSwapBuyback_deploy* contract. Anyone can call the *removeLiquidity* to convert LP tokens into tokens in the corresponding pool. In addition, the contract also defines the *mainToken* array to store the supported token types. Anyone can call the *swapForMainToken* to convert the tokens in the contract into the supported token types in the corresponding pool. Anyone can call the *swapForGrasAndBurn* to convert the tokens supported in the contract into *Gra* through the corresponding pool and destroy it. The owner of the *GraSwapToken_deploy* contract can change the supported token types.

- Related functions: *addMainToken, removeMainToken, removeLiquidity, swapForMainToken, swapForGrasAndBurn*

- Result: Pass

3.4 Governance

- Description: The *GraSwapGov_deploy* contract implements governance-related functions. The owner of the *GraSwapGov_deploy* contract can call the *submitFundsProposal*, *submitParamProposal*, and *submitUpgradeProposal* to initiate a proposal to use the *Gra* balance of the governance contract, change the transaction pool fee, and change the transaction pool logical address. Any address holding more than 0.1% of the total *Gra* can call *submitTextProposal* to make a text proposal, but it must be one day after the end of the previous proposal. Each proposal lasts for three days, and only one proposal can exist at the same time. Any user can call the *vote* and lock *Gra* in *GraSwapGov_deploy* to express "YES" or "NO" to the proposal (votes can be changed before the proposal ends). By the end of the voting, more options for *Gra* are the result of the proposal. If the proposal fails, part *Gra* tokens of the initiator of the proposal will be deducted and destroyed. If the proposal is passed, anyone can call the tally method to execute the proposal. Users can call *withdrawGras* to withdraw the locked *Gra* when they are not voting for the current proposal.

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

- Result: Pass

3.5 Routing

- Description: The *GraSwapRouter_deploy* contract implements the functions of limit orders, market orders, and liquidity adding and removing, and supports market orders across trading pools. Anyone can call the *swapToken* to exchange for a single transaction pool or across transaction pools, call *limitOrder* to create a limit order, call *addLiquidity* to inject liquidity into the transaction pool (create if there is no such liquidity pool), and call *removeLiquidity* to remove the Liquidity in the pool.

- Related functions: *limitOrder*, *swapToken*, *addLiquidity*, *removeLiquidity*

- Result: Pass

3.6 Liquidity mining

- Description: The *GraPool* contract implements the function of depositing LP tokens to mine *Gra* and *BLW* tokens. The user deposits liquidity tokens to the *GraPool* contract for mining through the *deposit* function. The specific rate of return is determined by the *GraPerBlock* and *BLWPerBlock* parameters set when the owner adds the corresponding pool information. After the deposit period expires, the user can withdraw the deposited tokens by calling the *withdraw* and settle all *BLW* rewards. Of the two rewards, *BLW* rewards can be claimed at any time, but *Gra* rewards will be locked for a period of time.

- Related functions: *addPool, updatePool, reclaimBLWStakingReward, reclaimGraStakingReward, deposit, withdraw*

- Result: Pass

3.7 Split handling fee

- Description: The *PairFeeDistribution_deploy* contract implements the function of evenly dividing the handling fee. Every *GraSwapPair_deploy* contract will automatically send 25% of the handling fee to this contract when the handling fee is deducted. The *owner* of the contract can add members participating in dividends by calling the *addInvestors* function, and remove members by calling the *removeInvestors* function. Members will be distributed the fees (LP tokens) generated from joining. Members can receive the fees of a specified pool by calling the *withdrawfee*, or receive the commission fees of multiple pools in batches through the *batchwithdrawfee* function.

- Related function: *addpair, addInvestors, removeInvestors, updateInvestorPairPerShare, withdrawfee, batchwithdrawfee*

- Safety suggestion:

1） The count variable in *batchwithdrawfee* is used as a global variable, which causes interference when different users call this function, so that users need to call this function twice to get all the fees.

2） The *PairFeeDistribution_deploy* contract address can be changed in the *GraSwapFactory* contract. If the owner permission is lost, the user may not be able to receive the fees.

- Fixed result: ignore

- Result: Pass

4 Conclusion

Beosin (Chengdu LianAn) conducted a detailed audit on the design and code implementation of the Gra 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 Gra audit is pass (Distinction).

Beosin

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com