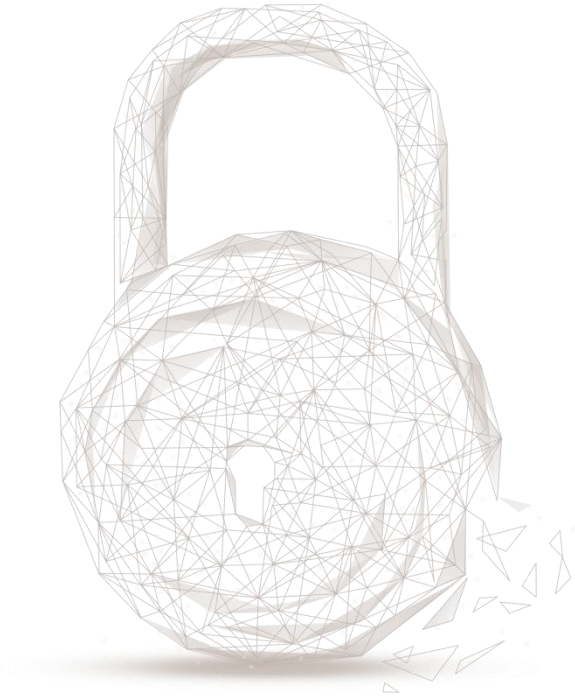
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202102012031

Report Query Name: RAMPSTAKINGV2

Smart Contract Name:

RampStakingV2

Smart Contract Link:

https://github.com/RAMP-DEFI/RAMP_STAKING/blob/master/contracts/v2/RampStakingV2.sol

Origin commit id: c5c7a8ebe856018e840ddb8903fe36f55da23c4d

Final commit id: 5612c3471c2326cd4caab593909447d617132869

Start Date：2021.01.26

Completion Date：2021.02.01

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

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 project RampStakingV2, including Coding Standards, Security, and Business Logic. **The RampStakingV2 project 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 the 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) *initialize* function

- Description: The contract uses the proxy-implementation framework. After setting the implementation contract address of the proxy contract to RampStakingV2, the project party will call the *initialize* function of the RampStakingV2 contract to initialize the contract information. The content includes the address of the RAMP token contract, the address of the RAMP token wallet, the contract owner, the fee percentage of the reward, and the parameters of the pool. The *initialize* function can only be called once.
- Related functions: *initialize*
- Result: Pass

(2) *setRampPerBlock* function

- Description: The contract owner can set the number of rewards for each block of the pool. After the pool rewards for each block is modified, it will affect the value of RAMP rewards when users withdraw or deposit tokens.
- Related functions: *setRampPerBlock, updatePool*
- Safety suggestion: The *setRampPerBlock* function should execute *updatePool* before changing *rampPerBlock*. If it is not executed, it will cause the *rampPerBlock* of the block after the last updatePool to be a new rampPerBlock.
- Fixed result: Fixed.
- Result: Pass

(3) *updatePool* function

- Description: Any user can call *updatePool* function to update latest RAMP rewards information of pool on current block.
- Related functions: *updatePool, getPoolReward*
- Result: Pass

(4) *deposit* function

- Description: The contract implements the *deposit* function for users to stake tokens, the user pre-approves this contract address and then calls this function to deposit tokens (require the pool is existed). Update the pool information and the user deposit information when the user is deposited, if the user has previous deposit, call the internal function *_harvest* to calculate the user's previous deposit reward.
- Related functions: *deposit, updatePool*

- Result: Pass

(5) *withdraw* function

- Description: The user can call the *withdraw* function to withdraw a specified number of deposits, and call the internal function *_harvest* to receive RAMP rewards. Withdraw tokens will be locked for 24 hours. After 24 hours, the user can call the *withdrawClaim* function to unlock and send it to the user address. If the user is a whitelisted address, the user can directly withdraw deposit through the *withdrawClaimWhitelist* function without locking it for 24 hours.

- Related functions: *withdraw, withdrawClaim, withdrawClaimWhitelist, updatePool, removeWhitelistAddress, addWhitelistAddress*

- Safety suggestion: Whitelist users can claim some locked tokens twice by calling the *withdrawClaim* function and then *withdrawClaimWhitelist*. It is recommended that the *withdrawClaimWhitelist* function starts to unlock from the index after the last unlock.

- Fixed result: Fixed.

- Result: Pass

(6) *claimReward* function

- Description: The user can manually claim the RAMP reward through the *claimReward* function. Before claiming the reward, the *updatePool* function will be called to update the pool's cumulative reward information, and then the internal function *_harvest* will be called to claim the reward. The reward RAMP will be directly burned 5% (the ratio is adjustable, the maximum is 10%), 25% of the remaining RAMP will be sent directly to users, and 75% vesting over 1 year (2425847 blocks). The user can call the *claimVestedReward* function to unlock the reward.

- Related functions: *claimReward, updatePool, claimVestedReward, viewVestedReward*

- Result: Pass

(7) *setFee* function

- Description: The contract owner can set the fee percentage for the reward. The maximum fee percentage for the reward is 10%.

- Related functions: *set*

- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the project RAMPSTAKINGV2. 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 audit result of the project RAMPSTAKINGV2 is Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com