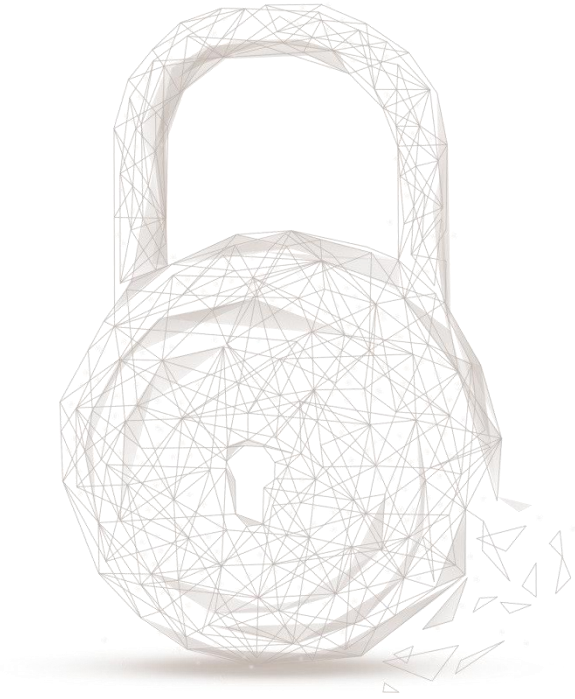
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202012081830

Report Query Name: RAMP_REWARDS_MANAGER

Smart Contract Name:

RewardsManager.sol

Smart Contract Link:

https://github.com/RAMP-DEFI/RAMP_REWARDS_MANAGER/blob/master/contracts/ramp/RewardsManager.sol

Origin commit id: b59a62270c7d97cf2210e8a3cc16c3a23a82aa21

Final commit id: a58d8bd9ee1d231047b25559d395215abb3607b0

Start Date：2020.12.04

Completion Date：2020.12.08

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 project RAMP_REWARDS_MANAGER, including Coding Standards, Security, and Business Logic. **The RAMP_REWARDS_MANAGER 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 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

3.1 Set chain enabled status

- Description: As shown in Figure 1 below, the contract operator can call *setChainEnabled* function to set chain enabled status. Only chains whose enabled status is 1 can be set exchange rates.

```
/// @dev Sets a chain enabled/disabled. Not possible for RAMP
/// @param _chainId Chain Id to set enabled/disabled
/// @param _enabled Enabled or disabled boolean
function setChainEnabled(uint8 _chainId, bool _enabled)
public
onlyOperator
{
    require(_chainId < 16, "chainId overflow");
    require(_chainId != CHAIN_ID_RAMP, "Cannot change RAMP");

    // Set the flag
    enabledChains[_chainId] = _enabled ? 1 : 0;

    // Emit event
    emit ChangeChainEnabled(_chainId, _enabled);
}
```

Figure 1 Source code of function *setChainEnabled*

- Related functions: *setChainEnabled*

- Result: Pass

3.2 Set exchange rate through Chainlink oracle

- Description: As shown in Figure 2 below, the contract operator can call *startRateUpdates* function to retrieve the exchange rates for the day. This function calls the *prepareAndSendChainlinkRequest* function (as shown in Figure 3) to retrieve the exchange rate of all enabled chains. The contract operator can also directly call *prepareAndSendChainlinkRequest* to retrieve the exchange rate of the specified chain.

```
/// @dev Trigger rate updates (chainlink) for all enabled chains
function startRateUpdates()
public
onlyOperator
{

    // Loop through all enabled chains.
    for (uint8 chainId = 0; chainId < 16; chainId++) {
        if (enabledChains[chainId] == 1) {
            prepareAndSendChainlinkRequest(chainId);
        }
    }

    emit StartRateUpdates();
}
```

Figure 2 Source code of function *startRateUpdates*

```
/// @dev Prepares and sends Chainlink request (coingecko api) for rates for chain.
/// @param _chainId Chain ID to do the request for
function prepareAndSendChainlinkRequest(uint8 _chainId)
public
onlyOperator
{
    require(_chainId < 16, "chainId overflow");
    require(enabledChains[_chainId] > 0, "Chain is not enabled");

    Chainlink.Request memory chainlinkRequest;

    string memory tokenId = chainTokenId[_chainId];

    chainlinkRequest = buildChainlinkRequest(
        _stringToBytes32(chainlinkJobId),
        address(this),
        this.fulfillRateRequest.selector
    );

    chainlinkRequest.add("get", string(abi.encodePacked("https://api.coingecko.com/api/v3/simple/price?vs_currencies=usd&ids=", tokenId)));
    chainlinkRequest.add("path", string(abi.encodePacked(tokenId, ".usd")));
    chainlinkRequest.addInt("times", int256(10 ** RATES_DECIMALS));

    bytes32 requestId = sendChainlinkRequestTo(
        chainlinkOracle,
        chainlinkRequest,
        chainlinkFee
    );

    // Store the request reference
    rateChainlinkRequests[requestId] = RateRequest(_chainId, _currentDaystamp());

}
```

Figure 3 Source code of function *prepareAndSendChainlinkRequest*

- Related functions: *startRateUpdates, prepareAndSendChainlinkRequest, fulfillRateRequest*

- Result: Pass

3.3 Set exchange rate manually

- Description: As shown in Figure 4 below, the contract operator can call *setUsdRate/setUsdRates* function to set exchange rate manually. The corresponding chain must be enabled, otherwise the exchange rate cannot be successfully set. In addition, manually setting the exchange rate cannot set the exchange rate of the chain in the future.

```
/// @dev Write exchange rates for 8 enabled chains for a specific day
/// @param _daystamp Daystamp (daynr since 1970-1-1)
/// @param _value Rate value (8-decimals integer)
function setUsdRates(uint16 _daystamp, uint256 _value)
onlyOperator
public
{
    // Loop 8 chainId's (we could only fit 8 in the _value)
    for (uint8 chainId = 0; chainId < 8; chainId++) {
        if (enabledChains[chainId] > 0) {
            _setUsdRate(chainId, _daystamp, uint32(_get32bitSlot(_value, chainId)));
        }
    }
}
```

Figure 4 Source code of function *setUsdRates*

- Related functions: *setUsdRates, setUsdRate*

- Result: Pass

3.4 Set rewards multiplier for a chain

- Description: As shown in Figure 5 below, the contract operator can call *setChainMultiplier* function to set rewards multiplier for a chain. The rewards multiplier can only be set once, and the past rewards multiplier cannot be set. The same as setting the exchange rate, the rewards multiplier can be set only when the corresponding chain is enabled.

```
/// @dev Sets rewards multiplier for a chain. Entire epoch (8 days) at once.
/// @param _chainId Chain Id to set multiplier values for
/// @param _epochNr EpochNr to set values for
/// @param _epochData Values (8 daily values each in 32-bit slot)
function setChainMultiplier(uint8 _chainId, uint16 _epochNr, uint256 _epochData)
public
onlyOperator
{
    uint16 currentEpoch = _currentEpoch();
    uint16 currentDaySlot = _currentDaystamp() % uint16(EPOCH_DAYS);
    uint256 epochData = _epochData;

    require(_chainId > 0 && _chainId < 16, "chainId invalid");
    require(enabledChains[_chainId] > 0, "Chain is not enabled");
    require(_epochNr >= currentEpoch, "Cannot change past values");

    if (_epochNr == currentEpoch && currentDaySlot > 0) {
        epochData = chainMultipliers[_epochNr][_chainId];

        // Transfer the future slots only. May be inefficient still.
        for (uint16 dayslot = currentDaySlot; dayslot <= 7; dayslot++) {

            epochData = _updateEpochSlot(
                epochData,
                dayslot,
                _get32bitSlot(_epochData, dayslot)
            );
        }
        chainMultipliers[_epochNr][_chainId] = epochData;

    } else {

        // Set entire epoch
        chainMultipliers[_epochNr][_chainId] = epochData;
    }

    emit ChangeChainMultiplier(_chainId, _epochNr, epochData);

}
```

Figure 5 Source code of function *setChainMultiplier*

- Related functions: *setChainMultiplier*

- Result: Pass

3.5 Claim Rewards

- Description: The contract user can call the *claimRewards* function to claim rewards. To call this function, the user needs to get the operator's signature corresponding to the reward information first, and then call the *claimRewards* function with the reward information and signature data as parameters. If the signature and reward information are correct, the contract will calculate all the rewards that should be

claimed according to the reward information, and send the receivable rewards directly to the user, and update the user reward receiving information.

- Related functions: *claimRewards, getLastClaimDayNr, getClaimableRewards*
- Result: Pass

4. Details of audit results

4.1 Overflow in *_processChainRewards* function

- Description: *_startDayNr* is passed in the *_processClaims* function, and *_startDayNr* > 1 is not detected in the *_processClaims* function. If the user mistakenly inputs 0, then the 322nd line of code of the *_processChainRewards* function will have an integer overflow. Similarly, if the *_data* passed by the user is empty or the length is less than 4 bytes, then the 331th line will overflow.

```
317     function _processChainRewards(uint8 _chainId, uint16 _startDayNr, bytes memory _data)
318     internal
319     returns (uint16 lastProcessedDayNr, uint256 rampBalanceIncrease)
320     {
321         // Set lastProcessedDayNr to previous day so we can increment conveniently
322         lastProcessedDayNr = _startDayNr - 1;
323
324         // Initialize struct, we use this to prevent "stack too deep"
325         RewardsCalcValues memory values = RewardsCalcValues(0, 0, 0, 0, 0, 0);
326
327         // Accumulator variable
328         rampBalanceIncrease = 0;
329
330         // Determine last data day (from input data or today, whichever is sooner)
331         uint16 lastDataDay = _startDayNr + uint16((_data.length / 4) - 1);
332         uint16 today = _currentDaystamp();
```

Figure 6 The origin source code of *_processChainRewards* function

- Suggestion for modification: Add the value detection of *_startDayNr* and *_data*.
- Fix Result: Fixed. The final code is shown below.

```
// Check if inputs are valid
require(chainId > 0 && chainId < 16, "Invalid chainId");
require(data.length % 4 == 0 && data.length > 0, "Data invalid");
require(startDayNr > 18600, "Invalid startDayNr");
require(startDayNr <= _currentDaystamp(), "Cannot get future rewards");
require(enabledChains[chainId] > 0, "Chain is not enabled");
require(claimNonce == accountData[account].claimNonce, "Invalid claim nonce");

// Load the last day we processed before (0 if never)
lastProcessDayNr = accountData[account].chainLastProcessDayNr[chainId];

// Set the lastProcessDayNr to day before startDay if its the first time
if (lastProcessDayNr == 0) lastProcessDayNr = startDayNr - 1;

// Only allow the startDayNr to be directly after lastProcessDayNr
require(startDayNr > lastProcessDayNr && startDayNr - lastProcessDayNr == 1, "Invalid startDayNr");

// Process the data
(lastProcessDayNr, rampTransferable) = _processChainRewards(chainId, startDayNr, data);
```

Figure 7 The final source code of *_processClaims* function

4.2 Code optimization in *_processChainRewards* function

- Description: The initial value of the *epochSlot* variable is always 0.

```
337         // Set epoch we're processing now, we'll add the loop counter to it to get actual
338         uint16 processingEpochNr = _startDayNr / uint16(EPOCH_DAYS);
339
340         // Declare the input variable that will receive the 32 bytes
341         uint256 epochData;
342
343         uint16 epochSlot = processingEpochNr - (_startDayNr / uint16(EPOCH_DAYS));
344
```

Figure 8 The origin source code of *_processChainRewards* function

- Suggestion for modification: Set the initial value of *epochSlot* to 0.
- Fix Result: Fixed. The final code is shown below.

```
// Set epoch we're processing now, we'll add the loop counter to it to get actual
uint16 processingEpochNr = _startDayNr / uint16(EPOCH_DAYS);

// Declare the input variable that will receive the 32 bytes
uint256 epochData;

uint16 epochSlot = 0;

uint256 dayslot = 0;
uint256 dayRewardsChain = 0;
```

Figure 9 The final source code of *_processChainRewards* function

4.3 Misplaced data in *_processChainRewards* function

- Description: In line 369, it has been assumed that the data for each epoch is entered completely. But if the data entered by the user does not start from a complete epoch, an exception will occur. For example, the entered *_startDayNr* is a certain day in the middle of the epoch, such as the sixth day. When reading the data of the next epoch, the data will be misaligned due to the data offset.

```
358         if (
359             dayNr == _startDayNr
360             || processingEpochNr < dayNr / uint16(EPOCH_DAYS)
361         ) {
362             // Set currently processing epoch Nr
363             processingEpochNr = dayNr / uint16(EPOCH_DAYS);
364
365             epochSlot = processingEpochNr - (_startDayNr / uint16(EPOCH_DAYS));
366
367             // Load 32 bytes into the "input" variable
368             assembly {
369                 epochData := mload(add(add(_data, mul(epochSlot, 32)), 32))
370             }
371
372             // For the first day, shift epochdata right if startDayNr does not align with epoch
373             if (dayNr == _startDayNr) epochData = epochData >> (_startDayNr % 8 * 32);
374
```

Figure 10 The origin source code of *_processChainRewards* function

- Suggestion for modification: Calculate the offset position of data and read the corresponding epoch value from the correct position.
- Fix Result: Fixed. The final code is shown below.

```
if (
    dayNr == _startDayNr
    || processingEpochNr < dayNr / uint16(EPOCH_DAYS)
) {
    // Set currently processing epoch Nr
    processingEpochNr = dayNr / uint16(EPOCH_DAYS);

    epochSlot = processingEpochNr - (_startDayNr / uint16(EPOCH_DAYS));

    // If startDayNr is also first day of epoch, offset is 32
    // If not first, we need to reduce offset by days * 32bit
    uint256 startOffset = 32 - (_startDayNr % 8) * 4;

    // Load 32 bytes into the "input" variable. Take some earlier bytes to properly align starting points
    assembly {
        epochData := mload(add(add(_data, mul(epochSlot, 32)), startOffset))
    }
```

Figure 11 The final source code of *_processChainRewards* function

5. Conclusion

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

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com