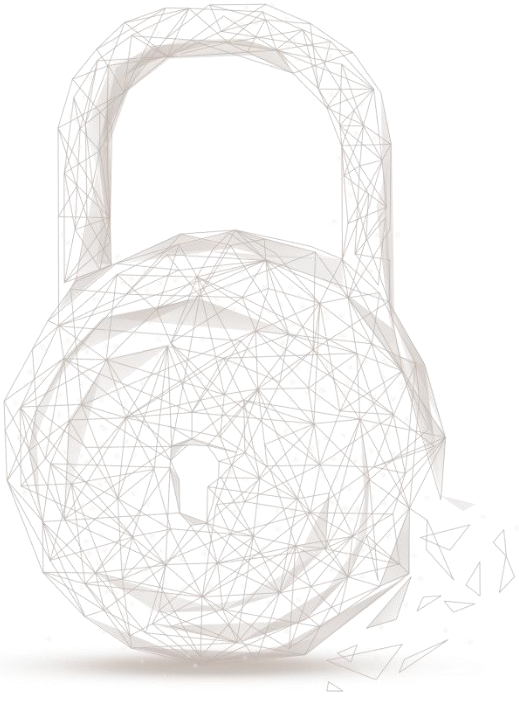
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202011271625

Report Query Name: RAMP_REWARDS_MANAGER

Smart Contract Name:

RewardsManager.sol

Smart Contract Link:

https://github.com/RAMP-DEFI/RAMP_REWARDS_MANAGER.git

Origin commit id: b4aa994d57b0db1f40192a5b813a87518ce31c77

Final commit id: 4c42767a38547e87f45c2757143bf716d796ab42

Start Date：2020.11.23

Completion Date：2020.11.27

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

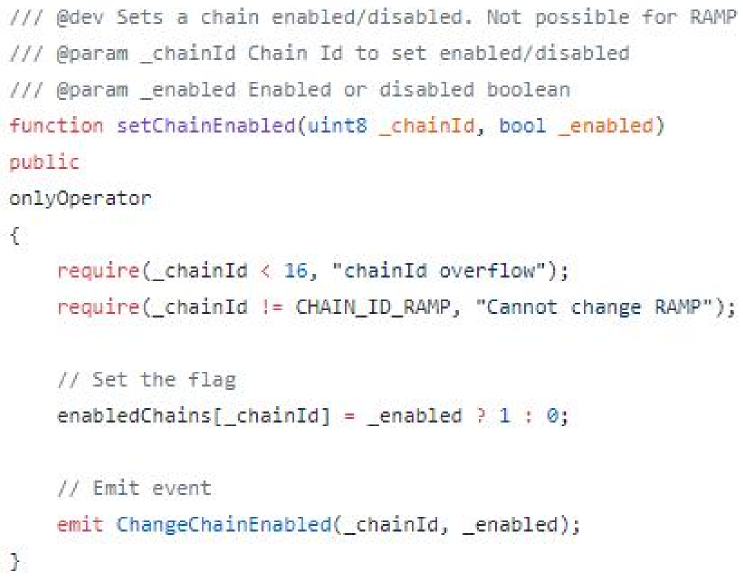
2.11 Overriding Variables

- 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);
}
```

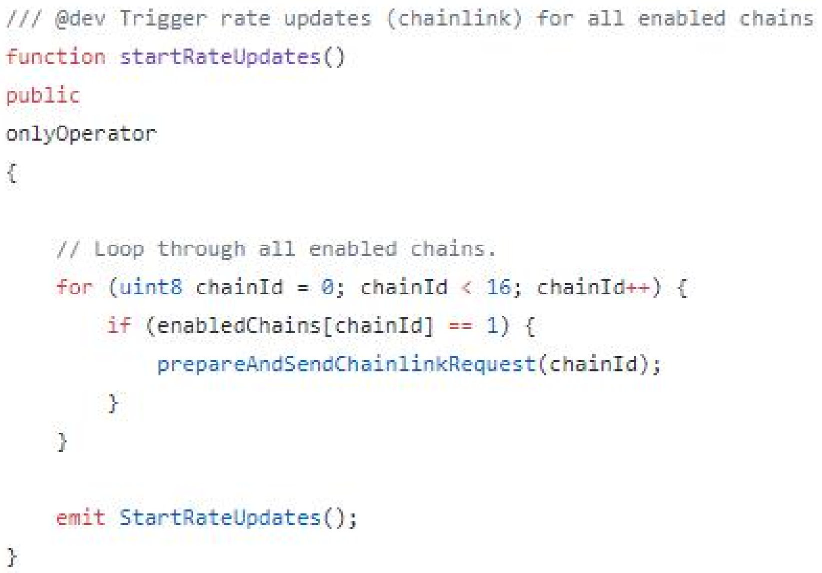
Figure 1 Source code of function *setChainEnabled*

- Related functions: *setChainEnabled*

- Result: Pass

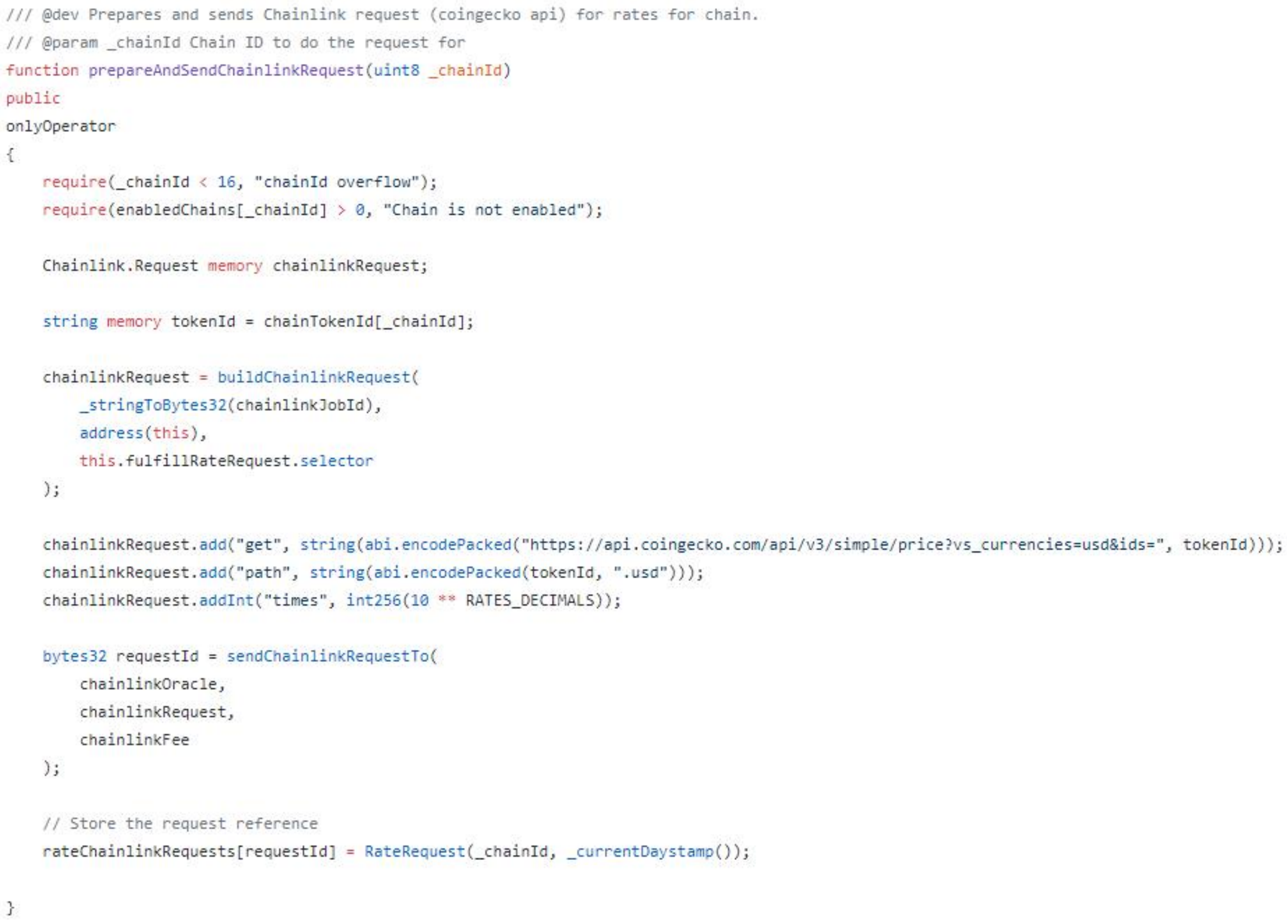
3.2 Set exchange rate through Chainlink oracle

- 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());

}
```

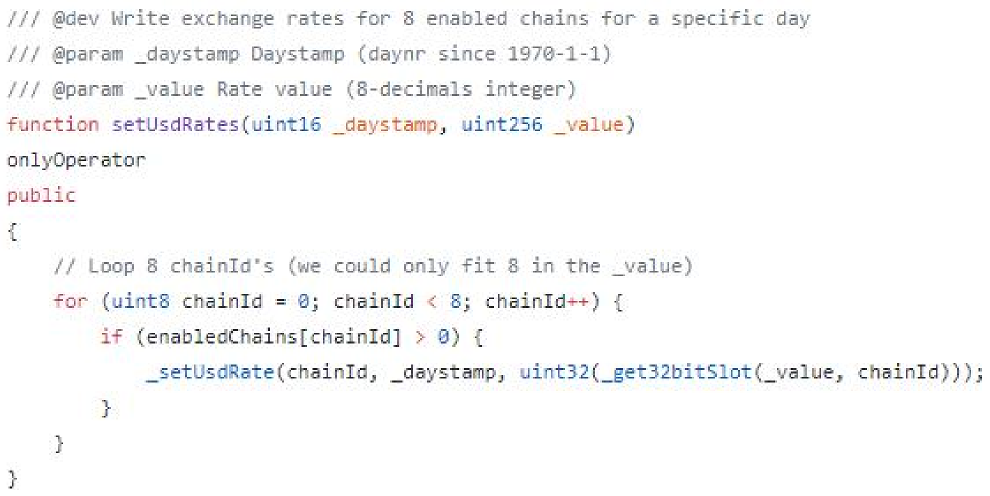
Figure 3 Source code of function *prepareAndSendChainlinkRequest*

- 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)));
        }
    }
}
```

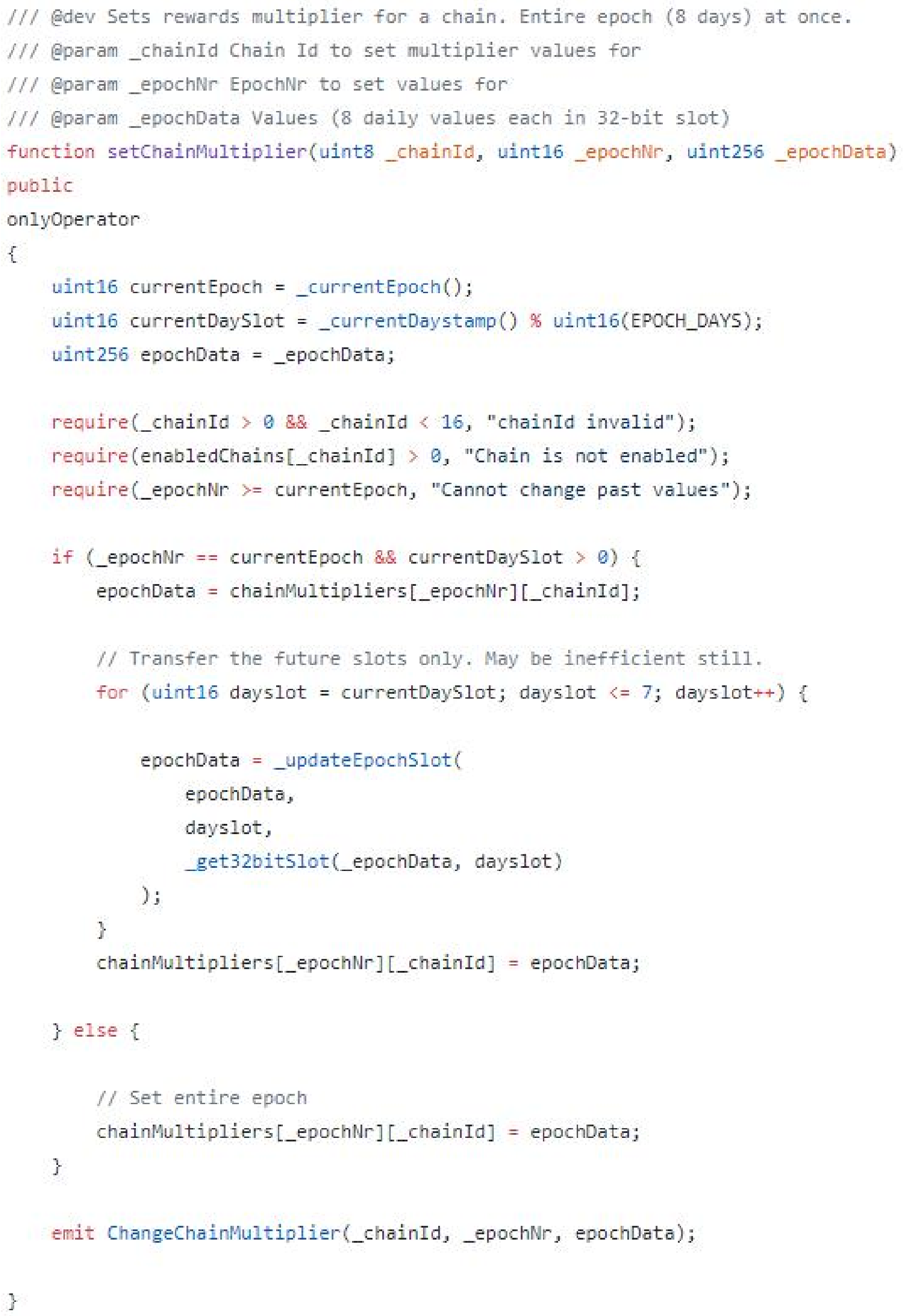
Figure 4 Source code of function *setUsdRates*

- 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

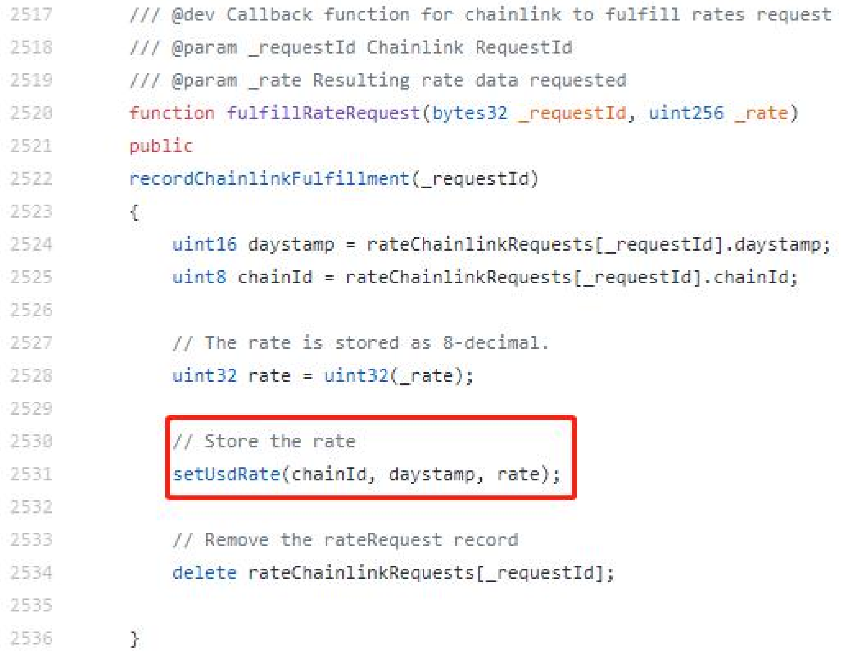
3.5 Process Rewards

- Description: The contract operator can call the *fulfillRewards* function to process rewards. If "_transferRewardsNow" is true, any unclaimed RAMP will be paid out immediately. The user can also receive the corresponding reward through the *claimRewards* function.
- Related functions: *fulfillRewards, claimRewards*
- Result: Pass

4. Details of audit results

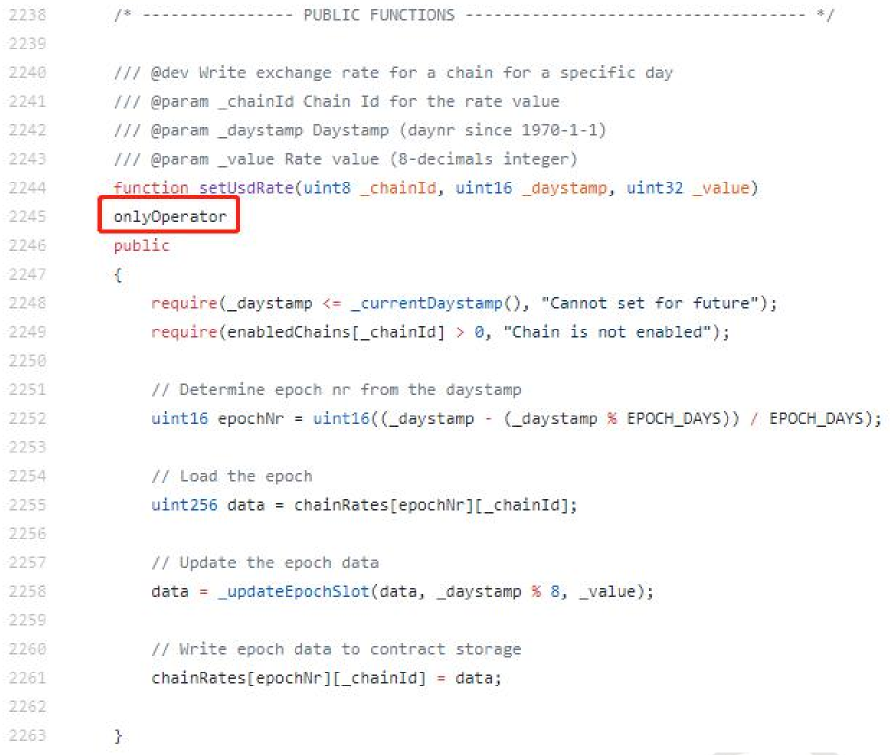
4.1 *fulfillRateRequest* and *setUsdRate* function

- Description: The *fulfillRateRequest* function calls the *setUsdRate* function, but the *setUsdRate* function uses the *onlyOperator* modifier, which causes the Oracle of Chainlink to throw an exception when calling *fulfillRateRequest*, and the corresponding exchange rate cannot be set successfully.



```
    /// @dev Callback function for chainlink to fulfill rates request
    /// @param _requestId Chainlink RequestId
    /// @param _rate Resulting rate data requested
    function fulfillRateRequest(bytes32 _requestId, uint256 _rate)
    public
    recordChainlinkFulfillment(_requestId)
    {
        uint16 daystamp = rateChainlinkRequests[_requestId].daystamp;
        uint8 chainId = rateChainlinkRequests[_requestId].chainId;

        // The rate is stored as 8-decimal.
        uint32 rate = uint32(_rate);

        // Store the rate
        setUsdRate(chainId, daystamp, rate);

        // Remove the rateRequest record
        delete rateChainlinkRequests[_requestId];

    }
```

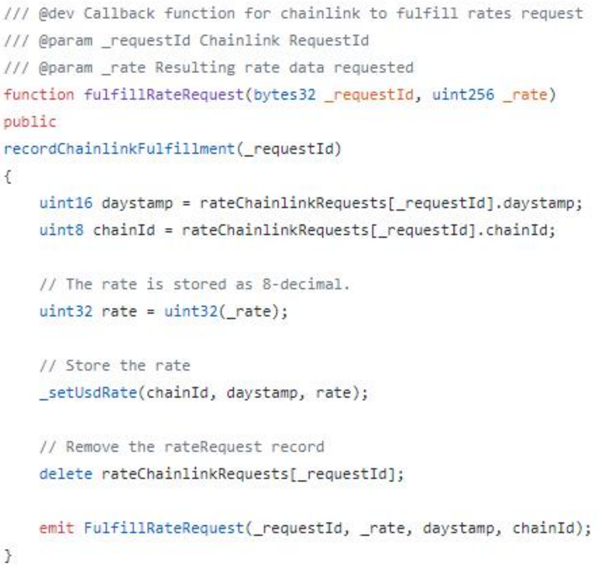
Figure 6 The origin source code of *fulfillRateRequest* function



```
    /* ---------------- PUBLIC FUNCTIONS ------------------------------------ */

    /// @dev Write exchange rate for a chain for a specific day
    /// @param _chainId Chain Id for the rate value
    /// @param _daystamp Daystamp (daynr since 1970-1-1)
    /// @param _value Rate value (8-decimals integer)
    function setUsdRate(uint8 _chainId, uint16 _daystamp, uint32 _value)
    onlyOperator
    public
    {
        require(_daystamp <= _currentDaystamp(), "Cannot set for future");
        require(enabledChains[_chainId] > 0, "Chain is not enabled");

        // Determine epoch nr from the daystamp
        uint16 epochNr = uint16((_daystamp - (_daystamp % EPOCH_DAYS)) / EPOCH_DAYS);

        // Load the epoch
        uint256 data = chainRates[epochNr][_chainId];

        // Update the epoch data
        data = _updateEpochSlot(data, _daystamp % 8, _value);

        // Write epoch data to contract storage
        chainRates[epochNr][_chainId] = data;

    }
```

Figure 7 The origin source code of *setUsdRate* function

- Suggestion for modification: Add an internal function *_setUsdRate*, which is used to implement the *setUsdRate* function, and then *fulfillRateRequest* can call *_setUsdRate* to set the corresponding value. *SetUsdRate* can be directly called *_setUsdRate* to set the corresponding exchange rate.
- Fix Result: Fixed. The final code is shown below.

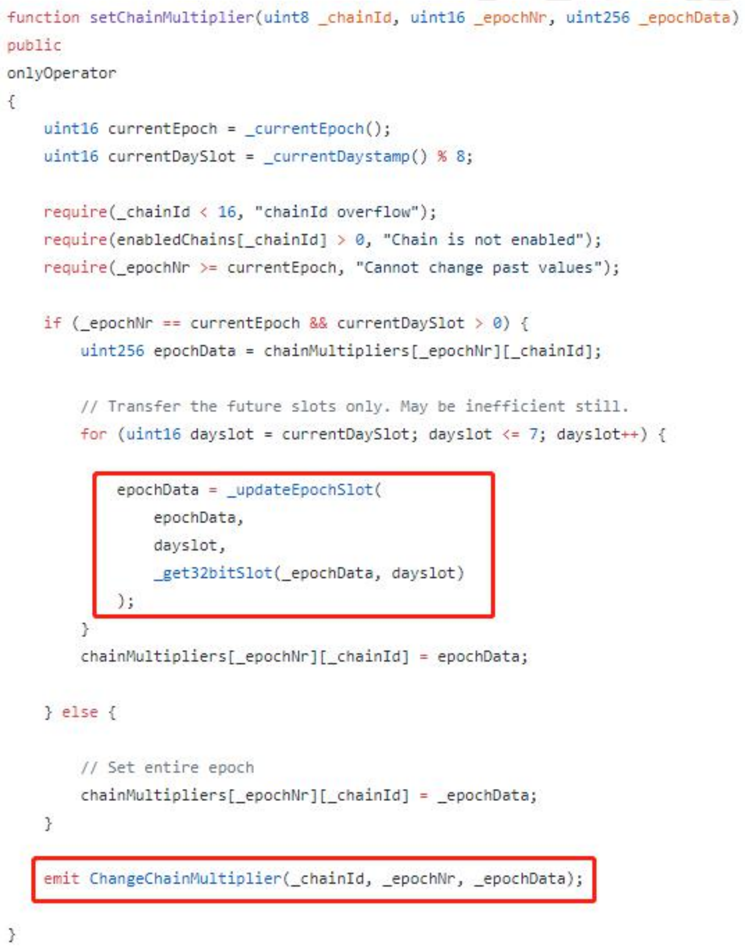


```
/// @dev Callback function for chainlink to fulfill rates request
/// @param _requestId Chainlink RequestId
/// @param _rate Resulting rate data requested
function fulfillRateRequest(bytes32 _requestId, uint256 _rate)
public
recordChainlinkFulfillment(_requestId)
{
    uint16 daystamp = rateChainlinkRequests[_requestId].daystamp;
    uint8 chainId = rateChainlinkRequests[_requestId].chainId;

    // The rate is stored as 8-decimal.
    uint32 rate = uint32(_rate);

    // Store the rate
    _setUsdRate(chainId, daystamp, rate);

    // Remove the rateRequest record
    delete rateChainlinkRequests[_requestId];

    emit FulfillRateRequest(_requestId, _rate, daystamp, chainId);
}
```

Figure 8 The final source code of *fulfillRateRequest* function

4.2 *setChainMultiplier* function

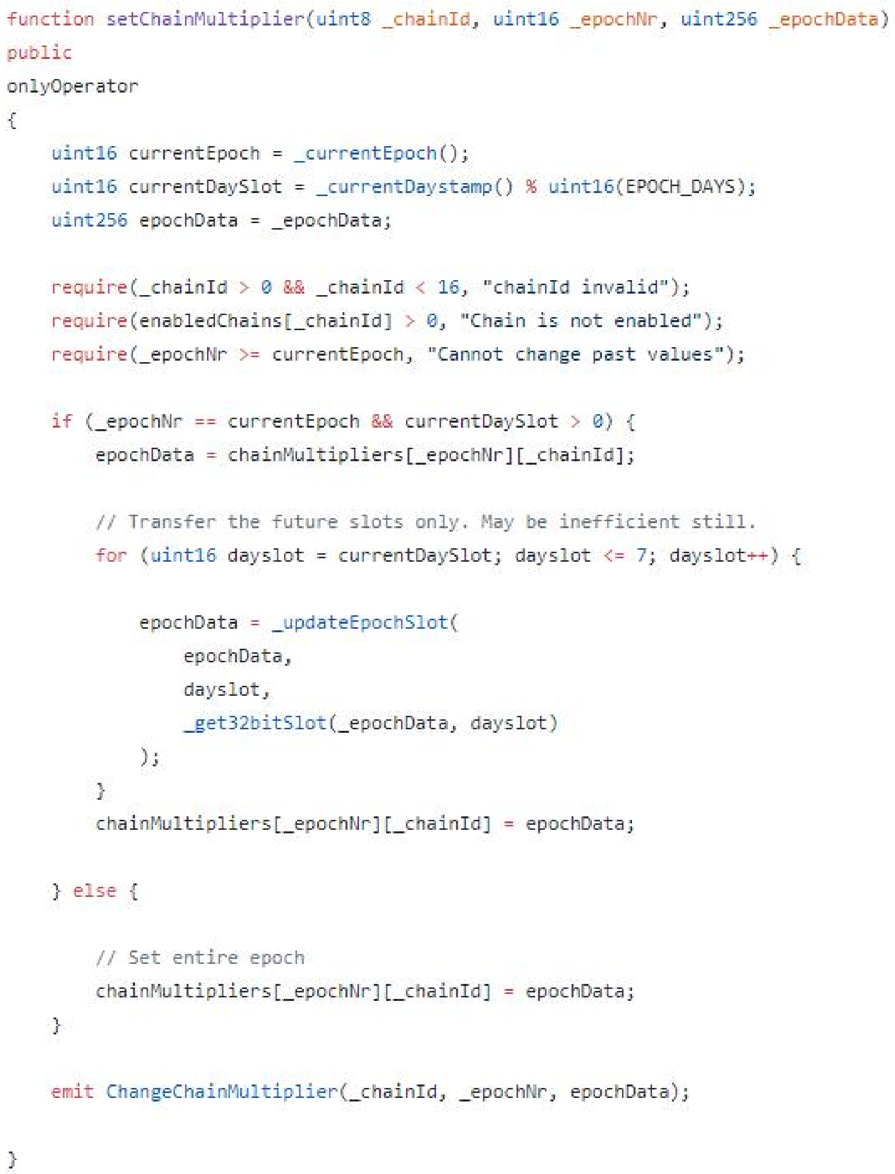
- Description: The *setChainMultiplier* function can modify the future slots of the current epoch in lines 500-503, but if the future slots are not 0, *_updateEpochSlot* will not modify the corresponding slot. At same time, the *epochData* in the *ChangeChainMultiplier* event does not match the actual *epochData*.



```
function setChainMultiplier(uint8 _chainId, uint16 _epochNr, uint256 _epochData)
public
onlyOperator
{
    uint16 currentEpoch = _currentEpoch();
    uint16 currentDaySlot = _currentDaystamp() % 8;

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

    if (_epochNr == currentEpoch && currentDaySlot > 0) {
        uint256 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 9 The origin source code of *setChainMultiplier* function

- Suggestion for modification: Modify *epochData* related code.
- Fix Result: Fixed. The final code is shown below.



```
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 10 The final source code of *fulfillRateRequest* 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 or have reached an agreement with the project party has on how to deal with it. **The overall audit result of the project RAMP_REWARDS_MANAGER is Pass.**

BEOSIN
Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com