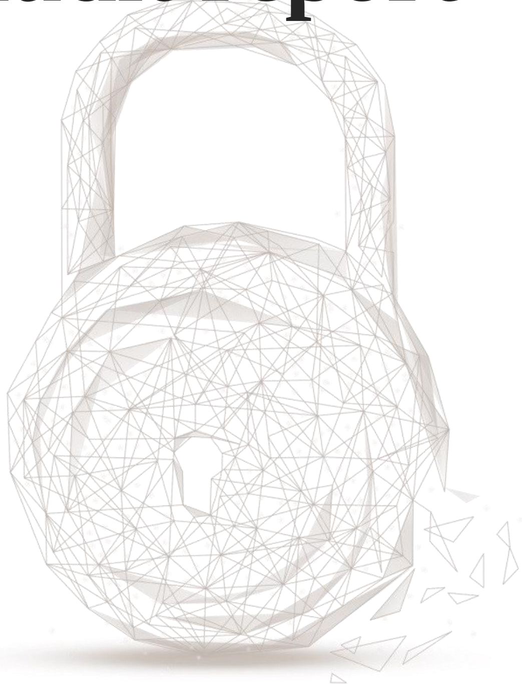
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202104211446

Report Query Name: ramp-protocol

Smart Contract Link:

https://github.com/RAMP-DEFI/ramp-protocol/tree/development

Start Commit Hash：

984d866c987c6543259b628401ed167d872729a4

Finish Commit Hash：

a6111042d048aaee29cf497ed9098b3b55b1c1cd

Start Date：2021.02.22

Completion Date：2021.04.21

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

Disclaimer: This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project. 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 smart contracts project ramp-protocol, including Coding Standards, Security, and Business Logic. **The ramp-protocol project passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

Audit Contents:

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 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

Check whether the business is secure.

3.1 Business analysis of Contract RUSD and RToken.

(1) Basic Token Information of RUSD

Token name	rUSD
Token symbol	rUSD
decimals	18
totalSupply	The initial supply is 0, mintable, burnable
Token type	ERC677

Table 1 Basic Token Information

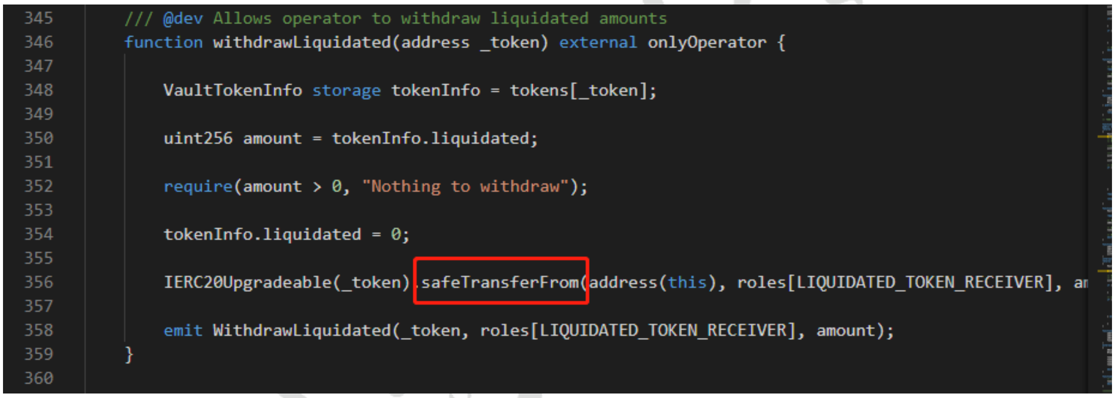
(2) Introduction of rToken

rToken is a token issued based on the deposit of tokens. When a user deposits tokens in the vault contract, the corresponding rToken will be mined and stake to the vault contract.

3.2 Contrat of Vault

The Vault contract is responsible for storing the rToken of the user's stake and acts as a bridge between the bank contract and the strategies contracts.

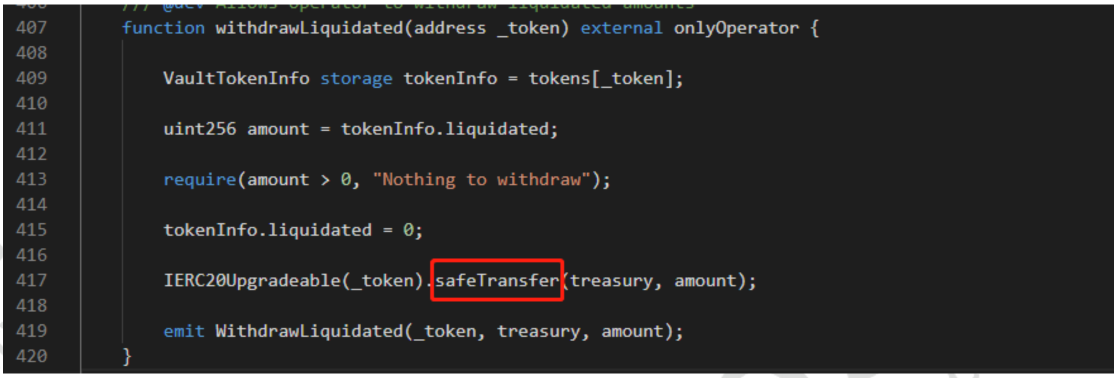
(1) It should use safeTransfer instead of safeTransferFrom here.



```
    /// @dev Allows operator to withdraw liquidated amounts
    function withdrawLiquidated(address _token) external onlyOperator {

        VaultTokenInfo storage tokenInfo = tokens[_token];

        uint256 amount = tokenInfo.liquidated;

        require(amount > 0, "Nothing to withdraw");

        tokenInfo.liquidated = 0;

        IERC20Upgradeable(_token).safeTransferFrom(address(this), roles[LIQUIDATED_TOKEN_RECEIVER], ar

        emit WithdrawLiquidated(_token, roles[LIQUIDATED_TOKEN_RECEIVER], amount);
    }
```

Figure 1 source code of withdraLiquidated

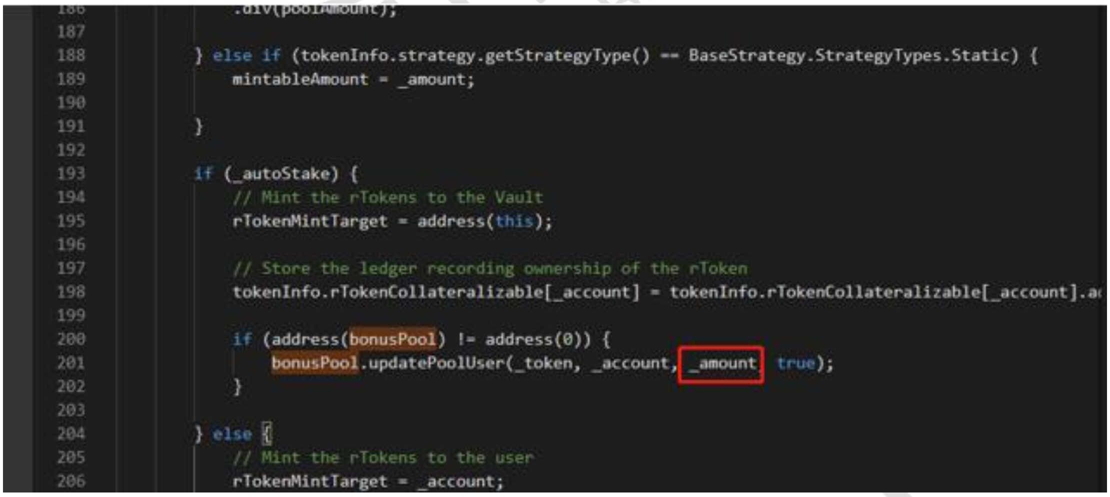
Fix result:



```
    /// @dev Allows operator to withdraw liquidated amounts
    function withdrawLiquidated(address _token) external onlyOperator {

        VaultTokenInfo storage tokenInfo = tokens[_token];

        uint256 amount = tokenInfo.liquidated;

        require(amount > 0, "Nothing to withdraw");

        tokenInfo.liquidated = 0;

        IERC20Upgradeable(_token).safeTransfer(treasury, amount);

        emit WithdrawLiquidated(_token, treasury, amount);
    }
```

Figure 2 fix result of withdraLiquidated

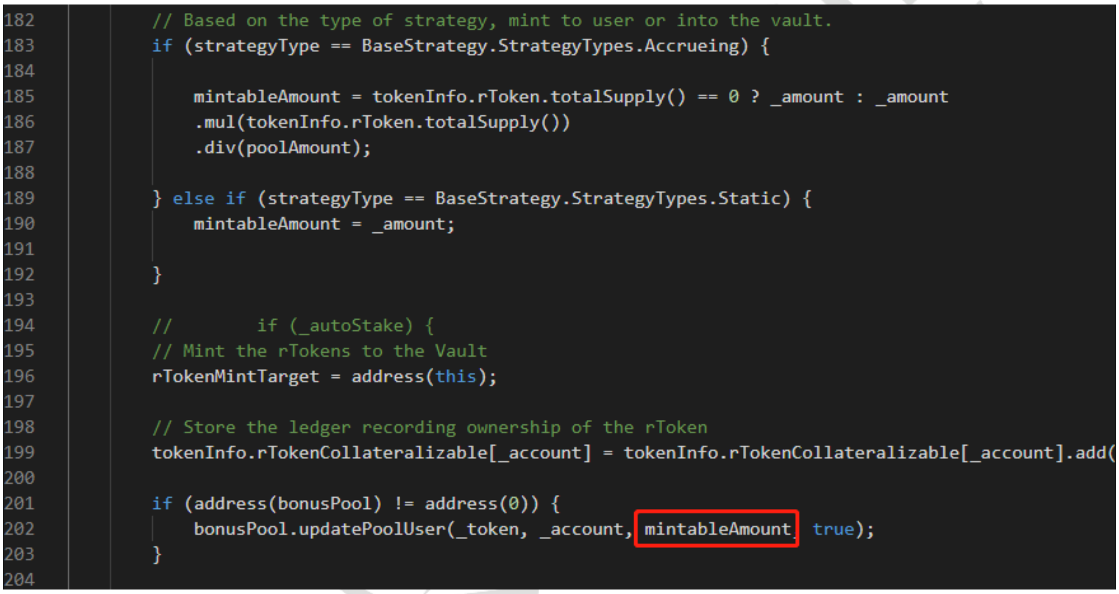
(2) The parameter in bonusPool.updatePoolUser should not use the value of the user's rToken, the value of rToken will increase and become greater than the value deposited by the user, which will cause the user to be unable to withdraw the deposited token



```
            .div(poolAmount);

        } else if (tokenInfo.strategy.getStrategyType() == BaseStrategy.StrategyTypes.Static) {
            mintableAmount = _amount;

        }

        if (_autoStake) {
            // Mint the rTokens to the Vault
            rTokenMintTarget = address(this);

            // Store the ledger recording ownership of the rToken
            tokenInfo.rTokenCollateralizable[_account] = tokenInfo.rTokenCollateralizable[_account].a

            if (address(bonusPool) != address(0)) {
                bonusPool.updatePoolUser(_token, _account, _amount, true);
            }

        } else {
            // Mint the rTokens to the user
            rTokenMintTarget = _account;
```

Figure 3 source code about updatePoolUser

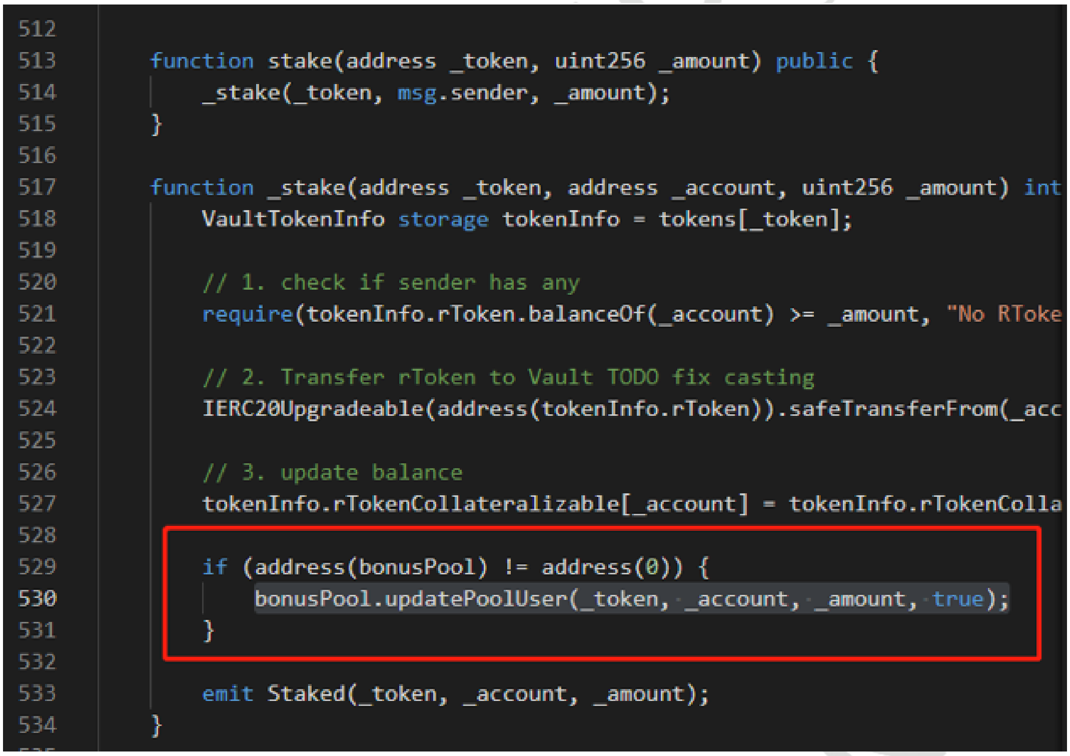
Fix result:



```
        // Based on the type of strategy, mint to user or into the vault.
        if (strategyType == BaseStrategy.StrategyTypes.Accrueing) {

            mintableAmount = tokenInfo.rToken.totalSupply() == 0 ? _amount : _amount
            .mul(tokenInfo.rToken.totalSupply())
            .div(poolAmount);

        } else if (strategyType == BaseStrategy.StrategyTypes.Static) {
            mintableAmount = _amount;

        }

        //        if (_autoStake) {
        // Mint the rTokens to the Vault
        rTokenMintTarget = address(this);

        // Store the ledger recording ownership of the rToken
        tokenInfo.rTokenCollateralizable[_account] = tokenInfo.rTokenCollateralizable[_account].add(

        if (address(bonusPool) != address(0)) {
            bonusPool.updatePoolUser(_token, _account, mintableAmount, true);
        }

```

Figure 4 fix result about updatePoolUser

(3) The data when the bonusPool is at zero address is not synchronized with when the bonusPool is not at zero, which will cause the user to be unable to withdraw the previously deposit tokens.

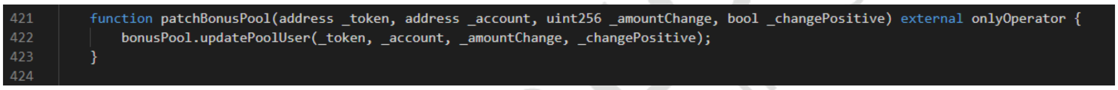


```
    function stake(address _token, uint256 _amount) public {
        _stake(_token, msg.sender, _amount);
    }

    function _stake(address _token, address _account, uint256 _amount) int
        VaultTokenInfo storage tokenInfo = tokens[_token];

        // 1. check if sender has any
        require(tokenInfo.rToken.balanceOf(_account) >= _amount, "No RToke

        // 2. Transfer rToken to Vault TODO fix casting
        IERC20Upgradeable(address(tokenInfo.rToken)).safeTransferFrom(_acc

        // 3. update balance
        tokenInfo.rTokenCollateralizable[_account] = tokenInfo.rTokenColla

        if (address(bonusPool) != address(0)) {
            bonusPool.updatePoolUser(_token, _account, _amount, true);
        }

        emit Staked(_token, _account, _amount);
    }
```

Figure 5 source code of _stake

Fix result:

On this issue, The ramp party inject a function that allows for syncing the user balances with the RToken balances, and the pool totals.

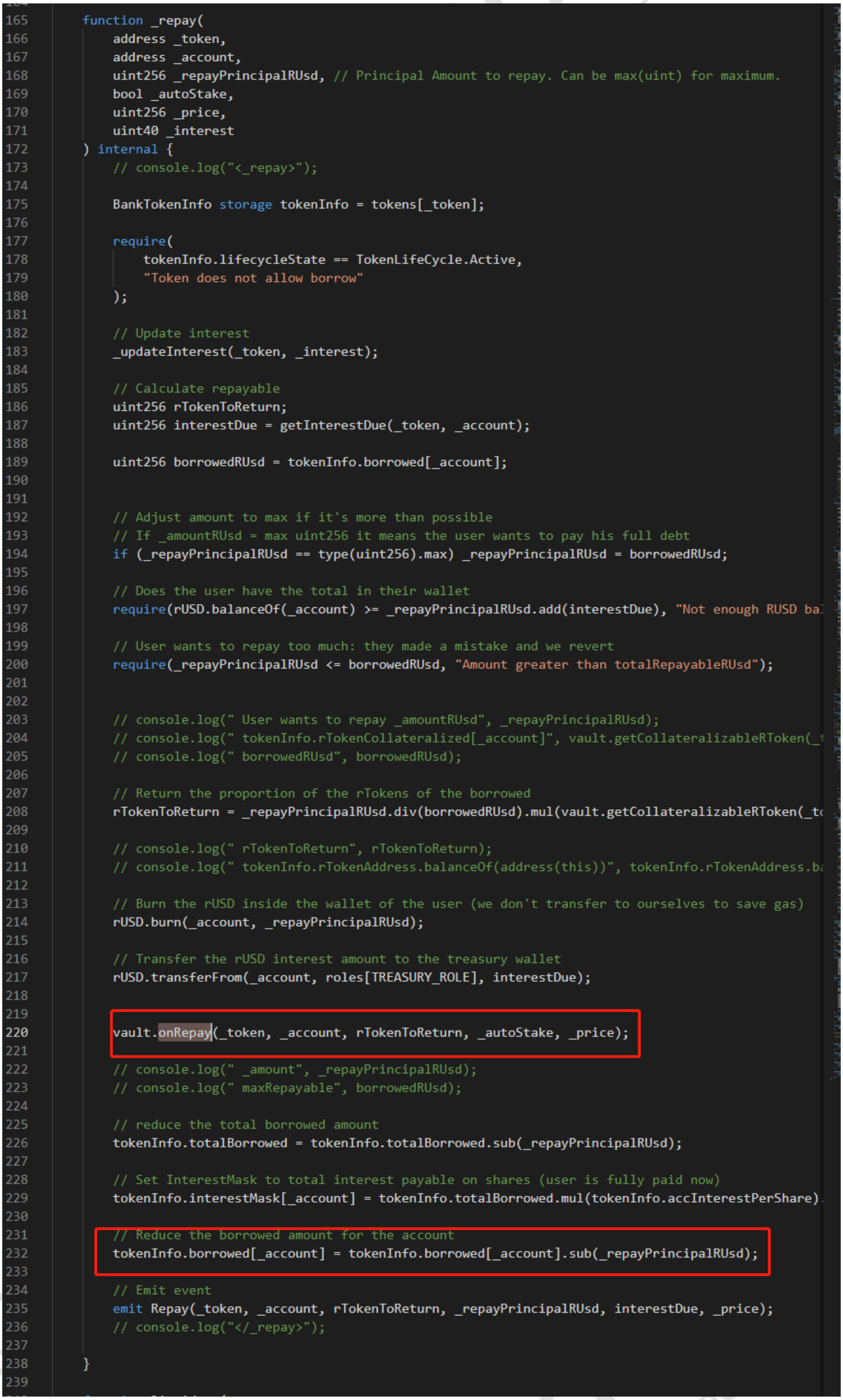


```
    function patchBonusPool(address _token, address _account, uint256 _amountChange, bool _changePositive) external onlyOperator {
        bonusPool.updatePoolUser(_token, _account, _amountChange, _changePositive);
    }
```

Figure 6 source code of patchBonusPool

3.3 Contrat of Bank

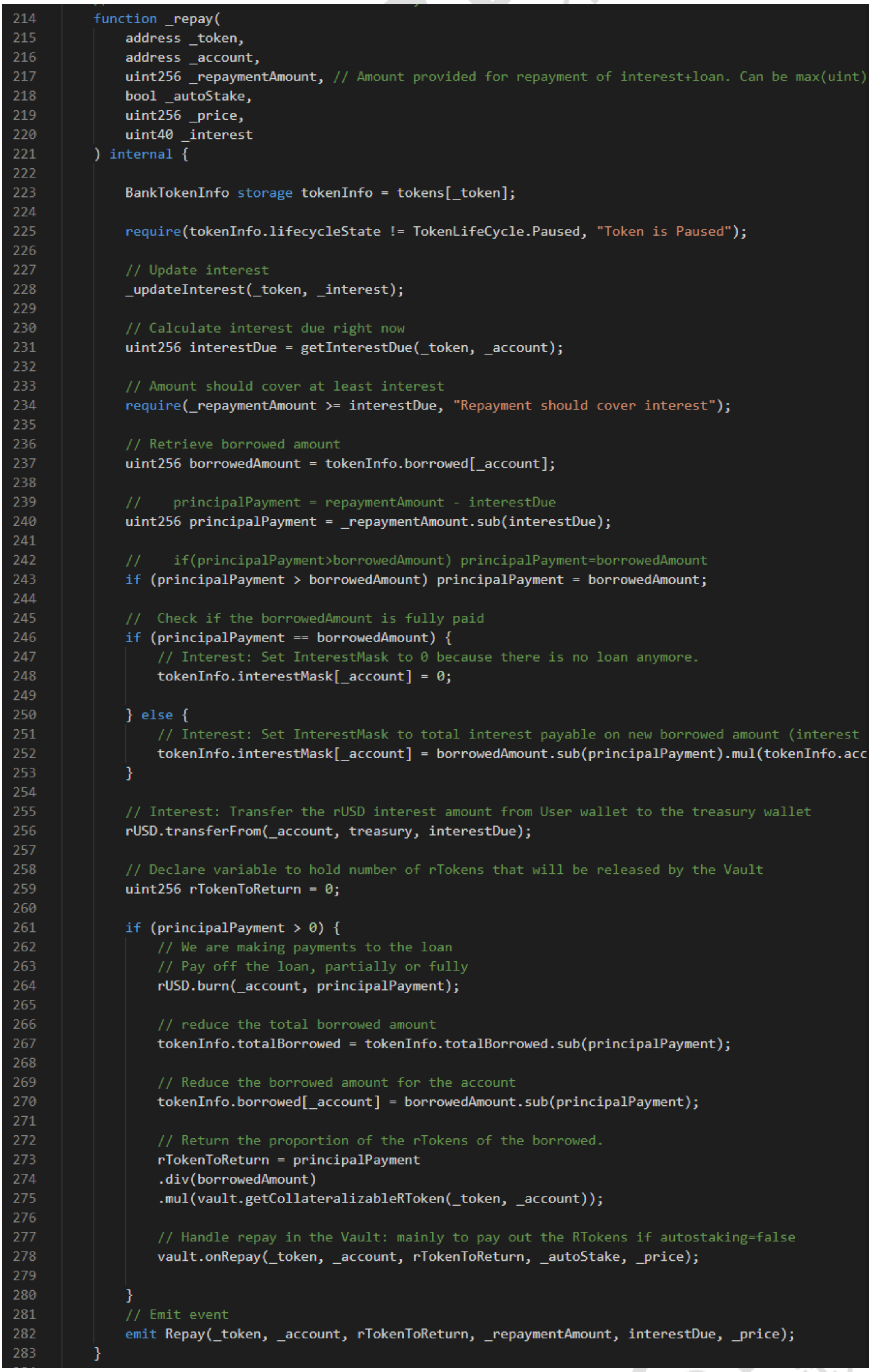
(1) The function vault.onRepay has function getMaxUnstakeable to calculate max unstake rToken, To ensure the correct calculation of getMaxUnstakeable, the borrowed value should be updated first



```
    function _repay(
        address _token,
        address _account,
        uint256 _repayPrincipalRUsd, // Principal Amount to repay. Can be max(uint) for maximum.
        bool _autoStake,
        uint256 _price,
        uint40 _interest
    ) internal {
        // console.log("<_repay>");

        BankTokenInfo storage tokenInfo = tokens[_token];

        require(
            tokenInfo.lifecycleState == TokenLifeCycle.Active,
            "Token does not allow borrow"
        );

        // Update interest
        _updateInterest(_token, _interest);

        // Calculate repayable
        uint256 rTokenToReturn;
        uint256 interestDue = getInterestDue(_token, _account);

        uint256 borrowedRUsd = tokenInfo.borrowed[_account];


        // Adjust amount to max if it's more than possible
        // If _amountRUsd = max uint256 it means the user wants to pay his full debt
        if (_repayPrincipalRUsd == type(uint256).max) _repayPrincipalRUsd = borrowedRUsd;

        // Does the user have the total in their wallet
        require(rUSD.balanceOf(_account) >= _repayPrincipalRUsd.add(interestDue), "Not enough RUSD ba

        // User wants to repay too much: they made a mistake and we revert
        require(_repayPrincipalRUsd <= borrowedRUsd, "Amount greater than totalRepayableRUsd");


        // console.log(" User wants to repay _amountRUsd", _repayPrincipalRUsd);
        // console.log(" tokenInfo.rTokenCollateralized[_account]", vault.getCollateralizableRToken(_t
        // console.log(" borrowedRUsd", borrowedRUsd);

        // Return the proportion of the rTokens of the borrowed
        rTokenToReturn = _repayPrincipalRUsd.div(borrowedRUsd).mul(vault.getCollateralizableRToken(_to

        // console.log(" rTokenToReturn", rTokenToReturn);
        // console.log(" tokenInfo.rTokenAddress.balanceOf(address(this))", tokenInfo.rTokenAddress.ba

        // Burn the rUSD inside the wallet of the user (we don't transfer to ourselves to save gas)
        rUSD.burn(_account, _repayPrincipalRUsd);

        // Transfer the rUSD interest amount to the treasury wallet
        rUSD.transferFrom(_account, roles[TREASURY_ROLE], interestDue);


        vault.onRepay(_token, _account, rTokenToReturn, _autoStake, _price);

        // console.log(" _amount", _repayPrincipalRUsd);
        // console.log(" maxRepayable", borrowedRUsd);

        // reduce the total borrowed amount
        tokenInfo.totalBorrowed = tokenInfo.totalBorrowed.sub(_repayPrincipalRUsd);

        // Set InterestMask to total interest payable on shares (user is fully paid now)
        tokenInfo.interestMask[_account] = tokenInfo.totalBorrowed.mul(tokenInfo.accInterestPerShare).

        // Reduce the borrowed amount for the account
        tokenInfo.borrowed[_account] = tokenInfo.borrowed[_account].sub(_repayPrincipalRUsd);

        // Emit event
        emit Repay(_token, _account, rTokenToReturn, _repayPrincipalRUsd, interestDue, _price);
        // console.log("</_repay>");

    }
```

Figure 7 source code of _repay

Fix result:

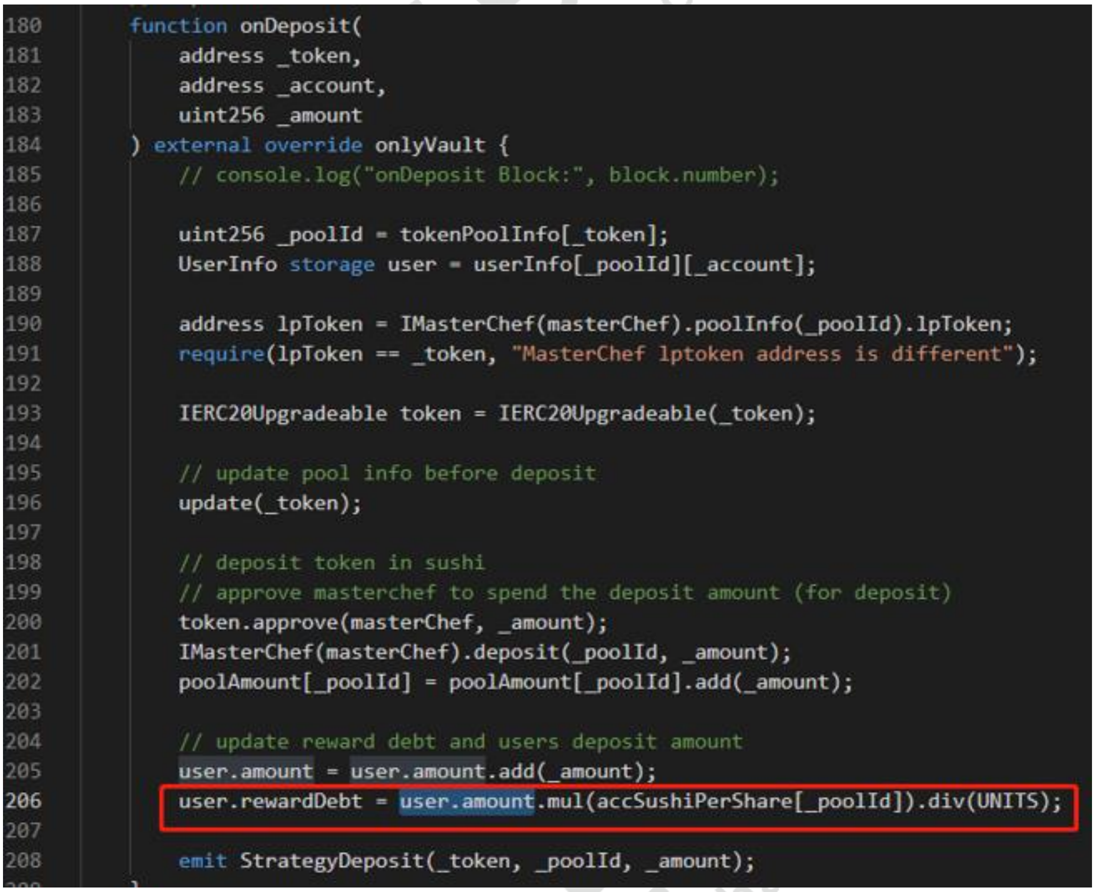


```
    function _repay(
        address _token,
        address _account,
        uint256 _repaymentAmount, // Amount provided for repayment of interest+loan. Can be max(uint)
        bool _autoStake,
        uint256 _price,
        uint40 _interest
    ) internal {

        BankTokenInfo storage tokenInfo = tokens[_token];

        require(tokenInfo.lifecycleState != TokenLifeCycle.Paused, "Token is Paused");

        // Update interest
        _updateInterest(_token, _interest);

        // Calculate interest due right now
        uint256 interestDue = getInterestDue(_token, _account);

        // Amount should cover at least interest
        require(_repaymentAmount >= interestDue, "Repayment should cover interest");

        // Retrieve borrowed amount
        uint256 borrowedAmount = tokenInfo.borrowed[_account];

        //    principalPayment = repaymentAmount - interestDue
        uint256 principalPayment = _repaymentAmount.sub(interestDue);

        //    if(principalPayment>borrowedAmount) principalPayment=borrowedAmount
        if (principalPayment > borrowedAmount) principalPayment = borrowedAmount;

        //  Check if the borrowedAmount is fully paid
        if (principalPayment == borrowedAmount) {
            // Interest: Set InterestMask to 0 because there is no loan anymore.
            tokenInfo.interestMask[_account] = 0;

        } else {
            // Interest: Set InterestMask to total interest payable on new borrowed amount (interest
            tokenInfo.interestMask[_account] = borrowedAmount.sub(principalPayment).mul(tokenInfo.acc
        }

        // Interest: Transfer the rUSD interest amount from User wallet to the treasury wallet
        rUSD.transferFrom(_account, treasury, interestDue);

        // Declare variable to hold number of rTokens that will be released by the Vault
        uint256 rTokenToReturn = 0;

        if (principalPayment > 0) {
            // We are making payments to the loan
            // Pay off the loan, partially or fully
            rUSD.burn(_account, principalPayment);

            // reduce the total borrowed amount
            tokenInfo.totalBorrowed = tokenInfo.totalBorrowed.sub(principalPayment);

            // Reduce the borrowed amount for the account
            tokenInfo.borrowed[_account] = borrowedAmount.sub(principalPayment);

            // Return the proportion of the rTokens of the borrowed.
            rTokenToReturn = principalPayment
            .div(borrowedAmount)
            .mul(vault.getCollateralizableRToken(_token, _account));

            // Handle repay in the Vault: mainly to pay out the RTokens if autostaking=false
            vault.onRepay(_token, _account, rTokenToReturn, _autoStake, _price);

        }
        // Emit event
        emit Repay(_token, _account, rTokenToReturn, _repaymentAmount, interestDue, _price);
    }
```

Figure 8 fix result of _repay

3.4 Contrat of strategies

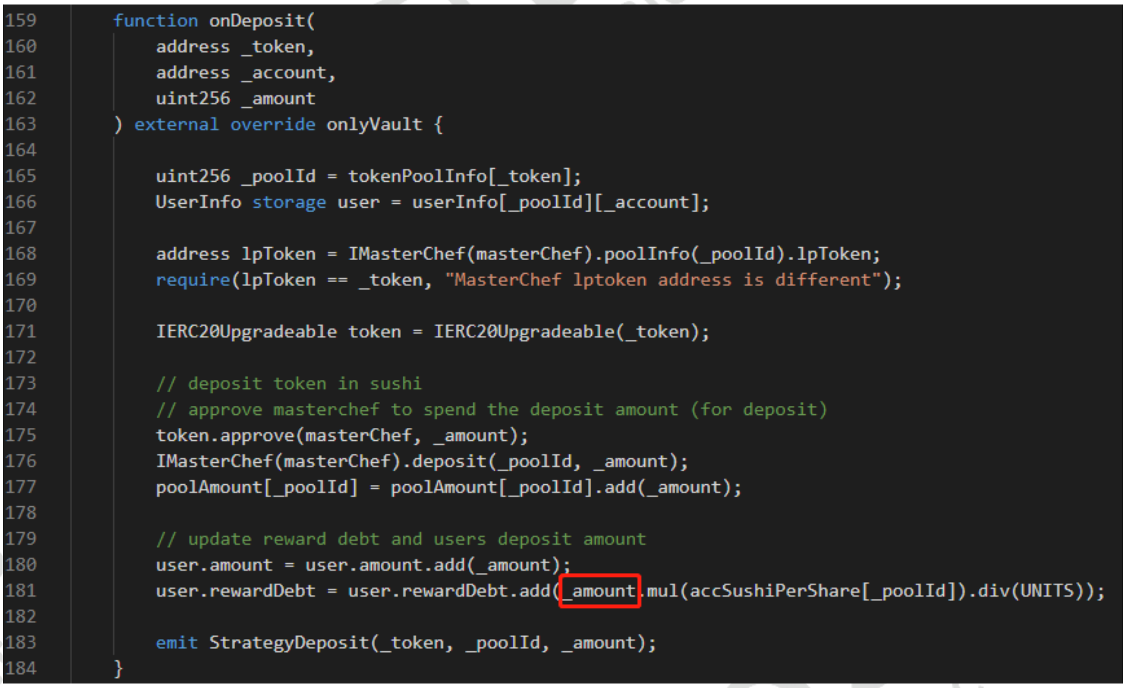
(1) In SushiLpStrategy.onDeposit, the _amount variable should be used instead of user.amount



```
    function onDeposit(
        address _token,
        address _account,
        uint256 _amount
    ) external override onlyVault {
        // console.log("onDeposit Block:", block.number);

        uint256 _poolId = tokenPoolInfo[_token];
        UserInfo storage user = userInfo[_poolId][_account];

        address lpToken = IMasterChef(masterChef).poolInfo(_poolId).lpToken;
        require(lpToken == _token, "MasterChef lptoken address is different");

        IERC20Upgradeable token = IERC20Upgradeable(_token);

        // update pool info before deposit
        update(_token);

        // deposit token in sushi
        // approve masterchef to spend the deposit amount (for deposit)
        token.approve(masterChef, _amount);
        IMasterChef(masterChef).deposit(_poolId, _amount);
        poolAmount[_poolId] = poolAmount[_poolId].add(_amount);

        // update reward debt and users deposit amount
        user.amount = user.amount.add(_amount);
        user.rewardDebt = user.amount.mul(accSushiPerShare[_poolId]).div(UNITS);

        emit StrategyDeposit(_token, _poolId, _amount);
```

Figure 9 source code of onDeposit

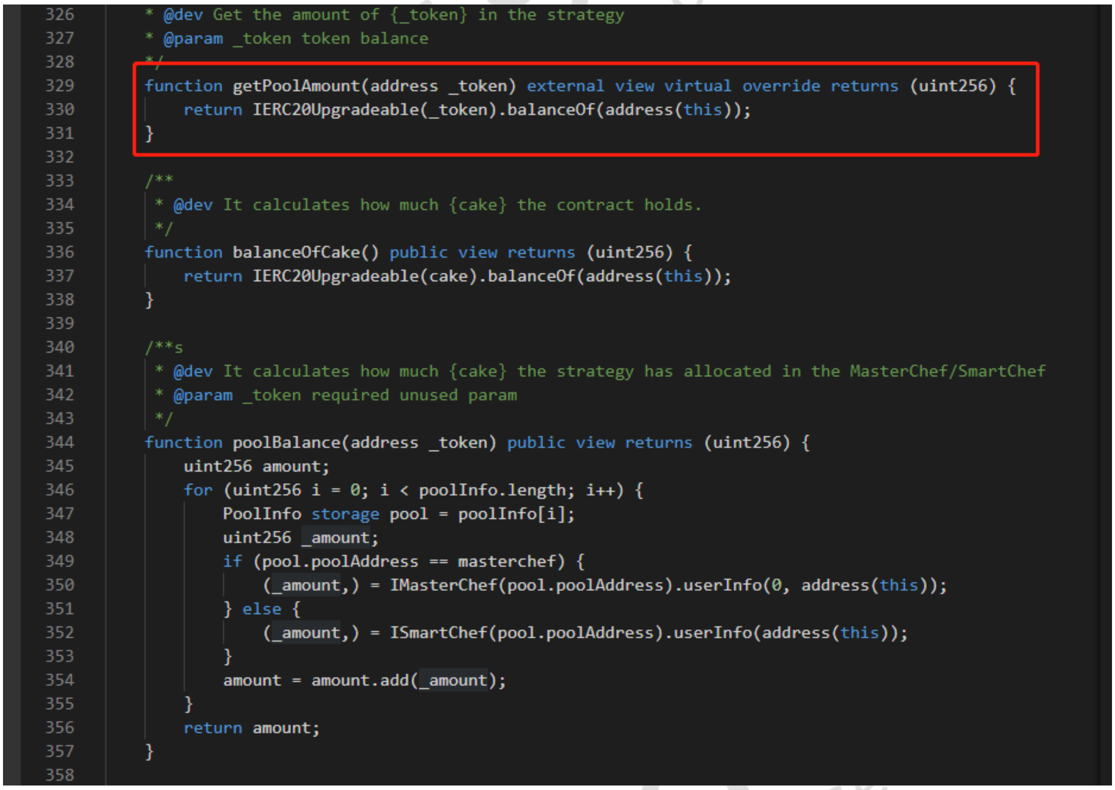
Fix result:



```
    function onDeposit(
        address _token,
        address _account,
        uint256 _amount
    ) external override onlyVault {

        uint256 _poolId = tokenPoolInfo[_token];
        UserInfo storage user = userInfo[_poolId][_account];

        address lpToken = IMasterChef(masterChef).poolInfo(_poolId).lpToken;
        require(lpToken == _token, "MasterChef lptoken address is different");

        IERC20Upgradeable token = IERC20Upgradeable(_token);

        // deposit token in sushi
        // approve masterchef to spend the deposit amount (for deposit)
        token.approve(masterChef, _amount);
        IMasterChef(masterChef).deposit(_poolId, _amount);
        poolAmount[_poolId] = poolAmount[_poolId].add(_amount);

        // update reward debt and users deposit amount
        user.amount = user.amount.add(_amount);
        user.rewardDebt = user.rewardDebt.add(_amount.mul(accSushiPerShare[_poolId]).div(UNITS));

        emit StrategyDeposit(_token, _poolId, _amount);
    }
```

Figure 10 fix result of onDeposit

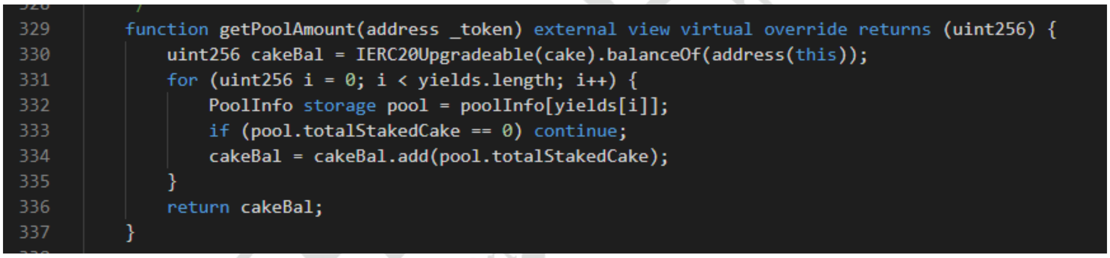
(2) The return value of getPoolAmount is wrong, the balance in the "pool.PoolAddress" contract is not added.



```
    * @dev Get the amount of {_token} in the strategy
    * @param _token token balance
    */
    function getPoolAmount(address _token) external view virtual override returns (uint256) {
        return IERC20Upgradeable(_token).balanceOf(address(this));
    }

    /**
     * @dev It calculates how much {cake} the contract holds.
     */
    function balanceOfCake() public view returns (uint256) {
        return IERC20Upgradeable(cake).balanceOf(address(this));
    }

    /**s
     * @dev It calculates how much {cake} the strategy has allocated in the MasterChef/SmartChef
     * @param _token required unused param
     */
    function poolBalance(address _token) public view returns (uint256) {
        uint256 amount;
        for (uint256 i = 0; i < poolInfo.length; i++) {
            PoolInfo storage pool = poolInfo[i];
            uint256 _amount;
            if (pool.poolAddress == masterchef) {
                (_amount,) = IMasterChef(pool.poolAddress).userInfo(0, address(this));
            } else {
                (_amount,) = ISmartChef(pool.poolAddress).userInfo(address(this));
            }
            amount = amount.add(_amount);
        }
        return amount;
    }
```

Figure 11 source code of getPoolAount

Fix result:



```
    function getPoolAmount(address _token) external view virtual override returns (uint256) {
        uint256 cakeBal = IERC20Upgradeable(cake).balanceOf(address(this));
        for (uint256 i = 0; i < yields.length; i++) {
            PoolInfo storage pool = poolInfo[yields[i]];
            if (pool.totalStakedCake == 0) continue;
            cakeBal = cakeBal.add(pool.totalStakedCake);
        }
        return cakeBal;
    }
```

Figure 12 fix result of getPoolAmount

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project ramp-protocol. The problems found by the audit team during the audit process have been notified to the project party and reached an agreement on the repair results, the overall audit result of the ramp-protocol project's smart contract is **Pass**.

BEOSIN
Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com