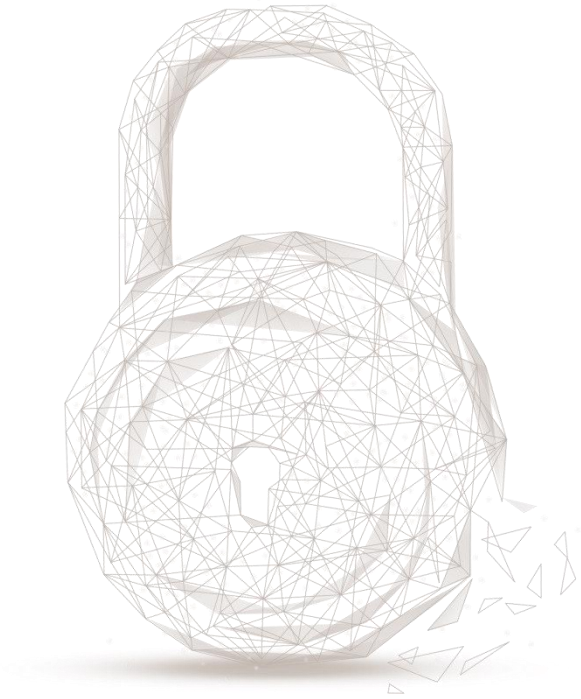
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number: 202107151414

Report Query Name: RampBunnyStrategy

Smart Contract Address Link:

https://github.com/RAMP-DEFI/ramp-protocol/blob/development/contracts/strategies/bsc/RampBunnyStrategy.sol

Commit Hash:

Start: e6dea339bc347bf1294dd79e93e4724118874423

Final: d48e2cc0dae8b6db4097ced4fa501ad2d594f5e9

Start Date: 2021.06.25

Completion Date: 2021.07.15

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

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 contract RampBunnyStrategy, including Coding Standards, Security, and Business Logic. **The RampBunnyStrategy contract 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 BNB.
- 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

This contract is a strategic contract of vault and bank contracts, used to specify assets to invest in Bunny projects.

(1) onDeposit

- Description: This function is used to trigger when the vault contract deposits the specified asset. The vault contract will trigger this function to invest. This function will send the funds to BUNNY_VAULT contract for investment.

```
132   function onDeposit(address _token, address _account, uint256 _amount) external override onlyVault{
133       // UNDERLYING_ASSET.safeTransferFrom(_account, address(this), _amount);
134       require(_amount > 0, "Amount cannot be 0");
135       uint256 amount = UNDERLYING_ASSET.balanceOf(address(this));
136       BUNNY_VAULT.deposit(amount);
137       update(_token);
138       emit StrategyDeposit(_amount);
139   }
```

Figure 1 Source code of function *onDeposit*

- Related functions: *deposit*, *update*
- Result: Pass

(2) onWithdraw

- Description: This function is used to trigger when the Vault contract withdraws money. The vault contract specifies the recipient and the withdrawal amount. This contract will take out the corresponding amount of funds from the invested project and send it to the user.

```
148   function onWithdraw(address, address _account, uint256 _amount) external override onlyVault {
149       //IERC20Upgradeable token = IERC20Upgradeable(_token);
150       uint256 underlyingBal = UNDERLYING_ASSET.balanceOf(address(this));
151       if (underlyingBal < _amount) {
152         uint256 unstakeAmount = BUNNY_VAULT.balanceOf(address(this)) < _amount.sub(underlyingBal) ? BUNNY_VAULT.balanceOf(address(this)) : _amount.sub(underlyingBal);
153         BUNNY_VAULT.withdrawUnderlying(unstakeAmount);
154       }
155
156       underlyingBal = UNDERLYING_ASSET.balanceOf(address(this));
157
158       if (underlyingBal > _amount) {
159         underlyingBal = _amount;
160       }
161
162       // withdrawal fee
163       uint256 fee = underlyingBal.mul(withdrawalFeePercentage).div(100);
164
165       // transfer underlying amount to user
166       UNDERLYING_ASSET.safeTransfer(_account, underlyingBal.sub(fee));
167
168       // withdrawal fee token to devaddress
169       if (fee > 0) UNDERLYING_ASSET.safeTransfer(devAddress, fee);
170
171       emit StrategyWithdraw(_account, underlyingBal.sub(fee), fee);
172
173   }
```

Figure 2 Source code of function *onWithdraw*

- Related functions: *withdrawUnderlying*, *safeTransfer*
- Result: Pass

(3) onLiquidate

- Description: This function is used to trigger when the bank contract liquidate, the vault contract will specify the amount of funds withdrawn, and the contract will withdraw the funds according to the specified amount and send them to the vault contract.

```
function onLiquidate(address _token, address _account, uint256 _amount) external override onlyVault {
  uint256 underlyingBal = UNDERLYING_ASSET.balanceOf(address(this));
  if (underlyingBal < _amount) {
    uint256 unstakeAmount = BUNNY_VAULT.balanceOf(address(this)) < _amount.sub(underlyingBal) ? BUNNY_VAULT.balanceOf(address(this)) : _amount.sub(underlyingBal);
    BUNNY_VAULT.withdrawUnderlying(unstakeAmount);
  }

  underlyingBal = UNDERLYING_ASSET.balanceOf(address(this));

  if (underlyingBal > _amount) {
    underlyingBal = _amount;
  }

  UNDERLYING_ASSET.safeTransfer(vault, underlyingBal);

  // Emit event for logging
  emit Liquidated(underlyingBal);
}
```

Figure 3 Source code of function *onLiquidate*

- Related functions: *withdrawUnderlying*, *safeTransfer*
- Result: Pass

(4) emergencyWithdraw

- Description: This function is used to trigger when the vault contract is withdrawn in an emergency. This function will withdraw all the investment and send it to the vault contract.

```
function emergencyWithdraw(address _token, bool _abandonRewards) external override onlyVault returns (uint256 lastPoolAmount) {
  // call update function if we are not _abandonRewards
  if (!_abandonRewards) {
    update(_token);
  }
  BUNNY_VAULT.withdrawUnderlying(getPoolAmount(_token));
  uint256 underlyingBal = UNDERLYING_ASSET.balanceOf(address(this));
  UNDERLYING_ASSET.transfer(vault, underlyingBal);
  lastPoolAmount = underlyingBal;
  // check whether any reward token is available. if so Transfer
  uint256 rewardBal = BUNNY.balanceOf(address(this));
  BUNNY.transfer(vault, rewardBal);
  emit EmergencyWithdraw(_token, underlyingBal, rewardBal);
}
```

Figure 4 Source code of *emergencyWithdraw*

- Related functions: *update*, *withdrawUnderlying*, *getPoolAmount*, *transfer*, *balanceOf*
- Result: Pass

(5) update

- Description: Anyone can call this function at any time, and this function will settle the investment reward to this contract.

```
function update(address _token) public override {
  // Get earned rewards from Bunny vault, BUNNY + Underlying Asset
  BUNNY_VAULT.getReward();
  // Swap BUNNY -> Underlying Asset to re-stake
  _swapRewards();

  uint256 amount = UNDERLYING_ASSET.balanceOf(address(this));
  if (amount > 0) BUNNY_VAULT.deposit(amount);

  lastHarvested = block.timestamp;
}
```

Figure 5 Source code of *update*

- Related functions: *getReward*, *_swapRewaeds*, *deposit*
- Result: Pass

4. Conclusion

Beosin(Chengdu LianAn) conducted a detailed audit on the design and code implementation of the smart contract RampBunnyStrategy. The contract RampBunnyStrategy passed all audit items, The overall audit result is **Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com