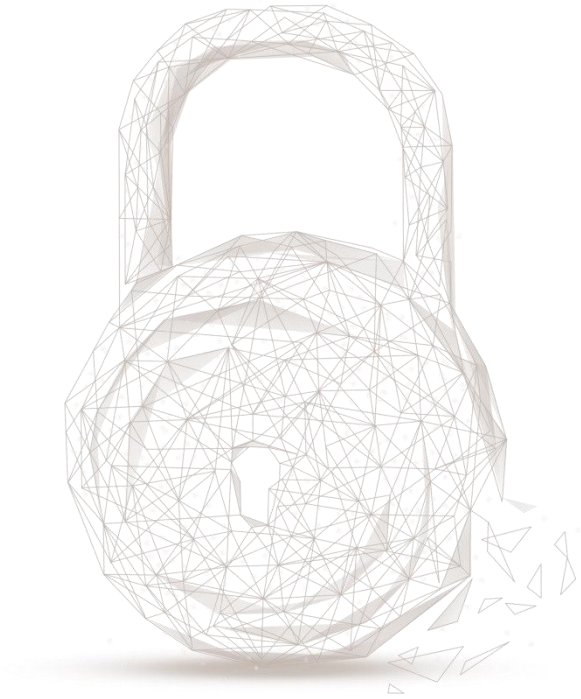
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202106111131

Report Query Name：QuickswapAccruingStrategy

Smart Contract Address Link:

https://github.com/RAMP-DEFI/strategy-quickswap-accruing/blob/main/contracts/strategies/polygon/QuickswapAccruingStrategy.sol

Commit Hash:

Start: 17836bc5dee3cbb920d7636cdffaf97d3bd442d0

Final: 04e2033f4baf2985d69f3d49c73963846b5fdd61

Start Date：2021.06.03

Completion Date：2021.06.11

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 QuickswapAccruingStrategy, including Coding Standards, Security, and Business Logic. **The QuickswapAccruingStrategy 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 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

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

(1) onDeposit

- Description: This function is used to trigger when the vault contract deposits the specified asset. The vault contract will inform the function of the amount to be invested. This function will send the funds to maticEthStakingPool contract for investment.

```
80      function onDeposit(
81          address,                // _token unused
82          address,                // _account unused
83          uint256 _amount
84      ) external override onlyVault {
85          // 1. Ensure _amount is greater than 0
86          require(_amount > 0, "Deposit amount cannot be 0");
87          // 2. Approve that the maticEthStakingPool can transfer the maticEthlpTokens from our address
88          maticEthlpToken.approve(address(maticEthStakingPool), _amount);
89          // 3. Stake the _amount in the maticEthStakingPool
90          maticEthStakingPool.stake(_amount);
91      }
```

Figure 1 Source code of function *onDeposit*

- Related functions: *approve*, *stake*
- 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.

```
122     function onWithdraw(
123         address,                // _token unused
124         address _account,
125         uint256 _amount
126     ) external override onlyVault {
127         // 1. Ensure _amount is greater than 0
128         require(_amount > 0, "Withdrawal amount cannot be 0");
129         // 2. Withdraw the _amount from the maticEthStakingPool to the strategy
130         maticEthStakingPool.withdraw(_amount);
131         // 3. Calculate the withdrawal fee (if any)
132         uint256 fee = _amount.mul(withdrawalFeePercentage).div(100);
133         // 4. Transfer lpTokens to the user
134         maticEthlpToken.safeTransfer(_account, _amount.sub(fee));
135         // 5. Transfer withdrawal fee token to dev
136         if (fee > 0) maticEthlpToken.safeTransfer(devAddress, fee);
137     }
```

Figure 2 Source code of function *onWithdraw*

- Related functions: *withdraw*, *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.

```
 99      function onLiquidate(
100          address, // _token unused
101          address, // _account unused
102          uint256 _amount
103      ) external override onlyVault {
104          // 1. Withdraw the _amount from the maticEthStakingPool to the strategy
105          maticEthStakingPool.withdraw(_amount);
106          // 2. Send _amount to vault
107          maticEthlpToken.safeTransfer(vault, _amount);
108          // 3. Emit event for logging
109          emit Liquidated(_amount);
110      }
```

Figure 3 Source code of function *onLiquidate*

- Related functions: *withdraw*, *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.

```
247      function emergencyWithdraw(
248          address, // _token unused
249          bool // _abandonRewards unused
250      ) external override onlyVault returns (uint256 lastPoolAmount) {
251          // 1. Withdraw funds from maticEthStakingPool
252          maticEthStakingPool.exit();
253          // We should have received lpTokens and possibly quick
254          // We will transfer the lpTokens to the vault
255          // and the Quick tokens to the devAddress
256          // 2. Transfer lpTokens (if any)
257          uint256 maticEthlpTokenBalance = maticEthlpToken.balanceOf(address(this));
258          if (maticEthlpTokenBalance > 0) {
259              maticEthlpToken.safeTransfer(vault, maticEthlpTokenBalance);
260          }
261          // 3. Transfer rewardTokens (if any)
262          uint256 quickTokenBalance = quickRewardToken.balanceOf(address(this));
263          if (quickTokenBalance > 0) {
264              quickRewardToken.safeTransfer(devAddress, quickTokenBalance);
265          }
266          // 4. Emit the EmergencyWithdraw event
267          emit EmergencyWithdraw(address(maticEthlpToken), maticEthlpTokenBalance, quickTokenBalance);
268          // 5. Return maticEthlpTokenBalance
269          return maticEthlpTokenBalance;
270      }
```

Figure 4 Source code of *emergencyWithdraw*

- Related functions: *withdraw*, *exit*, *safeTransfer*
- Result: Pass

(5) work

- Description: Anyone can call this function at any time. This function will redeem the quickRewardToken in contract for reinvestment.

```
144     function work(
145         address // _token unused
146     ) public override {
147         // 1. Ensure we have some rewards to reinvest
148         uint256 rewardBalance = quickRewardToken.balanceOf(address(this));
149         if (rewardBalance == 0) return;
150 
151         // 2. If applicable transfer reward fee to devAddress
152         if (devFeePercentage > 0) {
153             uint256 devFee = rewardBalance.mul(devFeePercentage).div(100);
154             quickRewardToken.safeTransfer(devAddress, devFee);
155             rewardBalance = rewardBalance.sub(devFee);
156         }
157 
158         // A. We need to swap half the quickRewardToken to wMatic
159         // B. And we need to swap the other half to wETH
160         // C. Then we provide liquidity with wMATIC / wETH in exchange for lpTokens
161         // D. And finally we stake the lpTokens in the maticEthStakingPool
162 
163         // A1. Approve to spend quick
164         // Example: https://explorer-mainnet.maticvigil.com/tx/0x4e263ea48ed065de04ac06e42b2100eefd59f63e73f77c239c3063027b618733
165         uint256 halfRewardBalance = rewardBalance.div(2);
166 
167         // A2. Approve the full rewardBalance for 2 transactions
168         require(quickRewardToken.approve(address(quickswapRouter), rewardBalance), 'work approve failed');
169 
170         // A3. Build path to swap quick to matic
171         address[] memory quickToMaticPath = new address[](2);
172         quickToMaticPath[0] = address(quickRewardToken);
173         quickToMaticPath[1] = address(wMaticToken);
174 
175         // A4. Swap Quick for wMatic.
176         quickswapRouter.swapExactTokensForTokens(halfRewardBalance, 0, quickToMaticPath, address(this), block.timestamp.add(600));
177         uint256 wMaticBalance = wMaticToken.balanceOf(address(this));
178 
179         // B1. Approve to spend quick not needed anymore. Build path to swap quick to ether.
180         address[] memory quickToEthPath = new address[](2);
181         quickToEthPath[0] = address(quickRewardToken);
182         quickToEthPath[1] = address(wEthToken);
183 
184         // B2. Swap Quick for wrapped Ether.
185         quickswapRouter.swapExactTokensForTokens(halfRewardBalance, 0, quickToEthPath, address(this), block.timestamp.add(600));
186         uint256 wEthBalance = wEthToken.balanceOf(address(this));
187 
188         // C1. Then we provide liquidity with wMATIC / wETH in exchange for lpTokens
189         require(wMaticToken.approve(address(quickswapRouter), wMaticBalance), 'work wmatic approve failed');
190         require(wEthToken.approve(address(quickswapRouter), wEthBalance), 'work weth approve failed');
191         quickswapRouter.addLiquidity(address(wMaticToken), address(wEthToken), wMaticBalance, wEthBalance, 0, 0, address(this), block.timestamp.add(600));
192         uint256 maticEthlpTokenBalance = maticEthlpToken.balanceOf(address(this));
193 
194         // D1. Stake the maticEthlpTokenBalance
195         require(maticEthlpToken.approve(address(maticEthStakingPool), maticEthlpTokenBalance), 'work maticethlptoken approve failed');
196         maticEthStakingPool.stake(maticEthlpTokenBalance);
197     }
```

Figure 5 Source code of *work*

- Related functions: *safeTransfer*, *swapExactTokensForTokens*,, *addLiquidity*, *stake*
- Result: Pass

(6) update

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

```
204     function update(
205         address     // _token unused
206     ) public override {
207         // 1. Do not call update multiple times on the same block
208         if (block.number <= lastUpdateBlock) return;
209         // 2. Update the lastUpdateBlock
210         lastUpdateBlock = block.number;
211         // 3. Do not continue if poolAmount == 0
212         if (maticEthStakingPool.balanceOf(address(this)) == 0) return;
213         // 4. Retrieve rewards from the maticEthStakingPool
214         maticEthStakingPool.getReward();    // This retrieves the reward to the strategy wallet
215         // 5. We have hopefully received some reward. Call work() to reinvest the reward
216         work(address(0));
217     }
```

Figure 5 Source code of *update*

- Related functions: *getReward*, *work*
- Result: Pass

4. Conclusion

Beosin(Chengdu LianAn) conducted a detailed audit on the design and code implementation of the smart contract QuickswapMaticEthAccruingStrategy. The contract QuickswapMaticEthAccruingStrategy 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