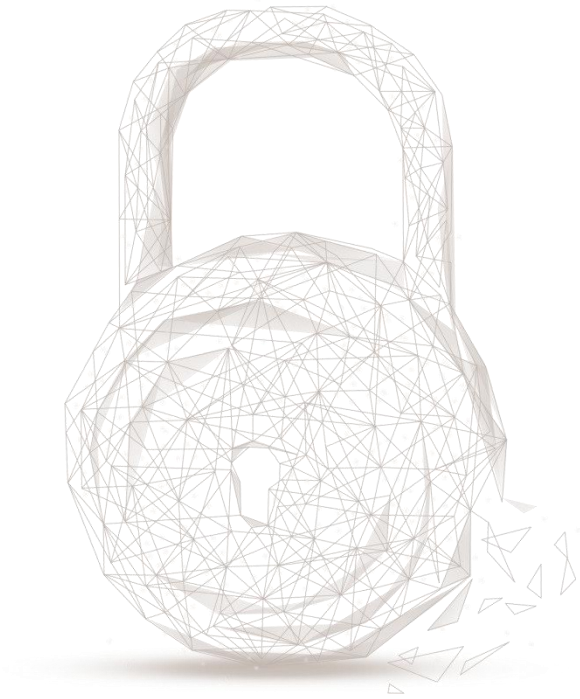
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202012281636

Report Query Name：freeliquid

Audit Project Name：freeliquid

Audit Project Contract Info：

Contract Name	Contract Address
FL	0xfFED56a180f23fD32Bc6A1d8d3c09c283aB594A8
USDFL	0x2B4200A8D373d484993C37d63eE14AeE0096cd12
StakingRewards	0x1a48B6151012a27A4ab2a8c1b8Ec108bAB9eF49c
StakingRewardsDecay(1)	0x975Aa6606f1e5179814BAEf22811441C5060e815
StakingRewardsDecayHolder(1)	0x34e2B546D1819fE428c072080829028aF36540DD
StakingRewardsDecay(2)	0x5E4935fe0f1f622bfc9521c0e098898e7b8b573c
StakingRewardsDecayHolder(2)	0x001F7C987996DBD4f1Dba243b0d8891D0Bf693A2
RewardDecayAggregator	0x51DB1Da6635578B9186B26871038F18351CDD527
GemForRewardChecker	0x096835f967D22EC35b78F887c3e9b936b84A3aF7
UniForRewardCheckerMainnet	0x933B0d1C324f6703536E888ce8C42175e8474283
PriceProvider	0x8177E21B333c7488993D89c11f889D78F1eADAE5
UniswapAdapterForStables	0x81f6E65493f430D520669E2139F96036102C5331
UniswapAdapterWithOneStable	0xC3dc053e111cA40f148C6E278B180C6F29742569
UniswapAdapterPriceOracle_Buck_Buck	0xc0FbaEeb737487A5B8990515d7eB6AFb404692E7
GemJoinWithReward(1)	0xef9564d9Ed617173e0c257D08B1EEB90E0e1cF28
GemJoinWithReward(2)	0x8c0929691A458f454cf3438Cf2EF8Bc901a72CcA
GemJoinWithReward(3)	0x1B9C400E36239c2649391c0179D9C3799c94fA6F
GemJoinWithReward(4)	0x18C480a97c5F36d6bB185741ad5df9ab9361050A
UniswapAdapterPriceOracle_USDT_Buck(1)	0x826e64E15af1CdcEd00032E985Ee51918397E60F
UniswapAdapterPriceOracle_USDT_Buck(2)	0x81CdB7EB973489526370141A7E3564211dC37Ad8
UniswapAdapterPriceOracle_USDT_Buck(3)	0x85FE3913Bc913f5C67B9AE3B7cc2785746979fec

Start Date：2020.12.03

Completion Date：2020.12.28

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

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

  The smart contracts of this project specify their corresponding minimum compiler version. Among them, using this version of the compiler to compile the uni contract (src/uni.sol), the compiler warning shown in the figure below will occur.

```
406 contract UniForRewardCheckerBase {
407     mapping(address => bool) public tokens;
408
409     function check(address gem) external {
410         address t0 = UniswapV2PairLike(gem).token0();
browser/usdfl/uni.sol:409:5: Warning: Function state mutability can be restricted to view
    function check(address gem) external {
    ^ (Relevant source part starts here and spans across multiple lines).
414     }
415 }
416
```

Figure 1 The compiler warning of uni contract

- Safety Recommendation: Modify the code to eliminate compiler warnings.
- Fix Result: Ignored

- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Safety Recommendation: None
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.

As shown in the figure below, the *StopRewarding* event is declared in the reward contract in the project, but this event is not emitted in the contract, which is redundant code.

```
76      event RewardAdded(uint256 reward);
77      event StopRewarding();
78      event Staked(address indexed user, address indexed gem, uint256 amount);
```

Figure 2 Declaration code of StopRewarding event

- Safety Recommendation: It is recommended to delete it.
- Fix Result: Ignored. It does not affect contract business logic.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.

Most of the mathematical operations in the contract of this project use anti-overflow libraries (such as SafeMath, math, etc.). Even if some business functions do not use functions in the SafeMath library, overflow checks will be performed on the corresponding variable values, as shown in the figure below, It shows that the deduction of line 277 in the *approveEpochsConsistency* function does not use the *sub* function in the SafeMath library, but the corresponding variable *i* has been restricted in line 274, indicating that *i* is an integer not less than 1, therefore, there is no overflow risk here.

```
267     function approveEpochsConsistency() public {
268         require(deployer == msg.sender);
269         require(epochInited == 0, "double call not allowed");
270
271         uint256 totalReward = epochs[0].initreward;
272         require(getStartTime() > 0);
273
274         for (uint256 i = 1; i < EPOCHCOUNT; i++) {
275             EpochData storage epoch = epochs[i];
276             require(epoch.starttime > 0);
277             require(epoch.starttime == epochs[i - 1].periodFinish);
278             totalReward = totalReward.add(epoch.initreward);
279         }
280
281         require(IERC20(gov).balanceOf(address(this)) >= totalReward, "GOV balance not enought");
282
283         epochInited = EPOCHCOUNT;
284     }
```

Figure 3 Source code of function approveEpochsConsistency

- Safety Recommendation: None
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Safety Recommendation: None
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.

The current audit test shows that the contract of this project only may have the issue ‘out of gas’ when the number of for loop variables is large. When registering gems, the *registerGem* function shown in the figure below will be called to check the currently registered gems. Considering that the contract cannot remove the gems, if the number of gems recorded by the contract is too large, there may be a large number of loop in the *registerGem* function. This makes the gas consumption of this function too large, causing the contract to fail to register gems.

```
287     function registerGem(address gem) internal {
288         for (uint256 i = 0; i < registeredGems.length; i++) {
289             if (registeredGems[i] == gem) {
290                 return;
291             }
292         }
293         registeredGems.push(gem);
294     }
```

Figure 4 Source code of function registerGem

- Safety Recommendation: Adding function of unregistering gem.
- Fix Result: Ignored. Considering the actual use, the gem number would not be too much, and can be controlled by the contract administrator.

- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Safety Recommendation: None
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Safety Recommendation: None
- 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.
- Safety Recommendation: None
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Safety Recommendation: None
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Safety Recommendation: None
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Safety Recommendation: None
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Safety Recommendation: None
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Safety Recommendation: None
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Safety Recommendation: None
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Safety Recommendation: None
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract. In this project, the contract
- Safety Recommendation: None
- Result: Pass

2.10 Replay Attack

- Description: Check the weather the implement possibility of Replay Attack exists in the contract.

The USDFL contract of this project implements function of approving by signature, that is, users can approve USDFL tokens to other users through signatures. As shown in the figure below, the signature information of this function sets a timeout; and the nonce of the account is checked.

```
118     function permit(address holder, address spender, uint256 nonce, uint256 expiry,
119                     bool allowed, uint8 v, bytes32 r, bytes32 s) external
120     {
121         bytes32 digest =
122             keccak256(abi.encodePacked(
123                 "\x19\x01",
124                 DOMAIN_SEPARATOR,
125                 keccak256(abi.encode(PERMIT_TYPEHASH,
126                                      holder,
127                                      spender,
128                                      nonce,
129                                      expiry,
130                                      allowed))
131         ));
132
133         require(holder != address(0), "Dai/invalid-address-0");
134         require(holder == ecrecover(digest, v, r, s), "Dai/invalid-permit");
135         require(expiry == 0 || now <= expiry, "Dai/permit-expired");
136         require(nonce == nonces[holder]++, "Dai/invalid-nonce");
137         uint wad = allowed ? uint(-1) : 0;
138         allowance[holder][spender] = wad;
139         emit Approval(holder, spender, wad);
140     }
141 }
```

Figure 5 Source code of function permit

- Safety Recommendation: None
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Safety Recommendation: None
- Result: Pass

3. Business Audit

3.1 The USDFL Contract Audit

3.1.1 Basic token information of USDFL

- Description: According to the project architecture, the USDFL contract implements a basic ERC20 token as the governance token, and its basic information is as follows:

Token name	USDFreeLiquidity
Token symbol	USDFL
decimals	18
totalSupply	Initial supply is 0 (Mintable; Burnable)
Token type	ERC20

Table 1 Basic Token Information

3.1.2 USDFL Token Functions

- Description: This contract token implements the basic functions of ERC20 standard tokens, and token holders can call corresponding functions for token transfer, approve and other operations.
- Related functions: *name, symbol, decimals, balanceOf, transfer, transferFrom, allowance, approve*
- Result: Pass

3.1.3 USDFL Token burning

- Description: Users who hold the tokens of this contract can call the *burn* function to destroy their specific number of tokens. Or approve to other addresses to delegate destroy their tokens.
- Related functions: *burn*
- Safety Suggestion: None
- Result: Pass

3.1.4 Contract Management

- Description: This contract declares the management authority, which will be granted to the deployer's address when the contract is deployed. The address with this permission can call the functions *rely* and *deny* to add and remove administrators.

As shown in Figure 7 and 8, combined with the actual deployment environment of the project on the main network, the USDFL contract(0x2B4200A8D373d484993C37d63eE14AeE0096cd12) is deployed by the DssDeploy contract(0x6a91178174995d6f43E3D29d57dC7D82b4c7EF15) by calling the function deployDai. According to the relevant code logic, after the USDFL contract is deployed, the DssDeploy contract and the DaiJoin contract(0x1856298fAD423F63158A3ED1c7d98490840E6C14) have corresponding management permissions of USDFL contract; however, the DssDeploy contract also has the *releaseAuth* function to release the management permission of USDFL contract (Figure 9), and the administrator of the DssDeploy contract has already called this function on the main network(Figure 10); That is to say, now, only DaiJoin contract has the minting permission of USDFL. The whole project uses DaiJoin contract to manage USDFL tokens, and other user-address cannot mint tokens.

```
340
341     function deployDai(uint256 chainId) public auth {
342         require(address(vat) != address(0), "Missing previous step");
343
344         // Deploy
345         dai = daiFab.newDai(chainId);
346         daiJoin = daiJoinFab.newDaiJoin(address(vat), address(dai));
347         dai.rely(address(daiJoin));
348     }
349
```

Figure 6 Source code of function deployDai

```
47  contract DaiFab {
48      function newDai(uint chainId) public returns (Dai dai) {
49          dai = new Dai(chainId);
50          dai.rely(msg.sender);
51          dai.deny(address(this));
52      }
53  }
```

Figure 7 Source code of contract DaiFab

```
459    function releaseAuth() public auth {
460        vat.deny(address(this));
461        cat.deny(address(this));
462        vow.deny(address(this));
463        jug.deny(address(this));
464        pot.deny(address(this));
465        dai.deny(address(this));
466        spotter.deny(address(this));
467        flap.deny(address(this));
468        flop.deny(address(this));
469        end.deny(address(this));
470    }
```

Figure 8 Source code of function releaseAuth

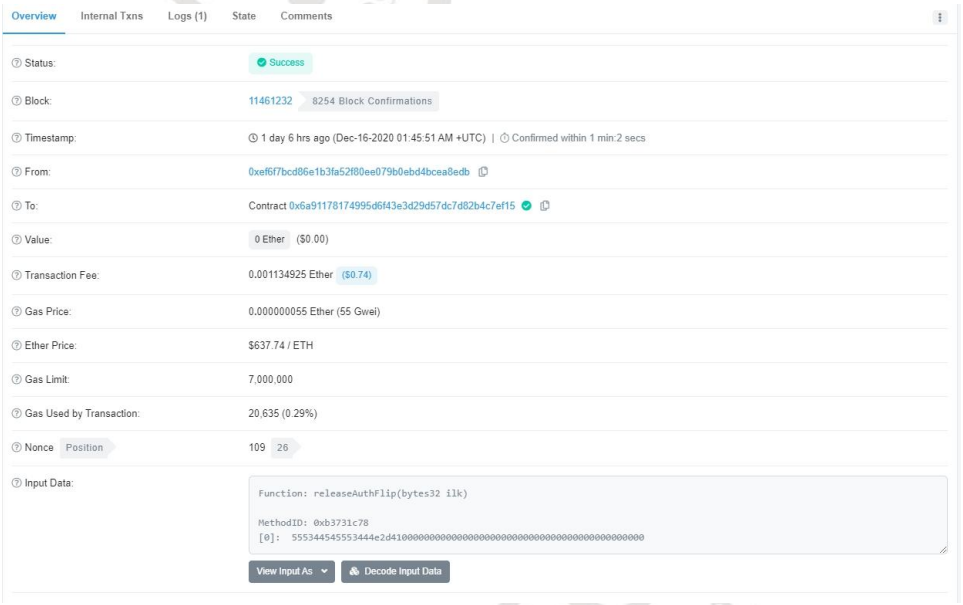


Figure 9 The transaction information that calls the function releaseAuth

```
220    }
221    function join(address usr, uint wad) external note {
222        vat.move(address(this), usr, mul(ONE, wad));
223        dai.burn(msg.sender, wad);
224    }
225    function exit(address usr, uint wad) external note {
226        require(live == 1, "DaiJoin/not-live");
227        vat.move(msg.sender, address(this), mul(ONE, wad));
228        dai.mint(usr, wad);
229    }
```

Figure 10 Source code of functions join and exit

- Related functions: *constructor, rely, deny*
- Safety Suggestion: None
- Result: Pass

3.1.5 Sign to approve

- Description: Contract token holders can permanently approve and cancel approval to other addresses through signatures. The corresponding code is shown in the figure below. In addition to the approve parameters, the signature data also needs to specify nonce and expiry to ensure the validity of the signature data.

```
    // --- Approve by signature ---
    function permit(address holder, address spender, uint256 nonce, uint256 expiry,
                    bool allowed, uint8 v, bytes32 r, bytes32 s) external
    {
        bytes32 digest =
            keccak256(abi.encodePacked(
                "\x19\x01",
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH,
                                     holder,
                                     spender,
                                     nonce,
                                     expiry,
                                     allowed))
        ));

        require(holder != address(0), "Dai/invalid-address-0");
        require(holder == ecrecover(digest, v, r, s), "Dai/invalid-permit");
        require(expiry == 0 || now <= expiry, "Dai/permit-expired");
        require(nonce == nonces[holder]++, "Dai/invalid-nonce");
        uint wad = allowed ? uint(-1) : 0;
        allowance[holder][spender] = wad;
        emit Approval(holder, spender, wad);
    }
}
```

Figure 11 Source code of function permit

- Related functions: *permit*
- Safety Suggestion: None
- Result: Pass

3.1.6 Other transfer function

- Description: In addition to ERC20 basic transfer functions, this contract also impelements other transfer functions to deal with transfers in different situations. Among them, *push* means that the caller transfers money to the target address; *pull* means that the delegate source address transfers money to the caller; *move* means that the delegate address transfers money.

```
    // --- Alias ---
    function push(address usr, uint wad) external {
        transferFrom(msg.sender, usr, wad);
    }
    function pull(address usr, uint wad) external {
        transferFrom(usr, msg.sender, wad);
    }
    function move(address src, address dst, uint wad) external {
        transferFrom(src, dst, wad);
    }
```

Figure 12 Source code of functions push, pull, move

- Related functions: *push, pull, move*
- Safety Suggestion: None
- Result: Pass

3.1.7 Contract Management

- Description: As shown in the figure below, *newDai* is implemented in the USDFLFab contract to create the USDFL contract. This function can be called by any user, but considering that other related

contracts in the system need to specify the gov contract address, the calling permission of this function can be ignored.

In addition, after this function creates the USDFL contract, it will directly call the *rely* function to grant the caller the management permission; call the *deny* function to cancel the contract's management permission for the newly created USDFL contract.

```
144 contract USDFLFab {
145     function newDai(uint chainId) public returns (USDFL dai) {
146         dai = new USDFL(chainId);
147         dai.rely(msg.sender);
148         dai.deny(address(this));
149     }
150 }
```

Figure 13 Source code of function newDai

- Related functions: *newDai*
- Safety Suggestion: None
- Result: Pass

3.2 The gemForRewardChecker Contract Audit

3.2.1 Add gem checker

- Description: When this contract is deployed, the deployment address management permission will be granted. The address with administrator permission can call the *addChecker* function to add the specified address to the checker list.

```
11     function addChecker(address checker) public {
12         require(deployer == msg.sender, "addChecker/auth-error");
13         checkers.push(checker);
14     }
```

Figure 14 Source code of function addChecker

- Related functions: *addChecker*
- Safety Suggestion: None
- Result: Pass

3.2.2 Check gem

- Description: Any address can call the *check* function of this contract to check whether the specified gem is valid. As shown in the figure below, the check function traverses the list of all checkers and calls *check* function of their contract.

```
16     function check(address gem) external returns (bool) {
17         for (uint256 i = 0; i < checkers.length; i++) {
18             (bool ret, ) = checkers[i].call(abi.encodeWithSignature("check(address)", gem));
19             if (ret) {
20                 return true;
21             }
22         }
23
24         return false;
25     }
```

Figure 15 Source code of function check

After testing, this function currently has two issues:

(1) If there is an account address in the checker list, the check function will return true if any address is checked;

(2) If the address of this contract (GemForRewardChecker) exists in the checker list, the function will consume the upper limit of block (cause "out of gas") gas because of the internal loop call.

- Related functions: *check*
- Safety Suggestion: When adding a checker in the *addChecker* function, check the specified address to avoid the above situation.
- Fix Result: Ignored. This issue can be avoided by administrator operation.
- Result: Pass

3.3 Adapter Related Contract Audit

This project provides a variety of Adapter contracts to adapt to the calculation of the value of tokens in different environments.

3.3.1 MooniAdapterForStables contract

- Description: The MooniAdapterForStables contract implements the price adaptation function of the corresponding LP token on the gem in the Mooni environment. The corresponding function *calc* is shown in the figure below. This function will get the balance of two tokens on the corresponding gem and convert it to the corresponding number of tokens; Then counts the total value of the gem with the smaller value of the number of tokens; Finally, the unit price of the LP token is calculated based on the total value and the number of LP tokens on the corresponding gem, and the value of the token under the value quantity is calculated.

```
20      function calc(
21          address gem,
22          uint256 value,
23          uint256 factor
24      ) external view returns (uint256) {
25          IERC20[] memory tokens = MooniPairLike(gem).getTokens();
26          uint256 reserve0 = tokens[0].balanceOf(gem);
27          uint256 reserve1 = tokens[1].balanceOf(gem);
28
29          uint256 r0 = uint256(reserve0).div(uint256(10)**tokens[0].decimals());
30          uint256 r1 = uint256(reserve1).div(uint256(10)**tokens[1].decimals());
31
32          uint256 totalValue = r0.min(r1).mul(2); //total value in uni's reserves for stables only
33
34          uint256 supply = MooniPairLike(gem).totalSupply();
35
36          return value.mul(totalValue).mul(factor).div(supply);
37      }
```

Figure 16 Source code of function calc in MooniAdapterForStables

- Related functions: *calc* of *MooniAdapterForStables* contract
- Safety Suggestion: None
- Result: Pass

3.3.2 UniswapAdapterForStables contract

- Description: The UniswapAdapterForStables contract implements the price adaptation function of the corresponding stable token on the gem. The corresponding function *calc* is shown in the figure below. This function will get the balance of two tokens on the corresponding gem and convert it to the corresponding number of stable tokens; Then counts the total value of LP tokens on the gem with the smaller USD value of the specified token; Finally, the unit price of the LP token is calculated based on the total value and the number of LP tokens on the corresponding gem, and the value of the token under the value quantity is calculated(Note: The value of the token here has been expanded by 10**18 times to avoid decimal problems).

```
42      function calc(
43          address gem,
44          uint256 value,
45          uint256 factor
46      ) external view returns (uint256) {
47          (uint112 _reserve0, uint112 _reserve1, ) = UniswapV2PairLike(gem).getReserves();
48
49          TokenPair memory tokenPair;
50          tokenPair.usdPrec = 10**6;
51
52          tokenPair.t0 = UniswapV2PairLike(gem).token0();
53          tokenPair.t1 = UniswapV2PairLike(gem).token1();
54
55          tokenPair.r0 = uint256(_reserve0).mul(tokenPair.usdPrec).div(
56              uint256(10)**IERC20(tokenPair.t0).decimals()
57          );
58          tokenPair.r1 = uint256(_reserve1).mul(tokenPair.usdPrec).div(
59              uint256(10)**IERC20(tokenPair.t1).decimals()
60          );
61
62          uint256 totalValue = tokenPair.r0.min(tokenPair.r1).mul(2); //total value in uni's reserves for stables only
63
64          uint256 supply = UniswapV2PairLike(gem).totalSupply();
65
66          return value.mul(totalValue).mul(factor).mul(1e18).div(supply.mul(tokenPair.usdPrec));
67      }
```

Figure 17 Source code of function calc in UniswapAdapterForStables

- Related functions: *calc* of *UniswapAdapterForStables* contract
- Safety Suggestion: None
- Result: Pass

3.3.3 UniswapAdapterWithOneStable contract

- Description: The UniswapAdapterWithOneStable contract implements the price adaptation function of the corresponding stable token on the gem. The corresponding function *calc* is shown in the figure below. This function will get the balance of two tokens on the corresponding gem and find the USD token according to the specified token; Then counts the total value of gem(without multiplying 2); Finally, the unit price of the LP token is calculated(multiply 2) based on the total value and the number of LP tokens on the corresponding gem, and the value of the token under the value quantity is calculated(Note: The value of the token here has been expanded by 10**18 times to avoid decimal problems).

```
99      function calc(
100         address gem,
101         uint256 value,
102         uint256 factor
103     ) external view returns (uint256) {
104         (uint112 _reserve0, uint112 _reserve1, ) = UniswapV2PairLike(gem).getReserves();
105
106         LocalVars memory loc;
107         loc.t0 = UniswapV2PairLike(gem).token0();
108         loc.t1 = UniswapV2PairLike(gem).token1();
109         loc.usdPrec = 10**6;
110
111         if (buck == loc.t0) {
112             loc.totalValue = uint256(_reserve0).mul(loc.usdPrec).div(
113                 uint256(10)**IERC20(loc.t0).decimals()
114             );
115         } else if (buck == loc.t1) {
116             loc.totalValue = uint256(_reserve1).mul(loc.usdPrec).div(
117                 uint256(10)**IERC20(loc.t1).decimals()
118             );
119         } else {
120             require(false, "gem w/o buck");
121         }
122
123         loc.supply = UniswapV2PairLike(gem).totalSupply();
124
125         return
126             value.mul(loc.totalValue).mul(2).mul(factor).mul(1e18).div(
127                 loc.supply.mul(loc.usdPrec)
128             );
```

Figure 18 Source code of function calc in UniswapAdapterWithOneStable

- Related functions: *calc of UniswapAdapterWithOneStable contract*
- Safety Suggestion: None
- Result: Pass

3.4 The oracles Contract Audit

3.4.1 The UniswapAdapterPriceOracle_USDT_Buck contract initialization

- Description: After the UniswapAdapterPriceOracle_USDT_Buck contract is deployed, the contract deployer can call the contract's *setup* function to initialize, as shown in the figure below, the initialization needs to be passed contract addresses including price contract of ETH to USDT, price contract of USDT to ETH, gem contract, USDT contract and whether to enable USDT String check for short name.

```
function setup(
    address _priceETHUSDT,
    address _priceUSDETH,
    address _gem,
    address _usdtAddress,
    bool usdtAsString
) public {
    require(deployer == msg.sender);
    require(_usdtAddress != address(0));
    require(_priceETHUSDT != address(0));
    require(_priceUSDETH != address(0));
    require(_gem != address(0));

    (bool success, bytes memory returndata) =
        address(_usdtAddress).call(abi.encodeWithSignature("symbol()"));
    require(success, "USDT: low-level call failed");

    require(returndata.length > 0);
    if (usdtAsString) {
        bytes memory usdtSymbol = bytes(abi.decode(returndata, (string)));
        require(keccak256(bytes(usdtSymbol)) == keccak256("USDT"));
    } else {
        bytes32 usdtSymbol = abi.decode(returndata, (bytes32));
        require(usdtSymbol == "USDT");
    }

    priceETHUSDT = AggregatorV3Interface(_priceETHUSDT); //1/354 USD kovan:0x0bF499444525a23E7Bb61997539725cA2e928138
    priceUSDETH = AggregatorV3Interface(_priceUSDETH); //354 USD kovan:0x9326BFA02ADD2366b30bacB125260Af641031331
    gem = UniswapV2PairLike(_gem);
    usdtAddress = _usdtAddress;

    deployer = address(0);
}
```

Figure 19 Source code of function setup

- Related functions: *setup*
- Safety Suggestion: None
- Result: Pass

3.4.2 Price calculation in UniswapAdapterPriceOracle_USDT_Buck contract

- Description: Users can call the *peek* function and *read* function of this contract to query the price. This function will call the *calc* function to get the real-time value and calculate the price. As shown in Figure 20 after checking the key system parameters, the *calc* function calls the *latestRoundData* function on the corresponding price contract to get the price of USD to ETH and ETH to USDT; then go back to the token address corresponding to gem to check the balance of USDT and USD; then calculate the actual price of USDT based on real-time data, and calculate the USDT value based on the actual price and the USDT stock on the gem; finally calculate the value of the corresponding LP token on the gem (note: the value of the token here Expanded by 10**18 times to avoid decimal problems).

```
96      */
97      function calc() internal view returns (bytes32, bool) {
98          if (
99              address(priceETHUSDT) == address(0x0) ||
100             address(priceUSDETH) == address(0x0) ||
101             address(gem) == address(0x0)
102         ) {
103             return (0x0, false);
104         }
105
106         (, int256 answerUSDETH, , , ) = priceUSDETH.latestRoundData();
107         (, int256 answerETHUSDT, , , ) = priceETHUSDT.latestRoundData();
108
109         if (answerUSDETH <= 0 || answerETHUSDT <= 0) {
110             return (0x0, false);
111         }
112
113         TokenPair memory tokenPair;
114         {
115             (uint112 _reserve0, uint112 _reserve1, ) = gem.getReserves();
116
117             if (gem.token1() == usdtAddress) {
118                 tokenPair.buck = gem.token0(); //buck
119                 tokenPair.buckReserve = uint256(_reserve0);
120
121                 tokenPair.usdt = gem.token1(); //USDT
122                 tokenPair.usdtReserve = uint256(_reserve1);
123             } else {
124                 tokenPair.usdt = gem.token0(); //USDT
125                 tokenPair.usdtReserve = uint256(_reserve0);
126
127                 tokenPair.buck = gem.token1(); //buck
128                 tokenPair.buckReserve = uint256(_reserve1);
129             }
130         }
```

Figure 20 Source code of function calc in UniswapAdapterWithOneStable contract (1/2)

```
96      */
97      function calc() internal view returns (bytes32, bool) {
98          if (
99              address(priceETHUSDT) == address(0x0) ||
100             address(priceUSDETH) == address(0x0) ||
101             address(gem) == address(0x0)
102         ) {
103             return (0x0, false);
104         }
105
106         (, int256 answerUSDETH, , , ) = priceUSDETH.latestRoundData();
107         (, int256 answerETHUSDT, , , ) = priceETHUSDT.latestRoundData();
108
109         if (answerUSDETH <= 0 || answerETHUSDT <= 0) {
110             return (0x0, false);
111         }
112
113         TokenPair memory tokenPair;
114         {
115             (uint112 _reserve0, uint112 _reserve1, ) = gem.getReserves();
116
117             if (gem.token1() == usdtAddress) {
118                 tokenPair.buck = gem.token0(); //buck
119                 tokenPair.buckReserve = uint256(_reserve0);
120
121                 tokenPair.usdt = gem.token1(); //USDT
122                 tokenPair.usdtReserve = uint256(_reserve1);
123             } else {
124                 tokenPair.usdt = gem.token0(); //USDT
125                 tokenPair.usdtReserve = uint256(_reserve0);
126
127                 tokenPair.buck = gem.token1(); //buck
128                 tokenPair.buckReserve = uint256(_reserve1);
129             }
130         }
```

Figure 21 Source code of function calc in UniswapAdapterWithOneStable contract (2/2)

- Related functions: *calc* of *UniswapAdapterWithOneStable* contract
- Safety Suggestion: None
- Result: Pass

3.4.3 Price calculation in UniswapAdapterPriceOracle_Buck_Buck contract

- Description: The user can call the *peek* function and *read* function of this contract to query the price. This function will call the *calc* function to get the balance of the token corresponding to the gem and calculate the corresponding balance value; then calculate the value of an LP token on the gem based on the total value of the LP on the gem.

```
217     function calc() internal view returns (bytes32, bool) {
218         (uint112 _reserve0, uint112 _reserve1, ) = gem.getReserves();
219
220         TokenPair memory tokenPair;
221         tokenPair.t0 = gem.token0();
222         tokenPair.t1 = gem.token1();
223
224         uint256 usdPrec = 10**6;
225
226         uint256 r0 =
227             uint256(_reserve0).mul(usdPrec).div(
228                 uint256(10)**uint256(IERC20(tokenPair.t0).decimals())
229             );
230         uint256 r1 =
231             uint256(_reserve1).mul(usdPrec).div(
232                 uint256(10)**uint256(IERC20(tokenPair.t1).decimals())
233             );
234
235         //we use the minimum USD value of the two tokens to prevent Uniswap disbalance attack
236         uint256 totalValue = r0.min(r1).mul(2); //total value in uni's reserves
237         uint256 supply = gem.totalSupply();
238
239         return (
240             bytes32(
241                 totalValue.mul(10**(uint256(gem.decimals()).add(18))).div(supply.mul(usdPrec))
242             ),
243             true
244         );
245     }
```

Figure 22 Source code of function calc in UniswapAdapterPriceOracle_Buck_Buck contract

- Related functions: *calc* of *UniswapAdapterPriceOracle_Buck_Buck* contract
- Safety Suggestion: None
- Result: Pass

3.5 The priceProvider Contract Audit

3.5.1 Contract initialization

- Description: After the priceProvider contract is deployed, its contract owner (contract deployer) can call the contract's *setup* function to initialize the contract. In addition to initializing the system parameters, this function will also determine the total reward and the reward start time and epoch according to the current LP token balance of the contract. Also note that this function can be called multiple times to update system parameters, but it does not affect the user's reward withdrawal.

```
51    function setup(
52        address _gov,
53        address _spot,
54        address _registry,
55        uint256 _updatePeriod,
56        uint256 _rewardTime
57    ) public {
58        require(owner == msg.sender, "auth-error");
59
60        require(_gov != address(0), "gov is null");
61        require(_spot != address(0), "spot is null");
62        require(_updatePeriod != 0, "updatePeriod is zero");
63        require(_registry != address(0), "registry is null");
64        require(_rewardTime > _updatePeriod * 10, "rewardTime vs updatePeriod inconsistence");
65
66        rewardToDistribute = IERC20(_gov).balanceOf(address(this));
67        require(rewardToDistribute > 0, "no reward to distribute");
68
69        uint256 chunks = _rewardTime.div(_updatePeriod);
70
71        registry = RegistryLike(_registry);
72        spot = SpotLike(_spot);
73        gov = _gov;
74        rewardTime = _rewardTime;
75        updatePeriod = _updatePeriod;
76        rewardPerPeriod = rewardToDistribute.div(chunks);
77        require(rewardPerPeriod > 0, "rewardPerPeriod is zero");
78    }
79
```

Figure 23 Source code of function setup

- Related functions: *setup*
- Safety Suggestion: None
- Result: Pass

3.5.2 Update price feed

- Description: The user can call the contract's *poke* function to update price feed, and the user who calls this function for the first time in the epoch will distribute the reward for that epoch.

```
97    function poke() public {
98        bytes32[] memory ilks = registry.list();
99        for (uint256 i = 0; i < ilks.length; i++) {
100           spot.poke(ilks[i]);
101       }
102
103       if (block.timestamp >= nextUpdate) {
104           rewards[msg.sender] = rewards[msg.sender].add(rewardPerPeriod);
105       }
106
107       nextUpdate = updatePeriod.add(block.timestamp);
108   }
```

Figure 24 Source code of function poke

- Related functions: *poke*
- Safety Suggestion: None
- Result: Pass

3.5.3 Withdraw reward

- Description: Users with rewards can call the *getReward* function of the contract to receive rewards.

```
83      function getReward() public returns (uint256) {
84          uint256 acc = rewards[msg.sender];
85          if (acc > 0) {
86              distributedReward = distributedReward.add(acc);
87              IERC20(gov).safeTransfer(msg.sender, acc);
88              emit RewardPaid(msg.sender, acc);
89              rewards[msg.sender] = 0;
90          }
91          return acc;
92      }
```

Figure 25 Source code of function getReward

- Related functions: *getReward*
- Safety Suggestion: None
- Result: Pass

3.6 The StakingRewards Contract Audit

3.6.1 Contract initialization

- Description: After the contract is deployed, the contract deployer can call the *initialize* function of the contract to initialize key parameters and epoch reward data. This function uses an initializer, which limits its call status.

```
106     function initialize(
107         address _gov,
108         uint256 _duration,
109         uint256 _initreward,
110         uint256 _starttime
111     ) public initializer {
112         // only deployer can initialize
113         require(deployer == msg.sender);
114
115         require(_starttime >= block.timestamp);
116
117         gov = _gov;
118
119         duration = _duration;
120         starttime = _starttime;
121         initRewardAmount(_initreward);
122     }
```

Figure 26 Source code of function initialize

- Related functions: *initialize*
- Safety Suggestion: None
- Result: Pass

3.6.2 Contract management

- Description: The contract deployer can call the functions *rely* and *deny* to add and remove administrators. The administrator of this contract has the permission to call the *registerPairDesc* function to register nodes.
- Related functions: *rely, deny, registerPairDesc*

- Safety Suggestion: None
- Result: Pass

3.6.3 Set the related contract address of the contract

- Description: The contract deployer can call the *setupGemForRewardChecker* function to set the gem's reward check contract; the *setupFairDistribution* function can be called to set the fair distribution parameters
- Related functions: *setupGemForRewardChecker, setupFairDistribution*
- Safety Suggestion: None
- Result: Pass

3.6.4 Register LP Pair token node

- Description: Users who has administrator permission can call the *registerPairDesc* function to register the node, as shown in the figure below. In the function, the validity of the relevant parameters will be checked and the corresponding gem will be added to the registration list.

```
161     function registerPairDesc(
162         address gem,
163         address adapter,
164         uint256 factor,
165         address staker
166     ) public auth nonReentrant {
167         require(gem != address(0x0), "gem is null");
168         require(adapter != address(0x0), "adapter is null");
169 
170         require(checkGem(gem), "bad gem");
171 
172         registerGem(gem);
173 
174         pairDescs[gem] = PairDesc({
175             gem: gem,
176             adapter: adapter,
177             factor: factor,
178             staker: staker,
179             name: "dummy"
180         });
181     }
```

Figure 27 Source code of function registerPairDesc

- Related functions: *registerPairDesc, registerGem*
- Safety Suggestion: None
- Result: Pass

3.6.5 Stake LP tokens

- Description: The staker of the registered gem can call the *stake* function of this contract to stake correspondingly. This function will call the *stakeLp* function to process the LP token stake logic. As shown in the figure below, the function calcCheckValue will calculate the token value of corresponding

amount of LP token (here divided by 10**18 corresponds to the *calc* function multiplied by 10**18); then update the relevant pledge data.

Also note that the stake function only processes the stake logic and does not involve the transfer of LP tokens (this part of the business logic is in the *join* function of the GemJoinWithReward contract).

```
334     function stakeLp(
335         uint256 amount,
336         address gem,
337         address usr
338     ) internal {
339         uint256 value = calcCheckValue(amount, gem).mul(prec);
340 
341         _balances[gem][usr] = _balances[gem][usr].add(value);
342         _amounts[gem][usr] = _amounts[gem][usr].add(amount);
343         _totalSupply[gem] = _totalSupply[gem].add(value);
344     }
```

Figure 28 Source code of function stakeLp

- Related functions: *stake, stakeLp, calcCheckValue*
- Safety Suggestion: None
- Result: Pass

3.6.6 Withdraw staked assets

- Description: The staker of the registered gem can call the *withdraw* function of this contract to withdraw the staked LP token of the specified user. This function will call the *withdrawLp* function to process the logic of staked LP token withdrawal. As shown in the figure below, this function will update the value of the corresponding account based on the ratio of the number of tokens passed in this call to the value of the whole staked tokens.

Also note that the *withdraw* function only processes the logic of withdrawing staked tokens and does not involve the transfer of LP tokens (this part of the business logic is in the *exit* function of the GemJoinWithReward contract).

```
349     function withdrawLp(
350         uint256 amount,
351         address gem,
352         address usr
353     ) internal {
354         uint256 value = amount.mul(_balances[gem][usr]).div(_amounts[gem][usr]);
355 
356         _balances[gem][usr] = _balances[gem][usr].sub(value);
357         _amounts[gem][usr] = _amounts[gem][usr].sub(amount);
358         _totalSupply[gem] = _totalSupply[gem].sub(value);
359     }
```

Figure 29 Source code of function withdrawLp

- Related functions: *withdraw*, *withdrawLp*
- Safety Suggestion: None
- Result: Pass

3.6.7 Withdraw stake reward

- Description: For staked users, the *getReward* function of the contract can be called to receive the current stake reward.

```
301     function getReward()
302         public
303         nonReentrant
304         updateReward(msg.sender)
305         checkFinish
306         checkStart
307         returns (uint256)
308     {
309         uint256 reward = earned(msg.sender);
310         if (reward > 0) {
311             rewards[msg.sender] = 0;
312             IERC20(gov).safeTransfer(msg.sender, reward);
313             emit RewardPaid(msg.sender, reward);
314             totalRewards = totalRewards.add(reward);
315             return reward;
316         }
317         return 0;
318     }
```

Figure 30 Source code of getReward

In addition, the *claimReward* function of the RewardProxyActions contract can be used for delegate receive stake rewards, but after analyzing the logic may have the following two problems:

(1) The function will use the RewardProxyActions contract as the caller to receive rewards to this contract, and then forward them to the function caller; instead of receiving the stake reward from the function caller.

(2) The claimReward function can be called by any user to receive the stake reward of the RewardProxyActions contract.

```
472  * @title MakerDAO like ProxyActions for freeliquid specific reward claiming
473  *
474  * see MakerDAO docs about Proxy for details
475  */
476 contract RewardProxyActions {
477     function claimReward(address rewarder) public {
478         uint256 reward = StakingRewards(rewarder).getReward();
479         IERC20(StakingRewards(rewarder).gov()).transfer(msg.sender, reward);
480     }
481 }
```

Figure 31 Source code of function claimReward

- Related functions: *getReward, RewardProxyActions, claimReward*
- Safety Suggestion: Determine the actual use environment of the *claimReward* function of the RewardProxyActions contract to avoid the above problems.
- Fix Result: Ignored
- Result: Pass

3.7 The StakingRewards Contract Audit

3.7.1 Close staking

- Description: The deployer of this contract can call the *cage* function of the contract to close/disable the LP token stake function of the contract. This function will modify the live state of the contract to 0. In this state, no user can call the *join* function. In addition, the live state of the contract after calling the *cage* function can no longer be modified to 1.
- Related functions: *cage*
- Safety Suggestion: None
- Result: Pass

3.7.2 Stake LP tokens

- Description: Users who hold the corresponding LP tokens on the gem can call the *join* function to stake, which will call the corresponding rewarder contract's *stake* to process the stake data, and call the *checkFairDistribution* function to check whether the specified address conforms to the requirements of fair distribution (if fair distribution is enabled); then transfer the corresponding LP tokens to this contract.

```
430     function join(address urn, uint256 wad) external note {
431         require(live == 1, "GemJoinWithReward/not-live");
432         require(int256(wad) >= 0, "GemJoinWithReward/overflow");
433         vat.slip(ilk, urn, int256(wad));
434 
435         // rewarder.stake(wad, address(gem), msg.sender);
436         (bool ret, ) =
437             address(rewarder).call(
438                 abi.encodeWithSelector(rewarder.stake.selector, wad, address(gem), msg.sender)
439             );
440         if (!ret) {
441             emit stakeError(wad, address(gem), msg.sender);
442         }
443 
444         rewarder.checkFairDistribution(msg.sender);
445 
446         require(
447             gem.transferFrom(msg.sender, address(this), wad),
448             "GemJoinWithReward/failed-transfer"
449         );
450     }
```

Figure 32 Source code of function join

- Related functions: *join, checkFairDistribution*
- Safety Suggestion: None
- Result: Pass

3.7.3 Other LP tokens

- Description: Users who participate in the staking of LP tokens through this contract can call the *exit* of this contract to withdraw the staked tokens.

```
452     function exit(address usr, uint256 wad) external note {
453         require(wad <= 2**255, "GemJoinWithReward/overflow");
454         vat.slip(ilk, msg.sender, -int256(wad));
455
456         require(rewarder.allowToStart(), "join-not-start");
457
458         // rewarder.withdraw(wad, address(gem), msg.sender);
459         (bool ret, ) =
460             address(rewarder).call(
461                 abi.encodeWithSelector(rewarder.withdraw.selector, wad, address(gem), msg.sender)
462             );
463         if (!ret) {
464             emit withdrawError(wad, address(gem), msg.sender);
465         }
466
467         require(gem.transfer(usr, wad), "GemJoinWithReward/failed-transfer");
468     }
```

Figure 33 Source code of function exit

- Related functions: *exit*
- Safety Suggestion: None
- Result: Pass

3.8 The StakingRewardsDecay Contract Audit

3.8.1 Contract initialization

- Description: After the contract is deployed, the deployer can call the *initialize* function to initialize the contract. This function will specify the reward token address gov of the contract and the number of reward epochs. At the same time, this function will create a StakingRewardsDecayHolder contract according to this contract.

```
83      function initialize(address _gov, uint256 epochCount) public initializer {
84          // only deployer can initialize
85          require(deployer == msg.sender);
86
87          gov = _gov;
88          require(gov != address(0));
89          require(epochCount > 0);
90
91          EPOCHCOUNT = epochCount;
92          EpochData memory data;
93          for (uint256 i = 0; i < epochCount; i++) {
94              epochs.push(data);
95          }
96
97          holder = new StakingRewardsDecayHolder(address(this));
98      }
99
```

Figure 34 Source code of function initialize

- Related functions: *initialize*
- Safety Suggestion: None
- Result: Pass

3.8.2 Contract management

- Description: The contract deployer can call the functions *rely* and *deny* to add and remove administrators. The administrator of this contract has the permission to call the *registerPairDesc* function to register nodes.
- Related functions: *rely, deny, registerPairDesc*
- Safety Suggestion: None
- Result: Pass

3.8.3 Epoch reward initialization

- Description: The contract deployer can call *initRewardAmount* to set the epoch reward information corresponding to the specified index. This function requires the epochInited value to be 0, that is, the contract has not completed the epoch reward data verification; and calls the *initEpoch* function to initialize the specified epoch.

```
135     function initRewardAmount(
136         uint256 reward,
137         uint256 starttime,
138         uint256 duration,
139         uint256 idx
140     ) public {
141         // only deployer can
142         require(deployer == msg.sender);
143         require(epochInited == 0, "not allowed after approve");
144         initEpoch(reward, starttime, duration, idx);
145     }
```

Figure 35 Source code of function initRewardAmount

```
163     function initEpoch(
164         uint256 reward,
165         uint256 starttime,
166         uint256 duration,
167         uint256 idx
168     ) internal {
169         require(idx < EPOCHCOUNT, "idx < EPOCHCOUNT");
170         require(duration > 0, "duration > 0");
171         require(starttime >= block.timestamp, "starttime > block.timestamp");
172 
173         EpochData storage epoch = epochs[idx];
174 
175         epoch.rewardPerTokenStored = 0;
176         epoch.starttime = starttime;
177         epoch.duration = duration;
178         epoch.rewardRate = reward.div(duration);
179         require(epoch.rewardRate > 0, "zero rewardRate");
180 
181         epoch.initreward = reward;
182         epoch.lastUpdateTime = starttime;
183         epoch.periodFinish = starttime.add(duration);
184 
185         emit RewardAdded(reward, idx, duration, starttime);
186     }
```

Figure 36 Source code of function initEpoch

The contract deployer can also call the *initAllEpochs* function to initialize all epoch reward information at once.

```
194     function initAllEpochs(
195         uint256[] memory rewards,
196         uint256 starttime,
197         uint256 duration
198     ) public {
199         // only deployer can
200         require(deployer == msg.sender);
201         require(epochInited == 0, "not allowed after approve");
202
203         require(duration > 0);
204         require(starttime > 0);
205
206         assert(rewards.length == EPOCHCOUNT);
207
208         uint256 time = starttime;
209
210         for (uint256 i = 0; i < EPOCHCOUNT; i++) {
211             initEpoch(rewards[i], time, duration, i);
212             time = time.add(duration);
213         }
214     }
```

Figure 37 Source code of function initAllEpochs

- Related functions: *initRewardAmount, initEpoch, initAllEpochs*
- Safety Suggestion: None
- Result: Pass

3.8.4 Epochs Consistency check

- Description: After completing the initialization of the epoch data, the contract deployer can call the *approveEpochsConsistency* function to verify the epoch data corresponding to the setting. As shown in the figure below, the function will check the validity of the epoch reward data and require that the current gov token held by the contract is not less than the specified amount of reward; the last updated epochInited value indicates that the epoch data setting is completed.

```
267     function approveEpochsConsistency() public {
268         require(deployer == msg.sender);
269         require(epochInited == 0, "double call not allowed");
270
271         uint256 totalReward = epochs[0].initreward;
272         require(getStartTime() > 0);
273
274         for (uint256 i = 1; i < EPOCHCOUNT; i++) {
275             EpochData storage epoch = epochs[i];
276             require(epoch.starttime > 0);
277             require(epoch.starttime == epochs[i - 1].periodFinish);
278             totalReward = totalReward.add(epoch.initreward);
279         }
280
281         require(IERC20(gov).balanceOf(address(this)) >= totalReward, "GOV balance not enought");
282
283         epochInited = EPOCHCOUNT;
284     }
```

Figure 38 Source code of function approveEpochsConsistency

- Related functions: *approveEpochsConsistency*
- Safety Suggestion: None
- Result: Pass

3.8.5 System key parameter setting

- Description: The contract deployer can call the *setupAggregator* function to set the aggregator contract address, or call the *setupGemForRewardChecker* function to set the gem's check contract address.
- Related functions: *setupAggregator, setupGemForRewardChecke*
- Safety Suggestion: None
- Result: Pass

3.8.6 Register LP Pair token node

- Description: Users who has administrator permission can call the *registerPairDesc* function to register the node, as shown in the figure below. In the function, the validity of the relevant parameters will be checked and the corresponding gem will be added to the registration list. It should be noted that unlike in the StakingRewards contract, the staker of the node registered by this function is a zero address.

```
342     function registerPairDesc(
343         address gem,
344         address adapter,
345         uint256 factor,
346         bytes32 name
347     ) public auth nonReentrant {
348         require(gem != address(0x0), "gem is null");
349         require(adapter != address(0x0), "adapter is null");
350 
351         require(checkGem(gem), "bad gem");
352 
353         require(pairNameToGem[name] == address(0) || pairNameToGem[name] == gem, "duplicate name");
354 
355         if (pairDescs[gem].name != "") {
356             delete pairNameToGem[pairDescs[gem].name];
357         }
358 
359         registerGem(gem);
360 
361         pairDescs[gem] = PairDesc({
362             gem: gem,
363             adapter: adapter,
364             factor: factor,
365             staker: address(0),
366             name: name
367         });
368 
369         pairNameToGem[name] = gem;
370     }
```

Figure 39 Source code of function registerPairDesc

- Related functions: *registerPairDesc, registerGem, checkGem*
- Safety Suggestion: None
- Result: Pass

3.8.7 Stake LP tokens

- Description: Users who hold the LP tokens corresponding to the specified gem can call the *stake* function of the StakingRewardsDecayHolder contract to stake tokens, which will call the *stake* function of this contract for staking; and the *stake* function will call the *stakeEpoch* function to receive the stake reward of corresponding users' and record the corresponding stake data.

```
536     */
537     function stakeEpoch(
538         uint256 amount,
539         address gem,
540         address usr,
541         EpochData storage epoch
542     ) internal updateReward(usr, epoch) {
543         gatherOldEpochReward(usr);
544         stakeLp(amount, gem, usr);
545         emit Staked(usr, gem, amount);
546     }
```

Figure 40 Source code of function stakeEpoch

- Related functions: *stake, stakeEpoch, gatherOldEpochReward, stakeLp, calcCheckValue*
- Safety Suggestion: None
- Result: Pass

3.8.8 Withdraw staked LP tokens

- Description: Stake users can call the *withdraw* function of the StakingRewardsDecayHolder contract to withdraw staked LP tokens. This function will call the *withdraw* function of this contract to process the logic of staked LP token withdrawal.
- Related functions: *withdraw, withdrawEpoch, withdrawLp*
- Safety Suggestion: None
- Result: Pass

3.8.9 Withdraw stake reward

- Description: Users can call the *getReward* function to receive the stake reward, or call the *getRewardEx* function to help the specified address to receive the stake reward. The main internal processing logic of these two functions is in the *getRewardCore* function. As shown in the figure below, the function calls the *takeStockReward* function to get the reward that can be received. Then update the reward data and send the reward.

```
593     function getRewardCore(address account)
594         internal
595         checkStart
596         updateCurrentEpoch
597         updateReward(account, epochs[currentEpoch])
598         returns (uint256 acc)
599     {
600         acc = takeStockReward(account);
601
602         acc = acc.add(yetNotClaimedOldEpochRewards[account]);
603         yetNotClaimedOldEpochRewards[account] = 0;
604
605         if (acc > 0) {
606             totalRewards = totalRewards.add(acc);
607             IERC20(gov).safeTransfer(account, acc);
608             emit RewardPaid(account, acc);
609         }
610     }
```

Figure 41 Source code of function getRewardCore

- Related functions: *getReward, getRewardEx, getRewardCore, takeStockReward*
- Safety Suggestion: None
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project freeliquid. All the issues found during the audit have been written into this audit report. The overall audit result of the smart contract project freeliquid is **Pass**.

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com