BEOSIN
Blockchain Security

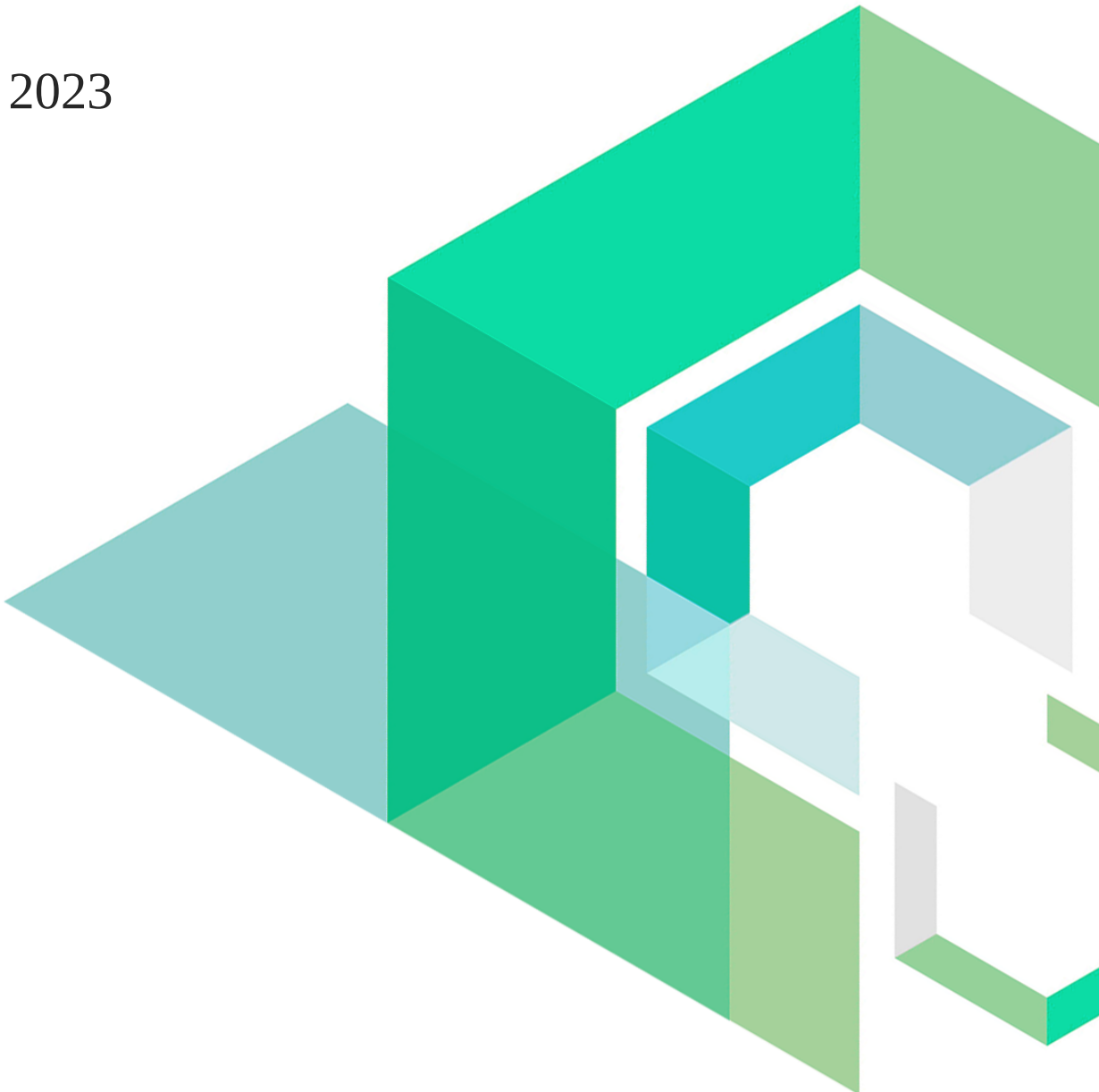
Quotafinance

Smart Contract Security Audit

V1.0

No. 202302031647

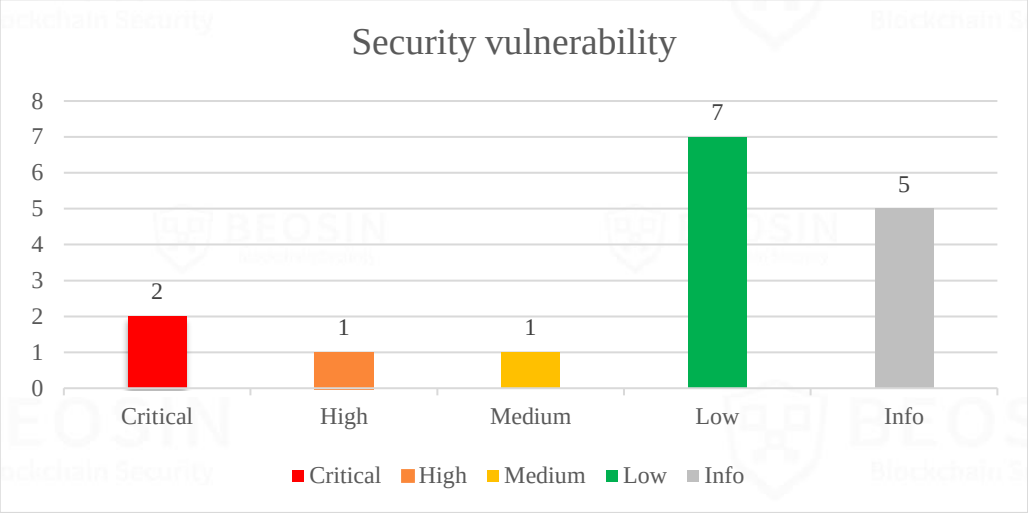
Feb 03rd, 2023



Contents

Summary of Audit Results

After auditing, 2 Critical-risk, 1 High-risk, 1 Medium-risk, 7 Low-risk and 5 Info-risk items were identified in the Quotafinance project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



***Notes:**

- **Risk Description:**

1. Unfixed problems may cause issue like function call failed in special cases, consume unnecessary gas and even Out of Gas, etc. Project party need to pay attention to unfixed problems, and users need to pay attention to the above risks when interacting with contracts.

- **Project Description:**

1. **Business overview**

The Quota Protocol is a multifaceted DeFi ecosystem.

Quota boasts a multifaceted DeFi ecosystem with a long list of features. Protocol features include but are not limited to:

Staking: This project includes a variety of modes of staking mining contracts. Users can stake specified tokens in the related contract to obtain ETF token rewards, and can also staking to improve their membership level. Reward Token ETF is a kind of inflation token. When the price of the token rises, the token will be issued according to the increase rate. This part of the newly issued token is used as a reward source for the recommendation system and staking mining system.

NFT Membership Referral System: NFT membership based multi-tiered referral system. This referral system functions as a membership system which allows for pseudonymous referrals by applying NFT technology. Users can invite other users to earn rewards, each NFT corresponds to a ReferralHandler contract and a DepositBox contract, where the ReferralHandler contract is used to record and process the ReferralHandler contract generated by the token inflation of the NFT owner. DepositBox contracts are used to process rewards allocated by project party.

An NFT Factory: Which allows users to directly issue their NFTs themselves based on their respective membership TIER requirements/criteria. Users can raise membership levels by inviting other members or staking to specified contracts.

Decentralized Community Governance: A decentralized on-chain community governance system, compound - like mechanism can be used to govern in related contracts to control the elements and parameters of protocol intelligent contracts.

Rebaser: The price of SNP and ETF token is obtained in the contract through the interface of ChainLink predictor and PancakeSwap pool, and the supply of ETF needs to be increased or decreased through calculation, so as to achieve the price/supply equilibrium.

1 Overview

1.1 Project Overview

Project Name	Quotafinance
Platform	BNB-Chain
Audit Scope	https://github.com/quotafinance/quota-core/tree/audit
Commit Hash	58fb5977fbf2ab7e97fae3b87599b8d605b15016 (Initial) d49744575ad051a5ed59753b886c67613c0257c8 (Jan 12, 2023) 71ccf0c0bbd06be3e103089c124efe8400946d7a (Jan 16, 2023) d1067f4564972ff554746584122bb143936c4400 (Jan 16, 2023) 88a6a24b71cd92329dfcd9da6d85e599aded3973 (Jan 19, 2023) e6002786b6ba5f1fb7550f91fe8036a3e5853e21 (Jan 19, 2023) 0e5946a2ae7c9df76071ec53039c69c542d25e30 (Jan 25, 2023) 3e9a14007d392092ede78573c373f9f724a2d7bc (Jan 26, 2023) 7bf67fa8200c407822f4b769918c0fc0db366d18 (Jan 27, 2023) 4fc879214b8c6b6209125417faf1ad6181d567a9 (Jan 27, 2023) df65652c969f8b1912131fdd9e76a5917f92c284 (Jan 27, 2023) ef4370c4771473c7773b5c085cc60bd6a9e120ba (Latest)

1.2 Audit Overview

Audit work duration: Jan 11, 2023 – Feb 03, 2023

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
Quotafinance-1	Rewards can be claimed repeatedly	Critical	Fixed
Quotafinance-2	The owner can withdraw the user's stake tokens	Critical	Fixed
Quotafinance-3	The *getEpoch* function will return 0 by default for unregistered accounts	High	Fixed
Quotafinance-4	The handler's referrer setting logic is unreasonable	Medium	Fixed
Quotafinance-5	Implementation logic is incorrect	Low	Partially Fixed
Quotafinance-6	Missing permission checks	Low	Fixed
Quotafinance-7	Multiple external initialize functions may cause misoperation	Low	Fixed
Quotafinance-8	Inefficient loop	Low	Acknowledged
Quotafinance-9	The logic of the query function and the actual execution function are inconsistent	Low	Fixed
Quotafinance-10	Flash loan risk	Low	Acknowledged
Quotafinance-11	User may cannot withdraw from the staking pool normally	Low	Acknowledged
Quotafinance-12	Loss of precision	Info	Fixed
Quotafinance-13	Code specification	Info	Acknowledged
Quotafinance-14	Token accuracy issue	Info	Acknowledged
Quotafinance-15	Missing event trigger	Info	Partially Fixed
Quotafinance-16	Redundant code	Info	Partially Fixed

Status Notes:

1. Quotafinance-5 is partially fixed and users can claim additional referral rewards in special cases.
2. Quotafinance-8 is not fixed and may consume unnecessary gas and even cause Out of Gas issue if the associated array is too long.
3. Quotafinance-10 is not fixed and may subject to flash loan attacks in special cases.
4. Quotafinance-11 is not fixed and may cause user cannot withdraw from the staking pool normally in special cases.
5. Quotafinance-13 is not fixed and will not cause any issue.

6. Quotafinance-14 is not fixed and may cause the calculated APY inaccurate or the function will call fail in special cases.

7. Quotafinance-15 is partially fixed and will not cause any issue.

8. Quotafinance-16 is partially fixed and will not cause any issue.

Finding Details:

[Quotafinance-1] Rewards can be claimed repeatedly

Severity Level	**Critical**
Type	Business Security
Lines	FixedTokenRewarder.sol#L81-89
Description	In the *withdraw* function of FixedTokenRewarder contract, it does not update the user's stake time (stakedFromTS), which will cause the stakedDuration to always be non-zero, then attackers can use the withdraw function to withdraw a small amount of tokens and thus repeat get the reward.

```
53    //计算抵押时间
54    function stakedDuration(address account) public view returns(uint256) {
55        uint256 secondsStaked = block.timestamp.sub(stakedFromTS[account]);
56        return secondsStaked;
57    }
58    //计算可领取的数量
59    function earned(address account) public view returns(uint256) {
60        return
61            balanceOf(account)//这里的balanof是抵押时才会记录的，不是代币的stake
62                .mul(stakedDuration(account))
63                .mul(rewardRate())
64                .div(1e18) // Div by 1e18 to offset the base multiplier of 1e18 in rewardRate()
65                .add(unclaimedRewards[account]);
66    }
67
68    function stake(uint256 amount) external {
69        require(amount > 0, "amount is <= 0");
70        require(token.balanceOf(msg.sender) >= amount, "balance is <= amount");//余额判断
71        token.transferFrom(msg.sender, address(this), amount);//代币转入
72        //如果抵押过
73        if (staked[msg.sender] > 0) {
74            unclaimedRewards[msg.sender] = earned(msg.sender);//更新为领取的数量
75        }
76        emit Staked(msg.sender, amount);
77        stakedFromTS[msg.sender] = block.timestamp;//更新抵押时间
78        staked[msg.sender] =  staked[msg.sender].add(amount);//更新用户的抵押量
79    }
80
81    function withdraw(uint256 amount) external {
82        require(amount > 0, "amount is <= 0");
83        require(staked[msg.sender] >= amount, "amount is > staked");
84        if (staked[msg.sender] > 0) {
85            unclaimedRewards[msg.sender] = earned(msg.sender);//更新未领取的奖励
86        }
87        emit Withdrawn(msg.sender, amount);
88        staked[msg.sender] = staked[msg.sender].sub(amount);
89        token.transfer(msg.sender, amount);
90    }
91
```

Figure 1 Source code of *FixedTokenRewarder* contract (unfixed)

Recommendations	It is recommended to update the user's stake time in the *withdraw* function.
Status	Fixed. The issue has been fixed according to suggestion. Timestamp is now updated when users withdraw.

```
79      function withdraw(uint256 amount) external {
80          require(amount > 0, "amount is <= 0");
81          require(staked[msg.sender] >= amount, "amount is > staked");
82          if (staked[msg.sender] > 0) {
83              unclaimedRewards[msg.sender] = earned(msg.sender);
84          }
85          stakedFromTS[msg.sender] = block.timestamp;
86          emit Withdrawn(msg.sender, amount);
87          staked[msg.sender] = staked[msg.sender].sub(amount);
88          token.transfer(msg.sender, amount);
89      }
```

Figure 2 Source code of *withdraw* function (fixed)

[Quotafinance-2] The owner can withdraw the user's stake tokens

Severity Level	**Critical**
Type	Business Security
Lines	TokenRewards.sol#L152-158 TokenRewardsDuration.sol#L153-158 PerpetualTokenRewards.sol#L151-157
Description	In the *recoverTokens* function in the above contract, the owner is allowed to call the function to extract any tokens of the contract, and the contract holds the user's stake tokens, which will lead to the risk that the user's stake tokens may be misappropriated.

```
152     function recoverTokens(
153         address _token,
154         address benefactor
155     ) public onlyOwner {
156         uint256 tokenBalance = IERC20(_token).balanceOf(address(this));
157         IERC20(_token).transfer(benefactor, tokenBalance);
158     }
```

Figure 3 Source code of *recoverTokens* function (unfixed)

Recommendations	It is recommended to determine in the function that it cannot be the user's stake tokens.
Status	Fixed. Added a check to ensure that only non-stake tokens can be transferred.

```
function recoverTokens(
    address _token,
    address benefactor
) public onlyOwner {
    require(_token != address(uni), "Cannot transfer LP tokens");
    uint256 tokenBalance = IERC20(_token).balanceOf(address(this));
    IERC20(_token).transfer(benefactor, tokenBalance);
}
```

Figure 4 Source code of *recoverTokens* function (fixed)

[Quotafinance-3] The *getEpoch* function will return 0 by default for unregistered accounts

Severity Level	**High**
Type	Business Security
Lines	NFTFactory.sol#L116-118 ReferralHandler.sol#L263-281
Description	For an address that has not created a handler, its claimedEpoch value returns 0 by default. Then, after the creator of the handler receives the reward, users can transfer the ETF token and NFT to the new address, and the new address will be able to start from the new address again. When the epoch is 0, start to receive reward.

```
function getEpoch(address user) external view returns (uint256) {
    return claimedEpoch[user];
}
```

Figure 5 Source code of *getEpoch* function

```
function claimReward() public { // Can be called by anyone but rewards always goes
    // This function mints the tokens that were deducted at rebase and disperses t
    // This also calls the claim function if there referral rewards from below ava
    address owner = ownedBy();
    ITaxManager taxManager =  getTaxManager();
    uint256 currentEpoch = getRebaser().getPositiveEpochCount();
    uint256 protocolTaxRate = taxManager.getProtocolTaxRate();
    uint256 taxDivisor = taxManager.getTaxBaseDivisor();
    uint256 claimedEpoch = INFTFactory(factory).getEpoch(ownedBy());
    if (claimedEpoch < currentEpoch) {
        claimedEpoch++;
        IRewarder rewarder = IRewarder(INFTFactory(factory).getRewarder());
        rewarder.handleReward(claimedEpoch, factory, address(token));
    }
    uint256 currentClaimable = token.balanceOf(address(this));
    if(currentClaimable > 0)
        handleClaimTaxAndDistribution(owner, currentClaimable, protocolTaxRate, ta
    levelUp();
```

Figure 6 Source code of *claimReward* function

Recommendations	It is recommended to delete the *transfer* function of NFT.
Status	Fixed. The repaired code updates the claimedEpoch value in the *_transfer* function, ensure that rewards cannot be claimed repeatedly by transferring NFT to a new address.

```
64      function _transfer(
65          address from,
66          address to,
67          uint256 tokenId
68      ) internal virtual override {
69          INFTFactory(factory).registerUserEpoch(to); // Alerting NFT Factory to update incase of new user
70          super._transfer(from, to, tokenId);
71      }
```

Figure 7 Source code of MembershipNFT contract (fixed)

[Quotafinance-4] The handler's referrer setting logic is unreasonable

Severity Level	**Medium**
Type	Business Security
Lines	NFTFactory.sol#L157-176
Description	The corresponding handler address is predictable, so the referrer filled in by the user may be the address of the subsequently created handler. If the user set it as referrer, then all referrers of the handler will be himself. In addition, what is more serious is that the referrer is still a malicious contract that can be deployed by an attacker.

```
function mint(address referrer) external returns (address) { //Referrer is address of NFT handler o
    uint256 nftID = NFT.issueNFT(msg.sender, tokenURI);
    uint256 epoch = IRebaser(rebaser).getPositiveEpochCount(); // The handlers need to only track p
    IReferralHandler handler = IReferralHandler(Clones.clone(handlerImplementation));
    // TODO: change the admin to not be static, instead change when NFT factory's admin is changed
    handler.initialize(epoch, token, referrer, address(NFT), nftID);
    IDepositBox depositBox =  IDepositBox(Clones.clone(depositBoxImplementation));
    depositBox.initialize(address(handler), nftID, token);
    handler.setDepositBox(address(depositBox));
    NFTToHandler[nftID] = address(handler);
    NFTToDepositBox[nftID] = address(depositBox);
    HandlerToNFT[address(handler)] = nftID;
    handlerStorage[address(handler)] = true;
    handlerStorage[address(depositBox)] = true; // Required to allow it fully transfer the collecte
    addToReferrersAbove(1, address(handler));
    emit NewIssuance(nftID, address(handler), address(depositBox));
    return address(handler);
}
```

Figure 8 Source code of *mint* function (unfixed)

Recommendations	It is recommended to add *require(handlerStorage[referrer] == true)*, i.e. requires that the corresponding referrer must already have a registered handler address.
Status	Fixed. Corresponding judgments have been added.

```
188     function mint(address referrer) external returns (address) { //Referrer is address of NFT handler of the guy abov
189         uint256 nftID = NFT.issueNFT(msg.sender, tokenURI);
190         uint256 epoch = IRebaser(rebaser).getPositiveEpochCount(); // The handlers need to only track positive rebase
191         IReferralHandler handler = IReferralHandler(Clones.clone(handlerImplementation));
192         require(address(handler) != referrer, "Cannot be its own referrer");
193         require(handlerStorage[referrer] == true || referrer == address(0), "Referrer should be a valid handler");
194         handler.initialize(token, referrer, address(NFT), nftID);
```

Figure 9 Source code of *mint* function (fixed)

[Quotafinance-5] Implementation logic is incorrect

Severity Level	**Low**
Type	Business Security
Lines	Rewarder.sol#L36-87
Description	There is a *claimReward* function in the ReferralHandler contract, which will call the *handleReward* function of the reward settlement, and the *handleReward* function in the rewarder contract is to get the owner of the NFT through the TokenID according to ownedBy(). Therefore, users can obtain referral rewards by transferring NFT to other owners.

```
35
36  function handleReward(
37      uint256 claimedEpoch,
38      address factory,
39      address token
40  ) external {
41      ITaxManager taxManager = getTaxManager(factory);
42      uint256 protocolTaxRate = taxManager.getProtocolTaxRate(); //protocolTaxRate + rightUpTaxRate
43      uint256 taxDivisor = taxManager.getTaxBaseDivisor();
44      uint256 rebaseRate = getRebaser(factory).getDeltaForPositiveEpoch(
45          claimedEpoch
46      );
47      address handler = msg.sender;
48      address owner = IReferralHandler(handler).ownedBy(); //
49      if (rebaseRate != 0) {
50          uint256 blockForRebase = getRebaser(factory)
51              .getBlockForPositiveEpoch(claimedEpoch);
52          uint256 balanceDuringRebase = IETF(token).getPriorBalance(
53              owner,
54              blockForRebase
55          ); // We deal only with underlying balances
56          balanceDuringRebase = balanceDuringRebase.div(1e6); // 4.0 token internally stores 1e24 not 1
57          uint256 expectedBalance = balanceDuringRebase
58              .mul(BASE.add(rebaseRate))
59              .div(BASE);
60          uint256 balanceToMint = expectedBalance.sub(balanceDuringRebase);
61          handleSelfTax(
62              handler,
63              factory,
64              balanceToMint,
```

Figure 10 Source code of *handleReward* function

Recommendations	It is recommended to delete all transfer function of NFT.
Status	Partially Fixed. Although it is still possible to obtain referral rewards under special circumstances, but the protocol will do corresponding processing during initialization to avoid this problem according to the project party.

```
301     function _mint(address to, uint256 tokenId) internal virtual {
302         require(to != address(0), "ERC721: mint to the zero address");
303         require(!_exists(tokenId), "ERC721: token already minted");
304
305         _beforeTokenTransfer(address(0), to, tokenId, 1);
306
307         // Check that tokenId was not minted by `_beforeTokenTransfer` hook
308         require(!_exists(tokenId), "ERC721: token already minted");
309         require(_balances[to] == 0, "One address cannot have multiple tokens");
```

Figure 11 Source code of UniqueERC721 contract

```
373     function _transfer(
374         address from,
375         address to,
376         uint256 tokenId
377     ) internal virtual {
378         require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
379         require(to != address(0), "ERC721: transfer to the zero address");
380 
381         _beforeTokenTransfer(from, to, tokenId, 1);
382 
383         // Check that tokenId was not transferred by `_beforeTokenTransfer` hook
384         require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
385         require(_balances[to] == 0, "One address cannot have multiple tokens");
```

Figure 12 Source code of UniqueERC721 contract

[Quotafinance-6] Missing permission checks

Severity Level	**Low**
Type	Business Security
Lines	StakingFactoryDuration.sol#L30 StakingFactory.sol#L31
Description	The lack of permission verification in the *initialize* function of the StakingFactoryDuration and StakingFactory contracts will cause anyone to create a pool through this function and transfer any number of EScriptoken to the pool. This is unreasonable.

```
29  //duration抵押总时间
30  function initialize (address lp, uint256 amount, uint256 duration) public {
31      address escrowToken = address(new EscrowToken(amount)); //凭证代币
32      address stakingPool = address(new TokenRewardsDuration(escrowToken, lp, duration));
33      pools.push(stakingPool);
34      IERC20(escrowToken).transfer(stakingPool, amount);//打入凭证币
35      address poolEscrow = address(new PoolEscrow(escrowToken, stakingPool, token, nftFactory));
36      TokenRewards(stakingPool).setEscrow(poolEscrow);
37      TokenRewards(stakingPool).setRewardDistribution(notifier);
38      TokenRewards(stakingPool).transferOwnership(owner);
39      PoolEscrow(poolEscrow).setGovernance(owner);
40  }
41
```

Figure 13 Source code of StakingFactoryDuration contract (unfixed)

```
30  //创建池子进行初始化
31  function initialize (address lp, uint256 amount) public {
32      address escrowToken = address(new EscrowToken(amount));
33      address stakingPool = address(new TokenRewards(escrowToken, lp));
34      pools.push(stakingPool);
35      IERC20(escrowToken).transfer(stakingPool, amount);
36      address poolEscrow = address(new PoolEscrow(escrowToken, stakingPool, token, nftFactory));
37      TokenRewards(stakingPool).setEscrow(poolEscrow);
38      TokenRewards(stakingPool).setRewardDistribution(notifier);
39      TokenRewards(stakingPool).transferOwnership(owner);
40      PoolEscrow(poolEscrow).setGovernance(owner);
41  }
```

Figure 14 Source code of StakingFactory contract (unfixed)

Recommendations	It is recommended to increase the permissions verification.
Status	Fixed. The initialize call is now onlyOwner.

```
30  function initialize (address lp, uint256 amount, uint256 duration) public onlyOwner {
31      address escrowToken = address(new EscrowToken(amount));
32      address stakingPool = address(new TokenRewardsDuration(escrowToken, lp, duration));
33      pools.push(stakingPool);
34      IERC20(escrowToken).transfer(stakingPool, amount);
35      address poolEscrow = address(new PoolEscrow(escrowToken, stakingPool, token, nftFactory));
36      TokenRewards(stakingPool).setEscrow(poolEscrow);
37      TokenRewards(stakingPool).setRewardDistribution(notifier);
38      TokenRewards(stakingPool).transferOwnership(owner);
39      PoolEscrow(poolEscrow).setGovernance(owner);
40  }
```

Figure 15 Source code of StakingFactoryDuration contract (fixed)

```
30      function initialize (address lp, uint256 amount) public onlyOwner {
31          address escrowToken = address(new EscrowToken(amount));
32          address stakingPool = address(new TokenRewards(escrowToken, lp));
33          pools.push(stakingPool);
34          IERC20(escrowToken).transfer(stakingPool, amount);
35          address poolEscrow = address(new PoolEscrow(escrowToken, stakingPool, token, nftFactory));
36          TokenRewards(stakingPool).setEscrow(poolEscrow);
37          TokenRewards(stakingPool).setRewardDistribution(notifier);
38          TokenRewards(stakingPool).transferOwnership(owner);
39          PoolEscrow(poolEscrow).setGovernance(owner);
40      }
```

Figure 16 Source code of StakingFactory contract (fixed)

[Quotafinance-7] Multiple external *initialize* functions may cause misoperation

Severity Level	**Low**
Type	Coding Conventions
Lines	ETF.sol#L74-85, 705-721
Description	As shown in the figure below, there are two initialize functions in the ETF contract, and the *initialize* function can only be called once, but according to the actual business, it should be called "function *initialize*(string, string, uint8, address, uint256) public".

```
74  function initialize(
75      string memory name_,
76      string memory symbol_,
77      uint8 decimals_
78  )
79  public
80  {
81      require(etfsScalingFactor == 0, "already initialized");
82      name = name_;
83      symbol = symbol_;
84      decimals = decimals_;
85  }
```

Figure 17 Source code of ETFToken contract (unfixed)

Recommendations	It is recommended to set the visibility of *initialize* in the parent contract to “internal”.
Status	Fixed. The issue has been fixed according to suggestion.

```
74  function initialize(
75      string memory name_,
76      string memory symbol_,
77      uint8 decimals_
78  )
79  internal
80  {
81      require(etfsScalingFactor == 0, "already initialized");
82      name = name_;
83      symbol = symbol_;
84      decimals = decimals_;
85  }
```

Figure 18 Source code of ETFToken contract (fixed)

[Quotafinance-8] Inefficient loop

Severity Level	**Low**
Type	Coding Conventions
Lines	Multiple contracts
Description	When deleting the elements of the specified index in the array, this project uses sequential replacement, which is not a good way (it will consume unnecessary gas, and even cause Out of Gas issue).

```
function removeDistributorsByIndex(uint256 index) onlyAdmin public returns(address) {
    require(index < distributors.length);
    for (uint i = index; i<distributors.length-1; i++){
        distributors[i] = distributors[i+1];
    }
    address removedDistributor = distributors[distributors.length-1];
    distributors.pop();
    return removedDistributor;
}

function addPools(address[] memory _pools) onlyAdmin public {
    for (uint256 i = 0; i < _pools.length; i++) {
        pools.push(_pools[i]);
    }
}

function removePoolByIndex(uint256 index) onlyAdmin public returns(address) {
    require(index < pools.length);
    for (uint i = index; i<pools.length-1; i++){
        pools[i] = pools[i+1];
    }
    address removedPool = pools[pools.length-1];
    pools.pop();
    return removedPool;
}
```

Figure 19 Source code of related functions

Recommendations	Refer to: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/afb20119b33072da041c97ea717d3ce4417b5e01/contracts/utils/structs/EnumerableSet.sol#L72.
Status	Acknowledged. According to the project party, the number of pools are less than double digits and will rarely be removed.

[Quotafinance-9] The logic of the query function and the actual execution function are inconsistent

Severity Level	**Low**
Type	Business Security
Lines	BasicRebaser.sol #L299-338
Description	The calculation logic in the *calculateRealTimeRebasePreTax* function is inconsistent with that in rebase.

```
304     if (averageETF > highThreshold) {
305       // ETF is too expensive, this is a positive rebase increasing the supply
306       uint256 factor = BASE.sub(BASE.mul(averageETF.sub(averageSNP)).div(averageETF.mul(10)));
307       uint256 increase = BASE.sub(factor);
308       uint256 realAdjustment = increase.mul(BASE).div(factor);
309       uint256 currentSupply = IERC20(etf).totalSupply();
310       uint256 desiredSupply = currentSupply.add(currentSupply.mul(realAdjustment).div(BASE));
311       uint256 upperLimit = currentSupply.mul(basisBase.add(positiveRebaseLimit)).div(basisBase);
312       if(desiredSupply > upperLimit) // Increase expected rebase is above the limit
313         desiredSupply = upperLimit;
314       uint256 secondaryPoolBudget = desiredSupply.sub(currentSupply).mul(10).div(100);
315       desiredSupply = desiredSupply.sub(secondaryPoolBudget);
316
317       // Cannot underflow as desiredSupply > currentSupply, the result is positive
318       // delta = (desiredSupply / currentSupply) * 100 - 100
319       uint256 delta = desiredSupply.mul(BASE).div(currentSupply).sub(BASE);
320       return (delta, secondaryPool == address(0) ? 0 : secondaryPoolBudget);
321     } else if (averageETF < lowThreshold) {
```

Figure 20 Source code of *calculateRealTimeRebasePreTax* function (unfixed)

Recommendations	The query should be consistent with the actual execution logic.
Status	Fixed. The query function has been updated to now use the same logic as rebase.

```
298   function calculateRealTimeRebasePreTax() public view returns (uint256, uint256) {
299     // only rebase if there is a 5% difference between the price of SNP and ETF
300     uint256 highThreshold = averageSNP.mul(105).div(100);
301     uint256 lowThreshold = averageSNP.mul(95).div(100);
302
303     if (averageETF > highThreshold) {
304       // ETF is too expensive, this is a positive rebase increasing the supply
305       uint256 factor = BASE.sub(BASE.mul(averageETF.sub(averageSNP)).div(averageETF.mul(10)));
306       uint256 increase = BASE.sub(factor);
307       uint256 realAdjustment = increase.mul(BASE).div(factor);
308       uint256 currentSupply = IERC20(etf).totalSupply();
309       uint256 desiredSupply = currentSupply.add(currentSupply.mul(realAdjustment).div(BASE));
310       uint256 upperLimit = currentSupply.mul(basisBase.add(positiveRebaseLimit)).div(basisBase);
311       if(desiredSupply > upperLimit) // Increase expected rebase is above the limit
312         desiredSupply = upperLimit;
313       uint256 perpetualPoolTax = taxManager.getPerpetualPoolTaxRate();
314       uint256 totalTax = taxManager.getTotalTaxAtMint();
315       uint256 taxDivisor = taxManager.getTaxBaseDivisor();
316       uint256 secondaryPoolBudget = desiredSupply.sub(currentSupply).mul(perpetualPoolTax).div(taxDivisor);
317       uint256 totalRewardBudget = desiredSupply.sub(currentSupply).mul(totalTax).div(taxDivisor); // This am
318       desiredSupply = desiredSupply.sub(totalRewardBudget);
```

Figure 21 Source code of *calculateRealTimeRebasePreTax* function (fixed)

[Quotafinance-10] Flash loan risk

Severity Level	**Low**
Type	Business Security
Lines	UniswapOracle.sol#L27-31
Description	As shown in the figure below, this project uses the token storage of the corresponding pair to calculate the token price, which may be subject to flash loan attacks.

```
function getPriceETF() public view returns (bool, uint256) {
  // returns the price with 6 decimals, but we want 18
  uint256[] memory amounts = IUniswapRouterV2(router).getAmountsOut(1e18, path);
  return (etf != address(0), amounts[2].mul(1e12));
}
```

Figure 22 Source code of *getPriceETF* function

Recommendations	It is recommended to use Uniswap's TWAP mechanism to obtain token prices.
Status	Acknowledged. According to the project party, the *getPriceETF* function is only being used in the BasicRebaser contracts's *recordPrice* call, which has a check to log the price unless the msg.sender is tx.origin, we are also using 24 different sets of price logs and averaging them for our use case.

[Quotafinance-11] User may cannot withdraw from the staking pool normally

Severity Level	**Low**
Type	Business Security
Lines	PerpetualPoolEscrow.sol#L56-65
Description	When the user calls the *exit* function to exit the staking, he will call the release of the PerpetualPoolEscrow contract to settle rewards, but this contract has a *recoverTokens* function, and the administrator can withdraw the reward tokens of the contract, then if the number of reward tokens is insufficient, the user will not be able to call the *exit* function to exit normally.
Recommendations	It is recommended to check the contract balance and the number of reward tokens, and use the minimum value as the reward payout value when sending reward.
Status	Acknowledged. According to the project party, the issue would only occur if the admin removes the rewards from the Escrow pool. This would only happen in case of an emergency or migration.

```
56      function release(address recipient, uint256 shareAmount) external {
57          require(msg.sender == pool, "only pool can release tokens");
58          IERC20(shareToken).safeTransferFrom(msg.sender, address(this), shareAmount);
59          uint256 reward = getTokenNumber(shareAmount);
60          ITaxManager taxManager = ITaxManager(INFTFactory(factory).getTaxManager());
61          uint256 protocolTaxRate = taxManager.getProtocolTaxRate();
62          uint256 taxDivisor = taxManager.getTaxBaseDivisor();
63          distributeTaxAndReward(recipient, reward, protocolTaxRate, taxDivisor);
64          IERC20Burnable(shareToken).burn(shareAmount);
65      }
```

Figure 23 Source code of *release* function

[Quotafinance-12] Loss of precision

Severity Level	Info
Type	Coding Conventions
Lines	FixedTokenRewarder.sol#L49
Description	The calculation method using first division and then multiplication will result in loss of calculation precision.

```
47
48      function rewardRate() public view returns(uint256) {
49          uint256 baseRewardRate = uint256(1e18).div(3.154e11); // 3.154e7 is
50          return baseRewardRate.mul(yearlyRate);
51      }
```

Figure 24 Source code of *rewardRate* function (unfixed)

Recommendations	The calculation method uses multiplication before division.
Status	Fixed. The issue has been fixed according to suggestion.

```
48      function rewardRate() public view returns(uint256) {
49          uint256 baseRewardRate = uint256(1e18).mul(yearlyRate).div(3.154e11);
50          return baseRewardRate;
51      }
```

Figure 25 Source code of *rewardRate* function (fixed)

[Quotafinance-13] Code specification

Severity Level	Info
Type	Coding Conventions
Lines	Multiple contracts
Description	According to the naming convention of Solidity, only internal functions or variables are prefixed with _, but some non-internal functions and variables in this project are added with _. For example, some functions in the ETF contract shown in the figure below.

```
 */
function _setPendingGov(address pendingGov_)
external
onlyGov
{
  address oldPendingGov = pendingGov;
  pendingGov = pendingGov_;
  emit NewPendingGov(oldPendingGov, pendingGov_);
}

/** @notice lets msg.sender accept governance
 *
 */
function _acceptGov()
external
{
  require(msg.sender == pendingGov, "!pending");
  address oldGov = gov;
  gov = pendingGov;
  pendingGov = address(0);
  emit NewGov(oldGov, gov);
}
```

Figure 26 Source code of related function

Recommendations	This issue does not affect the function of the contract, and can choose whether to modify it according to actual needs.
Status	Acknowledged.

[Quotafinance-14] Token accuracy issue

Severity Level	Info
Type	Business Security
Lines	ApyOracle.sol#L29-62
Description	As shown in the figure below, in the *getApy* function of the ApyOracle contract, the default precision of non-USDC tokens is 18. If not, the calculated APY will be inaccurate; the default stakeToken precision is not greater than 18. If it is greater than 18, the function will call fail.

```
function getApy(
  address stakeToken,
  bool isUni,
  address token,
  uint256 incentive, // amount of token loaded into the contract
  uint256 howManyWeeks,
  address pool) public view returns (uint256) {
  address[] memory p = new address[](3);
  p[1] = wNative;
  p[2] = usdc;
  p[0] = token;
  uint256[] memory tokenPriceAmounts = IUniswapRouterV2(router).getAmountsOut(1e18, p);
  uint256 poolBalance = IERC20(stakeToken).balanceOf(pool);
  uint256 stakeTokenPrice = 1000000;
  p[0] = stakeToken;
  if (stakeToken != usdc) {
    if (isUni) {
      stakeTokenPrice = getUniPrice(IUniswapV2Pair(stakeToken));
    } else {
      uint256 unit = 10 ** uint256(ERC20Detailed(stakeToken).decimals());
      uint256[] memory stakePriceAmounts = IUniswapRouterV2(router).getAmountsOut(unit, p);
      stakeTokenPrice = stakePriceAmounts[2];
    }
  }
  uint256 temp = (
    1e8 * tokenPriceAmounts[2] * incentive * (52 / howManyWeeks)
  ) / (poolBalance * stakeTokenPrice);
  if (ERC20Detailed(stakeToken).decimals() == uint8(18)) {
    return temp;
  } else {
    uint256 divideBy = 10 ** uint256(18 - ERC20Detailed(stakeToken).decimals());
    return temp / divideBy;
  }
}
```

Figure 27 Source code of *getApy* function

Recommendations	The functions involved in this issue is only for query and do not affect the business. If required, the relevant logic of the function can be modified.
Status	Acknowledged. According to the project party, this is only used by external off-chain function and is completely readonly, this does not affect the protocol. If required, this can be adjusted on those services.

[Quotafinance-15] Missing event trigger

Severity Level	Info
Type	Coding Conventions
Lines	NFTFactory.sol#L109-139 EscrowToken.sol#L21-23 IRewardDistributionRecipient.sol#L18-23
Description	In the multiple contracts, there is no event trigger for following functions.

```
function setAdmin(address account) public onlyAdmin {
    admin = account;
}

function setDefaultURI(string memory _tokenURI) onlyAdmin public {
    tokenURI = _tokenURI;
}

function setRewarder(address _rewarder) onlyAdmin public {
    rewarder = _rewarder;
}

function setNFTAddress(address _NFT) onlyAdmin external {
    NFT = IMembershipNFT(_NFT); // Set address of the NFT contract
}

function setRebaser(address _rebaser) onlyAdmin external {
    rebaser = _rebaser; // Set address of the Rebaser contract
}

function setToken(address _token) onlyAdmin external {
    token = _token; // Set address of the Token contract
}
```

Figure 28 Source code of NFTFactory contract

```
21  function setOwner(address account) public onlyOwner {
22      owner = account;
23  }
```

Figure 29 Source code of *setOwner* function

```
18  function setRewardDistribution(INotifier _rewardDistribution)
19      external
20      onlyOwner
21  {
22      rewardDistribution = _rewardDistribution;
23  }
```

Figure 30 Source code of *setRewardDistribution* function

Recommendations	It is recommended to declare and trigger the corresponding events.
Status	Partially fixed. The following functions still have no corresponding events defined.

```
21  function setOwner(address account) public onlyOwner {
22      owner = account;
23  }
```

Figure 31 Source code of *setOwner* function

```
18  function setRewardDistribution(INotifier _rewardDistribution)
19      external
20      onlyOwner
21  {
22      rewardDistribution = _rewardDistribution;
23  }
```

Figure 32 Source code of *setRewardDistribution* function

[Quotafinance-16] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	PerpetualPoolEscrow.sol#L25 NFTFactory.sol#L7, 33 ReferralHandler.sol#L37 BasicRebaser.sol#L21 DepositBox.sol#L15 StakingEscrowHandler.sol#L12 BalanceManagement.sol#L101 TokenStorage.sol#L89 FixedPoolEscrow.sol#L24, 169-175 IRewardDistributionRecipient.sol#L13 PoolEscrow.sol#L25
Description	The following code is not used in the contract.

```
11  contract PerpetualPoolEscrow {
12
13      using SafeERC20 for IERC20;
14      using SafeMath for uint256;
15
16      modifier onlyGov() {
17          require(msg.sender == governance, "only governa
18          _;
19      }
20
21      address public shareToken;
22      address public pool;
23      address public token;
24      address public factory;
25      address public distributor;
```

Figure 33 Source code of PerpetualPoolEscrow contract

```
6  import "./interfaces/IDepositBox.sol";
7  import "./interfaces/ITierManager.sol";
8  import "./interfaces/IRebaserNew.sol";
9  import "@openzeppelin/contracts/utils/math/SafeMath.sol";
10 import { Clones } from "@openzeppelin/contracts/proxy/Clones.sol";
11
12 contract NFTFactory {
13
14     address public admin;
15     address public tierManager;
16     address public taxManager;
17     address public rebaser;
18     address public token;
19     address public handlerImplementation;
20     address public depositBoxImplementation;
21     address public rewarder;
22     mapping(uint256 => address) NFTToHandler;
23     mapping(address => uint256) HandlerToNFT;
24     mapping(uint256 => address) NFTToDepositBox;
25     mapping(address => bool) handlerStorage;
26     IMembershipNFT public NFT;
27     string public tokenURI;
28
29     event NewIssuance(uint256 id, address handler, address depositBox);
30     event LevelChange(address handler, uint256 oldTier, uint256 newTier);
31     event SelfTaxClaimed(address indexed handler, uint256 amount, uint256 timestamp);
32     event RewardClaimed(address indexed handler, uint256 amount, uint256 timestamp);
33     event DepositClaimed(address  indexed handler, uint256 amount, uint256 timestamp);
```

Figure 34 Source code of NFTFactory contract

```
17 contract ReferralHandler {
18
19     using SafeERC20 for IERC20;
20     using SafeMath for uint256;
21     address public factory;
22     IMembershipNFT public NFTContract;
23     IETF public token;
24     uint256 public nftID;
25     uint256 public mintTime;
26     address public referredBy; // NFT address of the refe
27     address[] public referrals;
28     address public depositBox;
29     uint256 private tier;
30     bool private canLevel;
31     uint256 public claimedEpoch; // Contructor sets the la
32     // NFT addresses of those referred by this NFT and its
33     address[] public firstLevelAddress;
34     address[] public secondLevelAddress;
35     address[] public thirdLevelAddress;
36     address[] public fourthLevelAddress;
37     uint256 public BASE;
38     bool public initialized = false;
```

Figure 35 Source code of ReferralHandler contract

```
13  contract BasicRebaser {
14
15    using SafeMath for uint256;
16    using SafeERC20 for IERC20;
17
18    event Updated(uint256 snp, uint256 etf);
19    event NoUpdateSNP();
20    event NoUpdateETF();
21    event NoSecondaryMint();
22    event NoRebaseNeeded();
```

Figure 36 Source code of BasicRebaser contract

```
12  contract DepositBox {
13      using SafeMath for uint256;
14      using SafeERC20 for IERC20;
15      address public admin;
16      address public factory;
17      uint256 public nftID;
```

Figure 37 Source code of DepositBox contract

```
 9  contract StakingEscrowHandler {
10
11      using SafeMath for uint256;
12      address public admin;
13      address public factory;
```

Figure 38 Source code of StakingEscrowHandler contract

```
97      function _delegate(address delegator, address delegatee)
98          internal
99      {
100         address currentDelegate = _delegates[delegator];
101         uint256 delegatorBalance = _etfBalances[delegator];
102         _delegates[delegator] = delegatee;
103
104         _moveDelegates(currentDelegate, delegatee, delegatorBalance);
105     }
```

Figure 39 Source code of BalanceManagement contract

```
88
89    mapping(address => uint256) internal _etfBalances;
90
```

Figure 40 Source code of TokenStorage contract

```
11  contract FixedPoolEscrow {
12
13      using SafeERC20 for IERC20;
14      using SafeMath for uint256;
15
16      modifier onlyGov() {
17          require(msg.sender == governanc
18          _;
19      }
20
21      address public pool;
22      address public token;
23      address public factory;
24      address public distributor;
25      address public governance;
```

Figure 41 Source code of FixedPoolEscrow contract

```
169 interface IERC20Burnable {
170     function burn(uint256 amount) external;
171 }
172
173 interface IERC20Mintable {
174     function mint(address to, uint256 amount) external;
175 }
```

Figure 42 Source code of FixedPoolEscrow contract

```
10  contract IRewardDistributionRecipient is Ownable {
11      INotifier public rewardDistribution;
12
13      function notifyRewardAmount(uint256 reward) external;
```

Figure 43 Source code of IRewardDistributionRecipient contract

```
11  contract PoolEscrow {
12
13      using SafeERC20 for IERC20;
14      using SafeMath for uint256;
15
16      modifier onlyGov() {
17          require(msg.sender == governance,
18          _;
19      }
20
21      address public shareToken;
22      address public pool;
23      address public token;
24      address public factory;
25      address public distributor;
26      address public governance;
```

Figure 44 Source code of PoolEscrow contract

Recommendations	It is recommended to delete the redundant code.
Status	Partially fixed. The following redundant code has not been removed.

```
97      function _delegate(address delegator, address delegatee)
98          internal
99      {
100         address currentDelegate = _delegates[delegator];
101         uint256 delegatorBalance = _etfBalances[delegator];
102         _delegates[delegator] = delegatee;
103
104         _moveDelegates(currentDelegate, delegatee, delegatorBalance);
105     }
```

Figure 45 Source code of BalanceManagement contract

```
88
89      mapping(address => uint256) internal _etfBalances;
90
```

Figure 46 Source code of TokenStorage contract

```
169 interface IERC20Burnable {
170     function burn(uint256 amount) external;
171 }
172
173 interface IERC20Mintable {
174     function mint(address to, uint256 amount) external;
175 }
```

Figure 47 Source code of FixedPoolEscrow contract

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com