BEOSIN
Blockchain Security

Kalax

Smart Contract Security Audit

No. 202405211051

May 21th, 2024

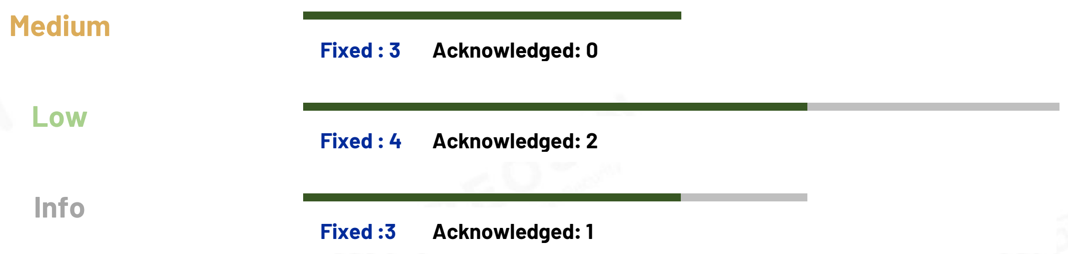
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 3 Medium items,6 Low items,and 4 info items were identified in the Kalax 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:

- When users withdraw principal, a certain handling fee will be charged. The handling rate is set by the project party and ranges from 0 to 0.2%.
- The Token contract mint all tokens to a EOA account during deployment, there is a certain risk of centralization.The Vault contract owner can set the strategy contract. If the strategy contract is set, the assets in the vault will be transferred to the strategy contract. The transfer of assets is managed by the vault contract owner, and there is a certain degree of centralization risk. Users should pay attention to the on-chain behavior of Token contract owner and Vault contract owner.

Business overview:

Kalax is a multi-pool, multi-reward staking project that allows users to stake multiple assets, including native token and other tokens. Each asset can only be staked in one pool. After staking assets, users can obtain a variety of reward tokens provided by the corresponding pool. The staking assets are first deposited into the vault contract. When the vault contract sets a strategy, the assets will be automatically deposited into the strategy contract. Users can withdraw staked assets and increase or decrease the staked amount. When increasing or decreasing the amount staked, the rewards for the previous staking period are settled and sent to the user. A handling fee will be deducted when the staked assets are retrieved. The handling rate is set by the project party and ranges from 0% to 0.2%. At the same time, the reward tokens received by users will also be deducted from the corresponding handling fees according to the set handling rate.

1 Overview

1.1 Project Overview

Project Name	Kalax
Project language	Solidity
Platform	Blast
Code base	https://github.com/Kalaxio/KalaxContracts/
Commit	8ee4d77d9667d2fb73f429e53c2961b408d91dbe 9f0fc44a74bce59e8c886b33eab6b65ee84de480

1.2 Audit Overview

Audit work duration: May 13, 2024 – May 21, 2024

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
Kalax-01	Native token cannot be withdrawn from strategy contract	Medium	Fixed
Kalax-02	The value of totalAllocPoint cannot be modified directly	Medium	Fixed
Kalax-03	Centralization risk	Medium	Fixed
Kalax-04	Gas fee problem	Low	Acknowledged
Kalax-05	WithdrawFee does not set a reasonable range	Low	Fixed
Kalax-06	Compilation problems	Low	Partially Fixed
Kalax-07	Pools should be updated	Low	Acknowledged
Kalax-08	Update the pool first, then set a new TokenPerBlock	Low	Fixed
Kalax-09	TotalUserRevenue and rewardDebt are not updated	Low	Fixed
Kalax-10	Function lacks check for depositing tokens and native tokens at the same time	Info	Fixed
Kalax-11	Missing trigger events	Info	Partially Fixed
Kalax-12	Redundant Code	Info	Partially Fixed
Kalax-13	Inappropriate data structure storage used	Info	Acknowledged

Finding Details:

[Kalax-01] Native token cannot be withdrawn from strategy contract

Severity Level	**Medium**
Type	Business Security
Lines	Vault.sol#178-180

Description	`depositNative` and `withdrawNative` are two functions in the strategy contract. When calling the depositNative method of the strategy contract in the vault contract to transfer native token to the strategy, The parameter of the `depositNative` function is incorrectly set to the user address and should be the Vault contract address. This will cause the strategy contract to record transfers to the user address instead of the Vault contract address. When the Vault contract address calls the `withdrawNative` function to withdraw user assets, it will fail, resulting in asset losses.
	`        if (address(strategy) != address(0)) {`
	`            IStrategy(strategy).depositNative{value:`
	`_amount}(_userAddr);`
	`        }`

Recommendation	It is recommended to modify _*userAddr* in the code to *address(this)*.

Status	**Fixed.**
	`        if (address(strategy) != address(0)) {`
	`            IStrategy(strategy).depositNative{value:`
	`_amount}(address(this));`
	`        }`

[Kalax-02] The value of totalAllocPoint cannot be modified directly

Severity Level	**Medium**
Type	Business Security
Lines	KalaxMultiRewardV2Farm.sol#191-193

Description	The `totalAllocPoint` is a state variable whose value is the sum of `allocPoint` values in all pools. When the value is modified directly through the `setTotalAllocPoint` function, it will cause problems with reward allocation.
	`    function setTotalAllocPoint(uint256 _totalAllocPoint) public onlyOwner {`
	`        totalAllocPoint = _totalAllocPoint;`
	`    }`

Recommendation	It is recommended to delete `setTotalAllocPoint` function.
Status	**Fixed.**

[Kalax-03] Centralization risk

Severity Level	**Medium**
Type	Business Security
Description	1. The Token contract mint all tokens to a EOA account during deployment, there is a certain risk of centralization 2. The Vault contract owner can set the strategy contract. If the strategy contract is set, the assets in the vault will be transferred to the strategy contract. The transfer of assets is managed by the vault contract owner, and there is a certain degree of centralization risk. 3. The KalaxMultiRewardV2Farm contract owner can call `setPoolAsset` to modify the asset. This is not allowed and is not necessary.
Recommendation	It is recommended to use a multi-signature wallet to manage tokens and the administrator rights of the vault contract and to delete the `setPoolAsset` function
Status	**Fixed.** The `setPoolAsset` function has been deleted. In addition, due to the lack of a standard multi-signature agreement on the Blast chain, the project team has not adopted a multi-signature solution; however, they have noticed this problem and promised to keep the private key properly.

[Kalax-04] Gas fee problem

Severity Level	Low
Type	General Vulnerability
Lines	KalaxMultiRewardV2Farm.sol#417-447
Description	When removing reward token, rewards need to be distributed separately to each user. When there are too many users, the gas fee will be too high, and even the function call will fail.

```
    function removeRewardTokenFromPool(uint256 _pid, IERC20
_rewardToken) public onlyOwner {
        require(address(_rewardToken) != address(0), "Invalid
rewardToken");
        PoolInfo storage pool = poolInfoList[_pid];
        // calculate the removing token rewards and transfer to user
        address[] memory userList = poolUserList[_pid].values();
        for (uint256 i = 0; i < userList.length; i++) {
            address userAddr = userList[i];
            uint256 _pendingRemovingRewards = pendingRewardToken(_pid,
userAddr, _rewardToken);
            if (_pendingRemovingRewards > 0) {
                safeTokenTransfer(_rewardToken, userAddr,
_pendingRemovingRewards);
            }
        }
        // find the reward token and remove it
        uint256 rewardLength = pool.rewards.length;
        for (uint256 i = 0; i < rewardLength; i++) {
            if (pool.rewards[i].token == _rewardToken) {
                pool.rewards[i] = pool.rewards[rewardLength - 1];
                pool.rewards.pop();
                rewardTokenSet.remove(address(_rewardToken));
                break;
            }
        }
        // update the pool
        updatePool(_pid);
    }
```

Recommendation	It is recommended to use another method to remove reward token. Specifically,

	set the block reward `tokenPerBlock` of the specified token to 0. Before setting `tokenPerBlock`, the pool needs to be updated. When removing reward token, rewards do not need to be distributed to users one by one. The next time the user performs a deposit or withdraw operation, the rewards will be automatically settled to the user.
Status	**Acknowledged**

[Kalax-05] WithdrawFee does not set a reasonable range

Severity Level	Low
Type	Business Security
Lines	KalaxMultiRewardV2Farm.sol#356-396,454-469

Description

The *addPool* function and *setPool* function do not check the reasonable range of the input parameter *_withdrawFee*. If *_withdrawFee* is set too large, the user will receive less funds than expected.

```
    function addPool(
        uint256 _allocPoints,
        address _token,
        bool _withUpdate,
        uint256 _withdrawFee,
        address _vault,
        bool _isEth,
        RewardInfo[] memory _rewards
      ) external onlyOwner {
      ...
       newPool.withdrawFee = _withdrawFee;
      ...
}
    function setPool(
        uint256 _pid,
        uint256 _allocPoints,
        bool _withUpdate,
        uint256 _withdrawFee
    ) external onlyOwner {
        ...
        poolInfoList[_pid].withdrawFee = _withdrawFee;
        ...
      }
```

Recommendation

It is recommended to set a reasonable range for the *withdrawFee* field of the pool.

Status

Fixed.

```
    function addPool(
        uint256 _allocPoints,
        address _token,
    bool _withUpdate,
    uint256 _withdrawFee,
    address _vault,
    bool _isEth,
    RewardInfo[] memory _rewards
) external onlyOwner {
    require(_withdrawFee <= 2, "Invalid withdraw fee");
    ...
}

function setPool(
    uint256 _pid,
    uint256 _allocPoints,
    bool _withUpdate,
    uint256 _withdrawFee
) external onlyOwner {
    require(_withdrawFee <= 2, "Invalid withdraw fee");
   ...
}
```

[Kalax-06] Compilation problems

Severity Level	Low
Type	General Vulnerability
Lines	ETHHelper.sol#4,Vault.sol#8-10,KalaxMultiRewardV2Farm.sol#9-11,145,330

Description	There are several problems in the project that cause the compilation to fail:
	1. Missing interface: The IETHHelper interface is imported in the ETHHelper contract, but the interface is not in the interface path.
	2. Import path error: When importing interfaces, the directory name in the path is incorrectly written as "interfaces" and should be "interface".In addition, when KalaxMultiRewardV2Farm and Vault contracts import interfaces, there are also errors in the path './' should be used, but it is incorrectly written as '../'.
	3. Data type mismatch: The initialize function attempts to assign a uint type value to the mapping type, which is not allowed in Solidity because they are different data types.
	4. Wrong and meaningless statement: The 330th line of code in the KalaxMultiRewardV2Farm contract is incorrect syntax and meaningless code.
	`import "../interfaces/IETHHelper.sol";`
	`import "../interfaces/IVault.sol";`
	`import "../comm/ETHHelper.sol";`
	`import "../comm/TransferHelper.sol";`
	`import "../comm/TransferHelper.sol";`
	`import "../interfaces/IVault.sol";`
	`import "../interfaces/IStrategy.sol";`
	`        totalUserRevenue = 0;`
	`        startBlock = block.number;`

Recommendation	It is recommended to add the IETHHelper interface, change Import path and delete `totalUserRevenue` and `startBlock` variable.
Status	**Partially Fixed.** Except for the second point where the directory name error has not been fixed, the other three points have been fixed.

[Kalax-07] Pools should be updated

Severity Level	Low
Type	Business Security
Lines	KalaxMultiRewardV2Farm.sol#356-396,454-469
Description	When a new pool is added, *totalAllocPoin* will change, and the proportion of each pool will also change. Pools need to be updated first. Modifying the pool's *allocPoint* also has this problem. (see code below)
Recommendation	It is recommended to update pools.
Status	**Acknowledged.**

```
  function addPool(
      uint256 _allocPoints,
      address _token,
      bool _withUpdate,
      uint256 _withdrawFee,
      address _vault,
      bool _isEth,
      RewardInfo[] memory _rewards
  ) external onlyOwner {
      ...
      if (_withUpdate) {
          updateMassPools();
      }
      ...
    }
  function setPool(
      uint256 _pid,
      uint256 _allocPoints,
      bool _withUpdate,
      uint256 _withdrawFee
  ) external onlyOwner {
      if (_withUpdate) {
          updateMassPools();
      }
      ...
  }
```

[Kalax-08] Update the pool first, then set a new TokenPerBlock

Severity Level	Low
Type	General Vulnerability
Lines	KalaxMultiRewardV2Farm.sol#154-164

Description	Before setting a new `TokenPerBlock` for the pool, you need to update the pool first.
	`function setPoolTokenPerBlock(uint256 _pid, uint256 _rewardIndex,`
	`uint256 _newTokenPerBlock) public onlyOwner {`
	`require(_pid >= 0, "Farm: invalid new pool pid");`
	`PoolInfo storage pool = poolInfoList[_pid];`
	`pool.rewards[_rewardIndex].tokenPerBlock = _newTokenPerBlock;`
	`// update the pool`
	`updateMassPools();`
	`}`

Recommendation	It is recommended to update the pool before setting a new `TokenPerBlock` for the pool.

Status	**Fixed.**
	`function setPoolTokenPerBlock(uint256 _pid, uint256 _rewardIndex,`
	`uint256 _newTokenPerBlock) public onlyOwner {`
	`require(_pid >= 0, "Farm: invalid new pool pid");`
	`// update the pool`
	`updateMassPools();`
	`PoolInfo storage pool = poolInfoList[_pid];`
	`pool.rewards[_rewardIndex].tokenPerBlock = _newTokenPerBlock;`
	`emit EventSetPoolTokenPerBlock(_pid, _rewardIndex,`
	`_newTokenPerBlock);`
	`}`

[Kalax-09] TotalUserRevenue and rewardDebt are not updated

Severity Level	Low
Type	General Vulnerability
Lines	KalaxMultiRewardV2Farm.sol#417-447
Description	When the reward token is removed, `totalUserRevenue` is not updated, causing the data of `totalUserRevenue` to be inaccurate.The user `rewardDebt` has not been updated. If the removed token is added again, the reward calculation will be incorrect.

```
    function removeRewardTokenFromPool(uint256 _pid, IERC20
_rewardToken) public onlyOwner {
        require(address(_rewardToken) != address(0), "Invalid
rewardToken");
        PoolInfo storage pool = poolInfoList[_pid];
        // calculate the removing token rewards and transfer to user
        address[] memory userList = poolUserList[_pid].values();
        for (uint256 i = 0; i < userList.length; i++) {
            address userAddr = userList[i];
            uint256 _pendingRemovingRewards = pendingRewardToken(_pid,
userAddr, _rewardToken);
            if (_pendingRemovingRewards > 0) {
                safeTokenTransfer(_rewardToken, userAddr,
_pendingRemovingRewards);
            }
        }
        // find the reward token and remove it
        uint256 rewardLength = pool.rewards.length;
        for (uint256 i = 0; i < rewardLength; i++) {
            if (pool.rewards[i].token == _rewardToken) {
                pool.rewards[i] = pool.rewards[rewardLength - 1];
                pool.rewards.pop();
                rewardTokenSet.remove(address(_rewardToken));
                break;
            }
        }
        // update the pool
        updatePool(_pid);
    }
```

Recommendation	It is recommended to update `totalUserRevenue` and `rewardDebt` when removing reward token.
Status	**Fixed.**

```
                user.rewardDebt[_rewardToken] =
user.rewardDebt[_rewardToken] + _pendingRemovingRewards;
                totalUserRevenue[_rewardToken] =
totalUserRevenue[_rewardToken] + _pendingRemovingRewards;
```

[Kalax-10] Function lacks check for depositing tokens and native tokens at the same time

Severity Level	Info
Type	Business Security
Lines	KalaxMultiRewardV2Farm.sol#616-630
Description	When a user deposits tokens, if accidentally transfers native tokens at the same time, the system will only record the deposited amount of tokens, causing the user to lose native tokens.

```
        if (address(pool.assets) == weth) {
            if (_amount > 0) {
                TransferHelper.safeTransferFrom(address(pool.assets), address(msg.sender), address(this), _amount);
                TransferHelper.safeTransfer(weth, address(wethHelper), _amount);
                wethHelper.withdrawETH(weth, address(this), _amount);
            }
            if (msg.value > 0) {
                _amount = _amount + msg.value;
            }
        } else {
            if (_amount > 0) {
                TransferHelper.safeTransferFrom(address(pool.assets), address(msg.sender), address(this), _amount);
            }
        }
```

Recommendation	It is recommended to add a check that the native token is 0 when depositing the token.
Status	**Fixed.**

```
        if (address(pool.assets) == weth) {
            if (_amount > 0) {
                TransferHelper.safeTransferFrom(address(pool.assets), address(msg.sender), address(this), _amount);
                TransferHelper.safeTransfer(weth, address(wethHelper), _amount);
                wethHelper.withdrawETH(weth, address(this), _amount);
            }
            if (msg.value > 0) {
                _amount = _amount + msg.value;
            }
        } else {
            require(msg.value == 0, "Deposit invalid token");
            if (_amount > 0) {
                TransferHelper.safeTransferFrom(address(pool.assets),
address(msg.sender), address(this), _amount);
            }
        }
```

[Kalax-11] Missing trigger events

Severity Level	Info
Type	Coding Conventions
Lines	KalaxMultiRewardV2Farm.sol#154-187,454-469,700-702,734-742,Vault.sol#81-116
Description	When the contract owner calls the set function to modify the key variable, many set functions will not trigger corresponding events, such as the `setPoolTokenPerBlock` function and the `setFarmPause` function.

```
    function setPoolTokenPerBlock(uint256 _pid, uint256 _rewardIndex,
        uint256 _newTokenPerBlock) public onlyOwner {
        require(_pid >= 0, "Farm: invalid new pool pid");
        PoolInfo storage pool = poolInfoList[_pid];
        pool.rewards[_rewardIndex].tokenPerBlock = _newTokenPerBlock;
        // update the pool
        updateMassPools();
    }
    function setFarmPause(bool _pause) external onlyOwner {
        _paused = _pause;
    }
```

Recommendation	It is recommended to emit events when modifying critical variables as it provides a standardized way to capture and communicate important changes within the contract. Events enable transparency and allow external systems and users to easily track and react to these modifications.
Status	**Partially Fixed.**

```
    function setPoolTokenPerBlock(uint256 _pid, uint256 _rewardIndex,
        uint256 _newTokenPerBlock) public onlyOwner {
        require(_pid >= 0, "Farm: invalid new pool pid");
        PoolInfo storage pool = poolInfoList[_pid];
        pool.rewards[_rewardIndex].tokenPerBlock = _newTokenPerBlock;
        // update the pool
        updateMassPools();
        emit EventSetPoolTokenPerBlock(_pid, _rewardIndex,
_newTokenPerBlock);
    }
```

[Kalax-12] Redundant Code

Severity Level	Info
Type	Coding Conventions
Lines	Token.sol#6,9,KalaxMultiRewardV2Farm.sol#116

Description	The imported ERC20Permit contract is not used, the inherited Ownable is not used, and the `userAddrList` in the KalaxMultiRewardV2Farm contract is not used. These are redundant codes.
	`import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";`
	`    Ownable(_owner){`
	`    EnumerableSet.AddressSet private userAddrList;`

Recommendation	It is recommended to delete redundant code.
Status	**Partially Fixed.** The import ERC20Permit contract statement has been deleted.

[Kalax-13] Inappropriate data structure storage used

Severity Level	Info
Type	General Vulnerability
Lines	Vault.sol#164
Description	If the same user interacts with the Vault contract multiple times, the `userList` variable will store the user's address multiple times. `userList.push(_userAddr);`
Recommendation	It is recommended to store the `userList` variable as a different data type that does not store duplicate values.
Status	**Acknowledged.**

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.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	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.

3.1.3 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.

3.1.4 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	Deprecated Items
		Redundant Code
		require/assert Usage
		Default Values
2	General Vulnerability	Insufficient Address Validation
		Lack Of Address Normalization
		Variable Override
		DoS (Denial Of Service)
		Function Call Permissions
		Call/Delegatecall Security
		Tx.origin Usage
		Returned Value Security
		Mathematical Risk
		Overriding Variables
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Arbitrage Attack
		Access Control

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 Rust 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.

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/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com