



BEOSIN
Blockchain Security

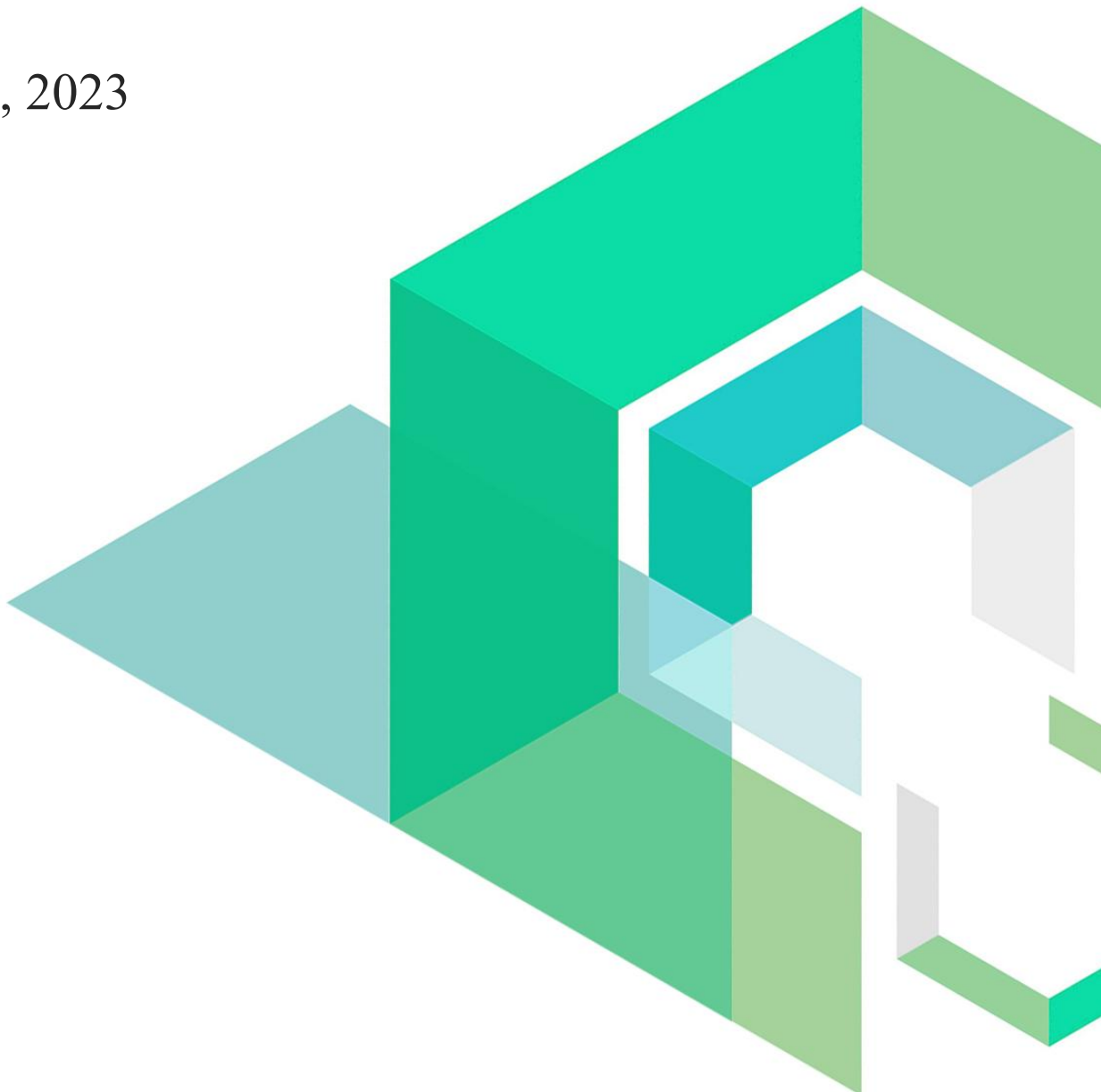
Earn Network

Smart Contract Security Audit

V1.0

No. 202305161600

May 16th, 2023

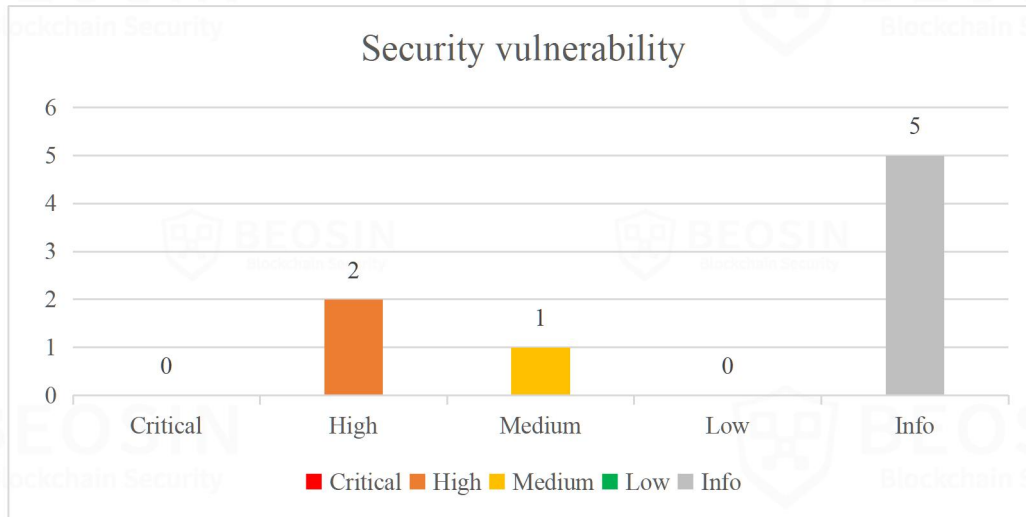


Contents

Summary of Audit Results	1
1 Overview	3
1.1 Project Overview	3
1.2 Audit Overview	3
2 Findings	4
[Earn Network-1] Reward Logic Update Error	5
[Earn Network-2] Owner has too much authority	6
[Earn Network-3] Fee Is Locked	7
[Earn Network-4] Missing Events	8
[Earn Network-5] Data update error	9
[Earn Network-6] Redundant Code	10
[Earn Network-7] No array length check	11
[Earn Network-8] USDT is not supported	13
3 Appendix	14
3.1 Vulnerability Assessment Metrics and Status in Smart Contracts	14
3.2 Audit Categories	16
3.3 Disclaimer	18
3.4 About Beosin	19

Summary of Audit Results

After auditing, **2 High, 1 Medium and 5 Info** risk items were identified in the Earn Network 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. The existence of the *emergencyWithdraw* function allows the owner to take all the funds in an emergency, and the user is advised to keep an eye on the management of the owner's address.

- **Project Description:**

- 1. Business overview**

The Earn Network project is a stake mining project. On this platform, users can add stake pools through factory, for which the project owner charges a fee. The project offers two types of staking pools. The first is a staking that can be taken away at any time and the user can choose the duration of the staking according to their needs. The longer the stake, the more rewards the user will receive. However, it is important to note that the pool has a start time and an end time, and if the user does not take the stake assets after the end time, no further rewards will be generated. Alternatively, the end time can be extended by the pool creator.

The second type of stake pool is one that has a fixed lock time. In this type of stake pool, the user needs to take out the principal and interest only after the lock-in time has expired. If the user needs to withdraw the tokens pledged earlier, a fee of a certain amount will be charged. Also, if the final number of pledges does not reach the maximum allowed, the pool creator can get back the excess rewards.

1 Overview

1.1 Project Overview

Project Name	Earn Network
Platform	EVM Compatible Chains
File Hash	8E660783D9BBABECB7F036F96FB74603E3EF0DCA2FD3D406EAB00D8B9131376A 8421BBE78F84BE9DAF143D522EBE4DBC671B1196C4E748CC4D46A1F3F2E82341 EC81C7DC7B329693EAEC98895B25E9FA1DCFB5C475277AC992294EDF0A87D40F 3BFE1D9760C1DC245E54DDE98A61BBF92FBA8D7F0B801ABD21ED62D0AF347C79

1.2 Audit Overview

Audit work duration: May 4, 2023 – May 16, 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
Earn Network-1	Reward Logic Update Error	High	Fixed
Earn Network-2	Owner has too much authority	High	Acknowledged
Earn Network-3	Fee Is Locked	Medium	Fixed
Earn Network-4	Missing Events	Info	Fixed
Earn Network-5	Data update error	Info	Acknowledged
Earn Network-6	Redundant Code	Info	Fixed
Earn Network-7	No array length check	Info	Fixed
Earn Network-8	USDT is not supported	Info	Acknowledged

Status Notes:

1. Earn Network-2 is unfixed, there is *emergencyWithdraw* function that allows the owner to take all the money in an emergency, we suggest that users pay attention to the management of the owner's address.
2. Earn Network-5 is unfixed and will not have any impact on the project.
3. Earn Network-8 is unfixed, so the program does not currently support USDT staking.

Finding Details:

[Earn Network-1] Reward Logic Update Error

Severity Level	High
Type	Business Security
Lines	FlexibleStaking.sol #L117
Description	The <i>withdraw</i> function is used to retrieve pledges and harvest rewards. When the user retrieves a portion of a pledge, the <i>staker.rewardDebt</i> will be updated to 0. This parameter indicates a pledge that the user has already taken and an incorrect update of this parameter will result in the user taking more rewards.

```

110     function withdraw(uint256 _amount) external {
111         Staker storage staker = stakers[msg.sender];
112         require(staker.amount > 0, "balance is zero");
113         require(staker.amount >= _amount, "amount > balance");
114         claimRewards();
115         staker.amount -= _amount;
116         amountOfTokensStaked -= _amount;
117         staker.rewardDebt = 0;
118         stakeToken.transfer(msg.sender, _amount);
119         staker.timestamp = 0;
120         emit Withdrawal(msg.sender, _amount, block.timestamp);
121     }
    
```

Figure 1 Source code of *withdraw* function (unfixed)

Recommendations	To update the <i>rewardDebt</i> , the <i>rewardDebt</i> should be calculated based on the updated amount.
-----------------	---

Status	Fixed.
--------	--------

```

135     function withdraw(uint256 _amount) external nonReentrant {
136         Staker storage staker = stakers[msg.sender];
137         require(staker.amount > 0, "balance is zero");
138         require(staker.amount >= _amount, "amount > balance");
139         _claimRewards();
140         staker.amount -= _amount;
141         amountOfTokensStaked -= _amount;
142         staker.rewardDebt =
143             (staker.amount * accRewardPerShare) /
144             REWARDS_PRECISION;
145         _withdrawTokensFromContract(msg.sender, _amount);
146         if(staker.amount == 0){
147             staker.timestamp = 0;
148         }
149         emit Withdrawal(msg.sender, _amount, block.timestamp);
150     }
    
```

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

[Earn Network-2] Owner has too much authority

Severity Level	High
Type	Business Security
Lines	LockedStaking.sol #L418-422, FlexibleStaking.sol #L231-234
Description	<p>The owner of the Factory contract has access to all project funds and a compromise of the private key could result in the loss of all project funds.</p> <pre> 418 function emergencyWithdraw(address _tokenAddress, uint256 _amount) external { 419 address owner = _factory.owner(); 420 require(msg.sender == owner, "Only protocol owner"); 421 IERC20(_tokenAddress).transfer(owner, _amount); 422 } </pre> <p>Figure 3 Source code of <i>emergencyWithdraw</i> function (unfixed)</p> <pre> 231 function emergencyWithdraw(address _tokenAddress, uint256 _amount) external { 232 address owner = IMYCStakingFactory(factory).owner(); 233 require(msg.sender == owner, "Only protocol owner"); 234 IERC20(_tokenAddress).transfer(owner, _amount); 235 } </pre> <p>Figure 4 Source code of <i>emergencyWithdraw</i> function (unfixed)</p>
Recommendations	<p>It is recommended that the owner's address use a multi-signature wallet.</p> <p>It is recommended that the <i>emergencyWithdraw</i> function be called by the user to forfeit the reward to retrieve their pledged tokens in an emergency.</p>
Status	Acknowledged.

[Earn Network-3] Fee Is Locked

Severity Level **Medium**

Type Business Security

Lines LockedStaking.sol #L375, #L400

Description The *claimFee* function is used to withdraw MYC fee, but here only the pledged portion is withdrawn, the remaining portion of the fee will be stranded in the contract and cannot be removed.

```

388     function claimFee() external returns (uint256) {
389         if (msg.sender != address(_factory)) {
390             revert OnlyFactory();
391         }
392         StakingPool memory sc = _stakePool;
393         if (sc.dateEnd >= block.timestamp && sc.dateEnd != 0)
394             revert StakingPeriodNotEnded();
395
396         uint256 sumRewards;
397         for (uint256 i = 0; i < _plans.length; i++) {
398             StakingPlan memory plan = _plans[i];
399             sumRewards +=
400                 (plan.rewardsWithdrawn * plan.mycFeesPool) /
401                 plan.rewardsPool;
402         }
403
404         if (sc.mycFeesWithdrawn >= sumRewards) {
405             revert NothingToWithdraw();
406         }
407
408         uint256 toWithdraw = sumRewards - sc.mycFeesWithdrawn;
409         IERC20(sc.tokenAddress).transfer(_factory.treasury(), toWithdraw);
410         sc.mycFeesWithdrawn += toWithdraw;
411         return (toWithdraw);
412     }
    
```

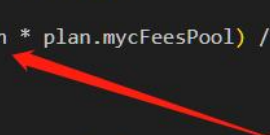


Figure 5 Source code of *claimFee* function (unfixed)

Recommendations It is recommended that the redundant fees be retrieved in the *claimUnusedRewards* function.

Status Fixed.

```

168         if (mycFeeSum > 0) {
169             IERC20(tokenAddress).transferFrom(
170                 msg.sender,
171                 _mycStakingManager.treasury(),
172                 mycFeeSum
173             );
174         }
    
```

Figure 6 Source code of *createPool* function(fixed)

[Earn Network-4] Missing Events

Severity Level	Info
Type	Coding Conventions
Lines	MYCStakingManager.sol #L128-142
Description	Missing events for important parameter changes.

```

128     function setTreasury(
129         address newTreasury
130     ) external noAddressZero(newTreasury) onlyOwner {
131         _treasury = newTreasury;
132     }
133
134     /**
135      * @dev Sets `newSigner` as new {signer} address
136      * @param newSigner new signer address
137      */
138     function setSigner(
139         address newSigner
140     ) external noAddressZero(newSigner) onlyOwner {
141         _signer = newSigner;
142     }
    
```

Figure 7 Source code of related function (unfixed)

Recommendations	It is recommended to add the corresponding event.
Status	Fixed.

```

136     /**
137      * @dev Sets `newTreasury` as new {treasury} address
138      * @param newTreasury new treasury address
139      */
140     function setTreasury(
141         address newTreasury
142     ) external noAddressZero(newTreasury) onlyOwner {
143         _treasury = newTreasury;
144         emit NewTreasury(newTreasury);
145     }
146
147     /**
148      * @dev Sets `newSigner` as new {signer} address
149      * @param newSigner new signer address
150      */
151     function setSigner(
152         address newSigner
153     ) external noAddressZero(newSigner) onlyOwner {
154         _signer = newSigner;
155         emit NewSigner(newSigner);
156     }
    
```

Figure 8 Source code of related function (fixed)

[Earn Network-5] Data update error

Severity Level	Info
Type	Business Security
Lines	FlexibleStaking.sol #L70-77, #L119
Description	The rewardAmount variable records the total reward of the pool, but is not updated when the pledge time is extended.

```

70     function extendStakingTime(uint256 _newEndDate) external {
71         require(msg.sender == creator, "creator mismatch");
72         require(_newEndDate > endTimestamp, "timestamp err");
73         uint256 tokenAmount = (_newEndDate - endTimestamp) *
74             rewardTokensPerSecond;
75         stakeToken.transferFrom(msg.sender, address(this), tokenAmount);
76         endTimestamp = _newEndDate;
77     }
    
```

Figure 9 Source code of *extendStakingTime* function (unfixed)

If not all withdraw, timestamp should not = 0.

```

110     function withdraw(uint256 _amount) external {
111         Staker storage staker = stakers[msg.sender];
112         require(staker.amount > 0, "balance is zero");
113         require(staker.amount >= _amount, "amount > balance");
114         claimRewards();
115         staker.amount -= _amount;
116         amountOfTokensStaked -= _amount;
117         staker.rewardDebt = 0;
118         stakeToken.transfer(msg.sender, _amount);
119         staker.timestamp = 0;
120         emit Withdrawal(msg.sender, _amount, block.timestamp);
121     }
    
```

Figure 10 Source code of *withdraw* function (unfixed)

Recommendations	Update the rewardAmount when the pledge time is extended. If not all withdraw, timestamp should not = 0.
------------------------	--

Status	Acknowledged.
---------------	---------------

[Earn Network-6] Redundant Code

Severity Level	Info
Type	Coding Conventions
Lines	LockedStaking.sol #L83
Description	<p><code>_withdrawnMYCSlots</code> is not used.</p> <pre> 79 StakingPool internal _stakePool; 80 mapping(address => mapping(uint256 => UserStake)) internal _userStake; 81 StakingPlan[] internal _plans; 82 IMYCStakingFactory internal _factory; 83 uint256 internal _withdrawnMYCSlots; </pre>

Figure 11 Source code of related code (unfixed)

Recommendations	Delete the relevant code
Status	Fixed.

```

79     StakingPool internal _stakePool;
80     mapping(address => mapping(uint256 => UserStake)) internal _userStake;
81     StakingPlan[] internal _plans;
82     IMYCStakingFactory internal _factory;
83
                    
```

Figure 12 Source code of related code (fixed)

[Earn Network-7] No array length check

Severity Level	Info
Type	Business Security
Lines	LockedStakingFactory.sol #L92-97
Description	The maxStakingAmount array length was not checked.

```

69     function createPool(
70         address poolOwner, // pool Owner
71         address tokenAddress, // staking token address
72         uint256[] memory durations, // for how long user cannot unstake
73         uint256[] memory maxTokensBeStaked, // maximum amount that can be staked among all stak
74         uint256[] memory rewardsPool, // reward pool for each duration
75         uint256[] memory mycFeesPool, //myc fees pools for each duration
76         uint256[] memory maxStakingAmount, //max staking amount
77         uint256 dateStart, // start date for all pools
78         uint256 dateEnd, // end date for all pools
79         uint256 deadline,
80         bytes memory signature
81     ) external {
82         //check pool owner
83         if (poolOwner != msg.sender && poolOwner != address(0)) {
84             revert WrongExecutor();
85         }
86
87         // checking dates
88         if (dateStart >= dateEnd) {
89             revert DatesSort();
90         }
91         // checking arrays
92         if (
93             durations.length != maxTokensBeStaked.length ||
94             maxTokensBeStaked.length != rewardsPool.length ||
95             rewardsPool.length != mycFeesPool.length ||
96             durations.length == 0
97         ) {
98             revert IncompleteArray();
    
```

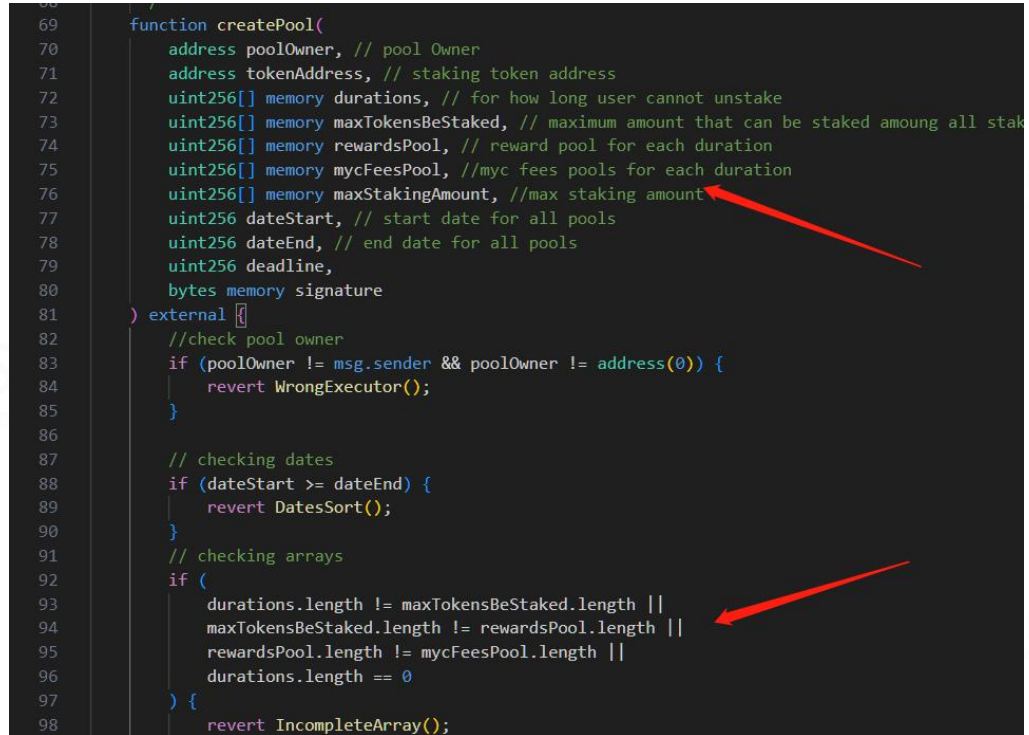


Figure 13 Source code of *createPool* function (unfixed)

Recommendations	Check the length of the maxStakingAmount array.
Status	Fixed.

```

75     function createPool(
76         address poolOwner, // pool Owner
77         address tokenAddress, // staking token address
78         uint256[] memory durations, // for how long user cannot unstake
79         uint256[] memory maxTokensBeStaked, // maximum amount that can be staked among all stakers for e
80         uint256[] memory rewardsPool, // reward pool for each duration
81         uint256[] memory mycFeesPool, //myc fees pools for each duration
82         uint256[] memory maxStakingAmount, //max staking amount
83         uint256 dateStart, // start date for all pools
84         uint256 dateEnd, // end date for all pools
85         uint256 deadline,
86         bytes memory signature
87     ) external {
88         //check pool owner
89         if (poolOwner != msg.sender && poolOwner != address(0)) {
90             revert WrongExecutor();
91         }
92
93         // checking dates
94         if (dateStart >= dateEnd) {
95             revert DatesSort();
96         }
97         // checking arrays
98         if (
99             durations.length != maxTokensBeStaked.length ||
100             maxTokensBeStaked.length != rewardsPool.length ||
101             rewardsPool.length != mycFeesPool.length ||
102             maxStakingAmount.length != mycFeesPool.length ||
103             durations.length == 0
104         ) {
105             revert IncompleteArray();
106         }
    
```

 Figure 14 Source code of *createPool* function (fixed)

[Earn Network-8] USDT is not supported

Severity Level	Info
Type	General Vulnerability
Lines	LockedStaking.sol #L4-7
Description	<p>The IERC20 interface is used in all projects and its <i>transfer</i> and <i>transferFrom</i> functions have bool return values. However, some token contracts do not have a return value, such as USDT, and the project cannot support these tokens.</p> <pre> 4 import "@openzeppelin/contracts/token/ERC20/IERC20.sol"; 5 import "../IMYCStakingFactory.sol"; 6 import "../IMYCStakingPool.sol"; 7 </pre>
Recommendations	Use OpenZeppelin's SafeERC20 library to transfer ERC-20 tokens.
Status	Acknowledged.

Figure 15 Source code of related code (unfixed)

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	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.



Official Website

<https://www.beosin.com>

Telegram

<https://t.me/+dD8Bnqd133RmNWNl>

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com

