BEOSIN
Blockchain Security

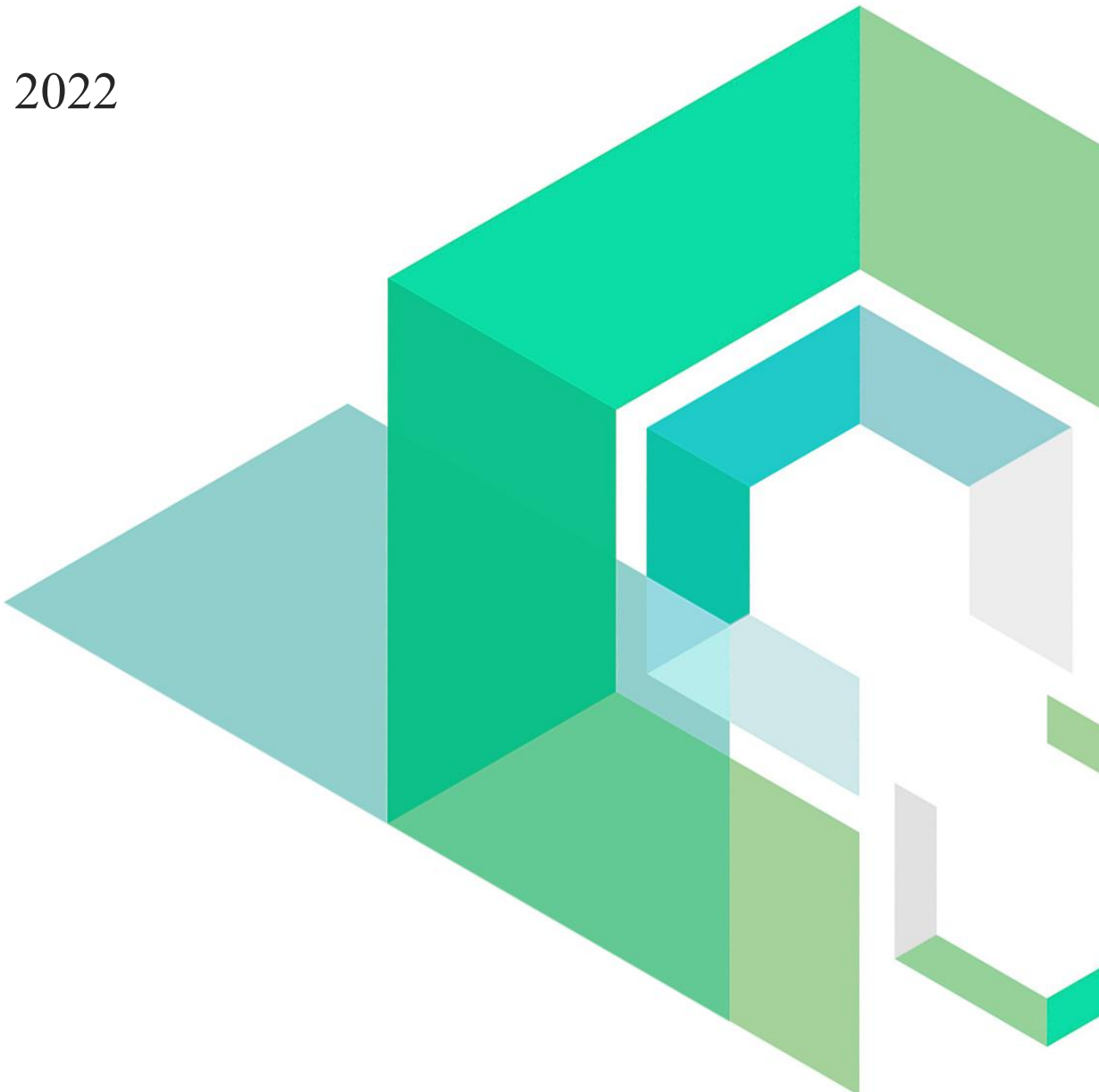
FTM

Smart Contract Security Audit

V1.1

No. 202203241830

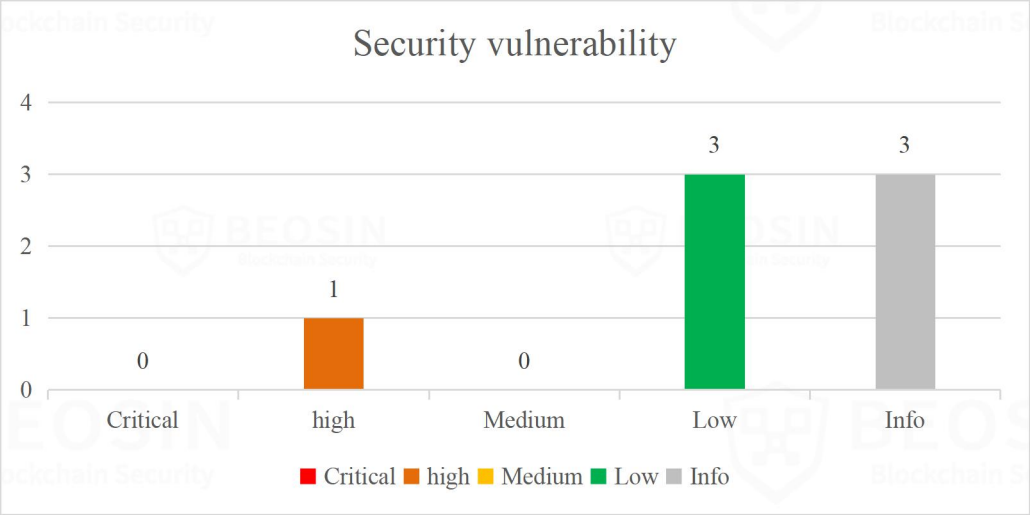
Mar 24th, 2022



Contents

Summary of audit results

After auditing, 1 High-risk, 3 Low-risks and 3 Info items were identified in the FTM 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. Centralization risk

The contract owner in the FTM project has full control of the contract. The owner can set the operator address, and both the owner and operator addresses can modify key parameters in the contract. For some parameters, only the owner has the permission to modify. The project party replies to retain the owner's full control over the contract. There may be some centralization risk.

- **Project Description:**

1. Basic Token Information

Token name	Ankr Fantom Reward Earning Bond
Token symbol	aFTMb
Decimals	18
Pre-mint	0
Total supply	Initial supply is 0 (Mintable, burnable)
Token type	ERC-20

Token name	Ankr FTM Reward Bearing Certificate
Token symbol	aFTMc
Decimals	18
Pre-mint	0
Total supply	The initial supply is the same as the total amount of aFTMb token shares at the time of deployment (Mintable, burnable)
Token type	ERC-20

2. Business overview

The FTM project contains two token contracts and two business contracts. In the aFTMb token contract, the number of shares is recorded inside the contract, and what the user queries is the number of bonds. Shares and bonds are converted according to a certain ratio (the ratio can be arbitrarily modified by the owner or operator address). In the aFTMc token contract, the total token supply is the same as the number of shares in the aFTMb token contract. aFTMb tokens and aFTMc tokens can be swapped in the aFTMb token contract, and a certain fee may be charged for the swap. Users can stake FTM tokens in the FantomPool contract to obtain aFTMb tokens, and the FTM staked by the user can be sent to the FantomStub contract by the owner or operator address. When users withdraw the FTM tokens staked by the FantomPool contract, they need to be operated by the operator address before they can be successfully withdrawn. The FantomStub contract is mainly used to interact with the SFC contract. The SFC contract is not within the scope of this audit.

1 Overview

1.1 Project Overview

Project Name	FTM	
Platform	Fantom	
File Hash (SHA256)	aFTMb_R1.sol	fc9d30dbff297974ff329783440eb2de9d44a7b6dfa8684d302395d34e7ccb4f (Initial) 67dd52344781a6a23d33b57a4d637e7482bfa2c9c4357765c8a167faf4b6cef5 (Final)
	FantomPool_R1.sol	a5871b1cc5430b6d5f57f5e3287efc53f7c9678d7377d0bdb2b8ba9dc5def3ac (Initial) ecfce4e30fa1dcf1954165f48ec0c00b5c8de2f40f141b4bae95dba249812942 (Final)
	FantomStub_R0.sol	ab4f9c1a0d0f8cca7af7d314faeeb7fb5decb62db60a0682c87a1b97e6924f86
	aFTMc_R0.sol	82e828d7adb1923c2beb5dfb167706a9779ea9baf71a1b1cb523cead9d996977

1.2 Audit Overview

Audit work duration: March 15, 2022 – March 24, 2022

Update Details: April 1, 2022. Code update.

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

Audit team: Beosin Technology Co. Ltd.

2 Findings

Index	Risk description	Severity level	Status
FTM-1	Operator has a high authority	High	Fixed
FTM-2	swapFeeOperator is not initialized and cannot be modified	Low	Fixed
FTM-3	Incorrect function call	Low	Fixed
FTM-4	Centralization risk	Low	Acknowledged
FTM-5	The corresponding event is not triggered	Info	Fixed
FTM-6	Public functions can be declared external	Info	Partially Fixed
FTM-7	Token name and symbol can be modified arbitrarily	Info	Fixed

Risk Details Description:

1. FTM-4 is not fixed and may cause a potential centralization risk.
2. FTM-6 is not fully fixed but does not cause security issues.

[FTM-1] Operator has a high authority

Severity Level	**High**
Type	Business Security
Lines	aFTMb_R1.sol#L115-137, L147-150
Description	In the aFTMb_R1 contract, the operator can call the *burnBondsFrom* and *burnSharesFrom* functions to burn aFTMb tokens at any address, and call the *mintBondsTo* and *mintSharesTo* functions to mint tokens to any address. There is a problem of excessive permissions.

```
function mintBondsTo(address account, uint256 amount) public override onlyPoolOrOperator {
    uint256 shares = balanceToShares(amount);
    _mint(account, shares);
    emit Transfer(address(0), account, amount);
}

function burnBondsFrom(address account, uint256 amount) public override onlyPoolOrOperator {
    uint256 shares = balanceToShares(amount);
    _burn(account, shares);
    emit Transfer(account, address(0), amount);
}

function mintSharesTo(address account, uint256 shares) public override onlyPoolOrOperator {
    _mint(account, shares);
    uint256 amount = sharesToBalance(shares);
    emit Transfer(address(0), account, amount);
}

function burnSharesFrom(address account, uint256 shares) public override onlyPoolOrOperator {
    _burn(account, shares);
    uint256 amount = sharesToBalance(shares);
    emit Transfer(account, address(0), amount);
}
```

Figure 1 Source code of related functions

```
modifier onlyPoolOrOperator() {
    require(msg.sender == pool || msg.sender == operator, "onlyPoolOrOperator: not allowed");
    _;
}
```

Figure 2 Source code of *onlyPoolOrOperator* modifier

Recommendations	It is recommended to modify the *burnBondsFrom*, *burnSharesFrom*, *mintBondsTo* and *mintSharesTo* functions to be called only by the pool contract.
Status	Fixed.

```
function mintBondsTo(address account, uint256 amount) public override onlyPool {
    uint256 shares = balanceToShares(amount);
    _mint(account, shares);
    emit Transfer(address(0), account, amount);
}

function burnBondsFrom(address account, uint256 amount) public override onlyPool {
    uint256 shares = balanceToShares(amount);
    _burn(account, shares);
    emit Transfer(account, address(0), amount);
}

function mintSharesTo(address account, uint256 shares) public override onlyPool {
    _mint(account, shares);
    uint256 amount = sharesToBalance(shares);
    emit Transfer(address(0), account, amount);
}

function burnSharesFrom(address account, uint256 shares) public override onlyPool {
    _burn(account, shares);
    uint256 amount = sharesToBalance(shares);
    emit Transfer(account, address(0), amount);
}
```

Figure 3 Source code of related functions

```
modifier onlyPool() {
    require(msg.sender == pool, "onlyPool: not allowed");
    _;
}
```

Figure 4 Source code of *onlyPool* modifier

[FTM-2] swapFeeOperator is not initialized and cannot be modified

Severity Level	**Low**
Type	Business Security
Lines	aFTMb_R1.sol#L31
Description	In the aFTMb token contract, the swapFeeOperator address is not initialized and cannot be changed.

```
address public crossChainBridge;
address public certToken; // aFTMc

address public swapFeeOperator;
uint256 public swapFeeRatio;
```

Figure 5 Source code of variable swapFeeOperator

Recommendations	It is recommended to add a function to modify the variable swapFeeOperator.
Status	Fixed.

```
function changeSwapFeeOperator(address newSwapFeeOperator) external onlyOwner {
    address oldSwapFeeOperator = swapFeeOperator;
    swapFeeOperator = newSwapFeeOperator;
    emit SwapFeeOperatorChanged(oldSwapFeeOperator, newSwapFeeOperator);
}
```

Figure 6 Source code of *changeSwapFeeOperator* function

[FTM-3] Incorrect function call

Severity Level	**Low**
Type	Business Security
Lines	aFTMc_R0.sol#L90
Description	The *balanceWithRewardsOf* function in the aFTMc_R0 contract calls the *balanceToShares* function of the aFTMb token contract, but the input is shares.

```
function balanceWithRewardsOf(address account) public view returns (uint256) {
    uint256 shares = this.balanceOf(account);
    return IAnkrBond_R1(bondToken).balanceToShares(shares);
}
```

Figure 7 Source code of *balanceWithRewardsOf* function (Unfixed)

Recommendations	It is recommended to modify the called function to *sharesToBalance*.
Status	Fixed.

```
function balanceWithRewardsOf(address account) public view returns (uint256) {
    uint256 shares = this.balanceOf(account);
    return IAnkrBond_R1(bondToken).sharesToBalance(shares);
}
```

Figure 8 Source code of *balanceWithRewardsOf* function (Fixed)

[FTM-4] Centralization risk

Severity Level	**Low**
Type	Business Security
Lines	aFTMb_R1.sol#L49-53 FantomPool_R0.sol#L114-122, L162-168, L174-188 FantomStub_R0.sol#L78-82
Description	The operator in the aFTMb_R1 contract can call the *updateRatio* function to modify the ratio of bonds and shares, and the *changeOperator* and *changePoolContract* functions to change operator and pool address. In the FantomStub_R0 contract, the operator and owner can call the *setWithdrawalBounds* function to modify the data corresponding to validatorId. The operator and owner in the FantomPool_R0 contract can call the *updateFeeParameters* function to modify the fee ratio in the contract. The owner can also call functions such as *changeOperator*, *changeBondContract*, *setMinimumStake*, *allowDeploy*, *disallowDeploy*, etc. There may be some centralization risk.

```
function updateRatio(uint256 newRatio) public override onlyPoolOrOperator {
    require(newRatio > 0, "Ratio must be positive");
    _ratio = newRatio;
    emit RatioUpdate(_ratio);
}
```

Figure 9 Source code of *updateRatio* function

```
function changeOperator(address newOperator) external onlyOwner {
    address oldOperator = operator;
    operator = newOperator;
    emit OperatorChanged(oldOperator, newOperator);
}

function changeBondContract(address newBondContract) external onlyOwner {
    address oldBondContract = bondContract;
    bondContract = newBondContract;
    emit BondContractChanged(oldBondContract, newBondContract);
}
```

Figure 10 Source code of *changeOperator* and *changeBondContract* functions

Recommendations	It is recommended to use DAO, governance contracts or multi-signature wallets as operator and owner.
Status	Acknowledged.

[FTM-5] The corresponding event is not triggered

Severity Level	Info
Type	Coding Conventions
Lines	FantomPool_R0.sol#L162-168, L174-188
Description	The *changeOperator*, *changeBondContract*, *setMinimumStake*, *setSFC*, *allowDeploy* and *disallowDeploy* functions in the FantomPool_R0 contract do not trigger corresponding events.

```
function changeOperator(address _operator) public onlyOwner {
    operator = _operator;
}

function changeBondContract(address _bondContract) public onlyOwner {
    bondContract = _bondContract;
}
```

Figure 11 Source code of related functions (Unfixed)

```
function setMinimumStake(uint256 minStake) public onlyOperator {
    minimumStake = minStake;
}

function setSFC(address _sfc) public onlyOwner {
    sfc = _sfc;
}

function allowDeploy(address deployer) external onlyOperator {
    canDeployMap[deployer] = true;
}

function disallowDeploy(address deployer) external onlyOperator {
    delete canDeployMap[deployer];
}
```

Figure 12 Source code of related functions (Unfixed)

Recommendations	It is recommended to declare and trigger the corresponding event.
Status	Fixed.

```
function changeOperator(address newOperator) external onlyOwner {
    address oldOperator = operator;
    operator = newOperator;
    emit OperatorChanged(oldOperator, newOperator);
}

function changeBondContract(address newBondContract) external onlyOwner {
    address oldBondContract = bondContract;
    bondContract = newBondContract;
    emit BondContractChanged(oldBondContract, newBondContract);
}
```

Figure 13 Source code of related functions (Fixed)

```
function setMinimumStake(uint256 newMinimalStake) external onlyOperator {
    minimumStake = newMinimalStake;
    emit MinimumStakeChanged(newMinimalStake);
}

function allowDeploy(address deployer) external onlyOperator {
    canDeployMap[deployer] = true;
    emit DeployAllowed(deployer);
}

function disallowDeploy(address deployer) external onlyOperator {
    delete canDeployMap[deployer];
    emit DeployDisallowed(deployer);
}
```

Figure 14 Source code of related functions (Fixed)

[FTM-6] Public functions can be declared external

Severity Level	Info
Type	Coding Conventions
Lines	FantomPool_R0.sol#L162-168, L174-180 aFTMb_R1.sol#L23-28, L53-57, L76-85, L91-141 FantomStub_R0.sol#L29-36, L40-82
Description	Functions such as changeOperator and changeBondContract function use the public modifier, which may cause more gas consumption.

```
function changeOperator(address _operator) public onlyOwner {
    operator = _operator;
}

function changeBondContract(address _bondContract) public onlyOwner {
    bondContract = _bondContract;
}
```

Figure 15 Source code of *changeBondContract* and *changeOperator* functions (Unfixed)

Recommendations	It is recommended to modify the visibility of only externally called functions to external.
Status	Partially Fixed. The visibility of some functions has been changed to external.

```
function changeOperator(address newOperator) external onlyOwner {
    address oldOperator = operator;
    operator = newOperator;
    emit OperatorChanged(oldOperator, newOperator);
}

function changeBondContract(address newBondContract) external onlyOwner {
    address oldBondContract = bondContract;
    bondContract = newBondContract;
    emit BondContractChanged(oldBondContract, newBondContract);
}
```

Figure 16 Source code of *changeBondContract* and *changeOperator* functions (Fixed)

[FTM-7] Token name and symbol can be modified arbitrarily

Severity Level	Info
Type	Business Security
Lines	aFTMb_R0.sol#L37-43
Description	The name and symbol of the aFTMb_R1 contract can be modified repeatedly.

```
function setName(string calldata name) external onlyOwner {
    _name = name;
}

function setSymbol(string calldata symbol) external onlyOwner {
    _symbol = symbol;
}
```

Figure 17 Source code of *setName* and *setSymbol* functions

Recommendations	It is recommended to remove the *setName* and *setSymbol* functions.
Status	Fixed. The related function has been removed.

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

Affiliated to BEOSIN Technology Pte. Ltd., 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