BEOSIN
Web3 Security & Compliance

TONCO

Smart Contract Security Audit

No. 202411221000

Nov 22nd, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 2 Low ,3 Info item was identified in the TONCO project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

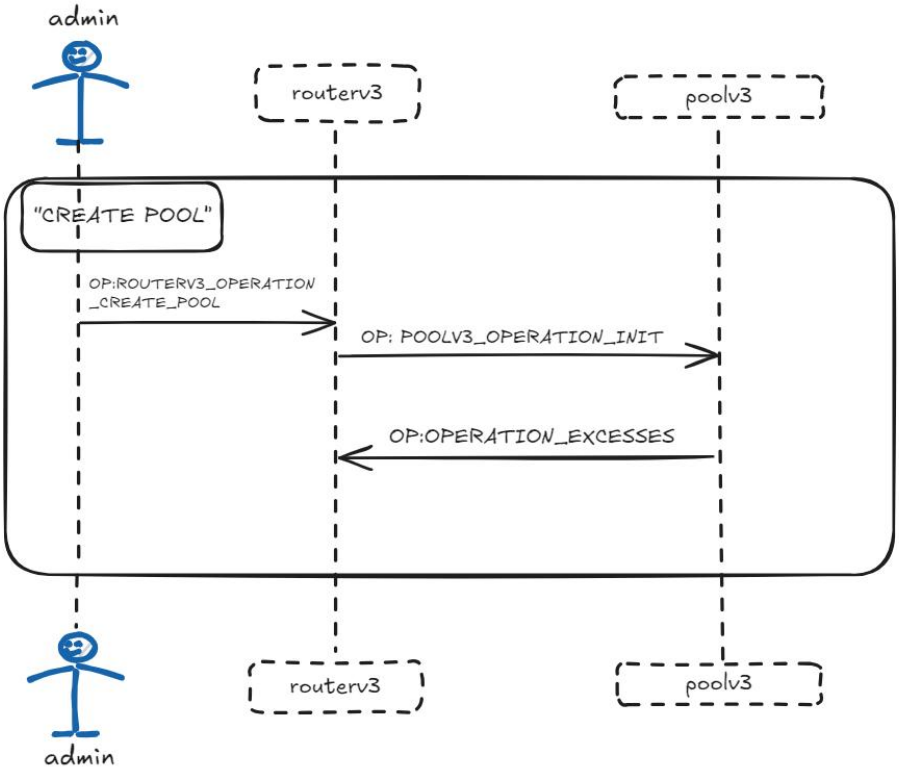
Low	Fixed : 0 Acknowledged: 2
Info	Fixed : 0 Acknowledged: 3

- **Project Description:**

The audited TONCO project is a decentralized exchange (DEX) platform, consisting of four core modules: The Router module is responsible for managing pool funds, handling the receipt and sending of funds, and supporting pool creation and related settings by administrators; the Account module allows users to add liquidity; the NFT module records the data of each liquidity addition by users, generating a new NFT each time liquidity is added; the Pool module provides functionality for adding and removing liquidity as well as token swaps. For a detailed process, please refer to the explanation below:

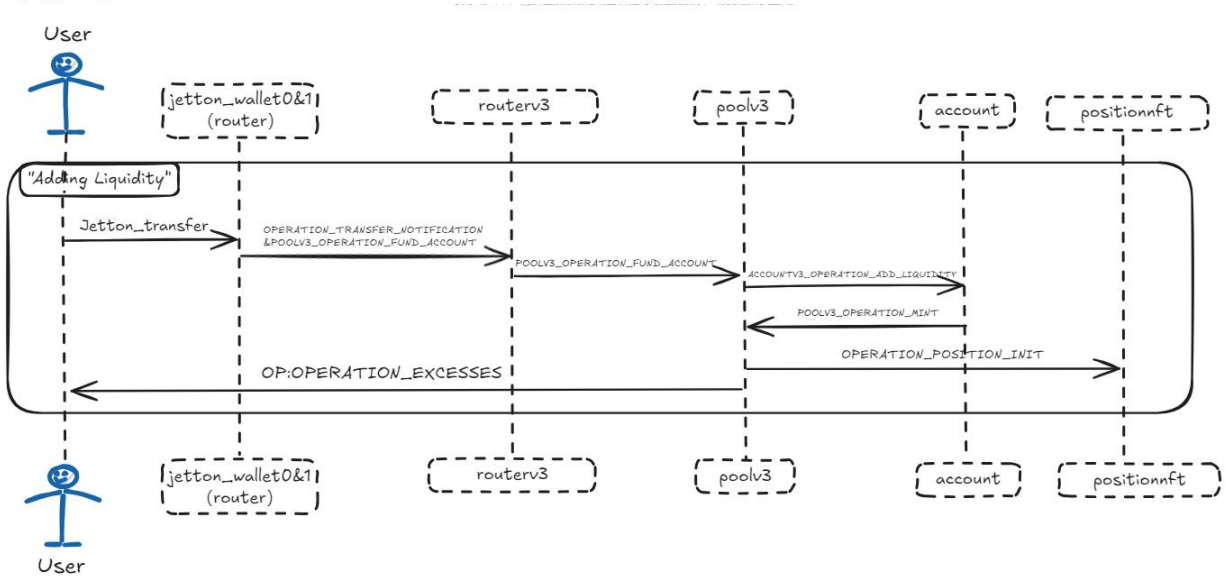
✧ Create pool function：

This diagram shows the process of the admin creating a pool. In this process, the admin first sends the `ROUTERV3_OPERATION_CREATE_POOL` command to `routerv3`, which includes information such as the `jetton_wallet0` and `jetton_wallet1` wallet address for the pool being created. Upon receiving the command, `routerv3` calculates the pool's address and sends the `POOLV3_OPERATION_INIT` operation to `poolv3` to initialize the pool. Once the pool initialization is complete, `poolv3` sends the `OPERATION_EXCESSES` command to `routerv3` to refund any excess gas.



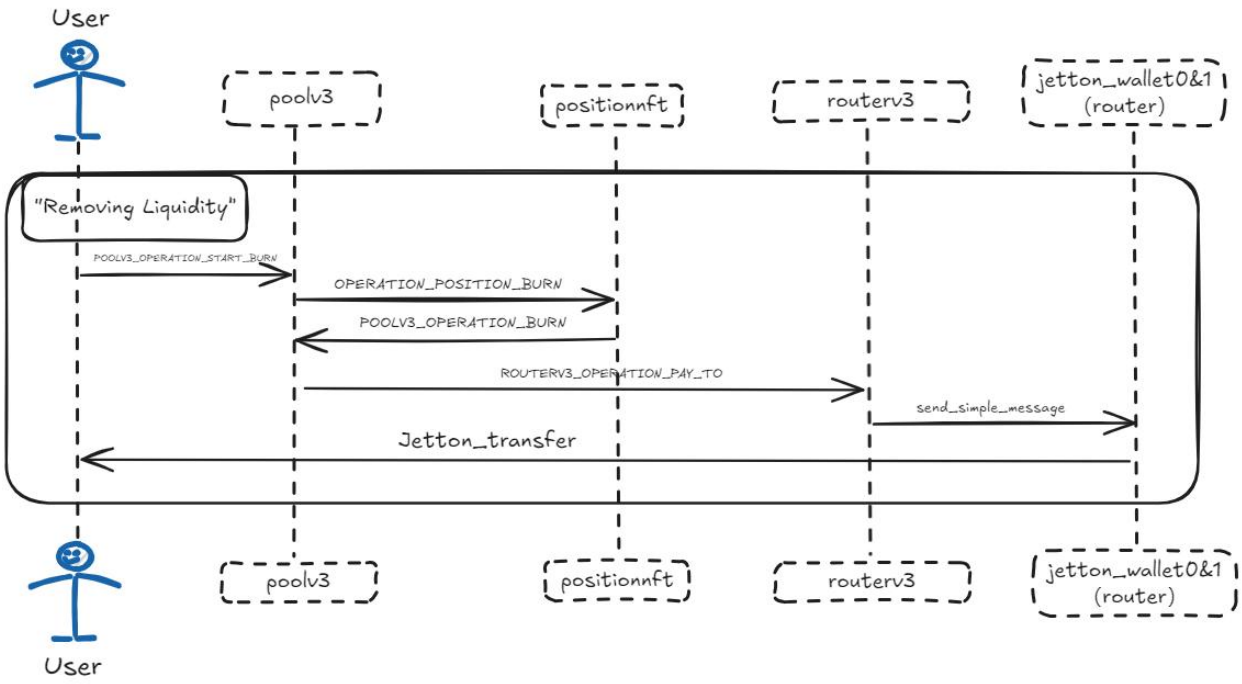
✧ Adding Liquidity function：

Below is the process flow for adding liquidity. First, the user transfers funds to the router contract's wallet0 and wallet1, as adding liquidity requires two types of tokens. Once the relevant wallets in the router contract receive the transfer, they notify the routerv3 contract to call the `POOLV3_OPERATION_FUND_ACCOUNT` operation to prepare for the liquidity addition. This step primarily involves calculating the poolv3 address and packaging the message. Next, routerv3 sends the `POOLV3_OPERATION_FUND_ACCOUNT` command to the poolv3 contract to compute the user's account address and package the corresponding message. The poolv3 contract then sends the `ACCOUNTV3_OPERATION_ADD_LIQUIDITY` command to the user's Account contract, where it verifies that the liquidity amount meets the requirements. If the requirements are met, the Account contract sends the `POOLV3_OPERATION_MINT` command to the poolv3 contract, which then calculates the user's NFT address. The poolv3 contract subsequently sends the `POOLV3_OPERATION_MINT` command to the user's NFT contract to record the amount of liquidity added and related parameters, as well as to initialize the NFT. Finally, the poolv3 contract calls the `OP:OPERATION_EXCESSES` command to refund the remaining gas to the user.



✧ Removing Liquidity function：

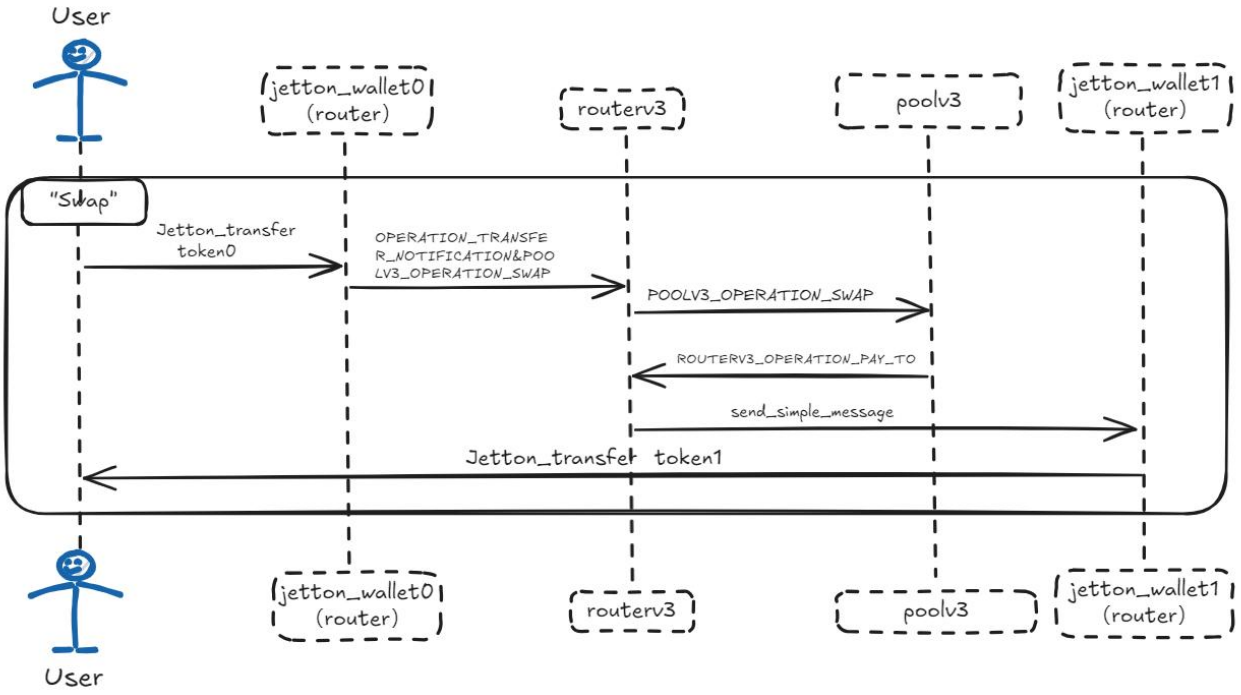
First, the user initiates a POOLV3_OPERATION_START_BURN operation request to the poolv3 contract. Upon receiving this instruction, poolv3 calculates the address of positionnft and updates the latest fee data such as feeGrowthInside0X128 and feeGrowthInside1X128. It then packages this data and sends an OPERATION_POSITION_BURN instruction to the positionnft contract. After receiving the instruction, the positionnft contract updates the user's feeGrowthInside0X128, feeGrowthInside1X128, and other parameters, then sends a POOLV3_OPERATION_BURN instruction to the routerv3 contract to calculate the rewards and corresponding liquidity amount to be removed for the user. Subsequently, routerv3 sends send_simple_message instructions to the wallet0 and wallet1 addresses of the routerv3 contract to execute the transfer. Finally, the corresponding funds are transferred to the user.



✧ Swap function：

The process of the user's token swap is as follows: First, the user transfers the token0 to the router's jetton_wallet0. Once the wallet receives token0, it sends the OPERATION_TRANSFER_NOTIFICATION instruction to the routerv3 contract, notifying routerv3 to perform the POOLV3_OPERATION_SWAP operation. This instruction is primarily used to package the message and calculate the corresponding pool address for token0 and token1. Then, routerv3 sends the POOLV3_OPERATION_SWAP instruction to poolv3. After receiving the message, poolv3 calculates the amount of token1 to be exchanged.

Subsequently, routerv3 sends the ROUTERV3_OPERATION_PAY_TO instruction to notify the router to initiate the transfer. Finally, routerv3 uses the send_simple_message instruction to notify jetton_wallet1 to transfer token1 to the user, completing the entire swap process.



1 Overview

1.1 Project Overview

Project Name	TONCO
Project language	Func
Platform	Ton Chain
Base code	https://github.com/cryptoTONCO/TONCO-ton-contracts/tree/apimenov_review_29_10_24
Commit	9ee8ae79b8a4f51ddec9b486051427ab5ec88807

1.2 Audit Overview

Audit work duration: Nov 6, 2024 – Nov 22, 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
TONCO-01	Incentive calculation flaws	Low	Acknowledged
TONCO-02	Lack of Lock function	Low	Acknowledged
TONCO-03	The pool can be initialized multiple times	Info	Acknowledged
TONCO-04	Poor calibration	Info	Acknowledged
TONCO-05	Redundant code	Info	Acknowledged

Finding Details:

[TONCO-01] Incentive calculation flaws

Severity Level	Low
Type	Business Security
Lines	poolv3.func#L396
Description	Because TON messages are processed asynchronously, the following scenario may occur: assume that the user removes mobility with a default `feeGrowthInside0LastX128` of 0 and an incoming `feeGrowthInside0CurrentX128` of 100. The message to remove mobility is first sent to the `positionnft` contract, which updates the user's `feeGrowthInside0LastX128` to 100. The remove mobility message is first sent to the `positionnft` contract, which updates the user's `feeGrowthInside0LastX128` to 100. The message is then sent to the pool contract to calculate the reward. However, during this process, the pool contract's `feeGrowthInside0X128` is updated to 110. At this point, the user's incoming `feeGrowthInside0CurrentX128` is 0, and the reward is calculated as 0 - 110, but is recorded as 100 in the `positionnft` contract. After that, the user can continue to calculate the reward based on the fact that `feeGrowthInside0LastX128` is 100, resulting in the user receiving an additional portion of the reward.
Recommendation	It is recommended that the latest `feeGrowthInside0X128` and `feeGrowthInside1X128` be passed to the `positionnft` contract to allow for updates and to ensure that the calculations are based on the latest fee growth data.
Status	**Acknowledged.** Description of the project party: It's quite difficult to make a synchronous system in an async environment, so we decided to go with our solution. Yes, a user that burns the NFT would lose the reward that arrives exactly during the burn process. However we think that if a user commits to burn knowing the state of the chain in t_0 it can't predict this reward. So for him/her nothing of value is lost.

[TONCO-02] Lack of Lock function

Severity Level	Low
Type	Business Security
Lines	routerv3.func
Description	The `is_locked` variable is defined in the router contract, but the contract does not include a function to modify `is_locked`.
Recommendation	It is recommended that the router contract add admin's modification of `is_locked` and implement the corresponding lock function.
Status	**Acknowledged**. Description of the project party: We re-woked router not to have is_locked variable instead each pool now locks separately and can be locked gracefully - swaps and mints are not allowed, burns however are still allowed.

[TONCO-03] The pool can be initialized multiple times

Severity Level	Info
Type	Business Security
Lines	poolv3.func#L50-132
Description	There is a potential multiple initialization problem when creating pools, as it allows repeated modification of multiple key states of the pool (e.g., administrator, controller, price, NFT content, etc.). These parameters can be updated with each call, and there is no mechanism to prevent multiple initializations, which may lead to problems such as inconsistent pool states, conflicting price settings, and confusing permissions management, thus affecting the proper operation and security of the pool.
Recommendation	It is recommended that pools be initialized only once.
Status	**Acknowledged**.

[TONCO-04] Poor calibration

Severity Level	Info
Type	Business Security
Lines	routerv3.func
Description	In the swap and add liquidity functions of router contracts, if a user inputs wrong tokens (e.g., the user transfers A tokens but incorrectly specifies A tokens as the exchange target), due to the lack of a strict checking and error bouncing mechanism, the funds transferred by the user can't be refunded, which results in the loss of the user's funds.
Recommendation	It is recommended to verify the reasonableness of the generated pool address when calculating it. The administrator should save the generated pool address when creating the pool and compare it with the saved address each time it is calculated, if it does not make sense, immediately roll back the transaction and return the funds to the wallet.
Status	**Acknowledged**.

[TONCO-05] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	get.func#L48,60
Description	As shown in the code below, the hash obtained in the function is not used and is redundant code.

```
(cell, cell, cell) getChildContracts() method_id {
  load_storage();
  int hash = cell_hash(router::accountv3_code);
  return (
        router::poolv3_code,
        router::accountv3_code,
        router::position_nftv3_code
  );
}
(cell) getPoolInitialData(slice jetton_wallet0, slice jetton_wallet1)
method_id {
  load_storage();
  cell data = pack_pool_data(jetton_wallet0, jetton_wallet1,
router::accountv3_code, router::position_nftv3_code);
  int hash = cell_hash(data);
  return data;
}
```

Recommendation	It is recommended that redundant code be removed.
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.

Likelihood \ Impact	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

- **Critical**

Critical 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	Redundant Code
		Deprecated Items
		Gas Consumption*
		Event Trigger
		Throw Usage
2	General Vulnerability	Message Forgery*
		Restore on Failure*
		Integer Overflow/Underflow
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)*
		Function Call Permissions
		Message Flow Error*
		Returned Value Security
		Data Structure Error*
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations

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
Web3 Security & Compliance

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

X
https://x.com/Beosin_com

Email
service@beosin.com

LinkedIn
https://www.linkedin.com/company/beosin/