BEOSIN
Web3 Security & Compliance

RateX

Smart Contract Security Audit

No. 202504231135

Apr 23rd, 2025

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 High-risk, 3 Medium-risks, 4 Low-risks and 2 Info items were identified in the RateX project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

- **Project Description:**

Business overview

RateX is a leveraged yield trading protocol deployed on BNB Chain. It allows users to take long or short positions on various YT (Yield Token, which entitles the holder to the yield generated by the underlying yield-bearing asset (YBA) from now until a specified maturity date), and also supports an Earn mode for realizing fixed returns. The following section provides a detailed overview of the protocol's mechanisms.

Clearinghouse serves as the unified interface for leveraged traders (who take long or short positions on YT Tokens), Earn users (who hold PT Tokens (Principal Token, which can be redeemed for an equivalent amount of principal after maturity; users hold a certain amount of PT assets and YT liabilities at purchase. As YT value decays over time and becomes zero at maturity, PT can be redeemed 1:1 for principal)), and LPs who provide liquidity for these two types of trading. The following section introduces each user role in more detail.

LP Users: To provide market liquidity, users must deposit margin (i.e., slisBnb, with an amount exceeding the minimum threshold) into the protocol. The deposited margin will be re-staked via the helioTokenProvider to generate yield. Based on the YT interest rate range specified by the LP, the system calculates the tick prices to be added (this calculation considers the maturity period and interest rate values to determine the precise tick). The deposited margin is minted into ST tokens (Standard Token, which is a 1:1 quoted asset pegged to the underlying value), and YT assets are borrowed and added to the pool. The LP user's liquidityPositions (remaining position) consist of assets (ST minted from margin minus the portion used for LP addition) and liabilities (borrowed YT used to provide liquidity). LPs earn trading fees and may profit by acting as counterparties to leveraged traders. However, due to holding liquidityPositions, LPs face liquidation if their liabilities go into bad debt.

Leveraged Users: After LP users provide liquidity, users who wish to long or short YT assets can initiate leveraged trades via the trade function. The platform currently supports both cross-margin and isolated-margin modes. Users can open positions with up to 10x leverage, and the maximum margin maintenance ratio allows up to 20x. When trading, the system determines whether the action is an opening or closing position based on the user's trade direction and existing positions (users can set slippage tolerance). The user's position will then be updated based on the actual traded volume in the market (for closing positions, PnL is settled: profits are added to margin, while losses reduce the

position). After each leveraged trade, the system checks whether the margin meets the required maintenance criteria. If the margin is insufficient, the trade will fail.

Earn Users: Unlike leveraged trading, earn users follow a bond-like return model. When initiating an earn position, users specify the amount of `PT` tokens they wish to purchase. The protocol will then sell an equivalent amount of `YT` tokens in the pool to obtain `ST` tokens (representing early realization of future yield, with a 5% protocol fee charged on this process). Based on the actual trade volume, the system back-calculates the required margin amount (derived from the number of `PT` tokens purchased). After this process, the user holds `PT` tokens, which can be redeemed at market maturity to receive an equal amount of `ST` tokens on a 1:1 basis, and these are eventually converted back into equivalent margin and returned to the user. If the user redeems PT before maturity, since `YT` still has value prior to expiry, only partial yield can be obtained at that time—resulting in the user receiving the principal plus a portion of the yield.

In addition to providing core functionalities to users, the RateX protocol also implements several risk management mechanisms, including liquidation, epoch updates, and socialized loss distribution. These mechanisms will be introduced below. The liquidation process is initiated by addresses with keeper privileges and targets three types of entities: leveraged trading users, liquidity providers (LPs), and the insurance fund.

Liquidate: For leveraged trading users, the contract first checks whether the ratio of their collateral plus assets to liabilities meets the minimum maintenance margin requirement. If the ratio falls below the threshold, liquidation will be triggered.

For users operating under cross-margin mode, the system iterates through all markets the user has positions in, prioritizing those with greater insurance fund shortfalls. The user's margin will be allocated accordingly to cover these deficits. The user's positions across markets will then be liquidated, and if the liquidation occurs in a timely manner, both the protocol and the insurance fund can obtain a share of the remaining margin as profit. After liquidation, the user's positions are transferred to the insurance fund, and the process is completed.

For isolated margin users, liquidation is evaluated and executed within the individual market they are participating in only.

LiquidateLp: When LP users add liquidity to the pool, they use margin to mint ST tokens and borrow `YT` tokens. Therefore, the remaining liquidity position after adding liquidity is referred to as

`liquidityPositions`, which consists of two parts: the `ST` token asset (the remaining amount after consumption during addition) and the `YT` token liability (the portion borrowed when adding liquidity).

In the liquidation process, if the value of the liability exceeds the value of the asset (without considering the liquidation coefficient), liquidation will be triggered.

When the user removes liquidity from the pool (retrieving the two added tokens), if there is still a liability in the remaining position, it will be transferred to the insurance fund. If, after removal, both tokens are positive (i.e., both assets are retrieved), no further action will be taken.

LiquidateInsurance: When the insurance assets receive the liquidated positions, if the current market price is worse than the cumulative liquidation price of the above positions, the insurance will trigger the `ADL` (Automatic Deleveraging) mechanism. This will be called by the keeper, who will select users with the highest `ADL` profitability ranking to reduce the insurance position. The selected users will settle their positions at the cumulative liquidation price instead of the current more profitable price. This helps mitigate the deterioration of the insurance assets.

If, after the `ADL` mechanism is applied, the insurance still has liability assets, these will be transferred to the social loss position and will be shared when LP users remove liquidity in the future.

Social Loss Sharing Mechanism:

The social loss sharing mechanism applies to unsettled LP users and the remaining assets in their liquidity positions. This mechanism ensures that losses incurred during liquidation are fairly distributed among all LP users, maintaining system stability and preventing any single party from bearing excessive risk.

When insurance positions still carry liabilities or losses, unsettled LP users will share these losses according to the proportion of their liquidity positions. When LP users remove liquidity, the system adjusts their account balance based on their loss-sharing ratio, ensuring that risks are fairly distributed among LP users and reducing exposure to market volatility or liquidation risks.

This mechanism protects the funds provided by LP users in the liquidity pool, with each user sharing the risk in proportion to their level of participation.

removeLpShares: When an LP user removes liquidity, their share of social loss distribution will be determined based on the ratio of their net assets to the total global net assets. Additionally, the system will assess whether the global net assets can fully cover the social losses to calculate the `executingRatio` (which has a maximum value of 1, meaning full coverage is possible). The system will

then calculate two specific execution amounts (the total accumulated value is considered as liabilities). When the user removes liquidity, they need to settle the base portion (whether it's a profit or a loss). Finally, any profit or loss will be reflected in the withdrawal amount and the position.

Epoch Update Mechanism:

Since the project involves trading `ST` and `YT` assets, and to reflect the characteristic of `YT` asset price decay as the market approaches its expiration, the project will periodically call epoch updates to reflect this time decay (the update is non-linear, not breakpoint-based). At the start of each update, market trading will be paused, and the interest rate of the updated price will be recorded. Then, liquidity added by users to the pool will be removed to recalculate the price based on the current update time and interest rate (which will be a decayed price).

Next, positions will be settled based on bad debts (after settlement, positions will be deleted), and any profits or losses will be reflected in the margin. Subsequently, liquidity will be added back to the pool based on the new price. This concludes the update cycle. After each epoch, leveraged trading users' positions will be settled, LP users' positions will be updated, and earn users will experience a reduction in their YT debt value due to the updated price.

Leveraged users can increase their margin before liquidation by using the deposit function. They can also withdraw funds through the withdraw function, provided they meet the necessary maintenance ratio. LP users can claim fees generated by leveraged users' trades using the `claimFee` function.

Project Fees:The project can extract various fees from each market, including the `KeeperFee`, `ProtocolFee`, `EarnFee`, and the insurance profits from markets that have expired.

1 Overview

1.1 Project Overview

Project Name	RateX
Project Language	Solidity
Platform	BNB Chain
Code Base	https://github.com/GombleLab/RateX_contract
Commit Hash	8f0a8845c34182afa97edf87d5e9351e767667a2 2d70ec1d96d2f57c1ebb0ca6953dabf561067948 2f4f857bc997ed066015e12c29ba66b3f72b7cd6

1.2 Audit Overview

Audit work duration: Mar 20, 2025 – Apr 23, 2025

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 function 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
RateX-01	Calculation Error in _handleSocialLoss Function	High	Fixed
RateX-02	Reward Miscalculation in epochUpdateRemove	Medium	Fixed
RateX-03	Incorrect isClose Logic in trade Function	Medium	Fixed
RateX-04	lpSlossQuoteQuota and totalSlossQuoteQuota are not validated for values less than 0	Medium	Fixed
RateX-05	Missing initializer Modifier in initializePool Function	Low	Fixed
RateX-06	Incorrect Expiry Check Logic in _checkExpiry Function	Low	Fixed
RateX-07	Margin Call Still Incurs Handling Fees	Low	Acknowledged
RateX-08	The addLpShares function does not check the health of liquidityPositions	Low	Acknowledged
RateX-09	Redundant codes	Info	Fixed
RateX-10	Missing Early Return in _updateEarnProtocolFee	Info	Fixed

Finding Details:

[RateX-01] Calculation Error in _handleSocialLoss Function

Severity Level	High
Lines	market\UpdateImp.sol#642-700
Type	Business Security
Description	There is a calculation error in the `_handleSocialLoss` function of the UpdateImp.sol contract. When `slossQuoteAmountLp` and `slossBaseAmountLp` are negative (indicating a loss), the code uses the `position.reserveQuoteAmount -= slossQuoteAmountLp` operation. Since two negatives equal a positive, this will cause the liquidity provider's reserve to increase, which is incorrectly manifested as a profit.
Recommendation	It is recommended to add `slossQuoteAmountLp` to `position.reserveQuoteAmount` and apply the same adjustment to `position.reserveBaseAmount`.
Status	**Fixed.** The formula has been modified by the project team.

```
        if (slossTotalQuote >= 0) {
            slossQuoteAmountLp = SafeMath.mulDiv(slossTotalQuote *
lpSlossQuoteQuota, executingRatio, totalSlossQuoteQuota * 1e18);
        } else {
            slossQuoteAmountLp = SafeMath.mulDivCeil(slossTotalQuote *
lpSlossQuoteQuota, executingRatio, totalSlossQuoteQuota * 1e18);
        }
        if (slossTotalBase >= 0) {
            slossBaseAmountLp = SafeMath.mulDiv(slossTotalBase *
lpSlossQuoteQuota, executingRatio, totalSlossQuoteQuota * 1e18);
        } else {
            slossBaseAmountLp = SafeMath.mulDivCeil(slossTotalBase *
lpSlossQuoteQuota, executingRatio, totalSlossQuoteQuota * 1e18);
        }
        // Update LP reserves
        position.reserveQuoteAmount -= slossQuoteAmountLp;
        position.reserveBaseAmount -= slossBaseAmountLp;
```

[RateX-02] Reward Miscalculation in epochUpdateRemove

Severity Level	**Medium**
Lines	market\UpdateImp.sol#356-359
Type	Business Security

Description	The `epochUpdateRemove` function in the `UpdateImp` contract contains a design flaw in its execution order. It first invokes `_handleKeeperFee` to deduct the keeper fee from the user's position, and then executes `_rebaseAndDistributePosition` to calculate and distribute rewards. As a result, the reward calculation is based on the reduced principal after fee deduction, which unintentionally diminishes the user's actual returns and poses a fairness concern.
	`// Handle keeper fee`
	`_handleKeeperFee(position);`
	`// Rebase and distribute`
	`_rebaseAndDistributePosition(position, vars.ammBaseAmountOld, vars.ammQuoteAmountOld);`

Recommendation	It is recommended to do `_rebaseAndDistributePosition` first followed by `_handleKeeperFee` operation.

Status	**Fixed.** The project team has modified the call order of the relevant functions.
	`// Rebase and distribute`
	`_rebaseAndDistributePosition(position, vars.ammBaseAmountOld, vars.ammQuoteAmountOld);`
	`// Handle keeper fee`
	`_handleKeeperFee(position);`

[RateX-03] Incorrect isClose Logic in trade Function

Severity Level	Medium
Lines	market\MarketImp.sol#321
Type	Business Security
Description	There is a problem with the logic of `isClose` in the `trade` function of the MarketImp.sol contract. To close a position, the original position and the new position should be in opposite directions, i.e., one positive and one negative, and the result of multiplication should be less than zero. However, the current implementation does not recognize this feature correctly, which may lead to incorrect determination of the `isClose` state, thus affecting the execution of the subsequent logic.
Recommendation	It is recommended to refactor the `isClose` logic to explicitly check for opposite directions by verifying whether the product of the existing position size and the trade size is less than zero.
Status	**Fixed.** The project team has fixed the logic by correctly identifying `isClose` when the existing position and the trade position are in opposite directions.

```
    function trade(
        bytes32 subAccountId,
        int256 baseAmount,
        uint160 sqrtPriceLimitX96,
        bool isIsolated
    ) external onlyClearingHouse returns (TradeResult memory result)
{
        if (baseAmount == 0) revert InvalidAmount();
        if (!isMarketActive()) revert MarketNotActive();
        // Get user's position
        YieldPositionLib.YieldPosition storage position =
yieldPosition[subAccountId];
        // Calculate standardized amount
        bool isClose = baseAmount * position.baseAssetAmount >= 0 &&
baseAmount.abs() == position.baseAssetAmount.abs();
```

```
        bool isClose = (baseAmount > 0 && baseAssetAmount < 0) ||
(baseAmount < 0 && baseAssetAmount > 0);
```

[RateX-04] lpSlossQuoteQuota and totalSlossQuoteQuota are not validated for values less than 0

Severity Level	Medium
Lines	market\LiquidityImp.sol#264-266
Type	Business Security
Description	The `lpSlossQuoteQuota` and `totalSlossQuoteQuota` parameters are not validated to ensure they are not less than 0. This could allow users in a liquidation state to participate in loss sharing, potentially resulting in profits during settlement.
Recommendation	It is recommended to add validation for `lpSlossQuoteQuota` and `totalSlossQuoteQuota` to ensure they are not less than 0, guaranteeing loss sharing occurs only under liquidation states with available assets.
Status	**Fixed.**

```
            // Calculate sloss quota for this LP position
            int256 lpSlossQuoteQuota =
SafeMath.mulDiv(_getLpSlossQuota(position), int256(percentage),
1e18);
            int256 totalSlossQuoteQuota = _getTotalSlossQuota();
```

```
            if (lpSlossQuoteQuota > 0 || totalSlossQuoteQuota > 0) {
                revert InvalidSlossQuota();
            }
```

[RateX-05] Missing initializer Modifier in initializePool Function

Severity Level	Low
Lines	market\MarketImp.sol#257-302
Type	Business Security
Description	In the MarketImp.sol contract, the `initializePool` function does not use the `initializer` modifier. As a result, the function can be invoked multiple times, potentially leading to unintended re-initialization of the market. Repeated initialization may overwrite critical configuration variables or reset important states, posing a risk to contract integrity and user funds.
Recommendation	It is recommended to protect the `initializePool` function by applying the initializer modifier provided by OpenZeppelin's upgradeable contract library. This ensures the function can only be called once during the contract's lifecycle. If a custom initialization pattern is used, implement appropriate state checks (e.g., require(initialized == false)) to enforce one-time execution.
Status	**Fixed.** The project team has restricted the `initializePool` function to be called only once.

```
        function initializePool(InitializeParams memory params)
external onlyOwner {

       if (initialized) revert AlreadyInitialized();
```

[RateX-06] Incorrect Expiry Check Logic in _checkExpiry Function

Severity Level	Low
Lines	market\UpdateImp.sol#157-160
Type	Business Security
Description	There is a logical error in the `_checkExpiry` function: when checking whether the market is not expired, the condition incorrectly uses "expiry time less than the current time". In fact, if the market has not expired, the correct condition should be "expiry time greater than the current time". `function _checkExpiry() internal { if (stats.status != MarketStatus.Expired || expireTs < IOracle(oracle).epochStartTs()) { revert YieldMarketNotExpired(); } }`
Recommendation	It is recommended to modify the condition to `expireTs > IOracle(oracle).epochStartTs()` to accurately determine if the market is not expired.
Status	**Fixed.** The project team has been changed to `expireTs > IOracle(oracle).epochStartTs()`. `function _checkExpiry() internal view { if (stats.status != MarketStatus.Expired && expireTs > IOracle(oracle).epochStartTs()) { revert YieldMarketNotExpired(); } }`

[RateX-07] Margin Call Still Incurs Handling Fees

Severity Level	Low
Lines	market\MarketImp.sol#435-441
Type	Business Security
Description	When liquidation is delayed and results in a margin call liquidation, the project still charges handling fees proportionally. This undermines the health of the insurance pool and heightens the likelihood of ADL (Auto-Deleveraging).
Recommendation	It is recommended that the project waive handling fees during margin call liquidations.
Status	**Acknowledged.**

```
        int256 liquidationFee = (baseAssetValue.abs() * liqFeeRate) /
1e18;
        int256 tradePrice = (position.quoteAssetAmount +
_marginToQuote(marginTransferred, false) - liquidationFee) * 1e18 /
(-position.baseAssetAmount);

        // Record amounts being transferred
        result.baseAmountTransferred = position.baseAssetAmount;
        result.quoteAmountTransferred = position.quoteAssetAmount;
        result.liquidationFee = liquidationFee;
```

[RateX-08] The addLpShares function does not check the health of liquidityPositions

Severity Level	Low
Lines	market\LiquidityImp.sol#153-154
Type	Business Security

Description	After users add liquidity using the `addLpShares` function, no health check is performed on `liquidityPositions`. This may result in a risk of liquidation following the addition.
	`// Update position state in memory first`
	`position.reserveQuoteAmount += mintedQuoteAmount - int256(amount1);`
	`position.reserveBaseAmount -= int256(amount0);`

Recommendation	It is recommended to add a health check for `liquidityPositions` in the `addLpShares` function.
Status	**Acknowledged.**

[RateX-9] Redundant codes

Severity Level	Info
Lines	clearinghouse\ClearingHouseImp.sol#76-91
Type	Coding Conventions
Description	In the ClearingHouseImp contract, the following state variables are declared but never used: `stakeManager`, `helioTokenProvider`, and `delegateTo`. These variables are redundant and serve no functional purpose in the current implementation. Leaving unused variables in the codebase increases code complexity, can cause confusion during maintenance, and may suggest incomplete or abandoned features.
Recommendation	It is recommended to remove unused variables from the contract to improve code clarity and maintainability. If these variables are placeholders for future features, consider adding comments to indicate their intended use, or define them in a future upgrade to keep the current implementation clean.
Status	**Fixed.** The redundant codes has been removed by the project team.

[RateX-10] Missing Early Return in _updateEarnProtocolFee

Severity Level	Info
Lines	market\EarnImp.sol#210-214
Type	Coding Conventions
Description	In the `_updateEarnProtocolFee` function, the condition `earnPosition.yieldPosition.lastRate == 0` is checked, but the function continues execution even when this condition is met. Since a `lastRate` of zero indicates that there is no yield data to process, continuing beyond this point results in unnecessary computations.
Recommendation	It is recommended to add an early return statement immediately after the `earnPosition.yieldPosition.lastRate == 0` . This will prevent redundant logic from executing and help reduce gas costs.
Status	**Fixed.**

```
function _updateEarnProtocolFee() internal {
    uint256 rate = uint256(IOracle(oracle).getValue());
    if (earnPosition.yieldPosition.lastRate == 0) {
        earnPosition.yieldPosition.lastRate = int256(rate);
    }
    int256 lastRate = earnPosition.yieldPosition.lastRate;
    if (lastRate != int256(rate)) {
```

```
function _updateEarnProtocolFee() internal {
    uint256 rate = uint256(IOracle(oracle).getValue());
    if (earnPosition.yieldPosition.lastRate == 0) {
        earnPosition.yieldPosition.lastRate = int256(rate);
        return;
    }
    int256 lastRate = earnPosition.yieldPosition.lastRate;
```

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	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

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

- **Coding Conventions**

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

- **General Vulnerability**

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

- **Business Security**

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

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

3.3 Disclaimer

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

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

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

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

3.4 About Beosin

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

BEOSIN
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/