BEOSIN
Blockchain Security

zeepr

Smart Contract Security Audit

No. 202309260959

Sep 26th, 2023

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

1 Overview 5

1.1 Project Overview 5

1.2 Audit Overview 5

1.3 Audit Method 5

2 Findings 7

[zeepr-01] Centralization Risk 8

[zeepr-02] Divisor Potentially Yielding Zero 9

[zeepr-03] Unchecked Parameter Ranges 10

3 Appendix 11

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts 11

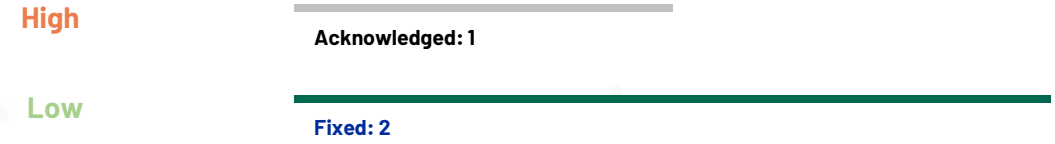
3.2 Audit Categories 14

3.3 Disclaimer 16

3.4 About Beosin 17

Summary of Audit Results

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



- **Risk Description:**

1. The current default leverage in the contract is set at 100x, and the project team may adjust the leverage multiplier based on the actual circumstances after the project goes live.

- **Project Description:**

Business overview

The audit project consists of a collateral-based perpetual futures contract, divided into two main components: the Perpetual contract section and the Private Pool section. The Private Pool contract manages the counterparties' pool for the perpetual contract, while the perpetual contract handles market and limit orders.

The Perpetual contract comprises three main sections: collateral deposits, futures trading, and liquidation. In the collateral deposits module, users (takers) can collateralize specified tokens using the `deposit` function within the perpetual contract. These funds serve as margin for future trading. Profits can be withdrawn using the `withdraw` function. Futures trading includes market orders and limit orders. Market orders allow users to go long or short directly using the current market token price. Limit orders permit users to specify a price limit, and when the current price matches the limit, the contract automatically creates a market order for purchase. The contract also supports take profit and stop loss functionality, allowing users to set take profit and stop loss limits on active orders. When these price limits are met, the orders are automatically executed and closed. The futures trading also includes leverage functionality, with the current contract's default set to a leverage of 100x, which users cannot modify. Liquidation is called by the Liquidator and can be used to liquidate both takers (perpetual contract users) and makers (private pool users). Liquidation implementation is managed by the Private Pool contract.

The Private Pool contract consists of collateral deposits, order matching, and core liquidation functions. In the collateral deposits module, users (makers) can collateralize funds in the Private Pool contract as part of market-making. There are two collateralization modes: private pool collateralization and public pool collateralization. In the private pool collateralization mode, users collateralize funds into their personal accounts, and when matching counterparties are needed, random independent users are selected as counterparties. In this mode, users are individually responsible for profits and losses. In the public pool collateralization mode, users collateralize funds into an administrator's account, and the administrator matches counterparties. Profits and losses from matches are calculated and distributed as shares. The order matching function is called when opening positions in the perpetual contract. When the private pool is active, a random private pool is selected for counterparties. If there are no suitable private pools, the public pool is used for order matching. In the core liquidation function, the profit and loss distribution for liquidations that occur in the perpetual contract is ultimately handled in the "close" function within the Private Pool contract.

1 Overview

1.1 Project Overview

Project Name	zeepr
Project language	Solidity
Platform	BNB Chain
Github Link	https://github.com/Zeeprlabs/zeepr-contract/tree/main
Commit	d07876762ad6fa4f271838e00f4f0ee347b3b674 8a038dea4c128d36f5a2ddb842b6f5159ead006c 8a9abc6c1e9f3e5d0bb780e77a43001449ec87b5 2d6965154e33cb620f7f574e211ea4158eee8373

1.2 Audit Overview

Audit work duration: Sep 14, 2023 – Sep 26, 2023

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
zeepr-01	Centralization Risk	High	Acknowledged
zeepr-02	Divisor Potentially Yielding Zero	Low	Fixed
zeepr-03	Unchecked Parameter Ranges	Low	Fixed

Finding Details:

[zeepr-01] Centralization Risk

Severity Level	High
Type	Business Security
Lines	Underlying.sol #L200-202

Description	Several high-privilege functions of the contract can be called and modified parameters by the Owner and Keeper. For instance, when using the `getPrice` function to fetch prices, if the Owner activates the `priceThirdFlag` mode, the obtained price will be directly set to the default parameter price.
	
	`function getPrice(bool melt) public view returns (uint256 latestPrice) {`
	`if (priceThirdFlag) {`
	`latestPrice = thirdProxyPrice;`
	`}`

Recommendation	It is recommended that after the contract is deployed, a multi-signature wallet be employed to manage the Owner and Keeper addresses.
Status	**Acknowledged.**

[zeepr-02] Divisor Potentially Yielding Zero

Severity Level	Low
Type	Coding Conventions
Lines	PrivatePool.sol #L163, L756-758

Description	In the `getShareNetValue` function of the PrivatePool contract, the relationship between `totalValue` and loss is not properly handled. If the situation arises where `totalValue` equals loss, it can cause the `getShareNetValue` function to return a value of 0. This, in turn, can lead to the division by zero error when the `convertAmountToShare` function in the upper layer attempts to execute, resulting in a transaction failure.
	
	getShareNetValue:
	
	`if (totalValue >= loss) { return (totalValue - loss) * PRICE_DECIMALS / accountShare[shareRound][escrowAccount]; } else { require(false, "net value was negative."); }`
	
	convertAmountToShare:
	
	`function convertAmountToShare(uint256 _amount) internal view returns (uint256){ return _amount * PRICE_DECIMALS / getShareNetValue(); }`

Recommendation	It is recommended to handle cases where `totalValue` equals loss differently.
Status	**Fixed.** The project team has implemented special handling for cases where `totalValue` equals loss. `if (totalValue > loss) { return (totalValue - loss) * PRICE_DECIMALS / accountShare[shareRound][escrowAccount]; } else { require(false, "escrow account should be detonate"); }`

[zeepr-03] Unchecked Parameter Ranges

Severity Level	Low
Type	Business Security
Lines	PrivatePool.sol #L83-96
Description	In the PrivatePool contract's `provide` function, there is no check for the user's input `_priOrPub` value. If the user enters an inappropriate `_priOrPub` value, it can result in a loss of funds.

```
        _safeTransferFrom(tokenAddress, msg.sender, address(this),
_amount);
        createAccountIfAbsent();
        LP2Account memory lp2Account = lpAccount[msg.sender];
        if (_priOrPub == 1) {
            uint256 share = convertAmountToShare(_amount);
            lpAccount[escrowAccount].amount += _amount;
            lpAccount[escrowAccount].availableAmount += _amount;
            accountShare[shareRound][escrowAccount] += share;
            accountShare[shareRound][lp2Account.holder] += share;
        } else if (_priOrPub == 2) {
            require(everyOneAsMaker || makerMap[msg.sender], "caller is
unAllowed maker");
            lp2Account.amount += _amount;
            lp2Account.availableAmount += _amount;
        }
```

Recommendation	It is recommended to add an else statement to validate the user's input for `_priOrPub`.
Status	**Fixed.** The project team has added handling for `_priOrPub` values other than 1 and 2.

```
        } else {
            require(false, "wrong deposit flag");
        }
```

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

- **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.

BEOSIN
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com