BEOSIN
Web3 Security & Compliance

Jasper Vault

Smart Contract Security Audit

No. 202503251614

Mar 25th, 2025



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

[Jasper Vault-01] Lack of permission checking for key addresses 8

[Jasper Vault-02] Signature bypassing 10

[Jasper Vault-03] Token Repeat Additions 11

[Jasper Vault-04] Redundant codes 12

[Jasper Vault-05] Comment error 13

[Jasper Vault-06] Missing events 14

3 Appendix 15

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts 15

3.2 Audit Categories 18

3.3 Disclaimer 20

3.4 About Beosin 21

Summary of Audit Results

After auditing, 1 Critical, 1 Medium, 1 Low-risk and 3 Info items were identified in the Jasper Vault 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

Jasper Vault is a peer-to-peer Bitcoin options trading platform designed to provide a safe, simple, and efficient way for users to trade with leverage without the worry of liquidation. The platform offers flexible options, including 30-minute trades via its Telegram Mini App, 2-hour expirations for degen options, and user-adjustable 0DTE (zero days to expiration) in advanced mode. Operating on multiple chains, Jasper Vault enhances the trading experience with NFTs and GameFi elements like the Moonlight Box and Infinity Stones, which boost user engagement and flexibility. By enabling direct trading from self-managed wallets, Jasper Vault ensures security, transparency, and a seamless experience for both beginners and advanced traders.

This audit focuses specifically on the OptionModuleV4, and LPPoolMoudle contracts within the Jasper Vault project.

- OptionModuleV4:

An upgradable options management contract that supports the creation and management of call/put option orders, order signature verification, fee settlement, asset transfers, and other related features. Users can pay premiums (premiumAsset) using ETH or ERC20 tokens and access flexible options configuration and querying capabilities. The contract includes management for price oracle whitelists and fee discount whitelists, allowing users holding specified NFTs to receive fee discounts. Option fees are distributed proportionally at settlement, with a portion paid to the option writer and the remaining portion allocated as platform fees.

- LPPoolMoudle:

Validated vault contracts can be used in LPPoolMoudle to buy LP tokens with backing tokens or ETH via the `buy` function, or to sell LP tokens for backing tokens or ETH via the `sell` function (Two functions calculate the current net worth first, then the lp quantity, and finally make the transfer). Fees are set by the owner. The supported tokens are configured in the vault contract.

1 Overview

1.1 Project Overview

Project Name	Jasper Vault
Project Language	Solidity
Platform	Arbitrum,BASE,Bitlayer,Sei Chain
Base code	https://github.com/jaspervault/jasper-vaults-contracts
Audit Scope	OptionModuleV4.sol LPPoolMoudle.sol
Commit	dc71bd5fb09292e039a64b17fd7785d35d2e292e ec95fb94cfed8bea88aaab4803e7c8d1f2a39949 b49d33d4e72069ecd043167dc2a600640710187b bfbf040dd68a250ee4c77ec6ea3595dc97a60230

1.2 Audit Overview

Audit work duration: Mar 14, 2025 – Mar 25, 2025

Update date: Apr 9, 2025

Update details: Fixed incorrect spelling

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
Jasper Vault-01	Lack of permission checking for key addresses	**Critical**	**Fixed**
Jasper Vault-02	Signature bypassing	**Medium**	**Fixed**
Jasper Vault-03	Token Repeat Additions	**Low**	**Fixed**
Jasper Vault-04	Redundant codes	**Info**	**Acknowledged**
Jasper Vault-05	Comment error	**Info**	**Fixed**
Jasper Vault-06	Missing events	**Info**	**Fixed**

Finding Details:

[Jasper Vault-01] Lack of permission checking for key addresses

Severity Level	**Critical**
Lines	LPPoolMoudle .sol #L170-303
Type	General Vulnerability
Description	The `buy` function of LPPoolMoudle does not check the legitimacy of the from address, so if a malicious contract address is passed in, the transfer can be avoided by constructing the wrong `invokeTransferEth`, thus buying Lp. at 0 cost.
Recommendation	It is recommended to check the _form address.
Status	**Fixed.** Added check by onlySameOwnerVault modifier for _from addresses.

```
    function buy(
        address _vault,
        address payable _from,
        address _asset,
        uint256 _assetAmount
    ) payable external nonReentrant {
        // 1. Parameter validation
        validAsset(_vault, _asset);
        require(_assetAmount > 0, "LPPoolMoudle:amount must be greater
than 0");

        IVaultFacet vaultFacet = IVaultFacet(diamond);
        require(
            !vaultFacet.getVaultLock(_vault),
            "LPPoolMoudle:vault is locked"
        );
        IPlatformFacet platformFacet = IPlatformFacet(diamond);
        uint vaultIndex = platformFacet.getVaultToSalt(_vault);
        require(
            vaultIndex == 1,
            "LPPoolMoudle:only supports vault1"
        );
```

```
    function buy(
        address _vault,
```

```
        address payable _from,
        address _asset,
        uint256 _assetAmount
    ) payable external nonReentrant onlySameOwnerVault(_from) {
```

[Jasper Vault-02] Signature bypassing

Severity Level	**Medium**
Lines	OptionModuleV2Handle.sol #L84-86
Type	Business Security
Description	In the `handlePremiumSign` function of the OptionModuleV2Handle contract, the signature checksum will be bypassed if the length of the _sign.oracleSign.length entered by the user is zero. `for(uint256 i; i<_sign.oracleSign.length; i++) { require(optionModuleV2.getOracleWhiteList(ECDSA.recover(digest, _sign.oracleSign[i])), "OptionModule:handlePremiumSign not from whiteList error"); }`
Recommendation	It is recommended to check the value of _sign.oracleSign entered by the user.
Status	**Fixed.** Added judgment that requires the length of the signature array to be greater than 0. `require(_sign.oracleSign.length>0,"OptionModule:handlePremiumSign oracleSign length error"); for(uint256 i; i<_sign.oracleSign.length; i++) { require(optionModuleV2.getOracleWhiteList(ECDSA.recover(digest, _sign.oracleSign[i])), "OptionModule:handlePremiumSign not from whiteList error"); }`

[Jasper Vault-03] Token Repeat Additions

Severity Level	Low
Lines	OptionFacetV3 .sol #L61-76
Type	Business Security
Description	The tokens with flag true are added repeatedly in the `addUnderlyingAssets` function in the OptionFacetV3 contract.

```
    function addUnderlyingAssets(
        address _writer,
        address _underlyingAsset
    ) public {
        OptionV3 storage ds = diamondStorage();
        address[] memory assetList =
ds.optionUnderlyingAssets[_writer];
        bool flag;
        for (uint256 i = 0; i < assetList.length; i++) {
            if (assetList[i] == _underlyingAsset) {
                flag = true;
            }
        }
        if (flag == true) {
            ds.optionUnderlyingAssets[_writer].push(_underlyingAsset)
;
        }
    }
```

Recommendation	It is recommended that the push be done only when the change is false.
Status	**Fixed.** Modified the judgment condition of flag to meet the actual running requirements.

```
        if (flag == false) {
            ds.optionUnderlyingAssets[_writer].push(_underlyingAsset)
;
        }
```

[Jasper Vault-04] Redundant codes

Severity Level	Info
Lines	LPPoolMoudle .sol #L86-90
Type	Coding Conventions
Description	`minWithdrawInterval` is not used elsewhere in the contract after it is set and is redundant code.
Recommendation	It is recommended to remove redundant code or add corresponding call logic.
Status	**Acknowledged.**

```
    function setMinWithdrawInterval(uint256 _interval) external
onlyOwner {
        require(_interval <= 30 days, "LPPoolMoudle: interval too
long"); // Max 30 days
        require(_interval >= 1 hours, "LPPoolMoudle: interval too
short"); // Min 1 hour
        minWithdrawInterval = _interval;
    }
```

[Jasper Vault-05] Comment error

Severity Level	Info
Lines	LPPoolMoudle .sol #L58-66
Type	Coding Conventions
Description	The comment for contract initialization `minWithdrawInterval` does not match the actual value.

```
    function initialize(address _diamond,address _priceOracle) public
initializer {
        __UUPSUpgradeable_init();
        diamond = _diamond;
        platformFee = 100; // 1%
        minWithdrawInterval = 4 days; // Default to 1 day
        priceOracle = _priceOracle;
    }
```

Recommendation	It is recommended to change the comment.
Status	**Fixed.** The incorrect comment has been corrected.

```
    function initialize(address _diamond,address _priceOracle) public
initializer {
        __UUPSUpgradeable_init();
        diamond = _diamond;

        platformFee = 100; // 1%
        minWithdrawInterval = 4 days; // Default to 4 day
        priceOracle = _priceOracle;
    }
```

[Jasper Vault-06] Missing events

Severity Level	Info
Lines	LPPoolMoudle .sol #L310-316
Type	Coding Conventions
Description	When modifying important parameters in a contract, it is recommended to trigger the corresponding event for transparent and traceable recording on the blockchain. `function setVaultLPTokenAddress(address _vault, address _lpToken) external onlySameOwnerVault(_vault) { require(_vault != address(0), "LPPoolMoudle: invalid vault address"); require(_lpToken != address(0), "LPPoolMoudle: invalid LP token address"); ILPPoolFacet lpPoolFacet = ILPPoolFacet(diamond); lpPoolFacet.setVaultLPTokenAddress(_vault, _lpToken); }`
Recommendation	It is recommended to add corresponding events and trigger them.
Status	**Fixed.** Corresponding events have been added and triggered. `function setVaultLPTokenAddress(address _vault, address _lpToken) external onlySameOwnerVault(_vault) { require(_vault != address(0), "LPPoolMoudle: invalid vault address"); require(_lpToken != address(0), "LPPoolMoudle: invalid LP token address"); ILPPoolFacet lpPoolFacet = ILPPoolFacet(diamond); lpPoolFacet.setVaultLPTokenAddress(_vault, _lpToken); emit VaultLPTokenAddressSet(_vault, _lpToken); }`

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/