owlto-finance

Smart Contract Security Audit

V1.0

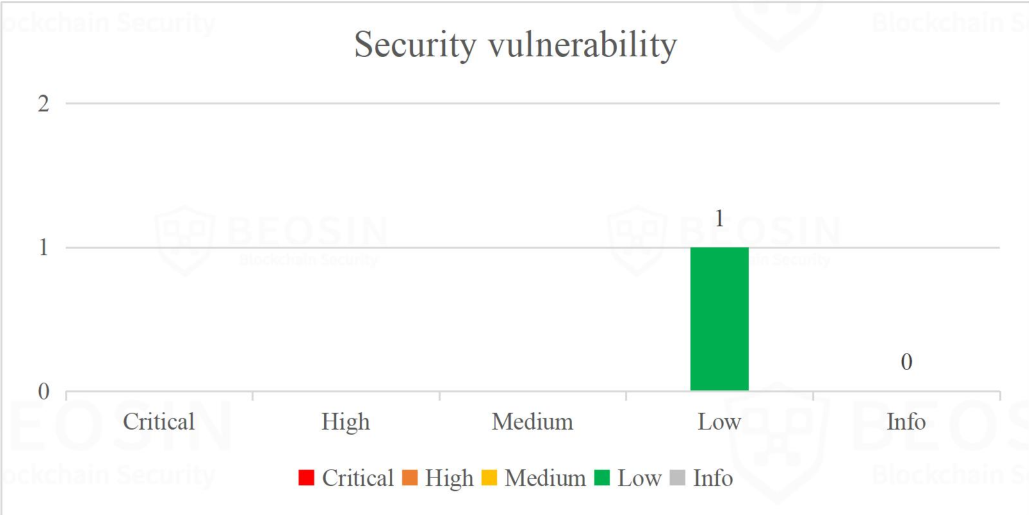
No. 202305051632

May 5th, 2023

Contents

Summary of Audit Results

After auditing, 1 Low-risk item were identified in the owlto-finance 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:**

1. **Business overview**

The LPManager contract in the owlto-finance project for this audit mainly implements the function of adding, updating and removing LP pools. The add and update functions can only be called by the owner of the contract, the maker of the LP pool is specified by the owner at the time of adding and cannot be changed. *removeLP* can only be called by the maker of the LP pool.

1 Overview

1.1 Project Overview

Project Name	owlto-finance
Platform	gcd aws
File Hash	EDD0FE459DB4429B79252567C6E684EFF1EE23877C9707F7B21547FCE1893D1A（Initial） E67C3BA705BDD3FD6FF6EBAEDEC4B91D60824A6105D1A66A46B1AA9C1E013CEA（Latest）

1.2 Audit Overview

Audit work duration: May 4, 2023 – May 5, 2023

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

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
owlto-finance-1	Array Override	Low	Fixed

Finding Details:

[owlto-finance-1] Array Override

Severity Level Low

Type Business Security

Lines LPManager.sol#L38-76

Description

When using the *addLP* function to update an LP, if the LP before that LP is deleted (represented as 0 in the array), the new updated data will be stored in the location before the deleted one. There may be more than one identical lpid in the array, and the query result will be based on the top sorted one, which may not be the latest.

```
38     function addLP(Lib_Type.Token memory baseToken,
39             Lib_Type.Token memory token_1,
40             Lib_Type.Token memory token_2,
41             address maker,
42             uint256 gasCompensation,
43             uint256 txFeeRatio,
44             uint256 startTimestamp,
45             uint256 stopTimestamp) external onlyOwner {
46         bytes32 lpId = Lib_Type.getLpId(token_1, token_2, maker);
47     lps[lpId] = Lib_Type.LP(
48             lpId,
49         baseToken,
50         token_1,
51         token_2,
52         maker,
53         gasCompensation,
54         txFeeRatio,
55         startTimestamp,
56         stopTimestamp
57     );
58
59     Lib_Type.LpKey memory lpKey = Lib_Type.LpKey(lpId, false);
60     uint256 index = 0;
61     for (; index < lpKeys.length; index++) {
62             if (lpKeys[index].isDeleted) {
63             lpKeys[index] = lpKey;
64         break;
65         }
66         if (lpKeys[index].lpId == lpId) {
67             lpKeys[index] = lpKey;
68         break;
69         }
70     }
71     if (index == lpKeys.length) {
72         lpKeys.push(lpKey);
73     }
74
75     emit AddPoolEvent(maker, lpId);
76     }
```

Figure 1 Source code of *addLP* function

Recommendations It is recommended that the update operation be implemented as a separate function, first finding the corresponding lpid and then updating the corresponding data.

Status Fixed.

```
function updateLP(Lib_Type.Token memory baseToken,
        Lib_Type.Token memory token_1,
        Lib_Type.Token memory token_2,
        address maker,
        uint256 gasCompensation,
        uint256 txFeeRatio,
        uint256 startTimestamp,
        uint256 stopTimestamp) external onlyOwner {
    bytes32 lpId = Lib_Type.getLpId(token_1, token_2, maker);
    (, bool ok) = findLpIndex(lpId);
    require(ok, "lp not found");

    lps[lpId] = Lib_Type.LP(
        lpId,
        baseToken,
        token_1,
        token_2,
        maker,
        gasCompensation,
        txFeeRatio,
        startTimestamp,
        stopTimestamp
    );

    emit UpdatePoolEvent(maker, lpId);
}
```

Figure 2 Source code of *updateLP* function

```
function addOrUpdateLP(Lib_Type.Token memory baseToken,
        Lib_Type.Token memory token_1,
    Lib_Type.Token memory token_2,
        address maker,
        uint256 gasCompensation,
        uint256 txFeeRatio,
        uint256 startTimestamp,
        uint256 stopTimestamp) external onlyOwner {
    bytes32 lpId = Lib_Type.getLpId(token_1, token_2, maker);
lps[lpId] = Lib_Type.LP(
        lpId,
    baseToken,
    token_1,
    token_2,
    maker,
    gasCompensation,
    txFeeRatio,
    startTimestamp,
    stopTimestamp
);

(, bool ok) = findLpIndex(lpId);
if (ok) {
    emit UpdatePoolEvent(maker, lpId);
} else {
        Lib_Type.LpKey memory lpKey = Lib_Type.LpKey(lpId, false);
    uint256 index = findFirstAvailableIndex();
    if (index == lpKeys.length) {
            lpKeys.push(lpKey);
    } else {
            lpKeys[index] = lpKey;
    }
    emit AddPoolEvent(maker, lpId);
}
}
```

Figure 3 Source code of *addOrUpdateLP* function

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.

Impact / Likelihood	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	Redundant Code
		require/assert Usage
		Cycles Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		Returned Value Security
		Rollback Risk
		Replay Attack
		Overriding Variables
		Call Canister controllable
		Canister upgrade risk
		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.

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com