BEOSIN
Blockchain Security

MOMO.FUN-Swap-v2

Smart Contract Security Audit

No. 202408141122

Aug 14th, 2024

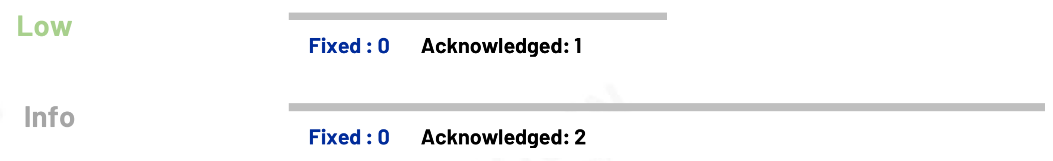
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 Low and 2 Info-risk item were identified in the MOMO.FUN-Swap-v2 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

The main functions of the v2 part of MOMO.FUN in the scope of this audit are similar to Uniswap v2. It MOMO.FUN operates on an automated market maker (AMM) model, where liquidity providers (LPs) contribute funds to liquidity pools, and trades are executed against these pools.allows users to freely create trading pairs, add and remove liquidity, and exchange related tokens in trading pairs. Key features include:

- Token Swapping:

MOMO.FUN enables users to swap between different ERC-20 tokens. The exchange rate between the two tokens is determined by the ratio of their quantities in the liquidity pool, following the constant product formula $x * y = k$.

- Liquidity Provision:

Users can become liquidity providers by depositing an equal value of two tokens into a liquidity pool. In return, they receive liquidity tokens, which represent their share of the pool and entitle them to a portion of the trading fees generated.

- Automated Market Making (AMM):

MOMO.FUN uses an AMM system, where trades are executed against a smart contract rather than directly between buyers and sellers. The AMM model ensures that there is always liquidity available for token swaps, but the price of tokens adjusts based on supply and demand within the pool.

- Fee Structure:

MOMO.FUN charges a 0.3% fee on all trades, which is distributed to liquidity providers in proportion to their share of the liquidity pool. This incentivizes users to provide liquidity and helps maintain the stability of the platform.

- Flash Swaps:

MOMO.FUN allows users to borrow tokens from a liquidity pool without providing collateral, as long as the borrowed tokens are returned by the end of the transaction. This feature enables advanced trading strategies, such as arbitrage and collateral swaps, without the need for upfront capital.

1 Overview

1.1 Project Overview

Project Name	MOMO.FUN-Swap-v2
Project Language	Solidity
Platform	EVM Chain
Code Base	https://github.com/ido-fe/contract-v3/tree/main/contracts/v2-periphery/contracts https://github.com/ido-fe/contract-v3/tree/main/contracts/v2-core/contracts
Commit Hash	bbb1906c76a0989f33c8d8d18981922fd1a9cf04

1.2 Audit Overview

Audit work duration: Aug 12, 2024 – Aug 14, 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
Swap-v2-01	Code optimization	Low	Acknowledged
Swap-v2-02	Redundant codes	Info	Acknowledged
Swap-v2-03	Missing trigger event	Info	Acknowledged

Finding Details:

[Swap-v2-01] Code optimization

Severity Level	Low
Type	Business Security
Lines	UniswapV2Pair.sol#L60-115
Description	When the UniswapV2Pair contract is initialized, the _initialize function is called once in both _initializeName and _initializeSymbol, and with exactly the same arguments in both calls. And as an internal function in the constructor, its return name_ and symbol_ are also not used and are meaningless returns. The main purpose of the constructor is to initialize the state of the contract, not to handle and use the return values.

```
    constructor(
        address _token0,
        address _token1
    )
        ERC20WithPermit(
            _initializeName(_token0, _token1),
            _initializeSymbol(_token0, _token1)
        )
    {}
    function _initializeName(
        address _token0,
        address _token1
    ) internal returns (string memory name_) {
        (string memory name_, string memory symbol_) = _initialize(
            _token0,
            _token1
        );
        return name_;
    }
    function _initializeSymbol(
        address _token0,
        address _token1
    ) internal returns (string memory symbol_) {
        (string memory name_, string memory symbol_) = _initialize(
            _token0,
            _token1
        );
        return symbol_;
    }
```

Recommendation	It is recommended that the _initializeName and _initializeSymbol functions be removed and modified to call the _initialize function only once in the constructor and remove the returned name_ and symbol_.
Status	**Acknowledged.**

[Swap-v2-02] Redundant codes

Severity Level	Info
Type	Coding Conventions
Lines	UniswapV2Factory.sol#L43-51 UniswapV2Pair.sol#L84-85
Description	The setFeeTo and setFeeToSetter functions in the UniswapV2Factory contract do not need to be set to override. And The initial values "Dex LP Token" and "LP" set when declaring temporary variables name_ and symbol_ in the UniswapV2Pair contract's _initialize function were not used subsequently. These are all redundant codes.

```
     function setFeeTo(address _feeTo) external override {
        require(msg.sender == feeToSetter, "UniswapV2: FORBIDDEN");
        feeTo = _feeTo;
    }
    function setFeeToSetter(address _feeToSetter) external override {
        require(msg.sender == feeToSetter, "UniswapV2: FORBIDDEN");
        feeToSetter = _feeToSetter;
    }
    function _initialize(
        address _token0,
        address _token1
    ) internal returns (string memory name_, string memory symbol_) {
        factory = msg.sender;
        token0 = _token0;
        token1 = _token1;
        (bool success0, string memory symbol0) =
MetadataHelper.getSymbol(
            _token0
        );
        (bool success1, string memory symbol1) =
MetadataHelper.getSymbol(
            _token1
        );
        string memory name_ = "Dex LP Token";
        string memory symbol_ = "LP";
        if (success0 && success1) {
            name_ = string(
```

	```
	abi.encodePacked("ZkDex ", symbol0, "/", symbol1, " LP Token")
	);
	symbol_ = string(abi.encodePacked(symbol0, "/", symbol1, " ZLP"));
	}
	return (name_, symbol_);
	}
	```
Recommendation	It is recommended to delete redundant codes.
Status	**Acknowledged.**

[Swap-v2-03] Missing trigger event

Severity Level	Info
Type	Coding Conventions
Lines	UniswapV2Factory.sol#L43-51

Description	The UniswapV2Factory contract did not trigger an event when the `feeTo` and `feeToSetter` addresses were modified.
	`function setFeeTo(address _feeTo) external override {`
	`    require(msg.sender == feeToSetter, "UniswapV2: FORBIDDEN");`
	`    feeTo = _feeTo;`
	`}`
	`function setFeeToSetter(address _feeToSetter) external override {`
	`    require(msg.sender == feeToSetter, "UniswapV2: FORBIDDEN");`
	`    feeToSetter = _feeToSetter;`
	`}`

Recommendation	It is recommended to emit events when modifying critical variables is a recommended practice as it provides a standardized way to capture and communicate important changes within the contract. Events enable transparency and allow external systems and users to easily track and react to these modifications.
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.

Impact / Likelihood	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.

3.1.3 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
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com