BEOSIN
Blockchain Security

MOMO.FUN-Swap-v3

Smart Contract Security Audit

No. 202408241154

Aug 24th, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 Info risk item was identified in the MOMO.FUN-Swap-v3 project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

Info	Fixed : 0 Acknowledged: 1

- **Project Description:**

1. Business overview

This project is a decentralized exchange (DEX) protocol inspired by Uniswap V3, allowing users to earn trading fee rewards by providing liquidity through IDO. The project is divided into Core and Periphery modules.

The Core module is responsible for implementing the specific functionalities of the Automated Market Maker (AMM). It includes the creation of liquidity pools, management of user liquidity within individual pools, and price calculations during token swaps. The UniswapV3Factory contract primarily manages the liquidity pools, where the `createPool` function can be used to create a liquidity pool for a specified tokenA, tokenB, and fee. The UniswapV3Pool contract handles the addition and removal of liquidity within a pool, calculates token amounts and prices during swaps, and manages user interactions. The `mint` function allows users to add specific liquidity within a chosen price range, while the `burn` function facilitates liquidity removal. Additionally, the `swap` function calculates the token amounts based on the current price ranges in the pool, the user's desired execution price (sqrtPriceLimitX96), and the current pool price. Liquidity providers in UniswapV3Pool do not earn fee rewards continuously but only when their liquidity is utilized in a swap.

The Periphery module provides tools and interfaces to simplify user interactions. Through this module, users can easily create pools, add or remove liquidity, set price ranges, customize fee tiers, and perform token swaps. The PoolInitializer contract is responsible for creating and initializing liquidity pools. Users can call the `createAndInitializePoolIfNecessary` function to create a pool and set an initial price, with the option to select different fee rates (e.g., 0.01%, 0.05%, 0.3%, 1%), each corresponding to different pools. The NonfungiblePositionManager contract manages liquidity positions, allowing users to add liquidity within a specified price range and receive an NFT representing the position through the `mint` and `increaseLiquidity` functions, or remove liquidity using the `decreaseLiquidity` and `burn` functions. The SwapRouter contract facilitates token swaps, where users can use functions like `exactInputSingle`, `exactOutputSingle`, and `exactInput` to perform swaps. The fees generated during a swap are not shared among all liquidity providers but are only distributed to those providing liquidity in the price range used for that specific swap.

1 Overview

1.1 Project Overview

Project Name	MOMO.FUN-Swap-v3
Project Language	Solidity
Platform	EVM Chain
Code Base	https://github.com/ido-fe/contract-v3/tree/main/contracts/v3-periphery https://github.com/ido-fe/contract-v3/blob/main/contracts/NoDelegateCall.sol https://github.com/ido-fe/contract-v3/blob/main/contracts/UniswapV3Factory.sol https://github.com/ido-fe/contract-v3/blob/main/contracts/UniswapV3Pool.sol https://github.com/ido-fe/contract-v3/blob/main/contracts/UniswapV3PoolDeployer.sol
Commit	a2f1828d8632755f8fa2a5aad22d8c800aaa77fd

1.2 Audit Overview

Audit work duration: Aug 23, 2024-Aug 24, 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-v3-01	Gas optimization suggestions	Info	**Acknowledged**

Finding Details:

[Swap-v3-01] Gas optimization suggestions

Severity Level	Info
Lines	interfaces\INonfungiblePositionManager.sol #L12 libraries\ChainId.sol #L8-12
Type	Coding Conventions

Description	The current contract has some code that can be further optimized for gas usage. This optimization can reduce the gas consumption during the deployment and execution of the contract, while also providing additional assurance of code rigor.
	1. Unused Import Contracts:
	In the INonfungiblePositionManager interface contract, PoolAddress.sol is only referenced and not actually used, leading to unnecessary gas consumption.
	`import '../libraries/PoolAddress.sol';`
	2. Keyword Optimization:
	In the ChainId library, the `get` function only retrieves the current chain ID and does not involve any other function calls. Therefore, the view keyword can be optimized to pure to reduce gas consumption.
	`function get() internal view returns (uint256 chainId) { assembly { chainId := chainid() } }`

Recommendation	Gas optimization in smart contracts can lower transaction execution costs and improve transaction efficiency, enabling developers and users to deploy, interact with, and process data in smart contracts at a lower cost. Therefore, we currently propose the following two suggestions for gas optimization: 1. Remove the unused reference to PoolAddress.sol. 2. Change the keyword of the get function from view to pure.
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.

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.

3.1.3 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 FunC language should strictly check for gas consumption and trigger events on critical changes.

- **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.

TON Features

*Gas Consumption:

Contracts should be strictly checked for gas, so that execution does not fail due to insufficient gas in a TON to execute all the code, and that code already executed is not rolled back.

*Message Forgery:

In FunC there are parent and child contracts that need to verify messages between the parent and child, and if a forged message is accepted there may be risks such as arbitrary coin minting by an attacker.

*Restore on Failure:

When message processing fails, the contract should throw an exception and senders with a fallback flag set should fall back, otherwise a partial execution of the transaction may occur, resulting in a loss of assets.

*DoS (Denial of Service):

As TON supports asynchronous execution, the nature of the func programming language can introduce contention conditions and logic errors due to asynchronous and threaded execution, which can lead to denial of service vulnerabilities.

*Message Flow Error:

In FunC, there are message calls between contracts and the message flow should be checked rigorously for design compliance, otherwise unexpected errors and losses can be introduced.

*Data Structure Error:

When calling the set_data function, you need to pay attention to the order of the arguments, otherwise it may lead to confusing data stored in the contract and seriously affect the business logic.

*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