QuillAudits

Audit Report
July, 2024

For

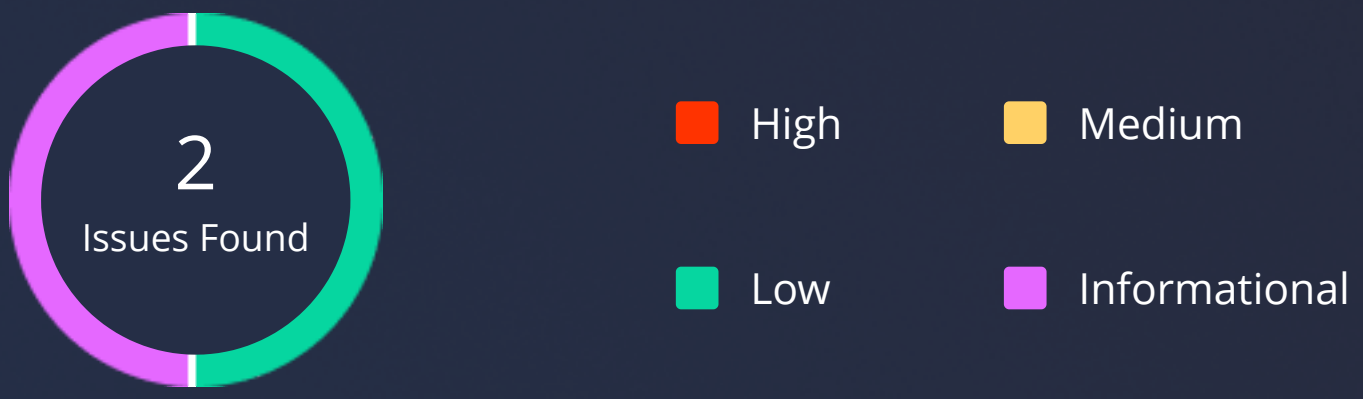
Slinky

Table of Content

Executive Summary

Project Name	Slinky
Overview	Slinky contracts include a token factory and the token smart contracts. The token contract integrates the Initializable and ERC20Upgradeable contracts from Openzeppelin library to initialize a token. The factory creates clones of the token and afterwards creates a pool on UniswapV3 and provide liquidity,
Timeline	13th July 2024 - 20th July 2024
Updated Code Received	22nd July 2024
Second Review	22nd July 2024 - 23rd July 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	This audit aimed to analyze the Slinky Codebase for quality, security, and correctness. 1. TokenFactory.sol 2. StandardToken.sol
Source Code	https://github.com/slinky-ai/token-contracts-v2
Branch	Main
Fixed In	https://github.com/slinky-ai/token-contracts-v2/commit/bd5c725ac7f6003579d8d4bf7df64eeb2b70a67b

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	1

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

No issues found.

Medium Severity Issues

No issues found.

Low Severity Issues

1. Malicious users can create multiple tokens with same token names and symbols

Path

TokenFactory

Function

createToken

Description

Malicious token creators can create multiple tokens with same token names and symbols. The contract is designed not to track unique addresses to their token names.

Status

Acknowledged

Slinky Team's Comment

We are aware of the potential for redundant token names and symbols. However, we have chosen to maintain the simplicity of the contract logic by recognizing only the first created token of a given symbol within our application.

Informational Issues

2. Variables only set at the constructor should be made immutable

Path

TokenFactory

Variables

```
// make variables immutable
IUniswapV3Factory public uniswapV3Factory;
INonfungiblePositionManager public nonfungiblePositionManager;
```

Description

These variables are only set at the constructor level within the contract. This will create a storage slot for each of the variables.

Recommendation

Immutable variables do not take up a storage slot.

Status

Resolved

Functional Tests Cases

Some of the tests performed are mentioned below:

- Should allow the contract owner to update key addresses and base assets info.
- Should activate and deactivate the creation of the tokens
- Should revert when malicious users attempt to create tokens with unapproved baseAsset
- Should confirm that pools are created and initialized on Uniswapv3 for created tokens
- Should allow users to create tokens multiple times with the same token name and tinker
- Should revert when baseAsset configuration is inaccurate.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Slinky codebase. We performed our audit according to the procedure described above.

One Low and One informational severity issues were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the End, Slinky Team Resolved one issue and Acknowledged other.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Slinky smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Slinky smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Slinky to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2024

For

Slinky

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com