QuillAudits

Audit Report
July, 2024

For

Table of Content

Executive Summary

Project Name	Eddy Finance
Overview	Eddy Finance is a decentralized exchange (DEX) built on ZetaChain, aimed at facilitating seamless asset transfers across various blockchains, including Bitcoin (BTC) and Ethereum Virtual Machine (EVM) networks. It is designed to optimize the transfer process by eliminating the need to wrap assets, thereby enabling direct transactions with native assets.
Timeline	27th June 2024 to 29th June 2024
Updated Code Received	26th July 2024 & 10th August 2024
Second Review	26th July 2024
Third Review	12th August 2024 - 13th August 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	The scope of this audit was to analyse the Eddy Finance Codebae for quality, security, and correctness.
Source Code	https://github.com/EddyFinance/Eddy_Stable_Swap/tree/main/contracts
Contracts In-Scope	Branch: Main 1. contracts/LpToken.vy 2. contracts/StableSwap.vy 3. contracts/LpToken.vy 4. contracts/StableSwap.vy
Fixed In	https://github.com/EddyFinance/Eddy_Stable_Swap/commit/72e37b1238ae9c9ec59890d33ef51fa57e6af5fa

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	**1**	**1**	**1**	**3**

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's 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

1. Infamous re-entrancy bug in StableSwap

Path

contracts/StableSwap.vy#L1

Description

StableSwap contract is using vyper compiler version 0.2.8, Which has the infamous re-entrancy bug i.e. re-entrancy lock does not work as desired and allow the malicious user to re-enter into the function and affect the function which gives away the execution to the external contract like remove_liquidity. A detailed explanation exists **here**.

Recommendation

We recommend to use vyper compiler version >= 0.3.1 through out the codebase.

Status

Resolved

Medium Severity Issues

2. Missing ability to handle token transfers that deviate from the ERC20 standard

Path

contracts/TriPool.vy#L452

Function

exchange

Description

The exchange function within the TriPool contract makes an assumption: any exchanged token adheres to the ERC20 standard. Consequently, the implementation utilizes the native transfer function via the ERC20 interface instead of the recommended raw_call method.

This approach presents a potential risk. The native transfer function doesn't offer the same level of control as raw_call. Notably, the current design lacks the ability to programmatically handle token transfers that deviate from the ERC20 standard (i.e., those that don't return a boolean value upon transfer completion).

As a result, many tokens might be inherently incompatible with the pool. Even if such tokens are used, the exchange function would likely revert due to the unsupported transfer behavior.

Recommendation

It is recommend to use raw_call as it has been used throughout the codebase.

Status

Resolved

Low Severity Issues

3. LP tokens can be transferred to empty address

Path

contracts/LpToken.vy#L61 & contracts/LpToken.vy#L77

Function

transfer & transfer_from

Description

The current design of the transfer and transfer_from functions within the LpToken module permits transfers to empty addresses. Ideally, users wouldn't want to send their LP tokens to an empty address, as this would result in losing access to their liquidity or underlying assets.

Recommendation

To prevent accidental loss due to potentially malicious frontend interfaces, it's recommended to disallow transfers to empty addresses until a valid use case for such functionality is identified. This would safeguard users from unknowingly losing access to their LP tokens.

Status

Resolved

Informational Issues

4. Un-necessary event declaration in StableSwap contract

Path

contracts/StableSwap.vy#L17

Description

The StableSwap contract currently includes a TokenExchangeUnderlying event that goes unused. To improve code clarity and maintainability, it's recommended to remove this unnecessary event. This will streamline the code and make it easier to understand.

Recommendation

It is recommended to remove TokenExchangeUnderlying event from the contract.

Status

Resolved

5. Potential for optimization in division by zero handling

Path

contracts/StableSwap.vy#L234

Function

get_D

Description

The current implementation of the StableSwap contract uses a technique to prevent division by zero errors. In the denominator of a calculation, the value 1 is added. While this approach works because Vyper reverts division by zero by default, it's not the most efficient solution.

Vyper offers a built-in function called unsafe_div that replicates the behavior of the Ethereum Virtual Machine (EVM) in division operations. When using unsafe_div, even if the denominator is zero, the function simply returns zero instead of causing a revert.

Recommendation

It is recommended to consider using unsafe_div in place of the current approach to improve code efficiency and readability. This would align the contract's behavior with the standard division behavior within the EVM.

Status

Resolved

6. Infamous oracle maipulation

Path

contracts/StableSwap.vy#L17

Description

It is possible to manipulate the oracle build by external contracts which uses getter function i.e. get_virtual_balance(). This is the known attack vector exists in the curve finance and it also applied in the Eddy Finance StableSwap implementation. More details about the attack vector can be found <u>here</u> .

Recommendation

It is recommended to understand the attack vector from <u>here</u> and recommend suggested fix to the integrators.

Status

Resolved

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 Eddy Finance codebase. We performed our audit according to the procedure described above.

Some issues of High,Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.In The End, Eddy Finance team,Resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Eddy Finance 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 Eddy Finance 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 Eddy Finance 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

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com