QuillAudits

Audit Report
October, 2024

For

KYEX

Table of Content

Executive Summary 03

Number of Security Issues per Severity 04

Checked Vulnerabilities 05

Techniques and Methods 07

Types of Severity 08

Types of Issues 08

High Severity Issues 09

1. Incorrect Handling of Cross-Chain Transfer for WZETA in zrcSwapToNative Function 09

2. Unaccounted Gas Fee Deduction in swapTokens Function 10

Medium Severity Issues 11

3. Denial of Service Due to Zero withdrawGasFee 11

4. Incorrect Platform Fee Calculation Leading to Overcharging Users 12

5. Zero-Value Treasury Transfers May Pause In-Flight Messages in KYEXSwap02 13

6. Incorrect Unwrapping and Transfer of WZETA in onCrossChainCall Function 14

7. Wrong handling of slippage 15

Low Severity Issues 16

8. Redundant Approval Call in transferERC20 Function 16

9. Incomplete Implementation of withdrawBTC Function in onCrossChainCall 17

Table of Content

Informational Issues 18

10. Use safe transfer 18

11. Disable Initializers 18

Automated Tests 19

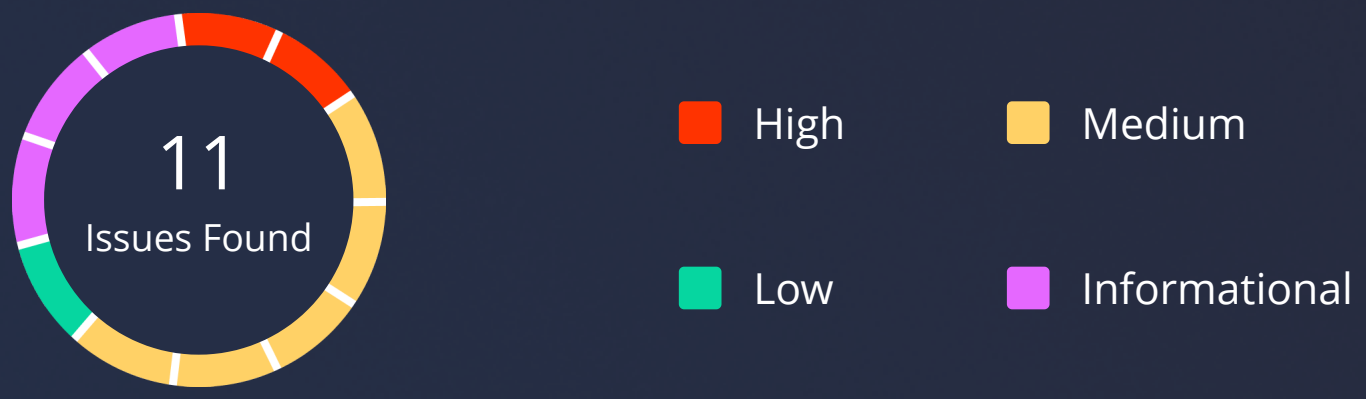
Closing Summary 19

Disclaimer 19

Executive Summary

Project Name	KYEX
Project URL	https://kyex.io/
Audit Scope	The Scope of the Audit is to analyse code quality,security and correctness of Kyex Contracts.
Contracts In-Scope	contracts/KYEXSwap01.sol contracts/KYEXSwap02.sol
Source Code	https://github.com/kyexHead/kyex-swap
Branch	Master
Commit Hash	a7ec7e26b65aa393472a2c5b52758575555642cc
Language	Solidity
Blockchain	Zetachain
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	25th September 2024 - 4th October 2024
Updated Code Received	17th October 2024
Second Review	22nd October 2024 - 24th October 2024
Fixed In	be13bc107db8acc352702ac0f9a630519f77806e

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	0	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	4	2	1

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Style guide violation
- 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 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

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis.

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. Incorrect Handling of Cross-Chain Transfer for WZETA in zrcSwapToNative Function

Path

KYEXSwap01.sol

Function

zrcSwapToNative

Description

In the zrcSwapToNative function of the KYEXSwap01 contract, there is a logical flaw when handling the case where tokenOutOfZetaChain is equal to WZETA.
The issue arises because the transferZETA function is designed to transfer WZETA tokens within the same chain (i.e., it refunds the user on the source chain). However, in the context of a cross-chain swap, the intended behavior is to transfer WZETA tokens to the user's address on a different chain. The current implementation does not do this, it just swaps the inputToken to WZETA in the swapTokens function, then sends it to the user on the ZETA chain.

amountOut = swapTokens(tokenInOfZetaChain, tokenOutOfZetaChain, amountIn, isWrap, slippageTolerance);

```
        if (tokenOutOfZetaChain == WZETA) {
            transferZETA(tokenOutOfZetaChain, msg.sender, amountOut, isWrap);
```

Recommendation

I suggest usage of the ZetaConnectorZEVM contract from the zeta protocol to handle bridging of the ZETA tokens.
After the swap, the contract can unwrap the WZETA tokens and then bridge to the users address.

Status

Resolved

2. Unaccounted Gas Fee Deduction in swapTokens Function

Path

KyexSwap01

Function

swapTokens

Description

In the swapTokens function of the KYEXSwap01 contract, there is an issue where the gas fee (gasFee) is deducted from the input amount when swapping ZETA (WZETA) to another token, but the deducted gas fee is not properly handled or utilized. This leads to the gas fee amount being unaccounted for and remaining in the contract, which can cause discrepancies in the contract's balance and loss of user funds.

When swapping from WZETA to another token (i.e., tokenA == WZETA), the gas fee is subtracted from amountA (amountA - gasFee), but the gasFee is not subsequently used or transferred. This results in the gasFee amount remaining idle in the contract. The unhandled gasFee effectively locks a portion of the user's funds within the contract, leading to an unintended loss for the user.

```
 if (tokenA == WZETA || tokenB == WZETA) {
path = new address[](2); path[0] = tokenA;path[1] = tokenB;
uint256[] memory amounts =
IUniswapV2Router02(UniswapRouter).swapExactTokensForTokens(
 okenB == WZETA ? amountA : amountA - gasFee,
 calculateMinimumOutputAmount(tokenB == WZETA ? amountA : amountA - gasFee, path,
slippageTolerance),path, address(this), block.timestamp + MAX_DEADLINE );amountOut =
amounts[1];
```

Also note that these tokens cannot be withdrawn from the contract.

Recommendation

Since the actual gas fee is deducted when the tokens are being withdrawn from ZETA in the withdrawBTC and withdrawToken function.
Also for non-crosschain swaps, the user gets two deductions instead of one: Gasfee and Platformfee

Status

Resolved

Medium Severity Issues

3. Denial of Service Due to Zero withdrawGasFee

Path

KYEXSwap01 & KYEXSwap02

Description

In the KYEXSwap01 & KYEXSwap02 contracts, there is a vulnerability where the protocol can become unusable if the withdrawGasFee function returns zero. This function call is crucial as it determines the gas fee required for token withdrawal operations. The ZETA protocol, which the contract interacts with, has the ability to set the withdrawGasFee to zero through the system contract.

Although unlikely, if the fees are set to zero, all swap calls to swap from ZRC20 to their respective gasZRC20s will make the function break, causing multiple reverts and hindering actual usage of the contract.

In the KYEXSwap02 contract, the inflight messages from the System Contract will also not succeed

Recommendation

Always check that the fee is always greater than zero before attempting the swap through the Uniswap Contract

Status

Resolved

4. Incorrect Platform Fee Calculation Leading to Overcharging Users

Path

KYEXSwap01

Function

swapTokens

Description

In the swapTokens function of the KYEXSwap01 contract, there is a miscalculation in how the platform fee is applied. The platform fee is calculated based on the amountOut before deducting the gas fee, which results in users being overcharged. Since the gas fee is an additional cost that should not be included in the platform's revenue, the platform fee should be calculated after the gas fee has been deducted.

By calculating the platform fee on the total amountOut, including the gas fee, users are effectively paying a fee on funds that are not part of their total funds

//@audit-ok Med platformfee deducted before gasfee deduction, this is an issue because platform fee takes a percentage

```
uint256 feeAmount = amountOut * platformFee / 10000;
uint256 newAmount = amountOut - feeAmount;
if (feeAmount > 0) {
   TransferHelper.safeTransfer(tokenB, kyexTreasury, feeAmount);
}
```

Recommendation

To correct this issue, the platform fee should be calculated based on the amount after the gas fee has been deducted. This would be done in the functions after the swapTokens function, majorly the transferZETA, withdrawBTC, withdrawToken, transferZRC.

Status

Resolved

5. Zero-Value Treasury Transfers May Pause In-Flight Messages in KYEXSwap02

Path

KYEXSwap02

Description

In the KYEXSwap02 contract, there is an issue where zero-value transfers to the treasury are not checked which will always cause reverts if feeAmount == 0, causing transactions to fail. This occurs because the contract does not check whether the feeAmount is greater than zero before attempting to transfer it to the kyexTreasury. Since ERC20 tokens generally revert on zero-value transfers, the absence of this check can halt the execution of the onCrossChainCall function which is a call coming from the SystemContract, effectively pausing in-flight messages.

In the KYEXSwap01 contract, this is done properly, as each zero value transfer to the treasury is checked and omitted. The absence of if (feeAmount > 0) before the transfer means that zero-value transfers are attempted, leading to reverts.

Recommendation

To resolve this issue, the KYEXSwap02 contract should include a check to prevent zero-value transfers to the treasury so that small token transfers do not cause reverts.

```
uint256 feeAmount = swapAmounts.outputAmount * platformFee / 10000;
uint256 newAmount = swapAmounts.outputAmount - feeAmount;

// Add zero-value check to prevent reverts
if (feeAmount > 0) {
    TransferHelper.safeTransfer(targetTokenAddress, kyexTreasury, feeAmount);
}
```

Status

Resolved

6. Incorrect Unwrapping and Transfer of WZETA in onCrossChainCall Function

Path

KYEXSwap02

Function

onCrossChainCall

Description

In the onCrossChainCall the, when isWIthdraw == 3, the function calls transferERC20 to handle ERC20 tokens, however this is not completely done properly.

```
} else if (isWithdraw == 3) {
        //Unwrap and transfer (for WZETA)
        transferERC20(zrc20, targetTokenAddress, amount, recipientAddress, slippage);
```

This particular condition for unwrapping before transferring WZETA was not handled in the transferERC20 function. Although this does not cause any reverts, the code does not provide the required output.
If the recipient is a contract, expecting native ZETA tokens, it would receive WZETA tokens which are ERC20 tokens it is unable to handle.

Recommendation

Implement a check in transferERC20 function to handle WZETA tokens, i.e, unwrapping and transferring tokens if targetTokenAddress == WZETA

Status

Resolved

7. Wrong handling of slippage

Path

KYEXSwap02, KYEXSwap01

Description

Although slippage is handled in the contracts, and calculated properly. There is a critical oversight

Because the values are gotten onchain, within the same transaction as the swap,if the pool is imbalanced, the price used for calculations would be imbalanced too. This leaves the contract vulnerable as well. A MEV researcher can still manipulate a pool before transactions are actually processed.

Recommendation

It is best to utilize an offchain slippage method, i.e, enable users to pass in the slippage value as a parameter, not just a percentage or calculating based on an oracle. This would give the maximum protection.

Status

Acknowledged

Low Severity Issues

8. Redundant Approval Call in transferERC20 Function

Path

KYEXSwap02

Function

transferERC20

Description

In the transferERC20 function of the KYEXSwap02 contract, there is a redundant line of code that unnecessarily attempts to approve the token contract to spend tokens on behalf of itself.

if (!IZRC20(targetTokenAddress).approve(targetTokenAddress, outputAmount)) revert Errors.ApprovalFailed(); // @audit LOW redundant line

This line calls the approve function on targetTokenAddress, allowing targetTokenAddress to spend outputAmount tokens from the contract's balance. However, since the contract is transferring tokens directly to the recipient and does not require any external contract to spend tokens on its behalf, this approval is unnecessary.

Recommendation

To resolve this issue, remove the redundant approval line from the transferERC20 function

Status

Resolved

9. Incomplete Implementation of withdrawBTC Function in onCrossChainCall

Path

KYEXSwap02

Function

onCrossChainCall

Description

In the KYEXSwap02 contract, the withdrawBTC function is intended to handle the withdrawal of BTC tokens. However, there is an issue where this function is not properly integrated within the onCrossChainCall function. Specifically, when the contract is supposed to process BTC withdrawals during cross-chain calls, the withdrawBTC function is either unimplemented or not invoked as needed.

The onCrossChainCall function should handle different types of token transfers based on the isWithdraw parameter. However, for cases involving BTC (e.g., when tokenOutOfZetaChain == BITCOIN), the function does not invoke withdrawBTC as required.

Recommendation

Consider implementing the withdrawBTC function within the onCrossChainCall to enable its usage.

Status

Resolved

Informational Issues

10. Use safe transfer

Path

KYEXSwap02, KYEXSwap01

Description

Although there are no unexpected tokens to be used in the protocol asides ZETA tokens, it would still be a good practice to consistently use the TransferHelper library to handle all ERC20 token transfers.

Recommendation

Use the appropriate function from the TransferHelper library to handle all necessary transfers.

Status

Resolved

11. Disable Initializers

Path

KYEXSwap02, KYEXSwap01

Recommendation

To prevent reinitialization of the implementation contract, it is advised to use the _disableInitializers function from the openzeppelin library.

Status

Acknowledged

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 KYEX. We performed our audit according to the procedure described above.

Some issues of High,low,medium and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in KYEX. 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 KYEX. 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 KYEX 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
October, 2024

For

KYEX

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com