QuillAudits

Audit Report
July, 2024

For

InterSwap

Table of Content

Executive Summary 03

Number of Security Issues per Severity 05

Checked Vulnerabilities 06

Techniques and Methods 07

Types of Severity 08

Types of Issues 08

General Contract Overview 09

Low Severity Issues 12

1. Use of incorrect error 12

Informational Issues 12

2. Redundant import 12
3. Redundant function 13
4. Check transferFrom allowance subtraction logic 14
5. Log ln() function will give error on values less than 0 15
6. No zero value check for variable 15
7. Unused function and error 16
8. Floating solidity (pragma solidity ^0.8.24) 16
9. Poor in-line code documentation 17

Table of Content

Functional Tests Cases 18

Automated Tests 19

Closing Summary 20

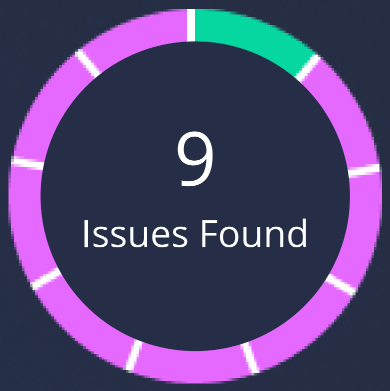
Disclaimer 20

Executive Summary

Project Name	Interswap
Overview	Described in [General Contract Overview]
Timeline	9th May, 2024 - 30th May, 2024
Updated Code Received	NA
Second Review	NA
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 Interswap codebase for quality, security, and correctness.
Source Code	*https://github.com/Interswap-labs/interswap-contracts*
Commit	4ecedef
Contracts In-Scope	- contracts/Interswap.sol - contracts/base/factories/BasePoolFactory.sol - contracts/base/factories/InterswapFactory.sol - contracts/base/factories/WeightedPoolFactory.sol - contracts/base/pools/InterswapPool.sol - contracts/base/pools/WeightedPool.sol - contracts/base/routers/BaseRouter.sol - contracts/base/routers/ChildRouter.sol - contracts/base/routers/MasterRouter.sol - contracts/base/InterswapERC20.sol - contracts/base/InterswapLock.sol - contracts/base/InterswapTreasury.sol

	- contracts/gmp/axelar/AxelarGMP.sol - contracts/gmp/BaseGMP.sol - contracts/libraries/AccessControlExtended.sol - contracts/libraries/ExcessivelySafeCall.sol - contracts/libraries/FixedPoint.sol - contracts/libraries/InterswapConstants.sol - contracts/libraries/InterswapErrors.sol - contracts/libraries/InterswapLibrary.sol - contracts/libraries/LogExpMath.sol - contracts/libraries/Salvagable.sol - contracts/libraries/TransferHelper.sol - contracts/libraries/WeightedMath.sol - contracts/libraries/WordCodec.sol
Branch	main
Contracts Out of Scope	In-scope contract has been audited by QuillAudits. However, these contracts inherit functionality from out-of-scope Smart contracts that were not audited. Vulnerabilities in unaudited contracts could impact in-scope Smart Contracts functionality. QuillAudits is not responsible for such vulnerabilities. Below are Out of Scope Contracts: - contracts/interfaces/* - OpenZeppelin contracts (Initializable.sol, ...)
Fixed In	NA

Number of Security Issues per Severity



🟥 High	🟨 Medium
🟩 Low	🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	**1**	**8**
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

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

Manual Review, Slither, Hardhat.

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.

General Contract Overview

A. Factories

BasePoolFactory

The BasePoolFactory contract is basically a getter contract that unfurls the internal _createPool function which creates a pool based on the bytecode passed in. It performs checks on the data attributes in between i.e. checksum with the bytes32(0) duplicate check, poolAddr with the create2 address(0) check which occurs when create2 is called a second time.

InterswapFactory

The InterswapFactory contract allows the MasterRouter to call createPool to make weighted pools (the only pool audited as of the time of this report). It sets the defaultSwapFee on the created pool and adds it to the allPools array.

WeightedPoolFactory

The WeightedPoolFactory contract creates a new weightedPool using type(weightedPool).creationcode.

B. Pools

InterswapPool

Privileged users here are provisioners (an array of addresses provided at initialization), the MasterRouter contract and Factory. The Factory contract can create new pools, while the MasterRouter is the only user allowed to swap, addLiquidity and removeLiquidity from the pool.

A security check for proper account liquidity handling is done with provisioners. Provisioners are the initial liquidity providing addresses that are expected to add liquidity in before the 24-hour window elapses for assets to be termed secure.

The pool initializes with a fixed number of tokenIds in its assetsArray. It maintains a minimum liquidity amount in its balance, (most likely) to avoid rounding errors in calculations. Tokens with up to 18 decimals of precision can be used in the created pools, tokens with more than 18 decimals are unsupported.

WeightedPool

This pool extends the functionality from the InterswapPool while implementing the functions which weren't in the base pool contract. The WeightedMath library is used to handle the calculations involved here. Weighted pools estimate the expected amountOut based on the current balances and weights of other tokens in the pool.

C. Routers

BaseRouter.sol

Inherits from the Salvagable contract, exposing the _salvage internal function which sends out all tokens left in the router. onlyAdmin can call this. Only Axelar's GMPService is able to call handle() which in turn calls the internal _handle() implemented in the ChildRouter and MasterRouter contracts individually.

ChildRouter.sol

The ChildRouter inherits from the BaseRouter contract and implements the internal _handle and _call functions here. When users addLiquidity or removeLiquidity or swap, if either of these functions is called on the masterChain, it sends the msgFee to the MasterRouter and performs a localCall using Axelar's gmpService, but if it is called on any other chain not the masterchain, it uses _sendMessage to make a remoteCall to the destination chain via Axelar while forwarding the msgFee to the gmpService contract.

The Axelar GMPService calls handle to execute the payloads transferred crosschain, it checks the first few bytes of the payload to determine what kind of payload it is (either a swap, remove liquidity, createpool or reverted payload).

MasterRouter.sol

The MasterRouter also inherits from the BaseRouter contract and handles the transactions that go across the srcChain and dstChain. LP tokens get minted into the treasury which is defined and updated here as well.

D. Other Libraries

AccessControlExtended

This contract extends the regular functionality provided by OpenZeppelin's AccessControl. It introduces two new roles, GOVERNANCE_ROLE and AUTHORIZED.

InterswapLibrary

The InterswapLibrary handles the majority of the math-heavy computation involved in the InterswapPool.

Low Severity Issues

1. Use of incorrect error

Path

InterswapFactory.sol

Function

setDefaultSwapFee() and setSwapFee()

Description

Both **setDefaultSwapFee()** and **setSwapFee()** functions are checking that if **_swapFee** is less than **InterswapConstants._MIN_FEE_NUMERATOR (99e16)** then it reverts the transaction with **InterswapErrors.TooBigFee()** error which is incorrect as its checking if **_swapFee** is **less** and reverting with error that says **TooBigFee**.

Remediation

Verify and change the error to meaningful error which will reperesent correct situtation e.g TooSmallFee.

Status

Acknowledged

Informational Issues

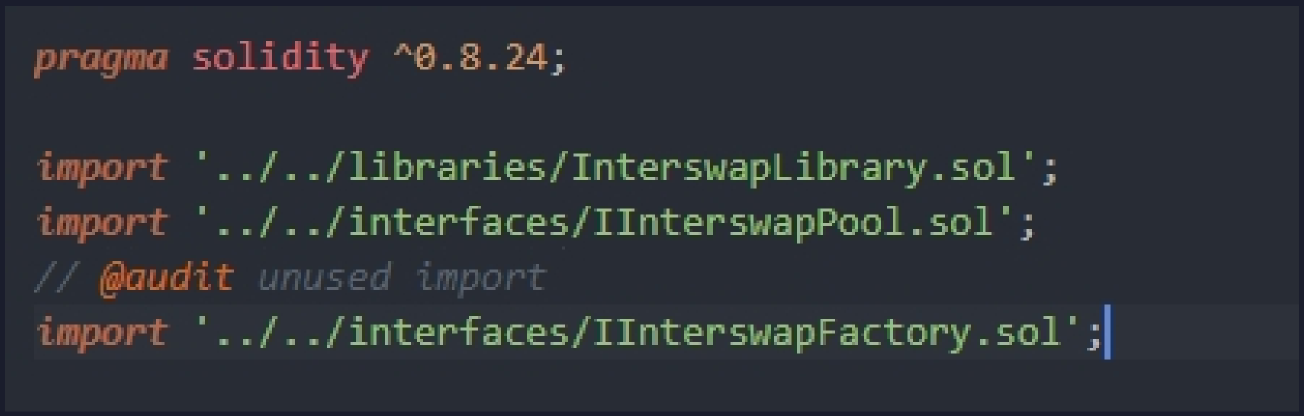
2. Redundant import

Path

BasePoolFactory

Description

import for InterswapConstants is redundant as anything from InterswapConstants is not getting used in BasePoolFactory



```
pragma solidity ^0.8.24;

import '../../libraries/InterswapLibrary.sol';
import '../../interfaces/IInterswapPool.sol';
// @audit unused import
import '../../interfaces/IInterswapFactory.sol';
```

Remediation

Consider removing the redundant imports.

Status

Acknowledged

3. Redundant function

Path

BasePoolFactory

Function

getPool()

Description

getPool() is redundant as **pools** mapping is public, the compiler automatically creates getter functions for all public state variables.

Remediation

Consider removing redundant functions if not needed.

Status

Acknowledged

4. Check transferFrom allowance subtraction logic

Path

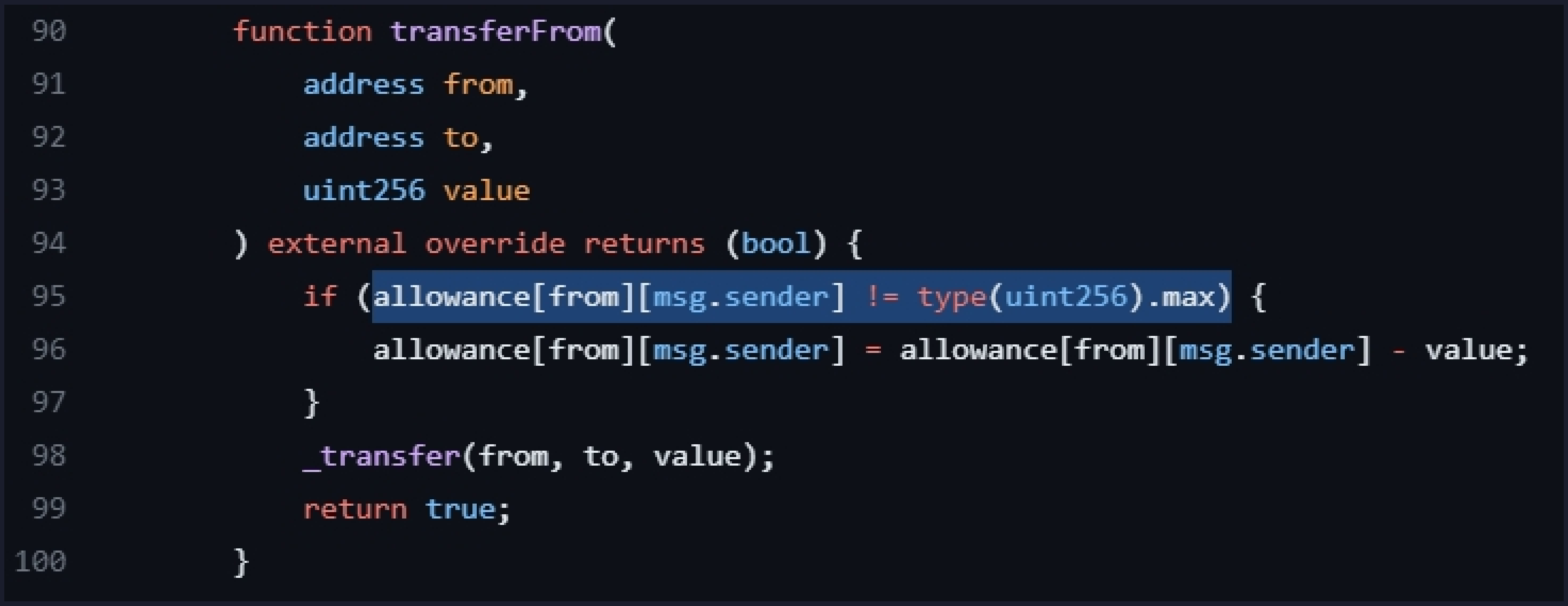
InterSwapERC20

Function

transferFrom

Description

In case someone sets a uint.max allowance then the contract never subtracts the allowance in transferFrom(). transferFrom function only subtracts the allowance when its not set as uint.max value (i.e when its less than uint.max) because of the if statement as shown below:



```
    function transferFrom(
        address from,
        address to,
        uint256 value
    ) external override returns (bool) {
        if (allowance[from][msg.sender] != type(uint256).max) {
            allowance[from][msg.sender] = allowance[from][msg.sender] - value;
        }
        _transfer(from, to, value);
        return true;
    }
```

Remediation

Check that it's intentional.

Status

Acknowledged

5. Log ln() function will give error on values less than 0

Path

LogExpMath

Function

Log ln()

Description

In library contracts LogExpMath there is log function ln() which takes int256 means it should not take any values less than 0 but the check which is implemented is:
if(a == 0) revert InterswapErrors.OutOfBounds() but it also should revert for values less than 0.

Remediation

Make sure to change the if condition to **if(a <= 0) revert InterswapErrors.OutOfBounds()**.

Status

Acknowledged

6. No zero value check for variable

Path

FixedPoint

Function

Log ln()

Description

In library contracts FixedPoint file there are functions divUp() and divDown() which take a,b as input parameters but they do not check zero value for b variable as it is used as denominator which might result in 0 value.

Remediation

Make sure to check for zero value for b variable.

Status

Acknowledged

7. Unused function and error

Path

BaseRouter.sol#L31, InterswapPool#L278-281

Function

_updateChecksum()

Description

In the InterswapPool contract, the _updateChecksum function is defined but never used. Same with the UnexpectedFailedMethod error in the BaseRouter. The unused functions add to the contract bytecode size and could increase deployment cost unnecessarily.

Remediation

Remove unused lines of code in the codebase.

Status

Acknowledged

8. Floating solidity (pragma solidity ^0.8.24)

Description

When specific solidity pragma version is not defined in a solidity program, it could cause the program to be compiled with either outdated or recent compiler, possibly introducing some known and/or undiscovered bugs. It is a good practice to have a locked solidity compiler version to avoid deployed contracts with bugs.

Remediation

Making the contract use a locked solidity pragma version prevents bugs that could be ushered in by prospective or outdated versions. Using a fixed solidity pragma version while deploying is recommended to avoid deployment with versions that could expose the contract to bugs.

Status

Acknowledged

9. Poor in-line code documentation

Description

Majority of the codebase does not have in-line comments explaining the expected flow of logic. Since this is a novel application which has the potential to be integrated/forked by other users, it would be best to reduce ambiguity to the barest minimum by using the Ethereum Natural Language Specification Format (NATSPEC) to document these.

Remediation

Consider writing comments in-line. The format specified in the Solidity Language documentation found here can be adapted.

Status

Acknowledged

Functional Tests Cases

- ✓ Test if liquidity can be added via unknown token
- ✓ Test if add Liquidity function can be called with 1 wei
- ✓ Test if swap can be performed on same chain
- ✓ Test if tokens not return to users if swap failed
- ✓ Test if tokens left in contract after revert
- ✓ Check tokens like USDT
- ✓ Test if fees are takes correctly
- ✓ Test createPool function
- ✓ Test if createPool can take same chainID
- ✓ Test all the getter functions
- ✓ Test that data is properly decoded during _handleX() calls
- ✓ Test tokens with differing decimals

InterswapERC20:

- ✓ Should be able to transfer tokens
- ✓ Should be able to approve tokens to other address
- ✓ Should be able to transferFrom approved tokens
- ✓ Should be able to permit approval of tokens to address
- ✓ Should revert while transferFrom if msg.sender isn't approved
- ✓ Should revert while transferring if sender doesn't has enough balance
- ✓ Should revert if user tries to use one signature more than once
- ✓ Should revert if user tries to use signature after deadline expired
- ✓ Should revert if recoverred address is not the owner address

TransferHelper:

- ✓ uniTransfer is able to tranfer ETH using safeTransferETH
- ✓ uniTransfer is able to transfer ERC20 tokens using safeTransfer
- ✓ Should be able to tranfer approved tokens using safeTransferFrom

InterswapTreasury:

- ✓ Only admin should be able to withdraw
- ✓ Should revert if token and amount length do not match

InterswapLock:

- ✓ Should be able to lock and unlock tokens
- ✓ Should be able to register tokens
- ✓ Should be able to add tokens
- ✓ Should be able to remove tokens

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

Issues of Low and Informational severity were found, suggestions and best practice 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 Interswap 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 Interswap 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 Interswap 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+	**$30B**	**1M+**
Audits Completed	Secured	Lines of Code Audited

Follow Our Journey

Audit Report
July, 2024

For

InterSwap

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com