QuillAudits

Audit Report
July, 2022

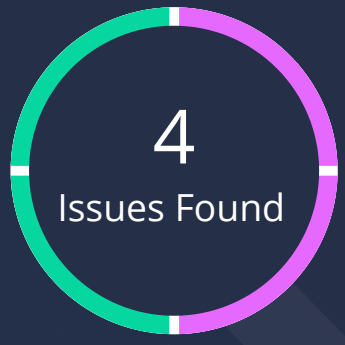
For

alium
finance

Table of Content

Executive Summary

Project Name	Alium Finance
Timeline	20 June, 2022 - 30 June, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Alium Finance(MulticallUserExecutable.sol) codebase for quality, security, and correctness.
Sourcecode	*https://github.com/Alium-Finance/tokenbridge-extended-contracts/blob/feature/fees-and-oracle/contracts/MulticallUserExecutable.sol*
Commit hash	824f21748658e55967c0dc9ee99694bc8706feb9
Fixed in	eced6f3167e3f279dafccba11b856dc9cf167f10



High Medium

Low Informational

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

Types of Severities

High

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

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

Low-level severity issues can cause minor impact and or 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 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.

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
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility leve

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour 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 analysed 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

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 analysed, 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 behaviour 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, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Analysis

A. Contract - MulticallUserExecutable.sol

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

A.1 Centralization Risk

Function - All the functions with the OnlyOwner modifier

Description

Using the “OnlyOwner” modifier means that only the owner of the contract will be able to call the functions like setting a new amb, alm, fee address, price oracle etc. and it poses a considerable centralization risk in a scenario where the private keys of the owner is lost/stolen then the attackers could simply control the price oracle or set the fee address that they control.

Remediation

Instead of a single owner of the contract, there can be multiple accounts that can be added and used as owners. Lost keys or compromised accounts can be blacklisted/blocked.

Status

Acknowledged

A.2 Price Oracle Manipulation

Description

This risk of the price being manipulated by an attacker is always there while using decentralized price oracles. There could be many scenarios where the price of a particular token can be manipulated by the attacker as this price data is fetched from an on-chain dependency, and the price data is determined in the current transaction context, the spot price can be manipulated in the same transaction while querying AMB contract via MulticallUserExecutable contract

Consider this scenario for example:

1. An attacker can take out a flash loan on the incoming asset A and on the relevant Uniswap pool, swap asset A for asset B with a large volume.
2. This trade will increase the price of asset B (increased demand) and reduce the cost of asset A (increased supply).
3. When asset B is deposited into the above function, its price is still pumped up by the flash loan.
4. Consequentially, asset B gives the attacker an over-proportional amount of shares.
5. These shares can be withdrawn, giving the attacker equal parts of asset A and asset B from the pool.
6. Repeating this process will drain the vulnerable pool of all funds.
7. With the money gained from the withdrawal of their shares, the attacker can repay the flash loan.

For more context refer to the following:
1. *https://consensys.github.io/smart-contract-best-practices/attacks/oracle-manipulation/#spot-price-manipulation*
2. *https://extropy-io.medium.com/price-oracle-manipulation-d46fd413cc17*

Remediation

Using a median of multiple oracles provides heightened security since it is harder and more expensive to attack various oracles. It also ensures that a smart contract gets the data it needs even if one oracle or API call fails. Moreover, a centralized oracle can be considered that could never be controlled by the attacker.

Status

Acknowledged

Informational Issues

A.3 Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.

Status

Fixed

A.4 Potential Risk of Uniswapv2 oracle

As we observed that the uniswapv2 oracle was used for testing purposes so we would like to mention a risk that it poses as well:

In the price oracle contract, it must be decided on what time interval to use, which can be tricky. Using TWAP for price calculation can be done but a shorter time interval means that we will see price updates quicker, but also lowers the cost of attack to manipulate the oracle. A longer time interval makes it much harder to manipulate the average price, but also means we won’t be able to react to the volatility in the markets.

Status

Fixed

Functional Testing

Some of the tests performed are mentioned below

- ✓ should be able to execute batch transfer of tokens by multicall contract to amb
- ✓ Should be able to deduct fee before transferring funds to amb
- ✓ Should be able to get functional signature of the input data and calculate scenario hash
- ✓ Should be able to get price of eth to deduct as fee by the oracle contract
- ✓ Should be able to log events.
- ✓ Should be able to set price oracle even with the zero address if fee deduction needs to be halted.
- ✓ Should revert if the destination in the Input data points to an Unverfied DEX, Unverified Bridge, or Unverified ALM
- ✓ Should revert if the fee address is set to zero or the fee amount is set to zero.
- ✓ Should revert if hash/signature counting is wrong

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

Some issues ofMedium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. At the End, Alium Finance team Acknowledged the Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Alium Finance Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Alium FinanceTeam put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2022

For

alium finance

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com