QuillAudits

Audit Report
August, 2024

For

Table of Content

Executive Summary

Project Name	AirLyft
Overview	AirLyft is a crypto-first growth platform built for Web3.0 - integrate our specialized apps to grow your outreach, integrate smart contracts, create bot-free giveaways with tokens, NFTs, coupons etc. as prizes.
Timeline	18th July 2024 - 29th July 2024
Updated Code Received	19th August 2024
Second Review	26th August 2024 - 27th 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 AirLyft Contracts for quality, security, and correctness.
Source Code	https://github.com/Kyte-Research/airlyft-contract/tree/master/contracts
Contracts In-Scope	1. contracts/erc20/airpools/ERC20AirPoolController.sol 2. contracts/erc20/airpools/IERC20AirPoolController.sol 3. contracts/Claimable.sol
Branch	Master
Fixed In	https://github.com/Kyte-Research/airlyft-contract/tree/fix/quillaudits-recomendations

Number of Security Issues per Severity

5
Issues Found

🟥 High	🟨 Medium
🟩 Low	🟪 Informational

	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	2	1	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'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. Fee on transfer tokens unhandled properly

Path

contracts/erc20/airpools/ERC20AirPoolController

Path

create(), deposit()

Description

This smart contract intends to be compatible with any ERC20 token, which includes the Fee-On-Transfer tokens .An example would be the USDT contract which has the functionality in its contract but is currently disabled. If enabled, it would break functionality of the contract since the recorded totalLiquidity is now less than the actual token balance of the contract.

This can be exploited in a scenario where a user USER A creates a pool to airdrop a popular token to multiple users. A malicious user USER B could create another pool with the same token, then run multiple deposits and withdrawals which will reduce the token balance of the contract address while maintaining the pool's totalLiquidity. USER B then calls withdraw() to pull their funds from the contract while the token balance is now less than the totalLiquidity of USER A pool.

POC

```
function create( address creator,  address poolToken, uint256 initialLiquidity,
 Datastructures.CertificateInfo calldata certificate) external override nonReentrant
  whenNotPaused returns (bytes32 poolId) {
  .........
  pool.totalLiquidity = initialLiquidity;
SafeERC20.safeTransferFrom( IERC20(poolToken),creator,address(this),
initialLiquidity);

function deposit(
   address user, bytes32 poolId,uint256 amount,
   Datastructures.CertificateInfo calldata certificate
 ) external payable override nonReentrant whenNotPaused {
   pool.totalLiquidity = pool.totalLiquidity.add(amount);
   if (pool.poolType == PoolType.LOCKED_TOKEN)
{ SafeERC20.safeTransferFrom( IERC20(pool.token), user, address(this), amount ); }
```

Recommendation

Get the token balance of the contract before and after the transfer has occurred. Then increment the pool's totalLiquidity with the delta of the two values.

Here is the changes to be added to create():

```
uint balanceBefore =  IERC20(poolToken).balanceOf(address(this));
   SafeERC20.safeTransferFrom( IERC20(poolToken), creator, address(this), initialLiquidity);
uint balanceAfter =  IERC20(poolToken).balanceOf(address(this));
  pool.totalLiquidity = balanceAfter - balanceBefore;
```

Status

Resolved

Medium Severity Issues

2. SafeTransfer/SafeTransferFrom

Path

contracts/erc20/airpools/ERC20AirPoolController

Function

withdraw(), claim(), claimBatch()

Description

Not all tokens match the standard of returning false on failure, some tokens might not revert on failure, while some do not return any boolean.

It is therefore advised to use the SafeERC20 library already imported from Open Zeppelin to handle transfers from the contract.

POC

```
if (pool.poolType == PoolType.LOCKED_TOKEN) {
    bool success = IERC20(pool.token).transfer(user, amount);
    require(success, "ERC20APC:claim-failed");
```

Recommendation

Use the SafeERC20 library to handle transfers while claiming and withdrawing.

```
if (pool.poolType == PoolType.LOCKED_TOKEN) {
    SafeERC20.safeTransfer(IERC20(pool.token), user, amount);
```

Status

Resolved

3. ETH Stuck in contract

Path

contracts/erc20/airpools/ERC20AirPoolController

Function

receive(), fallback(), deposit()

Description

In the deposit() function of the contract. It permits ERC20 deposits to ERC20 pools, however since it is a payable function, users can in error pass in eth to the deposit call as an msg.value .

Also the contract consists of a receive()and fallback()which allows eth to be sent directly to the contract. Any eth sent to the contract will be permanently stuck in the contract.

POC

```
 receive() external payable {}
 fallback() external payable {}
```

See the **deposit()** function.

Recommendation

In the deposit() function, while performing the token transfer in the if block, revert if there msg.value>0.

```
  if (pool.poolType == PoolType.LOCKED_TOKEN) {
    if(msg.value > 0) revert();
    SafeERC20.safeTransferFrom( IERC20(pool.token), user,address(this),
      amount );
  }
```

Also remove the receive()and fallback()functions or add an admin withdrawal function and add the onlyOwner() modifier to it.

Status

Resolved

Low Severity Issues

4. use call() instead of transfer() for EVM compatibility

Path

contracts/erc20/airpools/ERC20AirPoolController

Function

withdraw(), claim(), claimBatch()

Description

Usage of .transfer() instead of .call() to send ether during withdrawals has its potential downsides especially when using this across multiple EVM chains as mentioned by the Client.

Some chains might not support the opcode, something similar would be the **Zksync bug involving 921 ETH** .

Also there are potentials for break in smart contract functionality due to future **upgrades** in gas prices.

POC

```
  payable(user).transfer(amount);
```

Recommendation

Use call() instead of transfer as seen below.

```
 (bool success, ) = user.call{value:amount}("");
  require(success, "Transfer failed.");
```

Status

Resolved

Informational Issues

5. use of safemath library

Path

ERC20AirPoolController.sol,Claimable.sol;

Description

At multiple points within the repo, SafeMath is used to handle addition .add() and subtraction .sub(), however it is important to note that the solidity version used here being : pragma solidity 0.8.17; already handles overflow and underflow checks, hence it holds no functionality within this contract.

Recommendation

Use + instead of .add() and - instead of .sub()

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 AirLyft 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, the AirLyft Team resolved all issues

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in AirLyft 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 AirLyft 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 AirLyft 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
August, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com