QuillAudits

Audit Report
August, 2024

For

Table of Content

Table of Content

Executive Summary

Project Name	Lanshare
Overview	The project comprises a suite of smart contracts designed to facilitate secure cross-chain token transfers and staking using Chainlink's Cross-Chain Interoperability Protocol (CCIP). Users initiate cross-chain messages via the CrossChainSender, which are received and processed by the CrossChainReceiver, subsequently interacting with the CrossChainVault to deposit bridged tokens. The CrossChainVault remains synchronized with the MasterChef staking protocol, and upon withdrawal requests, tokens are transferred back to the user using the CCIPTokenSender, ensuring seamless cross-chain operations and robust security.
Timeline	30th July 2024 - 6th August 2024
Updated Code Received	08th August 2024
Second Review	08th August 2024 - 09th 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 analyze the Landshare Contract for quality, security, and correctness.
Source Code	https://github.com/mogw/land-ccip/blob/main/contracts/v1/CrossChainVault.sol https://github.com/mogw/land-ccip/blob/main/contracts/v1/CrossChainSender.sol https://github.com/mogw/land-ccip/blob/main/contracts/v1/CrossChainReceiver.sol https://github.com/mogw/land-ccip/blob/main/contracts/v0/CCIPTokenSender.sol

Executive Summary

Contracts In-Scope	CrossChainVault.sol CrossChainSender.sol CrossChainReceiver.sol CCIPTokenSender.sol
Branch	Main
Fixed In	7bb3489280caa7ab5059e17a22a77b6b17e39bb6

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	3
Partially Resolved Issues	0	0	0	0
Resolved Issues	5	1	0	2

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. Unauthorized “Harvest” Call Allows Users to Drain Admin-Provided LINK and Receive Call Fees

Path

contracts/v1/CrossChainSender.sol

Function

transfer

Description

User A can initiate a “Harvest” instruction in CrossChainSender.sol and set themselves as _onBehalfOf. The cost for bridging this transaction is paid by the admin, but the user receives the call fee. This allows the user to exploit the system by repeatedly initiating harvest calls, draining the admin-provided LINK in CrossChainSender contract, and collecting call fees without making any actual investments.

Recommendation

Restrict the “Harvest” instruction so that only the admin can execute it, ensuring that only the entity paying the bridge fees can initiate the harvest. This prevents unauthorized users from exploiting the system to drain LINK and receive call fees.

Status

Resolved

2. The onBehalfOf Parameter in CrossChainSender Allows Unauthorized Withdrawals

Path

contracts/v1/CrossChainSender.sol

Function

transfer

Description

In the CrossChainSender contract, the onBehalfOf parameter can be set to any user, enabling one user to withdraw tokens on behalf of another user. This creates a vulnerability where an attacker can force unauthorized withdrawals, causing the target user to pay a withdrawal fee if the tokens are withdrawn before a set time. The withdrawal fee is a percentage value, and for users with large investments, this fee can sum up to significant amounts, leading to substantial financial losses. Additionally, it is inappropriate and insecure for an attacker to decide when another user's tokens should be unstaked, as this undermines the user's control over their investments and timing of withdrawals.

Recommendation

Remove the onBehalfOf parameter from the transfer function in the CrossChainSender contract. Ensure that onBehalfOf is always set to msg.sender to prevent unauthorized withdrawals and to maintain user control over the timing and execution of their token unstaking.

Status

Resolved

3. Incorrect Financial Model for Sending Cross-Chain Transactions Leads to Potential LINK Drain

Path

contracts/v1/CrossChainSender.sol

Function

transfer

Description

In the CrossChainSender contract, the admin pays for the Chainlink Cross-Chain Interoperability Protocol (CCIP) call fees in the transfer function. However, this fee is incurred even if the transaction reverts. A malicious user could exploit this by sending a withdraw message with a large amount to bypass contract protections, knowing that the transaction might revert due to insufficient tokens in the CrossChainVault contract. Despite the transaction reverting, the admin would still pay for the CCIP call, leading to a potential drain of LINK tokens from the CrossChainSender, causing financial loss for the protocol and risking a denial of service (DoS) if the LINK balance is depleted.

Recommendation

Revise the financial model to require users to cover the CCIP message fees instead of the admin. This change will ensure that users bear the cost of their transactions and mitigate the risk of malicious users draining LINK tokens from the protocol.

Status

Resolved

4. Potential Gas Limit Exploitation in CrossChainSender

Path

contracts/v1/CrossChainSender.sol

Function

transfer

Description

According to CCIP documentation, the gasLimit specifies the maximum amount of gas that CCIP can consume to execute ccipReceive() on the destination blockchain. This gas limit directly influences the fee for sending a message, and any unspent gas is not refunded. In the CrossChainSender.sol transfer instruction, users can specify the gas limit to be sent. Since the admin sponsors these CCIP transactions, a malicious user could set an excessively high gas limit, leading to a significant drain of ETH from the contract.

Recommendation

Set a maximum gas limit that users can specify to prevent excessive gas usage. Alternatively, modify the financial model to require users to pay for their own CCIP transaction fees, ensuring that any excessive gas costs are borne by the users and not the admin.

Status

Resolved

5. Inflation Attack in deposit Function

Path

contracts/v1/CrossChainVault.sol

Function

deposit

Description

The deposit function in the contract allows users to deposit tokens and receive shares representing their stake in the pool. The current implementation calculates shares based on the formula (_amount * totalShares) / pool when totalShares is not zero. This can lead to an inflation attack, where an attacker can manipulate the share calculation to their advantage. This attack can occur as follows:

1. The attacker deposits a small amount of tokens (e.g., 1 token) to create initial shares. The attacker then directly sends a large amount of staking tokens (e.g., 10e18 staking
2. tokens) to the contract without going through the deposit function (sending directly to contract using ERC20 transfer).
3. When a legitimate user subsequently deposits a substantial amount of tokens (e.g., 10e18 staking tokens), the calculation (10e18 stakingTokens * totalShares) / pool
4. results in zero shares due to solidity truncation: formula is (amount + totalShares / balance) - 10e18 * 1 / (10e18 + 1) = 0. Total amount of shares is going to be 1 (no
5. change).
   Attacker can run withdraw function and withdraw all tokens - 20e18 + 1 from the vault with formula: (balance * shares / totalShares) - 20e18 * 1 / 1 = 20e18.
   This way the attacker can withdraw 20e18 tokens and supply only 10e18 + 1 tokens.

Recommendation

To prevent this inflation attack take into account only tokens that were sent to contract with deposit instruction and do not account tokens send directly to contract. This can be achieved by creating additional variable like for example tokenBalance - and adjust value of variable with token withdrawals and deposits.

Status

Resolved

Low Severity Issues

6. Existence of Test Code in Contract Enables Potential Centralization and Censorship

Path

contracts/v1/CrossChainReceiver.sol

Function

setSimRevert

Description

The setSimRevert function allows the contract owner to toggle the simulation of a revert condition for testing purposes. This function can be exploited by the admin to block upcoming transfers and disallow users from depositing, thereby introducing centralization and enabling potential censorship of transactions. This capability undermines the decentralized nature of the contract and can lead to trust issues among users.

Recommendation

Remove the setSimRevert function and any related test code from the production contract. Ensure that testing and simulation are conducted in a separate environment to maintain the integrity and decentralization of the live contract.

Status

Resolved

Informational Issues

7. Risk of Improper Contract State Due to Manual Allowlist Configuration

Path

*

Function

*

Description

The contracts rely on strict access control mechanisms, where each contract has specific counterparties it should accept transactions from:

- CrossChainVault should accept transactions only from CrossChainReceiver.
- CCIPTokenSender should accept transactions only from CrossChainVault.
- CrossChainReceiver should accept transactions only from IRouterClient.
- CrossChainSender is available for all token holders
- CCIPTokenSender should accept transactions only from trusted addresses

However, the allowlists are managed via separate instructions from the contract owner. If the allowlist configurations are set incorrectly, the contracts could end up in an improper state, potentially leading to security vulnerabilities or operational failures. It is crucial to document the access control processes thoroughly and ensure that the allowlist configuration is managed carefully to prevent such issues.

Recommendation

Document the process and best practices for configuring and managing access control and allowlists comprehensively. Implement automated checks or safeguards to verify allowlist settings.

Status

Acknowledged

8. Function Can Be Set as External

Path

contracts/v1/CrossChainSender.sol, CrossChainVault.sol, CrossChainReceiver.sol
contracts/v0/CCIPTokenSender.sol

Function

setMinimumAmount, withdraw, withdrawToken

Description

Several functions are currently defined as a public function, which consumes more gas than necessary. Since these functions are only intended to be called by the contract owner, it can be set to external to save on gas costs.

Following functions are affected:

- CrossChainSender.sol: setMinimumAmount, withdraw, withdrawToken
- CrossChainReceiver.sol: withdrawETH, withdrawToken
- CrossChainVault.sol: withdrawETH, withdrawToken
- CCIPTokenSender.sol: withdrawETH, withdrawToken

Recommendation

Change the visibility of the setMinimumAmount function from public to external to reduce gas usage and improve the contract's efficiency.

Status

Resolved

9. Potential Gas Reversion in getFailedMessagesIds Function Due to Large Number of Entries

Path

contracts/v1/CrossChainReceiver.sol

Function

getFailedMessagesIds

Description

The getFailedMessagesIds function retrieves the IDs of failed messages by iterating over the entire s_failedMessages map. If the number of failed messages is large, this loop could consume excessive gas and potentially cause the transaction to revert due to reaching the gas limit.

Recommendation

Implement a pagination mechanism to retrieve the failed message IDs in smaller batches, preventing gas limit reversion. This change will ensure the function remains efficient and functional even with a large number of failed messages.

Status

Acknowledged

10. Lack of Zero Address Checks for Addresses Set in Constructors

Path

*

Function

constructor

Description

The constructors of all four audited contracts (CrossChainVault, CrossChainReceiver, CrossChainSender, and CCIPTokenSender) do not currently include checks to prevent zero addresses from being set. If any of these addresses are inadvertently set to the zero address, it could lead to the contracts being in an incorrect or unusable state, potentially causing operational issues and security vulnerabilities.

Recommendation

Add zero address checks in the constructors of all four contracts to ensure that no critical addresses are set to the zero address. This will help maintain the correct state and functionality of the contracts.

Status

Acknowledged

11. Unused Function _isContract in Contract Code

Path

CrossChainVault.sol

Function

_isContract

Description

The function _isContract, which checks whether a given address is a contract, is defined but not utilized anywhere within the contract code. This function is intended to prevent contracts from being targeted; however, since it is not called, it adds unnecessary complexity to the codebase without providing any functional benefit.

Recommendation

Remove the unused _isContract function from the contract code to simplify the implementation and enhance readability. If the function is intended for future use or testing, consider documenting its purpose and potential applications, or integrate it appropriately into the existing logic.

Status

Resolved

Functional Tests Cases

Initialization

- Confirm zero address checks during constructor execution.

Access Control

- Test onlyAllowlistedSender modifier to ensure only allowed addresses can call restricted
- functions.
- Verify that onlyOwner functions are accessible only to the contract owner.
- Ensure allowlist functions correctly add and remove addresses.
- Validate that CrossChainVault accepts transactions only from CrossChainReceiver.
- Confirm CCIPTokenSender accepts transactions only from CrossChainVault.
- Ensure CCIPReceiver accepts transactions only from the router.
- Validate that CrossChainSender is accessible to all token holders.

Deposit Functionality

- Verify successful deposits by users.
- Test share calculation with initial deposits.
- Ensure accurate share distribution with multiple deposits.
- Validate prevention of inflation attacks as per remediation.
- Test that deposits fail if the contract is paused.

Withdrawal Functionality

- Verify successful withdrawals by users.
- Ensure correct share reduction and token transfer.
- Test handling of withdrawal fees, including time-sensitive reductions.
- Confirm that unauthorized users cannot trigger withdrawals for others.

Functional Tests Cases

Cross-Chain Transfers

- Test initiation of cross-chain messages using CrossChainSender.
- Verify message receipt by CrossChainReceiver.
- Confirm interaction with CrossChainVault for token deposits.

Fee Management

- Confirm proper fee calculation and enforcement.
- Test handling of maximum gas limits for user-supplied transactions.

Administrative Functions

- Test updating of allowlists by the contract owner.
- Verify retrying of failed messages by the contract owner.
- Confirm toggling of simulation settings for testing purposes.

Compliance with CCIP Documentation

- Validate functionality against CCIP specifications and recommendations.
- Ensure proper handling of gas limits as per CCIP guidelines.
- Confirm adherence to security best practices outlined in CCIP documentation.

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 Landshare 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, Out of all the issues found in various severity, the Landshare Team resolved all high and low-severity issues, Resolved two informational severity issues and acknowledged three informational issues.

Disclaimer

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