QuillAudits

Audit Report
December, 2024

For

Table of Content

Executive Summary

Project Name	W Coin Migration
Project URL	*https://w-chain.com/*
Overview	WChainMigration contract facilitates secure asset migration between chains with multi-signature verification (2/3 signers), nonce-based replay protection, and robust tracking of claims. It incorporates a designated backend address for executing claims, manages ETH and ERC20 transfers safely, and prevents unauthorized actions. Features include signer management, claim tracking, and stringent validation for addresses and transactions.
Audit Scope	The scope of this Audit was to analyze the W Coin Migration Smart Contracts for quality, security, and correctness. 0x43cbB94a3B14C1D5e66104C835B6FF31c6595cAf
Contracts In Scope	WChainMigration.sol
Language	Solidity
Blockchain	W Chain
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	25th December 2024 - 27th December 2024
Updated Code Received	30th December 2024
Review 2	2nd January 2024
Fixed In	WChainMigration sepolia 0xC9c7c4065ed6f40D65AF4370f7bF019eC225b43F WChain Testnet 0xe746CD88Cd41673Df0e60f7b5DDB71Ee8A77e316

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	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
- Unsafe type inference
- Style guide violation
- 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.

Medium Severity Issues

1. Risk of DOS due to signer array becoming very large

Path

WChainMigration.sol

Function name

addSigner(), removeSigner()

Description

The **addSigner** function contains a potential Denial of Service (DOS) vulnerability due to an unbounded array of signers. The function performs a linear search through the **_signers** array to check for duplicates before adding a new signer. As the number of signers grows, the gas cost of this operation increases linearly, potentially making the function unusable if the array becomes too large.

The issue is present in two locations:

1. The **addSigner** function's duplicate check loop
2. The **removeSigner** function's search loop

This could lead to:

- Excessive gas consumption for signer management operations
- Function calls potentially exceeding the block gas limit
- Contract becoming unusable if too many signers are added

Recommendation

Consider using a more gas-efficient data structure:

- Replace the linear search with direct mapping access
- Use EnumerableSet from OpenZeppelin for efficient membership checks and enumeration

Status

Resolved

Low Severity Issues

2. Use call instead of transfer

Path

WChainMigration.sol

Function name

claim()

Description

The **claim** function uses the deprecated **transfer()** method to send ETH to users. The **transfer()** function forwards a fixed gas stipend of 2300 gas, which can cause the transaction to fail if the recipient is a smart contract with complex logic in its receive/fallback functions.

Recommendation

Replace **transfer()** with a **call()**.

Status

Resolved

3. CEI pattern is not followed

Path

WChainMigration.sol

Function name

claim()

Description

The **claim** function violates the Checks-Effects-Interactions (CEI) pattern by performing an external call (**transfer**) before updating the contract's state variables. This creates a potential reentrancy vulnerability where an attacker could re-enter the contract before state changes are applied.

Recommendation

Reorganize the code to follow the CEI pattern.

Status

Resolved

4. Use abi.encode instead of abi.encodePacked

Path

WChainMigration.sol

Description

abi.encodePacked() should not be used with dynamic types when passing the result to a hash function such as **keccak256()**.

Use **abi.encode()** instead which will pad items to 32 bytes, which will **prevent hash collisions** (e.g. **abi.encodePacked(0x123,0x456)** => **0x123456** => **abi.encodePacked(0x1,0x23456)**, but **abi.encode(0x123,0x456)** => **0x0...1230...456**).

Unless there is a compelling reason, **abi.encode** should be preferred. If there is only one argument to **abi.encodePacked()** it can often be cast to **bytes()** or **bytes32()** **instead**. If all arguments are strings and or bytes, **bytes.concat()** should be used instead.

Status

Acknowledged

W Chain Team's Comment

We didn't update **abi.encodePacked** because it uses the same parameters as **uint256**. So the result of **abi.encodePacked** is the same as **abi.encode** and therefore doesn't cause any security issues.

Informational Issues

5. Immutable chainId can be problematic in case of hard fork

Path

WChainMigration.sol

Description

Using an immutable chainId in the contract poses a risk during network hard forks. When a network undergoes a hard fork, it can result in two separate chains with different chainIds. If the contract's chainId is immutable (hardcoded), the contract will continue to operate with the original chainId.

Recommendation

Make chainId updateable by governance.

Status

Resolved

Functional Tests

Some of the tests performed are mentioned below:

- signature verification system with multi-signature implementation correctly enforces 2/3 threshold verification, validates signatures via ecrecover, prevents duplicate signers, and employs a strong nonce mechanism to mitigate replay attacks
- Claim a contract doesn't verify if it has enough ETH balance before the transfer

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 W Coin Migration. We performed our audit according to the procedure described above.

Some issues of medium, low 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. In the end, W Coin Migration Team, almost resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in W Coin Migration. 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 W Coin Migration. 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 W Coin Migration 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
December, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com