QuillAudits

Audit Report
December, 2024

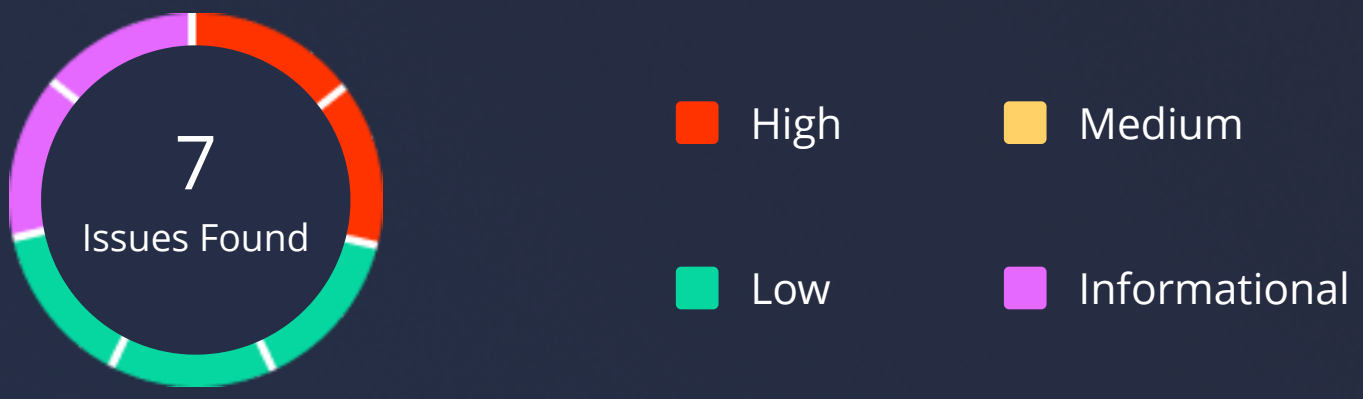
For

Table of Content

Executive Summary

Project Name	W Chain Bridge
Project URL	*https://w-chain.com/*
Overview	W Chain Bridge is the native bridge of a new L1 chain that currently bridges their native token. Users can bridge their tokens and claim with the signatures of the required signers ratio of the protocol.
Audit Scope	The scope of this Audit was to analyze the W Chain Bridge Smart Contracts for quality, security, and correctness.
Contracts In Scope	0xbE45De95AC59AC879526B806CC095348a9F75647
Commit Hash	NA
Language	Solidity
Blockchain	W Chain
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	13th December 2024 - 19th December 2024
Updated Code Received	23rd December 2024
Review 2	24th December 2024
Fixed In	WChainBridge sepolia 0x4093934863e1C7dA6D6F4Fff60277Ae206263AD2 WChain Testnet 0xDA7489e962F982D0f00d5201900B315C5d01AF52

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	0	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
- Style guide violation
- 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 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.

High Severity Issues

1. Signer security can be less than expected

Path

WadzBridge.sol

Function name

claim()

Description

The claim function checks that the length of the _v array is greater than two-thirds the signers present. With 4 signers it is expected that _v is (⅔ * 4) which yields 2.667 but Solidity rounds down from 2.667 to 2 thereby reducing the effectiveness of the check for signers by 1 signer.

Recommendation

Consider wrapping up instead of down to avoid diluting the security checks put in place, i.e. two-thirds of the signers registered must sign.

Status

Resolved

2. Users can invoke the claim function with same signature in the v,r,s array to bypass the required signers ratio

Path

WadzBridge.sol

Function name

claim()

Description

When claiming, the valid_count variable is incremented if the signer is true in the is_signer mapping. Users can exploit this by providing an array of v, r, and s that is populated with a single signer address vrs. Due to the absence of a check to validate that a signer signature has been checked before, the user can repeatedly use one signer address, populate the array to meet the signers ratio, and then claim their funds without appropriate checks. If

this happens, valid_count can be incremented any number of times till it exceeds the two-thirds of _signers security limit allowing the msg.sender to receive the amount.

POC

```
function testSignatureClaimWithRepeatedVRS() external {
    // add signers
    (address signer1, uint256 key1) = makeAddrAndKey("SIGNER-1");
    (address signer2, uint256 key2) = makeAddrAndKey("SIGNER-2");
    (address signer3, uint256 key3) = makeAddrAndKey("SIGNER-3");

    // owner adds all three signers
    mainContract.addSigner(signer1);
    mainContract.addSigner(signer2);
    mainContract.addSigner(signer3);

    // fund users
    address user1 = makeAddr("USER1");
    address user2 = makeAddr("USER2");
    vm.deal(user1, 100 ether);
    vm.deal(user2, 100 ether);

    //deposit to bridge
    vm.startPrank(user1);
    mainContract.bridge{value: 30 ether}(30 ether);
    vm.stopPrank();
    assertEq(address(mainContract).balance, 30 ether);

    // hash message and sign
    vm.startPrank(signer1);
    bytes32 _hashedData = keccak256(abi.encode(1, user1, 30 ether, 1));
    bytes memory prefix = "\x19Ethereum Signed Message:\n32";
    bytes32 prefixedHashMessage = keccak256(abi.encodePacked(prefix, _hashedData));
    (uint8 _v, bytes32 _r, bytes32 _s) = vm.sign(key1, prefixedHashMessage);
    address signedSigner = ecrecover(prefixedHashMessage, _v, _r, _s);
    assertEq(signer1, signedSigner);
    vm.stopPrank();

    // start claiming for user1 with populated v,r,s arrays with a single signer vrs.
    uint8[] memory _vs = new uint8[](2);
    _vs[0] = _v;
    _vs[1] = _v;

    bytes32[] memory _rs = new bytes32[](2);
    _rs[0] = _r;
    _rs[1] = _r;

    bytes32[] memory _ss = new bytes32[](2);
    _ss[0] = _s;
    _ss[1] = _s;

    vm.startPrank(user1);
    mainContract.claim(1, 30 ether, 1, _vs, _rs, _ss);
    vm.stopPrank();

    assertEq(address(mainContract).balance, 0 ether);
}
```

Recommendation

Include a check for addresses and signatures to avoid successful claims with less signers ratio.

Status

Resolved

Auditor's Question

The bridge function was redesigned to check for valid signers count before the deposit. This implies that for a user to invoke the bridge function, the signers must have signed a message to allow this user to deposit the signed amount in the message. How do you intend to go about this for prospective users?

Wadzchain Team's Comment

That's the whole point of the bridge. It has multiple signers for security so it needs to check that those signers are valid. It's more security conscious to check before a user makes a deposit that the signers are valid obviously. Users should call the bridge function only using the portal app. If not, this should be considered abnormal behavior.

Low Severity Issues

3. Contract ownership can transferred to the wrong owner

Path

WadzBridge.sol

Function name

transferOwnership()

Description

Access control is a critical part of smart contract security. Within this contract, access control is handled by the Ownable library which allows the owner to relinquish their rights to a new owner in one function call, **transferOwnership()**. This is risky if the new owner address passed in is malicious or is a victim of an address poisoning attack where some bytes of the address are changed when copied to the clipboard.

Recommendation

Implement two-step ownership that includes confirmation of ownership transfer from the new owner before it is accepted.

Status

Resolved

4. EnumerableSet.AddressSet is more efficient than looping over arrays

Path

WadzBridge.sol

Function name

addSigner(), removeSigner()

Description

For adding and removing signers in WadzBridge, a for loop is used to add or remove addresses from the array - realistically, the array wouldn't get large enough to be prohibitively expensive. In the current implementation, an array (**_signers**) and a mapping (**is_signer**) are used to determine the validity of addresses but the EnumerableSet provides a more efficient route to manipulate addresses in the same format.

Recommendation

Use the Openzeppelin EnumerableSet library.

Status

Acknowledged

5. Missing check for chains

Path

WadzBridge.sol

Function name

bridge(), claim()

Description

There currently is no check in the bridge and claim function that the chainId passed is not the parent chain. With the contract storing the claim data attached to the various chains, a user could pass in the same chainId for the _from_chain and _to_chain causing accounting errors.

Recommendation

Include a sanity check that the chains are not the same.

Status

Resolved

Informational Issues

6. Use newer libraries

Path

WadzBridge.sol

Function name

Ownable.constructor()

Description

In the newer openzeppelin-contracts implementation (>5.0), the Ownable contract expects to pass the owner address in the constructor, but this doesn't occur in the WadzBridge.sol contract currently and will revert when compiling from imports of the latest OpenZeppelin contracts.

Recommendation

Use the latest libraries to ensure adequate security patches and updates have been made.

Status

Acknowledged

7. Adopt the use of Event emission to ease querying data from the contract

Path

WadzBridge.sol

Description

There are a couple of getter functions present in the contract which implies that the bridge relies on constantly querying data from the contract. However, blockchain bridges over time, relies on event emission and has a fast way to query data; it is cheaper and quicker to fetch on the client side. Emitting events for critical state changes is an appropriate way to track important transactions performed from the contract.

Recommendation

Use events in the contract for speedy data querying.

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 W Chain Bridge. We performed our audit according to the procedure described above.

Some issues of High, 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.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in W Chain Bridge. 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 Chain Bridge. 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 Chain Bridge 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