CONTRACT WOLF

Security Assessment

Roburna Bridge

Verified on 01/09/2026

SUMMARY

Project	CHAIN	METHODOLOGY
Roburna Bridge	Roburna Chain	Manual & Automatic Analysis

FILES	DELIVERY	TYPE
Single	01/09/2026	Standard Audit

7	1	0	4	2	0	7
Total Findings	Critical	Major	Medium	Minor	Informational	Resolved

🟥 1 Critical	An exposure that can affect the contract functions in several events that can risk and disrupt the contract
🟧 0 Major	An opening & exposure to manipulate the contract in an unwanted manner
🟧 4 Medium	An opening that could affect the outcome in executing the contract in a specific situation
⬜ 2 Minor	An opening but doesn't have an impact on the functionality of the contract
🟦 0 Informational	An opening that consists information but will not risk or affect the contract
🟩 7 Resolved	ContractWolf's findings has been acknowledged & resolved by the project

STATUS ✔️**AUDIT PASSED**

TABLE OF CONTENTS | Roburna Bridge

DISCLAIMER | Roburna Bridge

ContractWolf audits and reports should not be considered as a form of project's "Advertisement" and does not cover any interaction and assessment from "Project Contract" to "External Contracts" such as PancakeSwap, UniSwap, SushiSwap or similar.

ContractWolf does not provide any warranty on its released report and should not be used as a decision to invest into audited projects.

ContractWolf provides a transparent report to all its "Clients" and to its "Clients Participants" and will not claim any guarantee of bug-free code within its **SMART CONTRACT**.

ContractWolf's presence is to analyze, audit and assess the Client's Smart Contract to find any underlying risk and to eliminate any logic and flow errors within its code.

Each company or project should be liable to its security flaws and functionalities.

SCOPE OF WORK | Roburna Bridge

Roburna Bridge team has agreed and provided us with the files that need to be tested (*Github, BSCscan, Etherscan, Local files etc*). The scope of audit is the main contract.

The goal of this engagement is to identify if there is a possibility of security flaws in the implementation of smart contract and its systems.

ContractWolf will be focusing on contract issues and functionalities along with the project claims from smart contract to their website, whitepaper, repository which has been provided by **Roburna Bridge**.

AUDITING APPROACH | Roburna Bridge

Every line of code along with its functionalities will undergo manual review to check for security issues, quality of logic and contract scope of inheritance. The manual review will be done by our team that will document any issues that they discovered.

METHODOLOGY

The auditing process follows a routine series of steps :

1. Code review that includes the following :
   - Review of the specifications, sources and instructions provided to ContractWolf to make sure we understand the size, scope and functionality of the smart contract.
   - Manual review of code. Our team will have a process of reading the code line-by-line with the intention of identifying potential vulnerabilities, underlying and hidden security flaws.

2. Testing and automated analysis that includes :
   - Testing the smart contract function with common test cases and scenarios to ensure that it returns the expected results.

3. Best practices and ethical review. The team will review the contract with the aim to improve efficiency, effectiveness, clarifications, maintainability, security and control within the smart contract.

4. Recommendations to help the project take steps to eliminate or minimize threats and secure the smart contract.

TOKEN DETAILS | Roburna Bridge

-

Token Name	Symbol	Decimal	Total Supply	Chain
-	-	-	-	Roburna Chain

SOURCE

Source	*Github.com or 0x0000, Sent Via local-files*

FINDINGS | Roburna Bridge

7	1	0	4	2	0	7
Total Findings	Critical	Major	Medium	Minor	Informational	Resolved

This report has been prepared to state the issues and vulnerabilities for Roburna Bridge through this audit. The goal of this report findings is to identify specifically and fix any underlying issues and errors

ID	Title	File & Line #	Severity	Status
	Cross Chain Replay Attack	BridgeCore	Critical	• Resolved
	Inconsistent State in Batch Refund	BridgeCore	Medium	• Resolved
SWC-101	Integer Overflow/Underflow	BridgeCore	Medium	• Resolved
SWC-104	Unchecked Call Return Value	BridgeCore	Medium	• Resolved
SWC-101	Integer Overflow/Underflow	Fee Manager	Medium	• Resolved
	Remove Validator	Fee Manager	Minor	• Resolved
	Unbound supportedTokens Array	Vault	Minor	• Resolved

SWC ATTACKS | Roburna Bridge

Smart Contract Weakness Classification and Test Cases

ID	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELF DESTRUCT Instruction	Passed
SWC-107	Reentrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegate call to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed
SWC-116	Block values as a proxy for time	Passed
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed

ID	Description	Status
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character(U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected Ether balance	Passed
SWC-133	Hash Collisions With Multiple Variable Arguments	Passed
SWC-134	Message call with hardcoded gas amount	Passed
SWC-135	Code With No Effects	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed

CW ASSESSMENT | Roburna Bridge

ContractWolf Vulnerability and Security Tests

ID	Name	Description	Status
CW-001	Multiple Version	Presence of multiple compiler version across all contracts	✔
CW-002	Incorrect Access Control	Additional checks for critical logic and flow	✔
CW-003	Payable Contract	A function to withdraw ether should exist otherwise the ether will be trapped	✔
CW-004	Custom Modifier	major recheck for custom modifier logic	✔
CW-005	Divide Before Multiply	Performing multiplication before division is generally better to avoid loss of precision	✔
CW-006	Multiple Calls	Functions with multiple internal calls	✔
CW-007	Deprecated Keywords	Use of deprecated functions/operators such as block.blockhash() for blockhash(), msg.gas for gasleft(), throw for revert(), sha3() for keccak256(), callcode() for delegatecall(), suicide() for selfdestruct(), constant for view or var for actual type name should be avoided to prevent unintended errors with newer compiler versions	✔
CW-008	Unused Contract	Presence of an unused, unimported or uncalled contract	✔
CW-009	Assembly Usage	Use of EVM assembly is error-prone and should be avoided or double-checked for correctness	✔
CW-010	Similar Variable Names	Variables with similar names could be confused for each other and therefore should be avoided	✔
CW-011	Commented Code	Removal of commented/unused code lines	✔
CW-012	SafeMath Override	SafeMath is no longer needed starting with Solidity v0.8+. The compiler now has built-in overflow checking.	✔

FIXES & RECOMMENDATION

Cross-chain Replay Attack

From : Bridge Core

The function uses a deposit's timestamp and a global **SIGNATURE_VALIDITY_PERIOD** to validate a cross-chain message. This is insufficient, as the depositId is not guaranteed to be globally unique across chains. An attacker could replay a validated message from one chain or another.

Could lead to double-minting of tokens on multiple chains, draining the protocol.

Recommendation

Incorporate the source chain ID into the depositId generation and validation to ensure chain-specific uniqueness.

```
// In deposit function, update depositId generation:
bytes32 depositId = keccak256(abi.encodePacked(
    msg.sender,
    token,
    amount,
    block.chainid, // Source Chain ID
    targetChainId, // Explicitly include target chain
    nonce++,
    block.timestamp
));

// In 'executeReleaseFromValidator', add a check:
require(sourceChainId != block.chainid, "BridgeCore: invalid source chain");
// The depositId from the event already encodes sourceChainId, making it unique.
```

SWC-101 | Integer Overflow/Underflow

From : Bridge Core

The code performs **`uint256 amountAfterFee = usdcAmount - fee`**; without verifying that **`usdcAmount > fee`**. If the fee manager calculates a fee larger than the amount, it will cause an underflow, reverting the transaction and locking funds.

It also could permanently brick the release of funds for specific deposits.

Recommendation

Add a safety check before subtraction.

```
// In 'executeReleaseFromValidator', before calculating amountAfterFee:
require(usdcAmount > fee, "BridgeCore: fee exceeds amount");
uint256 amountAfterFee = usdcAmount - fee;
```

Inconsistent State in Batch Refund

From : Bridge Core

The function sets both **`depositInfo.refunded = true`** and **`processedDeposits[depositId] = true`**. However, **`processedDeposits`** is meant to indicate a successful cross-chain release, not a refund. This conflates two different states and could cause confusion in off-chain monitoring.

Recommendation

```
// In the refund logic, change to:
depositInfo.refunded = true;
// DO NOT set processedDeposits[depositId] = true;
```

SWC-104 | Unchecked Call Return Value

From : Bridge Core

The try/catch blocks only catch Error(string memory reason). They do not catch low-level reverts without a reason (e.g. **`assert()`** failures, out-of-gas errors). These would bypass the catch block and cause the entire deposit to revert.

Could make the bridge unreliable if the swap adapter has an assert-style failure.

Recommendation

Use a low-level call or catch all revert types.

```
// Example of catching all reverts (current version of bridgecore : 8.19)
try swapAdapter.swapExactNativeForTokens{value: amount}(//inputs) returns (uint256[]
memory amounts) {
    // success
} catch (bytes memory /*lowLevelData*/) {
    // catches EVERYTHING (revert, assert, panic)
    revert("BridgeCore: swap failed");
}
```

SWC-101 | Integer Overflow/Underflow

From : Fee Manager

The formula **`(amount * feeRate) / 10000`** performs division after multiplication, which discards the remainder. For small amount values or low feeRate values, this can result in a calculated fee of zero, even when a fee should apply.

Recommendation

Implement "round up" logic for the protocol's benefit using **`Math.ceilDiv`** or a custom function to ensure minimum fee collection.

```
// Add import at the top of the contract
import "@openzeppelin/contracts/utils/math/Math.sol";

// Modify the percentage fee calculation
function calculateFee(uint256 amount) external view returns (uint256) {
    if (isPercentage) {
        // Ceiling division to round up, ensuring at least 1 wei fee for non-zero
amounts when feeRate > 0
        return Math.mulDiv(amount, feeRate, 10000, Math.Rounding.Ceil);
    } else {
        return feeRate;
    }
}
```

Remove Validator

From : Validator Manager

The Code

```
require(validatorCount > threshold, "ValidatorManager: cannot remove validator below threshold");
```

prevents removing validators when **`validatorCount == threshold`**. However, if **`threshold == validatorCount`**, no validator can ever be removed, creating a permanent deadlock.

Recommendation

Allow removal with threshold adjustment, or implement emergency mechanisms.

```
function removeValidator(address validator) external onlyAdmin {
    require(validators[validator], "ValidatorManager: validator does not exist");

    // Calculate new count
    uint256 newCount = validatorCount - 1;

    // If removing would make count < threshold, lower threshold first
    if (newCount < threshold) {
        // Option 1: Automatically lower threshold
        threshold = newCount;
        emit ThresholdUpdated(threshold + 1, newCount);

        // Option 2: Require explicit threshold update first
        // revert("ValidatorManager: threshold must be lowered first");
    }

    validators[validator] = false;
    validatorCount--;

    // Remove from array (existing code)
    for (uint256 i = 0; i < validatorList.length; i++) {
        if (validatorList[i] == validator) {
            validatorList[i] = validatorList[validatorList.length - 1];
            validatorList.pop();
            break;
        }
    }

    emit ValidatorRemoved(validator);
}
```

Unbounded supportedTokens Array

From : Vault

The supportedTokens array grows indefinitely. When **`getSupportedTokens()`** is called, it returns the entire array, which could exceed gas limits with many tokens.

Recommendation

Implement pagination or remove the array and use events/logs for tracking.

```
// Add pagination
function getSupportedTokensPaginated(uint256 start, uint256 count) external view
returns (address[] memory) {
    require(start < supportedTokens.length, "Vault: start out of bounds");

    uint256 end = start + count;
    if (end > supportedTokens.length) {
        end = supportedTokens.length;
    }

    address[] memory result = new address[](end - start);
    for (uint256 i = start; i < end; i++) {
        result[i - start] = supportedTokens[i];
    }
    return result;
}
```

CONTRACTWOLF

Blockchain Security - Smart Contract Audits