MixBytes()

Threshold Network tBTC v2 Security Audit Report

SEPTEMBER 08, 2025

Table of Contents

1. Introduction

1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

tBTC v2 is a cross-chain bridge that lets users burn canonical tBTC on supported L2s and side-chains, send a Wormhole message containing their Bitcoin output script, and receive the equivalent amount of BTC on the Bitcoin network via the Threshold L1 Bridge.
The reverse flow - depositing BTC on mainnet to mint canonical tBTC on those L2s - is also supported by the protocol, but this path was outside the scope of the present audit.

The codebase was audited over 3 days by 3 auditors through a combination of manual review and automated tooling.

Throughout the audit, besides checking for common attack vectors and items from our internal checklist, we conducted an in-depth examination of the following areas:

- **Token Amount Discrepancy in Cross-Chain Bridge Transfers**
  During cross-chain bridge transfers, it often happens that the amount of tokens arriving on the recipient side differs—typically being lower—from the amount sent. There are also cases where the bridge itself deducts (truncates) a portion of the tokens being transferred. We have verified that in this project, this behavior will not lead to any loss of user funds or disruption of the system's logical operations.

- **Gas Reimbursement Integrity.**
  The project includes functionality for incentivizing transactions based on the current gas price. We have confirmed that the algorithm in use does not allow, whether accidentally or through malicious manipulation, for obtaining an unreasonably high reward.

- **Unauthorized L1 Mint / Unlock without L2 Burn.**
  The intended flow requires a user to burn or lock tBTC on the L2 chain, obtain a guardian-signed VAA, and then redeem it on L1. We examined the redemption path in L1BTCRedeemerWormhole and related contracts: the first step is always wormholeTokenBridge.completeTransferWithPayload, which internally validates the guardian signatures and ensures the VAA represents an actual burn on the source chain. Without a valid, unique VAA the call reverts, so an attacker cannot bypass the L2 burn step to mint or unlock tokens on L1.

- **Double-Claim of Bridged Tokens.**
  We confirmed that re-submitting the same VAA is rejected by Wormhole, which stores the message-body hash of every redeemed VAA. A second redemption attempt with the same hash is

therefore impossible. None of the project's contracts bypass this safeguard—every redemption call ultimately goes through completeTransferWithPayload, so once a VAA is consumed it cannot be reused to obtain additional tokens.

- **Access Controls on Critical Parameter Changes.**
  We inspected every function that alters governance-sensitive parameters—such as minting limits, gateway addresses, redemption thresholds, gas offsets, and reimbursement authorisations. Each of them is protected by an explicit access-control modifier. There are no alternate code paths that bypass these checks, so regular users cannot invoke these functions or modify protocol-critical settings.

- **Correct tBTC Accounting in Gateway Transfers.**
  For every outbound bridge call (sendTbtc and sendTbtcWithPayloadToNativeChain) we confirmed that the gateway burns the specified amount of canonical tBTC from the caller and decrements mintedAmount by the same value before forwarding Wormhole tokens.

Some external contracts and libraries that the audited contracts depend on were outside the scope of this review. Notably, this includes:
- wormholeTokenBridge (IWormholeTokenBridge.sol)
- bank (IBank.sol)
- thresholdBridge (IBridge.sol)
- tbtcVault (ITBTCVault.sol)
- reimbursementPool (ReimbursementPool.sol)
- BTCUtils.sol
- Wormhole.sol

The code is well-written, clearly commented, and our review uncovered no critical vulnerabilities. However, we identified several areas where minor refinements could improve robustness, readability, and maintainability. These suggestions are detailed in the **Findings Report** below.

1.3 Project Overview

Summary

Title	Description
Client Name	Threshold Network
Project Name	tBTC v2
Type	Solidity
Platform	EVM
Timeline	03.07.2025 - 05.09.2025

Scope of Audit

File	Link
solidity/contracts/integrator/ AbstractBTCRedeemer.sol	AbstractBTCRedeemer.sol
solidity/contracts/cross-chain/wormhole/ L2WormholeGateway.sol	L2WormholeGateway.sol
solidity/contracts/cross-chain/wormhole/ L2BTCRedeemerWormhole.sol	L2BTCRedeemerWormhole.sol
solidity/contracts/cross-chain/wormhole/ L1BTCRedeemerWormhole.sol	L1BTCRedeemerWormhole.sol

Versions Log

Date	Commit Hash	Note
03.07.2025	f709f4772d467d12ee611e8996d68d72a5c38ada	Initial commit
07.07.2025	166bdafe5013773a361a0e53f78f6477b2e48612	Updated commit
04.09.2025	042d7f19de663ad659854fc1bdb78bf8ccd64d77	Commit for the re-audit

Mainnet Deployments

File	Address	Blockchain
TransparentUpgradeableProxy.sol	0x5D4d83...3823DbDe	Ethereum
L1BTCRedeemerWormhole.sol	0x14D93D...D8eB9DC0	Ethereum
TransparentUpgradeableProxy.sol	0xd7Cd99...0334d9b7	Arbitrum
L2BTCRedeemerWormhole.sol	0x03e342...424ee0f6	Arbitrum
TransparentUpgradeableProxy.sol	0xe931F1...64d2D88B	Base
L2BTCRedeemerWormhole.sol	0x7926eb...8C7AEA2E	Base

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	1
Medium	0
Low	9

Findings Statuses

ID	Finding	Severity	Status
H-1	`requestRedemption()` Reverts Because `L1BTCRedeemerWormhole's` Bank Balance Is Not Credited	High	Fixed
L-1	Unused `Crosschain` Import in `L1BTCRedeemerWormhole` Contract	Low	Fixed
L-2	Deprecated `safeApprove()` Function Usage	Low	Acknowledged
L-3	L1 Redeemer Rescue Inoperable for tBTC Stuck on Wormhole Bridge	Low	Acknowledged
L-4	Missing Wormhole Peer Validation	Low	Fixed
L-5	Non-Descriptive Underflow Revert on `mintedAmount` Subtraction	Low	Fixed
L-6	Typos in comments	Low	Fixed
L-7	Unused `gasReimbursements` Mapping	Low	Fixed
L-8	String Revert Messages Increase Gas Consumption	Low	Fixed
L-9	`rescueTbtc` Function Lacks Event Emission	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

H-1	requestRedemption() Reverts Because L1BTCRedeemerWormhole's Bank Balance Is Not Credited		
Severity	High	**Status**	Fixed in 166bdafe

Description

L1BTCRedeemerWormhole.requestRedemption() grants thresholdBridge an allowance to withdraw the satoshi-denominated balance, but it never credits that balance to the Bank contract. When Bridge.requestRedemption() later calls bank.transferBalanceFrom(), the call reverts because L1BTCRedeemerWormhole has a zero balance. As a result every redemption request sent through the Wormhole flow becomes stuck: the VAA remains un-redeemed, tBTC stays locked in the Wormhole bridge contract, and the redemption path is effectively disabled.
This issue is classified as **High** severity because it completely blocks the Wormhole redemption functionality and leaves users unable to recover their bridged tBTC until the deployed code is upgraded.

Recommendation

We recommend increasing the L1BTCRedeemerWormhole balance in the Bank contract before calling thresholdBridge.requestRedemption().

Client's Commentary

MixBytes: The issue was identified during the audit, but the client had already fixed it independently prior to our reporting, without notifying us about the fix.

2.3 Medium

Not Found

2.4 Low

L-1	Unused Crosschain Import in L1BTCRedeemerWormhole Contract		
Severity	Low	**Status**	Fixed in 042d7f19

Description

The L1BTCRedeemerWormhole contract imports the Crosschain file:

```
import "../utils/Crosschain.sol";
```

Nothing from Crosschain.sol is referenced in the contract, which leads to longer compilation times, larger bytecode, and an unnecessary dependency that complicates maintenance.

Recommendation

We recommend removing the unused import.

Client's Commentary

Addressed in the commit: PR-882

L-2	Deprecated `safeApprove()` Function Usage		
Severity	Low	**Status**	Acknowledged

Description

The following functions use the `safeApprove()` method, which is marked as deprecated in OpenZeppelin's `SafeERC20` library:

- `L2WormholeGateway.sendTbtc()`
- `L2WormholeGateway.sendTbtcWithPayloadToNativeChain()`
- `AbstractBTCRedeemer._requestRedemption()`

Although the current `safeApprove()` calls work as expected, it is still preferable to avoid using deprecated functions. Replacing them with `forceApprove()` helps prevent potential issues during future codebase upgrades.

Recommendation

We recommend upgrading to the latest version of OpenZeppelin Contracts and replacing all instances of `safeApprove()` with `forceApprove()`.

Client's Commentary

We acknowledge the finding regarding safeApprove usage. However, we have determined that: 1. We are using OpenZeppelin v4.8.1, where safeApprove is NOT deprecated and functions correctly. 2. Our implementation follows the secure pattern of resetting approval to 0 before setting new values. 3. Upgrading to OpenZeppelin v5+ would require extensive changes across our entire codebase, introducing significantly more risk than the current implementation. 4. The current code has been battle-tested and functions correctly with no security vulnerabilities. We will consider using forceApprove in future greenfield projects that start with OpenZeppelin v5+, but for this established codebase, we will maintain the current secure implementation.

L-3	L1 Redeemer Rescue Inoperable for tBTC Stuck on Wormhole Bridge		
Severity	Low	**Status**	Acknowledged

Description

In the current implementation, the `L1BTCRedeemerWormhole` contract provides a built-in procedure for rescuing tBTC that remains **stuck** after a failed redemption on L1. However, when a cross-chain redemption is initiated from an L2 Redeemer via Wormhole, any permanent revert during on-L1 validation (e.g., due to malformed or invalid payload) causes the incoming tBTC to become locked in the Wormhole bridge contract rather than in the L1 Redeemer itself.

Since the L1 rescue logic only applies to tokens trapped in the L1 Redeemer, these funds cannot be recovered without performing a full contract upgrade.

Recommendation

We recommend implementing an emergency `pull-through` rescue function in the `L1BTCRedeemerWormhole` contract (or an associated recovery helper contract) that:

1. Allows the contract owner or a designated rescue agent to read a list of stuck redemption keys or VAAs currently held by Wormhole.
2. Invokes `wormholeTokenBridge.completeTransferWithPayload` (or a custom bridge interaction) to claim those VAAs/tokens and forward them into the L1 bridge's `requestRedemption` flow or directly to a configurable recovery address.
3. Includes appropriate access controls and events to ensure transparency and prevent misuse.

Client's Commentary

We have acknowledged the issue and added it to our internal roadmap: https://github.com/threshold-network/tbtc-v2/issues/883 - Implementing a robust, owner-gated rescue path touches multiple cross-chain components and will benefit from a design review that we are already scheduling. Because the current production flow is not affected, we do not consider an emergency patch necessary at this time.

L-4	Missing Wormhole Peer Validation		
Severity	Low	**Status**	Fixed in 042d7f19

Description

The message-processing functions for incoming Wormhole VAAs do not verify that the message was emitted by a trusted peer on the source chain. Specifically, the `L1BTCRedeemerWormhole` contract relies on `wormholeTokenBridge.completeTransferWithPayload()` and then parses the payload via `parseTransferWithPayload()`, but never checks the **`fromAddress`** or **`tokenChain`** fields against an allowlist. While this omission does not introduce an observable issue in the current code, it creates a latent risk: a future change in the code could allow a malicious actor to submit a spoofed VAA and alter the redemption flow.

Recommendation

We recommend introducing an on-chain whitelist of trusted Wormhole peers and enforce validation of both the **`fromAddress`** and **`tokenChain`** identifier immediately after parsing the VAA. Reject any VAAs whose origin does not match an entry in the whitelist.

Client's Commentary

Addressed in the commit: PR-882

L-5	Non-Descriptive Underflow Revert on mintedAmount Subtraction		
Severity	Low	**Status**	Fixed in 042d7f19

Description

The L2WormholeGateway contract subtracts amount from mintedAmount without first verifying that mintedAmount >= amount. If mintedAmount is ever less than amount (e.g., if tokens were minted by another bridge), the subtraction will underflow and trigger a generic arithmetic underflow revert.

Such non-descriptive errors make it difficult to diagnose issues and understand the reason for the transaction failure.

This issue exists in the following functions:

- L2WormholeGateway.sendTbtc()
- L2WormholeGateway.sendTbtcWithPayloadToNativeChain()

Recommendation

We recommend adding an explicit check with a clear and descriptive error message before performing the subtraction.

Client's Commentary

Addressed in the commit: PR-882

L-6	Typos in comments		
Severity	Low	**Status**	Fixed in 042d7f19

Description

Several typos and formatting issues were found in inline comments and documentation across the codebase:

1. Incorrect verb usage in a comment for the `gap` storage variable:

```
/// added in the upcoming versions one **need**
/// to reduce the array size accordingly.
```

Should be:

```
/// added in the upcoming versions one **needs**
/// to reduce the array size accordingly.
```

- AbstractBTCRedeemer.sol#L91

2. Extra punctuation in the documentation of `L2BTCRedeemerWormhole`:

```
/// - A unique identifier for the transaction..
```

Should be:

```
/// - A unique identifier for the transaction.
```

- L2BTCRedeemerWormhole.sol#L41

3. Redundant word in a parameter comment:

```
/// @param _requestRedemptionGasOffset New **initialize** redemption gas offset.
```

Should be:

```
/// @param _requestRedemptionGasOffset New redemption gas offset.
```

- L1BTCRedeemerWormhole.sol#L137

Recommendation

We recommend updating the comments to improve clarity.

Client's Commentary

Addressed in the commit: PR-882

L-7	Unused gasReimbursements Mapping		
Severity	Low	**Status**	Fixed in 042d7f19

Description

L1BTCRedeemerWormhole declares a public gasReimbursements mapping that is never written to or read from. Because each mapping occupies a dedicated storage slot, this increases contract size and deployment cost without providing any functionality. Unused state variables also confuse future maintainers and may hide logic that was left unfinished.

Recommendation

We recommend removing the mapping entirely or implementing the reimbursement-tracking logic.

Client's Commentary

Addressed in the commit: PR-882

L-8	String Revert Messages Increase Gas Consumption		
Severity	Low	**Status**	Fixed in 042d7f19

Description

The project uses `require` statements with literal string messages. Storing these strings in contract bytecode and returning them on failure increases both deployment size and runtime gas. Solidity's custom errors (introduced in 0.8.4) encode only a selector plus arguments, reducing gas per failing call.

Recommendation

We recommend defining custom errors and replacing statements of the form:

```
require(condition, "error description")
```

with:

```
if (!condition) revert Error()
```

Client's Commentary

Thanks for the suggestion to replace literal require() strings with custom errors. Action taken: We have migrated to custom errors in all new or upgrade-ready contracts covered by the audit: L1BTCRedeemerWormhole, AbstractBTCRedeemer, L2BTCRedeemerWormhole. These changes are included in commit: PR-882
Exception: We intentionally left L2WormholeGateway unchanged.
This contract's bytecode is already deployed on several L2 networks and is referenced by external integrations. Altering its revert strings would require fresh deployments and guardian-signer updates across those chains. After weighing the modest gas savings against the operational overhead and the value of keeping a canonical code base for developers, we decided to retain the existing string messages for that contract.

L-9	rescueTbtc Function Lacks Event Emission		
Severity	Low	**Status**	Fixed in 042d7f19

Description

AbstractBTCRedeemer.rescueTbtc() transfers arbitrary amounts of tBTC from the contract to an external address but emits no event. Without an event, off-chain indexers cannot easily observe token recoveries, hindering transparency and incident response.

Recommendation

We recommend emitting an event in the rescueTbtc function.

Client's Commentary

addressed in the commit: PR-882

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

https://mixbytes.io/

https://github.com/mixbytes/audits_public

hello@mixbytes.io

https://x.com/mixbytes