2026

BTC-BRIDGE AUDIT REPORT

NEAR ONE

DATE :
31 January 2026

PRESENTED BY :
Sherief El Sayad

Table Of Contents

Executive Summary

The audited project is the **BTC-Bridge on Near Protocol**, a smart-contract system that facilitates cross-chain transfers between Bitcoin (and optionally Zcash) and NEAR by verifying BTC transactions via a light client and minting/burning nBTC. Overall, the system is **generally robust** but contains **one high-severity issue** that could lead to loss of funds under specific failure conditions, along with several lower-severity and informational observations. One previously flagged configuration item (BTCB-003) was reassessed and finalized as **not a vulnerability** under the stated trust model.

Summary of findings by severity:

Severity	Count	Resolved
Critical	0	0
High	1	0
Medium	0	0
Low	1	0
Informational	3	N/A

Key recommendations:

- Treat failed `safe_mint` promise results as failures, not successes, to avoid consuming UTXOs without minting nBTC.
- Add configuration invariants (e.g., `fee_min <= min_deposit_amount`, `withdraw_fee + gas_fee <= min_withdraw_amount`) as operational guardrails.
- Align post-action gas validation with execution constraints.

Introduction

This audit assesses the Near One **BTC-Bridge** implementation from the repository `https://github.com/Near-One/btc-bridge`, which provides a Bitcoin bridge for NEAR by verifying on-chain BTC transactions and minting/burning the nBTC token accordingly.

Objectives:

- Identify vulnerabilities and logic flaws.
- Evaluate access control and upgradeability risks.
- Validate correctness of deposit/withdrawal flows and UTXO management.
- Review adherence to NEAR and general smart contract security best practices.

Audit period: 01/01/2026 - 31/01/2026
Auditor: Sherief El Sayad
High-level bridge risks: security depends on external components such as the BTC light client and chain-signatures contract, as well as DAO-controlled configuration and upgradeability. These are trusted dependencies that can materially affect safety.

Scope of Audit

Commit hash audited: `533e4617c7bcd8fadc77d842c1f8a9002fb71425`

In-scope:

- `contracts/satoshi-bridge/src/**` (core bridge contract)
- `contracts/nbtc/src/lib.rs` (nBTC token contract)

Excluded:

- `contracts/mock-*` (mock/test helper contracts used for local or integration testing)
- Tests and fixtures under `contracts/**/tests`
- Off-chain relayer and infrastructure components

Protocols/Standards:

- NEAR runtime and NEP-141 (Fungible Token standard)
- Bitcoin transaction verification via external light client
- Chain signatures / MPC signing contract

Lines of Code Audited:

Component	Files	Lines of Code
Core Bridge Contract	`contracts/satoshi-bridge/src/**`	8,086
nBTC Token Contract	`contracts/nbtc/src/lib.rs`	418
Total	**All in-scope**	**8,504**

Methodology

- **Manual review** of contract logic, storage flows, access control, and cross-contract calls.
- **Targeted reasoning** for UTXO management, PSBT construction, mint/burn safety, and RBF flows.
- **Lightweight tooling** (ripgrep, git, wc) to map code paths and collect scope metrics.

Standards referenced:

- NEAR smart contract best practices.
- NEP-141 fungible token standard behaviors.

Phases:

1. Repository inventory and scoping.
2. Core logic deep dive (deposit, withdraw, mint, burn, RBF).
3. Admin/upgrade and configuration review.
4. Findings consolidation and report drafting.

Severity model (impact × likelihood):

- **Critical**: Immediate, systemic loss of funds or irreversible compromise.
- **High**: Loss of funds or severe integrity breach under plausible conditions.
- **Medium**: Partial loss of funds, long-term DoS, or significant misbehavior.
- **Low**: Limited impact or edge-case operational failures.
- **Informational**: Best practices, clarity, or trust assumptions.

System Overview

Key components:

- **BTC Light Client**: validates Bitcoin transaction inclusion and confirmations.
- **Bridge Contract**: manages UTXOs, verifies deposits/withdrawals, and orchestrates mint/burn.
- **nBTC Contract**: NEP-141 token with mint/burn controlled by the bridge.
- **Chain Signatures**: external MPC signer used to sign Bitcoin transactions.

Data flows:

- **Deposit**: user sends BTC → relayer calls `verify_deposit` → light client validates → bridge mints nBTC.
- **Withdraw**: user sends nBTC to bridge via `ft_transfer_call` → bridge constructs PSBT → chain-signatures signs → relayer broadcasts → light client verifies → bridge burns nBTC and records change UTXOs.

Dependencies:

- `near-sdk`, `near-contract-standards`, `bitcoin` crate, `near-plugins`.

Environment assumptions:

- NEAR runtime guarantees atomic execution per transaction.
- BTC confirmations and chain data are correct in light client.

Findings

8.1 Critical Severity Findings

None identified.

8.2 High Severity Findings

ID	Title	Status
BTCB-001	Safe deposit callback treats failed promise as success	Open

BTCB-001 — Safe deposit callback treats failed promise as success

Severity: High

Description: In `safe_mint_callback`, success is derived from `!is_refund_required()`. The callback reads the **return value of** `nbtc::safe_mint`, which (when `msg` is provided) comes from NEP-141 `ft_resolve_transfer` and represents **used_amount**, not the receiver's raw `ft_on_transfer` return. That path is correct. The issue is that `is_refund_required()` treats `PromiseResult::Failed` **as "no refund required"**, so a failed `safe_mint` promise is incorrectly handled as success. This can mark the deposit UTXO as spent even though nBTC was never minted.

Impact: Loss of user funds (BTC deposit becomes consumed without receiving nBTC) and inability to re-verify the deposit.

PoC:

1. Deploy an nBTC contract that panics or call `safe_mint` with insufficient gas.
2. Call `safe_verify_deposit` with a valid BTC proof.
3. Observe that the callback records success and adds the UTXO, but no nBTC is minted.

How to reproduce (test PoC):

- PoC test file: `contracts/satoshi-bridge/src/unit/btcb_001.rs`
- Run:

```
cargo test -p satoshi-bridge btcb_001
```

- Proof: the test simulates a `PromiseResult::Failed` callback and asserts that `safe_mint_callback` still returns success and keeps the UTXO recorded.

Location: `contracts/satoshi-bridge/src/btc_light_client/deposit.rs:214-274`

Status: Open

8.3 Medium Severity Findings

None identified.

8.4 Low Severity Findings

ID	Title	Status
BTCB-002	Post-action gas boundary mismatch	Open

BTCB-002 — Post-action gas boundary mismatch

Severity: Low

Description: `check_deposit_msg` allows per-post-action gas down to **30 Tgas**, but `handle_post_action` requires `prepaid_gas > 30 Tgas`. Using exactly 30 Tgas passes validation but will always fail at execution time.

Impact: Post-actions fail unexpectedly for boundary values; could break integrations relying on minimum gas.

Likelihood: High — integrations may choose the minimum allowed gas.

PoC:

1. Include a post-action with `gas = 30 Tgas` in `DepositMsg`.
2. Validation passes; execution fails with "More gas is required."

   How to reproduce (test PoC):

- PoC test file: `contracts/nbtc/tests/btcb_002.rs` (demonstrates BTCB-002)
- Run:

```
cargo test -p nbtc btcb_002
```

- Proof: the test sets `prepaid_gas` to exactly 30 Tgas and calls `handle_post_action`, which panics due to the strict `>` check, demonstrating the boundary mismatch.

  Location: `contracts/satoshi-bridge/src/deposit_msg.rs:5-8, 127-149`, `contracts/nbtc/src/lib.rs:398-401`

  Status: Open

8.5 Informational Findings

ID	Title	Status
BTCB-003	Configuration invariants missing (operational hardening only)	Open
BTCB-004	`safe_deposit` / `post_actions` exclusivity not enforced	Open
BTCB-005	External dependency and upgrade trust assumptions	Open

BTCB-003 — Configuration invariants missing (operational hardening only)

Severity: Informational

Final judgment: **Not a vulnerability**

Rationale: The potential underflow/panic scenarios arise only if DAO-controlled configuration is set to invalid values. Under the stated threat model, configuration is trusted and not attacker-controlled, so this is **not an exploitable vulnerability**. It is best treated as an **operational hardening recommendation** (add invariants to reduce misconfiguration risk).

Location: `contracts/satoshi-bridge/src/config.rs:18-142`, `contracts/satoshi-bridge/src/btc_light_client/deposit.rs:41-50`, `contracts/satoshi-bridge/src/psbt.rs:230-235`

Status: Informational

BTCB-004 — `safe_deposit` / `post_actions` exclusivity not enforced

Severity: Informational

Description: The `DepositMsg` comment states that `safe_deposit` should not coexist with `post_actions`, but `safe_verify_deposit` does not enforce this. The deposit path includes the full message (including `post_actions`), yet the safe path ignores those actions.

Impact: Potential integration confusion; mismatched expectations for deposit address and executed actions.

Likelihood: Medium.

Location: `contracts/satoshi-bridge/src/deposit_msg.rs:21-24`, `contracts/satoshi-bridge/src/api/bridge.rs:95-113`

Status: Open

BTCB-005 — External dependency and upgrade trust assumptions

Severity: Informational

Description: The bridge's correctness depends on external contracts (BTC light client, chain-signatures, nBTC) and DAO-controlled upgrades/configuration. These are trusted dependencies, and any compromise or misconfiguration can undermine security guarantees.

Impact: Systemic risk if governance or external dependencies are compromised.

Likelihood: Context-dependent.

Location: `contracts/satoshi-bridge/src/api/management.rs:256-270`, `contracts/satoshi-bridge/src/chain_signature.rs:45-120`

Status: Open

Recommendations

1. Treat `PromiseResult::Failed` as a failed safe deposit and revert/clean state accordingly.
2. Add explicit configuration invariants for fees and minimum amounts as operational guardrails; validate during initialization and DAO updates.
3. Align post-action gas validation with execution requirements (e.g., change `>` to `>=` or raise minimum to 31 Tgas).
4. Add explicit validation for `safe_deposit` vs. `post_actions` in `safe_verify_deposit`.
5. Consider documenting and operationally enforcing trust assumptions around upgradeability and external dependencies.
6. Re-audit after fixes and before mainnet deployments.

10. Conclusion

The BTC-Bridge implementation demonstrates solid structure and careful handling of UTXOs, PSBT validation, and cross-contract flows. One high-severity issue in the safe deposit path could lead to lost funds under failure conditions. A configuration item was reclassified as **not a vulnerability** (operational hardening only). With the safe-deposit fix and ongoing governance diligence, the bridge can reach a stronger security posture.

Confidence level: Moderate — code coverage and logic review were thorough, but no automated testing or fuzzing was performed during this audit.
Next steps: implement remediations, run regression tests, and re-audit critical paths.

Appendices

11 Code Snippets

Safe deposit callback success decision (BTCB-001)

```
let is_success = !is_refund_required();
...
PromiseResult::Failed => false,
```

Deposit fee subtraction (BTCB-002)

```
let deposit_fee = config.deposit_bridge_fee.get_fee(deposit_amount);
let mint_amount = deposit_amount - deposit_fee;
```

Post-action gas boundary (BTCB-002)

```
const MIN_PER_POST_ACTIONS_GAS: Gas = Gas::from_tgas(30);
require!(env::prepaid_gas() > GAS_FOR_FT_TRANSFER_CALL, "More gas is
```

11.1 Tools Used

- Manual review
- `ripgrep 13.0.0`
- `git 2.39.5`
- `rustc 1.86.0`

11.2 References

- https://github.com/Near-One/btc-bridge
- https://nomicon.io/ (NEAR smart contract security best practices)
- https://nomicon.io/Standards/FungibleToken/Core (NEP-141 standard)
- https://developer.bitcoin.org/reference/transactions.html
- https://github.com/near/near-sdk-rs