OFFBEAT SECURITY

GOAT Safebox Review

April 18, 2025

Prepared for GOAT Network

Conducted by:

Vara Prasad Bandaru (s3v3ru5)

About the GOAT Safebox Review

The GOAT Safebox (implemented in the TaskManagerUpgradeable contract) manages the lifecycle of bridging BTC from the Bitcoin chain to the Goat Network and back as liquidity for a fixed period of time. The process involves an admin creating tasks for users to provide BTC as liquidity, users transferring BTC to a GOAT Network partner wallet, relayers minting gBTC (Goat's native token) as liquidity, and initiating timelock transactions for returning BTC to users after the timelock period. The system verifies Bitcoin transactions through merkle proofs and manages the conversion between BTC and gBTC.

About Offbeat Security

Offbeat Security is a boutique security company providing unique security solutions for complex and novel crypto projects. Our mission is to elevate the blockchain security landscape through invention and collaboration.

Summary & Scope

The src folder was reviewed at commit 0ee9a29.

The following **4 files** were in scope:

- src/interfaces/IBitcoin.sol
- src/interfaces/IBridge.sol
- src/TaskManagerUpgradeable.sol
- src/UpgradeableProxy.sol

The Safebox manages the lifecycle of BTC liquidity provision on the Goat Network. It stores information about tasks, tracks their states, and handles the burning of gBTC tokens after the liquidity period ends.

The review identified 1 MEDIUM and 1 LOW severity issues related to merkle proof verification and protocol scalability. Additional recommendations focus on code quality improvements and centralization risks in the trusted roles system.

Summary of Findings

Identifier	Title	Severity	Fixed
M-01	Merkle proof verification allows intermediate nodes to pass as transaction IDs	Medium	d9c2aab, ee8ed65, 986d765, f71d3e4, 0000719, ba0c060
L-01	Deposit address restriction limits protocol scalability	Low	Acknowledged

Additional Recommendations

Code Quality

1. **Remove unused EnumerableSet library import in TaskManagerUpgradeable contract.** The EnumerableSet library is imported but never used anywhere in the contract, eliminating it would reduce deployment gas costs.

2. **Define an enum for task states instead of using uint8 with comments.** Using an enum would improve code readability and reduce the risk of errors by providing named constants for each state.

3. **Remove uninitialized and unused TaskManagerUpgradeable.partnerBeacon immutable variable.** This variable is declared but never initialized or used, removing it would simplify the contract.

4. **Call _disableInitializers() in the TaskManagerUpgradeable contract's constructor.** Adding this call would prevent the initializer function from being called during deployment, which is a security best practice for upgradeable contracts.

5. **Change TaskManagerUpgradeable.hasPendingTask mapping to use bool instead of uint256.** Using a boolean value would be more gas efficient and provide clearer semantics than using numeric values (0/1/2).

Centralization Risks

This appendix identifies trusted roles in the system and the specific trust assumptions for each role.

1. Admin is trusted to not cancel tasks abruptly without providing users sufficient time to perform BTC deposits.

2. Relayer is trusted to perform the correct timelock transaction and provide the same transaction ID in the call to initialize timelock transaction function. A correct timelock transaction transfers the correct BTC amount to the user with a timelock period equal to the one specified in the Task.

3. The deposit address (safe wallet) is trusted to transfer bridged BTC after timelockEndTime to the TaskManager contract.

Detailed Findings

Medium Findings

[M-01] Merkle proof verification allows intermediate nodes to pass as transaction IDs

Impact: High
Likelihood: Low

Description

The TaskManagerUpgradeable contract's implementation of Bitcoin merkle proof verification has a vulnerability that allows relayers to prove non-transaction data as valid transaction IDs. The current verification only confirms that the provided hash, when combined with the proof, results in the expected root hash without validating that the hash belongs to an actual transaction.

This issue exists because Bitcoin's merkle trees have transaction IDs as leaves, with internal nodes created by hashing pairs of children. The current implementation cannot distinguish between actual transaction IDs and intermediate merkle nodes.

Additionally, the contract relies extensively on trusting relayers to provide legitimate transaction IDs that transfer the correct amount of BTC to users with the proper timelock. This centralization risk is compounded by the lack of on-chain verification of transaction details, allowing a relayer to exploit the system with relatively simple operations.

Exploit Scenario

A malicious relayer could exploit this vulnerability through the following steps:

1. Select an intermediate node from a Bitcoin block's merkle tree (or even a block hash itself)
2. Use this as the `timelockTxHash` in `initTimelockTx()`
3. Provide a partial merkle proof in `processTimelockTx()` that verifies this node
4. Successfully transition the task to the CONFIRMED state without creating an actual BTC transaction
5. Allow gBTC to be burned on the Goat network without the user receiving their BTC

This allows the relayer to complete the task lifecycle without sending BTC back to users, resulting in direct loss of user funds.

Recommendation

Instead of accepting arbitrary transaction hashes from relayers, the contract should compute the transaction hash from the raw transaction data. This would allow verification that:

1. The hash represents an actual transaction, not an internal merkle node
2. The transaction contains the correct parameters (recipient address, amount, timelock)
3. The transaction is properly included in the Bitcoin blockchain

This approach reduces centralization risk by ensuring cryptographic verification of the complete transaction rather than relying solely on trusted relayers.

Low Findings

[L-01] Deposit address restriction limits protocol scalability

Impact: Medium
Likelihood: Low

Description

The TaskManagerUpgradeable contract restricts each deposit address (which is a Gnosis safe wallet) to be used for only one task at a time, creating a significant bottleneck for protocol scalability. Deposit addresses remain locked for extended periods that include Bitcoin transaction confirmation times for bridging BTC onto the Goat network and for subsequent timelock transaction confirmations on the Bitcoin network.

When a task is created via `setupTask()`, a deposit address is exclusively reserved by setting `hasPendingTask[_depositAddress] = 2`. This address remains blocked until either:

1. The task is canceled via `cancelTask()` (requiring ADMIN_ROLE and only for tasks in created state). The Admin is typically expected to wait until the task deadline passes before cancellation.
2. The task progresses through Bitcoin timelock transaction confirmation and reaches the confirmed state via `processTimelockTx()`

This limitation creates serious scalability issues as the protocol's capacity becomes directly limited by the number of available deposit addresses. During periods of high demand, the protocol would quickly exhaust available addresses, causing denials of service without requiring any malicious behavior.

Recommendation

Remove the restriction that limits deposit addresses to a single task.

Fix Review Result

Acknowledged.

GOAT Network

This is a private system where each user is individually engaged and served. Therefore, we don’t expect a large number of users, nor do we anticipate any scaling issues in the future.