Security Audit Report

Mezo:

Portal Smart Contracts

Final Audit Report: March 14, 2024

thesis defense
defense@thesis.co

Table of Contents

About Thesis Defense

Thesis Defense serves as the auditing services arm within Thesis, Inc., the venture studio behind tBTC, Fold, Taho, Etcher, and Embody. Our team of senior security and cryptography auditors has extensive security experience in the decentralized technology space. In addition, the Thesis Defense team has a demonstrated track record in a variety of languages and technologies, including, but not limited to, smart contracts, cryptographic protocols including zk-cryptography, dApps including wallets and browser extensions, and bridges. Thesis Defense has extensive experience conducting security audits across a number of ecosystems, including, but not limited to, Ethereum, Zcash, Stacks, Mina, Polygon, Filecoin, and Bitcoin.

Thesis Defense will employ the Thesis Defense Audit Approach and Audit Process to the above in-scope service. In the event that certain processes and methodologies are not applicable to the aforementioned in-scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful Security Audit Preparation.

Scope

Overview

Thesis Defense conducted a manual code review of Mezo's Portal Smart Contracts implementation.

Project Team

- **Ahmad Jawid Jamiulahmadi,** Senior Security Auditor
- **Mukesh Jaiswal,** Security Auditor
- **Bashir Abu-Amr,** Senior Technical Writer

Schedule

- **Code Review:** March 4 - 8, 2024
- **Audit Report Delivery:** March 8, 2024
- **Final Report Delivery:** March 14, 2024

Code

- **Mezo Portal**

    - **Repository:** https://github.com/thesis/mezo-portal/tree/main/solidity
    - **Branch:** `main`
    - **Hash:** `39a312c36a9e1abd5a7f3d982fad2b5ed452c8eb`
    - **Files:** Portal.sol, BitcoinDepositor.sol
- **tBTC v2**
    - **Repository:** https://github.com/keep-network/tbtc-v2/tree/main
    - **Branch:** `main`
    - **Hash:** `9e047d11703415e1a1844a64b4985a181570fcdd`
    - **Files:** AbstractTBTCDepositor.sol
- **Mezo Portal Verification Commit**
    - **Repository:** https://github.com/thesis/mezo-portal/releases/tag/solidity%2Fv0.1.0
    - **Branch:** `'solidity/v0.1.0'`
    - **Hash:** `0000ff5c322edeb69b18072c7cd2455b8afc8bf2`

Project Documentation

- **Mezo Security Audit Document:** https://coda.io/d/Mezo_d_vIDNPK008/Internal-security-audit_suQky#_lu8y4
- **Mezo Release 1 Specification Document:** https://coda.io/d/Mezo_d_vIDNPK008/Release-1-Stoker_suYGz#_luPrn
- **tBTC Repository Documentation:** https://github.com/keep-network/tbtc-v2/blob/main/docs/rfc/rfc-11.adoc#22-implementation

---

Findings

The Mezo Portal smart contracts are intended to allow BTC and tBTC holders to deposit and lock tokens in order to earn points. These points will be to determine airdrops from the Mezo Blockchain mainnet. The smart contracts are intended to be integrated with a frontend interface and the tBTC bridge, which were considered out of scope for this audit.

Threat Model

For this review, our team considered a threat model whereby the smart contracts assume all external components as untrusted. For those components that are out of scope for this review, we considered the components to be untrusted, but functioning as intended. The Mezo Portal smart contracts are designed to be integrated with the tBTC v2 bridge. We assumed to tBTC the bridge behaves as expected. The smart contracts are designed to be integrated with a user

interface which we considered untrusted. Furthermore, we assume that the governance of the Portal Smart Contracts is sufficiently decentralized and not malicious.

Security by Design

We identified issues in some design elements of the smart contracts. We found that the `BitcoinDepositor` smart contract does not check that the user selected locking period specified in the extra data section of the Bitcoin script is in range, which could make the user unable to withdraw their tokens from the protocol (Issue A). We recommend a solution in the smart contract.

We also found that the smart contracts do not implement a 2-step process for transferring the smart contract ownership address (Issue C).

In looking at the out-of-scope tBTC bridge implementation, to have a more precise understanding of the smart contracts in scope, we discovered an unlikely but possible condition in the tBTC bridge where users would be unable to make deposits. We recommend that the user interface alert users of the current status of the bridge (Issue B).

In our audit, our team also noted a lock period entered as non-integer weeks (4.5 or 5.5 weeks), the deposit interval is rounded down. Although this is intended behavior, we recommend clarifying this to the users.

Secure Implementation

We conducted an in-depth examination and manual review of the files in scope and found them implemented in adherence to best practices.

Use of Dependencies

The project uses an OpenZeppelin library whose version is not the most recent. We recommend using the most recent version that includes up to date security fixes. (Suggestion 4).

Tests

There is sufficient testing implemented, which covers most of the functionality of the `Portal` and `BitcoinDepositor` smart contracts.

Project Documentation

The files are well commented and adhering to NatSpec, but there are some instances of code comments that should be updated to reflect the implementation more accurately (Suggestion 1).

Issues and Suggestions

Issues	Status
Issue A: BitcoinDepositor Might Fail to Finalize Some Deposits	Reported
Issue B: Optimistic Pause of Bridge (Out-of-Scope)	Reported
Issue C: Lack of a Two-Step Process for Ownership Change	Fixed

Suggestions	Status
Suggestion 1: Update Code Comments to Reflect the Implementation	Fixed
Suggestion 2: Check the Sanity of Lock Interval Parameters When Setting MinLockPeriod and MaxLockPeriod	Fixed
Suggestion 3: Prevent Adding a Supported Token With None Ability	Fixed
Suggestion 4: Use latest Open Zeppelin Library Implementation	Fixed
Suggestion 5: Pin and Lock Pragma	Fixed
Suggestion 6: Implement 0 Address Check	Fixed
Suggestion 7: Prevent Resetting the depositInfo.unlockAt	Fixed
Suggestion 8: Check For Equality When Setting minLockPeriod and maxLockPeriod	Fixed

Issues

Issue A: BitcoinDepositor Might Fail to Finalize Some Deposits

Location

BitcoinDepositor.sol#L258-L263
BitcoinDepositor.sol#L258-L263
BitcoinDepositor.sol#L258-L263

Description

The `finalizeDeposit` function in the `BitcoinDepositor` smart contract finalizes tBTC deposits revealed to the tBTC bridge, and deposits them on behalf of the actual depositor to the `Portal` smart contract. The Bitcoin transaction token, which is a P2(W)SH Bitcoin script, includes the deposit owner address and deposit lock time in seconds in the `depositor-extra-data` section of the token.
Deposits made to the `Portal` smart contract can be locked for a user specified period which must be into a specific range i.e., more than `minLockPeriod` and less than `maxLockPeriod`. Deposits with a locking period less than the current minimum locking period and more than the current maximum locking period will be disallowed and the transaction will be reverted. Therefore, if the locking period specified in the `depositor-extra-data` section in the P2(W)SH Bitcoin script, is not in the specific range, the `BitcoinDepositor` smart contract will not be able to finalize the deposit in the `finalizeDeposit` function since the transaction is reverted due to a call to the `depositFor` function in the `Portal` smart contract which prevents deposits with a `lockingPeriod` less than the current minimum locking period or more than the current maximum locking period if the `lockingPeriod` is not zero.

Impact

Since the `finalizeDeposit` function in the `BitcoinDepositor` smart contract is always reverted due to the incorrect locking period, the user's deposit will be stuck in the `BitcoinDepositor` smart contract.

Remediation

Resolving this issue requires careful consideration of potential security risks and complexity that can be introduced with a specific solution. However, we can recommend the following solutions:
Before sending the `depositFor` transaction to the `Portal` smart contract, in the `finalizeDeposit` function in the `BitcoinDepositor` smart contract, check if the `lockingPeriod` provided is within the range of minimum locking period and maximum locking period, and if not:

1. Transfer the minted tBTC tokens directly to the deposit owner, or
2. Deposit tokens to the Portal smart contract without locking them (deposit with zero locking period), or
3. Deposit tokens to the Portal smart contract with the current maximum or minimum locking period.

Verification Status

Reported and unresolved. This issue can be mitigated in the Mezo interface dApp, which was out of scope for this security audit.

Issue B: Optimistic Pause of Bridge (Out-of-Scope)

Location

contracts/BitcoinDepositor.sol#L183

Description

The tBTC bridge handles user deposits through optimistic minting and sweeping. Optimistic minting allows the minting of tBTC prior to the `TBTCVault` receiving the `Bank` smart contract balance. Two permissioned sets, Minters and Guardians, operate in a 1-of-n mode. Minters monitor revealed deposits and can request the minting of tBTC, with any single Minter capable of initiating this action. A delay, known as "`optimisticMintingDelay`," occurs between the Minter's request and the actual minting of tBTC. Within this delay period, any Guardian has the authority to cancel the minting process.

In Sweeping the bridge active wallet periodically signs a transaction that unlocks all of the valid, revealed deposits above the dust threshold, combines them into a single UTXO, and the balances of depositors in the Bank smart contract are increased when the Simple Payment Verification sweep proof is submitted to the bridge.

As a result, there are conditions when tBTC minting is not allowed:

1. When the optimistic minting is paused, Minters will not be able to put in a request for an optimistic minting of tBTC
2. When the wallet is in the state “`MovingFunds`”. At this point, the wallet is expected to move outstanding funds to another wallet. The wallet can still fulfill pending redemption requests, although new redemption requests and new deposit reveals are not accepted.

As a result, when optimistic minting is paused and the wallet is in the state "`MovingFunds`", the user will not be able to reveal their deposited BTC to the `BitcoinDepositor` smart contract.

Impact

While the likelihood of such a situation occurring is minimal, it is still possible. If it does happen, users will have the option to claim their deposited BTC after the expiration of the lock time interval. This effectively renders the user's assets inaccessible for the duration of the lock period, and the user is unable to earn points.

Mitigation

We recommend that users are alerted in the interface about the paused status of the bridge and wallet status before allowing any deposit transactions.

Verification Status

Reported and unresolved. The Mezo team stated that the Mezo interface dApp will restrict deposits when the tBTC bridge is in optimistic pause mode.

Issue C: Lack of a Two-Step Process for Ownership Change **[Fixed]**

Location

Portal.sol#L9

Description

The `Portal` smart contract uses OpenZeppelin's `OwnableUpgradeable` library for the transfer of ownership of a smart contract from one owner address to another. This library does not implement a two-step ownership transfer.

Impact

A two-step process for ownership transfer significantly reduces the probability of incorrectly transferring ownership of a smart contract which would result in the permanent loss of control of the smart contract.

Remediation

We recommend implementing a Two-Step process for ownership change. We recommend using OpenZeppelin's Ownable2StepUpgradeable library for this purpose.

Verification Status

Fixed.

Suggestions

Suggestion 1: Update Code Comments to Reflect the Implementation **[Fixed]**

Location

Portal.sol#L305
Portal.sol#L315

Description

There are comprehensive code comments in the smart contracts that adhere to NatSpec guidelines. However, our team found instances of an inaccurate code comment

Remediation

We recommend updating the referenced code comments.

Verification Status

Fixed.

Suggestion 2: Check the Sanity of Lock Interval Parameters When Setting minLockPeriod and maxLockPeriod **[Fixed]**

Location

Portal.sol#L171-L182
Portal.sol#L189-L200

Description

The `setMinLockPeriod` and `setMaxLockPeriod` functions in the `Portal` smart contract do not check if the newly set locking period is normalized to weeks. Additionally, the `setMinLockPeriod` function currently lacks appropriate input validation for a minimum lock period of one week. The absence of a check allows the lock duration interval to be set to values that are not normalized.
Without input validation, the minimum lock period can be set to any value that is less than 1 week, which is not consistent with the intended functionality and could lead to unexpected outcomes.

Remediation

We recommend that the locking periods supplied in the aforementioned functions are checked for normalization. We also recommend implementing a sanity check on the minimum lock duration parameter to enforce the minimum one-week lock interval.

Verification Status

Fixed.

Suggestion 3: Prevent Adding a Supported Token With None Ability **[Fixed]**

Location

Portal.sol#L149-L166

Description

The `addSupportedToken` function in the `Portal` smart contract is used to add a new token to the list of supported tokens. A new supported token should be added with a token ability of `Deposit` or `DepositAndLock`. However, this function allows adding a supported token with the `None` token ability.

Remediation

We recommend adding to a check to prevent adding a supported token with the `None` token ability.

Verification Status

Fixed.

Suggestion 4: Use latest Open Zeppelin Library Implementation **[Fixed]**

Location

`package.json#L48`

Description

The current version of the smart contracts relies on a prior release of the OpenZeppelin library, wherein an issue with Base64 encoding exists. While this problem does not currently impact the functionality of the `Portal` smart contracts, upgrading the library helps the contract mitigates potential security issues related to the known issue as the project evolves over time

Remediation

We recommend that an upgrade be made to the Open Zeppelin library version to 5.0.2

Verification Status

Fixed.

Suggestion 5: Pin and Lock Pragma **[Fixed]**

Description

When using the pragma directive in Solidity, it is essential to specify the exact version of the Solidity compiler that your smart contract is compatible with. This practice, known as locking the pragma, ensures that your contract is compiled and executed as intended, avoiding potential issues caused by compiler version differences.

Remediation

We recommend specifying the most recent, exact version of the Solidity compiler.

Verification Status

Fixed.

Suggestion 6: Implement 0 Address Check **[Fixed]**

Location

Portal.sol#L380

Description

In the location referenced above, a zero address check is missing validating the correctness `depositOwner` address, thereby preventing an incorrectly set value. Zero address checks are essential when an address is the receiver of a token or value.

Remediation

We recommend adding a zero address check for `depositOwner` in the `_depositFor` function.

Verification Status

Fixed.

Suggestion 7: Prevent Resetting the depositInfo.unlockAt **[Fixed]**

Location

Portal.sol#L361-L369

Description

The referenced `if` condition in the `lock` function inside the `Portal` smart contract does not revert if the newly provided unlocking time is the current unlocking time hence resetting it with the same value.

Remediation

We recommend preventing the resetting of the `depositInfo.unlockAt` by adding an equality check in the revert condition referenced above.

Verification Status

Fixed.

Suggestion 8: Check For Equality When Setting minLockPeriod and maxLockPeriod **[Fixed]**

Location

Portal.sol#L171-L178
Portal.sol#L190-L196

Description

When setting `minLockPeriod` and `maxLockPeriod` in the `setMinLockPeriod` and `setMaxLockPeriod` functions respectively in the `Portal` smart contract, `minLockPeriod` can be set to `maxLockPeriod` and vice versa since the referenced revert conditions in both functions don't check for equality of `minLockPeriod` and `maxLockPeriod` to revert consequently. This might result in unexpected behavior.

Remediation

We recommend that a check for equality in the referenced functions be also implemented to prevent unexpected behavior.

Verification Status

Fixed.