

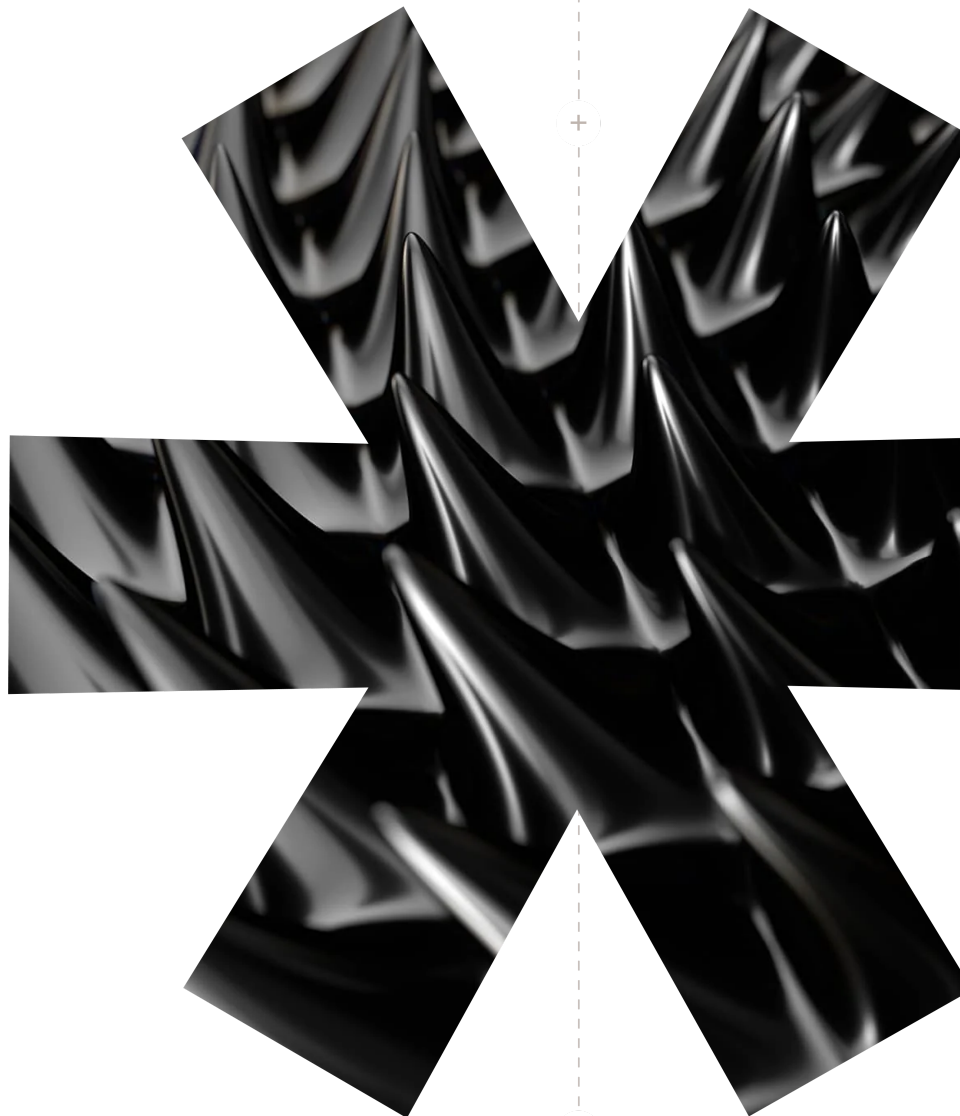
Security Audit Report

Acre

Acre Smart Contracts

Initial Report // April 26, 2024

Final Report // May 17, 2024



Team Members

Bernd Artmüller // Security Auditor

J4X // Security Auditor

Bashir Abu-Amr // Head of Delivery

Table of Contents

<u>1.0 Scope</u>	4
↳ <u>1.1 Technical Scope</u>	
↳ <u>1.2 Documentation</u>	
<u>2.0 Executive Summary</u>	5
↳ <u>2.1 Schedule</u>	
↳ <u>2.2 Overview</u>	
↳ <u>2.3 Threat Model</u>	
↳ <u>2.4 Security by Design</u>	
↳ <u>2.5 Secure Implementation</u>	
↳ <u>2.6 Use of Dependencies</u>	
↳ <u>2.7 Tests</u>	
↳ <u>2.8 Project Documentation</u>	
<u>3.0 Key Findings Table</u>	7
<u>4.0 Findings</u>	9
↳ <u>4.1 Withdrawal Exit Fee Is Not Considered When Pulling Additional Funds From Mezo Portal Resulting in Temporarily Halted Withdrawals</u>	
↳ <u>4.2 depositorFee Will Get Stuck in the Depositor Contract if depositorFee >= btcAmount</u>	Medium Fixed
↳ <u>4.3 Assets Can Be Stolen Due to Incorrect Implementation of totalAssets</u>	Medium Not Fixed
↳ <u>4.4 Virtual Shares Will Accrue Rewards and Still Allow for an Inflation Attack at a Loss</u>	Medium Fixed
↳ <u>4.5 stBTC Tokens Can Be Transferred When the Contract Is Paused</u>	Medium Reported
↳ <u>4.6 minDepositAmount Restriction of the Depositor Is Never Enforced</u>	Low Not Fixed
↳ <u>4.7 ERC4626 Max Functions Must Return 0 if the Protocol Is Paused</u>	Low Fixed
↳ <u>4.8 maxWithdraw Function Does Not Factor in Fees</u>	Low Fixed
↳ <u>4.9 stBTC.assetsBalanceOf Does Not Take Fees Into Consideration</u>	Low Not Fixed
↳ <u>4.10 BitcoinRedeemer Does Not Check for the Amount Being a Multiple of SATOSHI_MULTIPLIER</u>	None Not Fixed
↳ <u>4.11 depositorFee Always Rounds Down</u>	Low Fixed
↳ <u>4.12 Fees Are Not Checked for Being > Than 10.000 Basis Points</u>	Low Fixed
↳ <u>4.13 No Checks for minimumDepositAmount Being Lower Than depositDustThreshold</u>	Low Not Fixed
↳ <u>4.14 Missing Checks of Redeemer Address in Redeeming Process</u>	None Fixed



- ↳ [4.15 Missing Checks for Exorbitant Slippage on Calls to the Vault](#)
 -
 -
- ↳ [4.16 Missing Check for Ownership of tBTC Token When Updating The tbtcVault Address](#)
 -
 -
- ↳ [4.17 Multiple Missing Checks for newVal == oldVal](#)
 -
 -
- ↳ [4.18 BitcoinDepositor and BitcoinRedeemer Contracts Do Not Use the zeroAddress Error Provided in Errors.sol](#)
 -
 -
- ↳ [4.19 DepositFinalized Event Is Missing a Parameter for the Deposit Owner](#)
 -
 -
- ↳ [4.20 Documentation of DepositFinalized Event Is Missing the Referral Parameter](#)
 -
 -
- ↳ [4.21 Incorrect Documentation of 'stBTC.approveAndCall'](#)
 -
 -
- ↳ [4.22 Misleading Documentation of the MezoAllocator.addMaintainer Function](#)
 -
 -
- ↳ [4.23 Private and Internal Functions Do Not Adhere to the Solidity Style Guide](#)
 -
 -
- ↳ [4.24 Errors.sol Is Missing Natspec Documentation](#)
 -
 -
- ↳ [4.25 Use of Floating Pragma](#)
 -
 -
- ↳ [4.26 Check in BitcoinRedeemer.receiveApproval can be simplified](#)
 -
 -
- ↳ [4.27 Spelling Error in the BitcoinDepositor Tests](#)
 -
 -

5.0 Appendix A

_____ 37

- ↳ [5.1 Severity Rating Definitions](#)

6.0 Appendix B

_____ 38

- ↳ [6.1 Thesis Defense Disclaimer](#)



About Thesis Defense

Thesis Defense serves as the auditing services arm within Thesis, Inc., the venture studio behind tBTC, Fold, Tahoe, Etcher, and Mezo. Our team of security auditors have carried out hundreds of security audits for decentralized systems across a number of technologies including smart contracts, wallets and browser extensions, bridges, node implementations, cryptographic protocols, and dApps. We offer our services within a variety of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos / Cosmos SDK, NEAR and more.

Thesis Defense will employ the Thesis Defense Audit Approach and Audit Process to the in scope service. In the event that certain processes and methodologies are not applicable to the 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.

Section 1.0

Scope

Technical Scope

- **Repository:** <https://github.com/thesis/acre/tree/main/solidity>
- **Audit Commit:** 965f38be1ab0896cf26450a443789016529fa309
- **Verification Commit:** b4d39517ab33fe6986b0804d252b8ea493ae3270
- **Files in Scope:**
 - BitcoinDepositor.sol
 - BitcoinRedeemer.sol
 - MezoAllocator.sol
 - PausableOwnable.sol
 - ITBTCToken.sol
 - IDispatcher.sol
 - ERC4626Fees.sol
 - stBTC.sol
 - Errors.sol

Documentation

- [Project Documentation in Coda](#)
- [Acre_Solidity_Smart_Contracts_20240410.pdf](#) (*Provided in Discord April 10, 2024*)



Executive Summary

Schedule

This security audit was conducted from April 11, 2024 to April 26, 2024 by 2 security auditors for a total of 5 person-weeks.

Overview

Thesis Defense conducted a manual code review of Acre Portal smart contracts implementation. The Acre protocol implements a liquid staking token for tBTC, represented as the stBTC token. Users can either bridge BTC on the Bitcoin chain via the tBTC bridge or directly deposit tBTC tokens in the ERC4626 compatible stBTC contract to receive stBTC tokens. Unlike rebasing tokens, stBTC accumulates generated rewards and yields in the token's value without altering the number of tokens held by each account. stBTC tokens are always redeemable for tBTC on Ethereum or BTC on the Bitcoin network.

Threat Model

The implementation of Acre allows for multiple potential attack vectors, which we have grouped into 3 different categories:

1. Attacks directly on the stBTC vault (inflation attack, rounding errors, bypassing fees, etc.)
2. Attacks on the tBTC bridging functionality (double mint, DOS of bridged assets, redeeming without burning)
3. Attacks on the allocation functionality (freezing of staked fund, stealing rewards, etc.)

To limit the attack surface, the contract owner and the deposit maintainers are assumed to be trusted actors controlled by a multisig.

As there is no KYC or whitelist required to use the protocol, any address on the Ethereum network can be considered a potential attacker. Nevertheless, multiple potential actors can be identified:

1. stBTC token holders
2. tBTC bridge users
3. Outside actors

Security by Design

Acre's stBTC vault is built upon battle-tested code, such as Openzeppelin's ERC4626 implementation and the ownable and pausable smart contracts. The widespread usage of these building blocks significantly reduces smart contract risk.

Utilizing pausing functionality additionally offers enhanced security, as the protocol can be paused in case of an ongoing exploit. It should be mentioned that this requires efficient and consistent off-chain monitoring of the protocol to ensure pausing can be done in a timely manner.

The well-tested tBTC bridge also adds an additional level of security. In the case of malicious bridging attempts, the permissioned guardians and off-chain scripts will swiftly intervene and refrain from settling the bridging attempt, preventing potential issues such as double mints.



Secure Implementation

The protocol has a high standard of code quality. The code is well-documented and follows best practices. In the case of invalid inputs, custom errors are emitted, which allows for easier transaction monitoring. Additionally, events are emitted on state-changing transactions, which again improves the monitoring capabilities.

Use of Dependencies

The protocol imports 2 libraries. To ensure their safety, they were checked using snyk.

Name	Version	Vulnerabilities
@openzeppelin	5.0.0	Out-of-bounds Read in Based64.encode
@keep-network/tbtc-v2	1.7.0	None

While the used OpenZeppelin version contains vulnerabilities, they do not affect the project as the vulnerable Base64 library is not used.

Tests

The Acre repository contains unit and integration tests for the smart contracts in the scope of this review, with ~93% line and ~84% branch coverage, in accordance with best practices.

During our review, we ran the Foundry property-based test suite for ERC4626 standard conformance, provided by a16z in the [erc4626-tests](#) repository. According to this test suite, the stBTC vault conforms to the ERC4626 standard. However, as mentioned in several issues in this audit report, this test suite is not comprehensive and does not replace custom tests tailored for the specific stBTC vault implementation. Nevertheless, we recommend integrating it into the development workflow as part of the GitHub action workflow.

Project Documentation

The smart contracts implemented by Acre are briefly documented in the [resources](#) provided by the client. However, we encourage the client to enhance this documentation with more details by describing the externally available functions of the protocol in detail. Additionally, it is recommended to provide flow chart diagrams of the various processes that take place after calls to the external functions.

In addition, we recommend making the documentation publicly available to help developers better understand the protocol's inner workings.



Key Findings Table

Issues	Severity	Status
ISSUE #1 Withdrawal Exit Fee Is Not Considered When Pulling Additional Funds From Mezo Portal Resulting in Temporarily Halted Withdrawals	Medium	Fixed
ISSUE #2 <code>depositorFee</code> Will Get Stuck in the Depositor Contract if <code>depositorFee >= tbtcAmount</code>	Medium	Fixed
ISSUE #3 Assets Can Be Stolen Due to Incorrect Implementation of <code>totalAssets</code>	Medium	Fixed
ISSUE #4 Virtual Shares Will Accrue Rewards and Still Allow for an Inflation Attack at a Loss	Low	Not Fixed
ISSUE #5 stBTC Tokens Can Be Transferred When the Contract Is Paused	Low	Not Fixed
ISSUE #6 <code>minDepositAmount</code> Restriction of the Depositor Is Never Enforced	Low	Not Fixed
ISSUE #7 ERC4626 Max Functions Must Return 0 if the Protocol Is Paused	Low	Fixed
ISSUE #8 <code>maxWithdraw</code> Function Does Not Factor in Fees	Low	Fixed
ISSUE #9 <code>stBTC.assetsBalanceOf</code> Does Not Take Fees Into Consideration	Low	Not Fixed
ISSUE #10 <code>BitcoinRedeemer</code> Does Not Check for the Amount Being a Multiple of <code>SATOSHI_MULTIPLIER</code>	Low	Not Fixed
ISSUE #11 <code>depositorFee</code> Always Rounds Down	Low	Fixed
ISSUE #12 Fees Are Not Checked for Being > Than 10.000 Basis Points	Low	Fixed
ISSUE #13 No Checks for <code>minimumDepositAmount</code> Being Lower Than <code>depositDustThreshold</code>	Low	Not Fixed
ISSUE #14 Missing Checks of Redeemer Address in Redeeming Process	None	Fixed
ISSUE #15 Missing Checks for Exorbitant Slippage on Calls to the Vault	None	Not Fixed



ISSUE #17 Multiple Missing Checks for <code>newVal == oldVal</code>	None	Fixed
ISSUE #18 <code>BitcoinDepositor</code> and <code>BitcoinRedeemer</code> Contracts Do Not Use the <code>zeroAddress</code> Error Provided in <code>Errors.sol</code>	None	Not Fixed
ISSUE #19 <code>DepositFinalized</code> Event Is Missing a Parameter for the Deposit Owner	None	Not Fixed
ISSUE #20 Documentation of <code>DepositFinalized</code> Event Is Missing the Referral Parameter	None	Fixed
ISSUE #21 Incorrect Documentation of <code>stBTC.approveAndCall</code>	None	Fixed
ISSUE #22 Misleading Documentation of the <code>MezoAllocator.addMaintainer</code> Function	None	Not Fixed
ISSUE #23 Private and Internal Functions Do Not Adhere to the Solidity Style Guide	None	Not Fixed
ISSUE #24 <code>Errors.sol</code> Is Missing Natspec Documentation	None	Fixed
ISSUE #25 Use of Floating Pragma	None	Fixed
ISSUE #26 Check in <code>BitcoinRedeemer.receiveApproval</code> can be simplified	None	Fixed
ISSUE #27 Spelling Error in the <code>BitcoinDepositor</code> Tests	None	Fixed

Severity definitions can be found in [Appendix A](#)



Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE#1

Withdrawal Exit Fee Is Not Considered When Pulling Additional Funds From Mezo Portal Resulting in Temporarily Halted Withdrawals

Medium

Fixed

Location

[BitcoinDepositor.sol#L269](#)

Description

The withdraw function in the `stBTC` contract withdraws tBTC tokens from the vault and transfers them to the receiver, in return for burning `stBTC` tokens. The `assets` function parameter specifies the exact amount of tBTC tokens to withdraw and to transfer to the receiver. Additionally, an exit fee is charged on top of this amount, which is subsequently transferred to the treasury.

As the deposited tBTC funds are likely to be allocated to the `MezoPortal` contract, the withdraw function checks if the `stBTC` contract has sufficient liquidity. If not, it withdraws the required amount of tBTC tokens from `MezoPortal`.

```
uint256 currentAssetsBalance = IERC20(asset()).balanceOf(address(this));
if (assets > currentAssetsBalance) {
    dispatcher.withdraw(assets - currentAssetsBalance);
}
```

However, as the exit fee is charged on top of assets, too few tBTC tokens are withdrawn from Mezo Portal, resulting in insufficient tokens to transfer to the treasury and the receiver. Consequently, withdrawals are temporarily halted until the `stBTC` contract has sufficient liquidity again from new deposits.

Impact

`stBTC` withdrawals are temporarily halted.

Recommendation

We recommend considering the exit fee in the liquidity check and withdrawing the required amount of tBTC tokens from `MezoPortal`, including the exit fee.

Verification Status

The Acre team has implemented the recommended remediation.



ISSUE#2

depositorFee Will Get Stuck in the Depositor Contract if depositorFee >= tbtcAmount

Medium

Not Fixed

Location

[BitcoinDepositor.sol#L269](#)

Description

Due to the depositor fee being imposed on the initial amount instead of the bridged `tbtcAmount`, the sum of all fees that are imposed on the bridged amount (`BTCTxFee + treasuryFee + optimisticMintingFee + depositorFee`) may be bigger than 100%. In that case, the following code block will revert whenever someone tries to finalize the deposit.

```
if (depositorFee >= tbtcAmount) {
    revert DepositorFeeExceedsBridgedAmount(depositorFee, tbtcAmount);
}
```

This will lead to the deposit never being finalizable. As the deposit still got minted due to sweeping or optimistic minting, the user will also never be able to recover it on the BTC chain. The funds can also never be fully transferred to the treasury to at least recover parts of the `depositorFee`. The result of this is that the full `tbtcAmount` will stay locked in the depositor forever.

Impact

As a result, the bridged `tbtcAmount` will stay stuck inside the depositor, and neither the treasury nor the user will be able to retrieve it.

Recommendation

We recommend finalizing the deposit in the above-mentioned scenario and transferring the full `tbtcAmount` to the treasury. This way, the depositor will at least be refunded for part of the depositor fee.

Verification Status

The Acre team stated that a situation where the depositor fee exceeds the bridged amount is considered unlikely, and that it is the responsibility of the governance to adjust the minimum deposit and the depositor fee parameters so that an initialized deposit will always be finalizable.



ISSUE#3

Assets Can Be Stolen Due to Incorrect Implementation of totalAssets

Medium

Fixed

Location

[MezoAllocator.sol#L244](#)

Description

As per the ERC4626 [standard](#), the `totalAssets` function should return “Total amount of the underlying asset that is “managed” by Vault.”. The current implementation calculates the `totalAssets` as `tBTC.balanceOf(stBTC) + dispatcher.totalAssets()` .

```
/// @notice Returns the total amount of assets held by the vault across all
/// allocations and this contract.
function totalAssets() public view override returns (uint256) {
    return
        IERC20(asset()).balanceOf(address(this)) + dispatcher.totalAssets();
}
```

The dispatcher returns its total assets as follows:

```
/// @notice Returns the total amount of tBTC allocated to MezoPortal.
function totalAssets() external view returns (uint256) {
    return depositBalance;
}
```

The returned number of assets only contains the tBTC tokens deposited in the `MezoPortal` , excluding donations and rewards that were sent directly to `MezoAllocator` . As a result, the `totalAssets` function returns an incorrect value. This leads to bigger issues down the road, as the `totalAssets` function is used to calculate the current conversion rates for shares and assets.

```
* @dev Internal conversion function (from assets to shares) with support for
rounding direction.
*/
function _convertToShares(uint256 assets, Math.Rounding rounding) internal view
virtual returns (uint256) {
    return assets.mulDiv(totalSupply() + 10 ** _decimalsOffset(), totalAssets() + 1,
rounding);
}
```

As this rate would be set to low, it would allow users to deposit and receive more shares than would correspond to their deposit if tokens were directly sent to the `MezoAllocator` . This could either happen due to donations or rewards being sent from future implementations of the `MezoPortal` contract.

In a scenario where assets were transferred to the `MezoAllocator` , the attacker will wait for the next call to `allocate` , which will add the unaccounted assets to the `totalAssets` and raise the conversion rate. The attacker can then sandwich calls to `MezoAllocator.allocate` with deposit and withdraw calls. As a result, the attacker will profit from some of the tokens sent to the `MezoAllocator` .

Impact

This issue can lead to incorrect values being returned by the `totalAssets` function. As a result, new deposits will receive more shares than intended, as some of the assets in the vault are not accounted for.

As the protocol is a staking provider, rewards might be donated to the `MezoAllocator` . These rewards would not be accounted for, leading to an incorrect conversion rate of shares/assets.



Recommendation

We recommend adapting the MezoAllocator's totalAssets function so that it adds the current balance of tBTC held in the contract to the depositBalance.

```
function totalAssets() external view returns (uint256) {  
    return depositBalance + tbtc.balanceOf(address(this));  
}
```

Verification Status

The Acre team fixed this issue in [#387](#).



ISSUE#4

Virtual Shares Will Accrue Rewards and Still Allow for an Inflation Attack at a Loss

Medium

Not Fixed

Location

[stBTC.sol#L24](#)

Description

The Acre protocol uses a version of the Openzeppelin ERC4626 implementation, which uses virtual shares to mitigate potential inflation attacks. The current implementation of virtual shares leads to two issues:

1. **Virtual shares will accrue some rewards** A tradeoff of this functionality is that as the virtual shares exist in the vault, they also incur some of the rewards that will be sent to the vault later.
2. **Virtual shares still allow for an inflation attack** The current implementation only uses one single virtual share. This prevents an attacker from profiting from the inflation attack but still allows him to damage other users while only incurring 1/4 of their damages.

Impact

The current implementation leads to the virtual share accruing some of the rewards as well as still allowing for an inflation attack at a loss.

Recommendation

We recommend to keep using virtual shares as they are a proven safeguard against inflation attacks. Nevertheless, the team should take caution in choosing the right configuration of the virtual shares as choosing many as well as few virtual shares has distinct tradeoffs. If the team goes with using 1 virtual share, the inflation attack will still be theoretically possible, but at a loss. If the team chooses to use more virtual shares (e.g., 10 or 100), the inflation attack becomes 10x or 100x less feasible, but these additional virtual shares will also incur rewards.

Verification Status

The Acre team stated that as inflation attack preventative measures, the protocol will:

1. define a minimum deposit limit of 0.001 tBTC,
2. use 1 virtual share build into OpenZeppelin ERC4626 library,
3. create an initial deposit of a minimum 0.001 tBTC upon contracts deployment



ISSUE#5

stBTC Tokens Can Be Transferred When the Contract Is Paused

✓ Low

✗ Not Fixed

Location

[stBTC.sol](#)

Description

The stBTC contract implements the `PausableOwnable` contract to enable the pauser role to pause the `deposit`, `mint`, `withdraw`, and `redeem` functions in the event of an emergency. However, the ERC-20 `transfer` and `transferFrom` functions lack the `whenNotPaused` modifier, resulting in those functions not being paused when the contract is paused.

Impact

stBTC token transfers can still occur when the contract is paused.

Recommendation

We recommend overriding the `ERC20Upgradeable._update` function and adding the `whenNotPaused` modifier to ensure that the ERC-20 `transfer` and `transferFrom` functions are also paused.

Verification Status

The Acre team stated that pausing affects only deposits and withdrawals from the stBTC Vault. Users should not be restricted from transferring their stBTC tokens.



ISSUE#6

minDepositAmount Restriction of the Depositor Is Never Enforced

✓ Low

✗ Not Fixed

Location

[BitcoinDepositor.sol#L65-L68](#)

Description

The `BitcoinDepositor` smart contract declares the `minDepositAmount` variable, which should be used to restrict the deposit of low-value transfers. However, this restriction is currently never enforced in the code.

Impact

As a result, any amount can be bridged, even if it is smaller than `minDepositAmount`. The only restriction is the minimum bridge amount.

Recommendation

We recommend enforcing the minimum deposit amount.

Verification Status

The Acre team stated that the `minDepositAmount` is expected to be used by the dApp and is not intended to be enforced on-chain.



ISSUE#7

ERC4626 Max Functions Must Return 0 if the Protocol Is Paused

✓ Low

✓ Fixed

Location

[stBTC.sol](#)

Description

According to the ERC4626 standard, if a protocol is paused, the max functions (`maxDeposit` , `maxMint` , `maxRedeem` , `maxWithdraw`) need to return 0. The standard states for each of the functions:

MUST factor in both global and user-specific limits, like if withdrawals are entirely disabled (even temporarily) it MUST return 0.

This is currently not done in `stBTC` , as the function will keep returning values greater than 0, even if the protocol is paused.

Impact

This issue leads to noncompliance with the intended standard. Additionally, users or smart contracts might try to interact with the paused contract due to the view returning misleading values.

Recommendation

We recommend adapting the ERC4626 max functions to return 0 when the protocol is paused.

Verification Status

The Acre team fixed this issue in [#389](#)



ISSUE#8

maxWithdraw Function Does Not Factor in Fees

✓ Low

☑ Fixed

Location

[stBTC.sol](#)

Description

The Acre protocol implements a modified version of ERC4626 that includes fees on deposits and withdrawals. Nevertheless, the functionality of the vault should comply with the requirements stated in the [ERC3526 standard](#). According to this standard, the max functions must include fees:

MUST return the maximum amount of assets that could be transferred from owner through withdraw and not cause a revert, which MUST NOT be higher than the actual maximum that would be accepted (it should underestimate if necessary).

In the current implementation, stBTC.maxWithdraw inherits the standard implementation of the maxWithdraw function from the OZ implementation.

```
/** @dev See {IERC4626-maxWithdraw}. */  
function maxWithdraw(address owner) public view virtual returns (uint256) {  
    return _convertToAssets(balanceOf(owner), Math.Rounding.Floor);  
}
```

However, this does not account for the fees imposed on withdrawals. As a result, it will return more assets than the user can withdraw. This could lead to users first calling maxWithdraw and then trying to withdraw the returned amount, which will fail due to the fees.

Impact

The issue leads to the `maxWithdraw` function returning an incorrect value, which causes incompatibility with the ERC4626 standard.

Recommendation

We recommend adding the fees to the calculation in the `maxWithdraw` function.

Verification Status

The Acre team fixed this issue in [#391](#)



ISSUE#9

stBTC.assetsBalanceOf Does Not Take Fees Into Consideration

✓ Low

✗ Not Fixed

Location

[stBTC.sol#L290](#)

Description

The `stBTC` contract adds the `assetsBalanceOf` function with the following function description:

```
// Returns value of assets that would be exchanged for the amount of shares owned by the account
```

However, the current implementation does not account for the fees that were implemented in `stBTC`.

Impact

This issue might cause user withdrawals to fail because they use this view function to calculate the maximum number of assets they are eligible to withdraw.

Recommendation

We recommend factoring in the fees before returning the actual balance of assets.

Verification Status

The Acre team stated that the function is not expected to account for the fees. The documentation has been improved in [#412](#).



ISSUE#10

BitcoinRedeemer Does Not Check for the Amount Being a Multiple of SATOSHI_MULTIPLIER

✓ Low

✗ Not Fixed

Location

[BitcoinRedeemer.sol#L149](#)

Description

When redeeming `stBTC` tokens for native BTC, the tBTC tokens are unminted in the `tbtcVault`. The amount bridged back to the Bitcoin chain is then calculated by calculating the amount divided by the `SATOSHI_MULTIPLIER`.

```
/// @dev amount MUST be divisible by SATOSHI_MULTIPLIER with no change.
function _unmintAndRedeem(
    address redeemer,
    uint256 amount,
    bytes calldata redemptionData
) internal {
    emit Unminted(redeemer, amount);
    tbtcToken.burnFrom(redeemer, amount);
    bank.approveBalanceAndCall(
        address(bridge),
        amount / SATOSHI_MULTIPLIER,
        redemptionData
    );
}
```

Impact

Due to this rounding, up to `SATOSHI_MULTIPLIER - 1` can be lost when bridging to the Bitcoin chain. This happens due to the tokens being fully burned, but only the result of the division being bridged.

Recommendation

We recommend verifying that `redeemedAmount % SATOSHI_MULTIPLIER == 0` whenever a user tries redeeming funds using the `BitcoinRedeemer`.

Verification Status

The Acre team stated that they expect the dApp to calculate the redeemed amount off-chain in a way the transaction won't leave the dust remainder.



ISSUE#11

depositorFee Always Rounds Down

✓ Low

☑ Fixed

Location

[BitcoinDepositor.sol#L263-L265](#)

Description

The BitcoinDepositor includes the depositorFee variable, which is a fee imposed on every deposited BTC to the bridge.

```
// transaction amount, before the tBTC protocol network fees were taken.
uint256 depositorFee = depositorFeeDivisor > 0
? (initialAmount / depositorFeeDivisor)
: 0;
```

The fee is calculated by dividing the bridged amount by the depositorFeeDivisor, which will automatically round down the fee. This is unintended behavior, as fees should always be rounded up to ensure no losses to the protocol.

Impact

Due to the incorrect rounding, unless the bridged amount is a multiple of the `depositorFeeDivisor`, a loss of 1 wei of fees will occur on every bridging attempt. Additionally, `depositorFeeDivisor - 1` tokens can be bridged without any fees.

Recommendation

We recommend rounding up instead of down for the depositor fee.

Verification Status

The Acre team fixed this issue in [#398](#).



ISSUE#12

Fees Are Not Checked for Being > Than 10.000 Basis Points

✓ Low

☑ Fixed

Location

[stBTC.sol#L148](#)

[stBTC.sol#L158](#)

Description

The `stBTC` vault implements fees on deposits and withdrawals. These fees are set in basis points, which are 1/100 of a percent.

```
/// @notice Update the entry fee basis points.
/// @param newEntryFeeBasisPoints New value of the fee basis points.
function updateEntryFeeBasisPoints(
    uint256 newEntryFeeBasisPoints
) external onlyOwner {
    entryFeeBasisPoints = newEntryFeeBasisPoints;

    emit EntryFeeBasisPointsUpdated(newEntryFeeBasisPoints);
}

/// @notice Update the exit fee basis points.
/// @param newExitFeeBasisPoints New value of the fee basis points.
function updateExitFeeBasisPoints(
    uint256 newExitFeeBasisPoints
) external onlyOwner {
    exitFeeBasisPoints = newExitFeeBasisPoints;

    emit ExitFeeBasisPointsUpdated(newExitFeeBasisPoints);
}
```

The fees should be a maximum of 100% and never exceed 10,000 basis points. However, this upper bound is currently not checked.

Impact

As a result, the fees might accidentally be set to a value above 100%, which would lead to the fees either being deducted from other user's funds or the call reverting.

Recommendation

We recommend checking if fees are $\leq 10_000$ and reverting otherwise. Additionally, we recommend introducing a maximum upper bound below 100% to increase user trust in the protocol. The maximum fee could, for example, be set to 5%, which would ensure users that the fees can never exceed 5%.

Verification Status

The Acre team fixed this issue in [#392](#).



ISSUE#13

No Checks for `minimumDepositAmount` Being Lower Than `depositDustThreshold`

✓ Low

✗ Not Fixed

Location

[stBTC.sol#L116](#)

Description

The Acre protocol implements a minimum deposit restriction in stBTC. The `minimumDepositAmount` variable is described as follows:

```
/// Minimum amount for a single deposit operation. The value should be set
/// low enough so the deposits routed through Bitcoin Depositor contract won't
/// be rejected. It means that minimumDepositAmount should be lower than
/// tBTC protocol's depositDustThreshold reduced by all the minting fees taken
/// before depositing in the Acre contract.
uint256 public minimumDepositAmount;
```

However, when the variable is set, it is never checked if the variable is lower than `depositDustThreshold`.

```
/// @notice Updates minimum deposit amount.
/// @param newMinimumDepositAmount New value of the minimum deposit amount. It
/// is the minimum amount for a single deposit operation.
function updateMinimumDepositAmount(
    uint256 newMinimumDepositAmount
) external onlyOwner {
    minimumDepositAmount = newMinimumDepositAmount;

    emit MinimumDepositAmountUpdated(newMinimumDepositAmount);
}
```

Impact

As a result, it might happen that some deposits through the bridge can never be finalized due to the deposit amount being lower than `minimumDepositAmount`.

Recommendation

We recommend calling the `bridge.depositParameters` function and verify if the `newMinimumDepositAmount` is smaller than the `depositDustThreshold`. It needs to be noted that this does not account for the variable fees. So, it is still possible that bridged deposits get stuck, but this reduces the risk.

Verification Status

The Acre team stated that it is the governance's responsibility to adjust protocol parameters to the correct values and synchronize them across the contracts. No additional checks will be enforced to avoid a need to introduce a reference to the Bridge contract.



Missing Checks of Redeemer Address in Redeeming Process

None

Fixed

Location

[BitcoinRedeemer.sol#L152](#)

Description

Whenever a user wants to redeem `stBTC` for native BTC, the user has to provide the `extraData` parameter, which is structured as follows:

```
[
  address redeemer,
  bytes20 walletPubKeyHash,
  bytes32 mainUtxoTxHash,
  uint32 mainUtxoTxOutputIndex,
  uint64 mainUtxoTxOutputValue,
  bytes redeemerOutputScript
]
```

The redeemer parameter of this struct is described in the comments of the bridge (which parses the data):

```
/// @param redeemer The Ethereum address of the redeemer who will be able to
///     claim Bank balance if anything goes wrong during the redemption.
///     In the most basic case, when someone redeems their Bitcoin
///     balance from the Bank, `balanceOwner` is the same as `redeemer`.
///     However, when a Vault is redeeming part of its balance for some
///     redeemer address (for example, someone who has earlier deposited
///     into that Vault), `balanceOwner` is the Vault, and `redeemer` is
///     the address for which the vault is redeeming its balance to.
```

If something goes wrong while bridging the tokens, they will be credited to the redeemer address. However, in the current implementation, it is never verified if this is an invalid address (e.g., the zero address). If the user accidentally leaves the parameter at 0, all tokens will be lost if bridging fails.

Impact

None – no security impact.

Recommendation

The recommended minimal mitigation for this is to check if this variable is 0 and revert in that case. A more advanced, but also more invasive, mitigation would be to set this variable automatically to the current owner of the tokens. This way, tokens would always be refunded to the current owner in the case of issues while bridging.

Verification Status

The Acre team fixed this issue in [#399](#).



ISSUE#15

Missing Checks for Exorbitant Slippage on Calls to the Vault

None

Not Fixed

Location

[stBTC.sol](#)

Description

The Acre protocol implements an ERC4626 vault as the base of the stBTC token. This vault allows deposits and redemptions of the underlying asset, tBTC, in return for stBTC tokens. Due to rounding, it can occur that a user deposits assets and receives 0 shares, or a user withdraws shares but receives 0 shares. In the current implementation, the user is not protected against this case, so the user may experience a slippage of 100% in the case of rounding.

Impact

None – no security impact.

Recommendation

To protect unknowing users, we recommend adding a simple check of the return value of the `deposit`, `mint`, `redeem`, and `withdraw` functions. If the functions return 0, which indicates 100% slippage, the call should revert.

Verification Status

The Acre team states that considering all of the measures added for [Issue 4](#) it will be impossible to return 0 shares to depositors.



ISSUE#16

Missing Check for Ownership of tBTC Token When Updating The tbtcVault Address

None

Fixed

Location

[BitcoinRedeemer.sol#L125](#)

Description

The `updateTbtcVault` function updates the `tbtcVault` address, which is supposed to be the owner of the tBTC smart contract. However, this setter function only validates that the new address is not the zero address. As a result, should the new address not be the owner of the tBTC contract, the `redeemSharesAndUnmint` function would revert in line 155 due to the mismatching tBTC token owner, preventing redemptions.

Impact

None – no security impact.

Recommendation

We recommend adding a check to ensure that the new `tbtcVault` address is the owner of the tBTC token contract.

Verification Status

The Acre team fixed this issue in [#415](#).



ISSUE#17

Multiple Missing Checks for `newVal == oldVal`

None

Fixed

Location

[stBTC.sol#L100](#)

[stBTC.sol#L127](#)

[PausableOwnable.sol#L94](#)

Description

Multiple setter functions do not check if the newly set value is the same as the old one. This could lead to irrelevant logs being emitted and make monitoring harder.

Impact

None – no security impact.

Recommendation

We recommend adding a `require` statement similar to the following one:

```
require(oldValue != newValue, "Values are the same");
```

Verification Status

The Acre team fixed this issue in [#393](#).



ISSUE#18

BitcoinDepositor and BitcoinRedeemer Contracts Do Not Use the zeroAddress Error Provided in Errors.sol

None

Not Fixed

Location

[BitcoinDepositor.sol#L120](#)

[BitcoinDepositor.sol#L123](#)

Description

The Acre protocol includes the `Errors.sol` file. This file only includes the `zeroAddress` error, which is used throughout the protocol. However, the `BitcoinDepositor` and `BitcoinRedeemer` contracts do not use this custom error and instead implement their own errors for zero addresses.

Impact

None – no security impact.

Recommendation

We recommend either using the provided error in all contracts or adding the additional errors to the `Errors.sol` file.

Verification Status

The Acre team stated that the contracts use specific errors that contain the name of the parameter that is incorrect to reduce confusion, when one function validates multiple addresses.



ISSUE#19

DepositFinalized Event Is Missing a Parameter for the Deposit Owner

None

Not Fixed

Location

[BitcoinDepositor.sol#L280](#)

Description

The `DepositFinalized` event is emitted whenever a deposit through the `BitcoinDepositor` is finalized. It consists of the following parameters:

```
event DepositFinalized(  
    uint256 indexed depositKey,  
    address indexed caller,  
    uint16 indexed referral,  
    uint256 initialAmount,  
    uint256 bridgedAmount,  
    uint256 depositories  
);
```

In the current implementation, the `finalizeDeposit` function can be called by any address. As a result, the caller, who might not be the owner, will be emitted, but the event does not include to which address the tokens were sent. This makes monitoring and possible exploit detection harder once the contract is deployed.

Impact

None – no security impact.

Recommendation

We recommend adding the `depositOwner` variable to the emitted event.

Verification Status

The Acre team stated that the deposit owner is included in the Deposit event emitted by the ERC4626Upgradable contract emitted in the same transaction.



ISSUE#20

Documentation of DepositFinalized Event Is Missing the Referral Parameter

None

Fixed

Location

[BitcoinDepositor.sol#L93-L108](#)

Description

When a deposit gets finalized in the `BitcoinDepositor`, the `DepositFinalized` event is emitted. The documentation of the event is as follows:

```
/// @notice Emitted when a deposit is finalized.
/// @dev Deposit details can be fetched from {{ ERC4626.Deposit }}
///      event emitted in the same transaction.
/// @param depositKey Deposit key identifying the deposit.
/// @param caller Address that finalized the deposit.
/// @param initialAmount Amount of funding transaction.
/// @param bridgedAmount Amount of tBTC tokens that was bridged by the tBTC
bridge.
/// @param depositorFee Depositor fee amount.
event DepositFinalized(
    uint256 indexed depositKey,
    address indexed caller,
    uint16 indexed referral,
    uint256 initialAmount,
    uint256 bridgedAmount,
    uint256 depositorFee
);
```

However, this documentation is missing the referral parameter.

Impact

None – no security impact.

Recommendation

We recommend adding NatSpec documentation for the `referral` parameter.

Verification Status

The Acre team fixed this issue in [#412](#).



ISSUE#21

Incorrect Documentation of 'stBTC.approveAndCall'

None

Fixed

Location

[stBTC.sol#L180](#)

Description

The Acre protocol implements the `stBTC.approveAndCall` function to allow for a token spending approval as well as a subsequent call in the same call. The function comment states:

```
/// @dev If the `amount` is set to `type(uint256).max` then
///      `transferFrom` and `burnFrom` will not reduce an allowance.
function approveAndCall(
    address spender,
    uint256 value,
    bytes memory extraData
```

The function does not include an amount parameter; instead, the parameter is called `value`.

Impact

None – no security impact.

Recommendation

We recommend updating the comment to use `value` instead of `amount`.

Verification Status

The Acre team fixed this issue in [#404](#).



ISSUE#22

Misleading Documentation of the `MezoAllocator.addMaintainer` Function

None

Not Fixed

Location

Description

The Acre protocol implements the `MezoAllocator.addMaintainer` function to allow for the addition of new maintainers that manage the tBTC allocation. The function is documented as follows:

```
/// @notice Updates the maintainer address.
/// @param maintainerToAdd Address of the new maintainer.
function addMaintainer(address maintainerToAdd) external onlyOwner {
    if (maintainerToAdd == address(0)) {
        revert ZeroAddress();
    }
    if (isMaintainer[maintainerToAdd]) {
        revert MaintainerAlreadyRegistered();
    }
    maintainers.push(maintainerToAdd);
    isMaintainer[maintainerToAdd] = true;

    emit MaintainerAdded(maintainerToAdd);
}
```

However, this comment is slightly misleading as it indicates that there is only a single maintainer address that is replaced whenever a new maintainer is added.

Impact

None – no security impact.

Recommendation

We recommend updating the comment to “Adds a new maintainer address”.

Verification Status

The Acre team fixed this issue in [#404](#).



ISSUE#23

Private and Internal Functions Do Not Adhere to the Solidity Style Guide

None

Not Fixed

Location

[BitcoinRedeemer.sol#L149](#)

Description

To ensure good code readability and prevent future issues, it is highly recommended that Solidity code follow the Solidity Style Guide. The style guide states that non-external functions should be prefixed with an underline. This currently needs to be implemented for one function used in the protocol.

Impact

None – no security impact.

Recommendation

We recommend prefixing the private and internal functions accordingly.

Verification Status

The Acre team fixed this issue in [#416](#).



ISSUE#24

Errors.sol Is Missing Natspec Documentation

None

Fixed

Location

Errors.sol

Description

The Acre protocol implements a special file for common errors that are used throughout the different contracts. The file is called `Errors.sol`. However, the current version is missing all NatSpec documentation.

Impact

None – no security impact.

Recommendation

We recommend adding NatSpec documentation to `Errors.sol`.

Verification Status

The Acre team fixed this issue in [#404](#).



ISSUE#25

Use of Floating Pragma

None

Fixed

Location

[ITBTCToken.sol#L2](#)

[IDispatcher.sol#L2](#)

[ERC4626Fees.sol#L5](#)

[Errors.sol#L3](#)

[BitcoinDepositor.sol#L2](#)

[BitcoinRedeemer.sol#L2](#)

[MezoAllocator.sol#L2](#)

[PausableOwnable.sol#L2](#)

[stBTC.sol#L2](#)

Description

It is considered best practice to deploy contracts with the same compiler version and flags that they have been tested with. Locking the pragma helps ensure that contracts do not accidentally get deployed using an outdated compiler version, for example, that might introduce bugs that negatively affect the contract system.

Impact

None – no security impact.

Recommendation

We recommend locking the pragma version in the audited contracts and considering known bugs in the compiler version chosen.

Verification Status

The Acre team fixed this issue in [#406](#).



Check in `BitcoinRedeemer.receiveApproval` can be simplified

None
Fixed

Location

[BitcoinRedeemer.sol#L109-L110](#)

Description

To initiate the redeeming process, the user interacts with the `BitcoinRedeemer` contract through the `stBTC.approveAndCall` functionality, which subsequently calls the `receiveApproval` function in the `BitcoinRedeemer` contract.

```
function receiveApproval(
    address from,
    uint256 amount,
    address token,
    bytes calldata extraData
) external {
    if (token != address(stbtc)) revert UnsupportedToken(token);
    if (msg.sender != token) revert CallerNotAllowed(msg.sender);
    if (extraData.length == 0) revert EmptyExtraData();

    redeemSharesAndUnmint(from, amount, extraData);
}
```

This function validates the given parameters via multiple if statements. However, the first two checks can be merged. If the `msg.sender` is the `stBTC` contract, it will always call `receiveApproval` with itself set as the token parameter.

```
function approveAndCall(
    address spender,
    uint256 value,
    bytes memory extraData
) external returns (bool) {
    if (approve(spender, value)) {
        IReceiveApproval(spender).receiveApproval(
            msg.sender,
            value,
            address(this),
            extraData
        );
        return true;
    }
    return false;
}
```

Impact

None – no security impact.

Recommendation

We recommend merging the first two if statements, as shown in the following code snippet:



```
function receiveApproval(
  address from,
  uint256 amount,
  address token,
  bytes calldata extraData
) external {
  if (msg.sender != address(stbtc)) revert CallerNotAllowed(msg.sender);
  if (extraData.length == 0) revert EmptyExtraData();

  redeemSharesAndUnmint(from, amount, extraData);
}
```

Verification Status

The Acre team fixed this issue in [#405](#).

ISSUE#27

Spelling Error in the BitcoinDepositor Tests

None

Fixed

Location

[BitcoinDepositor.test.ts#L751](#)

Description

The Bitcoin depositor test case includes a spelling error, as the `expectedDepositOwner` is spelled as `expectedDepositOwner` .

Impact

None – no security impact.

Recommendation

We recommend fixing the spelling error and considering integrating an automated spell checker (e.g., CSpell) in the development process.

Verification Status



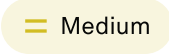
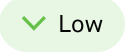
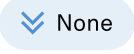
The Acre team fixed this issue in [#405](#).



Appendix A

Severity Rating Definitions

At Thesis Defense, we utilize the [Immunefi Vulnerability Severity Classification System - v2.3](#).

Severity	Definition
 Critical	<ul style="list-style-type: none"> • Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results • Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield • Permanent freezing of funds • Predictable or manipulable RNG that results in abuse of the principal • Protocol insolvency
 High	<ul style="list-style-type: none"> • Theft of unclaimed yield • Theft of unclaimed royalties • Permanent freezing of unclaimed yield • Permanent freezing of unclaimed royalties • Temporary freezing of funds • Temporary freezing NFTs
 Medium	<ul style="list-style-type: none"> • Smart contract unable to operate due to lack of token funds • Enabling/disabling notifications • Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol) • Theft of gas • Unbounded gas consumption
 Low	<ul style="list-style-type: none"> • Contract fails to deliver promised returns, but doesn't lose value
 None	<ul style="list-style-type: none"> • We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.



Appendix B

Thesis Defense Disclaimer

Thesis Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project's security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user's convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.

