Audit Report

CosmWasm WBTC

v1.0

August 18, 2023

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Osmosis Grants Company to perform a security audit of the CosmWasm WBTC contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/osmosis-labs/cw-wbtc
Commit	`f3aa39fdea5e2b5c545decfa7e02a62afb41b12c`
Scope	All contracts were in scope.
Fixes verified at commit	`876f865e0b06e519384808efe84ee906d8746c05`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The audit features the CosmWasm WBTC protocol, in which WBTC is minted following the amount of BTC stored in a Bitcoin blockchain's custody wallet. WBTC tokens will be created using the token factory module in the Osmosis chain. Access to the protocol is restricted to the following privileged roles: governor, member manager, custodian, and merchants.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	Detailed documentation was provided in the README file.
Test coverage	Medium-High	`cargo tarpaulin` reports a `75.67%` test coverage on the audited commit hash. The client extended the test coverage to `97.43%` on commit hash `876f865e0b06e519384808efe84ee906d8746c05`.

Summary of Findings

No	Description	Severity	Status
1	Removing merchant does not remove associated deposit addresses	Minor	Resolved
2	Approving mint requests does not ensure the requestor still holds the merchant role	Minor	Resolved
3	Incorrect governor attribute value emitted	Minor	Resolved
4	Merchant deposit addresses are not ensured to be unique	Minor	Resolved
5	Separation of privileged addresses is not enforced	Minor	Resolved
6	Attributes are not properly emitted during contract instantiation	Informational	Resolved
7	`GetMintRequestsCount` and `GetBurnRequestsCount` query fails when there are no mint or burn requests	Informational	Resolved
8	Contracts should implement a two-step ownership transfer	Informational	Resolved
9	Typographical error affects codebase readability	Informational	Resolved
10	Storage entries spread across multiple files increase the chances of storage key collision	Informational	Resolved

Detailed Findings

1. Removing merchant does not remove associated deposit addresses

Severity: Minor

In `contracts/wbtc-controller/src/auth/merchant.rs:45-66`, the `remove_merchant` function does not remove the merchant's associated deposit addresses from the `CUSTODIAN_DEPOSIT_ADDRESS_PER_MERCHANT` and `MERCHANT_DEPOSIT_ADDRESS` storage. When a merchant is removed, they should no longer have the associated roles assigned. However, the `GetCustodianDepositAddress` and `GetMerchantDepositAddress` queries will show that the removed merchant still has a valid deposit address on the Bitcoin blockchain.

Recommendation

We recommend removing the merchant's deposit addresses from the `CUSTODIAN_DEPOSIT_ADDRESS_PER_MERCHANT` and `MERCHANT_DEPOSIT_ADDRESS` storage when executing the `remove_merchant` function.

Status: Resolved

2. Approving mint requests does not ensure the requestor still holds the merchant role

Severity: Minor

In `contracts/wbtc-controller/src/contract.rs:149-189`, when approving a pending mint request, no validation ensures that the requester still holds the merchant role. For example, consider a scenario where the member manager revokes the merchant role post the issuance of a mint request. The mint requester, no longer holding the merchant role, should not be permitted to receive WBTC. However, due to a lack of validation when the custodian approves the mint request, the requester can still receive WBTC even though they no longer hold the merchant role.

We classify this issue as minor because the custodian is considered a privileged and trusted role.

Recommendation

We recommend adding a validation step to ensure the requester still holds the merchant role before approving a mint request.

Status: Resolved

3. Incorrect governor attribute value emitted

Severity: Minor

In `contracts/wbtc-controller/src/contract.rs:58`, the governor attribute value is emitted as `info.sender`. This is incorrect because the governor's address is set to `msg.governor`, causing off-chain event listeners and blockchain indexers to record incorrect data.

Recommendation

We recommend emitting the governor's address as the `msg.governor` value.

Status: Resolved

4. Merchant deposit addresses are not ensured to be unique

Severity: Minor

In `contracts/wbtc-controller/src/tokenfactory/deposit_address.rs:67-91`, the `set_custodian_deposit_address` function does not ensure that the provided `deposit_address` is unique across merchants. This could lead to a custodian mistakenly assigning the same `deposit_address` to multiple merchants. A malicious merchant could exploit this if the custodian unsuspectingly assigns an existing address to the malicious merchant. The malicious merchant could then issue duplicate mint requests with a transaction identifier of the duplicate address.

Assume the custodian sets the same deposit address for two merchants. The first merchant sends BTC to the deposit address. The second merchant did not make a deposit, but they took the first merchant's transaction identifier and issued a mint request for themselves. Since the deposit address for the second merchant matches the transaction identifier's receiver address, the custodian might approve this deceptive mint request.

We classify this issue as minor because it requires a misconfiguration by the custodian, which is a privileged and trusted role.

Recommendation

We recommend implementing a mechanism to ensure each `deposit_address` given to merchants is unique. One possible solution is to create an extra storage field in the form of a `Map<String, bool>`. This would store the deposit address as the key.

When setting the custodian's deposit address, the `set_deposit_address` function should load this map using the deposit address to verify if it has been previously used. If it has, the transaction should be reverted with an error message indicating a duplicate deposit address.

Status: Resolved

5. Separation of privileged addresses is not enforced

Severity: Minor

While extensive effort has been made to define privileged access roles within the protocol, privilege separation of these roles is not properly enforced. Currently, no validation exists to enforce that these privileges are designated to unique addresses. In the unlikely case of a compromised account, privilege sharing will have severe implications. While this measure cannot prevent account compromise, it can limit the impact and serve as an additional layer of security for the protocol.

Another step that can be taken to reduce the impact of a compromised account is to implement timelocks to prevent frequent updates of addresses.

Recommendation

We recommend adding validation to prevent privileged address reuse and consider implementing timelocks following the update of these privileged addresses.

Status: Resolved

6. Attributes are not properly emitted during contract instantiation

Severity: Informational

During the contract instantiation process, the `initialize_governor` function in `contracts/wbtc-controller/src/auth/governor.rs:16` returns attributes, but they are not used by the calling function in `contracts/wbtc-controller/src/contract.rs:44`.

Consequently, the attributes will not be properly emitted during instantiation.

Recommendation

We recommend adding these attributes to the response of the `instantiate` function.

Status: Resolved

7. `GetMintRequestsCount` and `GetBurnRequestsCount` query fails when there are no mint or burn requests

Severity: Informational

In `contracts/wbtc-controller/src/tokenfactory/nonce.rs:30-33`, the custom `get` function returns the stored nonce value in the storage using the `load` function. Since the `load` function fails when there is no previously stored value, the

`GetMintRequestsCount` and `GetBurnRequestsCount` queries will fail when no mint or burn requests are issued.

Recommendation

We recommend using `unwrap_or_default()` so the query defaults to 0 instead of failing.

Status: Resolved

8. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Resolved

9. Typographical error affects codebase readability

Severity: Informational

In `contracts/wbtc-controller/src/tokenfactory/deposit_address.rs:12`, the wrongly typed `DepositAddressMananger` occurs, which should be `DepositAddressManager`.

While the typography error doesn't affect the code functionality, it increases the chances of introducing mistakes that can be easily avoided.

Recommendation

We recommend correcting the typographic error as described above.

Status: Resolved

10. Storage entries spread across multiple files increase the chances of storage key collision

Severity: Informational

The contract contains multiple storage entries that are defined in separate files. While this does not affect the code quality, it increases the likelihood of accidental storage key collisions. One common pattern is consolidating all storage entries into a single file, increasing code readability and reducing the likelihood of storage key collisions.

Recommendation

We recommend consolidating all storage entries into a single file.

Status: Resolved

Threat Model Analysis

Methodology

Process Applied

The process performed to analyze the system for potential threats and build a comprehensive model is based on the approach first pioneered by Microsoft in 1999 that has developed into the STRIDE model (https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20).
Whilst STRIDE is aimed at traditional software systems, it is generic enough to provide a threat classification suitable for blockchain applications with little adaptation (see below).
The result of the STRIDE classification has then been applied to a risk management matrix with simple countermeasures and mitigations suitable for blockchain applications.

STRIDE Interpretation in the Blockchain Context

STRIDE was first designed for closed software applications in permissioned environments with limited network capabilities. However, the classification provided can be adapted to blockchain systems with small adaptations. The below table highlights a blockchain-centric interpretation of the STRIDE classification:

Spoofing	In a blockchain context, the authenticity of communications is built into the underlying cryptographic public key infrastructure. However, spoofing attack vectors can occur at the off-chain level and within a social engineering paradigm. An example of the former is a Sybil attack where an actor uses multiple cryptographic entities to manipulate a system (wash-trading, auction smart contract manipulation, etc.). The latter usually consists of attackers imitating well-known actors, for instance, the creation of an impersonation token smart contract with a malicious implementation.
Tampering	Similarly to spoofing, tampering of data is usually not directly relevant to blockchain data itself due to cryptographic integrity. One example of this is a supply chain attack that manages to inject malicious code or substitute trusted software that interacts with the blockchain (node software, wallets, libraries).
Repudiation	Repudiation, i.e. the ability of an actor to deny that they have taken action is usually not relevant at the transaction level of blockchains. However, it makes

	sense to maintain this category, since it may apply to additional software used in blockchain applications, such as user-facing web services. An example is the claim of a loss of a private key and hence assets.
Information Disclosure	Information disclosure has to be treated differently at the blockchain layer and the off-chain layer. Since the blockchain state is inherently public in most systems, information leakage here relates to data that is discoverable on the blockchain, even if it should be protected. Predictable random number generation could be classified as such, in addition to simply storing private data on the blockchain. In some cases, information in the mempool (pending/unconfirmed transactions) can be exploited in front-running or sandwich attacks. At the off-chain layer, the leakage of private keys is a good example of operational threat vectors.
Denial of Service	Denial of service threat vectors translates directly to blockchain systems at the infrastructure level. At the smart contract or protocol layer, there are more subtle DoS threats, such as unbounded iterations over data structures that could be exploited to make certain transactions not executable.
Elevated Privileges	Elevated privilege attack vectors directly translate to blockchain services. Faulty authorization at the smart contract level is only one example where users might obtain access to functionality that should not be accessible.

Assets

The following describes assets that may be valuable to an attacker or other stakeholders of the system.

Privileged roles

The defined messages in the contract can only be executed from privileged roles: the governor, member manager, custodian, and merchants. These accounts present an attractive target for attackers because each role is authorized to execute messages that may affect the circulating supply of WBTC tokens.

WBTC token

The WBTC token is created from the `x/tokenfactory` module, which is backed by Bitcoin on the Bitcoin blockchain. It is a native token type that holds the equivalent value of a BTC token, and it can be traded on the market, such as decentralized exchanges, or extract value in various ways, making it an attractive target for attackers.

Custody wallet

BitGo holds BTC in a custody wallet and acts as backing for WBTC minted by protocol. The WBTC/BTC peg relies on the security of this wallet. If an attacker managed to gain access to it and steal funds, the invariant that 1 BTC always backs 1 WBTC would no longer hold, causing WBTC to lose value.

Modules of the underlying blockchain

The architecture of the protocol relies on the `x/bank` and `x/tokenfactory` modules of the Osmosis chain. The former manages the circulating token supply as well as transfers, while the latter handles authentication for minting and burning WBTC. These modules are an attractive target to attackers as an unexpected vulnerability might allow them to mint, burn or take possession of WBTC tokens.

Private keys controlling external accounts

In the case of externally owned accounts, some entity holds a private key in the control of that account. An attacker finds access to a private key valuable since that gives them full control over the account, allowing them to impersonate and act on behalf of the account owner.

Migration authorization

In CosmWasm smart contracts, if there is a defined contract admin, the contract is not immutable. Gaining unauthorized access to act as the contract admin is valuable to an attacker, as it allows them to modify the contract's source code, potentially introducing backdoors or other malicious code.

Liveness of the protocol

An attacker may find it valuable to target the liveness of the protocol, such as preventing the minting and burning of WBTC tokens. This could allow the attackers to disrupt issuing of new tokens, delay trading, prevent tokens from being burned, or extort stakeholders to stop the attack.

Liveness of the underlying blockchains

Attacking the liveness of underlying blockchains can also be valuable to an attacker. By doing so, they may profit from shorting the tokens or extorting stakeholders to stop the attack, causing disruptions and instability in the blockchain ecosystem. Osmosis block times are dependent on the validator set coming to a consensus. Several factors may impact this, such as consensus failures that stem from building a block or issues that stem from the ABCI operations that may slow or halt the chain.

Blockchain light nodes

Many actors choose not to run a full blockchain node, mainly due to its resource demands. As an alternative, they may run a light node and rely on other full nodes to obtain relevant information. The custodian might opt for such a light node to determine whether BTC was deposited successfully to the merchant deposit address.

Unlike full nodes, light nodes do not validate whether all transactions comply with Bitcoin's consensus rules. Instead, the connected full nodes are trusted to validate them. Due to that, light clients are vulnerable to Eclipse attacks, where an attacker that controls all connected nodes is able to feed misinformation to the light node, particularly transactions that do not adhere to the consensus rules.

For example, spending a transaction output twice can lead to the issuance of unbacked WBTC. For this attack to be successful, the attacker would need to mine the block with the double-spend and then mine enough blocks to meet the confirmation thresholds of the custodian. While this is quite expensive in practice on the Bitcoin blockchain, it might be economically viable.

Blockchain consensus

Blockchain consensus protocols establish agreement among network participants regarding transaction validity and order, ensuring trust and security within the network. Suppose an attacker gains control over the Bitcoin network’s consensus via a 51% attack or the Cosmos SDK chain (either by compromising validator nodes or coordinating collusion). In that case, the integrity of the WBTC protocol could be compromised. The economic feasibility of such an attack grows as the total value locked in the protocol increases. This is problematic especially if one of the chains has a relatively low market capitalization/low staking ratio.

Stakeholders/Potential Threat Actors

WBTC token holders

The main users of the protocol transact with WBTC tokens, which they can exchange, trade on decentralized exchanges, use as collateral in lending protocols, or engage in other DeFi activities. However, rational users are expected to prioritize profit-making and prevent losses over the security and integrity of the protocol as well as the stability of the value of the WBTC token.

For example, if a vulnerability arises that causes WBTC to no longer be backed by legitimate Bitcoins, these users would likely rush to sell their WBTC to minimize losses. In fact, even the rumor of a vulnerability could cause such an event. This could cause a sharp WBTC price drop, potentially destabilizing the protocol and even affecting the broader DeFi market.

Governor account

The governor is the highest privileged account in the protocol. This account can grant the member manager role to any address, pausing or unpausing the protocol state and modifying the token metadata.

If an attacker gained access to this role, they could mint arbitrary WBTC from the contract by manipulating the member manager account to grant merchant and custodian ownership to their controlled address. Using both accounts, an attacker can request to mint a large amount of WBTC and approve it with the custodian account.

Such an attack could be very profitable, as the attacker could sell all of the freshly minted WBTC and deplete the liquidity of AMMs. If there were bridging solutions for WBTC, the impact would be even worse, as an attacker could also deplete the liquidity of AMMs on other chains by bridging the freshly minted tokens to other chains.

Member manager account

The member manager account has the ability to set the custodian address and add or remove merchants. As the member manager account controls both merchants and the custodian, an attacker that obtains unauthorized access to this role can create a merchant account to request minting a large amount of WBTC and manipulate the custodian account to approve it, thereby severely inflating the token supply and disrupting the market equilibrium.

Merchant accounts

There is a finite set of merchants which are assumed to be known, as they must be explicitly added by the member manager. Merchants can issue new mint or burn requests which need to be approved by the custodian. Moreover, they are responsible for sending BTC to the BitGo custody wallet before creating a mint request with the transaction identifier.

The role's privileges are limited, so an attacker could not abuse them directly to steal tokens if they gained access to the private keys. However, a loss of funds might occur if the custodian approves a pending mint request initiated by the previous merchant, allowing the attacker to steal the minted WBTC in the merchant account. The attacker can also configure their Bitcoin address using the `SetMerchantDepositAddress` message to receive BTC in the blockchain after the custodian has refunded them.

Custodian account

The custodian account has the ability to approve and reject mint requests, confirm burn requests, configure minimum burn amounts, and configure Bitcoin deposit addresses for merchants. When a merchant submits a mint request, the custodian must ensure the provided transaction identifier's deposit address aligns with the unique Bitcoin deposit address they've set. If not, the request will be rejected.

Similar to the merchants, the limited privileges effectively prevent attackers from directly stealing tokens if they gain access to the private keys. With that said, a compromised custodian could still divert funds by setting merchant deposit addresses to Bitcoin addresses under their control, allowing them to receive BTC directly. They could also execute a denial of service attack by setting the minimum burn amount to an excessively high value or rejecting all mint requests issued by merchants.

Additionally, the compromised custodian account can also collude with a merchant to mint an excessive amount of WBTC, potentially risking the protocol’s solvency.

Custody wallet holder

All WBTC minted by the protocol must have an associated 1:1 backing in the custody wallet created by BitGo. The custody wallet holder has control over the BTC inside that wallet, which directly influences the stability of the WBTC/BTC peg.

Contract admins

Contract admins have the power to migrate contracts. They may try to migrate contracts to a malicious codebase, introduce vulnerabilities and backdoors, or designate privileged addresses within the system. When the contract is instantiated, the caller can provide an optional admin address with the power to perform contract migration.

Validators

Nodes are responsible for verifying and validating transactions and adding them to the blockchain to produce blocks. Validators may get involved in blocking or censoring transactions, disrupting the normal functioning of the system. They can also re-order transactions, potentially running front-/back-running or sandwich attacks.

Additionally, validators might collude and equivocate. They also represent valuable targets for attackers, since compromised validators can be manipulated to carry out equivocation, which might allow double-spending of the locked tokens on the Bitcoin blockchain.

Governance participants

Governance participants are involved in the decision-making process related to the blockchain's governance, including creating and executing proposals. Depending on the enabled governance proposals, a successful governance vote could result in the ability to:

- Call sudo entry points in smart contracts, granting privileged access to specific functions or features within the contract.
- Perform actions on behalf of any users of any smart contracts, which might include modifying the contract’s owner or transferring funds from the contract.
- Perform a contract migration to modify its source code, potentially including a backdoor to the contract.

Threat Model

STRIDE Classification

The following threat vectors have been identified using the STRIDE classification, grouped by components of the system.

	Spoofing	**Tampering**	**Repudiation**	**Information Disclosure**	**Denial of Service**	**Elevated Privileges**
Externally owned account	Lost account	Pharming/ phishing/ social engineering	Denial of actions	Private key leakage Doxxing/identity disclosure	DOS of infrastructure	Compromised private key
Cosmos infrastructure	-	Backdoors	-	Spyware	Message filtering	Governance actions
Execute messages handling	Dishonest merchants	Accidental request approvals	-	-	Enormous minimum burn amount	Incorrect address update
Bitcoin blockchain infrastructure	-	-	Bitcoin blockchain forks Orphan blocks	WBTC-BTC peg integrity	Network congestion Network halted	Compromised counterparty

Mitigation Matrix

The following mitigation matrix describes each of the threat vectors identified in the STRIDE classification above, assigning an impact and likelihood and suggesting countermeasures and mitigation strategies. Countermeasures can be taken to identify and react to a threat, while mitigation strategies prevent a threat or reduce its impact or likelihood.

Externally owned account

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Lost account: The attacker claims that they own an account and the access to the private key has been lost	Medium	Low	Have a clear policy not to refund lost assets or restore privileges	Enforce policy strictly
Pharming/phishing/social engineering: The attacker may manipulate users' wallets, lure them to malicious frontends, manipulate DNS records, or use social engineering to trick users/teams into signing manipulated transactions transferring funds/permissions	Medium	Medium	Educate users and team, protect DNS records, create awareness, offer blacklists with malicious sites, create activity on social channels to build reputation, deploy frontends on IPFS or other decentralized infrastructure	Monitor all systems, monitor communities and impersonations/malicious copies of official channels, communicate attempted pharming/phishing/social engineering, have processes in place to recover from DNS manipulation/frontends quickly
Denial of actions: An attacker may deny a performed action in their account, leading to disputes	Low	Low	Have a clear policy not to refund lost assets or restore privileges	Enforce policy and strictness
Private key leakage: Private keys are accidentally shared or	High	Medium	Educate users and team, ensure private keys are properly	-

logged			handled in wallet software, use hardware wallets/air-gapped devices, security keys, multi-signatures	
Doxxing/identity disclosure: Private data such as the off-chain identity of users disclosed	Low	Medium	Educate users and team, no storage of identity/sensible data in databases that link identity to account addresses, follow privacy regulations and guidelines	-
DOS of infrastructure: DOS attack on the end user's device/network or on the blockchain node they interact with	Medium	Low	Educate users and team, use firewalls, sentry architecture, load balancers, VPNs	Monitor infrastructure, and have processes in place to elastically provision and deploy additional resources
Compromised private key: Private keys may be compromised	High	Medium	Educate users and team	-

Cosmos infrastructure

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Backdoors: The builder team/deployer may implement back doors or malicious code that modify the WBTC contract's messages before sending them to the other chain	High	Low	Internal code reviews, unit testing, integration tests, automatic software engineering tools, audits	Monitor upgrade attempts, enforce processes on deployment, offer bug-bounties
Spyware: Builder team/deployer may manipulate code to share private keys or private information of users	High	Low	Internal code reviews, unit testing, integration tests, automatic software engineering tools audits	Monitor upgrade attempts, enforce processes on deployment, offer bug-bounties
Message filtering: Underlying blockchains could filter WBTC contract's messages	High	Low	Internal code reviews, unit testing, integration tests, automatic software engineering tools, audits	-
Governance actions: Underlying blockchains could implement governance modules that enable governance to execute privileged operations on the WBTC contract, such as migrations	High	Low	Educate users and the community about the project and its scope	Follow proposal discussions and be active in the community

Execute messages handling

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Dishonest merchants: Merchants might issue a mint amount that is higher than the actual BTC sent or issue another request to mint WBTC with the same transaction identifier	High	Medium	Remove merchant account privilege to prevent future mint requests and follow up to ensure the minted WBTC is burned to stabilize the peg	Enforce a strict policy for amount and transaction identifier verification by the custodian
Accidental request approvals: Custodian might approve invalid mint requests	Medium	Low	Remove merchant account privilege to prevent future mint requests and follow up to ensure the minted WBTC is burned to stabilize the peg	Enforce a policy for verifying requests by custodian
Enormous minimum burn amount: Merchants cannot burn WBTC due to the minimum burn amount requirement	Low	Low	Lower the minimum burn amount	Enforce a specific range for the minimum burn amount
Incorrect address update: Updates of existing accounts set to wrong addresses	High	Low	Internal reviews and perform relevant queries towards configuration to ensure the updated address is intended	Enforce processes on deployment and offer bug-bounties

Bitcoin blockchain infrastructure

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Bitcoin blockchain forks: If a hard fork happened on the Bitcoin network, there might be confusion on which chain will be backing WBTC	Medium	Low	Coordinate with custody wallet holder to determine which chain's BTC will be used to back WBTC	Establish a contingency plan that includes a decision-making process for selecting the chain that will back WBTC while maintaining communication with the community
Orphan blocks: A confirmed transaction may not end up in the main chain, although the custodian already used it to confirm a mint/burn request	Medium	Medium	Enforce a sufficiently high block depth before considering BTC transactions	Monitor the Bitcoin blockchain from different nodes with different connectivity to prevent the consideration of orphan blocks
WBTC-BTC peg integrity: The 1:1 peg stability might break due to market conditions and the custodian's delay in approving mint and burn requests	Medium	Medium	Continuously monitor the 1:1 ratio between WBTC and BTC to detect discrepancies and set clear expectations for response times	Establish a contingency plan for potential delays in the custodian's response and deviations from the 1:1 peg
Network congestion: The Bitcoin blockchain may experience congestion, delaying transaction processing and disrupting the WBTC minting and burning	Low	Low	Slow down the minting and burning services for WBTC. For severe cases, consider pausing the contract to prevent aggravating the	Monitor the Bitcoin blockchain to identify early signs of potential congestion

services			congestion and maintain the peg integrity	
Network halted: The Bitcoin blockchain halted, but the WBTC token can still be transacted	Low	Low	Pause the WBTC contract to ensure peg stability	Monitor the Bitcoin blockchain to identify early signs of halts
Compromised counterparty: The counterparty that controls the custody wallet might be compromised	High	Low	Establish strong security measures to prevent unauthorized access to the custody wallet	Establish a contingency plan if the counterparty is compromised or acts dishonestly. This includes transferring the custody wallet to a secondary counterparty