PUBLIC

Code Assessment

of the Solana WBTC Smart Contracts

April 22, 2025

Produced for

by

Contents

1 Executive Summary

Dear WBTC team,

Thank you for trusting us to help WBTC with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Solana WBTC according to Scope to support you in forming an opinion on their security risks.

WBTC implements a system that allows for minting and burning WBTC on Solana.

The most critical subjects covered in our audit are correctness of the minting and burning flow, the implementation of the access control, and the sanitization of the data accounts. Security regarding all the aforementioned subjects is high. Only minor issues have been uncovered which have been addressed.

The general subjects covered are compute unit efficiency of the implementation, the documentation and specification, and testing. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	5
• Code Corrected	4
• Acknowledged	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Solana WBTC repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	13 March 2025	8f3df0b18850d543605508471cf97801846475a4	Initial Version
2	7 April 2025	2fbe5ba70ef3f5e584a6bb5492975c0385c768be	Fixes
3	22 April 2025	6e434c06bc5f06c952ddb2600ad33eb557456895	Final version

For the Solana programs, the anchor version 0.31.0 was used.

The following files were considered in scope for the assessment:

```
programs
|-- controller
|   `-- src
|       |-- errors.rs
|       |-- events.rs
|       |-- instructions
|       |   |-- claim_mint_authority.rs
|       |   |-- claim_ownership.rs
|       |   |-- initialize.rs
|       |   |-- mint.rs
|       |   |-- mod.rs
|       |   |-- set_factory.rs
|       |   |-- set_members.rs
|       |   |-- transfer_mint_authority.rs
|       |   `-- transfer_ownership.rs
|       |-- lib.rs
|       `-- state
|           |-- controller.rs
|           `-- mod.rs
|-- factory
|   `-- src
|       |-- errors.rs
|       |-- events.rs
|       |-- instructions
|       |   |-- burn.rs
|       |   |-- initialize.rs
|       |   |-- mint_requests.rs
|       |   |-- mod.rs
|       |   |-- set_custodian_btc_deposit_address.rs
|       |   `-- set_merchant_btc_deposit_address.rs
|       |-- lib.rs
|       `-- state
```

```
|           |-- address.rs
|           |-- factory_state.rs
|           |-- mod.rs
|           `-- request.rs
`-- members
    `-- src
        |-- errors.rs
        |-- events.rs
        |-- instructions
        |   |-- add_merchant.rs
        |   |-- claim_ownership.rs
        |   |-- initialize.rs
        |   |-- mod.rs
        |   |-- remove_merchant.rs
        |   |-- set_custodian.rs
        |   `-- transfer_ownership.rs
        |-- lib.rs
        `-- state
            |-- members.rs
            |-- merchant_info.rs
            `-- mod.rs
```

`programs/controller/src/instructions/burn` was part of (Version 1) but was later removed.

2.1.1 Excluded from scope

Third party libraries and dependencies were excluded from the scope of this assessment. The Solana programs make use of the Anchor framework which is assumed to function properly.

The off-chain components and the deployment of the bridge were excluded from the scope of this assessment as well as the creation and configuration of WBTC token mint. We assume all these happen correctly.

All the handling of Bitcoin transactions, secure custody of BTC, non-Solana actions of the custodian, KYC processes, and similar organizational matters are out of scope of this review, but important to the security of the system.

2.2 System Overview

This system overview describes the initially received version ((Version 1)) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

WBTC offers an implementation of a system to bring WBTC on Solana. Merchants can create requests to mint and burn WBTC in the form of an SPL token. The process is facilitated by the custodian role which monitors the Bitcoin chain and mints WBTC on Solana or release BTC on Bitcoin should it be burned on Solana.

The system comprises three programs:

Factory:

This program is the merchant interface for bridging Bitcoin assets. Merchants use it to create (and cancel) requests to mint WBTC to their Solana address. The Custodian reviews these requests,

confirming them on-chain after verifying the corresponding deposit on Bitcoin, which triggers the actual minting by the Controller program; the Custodian can also reject requests. To burn WBTC, a merchant sends the tokens to a specific Controller-owned account (ATA), leading the Controller program to burn them. For minting to occur, the Custodian must first create a `CustodianBtcDepositAddress` data account for the merchant, specifying the Bitcoin address for deposits. For burning (receiving BTC back), the merchant must create a `MerchantBtcDepositAddress` data account with their own Bitcoin receiving address. Each mint and burn request is assigned a unique, sequentially incrementing nonce (starting from 0).

Controller:

The controller can mint or burn WBTC tokens. It only accepts calls from the factory program. It maintains the `ControllerStore` which holds the mint authority for WBTC in the SPL program. It can transfer the mint authority and the ownership of the `ControllerStore` to any other public key via a two-step transfer process. Finally, the controller allows modifying the address of the factory program it accepts calls from and the members program.

Members:

It maintains the `MembersStore` data account which tracks the custodian and the number of merchant data accounts (`MerchantInfo`). The owner of the `MemberStore` creates and deletes `MerchanInfo` data accounts. They can also set the custodian or initiate a two-step ownership transfer of the account.

Below we describe the minting and the burning process:

WBTC Minting:

1. The custodian registers a BTC custodian address for the merchant
2. The merchant sends BTC to the address of the custodian.
3. The merchant creates a request on Solana (`add_mint_request`).
4. The custodian confirms the transaction (`confirm_mint_request`) and mints WBTC for the merchant.

WBTC Burning:

1. The merchant registers a BTC withdrawal address.
2. The merchant creates a burn request and transfers the assets to controller's ATA , and then the controller's ATA burns the received tokens (`burn`).
3. The custodian sends BTC to the merchant's BTC withdrawal address.
4. The custodian confirms the transaction (`confirm_burn_request`).

Regarding Bitcoin we assume that no significant changes will take place. That means that we assume that the restrictions on Transaction Length and Address Length the system uses will not become problematic. Furthermore, we assume that secure deposits can be made.

2.3 Trust Model

We infer the following trust model:

Deployer: It deploys the programs and initializes the accounts. The trust to the deployer is limited as they should not be able to change the parameters later. It needs to transfer its initial privileges correctly. It is up to the deployment administrators of the system to check for a correct deployment with correct configurations.

Big DAO: The role is **fully trusted**. It is the `owner` of the `ControllerStore`. Therefore, they can set the factory program to accept calls from the members program which is used to check for whitelisted merchants. If wrong programs are set then the controller could be receiving unverified calls and mint/burn WBTC arbitrarily.

Small DAO: The role is **fully trusted**. It is the `owner` of the `MemberStore`. Therefore, they can set the custodian and whitelist merchants. Setting a malicious custodian can lead to arbitrary minting of WBTC.

Custodian: The role is **fully trusted**. They are responsible to confirming mint and burn requests that happened on Bitcoin. If they act maliciously the can wrongly confirm transactions that have not happened.

Merchant: The trust to the role is limited. They are allowed to create mint and burn requests and cancel mint requests. Hence, they could theoretically spam requests to annoy the custodian, but can be removed if that is the case.

As seen above the system defines different roles. If roles are being transferred, weird states can arise. For example, a transfer of mint authority could prevent mint requests from being confirmed. It is assumed that, for all authority transfers and system changes, WBTC will ensure that all ongoing processes are completed before implementation and any address changes will be communicated to users in advance to maintain transparency and operational continuity.

2.4 Changes in Version 2

The flow of burning has been changed. In Version 2, the Factory no longer transfers tokens to the Controller. Instead, the Factory directly issues a burn CPI which results in the tokens being burned from the merchant's account.

For the mint flow, the Factory now calls the Controller with a PDA-signed invocation using the `factory_store`. This provides better compatibility for users.

Otherwise, smaller efficiency improvements have been implemented.

2.5 Changes in Version 3

Minor improvements were implemented.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

5.1 Missing Length Checks

Design Low Version 1 Acknowledged

CS-SOLWTBC-001

The `factory` program implements `ConfirmBurnRequest()` that allows the custodian to confirm a burn request by updating the request status and the `btc_txid`. However, the `btc_txid` string length is only checked to be at most 64 characters. Similarly, the merchant and custodian deposit addresses are only checked to be at most 62 characters.

WBTC acknowledges the issue with the following statement:

"We did consider but to cover a few exceptions scenarios and follow the implementation on ETH the team has decided not to do so"

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	4

- Bitcoin Addresses Can Be Larger Than 62 Characters Code Corrected
- Squads Compatibility Code Corrected
- Unused Storage Variable Code Corrected
- transfer CPI Can Be Avoided Code Corrected

Informational Findings	4

- Initially Created ATA Is Unused Code Corrected
- Inaccuracies in the Specification Specification Changed
- Redundant Constraint Code Corrected
- Unchecked mint_to and burn Code Corrected

6.1 Bitcoin Addresses Can Be Larger Than 62 Characters

Correctness Low Version 2 Code Corrected

CS-SOLWTBC-003

In the `factory` program, `RequestAccount` constraints the length of `btc_deposit_address` to be at most 62 characters. However, Bech32 addresses can have a length up to 90 characters. The factory program does not support such addresses.

Note that such addresses are supported in the WBTC bridge deployed on Ethereum mainnet as no length constraint is enforced there. Therefore, there is a discrepancy between the two implementations.

Code corrected:

The length constraint has been modified to ensure that the address length is at most 100 characters.

6.2 Squads Compatibility

Design Low Version 1 Code Corrected

CS-SOLWTBC-002

The different roles in the system might use Squads instead of a simple wallet account. In Squads, one can `batch_create()`, followed by `batch_add_transaction()`. Each added transaction to the batch will be executed in order as a CPI in `batch_execute_transaction()`. Therefore, the source of the CPI call will always be the Squads program. This leads to the following restrictions:

- Merchants using Squads cannot `burn` WBTC as the controllers verifies with `instruction_sysvar` that the instruction called is from the `factory`. However, the currently executing instruction will be `batch_execute_transaction()` from Squads.
- For a similar reason, a Custodian using Squads cannot `mint` WBTC as the controllers also verify with `instruction_sysvar` that the instruction called is from the `factory`.

Code corrected

WBTC removed the check using `instruction_sysvar` in the `mint` and `burn` instructions. Instead, the `factory_store` now signs the CPI and the controller expects in respectively `mint` and `burn` the `factory_store` to be the signer of the transaction. This way, the controller ensures that only the factory can call `mint` and `burn` without preventing the use of Squads by merchants or the custodian.

6.3 Unused Storage Variable

Design | Low | Version 1 | Code Corrected

CS-SOLWTBC-007

Some storage variables are never used:

1. The `ControllerStore` holds `pending_owner_program` and `pending_mint_authority_program` but they are never used a part from being set back to the default public key.
2. The same holds for the `pending_owner_program` of the `MembersStore` in `members`.
3. The `merchant` public key is stored in the `MerchantInfo` but it's not required as the `merchant` is also used in the seed.

Code corrected:

The `pending_owner_program` and `pending_mint_authority_program` were removed from the `ControllerStore` and the `pending_owner_program` was removed from the `MembersStore`.

WBTC decided to keep the `merchant` public key in `MerchantInfo` to retrieve data more conveniently.

6.4 `transfer` CPI Can Be Avoided

Design | Low | Version 1 | Code Corrected

CS-SOLWTBC-009

In the `factory` program, the `burn` instruction first transfers the WBTC to the `controller_token_account` and then invokes `burn` on the `controller` program to burn the token through a CPI call to the SPL token program.

However, the transfer call from the merchant to the `controller_token_account` can be avoided by burning the required amount from the merchant's token account directly.

Code corrected:

The burn process has been modified to directly burn the tokens from the merchant's token account, eliminating the transfer step.

6.5 Initially Created ATA Is Unused

Informational | Version 2 | Code Corrected

CS-SOLWTBC-004

In the `initialize` instruction the `Controller` will create an Associated Token Account (ATA) for the `controller_store`, called `controller_token_account`. Due to other design changes in the code in Version 2, this ATA is no longer being used. Hence, there is no reason to create it.

Code corrected:

The Associated Token Account `controller_store` has been removed from the `Controller`.

6.6 Inaccuracies in the Specification

Informational | Version 1 | Specification Changed

CS-SOLWTBC-005

The specification describes the minting and burning flows. However, the following inaccuracies were found:

- The burning flow indicates that the WBTC tokens are burned in the last step, but they are actually burned when the merchant initiates the burn request.
- The minting flow indicates that the merchant first initiates a mint request to the `factory` and then the custodian sets the BTC deposit address for that merchant. However, `add_mint_request_handler()` in the factory requires the BTC deposit address to be set.

Code corrected

WBTC updated the specification.

6.7 Redundant Constraint

Informational | Version 1 | Code Corrected

CS-SOLWTBC-006

In `confirm_mint_request` you specify the following constraints:

```
#[account(
    mut,
    address = controller_store.token_mint, // <-- C1
    mint::token_program = token_program,
    constraint = token_mint.key() == controller_store.token_mint @ FactoryError::InvalidTokenMint // <-- C2
)]
pub token_mint: InterfaceAccount<'info, Mint>,
```

The constraints C1 and C2 are equivalent, therefore one of them can be removed.

Another example of redundant constraints is

```
#[account(
    seeds = [
        MERCHANT_BTC_ADDRESS_SEED,
        payer.key().as_ref() // <-- C1
    ],
    bump = merchant_btc_address.bump,
    constraint = merchant_btc_address.merchant == payer.key() @ FactoryError::InvalidBtcAddress, // <-- C2
    constraint = !merchant_btc_address.btc_address.is_empty() @ FactoryError::InvalidBtcAddress
)]
pub merchant_btc_address: Account<'info, MerchantBtcDepositAddress>,
```

In this case, the derivation of the `merchant_btc_address` PDA address is done using the `payer` key, therefore the constraint C2 is redundant as the PDA is implicitly associated to the `payer` through the seed.

Code corrected:

- The first redundant constraint was removed.
- The second redundant constraint was kept to ensure that the content of `merchant_btc_address.merchant` is correct.

6.8 Unchecked `mint_to` and `burn`

Informational | Version 1 | Code Corrected

CS-SOLWTBC-008

The `controller` program uses `mint_to()` and `burn()` in respectively `mint` and `burn`. However, neither of these functions performs any checks on the token mint and decimals.

Code corrected

`mint_to()` and `burn()` were respectively replaced by `mint_to_checked()` and `burn_checked()`.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Burning Is Restricted to ATAs

Note Version 1

When a merchant wants to bridge back BTC from Solana to the Bitcoin network, it will create a burn request through the `factory` program with `burn()`. This function will burn the amount of WBTC to bridge from the merchant `token_account`. However, `token_account` is enforced to be an ATA of the merchant account. Thus, a merchant can't burn WBTC from a token account. To burn WBTC, the merchant must first transfer the WBTC from his token account to his ATA and then call `burn()`.

7.2 WBTC Is Not Pausable

Note Version 1

While the WBTC from Bitcoin network to Ethereum mainnet is pausable, it is not on Solana. This difference is notable as the design of the system on Solana tries to not deviate from the initial design on Ethereum.

7.3 `merchant_count` Size

Note Version 1

The current `merchant_count` is limited to 65535 merchants due to it being typed as a `u16`. This is sufficient with the current specification. However, it might not be in the future the number of merchants increases significantly.