Solana WBTC

Audit

Presented by:

OtterSec contact@osec.io

Nicola Vella nick0ve@osec.io

Robert Chen r@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Solana Labs engaged OtterSec to perform an assessment of the `wbtc` program. This assessment was conducted between May 11th and May 26th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 6 findings total.

In particular, we identified an issue regarding an erroneous setting for the rent payment (OS-SOL-ADV-00).

We also made recommendations around the presence of anti-patterns for the setting of new authority (OS-SOL-SUG-01), lack of checks for the BTC addresses and transaction (OS-SOL-SUG-02), and erroneous control during the setting of the new custodian (OS-SOL-SUG-03).

Scope

The source code was delivered to us in a git repository at github.com/solana-labs/wbtc. This audit was performed against commit 43e1042.

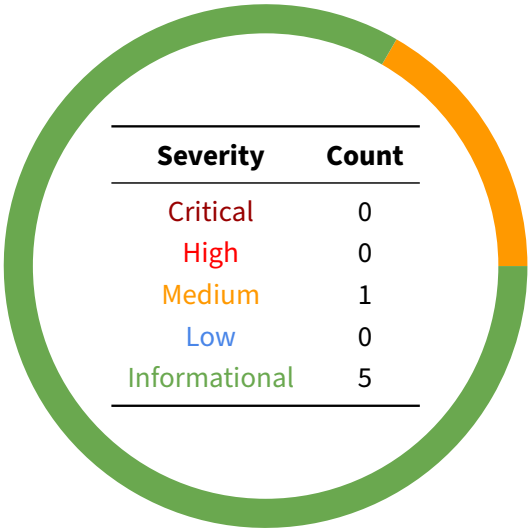
A brief description of the programs is as follows.

Name	Description
wbtc	A contract that enables the wrapping of tokens through collaboration between merchants and a third-party custodian.

02 | Findings

Overall, we reported 6 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	1
Low	0
Informational	5

03 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-SOL-ADV-00	Medium	Resolved	An incorrect account assignment leads to a rent misallocation in `approve_redeem_request`.

OS-SOL-ADV-00 [med]| Rent Misallocation

Description

In `approve_redeem_request.rs`, the rent for closing the `RedeemRequest` is directed to the `merchant`. However, it should be assigned to `merchant_authority`, which represents the account that executed the initial payment.

src/instruction/approve_redeem_request.rs RUST

```rust
#[derive(Accounts)]
pub struct ApproveRedeemRequestAccounts<'info> {
    #[account(mut)]
    pub custodian: Signer<'info>,

    #[account(has_one = custodian @ ErrorCode::InvalidCustodian)]
    pub config: Account<'info, Config>,

    /// CHECK: checked with constraint
    #[account(mut,
        constraint = merchant.authority == merchant_authority.key() @
    ↪  ErrorCode::InvalidMerchantAuthority
    )]
    pub merchant_authority: AccountInfo<'info>,

    #[account(mut)]
    pub merchant: Account<'info, Merchant>,

    #[account(mut,
        close = merchant,
        has_one = merchant,
    )]
    pub redeem_request: Account<'info, RedeemRequest>,

    pub token_program: Program<'info, Token>,
}
```

Remediation

Change the value of `close` in the `ApproveRedeemRequestAccounts` struct's `redeem_request` from `merchant` to `merchant_authority`.

Patch

Fixed in b559f5b.

04 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-SOL-SUG-00	The `timestamp` field of `MintRequest` is never used.
OS-SOL-SUG-01	The pattern for the assignment of the new authority may lead to a potential loss of control over the protocol.
OS-SOL-SUG-02	Unicode characters can be accepted as input and serialized into multiple bytes, which may exceed the allocated size.
OS-SOL-SUG-03	An erroneous check in `set_custodian` leads the program to abort.
OS-SOL-SUG-04	Presence of misspelled word in `set_merchant_btc_address`.

OS-SOL-SUG-00 | Presence Of Unused Field

Description

`state.rs` contains a declaration of the `MintRequest` object. However, despite including the `timestamp` field in its definition, this field is not utilized for either writing or reading data throughout the code.

src/state.rs RUST

```rust
#[account]
#[derive(Debug)]
pub struct MintRequest {
    pub req_id: u64,
    pub merchant: Pubkey,
    pub client_token_account: Pubkey,
    pub timestamp: u64,
    pub amount: u64,
    pub transaction_id: String,
}
```

Remediation

Fill the `timestamp` field when creating a `MintRequest`.

Patch

Fixed in fe6f752.

DIFF

```
pub fn handler(ctx: Context<CreateMintRequestAccounts>, args:
    ↪  CreateMintRequestArgs) -> Result<()> {
    ...
+   mint_request.timestamp = Clock::get()?.unix_timestamp as u64;
```

OS-SOL-SUG-01 | Unsafe New Authority Assignment

Description

In `set_authority.rs`, `handler` is responsible for setting the new authority. However, the program does not currently account for the scenario where the public key of the new authority is mistyped. This potentially leads to a loss of control over the protocol. Given the significance of `Config.authority` in the protocol, splitting the `Authority` change into two distinct phases is advised.

src/instructions/set_authority.rs RUST

```rust
#[derive(Accounts)]
pub struct SetAuthorityAccounts<'info> {
    #[account(mut)]
    pub authority: Signer<'info>,

    #[account(mut, has_one = authority @ ErrorCode::InvalidAuthority)]
    pub config: Account<'info, Config>,

    /// CHECK: nothing to check here, it's just the new authority address
    pub new_authority: AccountInfo<'info>,
}

pub fn handler(ctx: Context<SetAuthorityAccounts>) -> Result<()> {
    ctx.accounts.config.authority = ctx.accounts.new_authority.key();

    Ok(())
}
```

Remediation

Divide the `Authority` change into two separate phase:

1. `SetNewAuthority`, signed by the current authority and used for setting the new authority.
2. `ClaimNewAuthority`, signed by the new authority and used for setting the current authority.

Patch

Fixed in fe6f752.

OS-SOL-SUG-02 | Lack Of BTC Validation

Description

`utils.rs` contains checks to ensure the correctness of BTC addresses and transactions. However, these checks rely solely on the length of the input string without considering whether the characters are serialized using exactly one byte each.

It is possible to supply Unicode characters that may result in a serialized length of up to six bytes each, exceeding the allocated size for the account holding them.

src/utils.rs RUST

```rust
pub fn validate_btc_address(address: &String) -> Result<()> {
    msg!("len {}", address.len());
    require!(
        address.len() <= BTC_ADDRESS_MAX_LEN,
        ErrorCode::AddressTooLong
    );
    require!(
        address.len() >= BTC_ADDRESS_MIN_LEN,
        ErrorCode::AddressTooShort
    );
    Ok(())
}

pub fn validate_btc_transaction(transaction: &String) -> Result<()> {
    msg!("len {}", transaction.len());
    require!(
        transaction.len() == BTC_TRANSACTION_LEN,
        ErrorCode::InvalidTransactionLength
    );
    Ok(())
}
```

Remediation

Ensure that both `address` and `transaction` strings only contain ASCII alphanumeric characters.

Patch

Fixed in fe6f752.

OS-SOL-SUG-03 | Erroneous Check In Set Custodian

Description

In `set_custodian`, there is a condition that verifies if `custodians` are enabled and if the `key` of the `authority` differs from the `Key` of the `Custodian`. However, the program does not account for the situation where the `authority` itself is the `custodian` and the `custodian` is not enabled. In this scenario, the instruction would produce an error, even if the signing authority is the protocol authority.

src/instructions/set_custodian.rs RUST

```rust
pub fn handler(ctx: Context<SetCustodianAccounts>) -> Result<()> {
    // rationale: custodian should only be able to change its address if its flag
    ↪  is enabled
    require!(
        ctx.accounts.config.custodian_enabled
            || ctx.accounts.authority.key() != ctx.accounts.config.custodian,
        ErrorCode::CustodianDisabled
    );

    ctx.accounts.config.custodian = ctx.accounts.new_custodian.key();

    Ok(())
}
```

Remediation

Add the correct checks to ensure that `new_custodian` cannot be assigned as the `authority` and `new_authority`.

Patch

Fixed in fe6f752.

OS-SOL-SUG-04 | Misspelled Word

Description

`set_merchant_btc_address` contains a minor spelling error. `new_merchant_btc_addresss` contains an extra `s` at the end, which is likely unintentional and should be corrected.

src/instructions/set_merchant_btc_address.rs RUST

```rust
pub fn handler(
    ctx: Context<SetMerchantBtcAddressAccounts>,
    new_merchant_btc_addresss: String,
) -> Result<()> {
    validate_btc_address(&new_merchant_btc_addresss)?;

    ctx.accounts.merchant.btc_address = new_merchant_btc_addresss;

    Ok(())
}
```

Remediation

Remove the last `s` in `new_merchant_btc_addresss`.

Patch

Fixed in b559f5b.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority or access control validation • Improperly designed economic incentives leading to loss of funds
High	Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities that could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of sum, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.