
# Code Review and Security Assessment
# For
# Symbiosis Finance

April 14, 2022

Updated: June 8, 2022

Prepared For
Symbiosis.finance

Prepared by
Er-Cheng Tang | HashCloak Inc
Peter Lai | HashCloak Inc

# Table Of Contents

# Executive Summary

Symbiosis engaged HashCloak Inc for an audit of the Symbiosis Protocol, which is a smart contract written in CosmWasm. The audit was done with 2 auditors over 2 weeks, from March 28, 2022 to April 11, 2022. The relevant code base was the terra-audit repository, assessed at commit b224718ae0e1b9a26f1e430c75d7a33705db74d1. The scope of the audit was terra-audit/packages/symbiosis/src, terra-audit/contracts/portal/src and terra-audit/contracts/bridge/src. During the first week we familiarized ourselves with the documents and the codebase. In the second week we investigated the security of the codebase through various efforts. During the audit, as we gave them feedback, they provided an updated commit: a3804bc44334adc85417ce84cdf64e2803b8091d. On April 19 2022, we received an updated commit 154852354f49f893e7dd69e9134d3ab76edc14f1 for changes regarding our informational comments. The relevant findings have been updated to reflect this. On May 10 2022, we received an updated commit bd1d83531aaceeec60392aba93ee9c59d59ad753 for their metarouter changes. The relevant findings have been updated.

We found a variety of issues ranging from Critical to Informational, and we provide some general guidance to improve the code quality.

| Severity | Number of Findings |
|---|---|
| Critical | 1 |
| High | 0 |
| Medium | 1 |
| Low | 1 |
| Informational | 5 |

# Overview

Symbiosis Finance is a decentralized multi-chain liquidity protocol. It provides a synthesizing / unsynthesizing mechanism that transforms tokens between different chains. To achieve this, Symbiosis Finance creates smart contracts on the supported chains and maintains a group of relayers that passes transactions across chains.

When a user wants to synthesize tokens on chain B, it first locks its tokens in a smart contract on chain A. The smart contract will emit events to inform the relayers to call a smart contract on chain B, which mints tokens on chain B. When a user wants to unsynthesize tokens on chain B, it first burns its tokens by calling a smart contract on chain B. The smart contract will emit events to inform the relayers to call a smart contract on chain A, which unlocks the user's tokens on chain A. There are also revert-type functions that restore the state of one chain if the smart contracts on the other chain fail to process the task requested.

The codebase can be divided into several logical units.

- packages/symbiosis/src: The data structures for all kinds of messages
- contracts/portal/src: (Un)locking tokens and producing synthesis messages
- contracts/bridge/src: Passing transactions between the portal and the relayers
- contracts/metarouter/src: Swap A tokens to synthesizing tokens on DEX
- contracts/metarouter_gateway/src: Gateway of metarouter

# Methodology

We inspected the protocol design, the business logic and the consistency between the implementation and the documentation. We manually checked through common vulnerabilities regarding smart contract implementations, the Rust programming language, and CosmWasm.

# Findings

## Missing access control

**Type**: Critical
**Files affected**: bridge/src/execute.rs, portal/src/execute.rs

**Description**: Strict access control should be placed in every contract method that has intended callers. Otherwise, a malicious party can interfere with the business logic and violate the design goals of the contract. In symbiosis finance, the methods `change_mpc()`, `change_portal()`, `change_synthesis()`, `receive_request()` should only be called by trusted entities. However, they have no access control in the current codebase. Thus, the entire (un)synthesizing mechanism is put at risk since transactions submitted by malicious parties will also be treated as trusted.

**Impact**: An attacker can arbitrarily synthesize coins for itself at no cost, or make other users pay for multiple fees by unsynthesizing their coins repeatedly.

**Suggestion**: In the beginning of the methods `change_mpc()`, `change_portal()`, `change_synthesis()`, `receive_request()`, check that `info.sender` has the intended address. The method `revert_burn_request()` also lacks access control, but we do not know at this point whether it is intended to be so.

**Status**: The issue has been resolved as of [59086cd7053d81d8208ca1deccca0e9bdfdc3ffb](59086cd7053d81d8208ca1deccca0e9bdfdc3ffb).

## Unvalidated input addresses

**Type**: Medium
**Files affected**: bridge/src/contract.rs, portal/src/contract.rs, packages/symbiosis/src/bridge.rs, packages/symbiosis/src/portal.rs

**Description**: The input message types are defined using the `Addr` type, but these fields are never validated. It does not comply with the documentation of CosmWasm. ([https://github.com/CosmWasm/cosmwasm/blob/main/packages/std/src/addresses.rs#L19](https://github.com/CosmWasm/cosmwasm/blob/main/packages/std/src/addresses.rs#L19))

**Suggestion**: We recommend using strings instead of addresses within the definition of input messages. Afterwards, `deps.api.addr_validate()` can be used to convert input strings into the `Addr` type.

**Status**: The issue has been resolved as of 9b05e1122b684234d38428ded69939df83713f48.

### Improper transfer logic for native tokens

**Type**: Low
**Files affected**: metarouter/src/execute.rs, metarouter_gateway/src/execute.rs

**Description**: According to the documentation, the user can interact with metarouter alone if the origin token is a native token. However, the current implementation of the metarouter always calls the gateway for the funds, so the user is required to deposit funds into the gateway in advance. Moreover, the gateway does not keep track of the owner of each native token deposit. Instead, the gateway simply transfers its native tokens. As a result, a user who wishes to swap using a native token would face the risk of losing the token when the token is held by the gateway.

**Suggestion**: When the metarouter is executed with a native token, the metarouter shall receive the funds using `info.funds` instead of sending a `Claim` message to the gateway. Also, remove the native token transfer logic from `metarouter_gateway`.

### Incorrect business logic

**Type**: Informational
**Files affected**: portal/src/execute.rs

**Description**: In line 244, the recipient of the token transfer is intended to be `state.bridge.to_string()`, but placed incorrectly as `target.to_string()`.

**Suggestion**: We recommend making the corrections described above.
**Status**: This has been fixed as of commit 154852354f49f893e7dd69e9134d3ab76edc14f1.

### Inconsistent use of contract error

**Type**: Informational
**Files affected**: portal/src/execute.rs

**Description**: Some contract errors are given names in error.rs. It would be better to return these errors whenever it is the case. The code in line 117 uses `assert_eq!` instead of returning the more explanatory `ContractError::Unauthorized{}`. The latter usage appears in lines 199-201.

**Suggestion**: We recommend adopting the codes in lines 199-201 to line 117.
**Status**: This has been fixed as of commit [154852354f49f893e7dd69e9134d3ab76edc14f1](154852354f49f893e7dd69e9134d3ab76edc14f1).

### Possible price slippage

**Type**: Informational
**Files affected**: metarouter/src/execute.rs

**Description**: The AMM slippage might affect the token amount when the user swaps through metarouter. The user could get an amount smaller than expected.

**Suggestion**: Add `minimum_amount_out` as a parameter and compare this value.

### Permanent allowance

**Type**: Informational
**Files affected**: metarouter/src/execute.rs

**Description**: It is not a good idea in general to have a permanent allowance. In fact, it is sufficient to have an allowance for the current block height when someone wants to use the metarouter. Therefore we would recommend using a stricter policy.

**Suggestion**: Set the expiration time in line 49 as
`Expiration::AtHeight(env.block.height + 1)`

### Missing assertions

**Type**: Informational
**Files affected**: bridge/src/execute.rs

**Description**: The function `receive_request` has an argument `receive_side`. It would be better to assert that `receive_side` is equal to the address of the current contract. Otherwise the argument is useless throughout the function.

**Suggestion**: Assert that `receive_side` and `env.contract.address.to_string()` are equal.