SMART CONTRACTS REVIEW

zokyo

May 29th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING SYMBIOSIS

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 0 Medium issues: 0 points deducted
- 4 Low issues: 3 resolved and 1 acknowledged= - 3 points deducted
- 1 Informational issue: 1 acknowledged = 0 points deducted

Thus, 100 - 3 = 97

TECHNICAL SUMMARY

This document outlines the overall security of the Symbiosis smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Symbiosis smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Symbiosis team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

Auditing Strategy and Techniques Applied 5

Executive Summary 7

Structure and Organization of the Document 8

Complete Analysis 9

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Symbiosis repository:
Repo: https://github.com/symbiosis-finance/core-contracts

Last commit - d97a8a1b4e115ba996811995438e063d843eb625

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- TonBridge.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Symbiosis smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Foundry testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team has performed a security review of the provided codebase. The contract submitted for auditing is well-crafted and well organized.

The TonBridge.sol smart contract is a contract that is part of the Symbiosis protocol. Its main function is `callBridgeRequest`. This function allows users to input an amount of tokens, a Ton address, and a chainId to execute the request. This function passes calldata, a burn function, to the receive side, which executes it.

The contract contains several admin functions that allow the owner to change important parameters for the execution of the call, such as the chainId of the bridge, the tonBridge address, the address of the broadcaster, etc.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Symbiosis team and the Symbiosis team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	ATTACKER CAN GET OWNERSHIP OF THE CONTRACT	Low	Resolved
2	OWNERSHIP CAN BE TRANSFERRED TO ADDRESS(0)	Low	Resolved
3	MISSING SANITY CHECKS FOR IMPORTANT VARIABLES	Low	Resolved
4	USE A MULTISIG ACCOUNT FOR OWNER	Low	Acknowledged
5	PUBLIC FUNCTIONS CONSUME MORE GAS THAN EXTERNAL ONES	Informational	Acknowledged

ATTACKER CAN GET OWNERSHIP OF THE CONTRACT

The ownership of the `TonBridge.sol` contract is set in the `initialize()` function by the caller. This function, `initialize()` has no access control so anybody can call the function and become the owner. The `initialize()` function can only be called once due to the `initializer` modifier so when the contract is deployed and attacker can front-run the deployer transaction calling `initialize()` and get the ownership of the contract and it has been mentioned then the deployer would not be able to call `initialize()` again.

```
function initialize(address _tonBridge, address _symbiosisBridge, uint256 _bridgeChainId,
address _broadcaster) public virtual initializer {
        __Ownable_init();

        tonBridge = _tonBridge;
        symbiosisBridge = _symbiosisBridge;
        bridgeChainId = _bridgeChainId;
        broadcaster = _broadcaster;
    }
```

Recommendation:

So there are several mitigation actions that can be taken:
Verify after deployment, allow the call from one a certain address, include initialization call in your proxy setup, use a multicall contract to make the initialization call in the same transaction as the proxy setup.

OWNERSHIP CAN BE TRANSFERRED TO ADDRESS(0)

The `TonBridge.sol` contract is `OwnableUpgradeable` which contains a `renounceOwnership()` function that transfers ownership to address(0), which cause the same effect as having the contract without owner:

```
/**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
```

```
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(address(0));
    }
```

If ownership is renounced then some crucial functions that implements `onlyOwner` modifier will not be callable.

Recommendation:
Override `renounceOwnership()` function and revert the execution to do not allow ownership getting revoked:

function renounceOwnership() public override onlyOwner {
revert ("NotAllowed");
}

MISSING SANITY CHECKS FOR IMPORTANT VARIABLES

There are some important variables in the `TonBridge.sol` contract that are not checked before being set in the `initialize()` function and also in the functions used for changing their values.

- Initialize():

```
function initialize(address _tonBridge, address _symbiosisBridge, uint256 _bridgeChainId,
address _broadcaster) public virtual initializer {

        __Ownable_init();
        tonBridge = _tonBridge;
        symbiosisBridge = _symbiosisBridge;
        bridgeChainId = _bridgeChainId;
        broadcaster = _broadcaster;
}
```

- Functions used for changing values:

```
function changeBridgeChainId(uint256 _newBridgeChainId) external onlyOwner {
        bridgeChainId = _newBridgeChainId;
        emit ChangeBridgeChainId(_newBridgeChainId);
    }
```

```
/**
 * @notice Changes TON Bridge by owner
 */
function changeTonBridge(address _newTonBridge) external onlyOwner {
    tonBridge = _newTonBridge;
    emit ChangeTonBridge(_newTonBridge);
}

/**
 * @notice Changes Symbiosis Bridge by owner
 */
function changeSymbiosisBridge(address _newSymbiosisBridge) external onlyOwner {
    symbiosisBridge = _newSymbiosisBridge;
    emit ChangeSymbiosisBridge(_newSymbiosisBridge);
}

/**
 * @notice Changes Broadcaster by owner
 */
function changeBroadcaster(address _newBroadcaster) external onlyOwner {
    broadcaster = _newBroadcaster;
    emit ChangeBroadcaster(_newBroadcaster);
}
```

Recommendation:

Add a check to ensure that the set values in the initialize() function and in the functions used for changing the variable values are not equal to address(0).

USE A MULTISIG ACCOUNT FOR OWNER

The functions `changeBridgeChainId()`, `changeTonBridge()`, `changeSymbiosisBridge()` and `changeBroadcaster` are onlyOwner callable.This means the owner has the authority to change key variable values that affect core functionality. Therefore, it's advisable to use a multisig account to prevent accidental updates or to protect against the owner account being compromised.

```
function changeBridgeChainId(uint256 _newBridgeChainId) external onlyOwner {
        bridgeChainId = _newBridgeChainId;
        emit ChangeBridgeChainId(_newBridgeChainId);
    }

    /**
     * @notice Changes TON Bridge by owner
     */
    function changeTonBridge(address _newTonBridge) external onlyOwner {
        tonBridge = _newTonBridge;
        emit ChangeTonBridge(_newTonBridge);
    }

    /**
     * @notice Changes Symbiosis Bridge by owner
     */
    function changeSymbiosisBridge(address _newSymbiosisBridge) external onlyOwner {
        symbiosisBridge = _newSymbiosisBridge;
        emit ChangeSymbiosisBridge(_newSymbiosisBridge);
    }

    /**
     * @notice Changes Broadcaster by owner
     */
    function changeBroadcaster(address _newBroadcaster) external onlyOwner {
        broadcaster = _newBroadcaster;
        emit ChangeBroadcaster(_newBroadcaster);
    }
```

Recommendation:

Use a multisig account for owner's account.

PUBLIC FUNCTIONS CONSUME MORE GAS THAN EXTERNAL ONES

Calling an external function is usually more gas-efficient than a public function when called from outside the contract because external functions can bypass the step of copying argument data from calldata to memory.

In `TonBridge.sol` contract the `callBridgeRequest()` function has been marked as public but can be switched to external as it is not called internally within the contract.

```
Recommendation:

Switch visibility from `public` to `external` in the case of the `callBridgeRequest(
```

Recommendation:
Switch visibility from `public` to `external` in the case of the `callBridgeRequest()` function to improve gas saving.

	TonBridge.sol
Reentrance	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Symbiosis team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Symbiosis team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo