Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.08.30, the SlowMist security team received the StakeStone team's security audit application for STONE BTC, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

The code under audit in this section allows users to mint STONE BTC by depositing supported assets. The assets that can be deposited are BTC ERC20 tokens pegged to BTC at a 1:1 ratio. The protocol can accept assets from different EVM-compatible chains, minting STONE BTC on those chains. Additionally, STONE BTC can be transferred across supported chains. On the target chain, STONE BTC can be redeemed for the underlying asset at a 1:1 ratio. During the redemption process, the protocol deducts fees based on the specified fee rate.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Potential Token Compatibility Issues	Design Logic Audit	Suggestion	Acknowledged
N2	Potential risk of not being able to collect fees	Design Logic Audit	Low	Acknowledged
N3	Unnecessary unchecked	Gas Optimization Audit	Suggestion	Acknowledged
N4	Risk of DoS when removing supported tokens	Denial of Service Vulnerability	High	Acknowledged
N5	The actual deposit amount may differ from the contract balance	Design Logic Audit	Low	Acknowledged
N6	Not checking if withAmount is greater than 0 when retrieving all tokens	Design Logic Audit	Suggestion	Acknowledged
N7	Risks of excessive privilege	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/stakestone/stakestone-btc-vault

commit: 5430f96facfa32e0d80699f1f5b62e6d59621974

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

StoneBTC			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
mint	External	Can Modify State	onlyRole
burn	External	Can Modify State	onlyRole

StoneBTCLayerZeroAdapter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OFTAdapter Ownable
setCap	External	Can Modify State	onlyRole
getQuota	External	-	-
approvalRequired	External	-	-
_debit	Internal	Can Modify State	-
_credit	Internal	Can Modify State	-

StoneBTCVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deposit	External	Can Modify State	-
depositMultiple	External	Can Modify State	-
withdraw	External	Can Modify State	-
withdrawMultiple	External	Can Modify State	-
_checkDepositAllowed	Internal	-	-
getDepositAmounts	External	-	-

StoneBTCVault			
addSupportedTokens	External	Can Modify State	onlyRole
removeSupportedTokens	External	Can Modify State	onlyRole
setWithdrawFeeRate	External	Can Modify State	onlyRole
setDepositFeeRate	External	Can Modify State	onlyRole
setFeeRecipient	External	Can Modify State	onlyRole
setDepositPause	External	Can Modify State	onlyRole
setWithdrawPause	External	Can Modify State	onlyRole
setDepositCapacity	External	Can Modify State	onlyRole
setWhitelistMode	External	Can Modify State	onlyRole
setWhitelistAddress	External	Can Modify State	onlyRole

Proposal			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
propose	External	Can Modify State	onlyRole
revokeProposal	External	Can Modify State	onlyRole
voteFor	External	Can Modify State	-
retrieveTokenFor	External	Can Modify State	-
retrieveAllToken	External	Can Modify State	-
execProposal	External	Can Modify State	onlyRole
setVotePeriod	External	Can Modify State	onlyRole
getProposal	Public	-	-
getProposals	Public	-	-

Proposal			
canVote	Public	-	-
canExec	Public	-	-
invoke	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Suggestion] Potential Token Compatibility Issues

Category: Design Logic Audit

Content

In the StoneBTCVault contract, users can deposit funds through the deposit/depositMultiple functions. The contract directly transfers the user-specified amount of wrapped BTC tokens using the safeTransferFrom function. It is important to note that the contract is not compatible with fee-on-transfer wrapped BTC tokens.

Similarly, when users make deposits or withdrawals, the contract performs decimal conversion using `18-tokenDecimals[_token]`. This renders the contract incompatible with any wrapped BTC tokens that have a decimal greater than 18.

Code location: src/StoneBTCVault.sol#L66,L69,L98,L99,L138,L165

```
function deposit(
    address _token,
    uint256 _amount
) external {
    _checkDepositAllowed(_token, _amount, msg.sender);

    TransferHelper.safeTransferFrom(_token, msg.sender, address(this), _amount);

    IStoneBTC stone = IStoneBTC(stoneBTC);
    uint256 mintAmount = _amount * 10**(18-tokenDecimals[_token]);
    ...
}
```

Solution

To accommodate the aforementioned scenarios, the contract should calculate the difference in the contract's

balance before and after the token transfer to determine the actual deposit amount made by the user. Additionally, the contract should handle wrapped BTC tokens with decimals greater than 18.

Status

Acknowledged; After communicating with the project team, the project team stated that they would not support the above types of tokens.

[N2] [Low] Potential risk of not being able to collect fees

Category: Design Logic Audit

Content

In the StoneBTCVault contract, users are charged a certain fee when making deposits or withdrawals. The fee amount is determined by `amount * feeRate / FEE_BASE`. Due to Solidity's division operation truncating the decimal part, if the user's deposit or withdrawal amount is relatively small, the calculated fee will be 0. This prevents the contract from collecting deposit/withdrawal fees.

Code location: src/StoneBTCVault.sol#L73,L103,L133,L161

```
function deposit(
    address _token,
    uint256 _amount
) external {
    ...
    if (feeRate > 0 && recipient != address(0)) {
        uint256 feeAmount = mintAmount * feeRate / FEE_BASE;
        ...
    }
    ...
}
```

Solution

If this is not the intended design, it is recommended to set a minimum deposit/withdrawal amount to mitigate this risk.

Status

Acknowledged; After communicating with the project team, the project team stated that this was the expected design.

[N3] [Suggestion] Unnecessary unchecked

Category: Gas Optimization Audit

Content

In the StoneBTCVault contract, all the for loop functionalities use `unchecked` for incrementing i to reduce gas consumption. However, the contract's Solidity compilation uses `^0.8.26`, and Solidity introduced the unchecked loop increments feature in version 0.8.22, making the use of `unchecked` unnecessary.

Code location:src/StoneBTCVault.sol#L111,L169,L201,L219,L238,

```
function depositMultiple(
    address[] memory _tokens,
    uint256[] memory _amounts
) external {
    ...
    for (uint256 i; i < length;) {
        ...
        unchecked { ++i; }
    }
    ...
}
```

Solution

It is recommended to remove the usage of unchecked in the contract's for loop functionalities to improve readability.

Status

Acknowledged

[N4] [High] Risk of DoS when removing supported tokens

Category: Denial of Service Vulnerability

Content

In the StoneBTCVault contract, privileged roles can add/remove supported wrapped BTC tokens through the addSupportedTokens/removeSupportedTokens functions. When performing the removeSupportedTokens operation, the contract checks that the balance of the token being removed must be zero. This can be easily exploited, as users can donate a small amount of tokens to prevent the removeSupportedTokens function from working properly.

It is also important to note that when users withdraw, the contract converts the decimal to the decimal of the token

being withdrawn. When the decimal of this token is smaller than the decimal of STONE BTC, there will always be a small amount of dust tokens left in the vault. This indirectly prevents the removeSupportedTokens function from working correctly.

Code location:

src/StoneBTCVault.sol#L228

src/StoneBTCVault.sol#L138

```
    function withdraw(
        address _token,
        uint256 _stoneBTCAmount
    ) external {
        ...
        uint256 tokenAmount = remainingAmount / (10**(18-tokenDecimals[_token]));
        TransferHelper.safeTransfer(_token, msg.sender, tokenAmount);
        emit Withdraw(msg.sender, _stoneBTCAmount, _token, tokenAmount);
    }

    function removeSupportedTokens(
        address _token
    ) external onlyRole(SUPPORTED_TOKEN_OPERATION_ROLE) {

        if (ERC20(_token).balanceOf(address(this)) != 0) revert
NonEmptySupportedToken();

        ...
    }
```

Solution

It is recommended to use a global variable to record the actual deposit amount of tokens in the contract and implement a governance mechanism to withdraw the difference between the contract balance and the recorded deposit amount. This can help mitigate the aforementioned risks.

Status

Acknowledged; After communicating with the project team, the project team stated that when encountering such a situation, the project team will deposit other types of wrapped BTC to mint STONE BTC and take out the dust tokens from the contract.

[N5] [Low] The actual deposit amount may differ from the contract balance

Category: Design Logic Audit

Content

In the StoneBTCVault contract, the `_checkDepositAllowed` function checks the depositCapacity based on the balance of wrapped BTC tokens in the contract. Similarly, the `getDepositAmounts` function retrieves token balances to determine the deposit amounts. These values may differ from the actual deposit amounts made by users. Users might accidentally transfer supported tokens into the vault, or some users might send small donations to the vault. Both scenarios will cause the above two functions to obtain amounts that are greater than the users' actual deposit amounts.

Code location:

src/StoneBTCVault.sol#L185

src/StoneBTCVault.sol#L200

```
    function _checkDepositAllowed(address _token, uint256 _amount, address _user)
internal view {
        if (supportedTokens[_token] == false) revert UnsupportedToken();
        if (depositPaused[_token]) revert DepositPaused();
        if (_amount + ERC20(_token).balanceOf(address(this)) >
depositCapacity[_token]) revert CapacityReached();
        if (whitelistMode[_token] && !depositWhitelist[_token][_user]) revert
NotWhitelisted();
    }

    function getDepositAmounts() external view returns (address[] memory, uint256[]
memory) {
        uint256 l = supportedTokensArray.length;
        address[] memory tokens = new address[](l);
        uint256[] memory amounts = new uint256[](l);

        for (uint256 i; i < l;) {
            address _token = supportedTokensArray[i];
            tokens[i] = _token;
            amounts[i] = ERC20(_token).balanceOf(address(this));
            unchecked { ++i; }
        }
        return (tokens, amounts);
    }
```

Solution

It is recommended to use a global variable to record the actual deposit amounts of tokens in the contract and implement a governance mechanism to withdraw the difference between the contract balance and the recorded deposit amounts. This can help mitigate the aforementioned risks.

Status

Acknowledged; After communicating with the project team, the project team stated that users would not have any motivation to donate to the contract, so no changes would be made.

[N6] [Suggestion] Not checking if withAmount is greater than 0 when retrieving all tokens

Category: Design Logic Audit

Content

In the Proposal contract, users can retrieve all their STONE tokens used for voting through the retrieveAllToken function. It uses a temporary variable withAmount to record the amount of STONE tokens that can be withdrawn. However, it does not check if withAmount is greater than 0 before initiating the transfer, which may result in the contract sending a 0 transfer and wasting gas.

Code location: src/Governance/Proposal.sol#L134

```
    function retrieveAllToken() external {
        uint256 withAmount;

        uint256 length = proposals.length();
        for (uint i; i < length; i++) {
            address addr = proposals.at(i);
            uint256 voteAmount = polls[msg.sender][addr];

            if (!canVote(addr) && voteAmount != 0) {
                polls[msg.sender][addr] = 0;
                withAmount = withAmount + voteAmount;

                emit RetrieveToken(addr, voteAmount);
            }
        }
        TransferHelper.safeTransfer(stoneToken, msg.sender, withAmount);
    }
```

Solution

It is recommended to check if withAmount is greater than 0 before executing the transfer.

Status

Acknowledged

[N7] [Medium] Risks of excessive privilege

Category: Authority Control Vulnerability Audit

Content

In the StoneBTC contract, the contract deployer is set as the DEFAULT_ADMIN_ROLE. The admin role can arbitrarily change the MINTER_ROLE/BURNER_ROLE roles, which are involved in minting and burning STONE BTC. This leads to the risk of excessive privileges.

Similarly, in the StoneBTCVault and Proposal contracts, the initial DEFAULT_ADMIN_ROLE is also the deployer. Assigning sensitive permissions to an EOA address not only creates the risk of excessive privileges but also introduces a single point of failure.

Code location:

src/Governance/Proposal.sol#L54

src/StoneBTCVault.sol#L53

src/Token/StoneBTC.sol#L17

```
    constructor(
        string memory _name,
        string memory _symbol
    ) ERC20(_name, _symbol) {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    }
```

Solution

In the short term, transferring the protocol admin permissions to a multi-signature wallet can effectively mitigate the single point of failure risk, but it cannot address the excessive privilege risk. In the long run, once the protocol is running stably, transferring sensitive permissions to community governance can alleviate the excessive privilege risk and enhance community trust.

Status

Acknowledged; After communicating with the project team, the project team stated that after the project is deployed, MPC or multi-signature will be used to mitigate the above risks.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002409030001	SlowMist Security Team	2024.08.30 - 2024.09.03	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 1 medium risk, 2 low risks, and 3 suggestions. All the findings were acknowledged. Since the protocol has not yet been deployed to the mainnet, the risk of excessive privilege has not been resolved, so it is still in a medium-risk state.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist