Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.10.09, the SlowMist security team received the StakeStone team's security audit application for STONE BTC Vault, 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

This audit primarily focuses on the core functions of the STONEBTC Vault, including deposits, withdrawals, and price oracles. Users can mint STONEBTC by depositing assets supported by the vault.

The price of STONEBTC is constantly changing, depending on the amount and value of the underlying assets in the vault, as well as the total supply of STONEBTC. The price of the underlying assets in the vault is provided by an external oracle. Users cannot directly withdraw the underlying assets from the vault; they need to apply for withdrawal through the WithdrawalQueue.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Token compatibility issues	Design Logic Audit	Low	Acknowledged
N2	Incorrect price acquisition of LP tokens	Design Logic Audit	High	Acknowledged
N3	Lack of return value check	Others	Suggestion	Fixed
N4	No check on Oracle price validity	Design Logic Audit	Medium	Fixed
N5	Risk of excessive privilege	Authority Control Vulnerability Audit	Medium	Acknowledged
N6	Missing the event records	Others	Suggestion	Acknowledged
N7	Minimum number of receipts calculated in advance	Design Logic Audit	Information	Acknowledged
N8	Missing zero address check	Others	Suggestion	Fixed
N9	Array Length Mismatch	Others	Low	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

Project address:

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

commit: 10fdc48fade04c0312bec5a47a62ebb5fdeb0043

Fixed Version:

Project address:

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

commit: aef34121fc8c84a3cf029ad45d469c85407d77f8

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:

AssetVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deposit	External	Can Modify State	-
depositMultiple	External	Can Modify State	-
requestWithdraw	External	Can Modify State	onlyRole
cancelWithdrawal	External	Can Modify State	-
processWithdrawal	External	Can Modify State	onlyRole
processAllWithdrawal	External	Can Modify State	onlyRole
_finalizeWithdraw	Internal	Can Modify State	-
withdrawFromVault	External	Can Modify State	onlyRole
repayToVault	External	Can Modify State	onlyRole
addUnderlyingToken	External	Can Modify State	onlyRole
removeUnderlyingToken	External	Can Modify State	onlyRole
setDepositPause	External	Can Modify State	onlyRole
setWithdrawPause	External	Can Modify State	onlyRole
setWithdrawFeeRate	External	Can Modify State	onlyRole

AssetVault			
setDepositFeeRate	External	Can Modify State	onlyRole
setFeeRecipient	External	Can Modify State	onlyRole
getUnderlyings	External	-	-
getRequestsLength	External	-	-
getRequestWithdrawals	External	-	-

OracleConfigurator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updateOracle	External	Can Modify State	onlyRole
getPrice	External	-	-

Oracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getPrice	External	-	-

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

ChainlinkOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Oracle
getPrice	External	-	-

BTCPeggedOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Oracle
getPrice	External	-	-

WithdrawalProcessor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
processWithdrawals	External	Can Modify State	onlyRole
processWithdrawals	External	Can Modify State	onlyRole
callback	External	Can Modify State	-
setWithdrawalQueue	External	Can Modify State	onlyRole
withdrawLeftTokens	External	Can Modify State	onlyRole

WithdrawalQueue			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
queueWithdrawal	External	Can Modify State	-
cancelWithdrawals	External	Can Modify State	-
processWithdrawals	External	Can Modify State	-

WithdrawalQueue			
processWithdrawals	External	Can Modify State	-
setMinRequestAmount	External	Can Modify State	onlyRole
setActiveDuration	External	Can Modify State	onlyRole
getQueueLength	External	-	-
getQueueReceipts	External	-	-
checkReceipts	External	-	-

4.3 Vulnerability Summary

[N1] [Low] Token compatibility issues

Category: Design Logic Audit

Content

In the AssetVault contract, users can transfer underlying asset tokens into the contract and mint LP tokens by calling the deposit and depositMultiple functions. But the deposit amount is not calculated based on the difference in the contract balance before and after the transfer. Instead, the _amount parameter passed in is directly used. Therefore, if the transferred tokens are reflective tokens(deflation/inflation type tokens), the actual deposit amount will not be equal to the number of tokens transferred into the contract, resulting in a calculation error.

Code Location:

src/vault/AssetVault.sol

```
function deposit(
     address _token,
     uint256 _amount,
     uint256 _minLpAmount
 ) external returns (uint256 mintAmount) {
     ...

     TransferHelper.safeTransferFrom(
         _token,
         msg.sender,
         address(this),
```

```
            _amount
        );

        ...
    }

    function depositMultiple(
        address[] memory _tokens,
        uint256[] memory _amounts,
        uint256 _minLpAmount
    ) external returns (uint256 totalMintAmount) {
        ...

        for (i; i < length; i++) {
            ...

            TransferHelper.safeTransferFrom(
                token,
                msg.sender,
                address(this),
                amount
            );

            ...
        }

        ...
    }
```

Solution

It is recommended to use the difference in the token balance in the contract before and after the user's transfer as the actual deposit amount of the user, instead of directly using the _amount parameter passed in.

Status

Acknowledged; Project team response: We will not support reflective tokens.

[N2] [High] Incorrect price acquisition of LP tokens

Category: Design Logic Audit

Content

In the AssetVault contract, the processAllWithdrawal function is used to handle all withdrawal requests, and it orderly calls the _finalizeWithdraw function through a for loop to burn LP tokens and release the underlying asset tokens.

In the _finalizeWithdraw function, the amount of underlying asset tokens to be released is calculated using the price

of LP tokens (the lpPrice parameter passed in) as the numerator. However, the LP token price used in each loop iteration is the same. This may lead to unintended calculation errors because, normally, the price of LP tokens dynamically changes with the reserve amount of underlying asset tokens and the total supply of LP tokens.

Code Location:

src/vault/AssetVault.sol#L299

```
function processAllWithdrawal() external onlyRole(WITHDRAWAL_PROCESS_ROLE) {
    uint256 length = withdrawalRequests.length;
    if (length == 0) revert InvalidArrayLength();

    uint256 lpPrice = oracleConfigurator.getPrice(address(lpToken));

    uint256 i;
    for (i; i < length; i++) {
        WithdrawalRequest memory withdrawalRequest = withdrawalRequests[i];

        _finalizeWithdraw(
            withdrawalRequest,
            lpPrice,
            withdrawFeeRate,
            feeRecipient
        );
    }

    delete withdrawalRequests;
}
```

Solution

It is recommended to re-fetch the LP token price in each loop iteration before passing it into the _finalizeWithdraw function, rather than using the same price.

Status

Acknowledged; Project team response: The lp price will remain the same in each loop iteration, so we put it out of the iteration to save the gas during calculation.

[N3] [Suggestion] Lack of return value check

Category: Others

Content

In the WithdrawalProcessor contract, the callback function serves as an intermediary in the withdrawal process, transferring LP tokens to the assetVault contract by externally calling its requestWithdraw function and submitting a withdrawal request; it then calls the processWithdrawal function of the assetVault contract to handle this withdrawal request. However, the processWithdrawal function will refund the LP tokens and return a finalizedAmount value of 0 when the number of redeemed asset tokens is less than expected. There is no check on the return value in the callback function, which will lead to errors.

Code Location:

src/withdraw/WithdrawalProcessor.sol#L64

```
    function callback(
        address _token,
        uint256 _lpAmount,
        uint256 _tokenAmount
    ) external {
        ...

        assetVault.processWithdrawal(reqId);

        ...
    }
```

Solution

It is recommended to add a check for the return value of the processWithdrawal function in the callback function to ensure that the returned finalizedAmount value is not equal to 0. This way, subsequent authorization and transfer operations can be avoided, thus saving more gas.

Status

Fixed

[N4] [Medium] No check on Oracle price validity

Category: Design Logic Audit

Content

In the ChainlinkOracle contract, users can obtain prices from the Chainlink oracle through the getPrice function, but there is no further check on the price returned by the oracle. It is important to note that answer is of type int256, and

its value may be less than 0. Failing to check this may result in obtaining unexpected prices. updatedAt is the update time of the oracle price. Once the price deviation exceeds the threshold or the heartbeat time is reached, the price will be updated. Failing to check this will make it impossible to ensure that the obtained price is valid in real-time. Moreover, if the protocol is deployed on L2, the availability of the sequencer will affect the validity of the price. Therefore, when obtaining prices from Chainlink on L2, it is necessary to check the availability of the L2 sequencer.

Code Location:

src/oracle/ChainlinkOracle.sol

```
    function getPrice() external view override returns (uint256 price) {
        (, int answer, , , ) = dataFeed.latestRoundData();
        price = uint256(answer);

        ...
    }
```

Solution

It is recommended to check that answer must be greater than 0 when obtaining prices, check that the difference between updatedAt and the current time must be less than its heartbeat interval, and when obtaining prices on L2, the availability of the sequencer must be checked through the Chainlink sequencerUptimeFeed.

Status

Fixed

[N5] [Medium] Risk of excessive privilege

Category: Authority Control Vulnerability Audit

Content

1.In the AssetVault contract, the assets management role can transfer any tokens away from this contract by calling the rescueTokens function. If the privilege is lost or misused, this may have an impact on the user's assets.

Code Location:

src/vault/AssetVault.sol#L387-403

```
    function withdrawFromVault(
        address[] memory _tokens,
```

```
        uint256[] memory _amounts
    ) external onlyRole(ASSETS_MANAGEMENT_ROLE) {
        ...
    }
```

2.In the OracleConfigurator contract, the oracle manager role can set the price oracle addresses for different tokens by calling the updateOracle function. If the privilege is lost or misused, it could lead to the token's price oracle being set to a malicious one, resulting in a loss of user funds.

Code Location:

src/oracle/OracleConfigurator.sol#L21-31

```
    function updateOracle(
        address _token,
        address _oracle
    ) external onlyRole(ORACLE_MANAGER_ROLE) {
        ...
    }
```

3.In the Token contract, the minter role can mint tokens arbitrarily to any address, and the burner role can burn tokens held by any address at will. If the addresses of the minter and burner roles are controlled by EOA (Externally Owned Account) addresses, then if the authority is lost or stolen, it may affect users' assets.

Code Location:

src/token/Token.sol

```
    function mint(address _to, uint256 _amount) external onlyRole(MINTER_ROLE) {
        _mint(_to, _amount);
    }

    function burn(
        address _from,
        uint256 _amount
    ) external onlyRole(BURNER_ROLE) {
        _burn(_from, _amount);
    }
```

Solution

In the short term, during the early stages of the project, the protocol may need to frequently set various parameters to ensure the stable operation of the protocol. Therefore, transferring the ownership of core roles to a multisig management can effectively solve the single-point risk, but it cannot mitigate the excessive privilege risk. In the long run, after the protocol stabilizes, transferring the owner ownership to community governance and executing through a timelock can effectively mitigate the excessive privilege risk and increase the community users' trust in the protocol.

Status

Acknowledged; Project team response: Since the underlying assets need to be withdrawn from the protocol and deployed to various yield-generating scenarios, such as Babylon staking and RWA yield farming, and because this process cannot be completed in a decentralized way directly on Ethereum, it is inevitable that assets will need to be withdrawn from the contract. After the protocol goes live, the management of these assets will be handled through a multi-signature wallet, jointly managed by the StakeStone Team, as well as asset management platforms like CIAN and Plume Network.

[N6] [Suggestion] Missing the event records

Category: Others

Content

In the following contract, the owner role can modify the withdrawalQueueAddr parameters, but there is no event log in the function.

Code Location:

src/withdraw/WithdrawalProcessor.sol#L69-74

```
    function setWithdrawalQueue(
        address _withdrawalQueue
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        if (withdrawalQueueAddr != address(0)) revert NotZeroAddress();
        withdrawalQueueAddr = _withdrawalQueue;
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Acknowledged

[N7] [Information] Minimum number of receipts calculated in advance

Category: Design Logic Audit

Content

In the WithdrawalQueue contract, Users submit withdrawal requests by calling the queueWithdrawal function, where the _minReceiveAmount parameter represents the amount of underlying tokens the user expects to receive after the withdrawal. This value needs to be calculated in advance by the user before submitting the withdrawal request to ensure that _minReceiveAmount matches the expected token amount. If the _minReceiveAmount provided is less than the amount of tokens redeemed from the assetVault, other users can call the processWithdrawals function to claim the difference.

This is because the amount of tokens returned to the user in the processWithdrawals function directly depends on the value of _minReceiveAmount, rather than the difference in the amount of tokens obtained by the contract before and after the function call.

When there is a time difference between requesting a withdrawal and executing it, and if the price of the token fluctuates during this time difference, it will also result in a portion of the tokens being left in the WithdrawalProcessor contract at the time of withdrawal. This is because the callback function only authorizes the WithdrawalQueue contract to release the amount of tokens that the user expects to receive (i.e., the value of minReceiveAmount). However, the withdrawal process role can call the withdrawLeftTokens function to withdraw this portion of the tokens.

Code Location:

src/withdrawal/WithdrawalQueue.sol

```
    function queueWithdrawal(
        address _requestToken,
        uint256 _lpAmount,
        uint256 _minReceiveAmount
    ) external {
```

```
        ...

        withdrawalQueue.push(
            WithdrawalReceipt({
                requester: msg.sender,
                requestToken: _requestToken,
                lpAmount: _lpAmount,
                minReceiveAmount: _minReceiveAmount,
                deadline: deadline,
                isWithdrawn: false
            })
        );

        ...
    }

    ...

    function processWithdrawals(uint256[] memory _ids) external {
        uint256 length = _ids.length;
        uint256 i;
        for (i; i < length; i++) {
            ...

            TransferHelper.safeTransferFrom(
                requestToken,
                msg.sender,
                requester,
                minReceived
            );

            ...
        }
    }
```

Solution

N/A

Status

Acknowledged

[N8] [Suggestion] Missing zero address check

Category: Others

Content

Several of the following functions do not check for a zero address when setting an address.

Code location:

src/withdrawal/WithdrawalProcessor.sol#L21-26

```
    constructor(address _assetVault, address _lpToken) {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);

        assetVaultAddr = _assetVault;
        lpToken = _lpToken;
    }
```

src/withdrawal/WithdrawalQueue.sol#L44-54

```
    constructor(
        uint256 _minRequestAmount,
        address _lpToken,
        address _processor
    ) {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);

        lpToken = _lpToken;
        processor = _processor;
        minRequestAmount = _minRequestAmount;
    }
```

src/oracle/BTCPeggedOracle.sol#L9-12

```
    constructor(address _token, string memory _name) Oracle(_token, _name) {
        token = _token;
        name = _name;
    }
```

Solution

It is recommended that an address non-zero check should be added.

Status

Fixed

[N9] [Low] Array Length Mismatch

Category: Others

Content

In the AssetVault contract,the depositMultiple function allows users to deposit multiple tokens at once and receive corresponding LP tokens based on token values and the LP token price. However, the function lacks a check to ensure that the _tokens and _amounts arrays have the same length, which could lead to out-of-bounds errors or undefined behavior.

Code Location:

src/vault/AssetVault.sol#L154-203

```
    function depositMultiple(
        address[] memory _tokens,
        uint256[] memory _amounts,
        uint256 _minLpAmount
    ) external returns (uint256 totalMintAmount) {
        ...
    }
```

Solution

It is recommended to add a length check at the beginning of the function to ensure both arrays are of equal size.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002410120002	SlowMist Security Team	2024.10.09 - 2024.10.12	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, 2 medium risk, 2 low risk, 4 suggestion and 1 information. All the findings were fixed or acknowledged. Since the project has not yet been deployed to the mainnet and the permissions of the core roles have not yet been transferred, the risk level reported is temporarily medium.

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