Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.12.07, the SlowMist security team received the StakeStone team's security audit application for StakeStone, 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

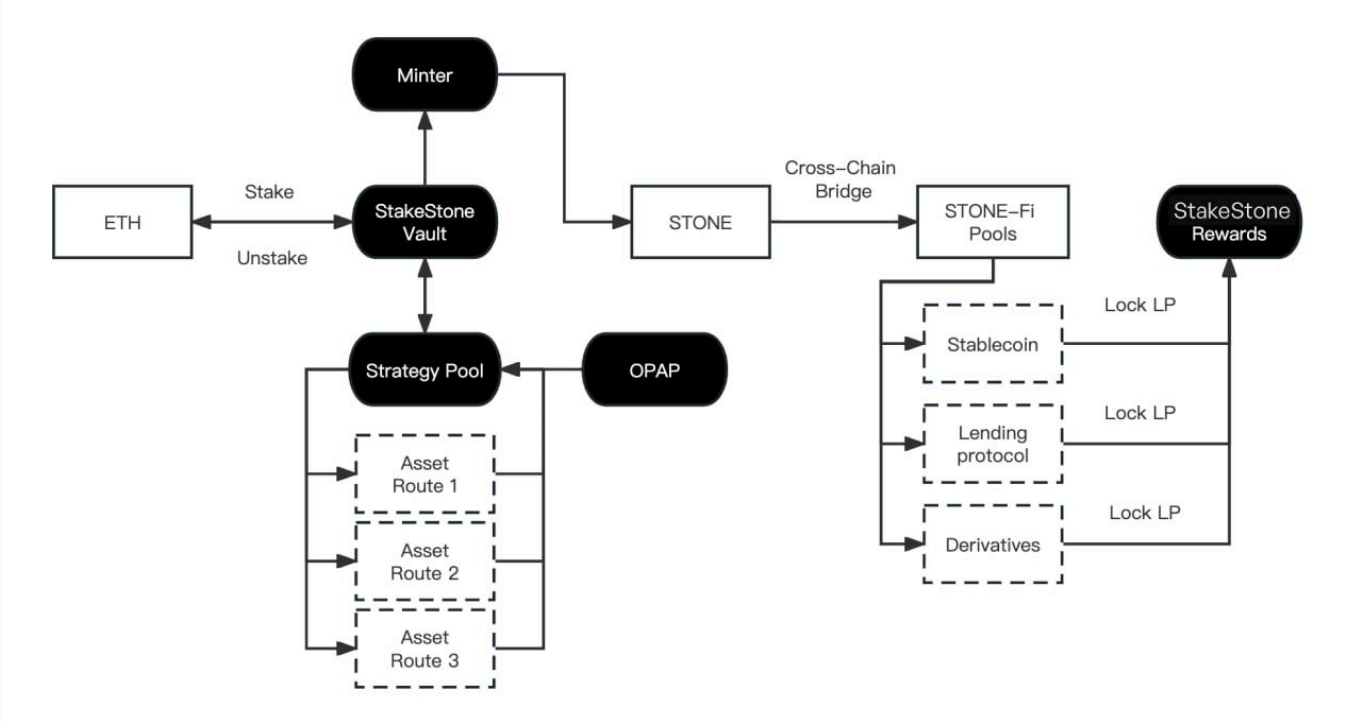
StakeStone is an omni-chain LST (Liquid Staking Token) protocol aiming to bring native staking yields and liquidity to Layer 2s in a decentralized manner. With its highly scalable architecture, StakeStone not only supports leading staking pools but is also compatible with the upcoming restaking. Meanwhile, it establishes a multi-chain liquidity market based on STONE, StakeStone's native LST, provides users of STONE with more use cases and yield opportunities.

Based on LayerZero, STONE is a non-rebase OFT (Omnichain Fungible Token) that supports both assets and prices to be transferred across multiple blockchains seamlessly.

StakeStone Vault serves as the fund buffering pool, retaining the deposited ETH within the contract until a new settlement occurs, at which point it will be deployed to the underlying strategy pool.

The Minter function decouples STONE token minting from its underlying assets. This separation allows for independent adjustments to the underlying assets and the circulation of issued STONE tokens, ensuring a higher level of token stability.

Strategy pool adopts a whitelist mechanism governed by OPAP, demonstrating a high level of asset compatibility, such as staking pools, restaking protocols and so on. Simultaneously, asset risks will be isolated within each individual strategy route, preventing cross-contamination of risks.



3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Inflation attack in StoneVault	Design Logic Audit	High	Acknowledged
N2	Missing setting rebaseTime when initializing	Others	Low	Fixed

NO	Title	Category	Level	Status
N3	Risks of incorrect withdrawableAmount InPast updates	Design Logic Audit	High	Fixed
N4	Missing check when migrating the vault	Design Logic Audit	High	Fixed
N5	Incorrect return value if the balance is sufficient	Design Logic Audit	High	Fixed
N6	Incorrect withdrawal quantity calculation	Design Logic Audit	Medium	Fixed
N7	Redundant code	Others	Suggestion	Fixed
N8	Incorrect PendingValue calculations in the STETHHoldingStrategy	Design Logic Audit	Medium	Fixed
N9	Lack of event records	Others	Suggestion	Acknowledged
N10	Authority transfer enhancement	Authority Control Vulnerability Audit	Suggestion	Acknowledged
N11	Lack of CrossChain fee checking in the bridgeTo function	Design Logic Audit	Low	Acknowledged
N12	Missing check for dstChainId on initialisation	Others	Suggestion	Acknowledged
N13	Lack of scope check	Design Logic Audit	Low	Acknowledged
N14	Potential governance attacks	Design Logic Audit	Information	Acknowledged
N15	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Fixed
N16	Using `block.timestamp` for swap deadline offers no protection	Reordering Vulnerability	Suggestion	Acknowledged
N17	Missing return value check when adding strategies	Others	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/stakestone/stone-vault-v1

commit: 2581ced4983fc72459d4301ef0e5496636371f68

Fixed Version:

https://github.com/stakestone/stone-vault-v1

commit: 67aa709f10b28b942ca9983f1e9951295ebd2a0e

Audit scope:

- contracts/AssetsVault.sol
- contracts/StoneVault.sol
- contracts/governance/*
- contracts/interfaces/*
- contracts/libraries/*
- contracts/mining/DepositBridge.sol
- contracts/strategies/RETHHoldingStrategy.sol
- contracts/strategies/SFraxETHHoldingStrategy.sol
- contracts/strategies/STETHHoldingStrategy.sol
- contracts/strategies/Strategy.sol
- contracts/strategies/StrategyController.sol
- contracts/strategies/SwappingAggregator.sol
- contracts/token/*

The main network address of the contract is as follows:

swappingAggregatorAddr.sol: 0x15469528C11E8Ace863F3F9e5a8329216e33dD7d,

stoneAddr.sol: 0x7122985656e38BDC0302Db86685bb972b145bD3C,

minterAddr.sol: 0xEc306E46549A7E8f4fCE823D3058f2D134133B17,

stoneVaultAddr.sol: 0xA62F9C5af106FeEE069F38dE51098D9d81B90572,

proposalAddr.sol: 0x3aa0670E24Cb122e1d5307Ed74b0c44d619aFF9b,

strategyControllerAddr.sol: 0x396aBF9fF46E21694F4eF01ca77C6d7893A017B2,

assetsVaultAddr.sol: 0x9485711f11B17f73f2CCc8561bcae05BDc7E9ad9,

stETHHoldingStrategyAddr.sol: 0xE942cDd0AF66aB9AB06515701fa3707Ec7deB93e

4.2 Visibility Description

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

Minter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
mint	External	Can Modify State	onlyVault
burn	External	Can Modify State	onlyVault
setNewVault	External	Can Modify State	onlyVault
getTokenPrice	Public	Can Modify State	-

Stone			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BasedOFT
mint	External	Can Modify State	onlyMinter
burn	External	Can Modify State	onlyMinter
sendFrom	Public	Payable	-
updatePrice	External	Payable	-
setEnableFor	External	Payable	onlyOwner
setCapFor	External	Payable	onlyOwner

Stone			
tokenPrice	Public	Can Modify State	-
getQuota	External	-	-

StoneCross			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OFT
sendFrom	Public	Payable	-
_nonblockingLzReceive	Internal	Can Modify State	-
getQuota	External	Can Modify State	-

StoneVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deposit	External	Payable	nonReentrant
depositFor	External	Payable	nonReentrant
_depositFor	Internal	Can Modify State	-
requestWithdraw	External	Can Modify State	nonReentrant
cancelWithdraw	External	Can Modify State	nonReentrant
instantWithdraw	External	Can Modify State	nonReentrant
rollToNextRound	External	Can Modify State	-
addStrategy	External	Can Modify State	onlyProposal
destroyStrategy	External	Can Modify State	onlyOwner
clearStrategy	External	Can Modify State	onlyOwner

StoneVault			
updatePortfolioConfig	External	Can Modify State	onlyProposal
updateProposal	External	Can Modify State	onlyProposal
migrateVault	External	Can Modify State	onlyProposal
currentSharePrice	Public	Can Modify State	-
getVaultAvailableAmount	Public	Can Modify State	-
setWithdrawFeeRate	External	Can Modify State	onlyOwner
setFeeRecipient	External	Can Modify State	onlyOwner
setRebaseInterval	External	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-

AssetsVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deposit	External	Payable	-
withdraw	External	Can Modify State	onlyPermit
setNewVault	External	Can Modify State	onlyPermit
getBalance	External	-	-
<Receive Ether>	External	Payable	-

RETHHoldingStrategy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Strategy
deposit	Public	Payable	onlyController notAtSameBlock

RETHHoldingStrategy			
withdraw	Public	Can Modify State	onlyController notAtSameBlock
instantWithdraw	Public	Can Modify State	onlyController notAtSameBlock
_withdraw	Internal	Can Modify State	-
clear	Public	Can Modify State	onlyController
getAllValue	Public	Can Modify State	-
getInvestedValue	Public	Can Modify State	-
getPendingValue	Public	Can Modify State	-
checkPendingStatus	Public	Can Modify State	-
setRouter	External	Can Modify State	onlyGovernance
<Receive Ether>	External	Payable	-

SFraxETHHoldingStrategy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Strategy
deposit	Public	Payable	onlyController notAtSameBlock
withdraw	Public	Can Modify State	onlyController notAtSameBlock
instantWithdraw	Public	Can Modify State	onlyController notAtSameBlock
_withdraw	Internal	Can Modify State	-
clear	Public	Can Modify State	onlyController
getAllValue	Public	Can Modify State	-
getInvestedValue	Public	Can Modify State	-
getPendingValue	Public	Can Modify State	-
checkPendingStatus	Public	-	-

SFraxETHHoldingStrategy			
setRouter	External	Can Modify State	onlyGovernance
<Receive Ether>	External	Payable	-

STETHHoldingStrategy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Strategy
deposit	Public	Payable	onlyController notAtSameBlock
withdraw	Public	Can Modify State	onlyController notAtSameBlock
instantWithdraw	Public	Can Modify State	onlyController notAtSameBlock
clear	Public	Can Modify State	onlyController
getAllValue	Public	Can Modify State	-
getInvestedValue	Public	Can Modify State	-
getPendingValue	Public	Can Modify State	-
getClaimableValue	Public	Can Modify State	-
checkPendingStatus	Public	Can Modify State	-
claimPendingAssets	External	Can Modify State	-
claimAllPendingAssets	External	Can Modify State	-
checkPendingAssets	Public	Can Modify State	-
setWithdrawQueueParams	External	Can Modify State	onlyGovernance
setRouter	External	Can Modify State	onlyGovernance
<Receive Ether>	External	Payable	-

Strategy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deposit	Public	Payable	onlyController notAtSameBlock
withdraw	Public	Can Modify State	onlyController notAtSameBlock
instantWithdraw	Public	Can Modify State	onlyController notAtSameBlock
clear	Public	Can Modify State	onlyController
execPendingRequest	Public	Can Modify State	-
getAllValue	Public	Can Modify State	-
getPendingValue	Public	Can Modify State	-
getInvestedValue	Public	Can Modify State	-
checkPendingStatus	Public	Can Modify State	-
setGovernance	External	Can Modify State	onlyGovernance
setBufferTime	External	Can Modify State	onlyGovernance

StrategyController			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
onlyRebaseStrategies	External	Can Modify State	-
forceWithdraw	External	Can Modify State	onlyVault
setStrategies	External	Can Modify State	onlyVault
addStrategy	External	Can Modify State	onlyVault
rebaseStrategies	External	Payable	onlyVault
destroyStrategy	External	Can Modify State	onlyVault

StrategyController			
_rebase	Internal	Can Modify State	-
_repayToVault	Internal	Can Modify State	-
_depositToStrategy	Internal	Can Modify State	-
_withdrawFromStrategy	Internal	Can Modify State	-
_forceWithdraw	Internal	Can Modify State	-
getStrategyValue	Public	Can Modify State	-
getStrategyValidValue	Public	Can Modify State	-
getStrategyPendingValue	Public	Can Modify State	-
getAllStrategiesValue	Public	Can Modify State	-
getAllStrategyValidValue	Public	Can Modify State	-
getAllStrategyPendingValue	Public	Can Modify State	-
getStrategies	Public	-	-
_initStrategies	Internal	Can Modify State	-
_setStrategies	Internal	Can Modify State	-
clearStrategy	Public	Can Modify State	onlyVault
_clearStrategy	Internal	Can Modify State	-
_destoryStrategy	Internal	Can Modify State	-
_couldDestroyStrategy	Internal	Can Modify State	-
setNewVault	External	Can Modify State	onlyVault
<Receive Ether>	External	Payable	-

SwappingAggregator			
Function Name	Visibility	Mutability	Modifiers

SwappingAggregator			
<Constructor>	Public	Can Modify State	-
swap	External	Payable	-
swapOnUniV3	Internal	Can Modify State	nonReentrant
swapOnCurve	Internal	Can Modify State	nonReentrant
getBestRouter	Public	Can Modify State	-
getUniV3Out	Public	Can Modify State	-
getCurveOut	Public	Can Modify State	-
getCurveCoinIndex	Public	-	-
calMinimumReceivedAmount	Internal	-	-
setSlippage	External	Can Modify State	onlyGovernance
setUniRouter	External	Can Modify State	onlyGovernance
setCurveRouter	External	Can Modify State	onlyGovernance
setNewGovernance	External	Can Modify State	onlyGovernance
<Receive Ether>	External	Payable	-

DepositBridge			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
bridgeTo	Public	Payable	-
bridge	Public	Payable	nonReentrant
estimateSendFee	Public	-	-
<Receive Ether>	External	Payable	-

Proposal			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
propose	External	Can Modify State	onlyProposer
voteFor	External	Can Modify State	-
retrieveTokenFor	External	Can Modify State	-
retrieveAllToken	External	Can Modify State	-
execProposal	External	Can Modify State	-
setProposer	External	Can Modify State	onlyProposer
setVotePeriod	External	Can Modify State	onlyProposer
getProposal	Public	-	-
getProposals	Public	-	-
canVote	Public	-	-
canExec	Public	-	-
invoke	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [High] Inflation attack in StoneVault

Category: Design Logic Audit

Content

In the StoneVault contract, users can can deposit assets and obtain the corresponding share of the vault by calling the deposit function, But there is a risk of interest rate inflation attacks here:

Consider this example: bob finds out that alice is making a deposit (e.g. via mempool).

Pre-condition: no one deposit before(latestRoundID == 0)

Assume raito = 1e18.

Now, alice wants to deposit 1 (1 * 1e18 wei) WETH and the tx is spied on by the attacker(bob). Here is the breakdown:

	totalStone	AssetsVault.getBalance()
original state	0	0
(after) Step 1	1	1
(after) Step 2	1	1e18 + 1
(after) Step 3	1	2 * 1e18 + 1

1.bob front-runs alice and deposits 1 wei WETH and gets 1 share: since totalStone is 0, shares = amount = 1.

2.bob also transfers 1 * 1e18 wei WETH, making the WETH balance of the AssetsVault (AssetsVault.getBalance()) become 1e18 + 1 wei. And then directly call the rollToNextRoundId function to update the latestRoundId and price. (Since rebaseTime starts at 0, it can be called successfully directly).

3.alice deposits 1e18 wei WETH. However, alice gets 0 shares: 1e18 * 1 (totalStone) / (1e18 + 1) = 1e18 / (1e18 + 1) = 0. Since alice gets 0 shares, totalStone remains at 1.

4.bob still has the 1 only share ever minted, thus after waiting for the next rollToNextWETH function call for updating the price and the withdrawal of that 1 share takes away everything in the AssetsVault, including the alice's 1e18 wei WETH.(Directly by calling the instantWithdraw function and passing in _amount parameter with a value of 0, _shares parameter with a value of 1).

Code Location:

contracts/StoneVault.sol#L150-173

```
    function _depositFor(
        uint256 _amount,
        address _user
    ) internal returns (uint256 mintAmount) {
        require(_amount != 0, "too small");
```

```
        uint256 sharePrice;
        uint256 currSharePrice = currentSharePrice();
        if (latestRoundID == 0) {
            sharePrice = MULTIPLIER;
        } else {
            uint256 latestSharePrice = roundPricePerShare[latestRoundID - 1];
            sharePrice = latestSharePrice > currSharePrice
                ? latestSharePrice
                : currSharePrice;
        }

        mintAmount = (_amount * MULTIPLIER) / sharePrice;

        AssetsVault(assetsVault).deposit{value: address(this).balance}();
        Minter(minter).mint(_user, mintAmount);

        emit Deposit(_user, _amount, mintAmount, latestRoundID);
    }
```

contracts/StoneVault.sol#L436-453

```
    function currentSharePrice() public returns (uint256 price) {
        Stone stoneToken = Stone(stone);
        uint256 totalStone = stoneToken.totalSupply();
        if (
            latestRoundID == 0 ||
            totalStone == 0 ||
            totalStone == withdrawingSharesInPast
        ) {
            return MULTIPLIER;
        }

        uint256 etherAmount = AssetsVault(assetsVault).getBalance() +
            StrategyController(strategyController).getAllStrategiesValue() -
            withdrawableAmountInPast;
        uint256 activeShare = totalStone - withdrawingSharesInPast;

        return (etherAmount * MULTIPLIER) / activeShare;
    }
```

Solution

Introduce a offset for internal accounting.

Refs: https://ethereum-magicians.org/t/address-eip-4626-inflation-attacks-with-virtual-shares-and-assets/12677/3

Status

Acknowledged; Response from the project team: Due to the large changes, no fixes will be made for the time being. And since the contract is already online and some eth has been pre-deposited, it does not pose a risk to the current contract on the chain.

[N2] [Low] Missing setting rebaseTime when initializing

Category: Others

Content

In the StoneVault contract, rebaseTime defaults to 0 and is not set in the constructor function. This could result in any user can call the function directly after the vault is created, potentially combining with other issues to have a significant impact.(Refer to the N1 issue)

Code Location:

contracts/StoneVault.sol#L347

```
    function rollToNextRound() external {
        require(
            block.timestamp > rebaseTime + rebaseTimeInterval,
            "already rebased"
        );

        ...
}
```

Solution

It is recommended to set the rebaseTime to block.timestamp in the constructor function.

Status

Fixed

[N3] [High] Risks of incorrect withdrawableAmountInPast updates

Category: Design Logic Audit

Content

In the StoneVault contract, the rollToNextRound function moves the contract to the next round and updates the current round, including withdrawableAmountInPast.

However, the price used in the withdrawableAmountInPast update is newSharePrice instead of the current round price (undPricePerShare [latestRoundID]). This results in newSharePrice being larger than the current round price (roundPricePerShare [latestRoundID]) if newSharePrice > previewSharePrice.

In the withdrawal operation (instantWithdraw), the number of user withdrawals is actually calculated using roundPricePerShare, so if newSharePrice is larger than roundPricePerShare in the round of commit withdrawals, it may cause the withdrawal withdrawableAmountInPast is actually larger than the total remaining withdrawals.

Then there may be the following situation: hypothesis After most withdrawals, totalStone has very little left (such as 1wei), and withdrawableAmountInPast the result of the price bias in statistics mentioned above is actually larger than expected. Then when calculating the current price (currentSharePrice), the calculation of etherAmount may be 0 or even an error due to overflow.

Code Location:

contracts/StoneVault.sol#L387

```
function rollToNextRound() external {
    ...

    uint256 newSharePrice = currentSharePrice();
    roundPricePerShare[latestRoundID] = previewSharePrice < newSharePrice
        ? previewSharePrice
        : newSharePrice;

    ...

    withdrawableAmountInPast =
        withdrawableAmountInPast +
        VaultMath.sharesToAsset(withdrawingSharesInRound, newSharePrice);
    withdrawingSharesInRound = 0;
    rebaseTime = block.timestamp;

    emit RollToNextRound(latestRoundID, vaultIn, vaultOut, newSharePrice);
}
```

Solution

It is recommended to use roundPricePerShare [latestRoundID] for calculation when updating withdrawableAmountInPast in rollToNextRound function instead of newSharePrice.

Status

Fixed

[N4] [High] Missing check when migrating the vault

Category: Design Logic Audit

Content

In the StoneVault contract, the migrateVault function is used to update the stoneVault address in the minter, assetsVault, and strategyController to achieve the effect of migration contracts. However, the migration does not check whether there are pending withdrawal requests in the current stoneVault contract.

If the withdrawal request pending in the old stoneVault contract has not been finished during the migration process, then the data has been reset in the new stoneVault contract after the migration, which will cause the shares (stone tokens) transferred to the old stoneVault contract when the user committed the withdrawal request before to be locked and cannot be retrieved.

Code Location:

contracts/StoneVault.sol#L430-434

```
    function migrateVault(address _vault) external onlyProposal {
        Minter(minter).setNewVault(_vault);
        AssetsVault(assetsVault).setNewVault(_vault);
        StrategyController(strategyController).setNewVault(_vault);
    }
```

Solution

It is recommended to check whether there are outstanding withdrawal requests in the contract when migrating, or set up a function to extract shares of unprocessed withdrawal requests in the old vault contract.

Status

Fixed

[N5] [High] Incorrect return value if the balance is sufficient

Category: Design Logic Audit

Content

When executing a withdrawal, if the eth balance in the AssetsVault contract is insufficient, the forceWithdraw function of the controller contract will be called to make up the remaining eth by forcing a withdrawal.

In the StrategyController contract, if the eth balance of this contract is sufficient, the return value of actualAmount should normally be just what is needed (i.e. the passed ethAmount). But here all eth balances in the contract are returned, which may cause the user to withdraw more tokens than expected.

Code Location:

contracts/strategies/StrategyController.sol#L63

```
function forceWithdraw(
    uint256 _amount
) external onlyVault returns (uint256 actualAmount) {
    uint256 balanceBeforeRepay = address(this).balance;

    if (balanceBeforeRepay >= _amount) {
        _repayToVault();

        actualAmount = balanceBeforeRepay;
    } else {
        actualAmount =
            _forceWithdraw(_amount - balanceBeforeRepay) +
            balanceBeforeRepay;
    }
}
```

Solution

It is recommended that the value of actualAmount returned should be exactly the numeric value passed in and not the total ETH balance in the contract.

Status

Fixed

[N6] [Medium] Incorrect withdrawal quantity calculation

Category: Design Logic Audit

Content

If the ETH balance in the contract is insufficient during forced withdrawals, the instantWithdraw function in each strategy will be called in a loop to make up the difference. The number of withdrawals for each strategy is calculated as _amount * ratios [strategy])/ONE_HUNDRED_PERCENT.

Then there is a situation where if the sum of all ratios is less than ONE_HUNDRED_PERCENT, then the total number of forced withdrawals will be less than expected. (This is possible because the ratio of each strategy is set to only require the sum of all ratios to be less than or equal to ONE_HUNDRED_PERCENT, or a strategy is cleared).

Code Location:

contracts/strategies/StrategyController.sol#L187

```
function _forceWithdraw(
    uint256 _amount
) internal returns (uint256 actualAmount) {
    uint256 length = strategies.length();
    for (uint i; i < length; i++) {
        address strategy = strategies.at(i);

        uint256 withAmount = (_amount * ratios[strategy]) /
            ONE_HUNDRED_PERCENT;

        if (withAmount != 0) {
            actualAmount =
                Strategy(strategy).instantWithdraw(withAmount) +
                actualAmount;
        }
    }

    _repayToVault();
}
```

Solution

It is recommended that the ONE_HUNDRED_PERCENT in the calculation should be changed to the sum of all ratios.

Status

Fixed

[N7] [Suggestion] Redundant code

Category: Others

Content

There are useless codes in the file and codes that are not used in actual business.

Code Location:

contracts/strategies/StrategyController.sol#L51-53

```
    function onlyRebaseStrategies() external {
        _rebase(0, 0);
    }
```

Solution

It is recommended to remove redundant commented code and useless code.

Status

Fixed

[N8] [Medium] Incorrect PendingValue calculations in the STETHHoldingStrategy

Category: Design Logic Audit

Content

In the STETHHoldingStrategy contract, the getPendingValue function is used to calculate the value of eth in the withdrawal process.

However, only the amount of eth that has not passed the request during the withdrawal process is calculated here, and the part that has passed the request but has not been claimed is not calculated.

This may cause the rollToNextRound function in the TokenVault contract to call the getAllStrategyPendingValue function to obtain all pending eth values less than expected.

Code Location:

contracts/strategies/STETHHoldingStrategy.sol#L155-157

```
function getPendingValue() public override returns (uint256 value) {
    (, , value) = checkPendingAssets();
}

function checkPendingAssets()
    public
    returns (
        uint256[] memory ids,
        uint256 totalClaimable,
        uint256 totalPending
    )
{
    ...

    for (uint256 i; i < length; i++) {
        ILidoWithdrawalQueue.WithdrawalRequestStatus
            memory status = statuses[i];
        if (status.isClaimed) {
            continue;
        }
        if (status.isFinalized) {
            ids[j++] = allIds[i];
            totalClaimable = totalClaimable + status.amountOfStETH;
        } else {
            totalPending = totalPending + status.amountOfStETH;
        }
    }

    ...
}
```

Solution

It is recommended that the getPendingValue function should calculate the value of all eth in the withdrawal process, including those that have been requested but not claimed. And remove + getClaimableValue in the getAllValue function to avoid double calculation.

Status

Fixed

[N9] [Suggestion] Lack of event records

Category: Others

Content

There is no corresponding event logged when a sensitive parameter in the contract is modified.

Code Location:

contracts/token/Minter.sol#L30-32

```
    function setNewVault(address _vault) external onlyVault {
        vault = payable(_vault);
    }
```

contracts/token/StoneCross.sol#L64-110

```
    function _nonblockingLzReceive(
        uint16 _srcChainId,
        bytes memory _srcAddress,
        uint64 _nonce,
        bytes memory _payload
    ) internal virtual override {
        ...

        if (packetType == PT_SEND) {
            _sendAck(_srcChainId, _srcAddress, _nonce, _payload);
        } else if (packetType == PT_FEED) {
            ...

            tokenPrice = price;
            updatedTime = time;
        } else if (packetType == PT_SET_ENABLE) {
            ...

            enable = flag;
        } else if (packetType == PT_SET_CAP) {
            ...

            cap = _cap;
        } else {
            revert("unknown packet type");
        }
    }
```

contracts/AssetsVault.sol#L35-37

```
    function setNewVault(address _vault) external onlyPermit {
        stoneVault = _vault;
```

```
    }
```

contracts/strategies/RETHHoldingStrategy.sol#L158-164

```
    function setRouter(
        bool _buyOnDex,
        bool _sellOnDex
    ) external onlyGovernance {
        buyOnDex = _buyOnDex;
        sellOnDex = _sellOnDex;
    }
```

contracts/strategies/SFraxETHHoldingStrategy.sol#L151-157

```
    function setRouter(
        bool _buyOnDex,
        bool _sellOnDex
    ) external onlyGovernance {
        buyOnDex = _buyOnDex;
        sellOnDex = _sellOnDex;
    }
```

contracts/strategies/STETHHoldingStrategy.sol#L253-259

```
    function setRouter(
        bool _buyOnDex,
        bool _sellOnDex
    ) external onlyGovernance {
        buyOnDex = _buyOnDex;
        sellOnDex = _sellOnDex;
    }
```

contracts/strategies/Strategy.sol#L89-91

```
    function setBufferTime(uint256 _time) external onlyGovernance {
        bufferTime = _time;
    }
```

contracts/strategies/StrategyController.sol#L322-324

```
    function setNewVault(address _vault) external onlyVault {
        stoneVault = _vault;
```

```
    }
```

contracts/strategies/SwappingAggregator.sol#L396-420

```
    function setUniRouter(
        address _token,
        address _uniPool,
        uint256 _slippage,
        uint24 _fee
    ) external onlyGovernance {
        require(_token != address(0), "ZERO ADDRESS");

        univ3Pools[_token] = _uniPool;
        slippage[_token] = _slippage;
        fees[_token] = _fee;
    }

    function setCurveRouter(
        address _token,
        address _curvePool,
        uint8 _curvePoolType,
        uint256 _slippage
    ) external onlyGovernance {
        require(_token != address(0), "ZERO ADDRESS");

        curvePools[_token] = _curvePool;
        curvePoolType[_curvePool] = _curvePoolType;
        slippage[_token] = _slippage;
    }
```

Solution

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

Status

Acknowledged

[N10] [Suggestion] Authority transfer enhancement

Category: Authority Control Vulnerability Audit

Content

The permission transfer method for the core roles(like proposer and governance) does not adopt the pending and access processes. If set incorrectly, the permission of the core roles will be lost.

Code Location:

contracts/strategies/Strategy.sol#L84-87

```
    function setGovernance(address governance_) external onlyGovernance {
        emit TransferGovernance(governance, governance_);
        governance = governance_;
    }
```

contracts/strategies/SwappingAggregator.sol#L422-426

```
    function setGovernance(address governance_) external onlyGovernance {
        emit TransferGovernance(governance, governance_);
        governance = governance_;
    }
```

contracts/governance/Proposal.sol#L138-142

```
    function setProposer(address _proposer) external onlyProposer {
        emit SetProposer(proposer, _proposer);

        proposer = _proposer;
    }
```

Solution

It is recommended to use the pending and access processes. After the setting is completed, it enters the pending acceptance state, and only the newly set ones can accept the transfer permission.

Status

Acknowledged

[N11] [Low] Lack of CrossChain fee checking in the bridgeTo function

Category: Design Logic Audit

Content

In the DepositBridge contract, there is no check in the bridgeTo function to see if the incoming

_gasPaidForCrossChain parameter is greater than or equal to the handling fee required to send across the chain.

If passed in too small it may cause the cross-chain operation to fail.

Code Location:

contracts/mining/DepositBridge.sol#L30-41

```
    function bridgeTo(
        uint256 _amount,
        bytes calldata _dstAddress,
        uint256 _gasPaidForCrossChain
    ) public payable returns (uint256 stoneMinted) {
        stoneMinted = bridge(
            msg.sender,
            _amount,
            _dstAddress,
            _gasPaidForCrossChain
        );
    }
```

Solution

It is recommended to add a check in the bridgeTo function that the incoming _gasPaidForCrossChain parameter is greater than or equal to the value of the handling fee calculated by the estimateSendFee function.

Status

Acknowledged

[N12] [Suggestion] Missing check for dstChainId on initialisation

Category: Others

Content

In the DepositBridge contract, the dstChainId is set on initialisation, but there is no check to see if the value set is not equal to the chainId of the current chain, which would cause the sendFrom function in the stone tokens to revert and the cross-chain operation to fail.

Code Location:

contracts/mining/DepositBridge.sol#L27

```
    constructor(address _stone, address payable _vault, uint16 _dstChainId) {
        ...
```

```
        dstChainId = _dstChainId;
    }
```

Solution

It is recommended to add a check in the constructor that dstChainId is not equal to the chainId of the current chain.

Status

Acknowledged

[N13] [Low] Lack of scope check

Category: Design Logic Audit

Content

1.In Strategy contracts, the setBufferTime function can be used to set the delay time for strategy operations. However, there is no check on the range of the _time parameter passed in, and if it is too large, the normal operation of the strategy contract will be affected.

Code Location:

contracts/strategies/Strategy.sol#L89-91

```
    function setBufferTime(uint256 _time) external onlyGovernance {
        bufferTime = _time;
    }
```

2.In SwappingAggregator contracts, the Governance role can set the slips corresponding to different tokens and the fees charged by calling the setSlippage function, the setUniRouter function, and the setCurveRouter function.

However, there is no range checking of incoming new slippage and fees at the time of setup, which could result in arbitrage or unintended depletion of the user's funds if set too high.

Code Location:

contracts/strategies/SwappingAggregator.sol#L387-420

```
function setSlippage(
    address _token,
    uint256 _slippage
) external onlyGovernance {
    emit SetSlippage(_token, slippage[_token], _slippage);

    slippage[_token] = _slippage;
}

function setUniRouter(
    address _token,
    address _uniPool,
    uint256 _slippage,
    uint24 _fee
) external onlyGovernance {
    require(_token != address(0), "ZERO ADDRESS");

    univ3Pools[_token] = _uniPool;
    slippage[_token] = _slippage;
    fees[_token] = _fee;
}

function setCurveRouter(
    address _token,
    address _curvePool,
    uint8 _curvePoolType,
    uint256 _slippage
) external onlyGovernance {
    require(_token != address(0), "ZERO ADDRESS");

    curvePools[_token] = _curvePool;
    curvePoolType[_curvePool] = _curvePoolType;
    slippage[_token] = _slippage;
}
```

Solution

It is recommended to add the range checking to the corresponding functions.

Status

Acknowledged

[N14] [Information] Potential governance attacks

Category: Design Logic Audit

Content

In a Proposal contract, users can call the voteFor function to transfer their holdings of stone tokens into that contract and vote on a specified proposal.

However, if the Proposer role is evil (e.g. in the case of lost permissions), it is possible to call the instantWithdraw function or requestWithdraw function in the Stone Vault contract by submitting a proposal and transferring a large number of stone tokens towards the end of the vote to ensure that the proposal passes. After the proposal is executed, it will consume other users' stone tokens and make additional profit (enough to cover the cost of the attack).

Code Location:

contracts/governance/Proposal.sol#L76-96

```
    function voteFor(address _proposal, uint256 _poll, bool _flag) external {
        require(canVote(_proposal), "cannot vote");

        TransferHelper.safeTransferFrom(
            stoneToken,
            msg.sender,
            address(this),
            _poll
        );

        ...
    }
```

Solution

It is recommended to lock the stone tokens transferred by the user with an additional contract, and allow users to withdraw them after the proposal has finished voting.

Status

Acknowledged

[N15] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In SwappingAggregator contracts, the Governance role can set the slips, the exchange router and the fee

corresponding to different tokens by calling the setSlippage function, the setUniRouter function, and the setCurveRouter function. If the privilege is lost or misused, this may have an impact on the user's assets.

Code Location:

contracts/strategies/SwappingAggregator.sol#L387-420

```
    function setSlippage(
        address _token,
        uint256 _slippage
    ) external onlyGovernance {
        emit SetSlippage(_token, slippage[_token], _slippage);

        slippage[_token] = _slippage;
    }

    function setUniRouter(
        address _token,
        address _uniPool,
        uint256 _slippage,
        uint24 _fee
    ) external onlyGovernance {
        require(_token != address(0), "ZERO ADDRESS");

        uniV3Pools[_token] = _uniPool;
        slippage[_token] = _slippage;
        fees[_token] = _fee;
    }

    function setCurveRouter(
        address _token,
        address _curvePool,
        uint8 _curvePoolType,
        uint256 _slippage
    ) external onlyGovernance {
        require(_token != address(0), "ZERO ADDRESS");

        curvePools[_token] = _curvePool;
        curvePoolType[_curvePool] = _curvePoolType;
        slippage[_token] = _slippage;
    }
```

2.In Proposal contracts, the Proposer role can initiate a proposal by calling the propose function. If the privilege is lost or misused, the Proposer role may launch a malicious proposal causing the user to suffer a loss of funds.

Code Location:

contracts/governance/Proposal.sol#L57-74

```
    function propose(bytes calldata _data) external onlyProposer {
        ...
    }
```

Solution

It is recommended that in the early stages of the project, the Governance role and the Proposer role should use multi-signatures to avoid single-point risks. After the project is running stably, the authority of these roles should be handed over to community governance for management, and strict identity authentication should be performed when adding roles.

Status

Fixed; The Governance role's permission of the SwappingAggregator contract has been transferred:

https://etherscan.io/tx/0x073d65049ae9572afcbc6fd9dc5b0329861725f2cc298c65dcacd66e12b18781

The Proposer role's permission in the Proposal contract has been transferred:

https://etherscan.io/tx/0x12000fc2e04ef405d7143fca632597d688317dc6767572de661a649ab6d6faf4

[N16] [Suggestion] Using `block.timestamp` for swap deadline offers no protection

Category: Reordering Vulnerability

Content

In SwappingAggregator contracts, the deadline used when exchanging tokens defaults to block.timestamp. In a Proof-of-Stake (PoS) model, malicious validators can hold back the transaction and execute it at a more favourable block number, as `block.timestamp` will have the value of whichever block the transaction is inserted into. Because of this, a malicious miner/sequencer can hold the transaction and execute it whenever wanted in order to acquire some profit from it.

Code Location:

contracts/strategies/SwappingAggregator.sol#L121-187

```
function swapOnUniV3(
    address _token,
    uint256 _amount,
    bool _isSell
) internal nonReentrant returns (uint256 amount) {
    ...

    if (_isSell) {
        ...

        ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
            .ExactInputSingleParams({
                tokenIn: _token,
                tokenOut: WETH9,
                fee: fees[_token],
                recipient: address(this),
                deadline: block.timestamp,
                amountIn: _amount,
                amountOutMinimum: minReceived,
                sqrtPriceLimitX96: 0
            });

        ...
    } else {
        ...

        ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
            .ExactInputSingleParams({
                tokenIn: WETH9,
                tokenOut: _token,
                fee: fees[_token],
                recipient: address(this),
                deadline: block.timestamp,
                amountIn: _amount,
                amountOutMinimum: minReceived,
                sqrtPriceLimitX96: 0
            });

        ...
    }
}
```

Solution

It is recommended that the function caller be allowed to specify the exchange deadline input parameter.

Status

Acknowledged

[N17] [Suggestion] Missing return value check when adding strategies

Category: Others

Content

In StrategyController contracts, the type of data structure used for strategy storage is the EnumerableSet library from openzeppelin. When using the .add() function, it will return false if the added data already exists and will not add the data repeatedly.

However, the StrategyController contract does not check the return value of the .add() function when adding or setting a strategy, which may result in strategies not being added but ratios being changed.

Code Location:

contracts/strategies/StrategyController.sol

```
    function _initStrategies(
        address[] memory _strategies,
        uint256[] memory _ratios
    ) internal {
        ...

        for (uint i; i < length; i++) {
            strategies.add(_strategies[i]);
            ratios[_strategies[i]] = _ratios[i];
            totalRatio = totalRatio + _ratios[i];
        }
        require(totalRatio <= ONE_HUNDRED_PERCENT, "exceed 100%");
    }

    function _setStrategies(
        address[] memory _strategies,
        uint256[] memory _ratios
    ) internal {
        ...

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

            strategies.add(_strategies[i]);
            ratios[_strategies[i]] = _ratios[i];
```

```
            totalRatio = totalRatio + _ratios[i];
        }
        require(totalRatio <= ONE_HUNDRED_PERCENT, "exceed 100%");
    }
```

Solution

It is recommended to add a check to see if the add function returns false when adding strategies, or use the contains function to do so.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002312180001	SlowMist Security Team	2023.12.07 - 2023.12.18	Low 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 4 high risk, 3 medium risk, 4 low risk, 5 suggestion and 1 info vulnerabilities. 3 high risk, 3 medium risk, 1 low risk, 1 suggestion and 1 info vulnerabilities were fixed. Other vulnerabilities were Acknowledged. Some of these contracts has been deployed to the main network.

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