Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

STAKESTONE

StoneVault-V1

Veridise Inc.
December 28, 2023

► **Prepared For:**

StakeStone
https://stakestone.io/

► **Prepared By:**

Ajinkya Rajput
Jacob Van Geffen

► **Contact Us:** contact@veridise.com

► **Version History:**

Dec. 22, 2023	V1
Dec. 19, 2023	Initial Draft

Contents

1 Executive Summary

From Dec. 8, 2023 to Dec. 15, 2023, StakeStone engaged Veridise to review the security of their StoneVault-V1. The review covered smart contracts that facilitate deposits and transfers for the Stone token, bridging between Stone and ETH, executing various investment strategies, and governance over proposals of new such strategies. Veridise conducted the assessment over 16 person-days, with 2 engineers reviewing code over 8 days on commit `0x8a49bb0`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The StoneVault-V1 developers provided the source code of the StoneVault-V1 contracts for review. The source code contains original logic for round-based withdrawals and a governance scheme based on the "Optimizing Portfolio and Allocation Proposal" (OPAP) mechanism.

To facilitate the Veridise auditors' understanding of the code, the StoneVault-V1 developers provided documentation for their protocol located at `https://docs.stakestone.io/stakestone/`.

The source code contained a test suite, which the Veridise auditors noted adequately tested the functional correctness of the StoneVault-V1 contract.

Summary of issues detected. The audit uncovered 14 issues, 1 of which is assessed to be of high or critical severity by the Veridise auditors. Specifically, issue (V-STN-VUL-001) introduces a reentrancy vulnerability which allows malicious strategies to steal ETH from the protocol. The Veridise auditors also identified several medium-severity issues, including one issue that enables users to bypass round-based withdraw logic (V-STN-VUL-002) and another stemming from missing initialization logic (V-STN-VUL-003), as well as 7 warnings and 2 informational findings. The StoneVault-V1 developers have acknowledged all issues.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve StoneVault-V1. First, we recommend adding detailed messages to `require` statements in order to more easily communicate causes of errors to callers into StoneVault-V1. Additionally, to ensure that the StoneVault-V1 code base remains maintainable, we recommend implementing modular functions and implementing common code in internal functions. Finally, we recommend additional in-line documentation with comments describing the intended behavior of functions across the code base. Several issues in Section 4 provide more detail on these recommendations. We believe that these changes will improve both the readability and maintainability of the StoneVault-V1 code base, leading to quicker development processes and fewer bugs in the future.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
StoneVault-V1	`0x8a49bb0`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Dec. 8 - Dec. 15, 2023	Manual & Tools	2	16 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	1	1
Medium-Severity Issues	2	2
Low-Severity Issues	2	2
Warning-Severity Issues	7	7
Informational-Severity Issues	2	2
TOTAL	14	14

Table 2.4: Category Breakdown.

Name	Number
Maintainability	7
Usability Issue	3
Logic Error	2
Reentrancy	1
Data Validation	1

Audit Goals and Scope 3

3.1 Audit Goals

The engagement was scoped to provide a security assessment of StoneVault-V1's smart contracts. In our audit, we sought to answer questions such as:

- Can an attacker steal ETH from the protocol?
- Does the protocol distribute the rewards to users correctly?
- Does the protocol collect appropriate amount of fees?
- Does the protocol interact correctly with the strategies?
- How do the downstream strategies affect the security of the protocol?
- Is the interaction between the vault and other components correct?
- Is the implementation of voting correct?
- Is the interaction between L1 and L2 correct?
- Is the implementation of pricing of Stone correct?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.

Scope. The scope of this audit is limited to the `contracts` folder of the source code provided by the StoneVault-V1 developers, which contains the smart contract implementation of StoneVault-V1.

During the audit, the Veridise auditors referred to the excluded files but assumed that they have been implemented correctly. Following files were excluded from the audit.

- `contracts/strategies/*`
- `contracts/mock/*`
- All files in`contracts/mining/` except `DepositBridge.sol`

Methodology. Veridise auditors inspected the provided tests, and read the StoneVault-V1 documentation. They then began a manual audit of the code assisted by static analyzers. During the audit, the Veridise auditors regularly met with the StoneVault-V1 developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-STN-VUL-001	A malicious strategy can force invest all ETH i. . .	High	Fixed
V-STN-VUL-002	Round based withdrawal logic can be bypassed	Medium	Intended Behavior
V-STN-VUL-003	withdrawFeeRate not initialised in constructor	Medium	Intended Behavior
V-STN-VUL-004	_ratios can be all zero	Low	Intended Behavior
V-STN-VUL-005	Addresses should not be hardcoded	Low	Fixed
V-STN-VUL-006	Separate functionality for instantWithdraw()	Warning	Acknowledged
V-STN-VUL-007	Better revert messages	Warning	Acknowledged
V-STN-VUL-008	Duplicate Code	Warning	Acknowledged
V-STN-VUL-009	Type address used instead of interfaces	Warning	Acknowledged
V-STN-VUL-010	Use consistent solidity version	Warning	Intended Behavior
V-STN-VUL-011	Use in-fix addition instead of uint256 add func. . .	Warning	Acknowledged
V-STN-VUL-012	Same constants defined in multiple contracts	Warning	Acknowledged
V-STN-VUL-013	Explicit return recommended	Info	Acknowledged
V-STN-VUL-014	Unnecessary ternary statement	Info	Acknowledged

4.1 Detailed Description of Issues

4.1.1 V-STN-VUL-001: A malicious strategy can force invest all ETH in itself

Severity	High	**Commit**	8a49bb0
Type	Reentrancy	**Status**	Fixed
File(s)	StrategyController.sol		
Location(s)	onlyRebaseStrategies()		
Confirmed Fix At	6834828		

The `StoneVault` contract receives deposits from the users and invests them in various strategies via `StrategyController`. There can be multiple strategies active at the same time and the whole available pool of user deposits is diversified across various strategies according to the `ratios`.

These ratios can change overtime and the funds are redistributed across the strategies. This process is called *rebasing* in the protocol. Rebasing can be performed by calling either `rebaseStrategies()` or `onlyRebaseStrategies()` functions in `StrategyController`. The function `rebaseStrategies()` is protected by `onlyVault()` modifier therefore can only be called by `StoneVault`.

However, `onlyRebaseStrategies()` is not protected.

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

Snippet 4.1: `onlyRebaseStrategies()` in `StrategyController.sol`

`onlyRebaseStrategies()` calls an internal function `_rebase()` with zero as arguments.

```
function _rebase(uint256 _in, uint256 _out) internal {
    require(_in == 0 || _out == 0, "only deposit or withdraw");

    if (_in != 0) {
        AssetsVault(assetsVault).withdraw(address(this), _in);
    }
    uint256 total = getAllStrategyValidValue();
    if (total < _out) {
        total = 0;
    } else {
        total = total + _in - _out;
    }

    uint256 length = strategies.length();
    StrategyDiff[] memory diffs = new StrategyDiff[](length);
    uint256 head;
    uint256 tail = length - 1;
    for (uint i; i < length; i++) {
        address strategy = strategies.at(i);
        if (ratios[strategy] == 0) {
            _clearStrategy(strategy, true);
            //@audit paying back to vault right away might not be good idea as the amount might be needed in other strategy
            continue;
        }
        uint256 newPosition = (total * ratios[strategy]) /
            ONE_HUNDRED_PERCENT;
        uint256 position = getStrategyValidValue(strategy);

        if (newPosition < position) {
            diffs[head] = StrategyDiff(
                strategy,
                false,
                position - newPosition
            );
            head++;
        } else if (newPosition > position) {
            diffs[tail] = StrategyDiff(
                strategy,
                true,
                newPosition - position
            );
            if (tail != 0) {
                tail--;
            }
        }
    }

    length = diffs.length;
    for (uint256 i; i < length; i++) {
        StrategyDiff memory diff = diffs[i];

        if (diff.amount == 0) {
            continue;
        }

        if (diff.isDeposit) {
            if (address(this).balance < diff.amount) {
                diff.amount = address(this).balance;
            }
            _depositToStrategy(diff.strategy, diff.amount);
        } else {
            _withdrawFromStrategy(diff.strategy, diff.amount);
        }
    }

    _repayToVault();
}
```

The `_rebase()` function performs following tasks in order

1. Calculate the amount of ETH to be deposited or withdrawn from each strategy and is stored in `StrategyDiff` structure according to ratios
2. Withdraw ETH from all strategies that need withdrawal according to step 1
3. Deposit ETH into all the strategies that need deposits according to step 1

a) NOTE: While depositing if the StrategyController does not have enough ETH, the controller does not revert and just deposits all available balance in the strategies and zero in all other strategies. _rebase() then silently returns.

During the step 3 above, when a deposit is called, the strategy contract or the token contract that have strategy interacts with may have a receive() hook. And as onlyrebaseStrategies() is not protected, a malicious strategy call make a reentrant call to onlyRebaseStrategy(). Therefore, the control flow ill return to Step 3 of rebase and will force the strategy to invest ETH again in the strategy till all available ETH is invested.

The vault allows adding strategies if there is a Proposal for the strategy and it has received enough votes. This makes malicious strategies to be an attack vector. Also, a malicious user can have an upgradable strategy that is benign for time being and then becomes malicious after it has earned trust of the user.

Impact A malicious strategy can force invest all funds of the StrategyController into itself. The severity of attack increases if the attacker can identify any call that updates the ratios and front run the subsequent transaction that performs a rebase.

Recommendation

1. Protect the onlyRebaseStrategies() with a nonRentrant() modifier.
2. Rebase in the same transaction whenever the ratios are updated.

Developer Response The developers acknowledged this issue.

4.1.2 V-STN-VUL-002: Round based withdrawal logic can be bypassed

Severity	Medium	**Commit**	8a49bb0
Type	Logic Error	**Status**	Intended Behavior
File(s)	`StoneVault.sol`		
Location(s)	instantWithdraw()		
Confirmed Fix At			

The `StoneVault` is the contract where users deposit and redeem their stones to withdraw ETH. An user can deposit ETH any time but withdrawals are handled in two step process which depend on rounds. For withdrawals the user has to perform 2 steps

1. Initiate a withdrawal by calling `requestWithdraw()`. Along with other book keeping, this records the roundID in which the withdrawal was requested.
2. Perform the actual withdrawal by calling `instantWithdraw()`. This function takes in two arguments, `_amount` and `_shares` . It adds the `_amount` to number of ETH withdrawn if the request was made in a roundID strictly less than current round.

`instantWithdraw()` also allows an user to redeem shares to get back ETH as shown in the following snippet

Impact The round based withdrawal logic can be bypassed by calling `instantWithdraw()` with non zero `_shares` argument.

Recommendation Remove the share redemption logic from instant withdraw.

Developer Response The instantWithdraw() and requestWithdraw() are two different patterns performed on withdrawals. instantWithdraw() makes users convert their STONEs to Ethers directly and in the meantime the STONEs user held is burned. requestWithdraw() will allow user to make a request on withdrawals to save gas.

```
function requestWithdraw(uint256 _shares) external nonReentrant {
    require(_shares != 0, "too small");
    require(latestRoundID != 0, "should withdraw instantly");
    Stone stoneToken = Stone(stone);
    Minter stoneMinter = Minter(minter);

    require(stoneToken.balanceOf(msg.sender) >= _shares, "exceed balance");

    TransferHelper.safeTransferFrom(
        stone,
        msg.sender,
        address(this),
        _shares
    );

    withdrawingSharesInRound = withdrawingSharesInRound + _shares;

    UserReceipt storage receipt = userReceipts[msg.sender];

    if (receipt.withdrawRound == latestRoundID) {
        receipt.withdrawShares = receipt.withdrawShares + _shares;
    } else if (receipt.withdrawRound == 0) {
        receipt.withdrawShares = _shares;
        receipt.withdrawRound = latestRoundID;
    } else {
        // Withdraw previous round share first
        uint256 withdrawAmount = VaultMath.sharesToAsset(
            receipt.withdrawShares,
            roundPricePerShare[receipt.withdrawRound]
        );

        stoneMinter.burn(address(this), receipt.withdrawShares);
        withdrawingSharesInPast =
            withdrawingSharesInPast -
            receipt.withdrawShares;

        receipt.withdrawShares = _shares;
        receipt.withdrawableAmount =
            receipt.withdrawableAmount +
            withdrawAmount;
        receipt.withdrawRound = latestRoundID;
    }

    emit InitiateWithdraw(msg.sender, _shares, latestRoundID);
}
```

Snippet 4.2: `requestWithdraw()` in `StoneVault`

```
if (_amount != 0) {
  UserReceipt storage receipt = userReceipts[msg.sender];

  if (
    receipt.withdrawRound != latestRoundID &&
    receipt.withdrawRound != 0
  ) {
```

Snippet 4.3: Snippet from `instantWithdraw()` in `StoneVault`

```
if (_shares != 0) {
 uint256 sharePrice;

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

   sharePrice = latestSharePrice < currSharePrice
     ? latestSharePrice
     : currSharePrice;
 }

 uint256 ethAmount = VaultMath.sharesToAsset(_shares, sharePrice);

 stoneMinter.burn(msg.sender, _shares);

 if (ethAmount <= idleAmount) {
   actualWithdrawn = actualWithdrawn + ethAmount;

   emit Withdrawn(msg.sender, ethAmount, latestRoundID);
 } else {
   actualWithdrawn = actualWithdrawn + idleAmount;
   ethAmount = ethAmount - idleAmount;

   StrategyController controller = StrategyController(
     strategyController
   );
   uint256 actualAmount = controller.forceWithdraw(ethAmount);

   actualWithdrawn = actualWithdrawn + actualAmount;

   emit WithdrawnFromStrategy(
     msg.sender,
     ethAmount,
     actualAmount,
     latestRoundID
   );
 }
}
```

Snippet 4.4: Branch for redeeming shares in `instantWithdraw()` in `StoneVault`

4.1.3 V-STN-VUL-003: withdrawFeeRate not initialised in constructor

Severity	Medium	**Commit**	8a49bb0
Type	Logic Error	**Status**	Intended Behavior
File(s)	`StoneVault.sol`		
Location(s)	N/A		
Confirmed Fix At			

`StoneVault` charges a percentage of withdrawal amount as `withdrawalFee`. This percentage is stored in `withdrawFeeRate` state variable. This fee is calculated at the end of `instantWithdraw()` as shown in following code snippet.

```
uint256 withFee;
if (withdrawFeeRate != 0) {
    withFee = (actualWithdrawn * withdrawFeeRate) / ONE_HUNDRED_PERCENT;
    aVault.withdraw(feeRecipient, withFee);

    emit FeeCharged(msg.sender, withFee);
}
aVault.withdraw(msg.sender, actualWithdrawn - withFee);
```

`withdrawFeeRate` is not initialised in `StoneVaults` constructor and has to be set by calling `setWithdrawFeeRate()`.

As the vault does not have functionality to pause, the vault is active as soon as it is instantiated. Therefore, this opens a window between the time when the `StoneVault` is instantiated and `setWithdrawFeeRate()` is not called. The `withdrawFeeRate` will be zero in this window. The window can grow arbitrary large if the deployment scripts miss calling `setWithdrawFeeRate()` .

Impact No fees will be collected until `setWithdrawFeeRate()` is called. This will lead to financial losses to the vault

Recommendation Initialise `withdrawFeeRate` with an initial value in constructor.

Developer Response The initial fee rate should be 0. We will not collect fees on vault at start.

4.1.4 V-STN-VUL-004: _ratios can be all zero

Severity	Low	**Commit**	8a49bb0
Type	Data Validation	**Status**	Intended Behavior
File(s)	`StoneVault.sol`		
Location(s)	constructor()		
Confirmed Fix At			

In `StoneVault` contract the `constructor()` validates and initializes the state variables for addresses of other contracts as well as parameters of the vault.

```
constructor(
    address _minter,
    address _proposal,
    address payable _assetsVault,
    address[] memory _strategies,
    uint256[] memory _ratios
) {
    require(
        _minter != address(0) &&
            _proposal != address(0) &&
            _assetsVault != address(0),
        "ZERO ADDRESS"
    );

    uint256 length = _strategies.length;
    for (uint256 i; i < length; i++) {
        require(_strategies[i] != address(0), "ZERO ADDRESS");
    }

    minter = _minter;
    proposal = _proposal;
    assetsVault = _assetsVault;

    feeRecipient = msg.sender;

    StrategyController controller = new StrategyController(
        _assetsVault,
        _strategies,
        _ratios
    );
```

Snippet 4.5: Snippet from `constructor()` in `StoneVault`

The argument `_ratios` is not validated and passed to `constructor()` of `StrategyController` which in turn calls `_initStrategies()`. This function checks if the sum of all ratios is upper bounded by the constant `ONE_HUNDRED_PERCENT`.

This function does not check if all the ratios are not zero simultaneously.

Impact If all the ratios are zero none of the strategies will be active.

Recommendation Validate if sum of all the ratios is not zero

```
function _initStrategies(
    address[] memory _strategies,
    uint256[] memory _ratios
) internal {
    require(_strategies.length == _ratios.length, "invalid length");

    uint256 totalRatio;
    uint256 length = _strategies.length;
    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%");
}
```

Snippet 4.6: `_initStrategies()` in `StrategyController`

Developer Response It is possible for the vault to set all ratios as zero. And we make a proposal to call `updatePortfolioConfig` to allocate the assets to different strategies.

4.1.5 V-STN-VUL-005: Addresses should not be hardcoded

Severity	Low	**Commit**	8a49bb0
Type	Usability Issue	**Status**	Fixed
File(s)	Multiple Strategies		
Location(s)	N/A		
Confirmed Fix At	5626240		

Various strategy contracts within `contracts/strategies` contain hard-coded addresses for tokens and other external contracts.

```
address public immutable STETH = 0xae7ab96520DE3A18E5e111B5EaAb095312D7fE84;
address public immutable WSTETH =
    0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0;
address public immutable VAULT = 0xBA12222222228d8Ba445958a75a0704d566BF2C8;
address public immutable LP_TOKEN =
    0x42ED016F826165C2e5976fe5bC3df540C5aD0Af7;
address public immutable BOOSTER =
    0xA57b8d98dAE62B26Ec3bcC4a365338157060B234;
address public immutable AURA_REWARD_POOL =
    0x032B676d5D55e8ECbAe88ebEE0AA10fB5f72F6CB;
address public immutable BAL_TOKEN =
    0xba100000625a3754423978a60c9317c58a424e3D;
address public immutable AURA_TOKEN =
    0xC0c293ce456fF0ED870ADd98a0828Dd4d2903DBF;
```

Snippet 4.7: Hard-coded addresses within `BalancerLPAuraStrategy.sol`

```
address public immutable WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
address public immutable RETH = 0xae78736Cd615f374D3085123A210448E74Fc6393;
address public immutable VAULT = 0xBA12222222228d8Ba445958a75a0704d566BF2C8;
address public immutable LP_TOKEN =
    0x1E19CF2D73a72Ef1332C882F20534B6519Be0276;

address public immutable AURA_REWARD_POOL =
    0xDd1fE5AD401D4777cE89959b7fa587e569Bf125D;
address public immutable BAL_TOKEN =
    0xba100000625a3754423978a60c9317c58a424e3D;
address public immutable AURA_TOKEN =
    0xC0c293ce456fF0ED870ADd98a0828Dd4d2903DBF;
address public immutable EXTRA_REWARD =
    0xf66a72886749c96b18526E8E124cC2e18b7c72D2;
```

Snippet 4.8: Hard-coded addresses within `RETHBalancerAuraStrategy.sol`

Instead of using hard-coded addresses, these addresses should be initialized during deployment.

Impact There are two downsides to hard-coding addresses in this way, rather than initializing addresses during construction:

1. Hard-coding addresses makes upgrading the corresponding token impossible. If new versions of the corresponding contracts are published, these new versions cannot be used

by stone vault.

2. Testing stone vault locally is difficult when addresses are hard coded, particularly when testing on mock blockchains.

Recommendation Initialize addresses during deployment instead of hard-coding them.

Developer Response Will fix.

4.1.6 V-STN-VUL-006: Separate functionality for instantWithdraw()

Severity	Warning	**Commit**	8a49bb0
Type	Usability Issue	**Status**	Acknowledged
File(s)	StoneVault.sol		
Location(s)	instantWithdraw()		
Confirmed Fix At			

The instantWithdraw() function takes two parameters as input: _amount, which specifies the amount of ETH to withdraw, and _shares, which specifies the number of shares to withdraw. Based on these inputs, instantWithdraw() then withdraws the combined amount of ETH from _amount and ETH equivalent to _shares through two different paths in the function.

```
function instantWithdraw(
    uint256 _amount,
    uint256 _shares
) external nonReentrant returns (uint256 actualWithdrawn) {
    ...

    if (_amount != 0) {
        ...
    }

    if (_shares != 0) {
        ...
    }

    ...
}
```

In order to more clearly delineate these two separate functionalities, we recommend separating instantWithdraw() into two external functions as demonstrated by the following pseudocode:

```
function completeWithdraw(
    uint256 _amount,
) external nonReentrant returns (uint256 actualWithdrawn) {
    require(_amount != 0);
    ...
}

function instantRedeem(
    uint256 _shares,
) external nonReentrant returns (uint256 actualWithdrawn) {
    require(_shares != 0);
    ...
}
```

Impact Since instantWithdraw performs varying functionality for withdrawing shares versus ETH directly, users may make erroneous assumptions on the affects of instantWithdraw. For example, if a user attempts to withdraw through _amount and _shares at the same time, but has not requested a withdraw previously, instantWithdraw() will revert even when the _shares are withdrawable (since receipt.withdrawableAmount will be 0).

Recommendation Separate `instantWithdraw` into two functions: `completeWithdraw` which takes an amount of ETH as input and `instantRedeem` which takes a number of shares as input.

4.1.7 V-STN-VUL-007: Better revert messages

Severity	Warning	**Commit**	8a49bb0
Type	Maintainability	**Status**	Acknowledged
File(s)	`StoneVault.sol`		
Location(s)	N/A		
Confirmed Fix At			

Various error messages in `require` statements lack necessary details to help users understand the cause for the revert. A few examples are as follows, with corresponding suggestions for more clear messages.

`StoneVault.sol` Line 105-110

```
require(
    _minter != address(0) &&
    _proposal != address(0) &&
    _assetsVault != address(0),
    "ZERO ADDRESS"
);
```

Change message to "Addresses for minter, proposal, and assetsVault must be non-zero"

`StoneVault.sol` Line 181

```
require(_shares != 0, "too small");
```

Change message to "Must withdraw a non-zero number of shares"

`StoneVault.sol` Line 333

```
require(aVault.getBalance() >= actualWithdrawn, "still need wait");
```

Change message to "StoneVault has insufficient balance to process the withdrawal"

Note that there are more occurrences outside of these cases (and outside of `StoneVault.sol`) where the reverting error message needs more detail.

Impact Callers into StoneVault may have trouble determining the cause of reverted transactions.

Recommendation Include more detailed error messages in revert statements.

Developer Response The developers acknowledged this issue.

4.1.8 V-STN-VUL-008: Duplicate Code

Severity	Warning	**Commit**	8a49bb0
Type	Maintainability	**Status**	Acknowledged
File(s)	`StoneVault.sol`		
Location(s)	N/A		
Confirmed Fix At			

In both `instantWithdraw` and `requestWithdraw`, the existing withdraw receipt of the requesting user is checked to determine if the receipt values should be updated (for example, in the case of a previous withdraw request that has not been satisfied).

```
// Withdraw previous round share first
uint256 withdrawAmount = VaultMath.sharesToAsset(
    receipt.withdrawShares,
    roundPricePerShare[receipt.withdrawRound]
);

stoneMinter.burn(address(this), receipt.withdrawShares);
withdrawingSharesInPast =
    withdrawingSharesInPast -
    receipt.withdrawShares;

receipt.withdrawShares = _shares;
receipt.withdrawableAmount =
    receipt.withdrawableAmount +
    withdrawAmount;
receipt.withdrawRound = latestRoundID;
```

Snippet 4.9: Logic performed in `requestWithdraw`, nearly identical to that performed in `instantWithdraw`

Since this logic is nearly identical in the two use cases, it should be moved into a separate internal function.

Impact If part of this logic to update user receipts changes in the future, it is possible that developers may erroneously change the logic in only one of these two locations.

Recommendation Create an internal function `updateWithdrawReceipt` that performs the logic of updated the user withdraw receipt. Then, call `updateWithdrawReceipt` from within both `instantWithdraw` and `requestWithdraw`.

Developer Response The developers acknowledged this issue.

4.1.9 V-STN-VUL-009: Type address used instead of interfaces

Severity	Warning	**Commit**	8a49bb0
Type	Maintainability	**Status**	Acknowledged
File(s)	Multiple		
Location(s)	N/A		
Confirmed Fix At			

The following contracts encode the logic of the protocol.

- `StoneVault`
- `StrategyController`
- `AssetsVault`
- `Minter`
- `Stone`

The address of these contracts is stored in each contract and is casted whenever an external call is made to these contracts.

```
StrategyController controller = StrategyController(
    strategyController
);
```

Snippet 4.10: Code snippet from `instantWithdraw()` in `StoneVault` The address `strategyController` is cast to contract `StrategyController`

Impact This pattern is error prone and type unsafe.

Recommendation Define interfaces for these contract and initialise have state variables with type of these interfaces.

Developer Response The developers acknowledged this issue.

4.1.10 V-STN-VUL-010: Use consistent solidity version

Severity	Warning	**Commit**	8a49bb0
Type	Usability Issue	**Status**	Intended Behavior
File(s)	`StoneCross.sol`		
Location(s)	N/A		
Confirmed Fix At			

All contracts except `StoneCross.sol` use solidity version `0.8.21`, but `StoneCross.sol` uses version `0.8.19`.

```
pragma solidity 0.8.19;
```

Snippet 4.11: Pragma statement in `StoneCross.sol`

`StoneCross.sol` does not require `0.8.19` specifically, so version `0.8.21` should be used to keep consistent with the rest of stone vault.

Impact Using inconsistent solidity versions unnecessarily complicates deployment and testing.

Recommendation Use solidity version `0.8.21` in `StoneCross.sol`.

Developer Response The developers acknowledged this issue.

4.1.11 V-STN-VUL-011: Use in-fix addition instead of uint256 add function

Severity	Warning	**Commit**	8a49bb0
Type	Maintainability	**Status**	Acknowledged
File(s)	`Proposal.sol`		
Location(s)	N/A		
Confirmed Fix At			

The functions `voteFor` and `retreiveAllToken` in `Proposal.sol` use the `uint256` function `add` instead of using the built-in solidity addtion.

```
function voteFor(address _proposal, uint256 _poll, bool _flag) external {
    ...

    if (_flag) {
        detail.support = detail.support.add(_poll);
    } else {
        detail.oppose = detail.oppose.add(_poll);
    }

    polls[msg.sender][_proposal] = polls[msg.sender][_proposal].add(_poll);

    ...
}
```

Snippet 4.12: The `add` function used in `voteFor`

```
function retrieveAllToken() external {
    ...
    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.add(voteAmount);

            ...
        }
    }
    ...
}
```

Snippet 4.13: The `add` function used in `retreiveAllToken`

While this could prevent overflow bugs in previous versions of solidity, version `0.8.x` has built-in overflow protection. For this reason, we recommend using built-in addition to improve code readability.

Impact Unnecessary use of `uint256` `add` reduces readability in `Proposal.sol`, and may incur a slightly higher gas cost.

Recommendation Using built-in addition with `+` and `+=` to update values in `voteFor` and `retreiveAllToken`.

Developer Response The developers acknowledged this issue.

4.1.12 V-STN-VUL-012: Same constants defined in multiple contracts

Severity	Warning	**Commit**	8a49bb0
Type	Maintainability	**Status**	Acknowledged
File(s)	Multiple		
Location(s)	N/A		
Confirmed Fix At			

The protocol defines multiple constants viz.

- `MULTIPLIER`
- `ONE_HUNDRED_PERCENT`
- `minVotePeriod`
- `DAY_INTERVAL`
- `MINIMUM_REBASE_INTERVAL`

These constants have same values but are defined in multiple contracts. This pattern is error prone.

Impact Any future change in the value of these constants will have to be be replicated in all the places where these constants are defined. Missing such a change might various open attack vectors.

Recommendation Define these contracts in a `Configuration` contract and inherit all the contracts which use any of these constants from `Configuration` contract

Developer Response The developers acknowledged this issue. But informed us that they would not update already deployed smart contracts

4.1.13 V-STN-VUL-013: Explicit return recommended

Severity	Info	**Commit**	8a49bb0
Type	Maintainability	**Status**	Acknowledged
File(s)	`StoneCross.sol`		
Location(s)	getQuota()		
Confirmed Fix At			

The `getQuota` function in `StoneCross.sol` uses the default return value when the `if` statement condition is not satisfied

```
function getQuota() external returns (uint256) {
    uint256 amount = quota[block.timestamp / DAY_INTERVAL];
    if (cap > amount && enable) {
        return cap - amount;
    }
}
```

Snippet 4.14: `getQuota` function within `StoneCross.sol`

To make the semantics of `getQuota` more clear for developers, we recommend an explicit return statement.

Impact Implicit return statements lead to code that is more difficult for developers to maintain.

Recommendation Explicitly return 0 at the end of `getQuota`.

Developer Response The developers acknowledged this issue.

4.1.14 V-STN-VUL-014: Unnecessary ternary statement

Severity	Info	**Commit**	8a49bb0
Type	Maintainability	**Status**	Acknowledged
File(s)	`Proposal.sol`		
Location(s)	canVote()		
Confirmed Fix At			

The `canVote` function in `Proposal.sol` uses a ternary statement to return a boolean value. However, this is unnecessary, as the ternary expression evaluates to the same value as the condition.

```
function canVote(address _proposal) public view returns (bool result) {
    if (!proposals.contains(_proposal)) {
        return false;
    }
    ProposalDetail memory detail = proposalDetails[_proposal];
    return block.timestamp < detail.deadline ? true : false;
}
```

Snippet 4.15: `canVote` function with unnecessary ternary statement

Impact Unnecessary ternary statements make the code less easily readable.

Recommendation Change the return statement to the following:

```
    return block.timestamp < detail.deadline;
```

Developer Response The developers acknowledged this issue.