Veridise Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Ribbon Finance: Earn Vault

Veridise Inc.

August 19, 2022

► **Prepared For:**

Ken Chan | Ribbon Finance
https://www.ribbon.finance/

► **Prepared By:**

Kostas Ferles
Benjamin Mariano
Xiangan He
Muhammad Khattak

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

► **Version History:**

August 19, 2022 V1

Contents

1 Executive Summary

From July 25 to August 15, Ribbon Finance engaged Veridise to review the security of their Earn Vault. The review covered the on-chain contracts that govern the logic of the vault. Veridise conducted the assessment over 6 person-weeks, with 3 engineers reviewing code over 2 weeks from commit `d166f84` to `eec7cc3` of the `ribbon-finance/ribbon-v2-earn` repository. The auditing strategy involved tool-assisted analysis of the source code performed by Veridise engineers. The tools that were used in the audit included a mix of static analyzers.

Summary of issues detected. The audit uncovered 17 issues, 11 of which were acknowledged and fixed by the developers. Most notably, the audit discovered a medium severity issue where the vault charges the wrong management fee to its users (V-RBE-VUL-002). Additionally, we discovered several low severity issues, including a data validation issue (V-RBE-VUL-005) and a state update after an interaction with external code (V-RBE-VUL-003). Finally, the audit identified several gas optimizations, including unnecessary state loads (V-RBE-VUL-010, V-RBE-VUL-011, V-RBE-VUL-012) and unnecessary arguments for an internal function (V-RBE-VUL-013).

Code assessment. The Earn Vault implements a multi-round protocol where users can deposits assets that can later be exchanged for interest accruing tokens. Each round has a fixed configurable duration and begins by allocating a percentage of its holdings to issue loans and the rest for buying options. At the beginning of a round, the vault sends all assets allocated for loans to a set of trusted borrowers. All loans are expected to be repaid, including interest, by the end of a round, whereas the decision for buying options is made entirely by a vault administrator. At the end of each round the vault calculates the new price of each token based on its current balance and any pending withdrawals. Users of the vault are free to choose when they can exchange their deposits for vault tokens. Finally, the vault provides several other features to users including the ability to pause their positions and stake their tokens to a liquidity gauge.

The artifacts provided by Ribbon Finance include a set of Solidity contracts as well as deployment and test suite written in Typescript. The test suite is extensive and includes several unit tests and has high code coverage. Furthermore, the client provided detailed documentation describing the intended behavior of the system.

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
Earn Vault	`d166f84` - `eec7cc3`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
July 29 - August 15, 2022	Manual & Tools	3	6 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Medium-Severity Issues	2	1
Low-Severity Issues	7	2
Warning-Severity Issues	5	5
Informational-Severity Issues	3	3
TOTAL	17	11

Table 2.4: Category Breakdown.

Name	Number
Logic Error	4
Gas Optimization	4
Data Validation	3
Usability Issue	2
Reentrancy	1
Dead Code	1
Deployment	1
Maintainability	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of the main logic of the Earn Vault. In our audit, we sought to answer the following questions:

- Can user funds be lost or locked?
- Is the vault guaranteed to handle every valid withdrawal successfully?
- Is it possible for a user to manipulate the vault's share price?
- Does the vault calculate share price correctly at each round?
- Does the vault state's get updated properly at each round?
- Does the pauser vault properly pause/resume a user's options?
- Can one malicious user prevent any other user's complete withdrawal?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis. In particular, we conducted our audit with the aid of our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.

Scope. This audit is restricted to the Solidity smart contracts of the Earn Vault. As such, Veridise engineers first reviewed the provided documentation to understand the desired behavior of the protocol as a whole. They then inspected the provided tests to understand the desired behavior of the protocol's contracts as well as how users are expected to interact with them. Afterward, the Earn Vault contracts were assessed for bugs and security issues.

In terms of the audit, the key components include the following:

- The logic for depositing assets to the vault.
- The logic for redeeming shares from current deposits.
- Implementation of function `rollToNextRound`.
- Implementation of `RibbonVaultPauser`.
- Calculation of management fees and token price per round.

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

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 this case, we judge the likelihood of a vulnerability as follows:

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

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., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-RBE-VUL-001	Closing price calculation can return zero	Medium	Expected Behavior
V-RBE-VUL-002	getVaultFees assumes rollover happens weekly	Medium	Fixed
V-RBE-VUL-003	State update after interaction call	Low	Fixed
V-RBE-VUL-004	No zero address check on setters	Low	Expected Behavior
V-RBE-VUL-005	commitOptionSeller: no zero address check	Low	Fixed
V-RBE-VUL-006	setOptionSeller blocks commitOptionSeller	Low	Expected Behavior
V-RBE-VUL-007	Deposits can go below minimum supply	Low	Expected Behavior
V-RBE-VUL-008	No check for _vaultParams.decimals	Low	Expected Behavior
V-RBE-VUL-009	Remove dead code	Low	Expected Behavior
V-RBE-VUL-010	Unnecessary map load in _initiateWithdraw	Warning	Fixed
V-RBE-VUL-011	Unnecessary map load in _completeWithdraw	Warning	Fixed
V-RBE-VUL-012	Unnecessary map load in pausePosition	Warning	Fixed
V-RBE-VUL-013	Unnecessary arguments for _rollToNextRaund	Warning	Fixed
V-RBE-VUL-014	Contracts post-fixed exceed size limit	Warning	Fixed
V-RBE-VUL-015	Consider allowing amount specification	Info	Fixed
V-RBE-VUL-016	Two different definitions of total percentage	Info	Fixed
V-RBE-VUL-017	Allow more than 0% of funds in vault at initialization	Info	Fixed

Note, for the statuses listed above, we define them as follows:

Fixed	Developers acknowledged the issue and added the fix suggested by Veridise auditors.
Addressed	Developers acknowledged the issue and added a different fix than the fix suggested by Veridise auditors. Veridise auditors verified that the fix performed as the customer expected.
Acknowledged	Developers acknowledged the issue but indicated that it should be addressed (or already is) by code outside of the scope of the current audit. If necessary, developers have forwarded the issue to the team/s in charge of the relevant code.
Open	Developers acknowledged the issue but have not yet addressed it.
Expected Behavior	Developers say the issue identified by Veridise auditors is actually the intended behavior of the code.

4.1 Trusted Entities

During the course of the audit, Veridise auditors confirmed with the Ribbon developers that the following entities are considered "trusted" to some degree:

1. `owner`: Owner of the contract who can set the hyperparameters of the protocol, including fees, deposit cap, loan/option allocations, loan length, option purchase frequency, and

addresses of the vault pauser and liquidity gauge. The owner is also able to recover tokens errantly sent to the vault and can assign critical trusted roles (i.e., the other roles listed here). Ribbon developers say this account will be controlled by a multisig.
2. `keeper`: The keeper is trusted to buy options from the `optionSeller` and to initiate the roll over to new rounds.
3. `optionSeller`: The option seller is trusted to pay back option yields on options when necessary.
4. `borrowers`: Borrowers are trusted to repay their loans and any interest accrued.
5. `feeRecipient`: The fee recipient is trusted to hold the fees collected by the vault.

Due to the trusted nature of these entities, we only report violations if a trusted entity can alter the protocol beyond the intention of the developers.

4.2 Detailed Description of Bugs

In this section, we describe each uncovered vulnerability in more detail.

4.2.1 V-RBE-VUL-001: Closing price calculation can return zero

Severity	Medium	**Commit**	d166f84
Type	Logic Error	**Status**	Expected Behavior
Files	VaultLifecycleEarn.sol		
Functions	rollover		

Description The calculation of `newPricePerShare` in function `VaultLifecycleEarn.rollover` can return zero. This will cause `RibbonEarnVault.rollToNextRound` to revert at the calculation of `mintShares` in `rollover` since `newPricePerShare` is the denominator of this calculation (see code snippets). This scenario is only possible if all users withdraw all of their funds from the vault.

```
    newPricePerShare = ShareMath.pricePerShare(
                    params.currentShareSupply - lastQueuedWithdrawShares,
                    currentBalance - params.lastQueuedWithdrawAmount,
                    pendingAmount,
                    params.decimals
                );
    ...

    mintShares = ShareMath.assetToShares(
                    pendingAmount,
                    newPricePerShare,
                    params.decimals
                );

```

Snippet 4.1: Price and mint shares calculations.

Impact This can prevent the protocol from moving to the next round unless Ribbon directly sends additional funds in the underlying asset to the vault.

Recommendation You can ensure that a minimum supply is properly enforced in such a way that `ShareMath.pricePerShare` always returns a positive number (see issue: "Deposits can go below minimum supply").

Developer Response Since this happens only if all users withdraw their funds, it is acceptable for `rollover` to revert.

Additionally, while our audit was in progress, the developers added a feature which allows the vault to maintain a minimum percentage of deposits at all time. Therefore, this issues can be avoided by configuring the vault accordingly.

4.2.2 V-RBE-VUL-002: getVaultFees assumes rollover happens weekly

Severity	Medium	**Commit**	d166f84
Type	Logic Error	**Status**	Fixed
Files	VaultLifecycleEarn.sol		
Functions	getVaultFees		

Description Function `VaultLifecycleEarn.getVaultFees` calculates the round's `_managementFeeInAsset` by simply multiplying by `managementFeePercent` . However, `managementFeePercent` is adjusted to a weekly basis in the construction of the vault.

```
    _managementFeeInAsset = managementFeePercent > 0
                    ? (lockedBalanceSansPending * managementFeePercent) /
                        (100 * Vault.FEE_MULTIPLIER)
                    : 0;

```

Snippet 4.2: Management Fee Calculation.

Impact If `currentLoanTermLength` is different than a week, then the vault can under- or over-charge users for the current round.

Recommendation Consider adjusting the management fee to take into consideration the amount of weeks in the current loan term.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.3 V-RBE-VUL-003: State update after interaction call

Severity	Low	**Commit**	d166f84
Type	Reentrancy	**Status**	Fixed
Files	RibbonEarnVault.sol		
Functions	completeWithdraw, _completeWithdraw		

Description Function `RibbonEarnVault.completeWithdraw` contains an update to state variable `lastQueuedWithdrawAmount` after a call to `_completeWithdraw()`, which might reach a reentrancy point.

Such patterns are discouraged regardless of the use of reentrancy guards.

Recommendation Even though the current version of the protocol has appropriate reentrancy guards, we would suggest moving this update before the call to `_completeWithdraw()`.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.4 V-RBE-VUL-004: No zero address check on setters

Severity	Low	Commit	d166f84
Type	Data Validation	Status	Expected behavior
Files	RibbonEarnVault.sol		
Functions	setVaultPauser & setLiquidityGauge		

Description Functions `setVaultPauser` and `setLiquidityGauge` in `RibbonEarnVault` do not check for the zero address when setting the pauser and liquidity gauge respectively.

Recommendation Consider adding appropriate checks in both functions.

Developer Response There are adequate checks when both fields are being used. Therefore, the logic is immune to invalid field initialization.

4.2.5 V-RBE-VUL-005: commitOptionSeller: no zero address check

Severity	Low	**Commit**	d166f84
Type	Data Validation	**Status**	Fixed
Files	RibbonEarnVault.sol		
Functions	commitOptionSeller		

Description & Recommendation Function `RibbonEarnVault.commitOptionSeller` can be called in a state where `pendingOptionSeller` is the zero address. Please consider adding a require statement to avoid this scenario.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.6 V-RBE-VUL-006: setOptionSeller blocks commitOptionSeller for 3 days

Severity	Low	Commit	d166f84
Type	Usability Issue	Status	Expected Behavior
Files	RibbonEarnVault.sol		
Functions	setOptionSeller & commitOptionSeller		

Description As shown in the figure below, function `setOptionSeller` updates storage variable `lastOptionSellerChange` to the current timestamp.

```
1    function setOptionSeller(address newOptionSeller) external onlyOwner {
2        require(newOptionSeller != address(0), "R9");
3        emit OptionSellerSet(optionSeller, newOptionSeller);
4        pendingOptionSeller = newOptionSeller;
5        lastOptionSellerChange = block.timestamp;
6    }
```

Snippet 4.3: Function setOptionSeller

Furthermore, function`commitOptionSeller` has the following require statement:

```
1    require(block.timestamp >= (lastOptionSellerChange + 3 days), "R10");
```

Snippet 4.4: Require Statement in `commitOptionSeller`.

As a consequence, `commitOptionSeller` can only run after 3 days from the execution of `setOptionSeller` has passed.

Recommendation We believe the intention here was for the `lastOptionSellerChange = block.timestamp` statement to be inside `commitOptionSeller`, and no in `setOptionSeller`. Please consider moving this statement.

Developer Response This is the expected behavior and it is meant to protect against scenarios where the owner of the contract is compromised.

4.2.7 V-RBE-VUL-007: Deposits can go below minimum supply

Severity	Low	**Commit**	d166f84
Type	Logic Error	**Status**	Expected behavior
Files	RibbonEarnVault.sol		
Functions	initiateWithdraw & withdrawInstantly		

Description Checks for enforcing the minimum supply of the vault only happens in function deposit (see code snippet). If maintaining a minimum supply of funds is an invariant of the system, then a check must be added in relevant withdraw functions (`initiateWithdraw`, `withdrawInstantly`).

```
1    ...
2    uint256 totalWithDepositedAmount = totalBalance() + amount;
3    require(totalWithDepositedAmount <= vaultParams.cap, "R22");
4    require(totalWithDepositedAmount >= vaultParams.minimumSupply, "R23");
```

Snippet 4.5: Relevant `Require` statements

Developer Response This is the intended behavior as the protocol should not prevent users from withdrawing their funds.

Additionally, the changes introduced after the start of our audit allow the vault to maintain a minimum amount of funds at all time, which can be used to ensure the above property.

4.2.8 V-RBE-VUL-008: No check for _vaultParams.decimals in verifyInitializerParams

Severity	Low	Commit	d166f84
Type	Data Validation	Status	Expected behavior
Files	VaultLifecycleEarn.sol		
Functions	verifyInitializerParams		

Description Function `verifyInitializerParams` in library contract `VaultLifecycleEarn` contains a check for every field of input parameter `_vaultParams` except field `decimal`.

Recommendation Consider adding a check that decimals is a non-negative number.

Developer Response Even though a value of zero for decimals is not common, it does not affect the correctness of the overall protocol.

4.2.9 V-RBE-VUL-009: Remove dead code

Severity	Low	**Commit**	d166f84
Type	Dead Code	**Status**	Expected behavior
Files	RibbonVaultPauser.sol		
Functions	resumePosition		

Description `RibbonVaultPauser` refers in several places to `STETH` and `STETH_VAULT` (see code snippets). However, `RibbonEarnVault` has no way of interacting with `STETH` .

```
    address public immutable STETH;
    address public immutable STETH_VAULT;
    ...
    constructor(
        address _keeper,
        address _weth,
        address _steth,
        address _steth_vault) {
        ...
        require(_steth != address(0), "!_steth");
        require(_steth_vault != address(0), "!_steth_vault");
        ...
        STETH = _steth;
        STETH_VAULT = _steth_vault;
    }
    ...
    if (_vaultAddress == STETH_VAULT) {
        totalWithdrawAmount = totalWithdrawAmount - 3;
        emit Resume(msg.sender, _vaultAddress, totalWithdrawAmount - 1);
        IERC20(STETH).safeApprove(_vaultAddress, totalWithdrawAmount);
        currentVault.depositYieldTokenFor(totalWithdrawAmount, msg.sender);
    }
```

Snippet 4.6: `STETH` related code snippets.

Recommendation Consider removing this logic to reduce contract size and attack surface.

Developer Response The `RibbonVaultPauser` might interact with other vaults that are outside the scope of this audit.

4.2.10 V-RBE-VUL-010: Unnecessary map loads in _initiateWithdraw

Severity	Warning	Commit	d166f84
Type	Gas Optimization	Status	Fixed
Files	RibbonEarnVault.sol		
Functions	_initiateWithdraw		

Description The following updates in `_initiateWithdraw` can happen through the storage local variable `withdrawal`. This way you can avoid loading `withdrawals[msg.sender]` multiple times.

```
    ...
    } else {
        require(existingShares == 0, "R25");
        withdrawalShares = numShares;
        withdrawals[msg.sender].round = uint16(currentRound); // update 1
    }
    ...
    withdrawals[msg.sender].shares = uint128(withdrawalShares); // update 2
```

Snippet 4.7: State Updates.

Recommendation Consider performing the updates through local variable `withdrawal`.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.11 V-RBE-VUL-011: Unnecessary map load in _completeWithdraw

Severity	Warning	**Commit**	d166f84
Type	Gas Optimization	**Status**	Fixed
Files	RibbonEarnVault.sol		
Functions	_completeWithdraw		

Description The following update in `_completeWithdraw` can happen through the storage local variable `withdrawal` . This way you can avoid loading `withdrawals[msg.sender]` multiple times.

```
1    withdrawals[msg.sender].shares = 0;
```

Snippet 4.8: Relevant state update.

Recommendation Consider performing the update through local variable `withdrawal`.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.12 V-RBE-VUL-012: Unnecessary map load in pausePosition

Severity	Warning	**Commit**	d166f84
Type	Gas Optimization	**Status**	Fixed
Files	RibbonVaultPauser.sol		
Functions	pausePosition		

Description The following update in `pausePosition` can happen through the storage local variable `pausedPosition`. This way you can avoid loading `pausedPositions[currentVaultAddress][_account]` multiple times.

```
    pausedPositions[currentVaultAddress][_account] = PauseReceipt({
            round: round,
            shares: uint128(_amount)
    });
```

Snippet 4.9: Relevant State Update.

Recommendation Consider performing the update through local variable `pausedPosition`.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.13 V-RBE-VUL-013: Unnecessary arguments for function _rollToNextRound

Severity	Warning	**Commit**	d166f84
Type	Gas Optimization	**Status**	Fixed
Files	RibbonEarnVault.sol		
Functions	_rollToNextRound		

Description Function `_rollToNextRound` is always called with state variables `lastQueuedWithdrawAmount` and `currentQueuedWithdrawShares`. Both variables can be directly accessed within `_rollToNextRound`.

```
1    function _rollToNextRound(uint256 lastQueuedWithdrawAmount,
2                              uint256 currentQueuedWithdrawShares)
```

Snippet 4.10: Function signature of `_rollToNextRound`.

Recommendation Consider removing both arguments from the function's signature.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.14 V-RBE-VUL-014: Contracts post-fix exceed size limit

Severity	Warning	**Commit**	fb0ed31
Type	Deployment.	**Status**	Fixed
Files	RibbonEarnVault.sol		
Functions	-		

Description Contract fails to deploy due to contracts exceeding contract size limit. When compiling, Solidity emits the following warning:

```
Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious
    Dragon). This contract may not be deployable on mainnet. Consider enabling the
    optimizer (with a low "runs" value!
```

Snippet 4.11: Solidity warning at compile-time

Impact Contract will not be able to deploy on mainnet.

Recommendation Consider reducing contract size via implementing some of the recommendations listed in this report.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.15 V-RBE-VUL-015: Consider allowing amount specification for recoverTokens

Severity	Informational	**Commit**	d166f84
Type	Usability	**Status**	Fixed
Files	RibbonEarnVault.sol		
Functions	recoverTokens		

Description & Recommendation Consider adding an argument for allowing the owner to specify an amount to be returned of the asset, in case the need arises to send two different users varying amounts of a token.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.16 V-RBE-VUL-016: Two different definitions of total percentage

Severity	Informational	Commit	eec7cc3
Type	Maintainability	Status	Fixed
Files	RibbonEarnVault.sol, VaultEarnCycle.sol		
Functions	-		

Description `TOTAL_PCT`, defined in `RibbonEarnVault`, and `totalPCT`, defined in `VaultEarnCycle`, both represent the total percentage for loan and option allocation amounts and are initialized to the same constant (currently `10000`). Using two different variables to represent the same value could lead to issues in the future if one is changed and the other is not.

Recommendation Remove the `totalPCT` variable from `VaultEarnCycle` and just use `TOTAL_PCT` from `RibbonEarnVault`.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.

4.2.17 V-RBE-VUL-017: Allow more than 0% of funds in vault at initialization

Severity	Informational	Commit	eec7cc3
Type	Logic Error	Fixed	Fixed
Files	VaultLifecycleEarn.sol		
Functions	verifyInitializerParams		

Description Currently, when the owner sets the allocation and option percentage, they are able to set the percentages such that some of the funds remain in the vault. However, at initialization, this is not allowed due to the following check in `VaultEarnCycle` :

```
    require(
        uint256(_allocationState.loanAllocationPCT) +
            _allocationState.optionAllocationPCT ==
            totalPCT,
            "R50"
    );
```

Snippet 4.12: Relevant require statement.

Recommendation Change == to <= to allow the initial amount left in the vault to be greater than 0%.

Developer Response Fixed in commit 0a4041f4eee773e0bb91d1ed63c03f3a74b74e32.