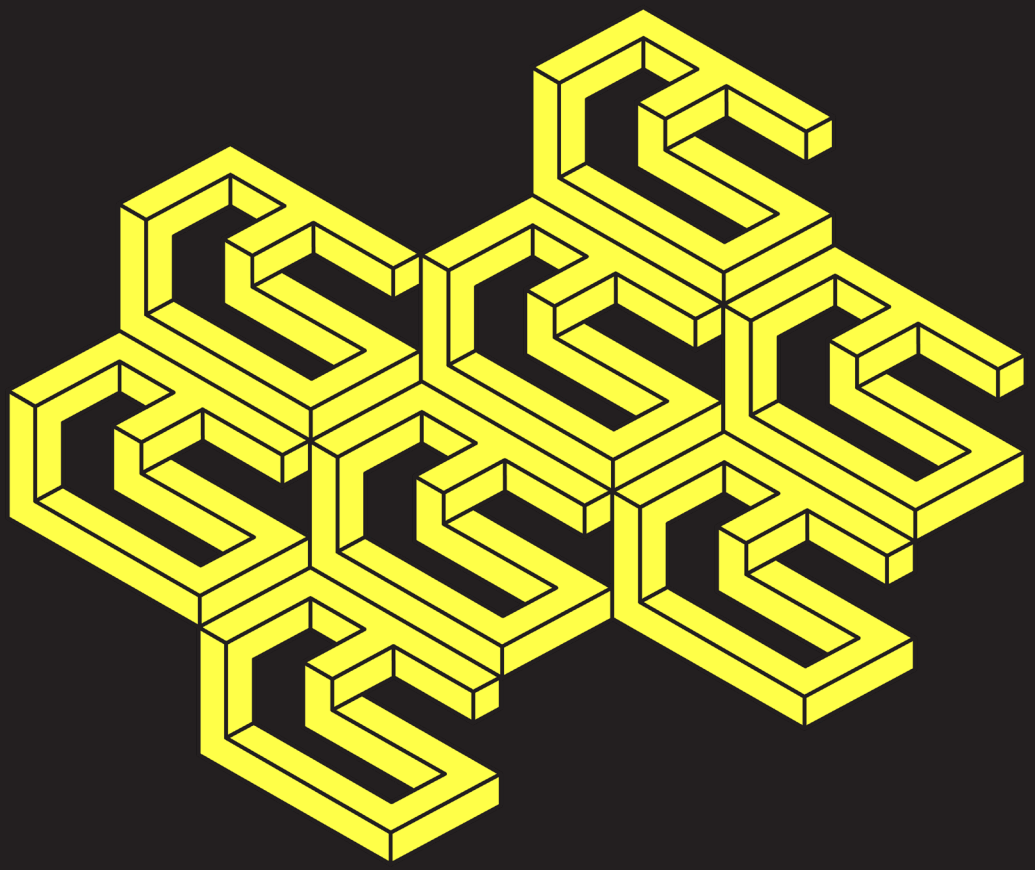

Ribbon Finance Smart Contracts Review

By ChainSafe Systems

June 2021

ChainSafe

Ribbon Finance Smart Contracts Review

Auditor: Oleksii Matiiasevych, Tanya Bushenyova

Warranty

Disclaimer

1. Introduction

Ribbon Finance requested ChainSafe Systems to perform a review of the RibbonThetaVaultYearn (with parts of GammaAdapter used in it) and YearnPricer smart contracts. The contract RibbonThetaVaultYearn can be identified by the following git commit hash:

`050a80b41275baa222ff5a7de013a65399255fe8`

The contract YearnPricer can be identified by the following git commit hash:

`9a75da2ad8beefdaa4caa97d17799b50552ca450`

There are 2 contracts in scope.

After the initial review, Ribbon Finance team applied a number of updates which can be identified by the following git commit hash:

`7346cfe8141e7050a33c565e82a3804bcec4254c`

Additional verification was performed after that.

2. Executive Summary

All the initially identified issues were promptly fixed and are not present in the final version of the contract.

There are **no** known compiler bugs for the specified compiler version (0.7.2), that might affect the contracts' logic.

There were 0 critical, 1 major, 3 minor, 8 informational/optimizational issues identified in the initial version of the contracts. The issues found in Ribbon contracts were not present in the final version of the contracts, and no new issues were discovered. They are described below for historical purposes.The two remaining issues were the informational notes to the YearnPricer contract and the need to keep withdrawal fees from the Vault sufficiently high. See 3.8 and 3.12 for more details.

RibbonThetaVaultYearn manages deposited funds to deposit them in the YearnVault and in turn create options on the OPYN platform. This review is made on the **assumption** that both OPYN and Yearn platforms are secure and safe to use. If that assumption is wrong, then all deposited funds could be lost.

RibbonThetaVaultYearn by itself is deployed using an **upgradable proxy** pattern that is supposed to be managed by Ribbon multisig wallet. This implies that the following review only applies to the exact version identified by the commits in section (1). As soon as Ribbon decides to utilize the upgrade function, this review becomes void, as the upgrade could change any piece of logic, ultimately **taking hold of any deposited funds**.

It is the users' responsibility to make sure that the Ribbon multisig is controlled by a sufficient number of trusted members.

We believe that while the upgradability could be useful, and safe, in the early stages of the protocol, the increase of total value locked (TVL) in the contract might cause unnecessary psychological pressure on the **manager** (being a group might lower the risk, but not remove it), by creating an incentive to seize the power for themselves and take hold of all the funds.

We enjoyed working with the Ribbon Finance team, they were eager to discuss all the feedback and improve their product.

3. Line-by-line review

There were 0 critical, 1 major, 3 minor, 8 informational/optimizational issues identified in the initial version of the contracts. They are described here for historical purposes.

3.1. RibbonThetaVaultYearn, line 315: Minor, `yieldTokensToWithdraw >= yieldTokenBalance` condition should be replaced with `withdrawAmount > yieldTokenBalance` to avoid doing `_withdrawSupplementaryAssetToken()` when everything is already withdrawn.

3.2. RibbonThetaVaultYearn, line 335-339: Minor, the `yieldTokensToWithdraw` value calculation logic could be simplified to `min(yieldTokenBalance, withdrawAmount)` to be cheaper and clearer.

3.3. RibbonThetaVaultYearn, line 349: Note, comment "Withdraws yvWETH from vault" is misleading — the function withdraws the asset to the user. Also, `yieldTokenBalance` param is not described.

3.4. RibbonThetaVaultYearn, line 358: Minor, in case `underlyingTokensToWithdraw` variable is 0, the function will still try to send 0 to the user. Will be automatically fixed by 3.1.

3.5. RibbonThetaVaultYearn, line 372: Note, "Burns vault shares and checks if eligible for withdrawal" comment is incomplete, this function now also unwraps the asset if necessary.

3.6. RibbonThetaVaultYearn, line 398: Note, "transfers amount to relevant recipient" comment is misleading. It transfers to the contract itself.

3.7. RibbonThetaVaultYearn, line 441: **Major**, there is a time window when the short's fate is already known (losing), but the short isn't closed yet (`commitAndClose` yet to be called). In this time window, shareholders could partially avoid losses by withdrawing (if the fee is lower than the loss rate). Especially if the shareholders deposit during the delay, schedule a withdrawal for the whole share and then withdraw before losing.

3.8. RibbonThetaVaultYearn, line 472: Note, there is a time window when the short's fate is already known (winning), but the short isn't closed yet (`commitAndClose` yet to be called). In this time window, someone could make a deposit that will produce up to 99% of the shares, then they will call `closeShort()` to get the profits withdrawn from the option, then they will withdraw 99% of the winning amount, leaving other shareholders with nothing.

3.8 [continued] This attack **will not work** as long as the withdrawal fees of the total vault supply is greater than the individual winning amount.

3.9. RibbonThetaVaultYearn, line 582, 584: Note, `underlyingAsset` and `collateralAsset` can be taken from this contract instead of oToken for gas optimization.

3.10. RibbonThetaVaultYearn, line 558: Optimization, most of the tokens don't require setting the allowance to 0 before changing it to `amountToWrap`. This set of approvals could be replaced by a helper that calls `approve` for `amountToWrap`, and if the approval fails, then it performs the set of 2 approvals (to 0 and to `amountToWrap`).

3.11. GammaAdapter, line 492: Optimization, most of the tokens don't require setting the allowance to 0 before changing it to `depositAmount`. This set of approvals can be replaced by a helper that calls `approve` for `depositAmount`, and if the approval fails, then it performs the set of 2 approvals (to 0 and to `depositAmount`).

3.12. YearnPricer, line 45: Note, since yToken address is not taken from `YearnRegistry` based on `underlyingToken` address, there is a possibility that the configuration will be incorrect. Consider checking the configuration before relying on it.

Tanya Bushenyova Oleksii Matiiasevych