OPTIMUM
BLOCKCHAIN SECURITY

Yearn Finance

Smart contract Security Assessment

Gen Lev Lending Strategy

April, 2022

Table Of Contents

Disclaimer

Overview Page

Summary

Project name	Yearn Finance
URL	https://yearn.finance/
Code	https://github.com/fp-crypto/yearnV2-gen-lev-lending
Commit hash	1f49166f6f5756e788db9d6b0e3f529da3efa848
Mitigations commit hash	
Language	Solidity

Contracts Assessed

Contract name	SHA-1
/contracts/Strategy.sol	745b1a67ae7c7d994e7b2ac0a90ed5f1fae95227
/contracts/LevAaveFactory.sol	692a5aab7c2c80af550c1edcd3f019764dd9098b
/contracts/FlashMintLib.sol	4739514d6a466c8a1cd957962989285f5eb82382

Findings Summary

Severity	Found	Resolved	Partially resolved	Acknowledged
High	0	0	0	0
Medium	0	0	0	0
Low	2	0	0	2
Informational	1	0	0	1
Total	3	0	0	3

Classification of issues

Severity	
High	Vulnerabilities that may directly result in loss of funds, and thus require an urgent fix.
Medium	Issues that may not be directly exploitable, or with a limited impact, are still required to be fixed.
Low	Subjective issues with a negligible impact.
Informational	Subjective issues or observations with negligible or no impact.

Findings

Issue #01	**ERC20 approvals of type(uint256).max**
Severity	Low
Location	Strategy.sol 1. *_initializeThis* (L168-L182) FlashMintLib 1. *doFlashMint* (L95)
Description	Approving the maximum value of uint256 is a known practice to save gas. However, this pattern was proven to increase the impact of an attack many times in the past, in case the approved contract gets hacked.
Recommendation	Consider approving the exact amount that's needed to be transferred, or alternatively, add an external function that allows the revocation of approvals.
Resolution	Acknowledged by the team

Issue #02	***Strategy._sellAAVEForWant* - Potential "sandwiching" front-running vectors**
Severity	Low
Location	Strategy.sol 1. *_claimAndSellRewards* (L542)
Description	*_claimAndSellRewards* is calling *_sellAAVEForWant* with *minOut = 0*, which makes it susceptible to "sandwiching" attacks where the front-runner can order the transactions so that he will be able to buy the asset on a different AMM #1, sell it on the current AMM that is used by the strategy (denoted by AMM #2), to push the price down, then include the transaction that calls *_claimAndSellRewards* which will push the price even lower, then the front-runner can buy the asset on AMM #2 and sell it back on AMM #1.
Recommendation	*_claimAndSellRewards* is used both in *manualClaimAndSellRewards* and in *prepareReturn*. The issue described might be partially resolved by adding a *minOut* parameter to *manualClaimAndSellRewards*, which can be set to a non-zero value when called from *manualClaimAndSellRewards*. Moreover, *prepareReturn* is called in the context of *harvest* which is called by a keeper bot using flashbots API, which is slashed if preforming a sandwich attack, which reduces the risk as well.
Resolution	Acknowledged by the team

Issue #03	Return values never used
Severity	**Informational**
Location	Strategy.sol 1. *_leverDownTo* (L664, L686) 2. *adjustPosition* (L389) 3. *liquidatePosition* (L424)
Description	*_withdrawExcessCollateral, _freeFunds* return values are never used.
Recommendation	Consider using these return values, or alternatively removing them.
Resolution	Acknowledged by the team

Observations

1. The want token, lending token, and borrowing token should be the same, otherwise, the contracts may break.
2. *Strategy.liquidatePosition* may report a non-existent loss, which might be considered a dust sized loss (depending on *minWant*) in case *maxIterations* is not large enough.
3. *minRewardToSell* should be adjusted to support tokens with different decimals (off-chain)
4. *Strategy.tokenToWant* is using spot prices, although the impact is not high since it is being used for read-only functions.
5. Maker governance may set the flash fee to a non-zero value, which will cause failures of main code paths (*adjustPosition, liquidatePosition*). It might be solved by disabling flash-loans in the contract (*isFlashMintActive = false)*.
6. In case of an emergency withdrawal of the position, *maxCollatRatio* should be greater than *targetCollatRatio*, otherwise, funds can not be withdrawn.

Recommendations

Recommendation #01	**Gas optimizations**
Description	1. The call to *IOptionalERC20(dai).decimals()* can be replaced with *DAI_DECIMALS*. 2. *_leverDownFlashLoan* can use a parameter for *borrows* rather than calling *getCurrentPosition* twice within the same code path.

Recommendation #02	***FlashMintLib.doFlashMint* - Add a revert message**
Description	Line 90 will revert for *fee != 0* without a proper message, which may be hard to debug. Consider adding a proper error message.

OPTIMUM

BLOCKCHAIN SECURITY