locus

Locus Security Analysis

by Pessimistic

This report is public

July 17, 2023

Abstract

In this report, we consider the security of smart contracts of Locus project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of Locus smart contracts. We described the audit process in the section below.

The audit showed several issues of medium severity: Excessive position exit, Already withdrawn want tokens are not accounted, Code logic, Do not invest tokens during emergency exit, Inaccurate slippage calculation, Incorrect earned CVX calculations, Suboptimal withdrawal. Also, a number of low-severity issues were found.

During the audit, the developers updated the codebase. This update included fixes and new contracts. However, new contracts were excluded from the scope of the audit.

After the audit, the developers performed another codebase update. They fixed most of the issues and added new tests. After a discussion, we moved the Hardcoded addresses issue from the Low severity issues section to Notes.

The overall code quality is good. The project does not contain any major vulnerabilities; the codebase is thoroughly covered with tests.

General recommendations

We recommend fixing the rest of the issues.

Project overview

Project description

For the audit, we were provided with Locus project on a public GitHub repository, commit 91a825efdcb1d49bc174e818a583dcd92a17fff3.

The scope of the audit included only contracts in the **contracts/strategies/** folder and their dependencies.

The documentation for the project included the following links:

- Locus Finance: One-Page Doc;
- dVault Complete Documentation;
- vETH Complete Documentation.

All 154 tests pass successfully. The code coverage is 99.11%.

The total LOC of audited sources is 3790.

Codebase update #1

During the audit, the developers updated the codebase. Therefore, we switched to commit a023d6db6cc55a57441460f119f41d05128367d4.

This update included fixes to Aura-based strategies, allowing an update of the Aura reward pools. The update also included new contracts in the **contracts/strategies/arbitrum/** folder. However, this folder was not included in the scope of the audit.

Codebase update #2

After the initial audit, the developers performed another codebase update. For the recheck, we were provided with commit def6c51529b4b4119859039d02adaf8da62aa6c9.

In this update, the developers fixed most of the issues mentioned in the initial report and commented on the remaining issues. Moreover, new tests were added.

All 157 tests pass successfully. The code coverage is 99.07%.

Audit process

We started the audit on May 29, 2023, and finished on June 30, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- All of the integrations for the contract: how funds are deposited and prices are calculated;
- Whether the recommendation for yearn vaults and other integrations are considered;
- The possibility of price manipulation attacks;
- The safety of user funds;
- Whether the gas costs could be optimized;
- The places where strategy funds could be saved by performing fewer trades.

We scanned the project with the static analyzer Slither and our plugin Slitherin with an extended set of rules and manually verified the output.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we discussed the results with the developers. On July 14, the developers provided us with an updated version of the code. In this update, they fixed most of the issues from our report and added new tests. They also provided comments on some other issues.

We reviewed the updated codebase and confirmed the fixes.

Finally, we updated the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Excessive position exit (fixed)

The `prepareReturn` function of strategies contracts does not consider the `want` token balance when calculating the amount to withdraw in the `withdrawSome` function. The contract may already be holding `want` tokens, which means a lesser amount can be withdrawn from the positions. This will increase the profit from the strategy.

The issue has been fixed and is not present in the latest version of the code.

M02. Already withdrawn want tokens are not accounted (fixed)

All the strategies contracts include the `liquidatePosition` function. However, in several contracts, the function does not consider the `want` tokens balance when calculating the required withdrawal amount for the `withdrawSome` function. These contracts will not withdraw from positions if the strategy has enough `want` tokens. However, if there are insufficient `want` tokens, the strategies will withdraw the entire `_amountNeeded` from positions. It is possible to partially cover the `_amountNeeded` with the available `want` balance, which could potentially increase the strategy's profitability as fewer tokens are withdrawn from the positions.

The issue occurs in the following contracts:

1. **AuraBALStrategy**,
2. **AuraWETHStrategy**,
3. **FraxStrategy**,
4. **LidoAuraStrategy**,
5. **RocketAura**.

The issue has been fixed and is not present in the latest version of the code.

M03. **Code logic** (fixed)

It is more efficient to sell the earned rewards first in the `_exitPosition` function of the **LidoAuraStrategy** and **RocketAuraStrategy** contracts. In addition to this, we recommend considering these rewards as described in the M07 issue.

The issue has been fixed and is not present in the latest version of the code.

M04. **Do not invest tokens during emergency exit** (fixed)

We recommend checking for emergency exit status in the `adjustPosition` function of the contracts before investing the tokens.

The issue has been fixed and is not present in the latest version of the code.

M05. **Inaccurate slippage calculation** (fixed)

The `_sellCrvAndCvx` function uses the `_expected` variable to reflect the minimal expected swap output in `CRV`. However, the `cvxToWant(_cvxAmount)` call returns a value in `USDC`. The `decimals` value of `USDC` is `8`. Therefore, so there is no slippage protection.

The issue affects two contracts:

1. **CVXStrategy** contract,
2. **FXSStrategy** contract.

The issues have been fixed and are not present in the latest version of the code.

M06. **Incorrect earned CVX calculations** (fixed)

The `convertCrvToCvx` function of the **CVXRewardsMath** library calculates the reward in `CVX` tokens relative to `CRV` as the `mint` function of CVX token. The `convertCrvToCvx` function returns the amount after `if (cliff < totalCliffs)`. If `totalSupply` reaches the maximum, the function should return `0` but returns the whole amount instead.

The issue has been fixed and is not present in the latest version of the code.

M07. **Suboptimal withdrawal** (fixed)

The `_withdrawSome` function of the **FXSStrategy** contract exits its Convex position so that the vault can collect the `_amountNeeded` of `want` tokens. Consider claiming the Convex rewards first to check if `_amountNeeded` does not exceed the available amount of `want` tokens so that the strategy would require withdrawing less tokens.

The issue has been fixed and is not present in the latest version of the code.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. BAL token sell (fixed)

On commit 91a825efdcb1d49bc174e818a583dcd92a17fff3, the `adjustPosition` function of the **AuraBALStrategy** contract sells all `BAL` tokens at line 269. However, later in the code, this function proceeds to purchase BAL tokens. Consequently, the strategy incurs unnecessary losses by selling and rebuying tokens within the same transaction. Similarly, the `prepareReturn` function also includes a `withdrawSome` call with the same sell logic at line 464 before invoking the `adjustPosition` function, leading to a similar loss of funds.

This issue is also present in the **AuraWETHStrategy** contract with the `AURA` tokens.

The issue has been fixed and is not present in the latest version of the code.

L02. High slippage (fixed)

The project uses high slippage rates on swaps in some strategies (5%–20%). These swaps might be vulnerable to sandwich attacks.

> *Comment from the developers:* *We try to use as low slippage rates as possible to make sure our strategies do not revert on swaps. All our strategies also have built-in functionality to change slippage rates in case of a more efficient rate.*

The issue has been fixed. Current slippage rates are mostly 1%–3%. The maximal rate does not exceed 7%.

L03. Code logic (fixed)

The `_sellBalAndAura` function of the **LidoAuraStrategy** contract checks if `(_balAmount == 0 || _auraAmount == 0)` and will not sell any tokens if this statement is `true`. However, it is possible that `_auraAmount` is equal to `0`, but `_balAmount` is not. In that case, no `_balAmount` will be sold.

The issue has been fixed and is not present in the latest version of the code.

L04. Gas usage (commented)

In the `adjustPosition` function, the strategies perform a check to determine that the balance of `want` tokens is sufficient to cover `debtOutstanding` before doing any swaps of `want` tokens. However, if there is an insufficient balance to cover the `debtOutstanding`, it implies that no `want` will be swapped, and hence no tokens could be deposited. As a result, it is possible to return from the function in case of insufficient balance to save gas on external balance check calls.

> *Comment from the developers: If no `want` tokens were swapped, it is still possible that there are other tokens that could be used for opening a position. Therefore, we prefer not to return from the function even in the case when `want` balance is not enough to cover `debtOutstanding`. Balance checks on other tokens should be made first to ensure the correct investment flow of a strategy.*

L05. Default params (fixed)

In the `exchange_multiple` function of the **CurveSwapRouter** contract, the `_pools` argument has a default value (array of zeroes). Thus, this argument can be omitted in swaps while performing a call to the contract.

The issue has been fixed and is not present in the latest version of the code.

L06. FXS rewards are not transferred upon migration (fixed)

`FXS` rewards should be withdrawn along with `CRV` и `CVX` tokens in the `prepareMigration` function of the **FXSStrategy** contract.

The issue has been fixed and is not present in the latest version of the code.

L07. Gas costs (commented)

All strategy contracts in the project request token decimals for every token conversion. We recommend either hardcoding them or storing them in `immutable` variables.

The developers declared decimals for the `want` token as `immutable`. However, other tokens still require external calls for scaling.

> *Comment from the developers:* *Acknowledged.*

L09. Inconsistent ways of price calculation (fixed)

The **LidoAuraStrategy** contract contains two methods for `BPT` price calculation: the `getBptPrice` function that uses the oracle and the `IBalancerPool(bStethStable).getRate()` call inside the `wantToBpt` function. Consider either removing one of the methods or clarifying the necessity of both in the documentation.

The issue has been fixed and is not present in the latest version of the code.

L10. Onchain price evaluation (fixed)

In the **AuraWETHStrategy** contract, the `getBptPrice` function uses onchain price evaluation. The Balancer documentation states that it is crucial to verify that the vault is not being re-entered since this price formula might be manipulated. Although we did not find any attacks on the current strategy, we still recommend checking for re-entrancy.

The issue has been fixed and is not present in the latest version of the code.

L11. Redundant code (commented)

The `adjustPosition` function of the **FXSStrategy** contract has unnecessary decimals scaling at lines 275–279 on commit 91a825efdcb1d49bc174e818a583dcd92a17fff3. This can be simplified by using `FXS.decimals()` in the formula at line 274 instead of `want` token decimals. A similar case is in the **CVXStrategy** contract.

> *Comment from the developers: Good point. However, using `FXS.decimals()` increases precision and this is not what we always want. In case when there is a very small amount of `want` tokens in strategy (i.e. dust), it is better to expect around 0 `FXS` in return. Increasing precision leads to expecting to receive a very small amount of `FXS` (0.00000000001 `FXS`, for example). This leads to UniswapV3 reverting with `Too little received` when trying to get that small amount of tokens. We tried to implement the proposed change and this led to tests failing. Therefore, we prefer to leave that code.*

L12. Redundant code (fixed)

In the **FraxStrategy** contract, the `if` statement at line 110 on commit 91a825efdcb1d49bc174e818a583dcd92a17fff3 is redundant: the contract does not have `frxEth` tokens on its balance since all the `frxEth` tokens are sold just after being redeemed.

The issue has been fixed and is not present in the latest version of the code.

L13. Redundant code (fixed)

In `estimatedTotalAssets` and `prepareMigration` functions of the **FraxStrategy** contract, native ether balance consideration makes no sense since all ether is deposited from the contract after the `WETH.withdraw` call.

The issue has been fixed and is not present in the latest version of the code.

L14. Redundant code (fixed)

The **AuraWETHStrategy** contract contains redundant code in the `liquidateAllPositions` function. Specifically, lines 515–518 on commit 91a825efdcb1d49bc174e818a583dcd92a17fff3 sell tokens and claim rewards, duplicating the code executed at the beginning of the `_exitPosition` function, which is called at line 519 in `liquidateAllPositions`.

The issue has been fixed and is not present in the latest version of the code.

L15. Unused ERC20 returns (fixed)

The return values of ERC20 `approve` calls are not handled throughout the codebase in the project.

The issue has been fixed and is not present in the latest version of the code.

L16. Use of assert (fixed)

The **LidoAuraStrategy** contract contains the `assert` instruction at line 307 on commit 91a825efdcb1d49bc174e818a583dcd92a17fff3. We recommend replacing it with `require` since `assert`s should only be used for cases that are considered impossible.

The issue has been fixed and is not present in the latest version of the code.

L17. Gas usage (fixed)

In aura strategies, `auraRewards` functions include the `balRewards` function call, which makes a call to `IConvexRewards(AURA_BASE_REWARD).earned()`. In some contracts, the `balRewards` function will be called twice (the first time to calculate `balRewards` and the second time to calculate `auraRewards`). We recommend saving the result of the `balRewards` call and reusing it for `auraRewards` calculation.

The issues have been fixed and are not present in the latest version of the code.

L18. Gas usage (fixed)

In convex strategies, `balanceOfCvxRewards` and `balanceOfCrvRewards` functions perform the same `IConvexRewards(FXS_CONVEX_CRV_REWARDS).earned(address(this))` call. Since both functions are always used together, consider performing a single external call and reusing its saved value in these functions to optimize gas consumption.

The issues have been fixed and are not present in the latest version of the code.

Notes

N01. Depeg of stablecoins (commented)

In the **FXSStrategy** contract, if the price of the `FRAX` is lower than the price of `USDC`, the `fxsToWant` function returns more `USDC` tokens than expected. It can lead to the following:

- In the `_sellFxs` function, the `amountOutMinimum` parameter will increase. So it may happen that swap can revert since the allowable price change is very small. Although, the effect might be mitigated because of TWAP used in the `fxsToWant` function.
- In the `adjustPosition` function, the slippage increases due to a decrease of the `fxsExpectedScaled` value.

> *Comment from the developers: Acknowledged.*

N02. Strategist reward that cannot be withdrawn (addressed)

The `harvest` function calls `vault.report` and it calls the `_assessFees` function, in which vault shares are sent to the strategy as a reward. They should generally belong to the owner of the strategy. However, the strategist will not be able to withdraw these tokens, and they get stuck in the strategy since there is no functionality to withdraw shares from the strategy.

> *Comment from the developers: The concept of Locus Finance does not imply the ownership of a strategy by an outside actor like a strategist. So the strategist's reward will always be zero and will not be transferred to the strategy contract.*

N03. Tokens depeg (commented)

In the case of the depeg of the `wstEth` and `rEth` tokens, there could be potential problems since the `bptPrice` is calculated related to the first token in the pool. In case the `bptPrice` is undervalued in the strategy, the strategy could lose tokens while exiting the pool due to slippage since the expected amount will be set lesser.

> *Comment from the developers: Acknowledged.*

N04. Incorrect reward calculation (commented)

Due to logic in `AuraRewardsMath.convertCrvToCvx` function, after the `inflationProtectionTime` time has passed, the `AURA` token rewards will be estimated as `0` though there still could be rewards earned. This affects the result of the `estimatedTotalAssets` function, decreasing its value.

> *<u>Comment from the developers:</u> Due to the fact that the theoretical possibility of minting AURA rewards is present, it is currently unknown how this will be implemented in the future. The current inflation protection algorithm is taken into account in the strategy.*

N05. **Hardcoded addresses** (commented)

Since Convex booster addresses might change, we recommend adding functionality to update them.

> *<u>Comment from the developers:</u> Since apparently changing the address of the booster is not a common scenario, we decided to leave the address hardcoded. If it changes, the strategy will be updated using the migration mechanism.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Pavel Kondratenkov, Senior Security Engineer
Daria Korepanova, Senior Security Engineer
Ivan Gladkikh, Security Engineer
Yhtyyar Sahatov, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

July 17, 2023