Locus Arbitrum Strategies Security Analysis

by Pessimistic

This report is public

July 28, 2023

Abstract

In this report, we consider the security of smart contracts of the 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 initial audit showed two issues of medium severity: Double accounting of the balance and Possible underflow. Also, a few low-severity issues were found.

After the initial audit, the codebase was updated. All issues were fixed or commented on. However, one new issue of low severity was found. This issue was quickly fixed by the developers.

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 remaining issues.

Project overview

Project description

For the audit, we were provided with the Locus project on a public GitHub repository, commit 86b37d91bd0b6d1613f13c668eeccab068080096.

The scope of the audit included only the **strategies/arbitrum/** folder.

The documentation for the project included Arbitrum Yield Index.

All 66 tests pass successfully. The code coverage is 98.49%.

The total LOC of audited sources is 1166.

Codebase update

After the initial audit, we were provided with commit cd82421626669515f6fadd844d411bc9230a9659.

In this update, the developers fixed most of the issues mentioned in the initial report and commented on the remaining issues. However, we found one new issue of low severity (L05), which was quickly fixed afterward by the developers.

All 66 tests pass successfully. The code coverage is 97.53%.

Audit process

We started the audit on July 13, 2023, and finished on July 21, 2023.

The audit scope included a new part of the project we reviewed previously. Thus, we were familiar with the project and inspected only particular contracts within the repository.

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 safety of user funds;
- Whether the gas costs could be optimized.

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 24, the developers provided us with an updated version of the code.

We reviewed the updated codebase and confirmed the fixes.

Moreover, we re-run the tools, tests and calculated the code coverage.

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. Double accounting of the balance (fixed)

In **GMDStrategy** and **GNSStrategy** contracts, the `_withdrawSome` function considers the balance of `want` tokens when calculating the amount of tokens to withdraw from the position: it subtracts the `want` balance from the amount to unstake. However, the following functions perform a duplicating subtraction of the `want` token balance:

- The `liquidatePosition` function passes the `_amountNeeded - _wantBal` value to the `_withdrawSome` function. As a result, a user will lose the amount of tokens equal to the balance of `want` tokens on the contract. Also, the call will revert if the `_amountNeeded` value is less than `2 * _wantBal`. Note that the user might not lose tokens if they pass the correct `maxLoss` parameter to the `withdraw` function of the vault contract;
- The `prepareReturn` function subtracts the balance of `want` from the value passed to the `_withdrawSome` function.

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

M02. Possible underflow (fixed)

In **GMDStrategy** and **GNSStrategy** contracts, the expression `_amountNeeded - balanceOfWant()` in the `_withdrawSome` function might underflow if the returned value of the `balanceOfWant` function is greater than the `_amountNeeded` value. Note that the balance could also become greater than the `_amountNeeded` after the `_sellRewards()` function is called.

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. Gas consumption (commented)

GMXStrategy and **JOEStrategy** contracts perform consequential calls to the `decimals` function of the token contract (directly and in `Utils.scaleDecimals()`) for every token conversion. We recommend hardcoding these values in the project or storing them as immutable variables to optimize gas consumption.

> *Comment from the developers: Acknowledged.*

L02. Incorrect argument usage (fixed)

In the **GMDStrategy** contract, consider replacing the `address(want)` argument with `WETH` at the lines 180 and 194.

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

L03. Unused return value (fixed)

In all strategies of the project, the returned value of the erc20 `approve` function is never used.

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. Redundant code (fixed)

We recommend removing the declaration of the `DAI_USDC_UNI_V3_POOL` variable in the **GNSStrategy** contract, since the variable is not used. This will improve the readability of the code.

This issue has been fixed in commit e2bd6811d1f875f66fa7f6d414c380fbec6f40e3.

Notes

N01. Depeg of stablecoin (commented)

In the **JOEStrategy** contract, the price of the `USDC` token is assumed as 1 USD, which could be wrong in the case of the `USDC` depeg.

> *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. Depeg of stablecoins (commented)

In the **GNSStrategy** contract, it is assumed that the price of the `USDC` token is equal to the price of the `DAI` token. It can lead to various problems if either of the stablecoins depegs.

We recommend getting the prices using the TWAP value from the pool (`0x7CF803e8d82A50504180f417B8bC7a493C0a0503`), where swaps of these tokens occur in this strategy.

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

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

July 28, 2023