ALGEBRA PERIPHERY SECURITY AUDIT REPORT

May 25, 2023

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals

- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Algebra Finance is an AMM and a concentrated liquidity protocol for decentralized exchanges running on adaptive fees.

Periphery part of the protocol allows users to control their liquidity and limit orders in an easier way. The current implementation of the periphery contracts doesn't allow to work with rebaseable tokens.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Algebra
Project name	Periphery
Timeline	May 11 2023 - May 23 2023
Number of Auditors	4

Project Log

Date	Commit Hash	Note
11.05.2023	bddd6487c86e0d6afef39638159dc403a91ba433	Commit for the audit

Date	Commit Hash	Note
22.05.2023	9d37b1b89da6d9aa1e235d4a198de384539e5a5b	Commit for the reaudit

Project Scope

The audit covered the following files:

File name	Link
LimitOrderManager.sol	LimitOrderManager.sol
BlockTimestamp.sol	BlockTimestamp.sol
ERC721Permit.sol	ERC721Permit.sol
LimitOrderManagement.sol	LimitOrderManagement.sol
Multicall.sol	Multicall.sol
PeripheryImmutableState.sol	PeripheryImmutableState.sol
PeripheryPayments.sol	PeripheryPayments.sol
PeripheryValidation.sol	PeripheryValidation.sol
SelfPermit.sol	SelfPermit.sol
CallbackValidation.sol	CallbackValidation.sol
ChainId.sol	ChainId.sol
PoolAddress.sol	PoolAddress.sol
PositionKey.sol	PositionKey.sol

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	3
Low	11

ID	Name	Severity	Status
M-1	Incorrect usage of the parameter	Medium	Fixed
M-2	The operator address is not reset after the position token transfer	Medium	Fixed
M-3	An inverted price is used for overflow validation during the creation of new limit orders	Medium	Fixed
L-1	Possible reverts with no message	Low	Fixed
L-2	Callback verification can be made more secure	Low	Acknowledged
L-3	Contract `LimitOrderManagement` can be removed	Low	Fixed
L-4	The `LimitOrderManagement._createLimitOrder()` duplicated pool address calculation	Low	Fixed
L-5	Accidental calling of `LimitOrderManagement.collectLimitOrder()` with zero recipient may lead to tokens stealing	Low	Acknowledged
L-6	Gas optimization	Low	Fixed
L-7	The missing check for a recipient's address	Low	Acknowledged

L-8	An unnecessary subtraction operation	Low	Fixed
L-9	`decreaseLimitOrder` can be called multiple times on an emptied order	Low	Fixed
L-10	Unprotected `sweep` function	Low	Acknowledged
L-11	Code impossible to reach out	Low	Acknowledged

1.6 Conclusion

During the audit, no CRITICAL or HIGH vulnerabilities were found. Only 3 MEDIUM and 11 LOW severity findings were spotted. After working on the reported findings, all of them were acknowledged or fixed by the client.

Disclaimer

The client could provide the smart contracts for the deployment by a third party. To make sure that the deployed code hasn't been modified after the last audited commit, one should conduct their own investigation and deployment verification.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Incorrect usage of the parameter
Severity	Medium
Status	Fixed in 9d37b1b8

Description

There are not enough checks on the amount of liquidity that can be removed from the limit order in the `decreaseLimitOrder` function. `liquidity` can be greater than the available liquidity on the limit order and in this case tx will revert. LimitOrderManager.sol#L162

Recommendation

We recommend decreasing `liquidity` to `cache.liquidityLast` if `liquidity > cache.liquidityLast`.

M-2	The operator address is not reset after the position token transfer
Severity	Medium
Status	Fixed in 9d37b1b8

Description

A NFT position owner can set token `operator` to use permit functionality - LimitOrderManager.sol#L275. The operator address is used at the line LimitOrderManager.sol#L270. The `getApproved` function is used in a ERC721 standard to get an address of the entity which is allowed to make transfers. The user may give a permit to desired `operator`, then sell/transfer the NFT token. A new owner may not know about the given permit - the token operator is still able to make transfers and burn/collect a limit order position on behalf of the new token owner.

Recommendation

We recommend overriding the ERC721 `_transfer` functionality and resetting the `_limitPositions[tokenId]` value on transfer.

M-3	An inverted price is used for overflow validation during the creation of new limit orders
Severity	Medium
Status	Fixed in 9d37b1b8

Description

In function `core.LimitOrderManagement.addOrRemoveLimitOrder` the price used to evaluate whether the parameter `amountToBuy` is prone to overflow or does not appear to be inverted: LimitOrderManagement.sol#L46-L48

```
uint256 amountToBuy = (tick > currentTick)
  ? FullMath.mulDivRoundingUp(_amountToSell, Constants.Q144, priceX144)
  : FullMath.mulDivRoundingUp(_amountToSell, priceX144, Constants.Q144);
```

Price is represented as the ratio `token1/token0`. In the specified expression, if `tick` is greater than `currentTick`, then the new limit order position will sell `token0` in exchange for `token1`. Accordingly, to calculate the `amountToBuy`, `_amountToSell` , it should be multiplied by `priceX144` instead of being divided.

Recommendation

We recommend inverting the specified expression.

2.4 Low

L-1	Possible reverts with no message
Severity	Low
Status	Fixed in 9d37b1b8

Description

There are some places in the contracts that have the `require` checks without a revert message:
LimitOrderManager.sol#L207
PoolAddress.sol#L30.

Recommendation

We recommend adding a message to all `require` statements.

L-2	Callback verification can be made more secure
Severity	Low
Status	Acknowledged

Description

The current callback verification is safe enough, but to increase the security of the protocol we would recommend saving the address of the pool that is used in the `_createLimitOrder` function to the storage variable to check that callback is called by the specific pool.
LimitOrderManagement.sol#L24

Recommendation

We recommend increasing the security of the callback validation.

Client's commentary

> In order to save gas, we prefer the existing mechanism, as it is reliable.

L-3	Contract `LimitOrderManagement` can be removed
Severity	Low
Status	Fixed in 9d37b1b8

Description

Some functions of `LimitOrderManager` are placed in `LimitOrderManagement`. This contract has only one inheritor. There is also another contract with name `LimitOrderManagement` in the core part of the project. It may lead to confusion.

Recommendation

We recommend moving functions from `LimitOrderManagement` to `LimitOrderManager` and removing `LimitOrderManagement`.

L-4	The `LimitOrderManagement._createLimitOrder()` duplicated pool address calculation
Severity	Low
Status	Fixed in 9d37b1b8

Description

`LimitOrderManagement._createLimitOrder()` is called only from `LimitOrderManager.addLimitOrder()`. But it recalculates the address of the pool that was already calculated in `LimitOrderManagement._createLimitOrder()`:
LimitOrderManager.sol#L81
LimitOrderManagement.sol#L39.

Recommendation

We recommend using a pool address as a parameter.

L-5	Accidental calling of `LimitOrderManagement.collectLimitOrder()` with zero recipient may lead to tokens stealing
Severity	Low
Status	Acknowledged

Description

If `LimitOrderManagement.collectLimitOrder()` is called by accident for zero recipient, tokens will be transferred to the contract address:
LimitOrderManager.sol#LL229
After that, anyone can claim them.

Recommendation

We recommend reverting `LimitOrderManagement.collectLimitOrder()` if `recipient` is `address(0)`.

Client's commentary

> This logic is used to handle wrapped native currency. After transferring tokens to LimitOrderManager, unwrapWNativeToken can be called in multicall.

L-6	Gas optimization
Severity	Low
Status	Fixed in 9d37b1b8

Description

There is a `_getAndIncrementNonce` function at the line LimitOrderManager.sol#L262. Its logic can be put into an `unchecked` block to skip overflow checks (like it is done here - NonfungiblePositionManager.sol#L480).

Recommendation

We recommend putting the function code into the `unchecked` block.

L-7	The missing check for a recipient's address
Severity	Low
Status	Acknowledged

Description

There are two functions - PeripheryPayments.sol#L21 and PeripheryPayments.sol#L32 which accept the `recipient` address as a parameter. As the `recipient` parameter value is not checked, it is possible to transfer tokens to zero address.

Recommendation

We recommend adding checks that the `recipient` is not a zero address in both functions.

Client's commentary

> These checks must occur on the side of the caller of these methods. We prefer not to add these checks at this level.

L-8	An unnecessary subtraction operation
Severity	Low
Status	Fixed in 9d37b1b8

Description

At the line LimitOrderManager.sol#L246 the `position.tokensOwed0` and `position.tokensOwed1` values are set to the result of subtraction of the previous `tokensOwed` value and the amount collected. But at line LimitOrderManager.sol#L239 the collected values are initialized with the owed values. It is unnecessary to do a subtraction when both operators have the same value.

Recommendation

We recommend assigning zero values to `position.tokensOwed0` and `position.tokensOwed1` here - LimitOrderManager.sol#LL246.

L-9	`decreaseLimitOrder` can be called multiple times on an emptied order
Severity	Low
Status	Fixed in 9d37b1b8

Description

There is a function LimitOrderManager.sol#L137 which is used to reduce a user's position (or fully close limit orders). When a user removes all liquidity from their position but not burns the NFT token, they are still able to call `decreaseLimitOrder` multiple times.

Recommendation

We recommend adding a check for `position.liquidity` being greater than zero at the beginning of the `decreaseLimitOrder` function.

L-10	Unprotected `sweep` function
Severity	Low
Status	Acknowledged

Description

Currently, any user can call the sweep function and get all tokens from the contract.
PeripheryPayments.sol#L32-L48

Recommendation

We recommend adding an admin role for calling this function.

Client's commentary

> This contract shares functionality with NonfungiblePositionManager, so we prefer to leave it as it is.

L-11	Code impossible to reach out
Severity	Low
Status	Acknowledged

Description

There is a branch in the `pay` function that cannot be reached: PeripheryPayments.sol#L64-L66.

Recommendation

We recommend removing this branch from the contract.

Client's commentary

> This code is used in contracts that were not included in the scope.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes