ALGEBRA FINANCE CORE SECURITY AUDIT REPORT

August 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 that allows the usage of adaptive fees. Algebra core is a base level of a protocol that contains all logic with swap calculations and manages users' liquidity.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Algebra Finance
Project name	Core
Timeline	August 02 2023 - August 22 2023
Number of Auditors	4

Project Log

Date	Commit Hash	Note
02.08.2023	6c22b64977e0b0266aec89470480df74977eb606	Commit for the audit
21.08.2023	5f182b8403ffd8d77aceef4383cd2f5283205e32	Commit for the reaudit

Project Scope

The audit covered the following files:

File name	Link

File name	Link
AlgebraCommunityVault.sol	AlgebraCommunityVault.sol
AlgebraFactory.sol	AlgebraFactory.sol
AlgebraPoolDeployer.sol	AlgebraPoolDeployer.sol
AlgebraPool.sol	AlgebraPool.sol
AlgebraPoolBase.sol	AlgebraPoolBase.sol
Positions.sol	Positions.sol
ReentrancyGuard.sol	ReentrancyGuard.sol
ReservesManager.sol	ReservesManager.sol
SwapCalculation.sol	SwapCalculation.sol
TickStructure.sol	TickStructure.sol
Timestamp.sol	Timestamp.sol
Constants.sol	Constants.sol
LiquidityMath.sol	LiquidityMath.sol
LowGasSafeMath.sol	LowGasSafeMath.sol
Plugins.sol	Plugins.sol
PriceMovementMath.sol	PriceMovementMath.sol
SafeCast.sol	SafeCast.sol
SafeTransfer.sol	SafeTransfer.sol
TickManagement.sol	TickManagement.sol
TickTree.sol	TickTree.sol

File name	Link
TokenDeltaMath.sol	TokenDeltaMath.sol

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	1
Medium	4
Low	9

ID	Name	Severity	Status
H-1	Pool initialization can be front-run	High	Acknowledged
M-1	Ownership renouncement will grant a role to zero address	Medium	Fixed
M-2	A broken hook can block user funds	Medium	Acknowledged
M-3	Active pool `plugin` can be modified without updating `pluginConfig`	Medium	Fixed
M-4	Admin can set non-zero `pluginConfig` in `pool` with zero `plugin`	Medium	Fixed
L-1	Requires without a message	Low	Fixed
L-2	Zero fees will reset accrued rewards	Low	Acknowledged

L-3	Dangerous type casting	Low	Acknowledged
L-4	A flash loan fee sandwich attack	Low	Acknowledged
L-5	No parameter redundancy checks	Low	Fixed
L-6	No explicit checks for `mint` and `swap` calls on uninitialized Pool	Low	Fixed
L-7	Missing event emissions in `AlgebraCommunityVault`	Low	Fixed
L-8	Users can't control upgrades in the plugin implementation	Low	Acknowledged
L-9	`swap` function can be executed without any tokens	Low	Acknowledged

1.6 Conclusion

During the audit process 1 HIGH, 4 MEDIUM, and 9 LOW severity findings were spotted. After working on the reported findings, all of them were acknowledged or fixed by the client. No other issues were identified that could affect the protocol's function.
Test coverage of the protocol is sufficient. A lot of different edge-case scenarios are covered in the tests. Also, there are a lot of fuzzing tests written with Echidna that help check invariants in the code. Protocol developers care about documentation and comments in the code, which helps understand complex parts. The overall quality of the code is very high which is why there are not so many findings in the report. One thing developers should keep in mind for the next scopes is the readability of the code. Several places in the code were changed to increase the readability of the code during the audit. The most complex part should be covered with comments and should be easy to read.

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

H-1	Pool initialization can be front-run
Severity	High
Status	Acknowledged

Description

After the pool deployment AlgebraFactory.sol#L95, the initialize function isn't called which allows anyone to set the initial price in the pool. The price setting can affect liquidity addition in the `mint` function. Initialization can also be front-run if the pool is created from the periphery, so this problem allows a malicious user to ddos any pool.
Also, the `swap` function can be executed without any tokens from the `msg.sender` if it operates entirely within an interval with zero active liquidity. Concurrently, the `mint` function in the `AlgebraPool` contract does not include the `amount0Max` and `amount1Max` parameters making it vulnerable to undesirable price fluctuations. This creates a loophole that malicious actors can exploit capitalizing on price slippage to their advantage.

For clarity, consider the following sequence of events (these events can also take place after the pool creation):

1. A liquidity provider (LP) initiates the `mint` transaction to the `AlgebraPool` with the current price set to `P` and the pool's actual liquidity standing at zero. The LP's intention is to set a position with the `amount0` of token0 and `amount1` of token1.
2. An exploiter front-runs this action by performing a `swap` which shifts the price to `P' < P` undervaluing `token0`. This operation costs the exploiter nothing in terms of `token0` due to the absence of liquidity.
3. Subsequently, the LP mints the new position acquiring `amount0' < amount0` of token0 and `amount1' > amount1` of token1.

4. Then the exploiter conducts another `swap` acquiring `token0` and reverting its price back to the prevalent market rate capitalizing on the arbitrage window.
5. These actions result in the LP contributing more of `token1` than it was initially planned incurring a loss.

This issue is classified as `high` severity. It offers bad actors an opportunity to exploit the lack of price slippage checks in the `mint` function. Even though conditions without `amount0Max` and `amount1Max` might be managed through the `periphery`, it still exposes the system to the near-zero cost of significant price shifts within intervals with zero active liquidity. This could potentially pave the way for DoS attacks targeting liquidity positions.

Recommendation

We recommend calling the initialize function after the pool deployment in the `createPool` function. Also, we recommend implementing constraints on the `swap` function, especially when the swap ends in a non-initialized tick with zero liquidity. This will retain malicious entities from manipulating prices within intervals of zero liquidity and will limit the feasibility of DoS attacks in such scenarios. For solving the problem when the pool is initialized with a correct price but a malicious actor uses token-free swaps to move the price, we recommend adding a method that allows users to move price and add liquidity in one tx. Price moving, in this case, will be token-free because total pool liquidity will be 0.

Client's commentary

> This scenario is expected and does not contradict the logic of the protocol, it is considered safe.
> First of all, by design, the Core contracts are intended to be interacted using special Periphery contracts that make all necessary security checks (including slippage checks for all actions). Direct interaction with pool contracts (as EOA) is impossible because of the callback mechanic. When a pool is created through Algebra's periphery contracts, the pool is initialised in the same transaction as its creation. After that, the mint is made with a slippage check, which prevents a significant front-run attack from being made. In addition, from the point of view of a potential attacker, it does not matter whether the initialization is performed immediately after the creation of the pool or not: the attacker could independently create the pool during the front run with the price he needs.
> Regarding the recommendation to combine a "free" swap (which is not really free) up to the desired price and a mint: such a mechanism can be implemented, but it must be taken into account that it can be prevented by preliminary placement of liquidity at the frontrun, which will make the swap not free and such a protective mechanism meaningless. In this case, tight slippage control is more effective, which makes frontrunning economically unfeasible.
> In our opinion, the scenario described here is not specific to the initialization of the pool: there are various scenarios in which the price of the pool may differ from the market one.
> Also, in our opinion, the described scenario does not correspond to the definition of severity "High" given in the description of the audit methodology, but rather corresponds to "Medium". As a result of setting the wrong price in the pool, after the initialization frontrun or for any other reason, the pool remains in a fully operational state, this is a normal situation, corresponding to the logic and

> mathematical model of AMM. This state cannot be described as "failure". Further, the price can be set at market values using standard pool mechanisms: swaps and liquidity providing.

2.3 Medium

M-1	Ownership renouncement will grant a role to zero address
Severity	Medium
Status	Fixed in 5f182b84

Description

After the ownership renouncement, the owner will be zero address, and it will be granted with `DEFAULT_ADMIN_ROLE` AlgebraFactory.sol#L165.

Recommendation

We recommend adding a check that `owner` is not a zero address.

M-2	A broken hook can block user funds
Severity	Medium
Status	Acknowledged

Description

For closing position, a user should call the `burn` function which uses hooks to external contract AlgebraPool.sol#L137. If a hook were broken, then users would not be able to get the deposited funds.

Recommendation

We recommend adding an emergency exit for users that can be triggered if the hook breaks.

Client's commentary

> Liquidity withdrawal prevention may be an expected behavior in this case: additional requirements can be part of the plugin design.
> If we consider the case of plugin failure, the addition of emergency withdraw mechanisms built into the pool may create new attack vectors and disrupt the logic of the protocol.
> Multisigs and other control mechanisms should ensure that a broken plugin can be disabled as well as prevent malicious actions by an administrator. See comment for Low#8.

M-3	Active pool `plugin` can be modified without updating `pluginConfig`
Severity	Medium
Status	Fixed in 5f182b84

Description

AlgebraPool.sol#L402
An issue has been found in the `setPlugin` function.
Pool administrators can alter the `plugin` of an active pool. However, there is a discrepancy as the `pluginConfig` of the newly-introduced `plugin` might differ from its predecessor. This could lead to invoking unimplemented hooks that were previously operational in the old `plugin`. Furthermore, if the former `plugin` employed dynamic fees set beforehand, and the succeeding plugin operates with static fees or vice versa, the pool `fee` parameter stays in an inconsistent state resulting in incorrect fee withdrawals. Additionally, when associating a `plugin` to an active `pool`, certain pre-initialization logic might be imperative, particularly when establishing the new plugin configuration, initializing internal local variables, and designating the new fee.

This vulnerability is classified as `medium` as linking a new `plugin` to the pool could temporarily block specific `pool` functionality and unintentionally trigger inappropriate `fee` actions, persisting until the correct `plugin` initialization is carried out.

Recommendation

We recommend adding a hook call to the `plugin` as part of the `setPlugin` function. This inclusion should initialize the `plugin`'s local variables and concurrently define the fresh `pluginConfig` and `fees` within the `pool`.

M-4	Admin can set non-zero `pluginConfig` in `pool` with zero `plugin`
Severity	Medium
Status	Fixed in 5f182b84

Description

AlgebraPool.sol#L431
The `Administrator` can set a non-zero value for `pluginConfig` while setting the `plugin` to zero. Such a configuration can potentially trigger reverts in pool functions.
This issue is classified as `medium`, as the combination of a non-zero `pluginConfig` with a zero `plugin` can disrupt certain functionalities of the `pool`.

Recommendation

In order to ensure consistency and prevent potential disruptions, we recommend including a conditional check: `if (plugin == 0) require(pluginConfig == 0);`.

2.4 Low

L-1	Requires without a message
Severity	Low
Status	Fixed in 5f182b84

Description

There are some `require` checks that do not contain a revert message (e.g., AlgebraCommunityVault.sol#L37). If some logic broke in the contract, then it would be tough to find the specific place which leads to the revert.

Recommendation

We recommend adding revert messages to all `require` checks.

L-2	Zero fees will reset accrued rewards
Severity	Low
Status	Acknowledged

Description

If the admin of the community vault decided to set algebra fee to 0, then all previously accumulated rewards would be lost
AlgebraCommunityVault.sol#L79-L82.

Recommendation

We recommend being aware of these fee mechanics.

Client's commentary

> Works as designed. The current implementation does not store information about collected fees and assumes an indefinite number of different tokens on the vault contract, this does not allow to implement the fees change otherwise.

L-3	Dangerous type casting
Severity	Low
Status	Acknowledged

Description

Type casting here AlgebraPool.sol#L146 can lead to a potential loss of user funds.

Recommendation

We recommend using a safe type cast there.

Client's commentary

> We prefer the current implementation as being more secure for users. By design, token reserves cannot exceed uint128, so this casting is safe. Even if an overflow were possible, it would be better than a complete withdrawal lock.

L-4	A flash loan fee sandwich attack
Severity	Low
Status	Acknowledged

Description

AlgebraPool.sol#L328
The flash loan fee is spread among current active liquidity providers. A malicious actor can get a flash loan transaction from the mempool and wrap it into a sandwich:

1. `AlgebraPool.mint()` with highly concentrated liquidity to the pool (for the current active tick with the minimal interval).
2. A flash loan transaction from the mem pool.
3. `AlgebraPool.burn()`.
   So, the malicious actor will get a big amount of flash loan fee from liquidity providers.

Recommendation

We recommend distributing flash loan commissions equally among all LPs.

Client's commentary

> The flashloan sandwich is not a problem in this case: by design, in the absence of active liquidity, the first active liquidity positions will receive the collected fees.
> Although it would be more preferable to distribute fees among all liquidity providers, adding such a mechanism is not trivial and may require very significant changes and have various additional consequences.

L-5	No parameter redundancy checks
Severity	Low
Status	Fixed in 5f182b84

Description

Certain setters in the `AlgebraCommunityVault` and `AlgebraPool` contracts are missing checks to ensure that the incoming parameter value isn't identical to the existing one. Specific instances include:
AlgebraCommunityVault.sol#L102
AlgebraCommunityVault.sol#L123
AlgebraCommunityVault.sol#L134
AlgebraPool.sol#L423.

Recommendation

We recommend integrating the necessary redundancy checks into the mentioned functions to ensure no reassignment with identical values.

L-6	No explicit checks for `mint` and `swap` calls on uninitialized Pool
Severity	Low
Status	Fixed in 5f182b84

Description

AlgebraPool.sol#L66 AlgebraPool.sol#L210 AlgebraPool.sol#L253 The `swap` and `mint` functions can be invoked on an uninitialized pool, leading to a potential `revert` midway through execution due to invariant checks. This might introduce vulnerabilities if there are modifications to the implementation in subsequent updates.

Recommendation

We recommend adding an explicit precondition to preempt the following errors: `require(globalState.price != 0)`, to the`swap`, `swapWithPaymentInAdvance`, and `mint` functions.

L-7	Missing event emissions in `AlgebraCommunityVault`
Severity	Low
Status	Fixed in 5f182b84

Description

Certain setters within the `AlgebraCommunityVault` contract do not emit events upon execution. Specific instances include:
AlgebraCommunityVault.sol#L111
AlgebraCommunityVault.sol#L123
AlgebraCommunityVault.sol#L129.

Recommendation

To enhance transparency and traceability, we recommend implementing appropriate event emissions for the above-mentioned functions.

L-8	Users can't control upgrades in the plugin implementation
Severity	Low
Status	Acknowledged

Description

Plugins can affect the number of tokens that users will receive after the swap, so updating the plugin without a time delay can negatively affect users (AlgebraPool.sol#L401).

Recommendation

We recommend adding a time delay before updating the plugin implementation.

Client's commentary

> By design, ensuring control must be implemented by external means, such as governance with timelock.

L-9	`swap` function can be executed without any tokens
Severity	Low
Status	Acknowledged

Description

The `swap` function can be executed without any tokens from the `msg.sender` if it operates entirely within an interval with zero active liquidity.

Recommendation

We recommend implementing constraints on the `swap` function, especially when the swap ends in a non-initialized tick with zero liquidity.

Client's commentary

> Works as designed. Such swaps can be used to adjust the incorrectly set price.
> Additionally, it is impossible to swap with 0 input tokens, it is necessary to provide a non-zero number of tokens.

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