BAILSEC.IO

OFFICE@BAILSEC.IO

X: @BAILSECURITY

TG: @HELLOATBAILSEC

FINAL REPORT:

Algebra Integral Update Audit (differential)

May 2024

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Algebra Integral - Update Audit (differential)
Website	algebra.finance
Type	AMM
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/cryptoalgebra/Algebra/tree/45c65f0687b59910b28f7fb8fc59f7caae22e958/src
Resolution 1	https://github.com/cryptoalgebra/Algebra/tree/0fe96d86be5ae446901d3abb244b96be7600adeb

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2			2
Medium	1			1
Low	4	2		2
Informational				
Governance	2			2
Total	9	2		7

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

Bailsec was tasked with a differential audit of Algebra's CLAMM:

Old Commit:

https://github.com/cryptoalgebra/Algebra/tree/eea3a9bda1b623dfc3b6f606a8ef08dec76f18b2/src

New Commit:

https://github.com/cryptoalgebra/Algebra/tree/45c65f0687b59910b28f7fb8fc59f7caae22e958/sr

Files in Scope:

1. src/core/contracts/AlgebraFactory.sol
2. src/core/contracts/AlgebraPool.sol
3. src/core/contracts/AlgebraPoolDeployer.sol
4. src/core/contracts/AlgebraVaultFactoryStub.sol
5. src/core/contracts/base/AlgebraPoolBase.sol
6. src/core/contracts/base/ReservesManager.sol
7. src/farming/contracts/FarmingCenter.sol
8. src/periphery/contracts/libraries/PoolAddress.sol
9. src/plugin/contracts/BasePluginV1Factory.sol

Primary Update Overview:

The key update in this differential audit involves mainly two distinct features:

a) **Custom Pool Deployment:** Whitelisted addresses have the authority to deploy custom pools. These pools are additionally to the standard base pools and can have the exact same token pairs. The distinction between these pools and the base pools is the incorporation of the “deployer” address, which is address(0) for the base pool deployment. The rationale behind custom pools is the implementation of various different plugins, which will have synergistic effects with Algebra’s plugin marketplace.
This is facilitated by modifications primarily in the AlgebraPoolDeployer.sol and AlgebraFactory.sol contracts.

b) **Skim and Sync:** As already known from the standard UniswapV2 implementation, the skim and sync functions are used to update the balance in an effort to match the reserves. For instance if a direct transfer of an ERC20 token is happening to the pair, this can result in a short term unupdated state of the pair. Usually the state is updated whenever an interaction is happening with the pair, as the _updateReserves is invoked which sets the reserves to the corresponding balance. Algebra has now implemented a sync and skim function, which is callable by the plugin. Skim allows for withdrawing the excess balance to the plugin and sync allows for force-matching the reserves via the already known _updateReserves function.

   This is facilitated by modifications within the AlgebraPool.sol and ReservesManager.sol contracts.

Security Considerations:

a) The deployment of custom pools by only whitelisted addresses limits the risk of unauthorized or malicious pool creations, thereby maintaining a controlled expansion environment. The security of these custom pools relies heavily on the integrity and robust management of the whitelist mechanism.

b) The incorporation of sync and skim exposes a slight modification of the core, security risk is moderate due to the fact that only the plugin can invoke this function and reentrancy checks are present. However, for custom pools with potentially malicious plugins, all scenarios must be considered to ensure maximum safety.

Disclaimer: This audit involves only the changes provided by the corresponding diffchecker files. Please be advised that for issues which are reported outside of the diffchecker scope, an additional resolution must be scheduled. A differential audit is always a constrained task because not the full codebase is re-audited. This will have inherent consequences if intrusive changes have side-effects on parts of a codebase/module, which is not part of the audit scope.

Deployment

AlgebraFactory

The AlgebraFactory contract is the entry contract for deploying pools. Anyone can deploy a pool as long as that pool is non-existent. Furthermore, it handles all privileged functions for factory pointers like the PluginFactory and VaultFactory as well as fee settings.

As part of the update, the createCustomPool function was added, which allows privileged addresses (with the CUSTOM_POOL_DEPLOYER role) to create additional pools, which are not connected to the base pools. One can trivially think of a second and separate factory for each “deployer” parameter, which allows to deploy all pairs from scratch. Interestingly, it has to be noted that whitelisted addresses are meant to be compatible “IAlgebraPluginFactory” contracts.

Additionally, it exposes functions to fetch the deterministic addresses for base pools and custom pools.

The following changes were identified (https://www.diffchecker.com/gG0D3Oqb/):

a) customPoolByPair: This new mapping was introduced, which serves as additional storage for custom pools with reference to a distinct deployer address.
b) POOL_INIT_CODE_HASH: This variable has been adjusted to reflect the hash of the new bytecode for the AlgebraPool contract.
c) defaultConfigurationForPool: This function has been adjusted to remove the communityVault return value. This is compatible with the fact that the communityVault is a default value for all pools.
d) Deployment of pools: This call-flow has been refactored to allow for deploying custom pools, which are referenced to a specific deployer address.

Issue_01	Governance Privilege: Deployment of custom plugins
Severity	**Governance**
Description	In the previous version, it was clear that the plugin which is initially connected to a pair is the AlgebraBasePluginV1. The source code of this contract is public and therefore risks can be assessed. For custom pools however, any plugin with any arbitrary code can be deployed and added to the pool, which increases DoS risk due to custom hook execution.
Recommendations	Consider ensuring that only trusted addresses will have the “CUSTOM_POOL_DEPLOYER” role.
Comments / Resolution	Acknowledged.

Issue_02	Distinct deployments for a “deployer” parameter can be frontrunned
Severity	**Medium**
Description	The createCustomPool function allows any address with the “CUSTOM_POOL_DEPLOYER” role to deploy a custom pool. This custom pool is mainly distinct by two facts: a) “deployer” parameter b) connected plugin Furthermore, there is no limitation on the “deployer” parameter, this means any privileged address can deploy a pool which is connected to this parameter. This paths the way for the following issue: a) Alice wants to deploy a custom pool for the deployer = 0x1111....111, this deployer is seen as the “main” custom reference b) Alice wants to deploy this pool with a very specific plugin. c) Bob, who is also privileged, frontruns Alice’s deployment with his own plugin idea, effectively preventing Alice from connecting her plugin to a custom pool with the “main” custom reference.
Recommendations	We suggest using a special CustomPoolEntryPoint that allows to create pools only from deployer contract specified in createCustomPool params, which will make the case described in this issue impossible.
Comments / Resolution	Acknowledged, the client indicated that the recommendation will be implemented. It is necessary for users to verify this on the corresponding on-chain implementation, as this is not part of the audit scope. This can be deemed as fixed if the implementation is correct.

Issue_03	Deployer parameter for createVaultForPool is incorrect
Severity	**Low**
Description	Whenever a new pool is deployed, the createVaultForPool function within the AlgebraVaultFactoryStub is invoked with the following parameters: pool, creator, address(0), token0, token1 The third parameter is likely the “deployer” parameter, which is incorrect for the custom pool deployment scenario. While this will not have an impact since the vault factory is just a stub, it might result in future issues.
Recommendations	Consider using the correct deployer parameter.
Comments / Resolution	Resolved.

Issue_04	Lack of ReentrancyGuard for createCustomPool and createPool
Severity	**Low**
Description	Upon the createCustomPool function and later _createPool function, a hook is invoked to the msg.sender of the call. In any instance, this allows for completely arbitrary execution during this hook, including reentering into the createPool and createCustomPool functions. While we could not directly identify a specific exploit scenario, we are still of the opinion that it is beneficial to limit such a flexibility.
Recommendations	Consider implementing the nonReentrant modifier for the said functions.

Comments / Resolution	Resolved.

AlgebraPoolDeployer

The AlgebraPoolDeployer contract is a helper contract for the AlgebraFactory and takes care of the create2 (under the hood) deployment for the corresponding pool. The salt is determined as follows:

a) Base Pools: keccak256 of token0, token1
b) Custom Pools: keccak256 of deployer, token0, token1

Upon the deployment flow, it tightly packs token0, token1 and plugin addresses into two storage slots, which are then fetched during the AlgebraPool constructor and then reset after the deployment process.

The following changes were identified (https://www.diffchecker.com/iCbc17J6/):

a) deploy: This function was refactored to incorporate the salt for custom pools: {deployer, token0, token1}

No issues found.

AlgebraVaultFactoryStub

The AlgebraVaultFactoryStub contract is just a simple stub implementation, which is invoked upon the pool creation. Usually, the VaultFactory is meant to deploy a new community vault for each pool. However, this feature will be implemented in the future and within the meantime, the stub contract simply returns the default AlgebraCommunityVault.

The following changes were identified (https://www.diffchecker.com/AKQxspCP/):

a) createVaultForPool: The function signature has been changed to ensure compatibility with the new deployment process.

No issues found.

Core

AlgebraPool

The AlgebraPool contract is the main contract of the Algebra Core. It enables permissionless swaps between two tokens, while users can add and remove liquidity in a specific range following the UniV3 style. To incentivize liquidity provision a swap fee is taken which is then distributed among the participants in the current active liquidity range.

Additionally, the AlgebraPool incorporates hook logic, which is executed before and after interactions.

The following changes were identified (https://www.diffchecker.com/ZG87XBkh/):

a) initialize: This function was smoothed out, removing the duplicate communityFee setting and outsourcing the communityVault setting.
b) swap / swapWithPaymentInAdvanced: Both events were merged into one event, exposing a _emitSwapEvent function.
c) sync/skim: These functions were implemented to force match the reserves / withdraw excess tokens from the AlgebraPool. They are permissioned and can only be called by the Plugin

Issue_05	Governance Risk: Deployers may skim funds
Severity	**Governance**
Description	Currently, governance of this contract has several privileges for invoking certain functions that can drastically alter the contracts behavior. The following risks were identified: a) The already known risk: Privileged addresses can change the plugin address. b) The newly introduced risk: A malicious plugin can skim idle tokens from the pool contract. For example, whenever a flashloan is taken, not all fees are incorporated into the reserves: https://github.com/cryptoalgebra/Algebra/blob/45c65f0687b59910b28f7fb8fc59f7caae22e958/src/core/contracts/AlgebraPool.sol#L376 This means that after a flashloan, there will always be a mismatch between the actual balance and the pool's reserves. This is a valid state for skimming funds (either business-logic wise or malicious).
Recommendations	Consider ensuring that no malicious plugins can/will be connected.
Comments / Resolution	Acknowledged.

Issue_06	Reserves/fees are not updated upon collect
Severity	**Low**
Description	*Disclaimer: This issue is not related to the differential audit. Bailsec however skimmed the codebase to fresh up familiarity with it and discovered the following issue:* Upon the collect function, users can claim the aggregated fees for their position. This allows for providing amount0Requested and amount1Requested as uint128.max, which then means to simply collect all fees from a position. The problem hereby is that the reserves and the pool state is not updated, which results in the collect function not distributing all fees which are rightfully allocated in this point. If for example a direct transfer has happened right before the collect function, these fees will not be incorporated during this collect call but only upon the next pair state change. The same applies for rebase tokens.
Recommendations	A potential solution would be to invoke this.burn with a zero amount to update the pool state. It has to be noted that this is rather a theoretical than a practical issue because usually, interactions will not happen directly with the pair but rather via the position manager, which then inherently updates the position state with the already mentioned recommendation: https://github.com/cryptoalgebra/Algebra/blob/45c65f0687b59910b28f7fb8fc59f7caae22e958/src/periphery/contracts/NonfungiblePositionManager.sol#L375 Given that the Algebra protocol is battle-tested, we are careful with sensitive implementation changes (specifically for the core) and therefore do not recommend any change. However, it might be beneficial to acknowledge this issue.

Comments / Resolution	Acknowledged.

AlgebraPoolBase

The AlgebraPoolBase contract incorporates the storage layout, view functions and internal setter functions as well as callback functions for the AlgebraPool contract. It is solely meant to be inherited and serves as an abstract contract.

The following changes were identified (https://www.diffchecker.com/5c1XWAHv/):

a) _getDefaultConfiguration: This function has been slightly adjusted and will now return standardized default values. The distinct difference is the fact that this version will have one default communityFeeVault vs. the previous version had a unique vault per pool.

No issues found.

ReservesManager

The ReservesManager is inherited by the AlgebraPool and takes care of the reserve accounting. Notably, it updates the reserves before any important interaction and adjusts the reserves whenever tokens are deposited / withdrawn / swapped within the pool.

The following changes were identified (https://www.diffchecker.com/80d0Fh5M/):

a) ExcessTokens: This event is emitted upon _updateReserves and returns the amount of excess tokens that were incorporated into the reserves.
b) _skimReserves: This internal function handles the skimming of excess reserves.

Farming

FarmingCenter

The FarmingCenter is the farming contract for positions which are created via the NonfungiblePositionManager, it serves as entry/exit point for the Farming module.

The Farming module in general allows users to receive rewards on their positions and mimics the pool behavior to distribute rewards fairly in the same manner as swap fees, incentivizing users to stay in the current liquidity range.

The following changes were identified (https://www.diffchecker.com/R2R2gNRQ/):

a) Update of NATSPEC.
b) _checkParamsForVirtualPoolToogle: This function has been updated to view-only.

Issue_07	FarmingCenter is not compatible with custom pools
Severity	**High**
Description	*Disclaimer: This issue is not related to the differential audit. Bailsec however skimmed the codebase to fresh up familiarity with it and discovered the following issue:* Currently, the IncentiveMaker can call the createEternalFarming function within the AlgebraEternalFarming contract, to create a new incentive for liquidity providers. This then invokes FarmingCenter.connectVirtualPoolToPlugin, which then maps the new virtualPool to a corresponding pool address. The problem hereby is that the following check is present:

	require(address(pool) == PoolAddress.computeAddress(algebraPoolDeployer, PoolAddress.PoolKey(pool.token0(), pool.token1())), 'Invalid pool'); which essentially only allows connecting a virtual pool to a base pool only and no custom pool, thus rendering the Farming module completely unusable for custom pools.
Recommendations	It seems that the Farming module is not yet completely ready to support custom pools. This might be intended because development is not yet finalized. Nevertheless, we recommend refactoring the whole Farming module to support custom pools. This will be subject to a separate audit.
Comments / Resolution	Acknowledged. A new farming module will be developed.

Periphery

PoolAddress

The PoolAddress library is used throughout the architecture to deterministically fetch the pool address for the corresponding token0/token1

The following changes were identified (https://www.diffchecker.com/YIHXWvYs/):

a) Change of POOL_INIT_CODE_HASH to reflect the adjusted AlgebraPool contract.

Issue_08	Salt will not work for custom pools
Severity	**High**
Description	As per create2 docs, the address determination is calculated as follows: address = keccak256(0xff + sender_address + salt + keccak256(initialisation_code)) The current implementation is as follows: keccak256(abi.encodePacked(hex'ff', poolDeployer, keccak256(abi.encode(key.token0, key.token1)), POOL_INIT_CODE_HASH)) The salt is only compatible for base pools and will not work for custom pools. This means that the NonfungiblePositionManager/SwapRouter and all its inherited contracts and dependencies are not compatible with custom pool creation and management.
Recommendations	Consider implementing a function which allows the deterministic fetching of custom pools, using the below displayed, correct salt: _encodedParams = abi.encode(deployer, token0, token1) salt: keccak256(_encodedParams) This also means that the NonfungiblePositionManager/SwapRouter

	must be adjusted to be able to handle the distinct difference between base and custom pools. A more convenient option would be to deploy a new *NonfungiblePositionManagerCustom* and SwapRouterCustom for each deployer, as this would greatly reduce security risk of commingling different custom pools. An additional audit for such a component is mandatory.
Comments / Resolution	Acknowledged, a new farming module and periphery will be developed.

Plugin

BasePluginV1Factory

The BasePluginV1Factory is responsible for deploying the base plugin (volatilityOracle) and linking it to a pool. It furthermore exposes configurational functionalities for the AlgebraBasePluginV1 contract.

The following changes were identified (https://www.diffchecker.com/wPaaZZo1/):

a) The createPlugin function has been renamed to beforeCreatePoolHook, to ensure compatibility with the adjusted _createPool function in the AlgebraPoolFactory contract.

Issue_09	createPluginForExistingPool is not compatible with custom pools
Severity	**Low**
Description	*Disclaimer: This issue is not related to the differential audit. Bailsec however skimmed the codebase to fresh up familiarity with it and discovered the following issue:* The createPluginForExistingPool function is used whenever pools are deployed without a corresponding plugin. While this will happen more often for base pools due to the following condition check: https://github.com/cryptoalgebra/Algebra/blob/45c65f0687b59910b28f7fb8fc59f7caae22e958/src/core/contracts/AlgebraFactory.sol#L125 it is not guaranteed that it will never happen for custom pools (we acknowledge that the beforeCreatePoolHook is inherently called for custom pool deployments). However, if such a scenario would occur and the BasePluginV1Factory contract is meant to be used for custom pools, this will not work. This issue is only rated as low severity because with high likelihood, a totally different PluginFactory will be used for custom pools. We can however not say this for certain.
Recommendations	Consider either adjusting the BasePluginV1Factory or simply acknowledge that this factory will solely be used for base pools.
Comments / Resolution	Acknowledged.