BAILSEC.IO

OFFICE@BAILSEC.IO

X: @BAILSECURITY

TG: @HELLOATBAILSEC

FINAL REPORT

XTrade

Update Audit (differential)

October 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	XTrade - Update Audit (differential)
Website	---
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/GammaStrategies/Algebra-Integral-Gamma/tree/d93e2461c0c8f38b9f418eb37d6c2477ed640b98/src/plugin
Resolution 1	https://github.com/GammaStrategies/Algebra-Integral-Gamma/tree/91920ff157de063bef1149f0ff4c9ac81b5f48cb/src/plugin

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1		
Medium				
Low				
Informational	4	4		
Governance				
Total	5	5		

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 XTrade compared to Algebra Integral V1:

Old Commit:

https://github.com/cryptoalgebra/Algebra/tree/v1.0-integral/src/plugin

New Commit:

https://github.com/GammaStrategies/Algebra-Integral-Gamma/tree/d93e2461c0c8f38b9f418eb37d6c2477ed640b98/src/plugin

Files in Scope:

1. AlgebraBasePluginV1.sol (https://www.diffchecker.com/5LTuz8DF/)
2. BasePluginV1Factory.sol (https://www.diffchecker.com/EzvQjo9h/)

Primary Update Overview:

By default, Algebra incorporates a dynamic fee which changes based on different triggers and underlies a sophisticated mathematical concept. In this differential audit, all components which are corresponding to the dynamic fee module are being removed and instead a binary fee module has been implemented which allows governance for setting a fee for X -> Y swaps and another fee for Y -> X swaps. The core swap concept has not been modified, which means the way how the fee is applied remains unchanged, the only change is the percentual fee value which is applied in the swap logic.

Furthermore, it is important to mention that the fee computation is handled off-chain which means that governance has the freedom to set the fee freely.

Security Considerations:
Since the main change is the removal of the dynamic fee and the introduction of a bidirectional, binary fee with a setter mechanism, the security considerations to this change are quite limited. The following points could introduce a risk:

a) Removal of dynamic fee logic: If the contract relies on the usage of the dynamic fee logic in some other places, this can result in some issues where other functionalities can be compromised.

b) Governance Freedom: Since the fee can be freely set by governance (up to 6%) this can result in some unexpected slippage scenarios.

c) Implementation of new fee: While the implementation of the new fee mechanism is rather trivial than complicated, that does not mean it is free of bugs. We will carefully examine the setter logic and its usage within the swap flow.

d) View-only limitations due to binary fee: The binary fee introduction means we have two fees instead of one fee. Algebra's globalState only reflects one fee. This will introduce limitations in the view-only functionality.

It is positive to mention that these changes only occur on the plugin and do not manipulate the Algebra core.

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.

AlgebraBasePluginV1

The AlgebraBasePluginV1 contract is a plugin for Algebra which implements bidirectional fees with dynamic adjustments by governance. It allows for separate fee configurations for zeroToOne and vice-versa token swaps, enabling more flexible fee structures based on the direction of the trade.

The contract also maintains a volatility oracle by storing timepoints, which are used to calculate historical price and volatility data essential for dynamic fee calculations.

Additionally, it manages incentives for liquidity providers by interacting with virtual pools and updating the pool's configuration during various events like swaps, position modifications, and flashes.

The following changes have been made:

a) AdaptiveFee.sol import removed
b) AlgebraFeeConfigurationU144.sol import removed.
c) AlgebraFeeConfigurationU144 private _feeConfig removed
d) uint16 public feeZeroToOne storage variable added
e) uint16 public feeOneToZero storage variable added
f) FeesChanged event added
g) Fees are initialized with 3000 (0.3%)
h) feeConfig function was removed
i) setFees function was introduced which allows governance to set the fee up to 6%
j) getCurrentFee function was adjusted which serves as view-only function to return X -> > fee
k) _calculateFee function was added which returns X -> Y or Y -> X fee depending on the swap direction
l) changeFeeConfiguration function was removed
m) _getFeeAtLastTimepoint function was removed
n) beforeSwap function was adjusted to incorporate the swapping direction
o) _writeTimepointAndUpdateFee function was adjusted to accommodate the fee update based on the swapping direction

Key Differences:

Fee Mechanism Implementation:

Algebra:

- Implements an **adaptive fee mechanism** based on market volatility using a complex configuration of parameters (alpha1, alpha2, beta1, beta2, gamma1, gamma2, baseFee).

XTrade:

- Implements **bidirectional static fees** with separate fees for zero-to-one (feeZeroToOne) and one-to-zero (feeOneToZero) swaps.

Fee Configuration Variables:

Algebra:

- Uses a packed AlgebraFeeConfigurationU144 struct (_feeConfig) to store adaptive fee parameters.

XTrade:

- Uses two public uint16 variables (feeZeroToOne, feeOneToZero) for fees.

Fee Calculation Functions:

Algebra:

- Uses getCurrentFee() to calculate the fee dynamically based on average volatility.
- Has _getFeeAtLastTimepoint() for fee calculation at the last timepoint.

XTrade:

- getCurrentFee() simply returns feeZeroToOne.
- Uses _calculateFee(bool zeroToOne) to determine the fee based on swap direction.

Adaptive Fee Mechanism:

Algebra:

- Relies on the AdaptiveFee library and VolatilityOracle to adjust fees dynamically.

XTrade:

- Does not use adaptive fees; fees are static per direction.

Imports and Libraries:

Algebra:

- Imports ./libraries/AdaptiveFee.sol.
- Uses AlgebraFeeConfigurationU144.sol.

XTrade:

- Does not import AdaptiveFee or AlgebraFeeConfigurationU144.
- Simplifies imports to only necessary components.

Constructor Differences:

Algebra:

- Does not set initial fee values; relies on adaptive fee configuration.

XTrade:

- Initializes feeZeroToOne and feeOneToZero to 3000 (0.3%).

Implementation of _writeTimepointAndUpdateFee:

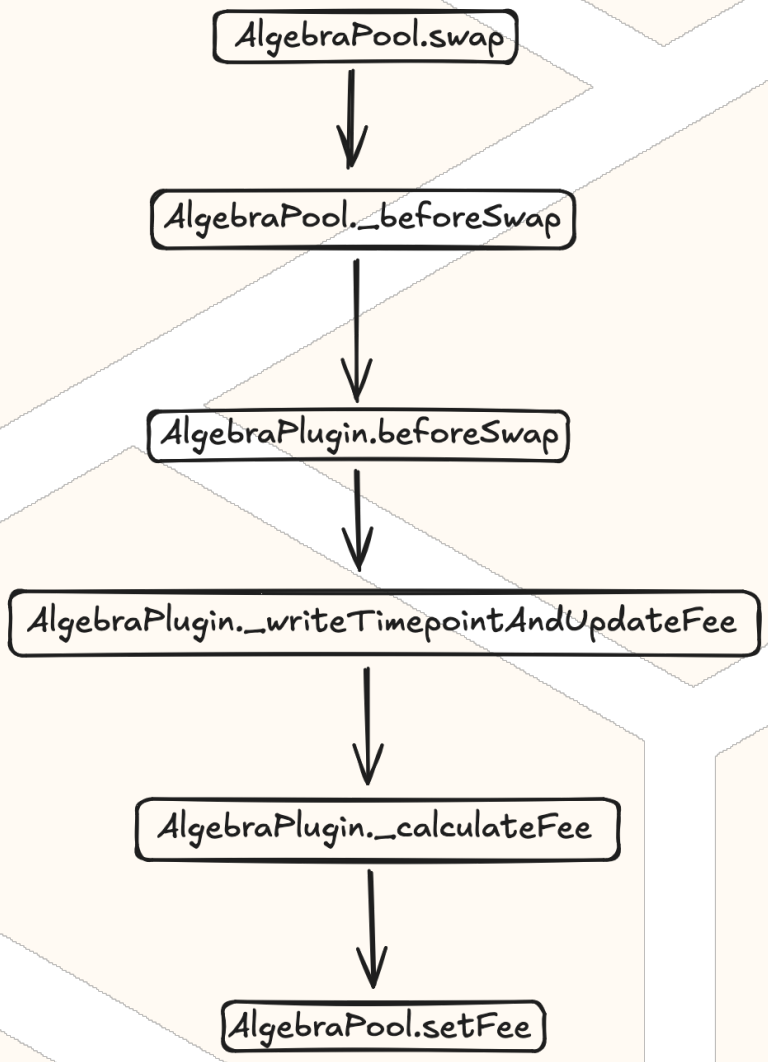
Algebra:

- Calculates the new fee based on volatility averages and adaptive parameters.

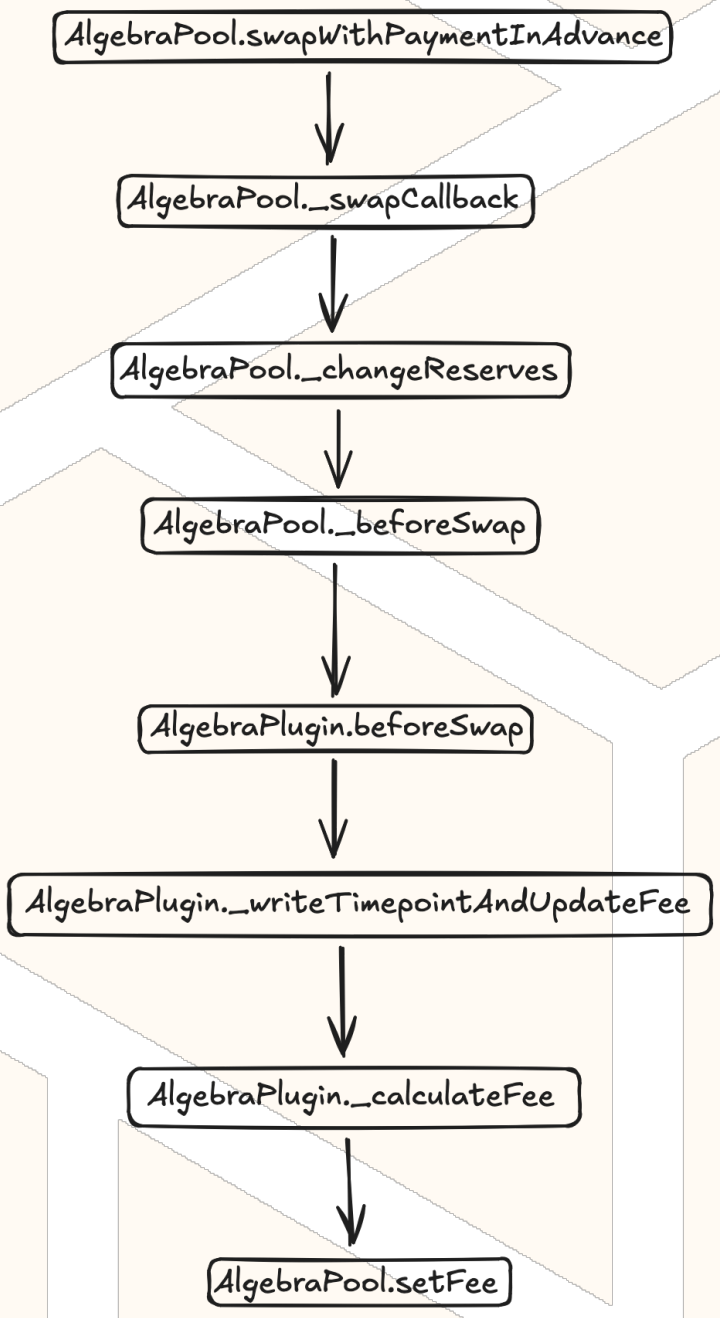
XTrade:

- Calculates the new fee based on the swap direction and the static fees.

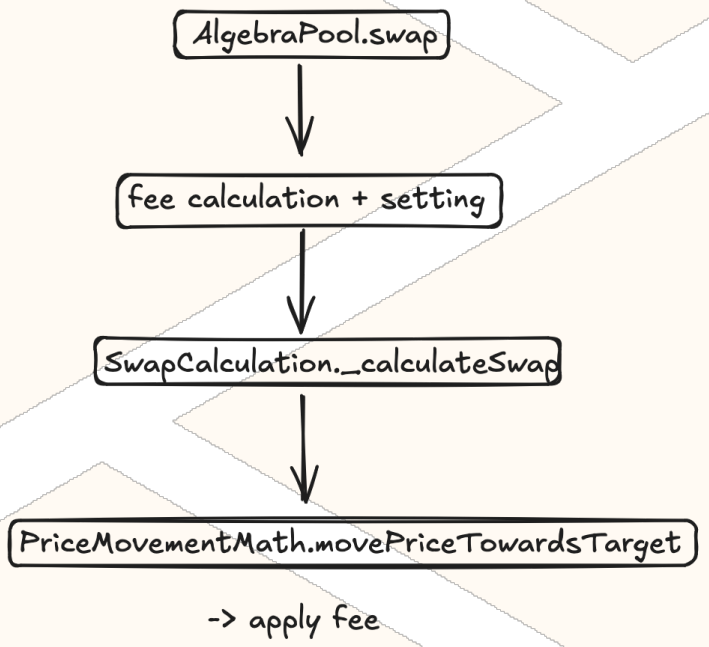
The fee determination process is as follows (swap):



The fee determination process is as follows (swapWithPaymentInAdvance):



The fee application process is as follows:



Issue_01	Early return within _writeTimepointAndUpdateFee makes bidirectional approach void
Severity	**High**
Description	The _writeTimepointAndUpdateFee function returns early in the scenario where an update has happened in the same block: *if (_lastTimepointTimestamp == currentTimestamp) return;* This means that the first swap within a block determines the swap fee for the whole block because any subsequent swaps in the same block will not update the fee, even if the direction is different to the first swap. PoC: Status Quo: feeZeroToOne = 60000 (6%) feeOneToZero = 0 (0%) 1. Alice swaps 10e18 tokenX to tokenY. This is the first swap within block 10_000 and the fee will be automatically set to 60000 (6%) 2. Bob swaps 10e18 tokenY to tokenX within block 10_000. Bob should not pay any fee. However, due to the early return within _writeTimepointAndUpdateFee, the fee will not be changed and Bob pays 6% fee instead of 0%.
Recommendations	A change to this logic modifies the Plugin inherently, therefore, it must be carefully tested. The following approach should be applied: In the scenario where *if (_lastTimepointTimestamp == currentTimestamp)*

	is reached, consider invoking _calculateFee and AlgebraPool.setFee once the fee is different. The following invariants must be ensured after the fix: Invariant 1: timepoints should always be updated only once per block Invariant 2: The fee should be updated whenever it is different from the previous fee
Comments / Resolution	Resolved, the _writeTimepointAndUpdateFee function has been refactored to update the fee in every iteration and timepoints are still only written once per block.

Issue_02	Sudden fee increase can result in unexpected loss during swaps
Severity	**Informational**
Description	If users execute swaps with a loose slippage configuration, a sudden increase of the fee via the setFees function can result in unexpected losses during swaps. Furthermore, if the pair is directly invoked for the swap, without a router, the only slippage check will be limitSqrtPrice, which does not include the fee.
Recommendations	Consider openly communicating that fees will be subject to change based on the determined off-chain logic.
Comments / Resolution	Acknowledged.

Issue_03	Fee change can be frontran
Severity	**Informational**
Description	The setFees function allows governance to change to set new bidirectional fees. If a new fee is meant to be applied, users can frontrun the fee change and still execute a swap using the old fee.
Recommendations	This is part of the setter design and can simply be acknowledged.
Comments / Resolution	Acknowledged.

Issue_04	getCurrentFee only returns feeZeroToOne
Severity	**Informational**
Description	The getCurrentFee function is responsible for returning the current fee. In the previous iteration it simply returned the current adapted fee, while in this iteration it returns the fee for X -> Y swaps. This limitation may confuse users by assuming this fee counts for both directions.
Recommendations	Usually we would recommend simply returning two fees. However, since that would violate Algebra's architecture, we tend to stay away from this recommendation to ensure no accidental side-effects are being introduced. We simply recommend making this clear on the frontend.
Comments / Resolution	Acknowledged.

BasePluginV1Factory

The AlgebraBasePluginV1Factory contract is a factory that creates instances of the AlgebraBasePluginV1 plugin for liquidity pools. It enables the deployment of default plugins for both new and existing pools, facilitating features like dynamic fees and volatility management and in this scope bidirectional fees.
The contract keeps track of plugins associated with each pool through the pluginByPool mapping and allows for administrative control over important parameters like the farmingAddress used for liquidity incentives.
By integrating with the AlgebraFactory, it ensures that only authorized administrators can create plugins and modify configurations.

The following changes have been made:

a) changeFeeConfiguration call to AlgebraBasePluginV1 has been removed. This call was only necessary to set up the dynamic fee module

b) Logic for defaultFeeConfiguration storage variable has been removed

Issue_05	Deployed address still says volatilityOracle
Severity	**Informational**
Description	The _createPlugin function deploys the AlgebraPlugin and links it to the corresponding pool. The deployed address is currently still called volatilityOracle: *IAlgebraBasePluginV1 volatilityOracle = new AlgebraBasePluginV1(pool, algebraFactory, address(this));*
Recommendations	In an effort to not modify a battle-tested codebase, we do simply recommend acknowledging this issue.
Comments / Resolution	Acknowledged.