MixBytes()

Algebra Upgradeable Plugins Security Audit Report

JANUARY 12, 2026

Table of Contents

1. Introduction

1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

The Algebra Protocol plugins system is a modular architecture for managing liquidity pools with advanced features including dynamic fee calculation, volatility oracles, automated liquidity management (ALM), security controls, and farming incentives. The system has been upgraded from a non-upgradeable architecture to an upgradeable Beacon Proxy pattern, allowing for unified management and updates of plugin implementations across all pools while maintaining per-pool state isolation.

This audit focused on the upgraded Beacon Proxy architecture and the complete implementation of the `DynamicFee` module, including its business logic, mathematical calculations, and state management. The following components and their associated business logic were outside the audit scope: `VolatilityOracleConnector.sol` and `VolatilityOraclePluginImplementation.sol`, `AlmConnector.sol` and `AlmPluginImplementation.sol`, `SecurityConnector.sol` and `SecurityPluginImplementation.sol`, and `FarmingProxyConnector.sol` and `FarmingProxyPluginImplementation.sol`. While `AlgebraUpgradeablePlugin.sol` inherits from these connectors and performs delegatecall operations to their implementations, the business logic, mathematical calculations, and state management of these non-scoped modules were not subject to security review.

This security audit was conducted over a period of 3 days by a team of 3 security auditors. The review process included thorough manual code analysis, architectural review, and automated static analysis tools to identify potential vulnerabilities, design flaws, and security risks in the upgradeable plugin system.

As part of our security review, we examined standard attack vectors and our internal security checklist, performing detailed verification of the following components:

- **Access Control Mechanisms.** We verified that all privileged operations are properly protected through role-based access control (`ALGEBRA_BASE_PLUGIN_MANAGER` role for fee operations, `onlyAdministrator` for beacon upgrades), modifier-based protection (`onlyPool` for all plugin hooks, `onlyPluginFactory` for initialization), and sender verification (factory hooks check `msg.sender == algebraFactory`).

- **Single Plugin Per Pool Enforcement.** The factory can create plugins for pools through `beforeCreatePoolHook()` and `createPluginForExistingPool()`. We verified that the factory cannot create more than one plugin for the same pool, as this is enforced by the `s.pluginByPool[pool] != address(0)` check in `_createPlugin()`.

- **Delegatecall Safety in Connectors.** Connectors use `_delegateCall()` to execute logic in module implementations. We verified that delegatecalls are performed to the correct implementation addresses (set immutably in constructors) and that the mechanism correctly shares the proxy's storage context with module implementations.

- **Pool Address Reading from Proxy Bytecode.** The `_getPool()` function reads the pool address from the proxy's runtime bytecode using `extcodecopy` at a hardcoded offset. We verified that this mechanism retrieves the immutable `pool` address from `AlgebraPluginProxy` for the audited build configuration; however, since this is a hardcoded bytecode offset, it should be treated as build-sensitive and validated if compiler settings change.

- **Storage Layout Isolation via ERC-7201.** The plugin system uses ERC-7201-style namespaced storage. We verified that all modules use unique namespace strings and that storage layouts are isolated, preventing state corruption between modules during upgrades. However, the implementation does not fully comply with the ERC-7201 standard formula (as defined in `EIP-7201`). A detailed analysis of this non-compliance is documented in finding L-2.

- **AdaptiveFee Mathematical Correctness.** We verified the `AdaptiveFee` library's mathematical implementation, including the sigmoid function calculations, exponential series approximations, and fee computation logic. We confirmed that the library correctly executes its functions across various volatility ranges and that the final fees are bounded above by `baseFee + alpha1 + alpha2`, with this sum validated to be within `type(uint16).max`.

Overall, the architectural upgrade from a non-upgradeable to an upgradeable Beacon Proxy pattern has been implemented correctly. The modular design enables secure and reliable operation of the upgradeable plugin system. No critical or high-severity vulnerabilities were identified during the audit. Several code quality improvements that can be adopted are documented in the **Findings Report** section.

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.

1.3 Project Overview

Summary

Title	Description
Client	Algebra
Category	DEX
Project	Dynamic Fee Plugin

Title	Description
Type	Solidity
Platform	EVM
Timeline	24.12.2025 - 09.01.2026

Scope of Audit

File	Link
abstract-plugin/contracts/ BaseConnector.sol	BaseConnector.sol
abstract-plugin/contracts/ UpgradeableAbstractPlugin.sol	UpgradeableAbstractPlugin.sol
abstract-plugin/contracts/ AlgebraPluginProxy.sol	AlgebraPluginProxy.sol
default-plugin/contracts/ AlgebraUpgradeablePlugin.sol	AlgebraUpgradeablePlugin.sol
default-plugin/contracts/ AlgebraUpgradeablePluginFactory.sol	AlgebraUpgradeablePluginFactory.sol
dynamic-fee/contracts/ DynamicFeeConnector.sol	DynamicFeeConnector.sol
dynamic-fee/contracts/ DynamicFeePluginImplementation.sol	DynamicFeePluginImplementation.sol
dynamic-fee/contracts/libraries/ AdaptiveFee.sol	AdaptiveFee.sol
dynamic-fee/contracts/libraries/ DynamicFeeStorage.sol	DynamicFeeStorage.sol
dynamic-fee/contracts/types/ AlgebraFeeConfiguration.sol	AlgebraFeeConfiguration.sol
dynamic-fee/contracts/types/ AlgebraFeeConfigurationU144.sol	AlgebraFeeConfigurationU144.sol

Versions Log

Date	Commit Hash	Note
24.12.2025	bcb19fdd59f88e5f592b76f48f87bb864e8ad7c3	Initial Commit
08.01.2026	e00ec6a932acadbfd11210a6abfdc7ee04245c23	Commit for the reaudit

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	0
Medium	0
Low	2

Findings Statuses

ID	Finding	Severity	Status
L-1	Factory Initialization Can Be Front-Run If Proxy Deployment Is Not Atomic	Low	Fixed
L-2	Storage Location Calculation Not ERC-7201 Compliant	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Factory Initialization Can Be Front-Run If Proxy Deployment Is Not Atomic		
Severity	Low	**Status**	Fixed in e00ec6a9

Description

The AlgebraUpgradeablePluginFactory.initialize() function can be front-run by an attacker if the TransparentUpgradeableProxy is deployed without atomic initialization.

All plugins created for new pools will execute malicious code, as the attacker controls the logic of all plugin proxies via the compromised beacon. However, the admin can call upgradePlugins() to fix the beacon implementation, which immediately updates ALL existing and future plugin proxies since they all reference the same beacon. After the upgrade, the malicious implementation remains deployed but is no longer used by any proxies. The fee configuration can be corrected per-proxy via changeFeeConfiguration() after the upgrade. For impact to occur, at least one pool must be created and users must interact with it before the admin notices and upgrades.

Attack Scenario:

1. The deployer deploys TransparentUpgradeableProxy with empty initialization data ('0x')
2. Before the deployer calls initialize(), an attacker front-runs the transaction
3. The attacker calls initialize() with:
   - A valid _algebraFactory address (to pass future checks)
   - A malicious pluginImplementation address (their own contract)
   - Any initialFeeConfig (can be corrected later via setDefaultFeeConfiguration())
4. The attacker's malicious implementation is set in the beacon
5. All future plugins created via beforeCreatePoolHook() will use the attacker's implementation

The deployment flow is demonstrably non-atomic in tests (TransparentUpgradeableProxy is deployed with empty init data and initialize() is called in a separate transaction), which creates a real front-running window. If front-run, an attacker can set a malicious

`pluginImplementation` in the beacon, altering the logic executed by plugin proxies (and therefore impacting core protocol behavior) until admin intervenes.

Recommendation

We recommend deploying the `TransparentUpgradeableProxy` with initialization calldata in the constructor to ensure atomic initialization. The proxy constructor should include the encoded `initialize()` call with all required parameters, preventing any window where the proxy exists in an uninitialized state.

Client's Commentary

We updated the deployment script so that the proxy is upgraded to the implementation and initialized in a single transaction. Fixed in e00ec6a932acadbfd11210a6abfdc7ee04245c23

L-2	Storage Location Calculation Not ERC-7201 Compliant		
Severity	Low	**Status**	Fixed in e00ec6a9

Description

Throughout the codebase, contracts and libraries that use namespaced storage (declared as ERC-7201-style) calculate storage slots using only `keccak256(...)` of the namespace string, rather than following the full ERC-7201 standard formula.

Current implementation (example from `AlgebraUpgradeablePluginFactory`):

```
bytes32 private constant STORAGE_LOCATION =
    keccak256('algebra.pluginfactory.storage');
```

ERC-7201 compliant formula should be:

```
bytes32 private constant STORAGE_LOCATION =
    bytes32(
        uint256(keccak256("erc7201:algebra.pluginfactory.storage")) - 1
    ) & ~bytes32(uint256(0xff));
```

The current implementation is missing:

1. The `erc7201:` prefix in the keccak256 input string
2. The `-1` subtraction operation
3. The `& ~bytes32(uint256(0xff))` mask operation that clears the last byte

Recommendation

We recommend updating all storage location calculations to follow the ERC-7201 standard formula.

Client's Commentary

Fixed in 87a2361ae8f3c02dbd07a50889c8b31911d28484

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

 https://mixbytes.io/

 https://github.com/mixbytes/audits_public

 hello@mixbytes.io

 https://x.com/mixbytes