BAILSEC.IO

OFFICE@BAILSEC.IO

X: @BAILSECURITY

TG: @HELLOATBAILSEC

FINAL REPORT:

Algebra Factory - Update Audit (differential)

June 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 Factory - Update Audit (differential)
Website	algebra.finance
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/cryptoalgebra/Algebra/blob/f330bffc1ac2b2c70c9a0f40e1f38e5ecca531f3/src/core/contracts/AlgebraFactory.sol
Resolution 1	

2. Detection Overview

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

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 Factory:

Old Commit:

https://github.com/cryptoalgebra/Algebra/blob/0fe96d86be5ae446901d3abb244b96be7600adeb/src/core/contracts/AlgebraFactory.sol

New Commit:

https://github.com/cryptoalgebra/Algebra/blob/f330bffc1ac2b2c70c9a0f40e1f38e5ecca531f3/src/core/contracts/AlgebraFactory.sol

Diffcheck to audit:

https://www.diffchecker.com/NC5xrOXX

Files in Scope:

1. AlgebraFactory.sol

Primary Update Overview:
The AlgebraFactory contract is responsible for deploying Pool contracts. Contrary to most Factories, Algebra implements a modular approach which means there are two types of pools:

a) **Default Pools:** These pools are the main pools which are used throughout the Algebra ecosystem, anyone can create these pools and they will always point to the defaultPluginFactory.

b) **Custom Pools:** These pools are additional to the default pools and they co-exist in the Algebra ecosystem. Privileged users can deploy these pools and they always point to a customPluginFactory. The main idea behind this concept is to support pools with new and sometimes even experimental plugins.

The only change which has been made compared to the previous version is the addition of a "**afterCreatePoolHook**" call to the plugin factory. For the default pool deployment this will just be a call to the defaultPluginFactory, whereas for the custom pool deployment this will be a call to the customPoolFactory (msg.sender).

Security Considerations:

For the default pool creation, there is absolutely no risk corresponding to that change, simply due to the fact that the defaultPluginFactory is predetermined and trusted.
In such a scenario where the defaultPluginFactory is untrusted, it may be possible to re enter upon the beforeCreatePoolHook, invoke the setDefaultPluginFactory function and then benefit from a afterCreatePoolHook call to a different factory. Or invoke the initialize function on the pair. However, this issue is negligible.

For the custom pool creation, there are two potential risks:

a) Risk of a reentrancy call: This risk is already mitigated due to the fact that a nonReentrant modifier is added on the entry point (createCustomPool and createPool). Theoretically it is possible to invoke the "initialize" function on the just newly deployed pair, however, this should not expose any problem. The general rule is that custom plugin factories should be audited anyways to avoid any problems.

b) Risk of function call revert: A revert can happen if the afterCreatePoolHook selector does not exist in the customPluginFactory or if the provided parameters do not match. We recommend thoroughly auditing any customPluginFactory implementations which are meant to be used in the future.

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.