Gelato G-UNI Security Audit

July 15, 2021

WatchPug

Table of Contents

Summary

This report has been prepared for Gelato's G-UNI v1 smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The scope of the audit were the 2 smart contracts: GUniPool.sol and GUniFactory.sol

The GUniFactory contract is deployed and managed by Gelato. New GUniPools can be created by anyone via the GUniFactory contract.

Every GUniPool contract allows users to deposit the underlying tokens to mint GUNI tokens. Subsequently, GUNI token holders can redeem (burn) them in exchange for the underlying ERC20s that the tokens represent.

No issues were found in the GUniFactory contract. 1 medium, 2 minor, and 1 informational issue were found in the GUniPool contract, all of which got fixed except one minor issue that Gelato team acknowledged but deemed not necessary to fix as it is not security related.

Overall, we think the code is clear and well-documented. We are pleased to see the use of small, encapsulated functions and isolated contracts. The repository contains a fair number of tests that increase the confidence in the codebase and functionality correctness.

Overview

Project Summary

Project Name	**Gelato G-UNI**
Codebase	**https://github.com/gelatodigital/g-uni-v1-core/**
Commit	**c9a0a46bc8f3f203f195f33fe843969fdad63441**
Language	**Solidity**
Platform	**Ethereum**

Audit Summary

Delivery Date	**Jul 15, 2021**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**4**

GU-01: Fees cannot be collected

Medium

GUniPool.sol#L579-L607

Issue Description

The burn function of the GUniPool contract charges a fee every time a user burns G-UNI tokens. However, in the _withdrawExact function, the adminFees is calculated and subtracted, but the logic does not collect or keep track of the added fees. Thus they will be lost.

Recommendation

Consider modifying the way in which fees are tracked or collected so that they can be effectively collected.

Resolution

This issue has been addressed with commit 6a35bb608c519211c25639bdfda86b6beceb41d3 by collecting admin fees in method GUniPool#burn().

Status

✓ **Fixed**

GU-02: Unneeded named return parameter

Minor

GUniPool.sol#L657-L675

Issue Description

In the `_swapAndDeposit` function, there are two unused and unneeded named return parameters (that actually instantiates a new variable in memory which is not used).

Plus, the `finalAmount0` and `finalAmount1` are not exactly the final amount after the `_swapAndDeposit` operation.

Recommendation

Consider removing unused named return parameters.

Status

ⓘ Acknowledged

GU-03: Casting between types without over/underflow checks

Minor

Issue Description

Multiple castings between different signed and unsigned integer sizes without check, in scenarios that may well be unlikely to happen, could result in an undesirable truncation leading to unexpected values.

Instances include: `GUniPool#_deposit()` `GUniPool#_swapAndDeposit()` `GUniPool#_checkSlippage()`

Recommendation

Consider using the SafeCast library, or check for over/under flows after casting.

Resolution

This issue has been addressed with commit 7273e3b6be145f44ac43c8c313d53c8df09d1514 by using SafeCast library for the casting operations.

Status

✓ Fixed

GU-04: Consider using the same abstract contract

Informational

Issue Description

Both `@openzeppelin/contracts/proxy/utils/Initializable.sol` and `@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol` (with the same code) are used.

Recommendation

For consistency, it’s recommended to use a single fixed version throughout all the contracts.

Status

✓ Fixed

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of Confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with WatchPug. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by WatchPug.

Disclaimer