OpenZeppelin | security

Gauntlet Aera Vault LlamaPay Oracle Audit

May 15, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-04-08 To 2024-04-23
Languages	Solidity

Total Issues	2 (0 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	1 (0 resolved)
Notes & Additional Information	1 (0 resolved, 1 partially resolved)

Scope

We audited the Gauntlet-xyz/aera-contracts-v2 repository at commit 88fbc9a.

In scope were the following files:

```
aera-contracts-v2/
└── periphery
    ├── interfaces
    │   ├── ILlamaPayRouterOracle.sol
    │   ├── IExecutor.sol
    │   ├── IAeraV2Oracle.sol
    │   └── ILlamaPayRouterOracleTypes.sol
    ├── LlamaPayRouterOracle.sol
    ├── Executor.sol
    └── AbstractAssetOracle.sol
```

System Overview

The Aera vault system is a collection of contracts which provides asset management functionality to a protocol's treasury. It consists of a set of core contracts, which center around a vault contract holding assets, and periphery contracts, which add functionality to the vault. The core contracts notably include `AeraVaultV2.sol`, which holds assets, `AeraVaultHooks.sol`, which acts as a set of checks and balances as to which actions can be performed on the assets, and `AeraVaultAssetRegistry.sol`, which holds the list of assets that the vault is managing.

Over the course of this audit, we were auditing the periphery contract `LlamaPayRouterOracle.sol`. This contract allows the vault to manage external LlamaPay streams in order for the vault to be able to pay third parties in a continuous fashion. To that end, the `LlamaPayRouterOracle` contract acts as a wrapper for up to ten LlamaPay instances, allowing the creation, modification, pausing and cancellation of streams. To allow the vault to ascertain the value locked in these streams, the `LlamaPayRouterOracle` contract additionally acts both as an ERC-20 token and its own price oracle, whose `totalSupply` is fixed at 1 and whose price reflects the total holdings of the managed LlamaPay instances. Notably, the contract adds an `endDate` to streams, which is not part of the LlamaPay contract.

The Aera vault system foresees two main actors, those being the vault owner and a "guardian". The vault owner is the treasury, who can uniquely create streams, and the guardian is a trust-minimized third party, which can generally perform white-listed actions on the vault's assets, and in the context of this audit specifically, is allowed to deposit assets in the `LlamaPayRouterOracle` contract.

Security Model and Privileged Roles

The vault owner is able to perform arbitrary actions on the `LlamaPayRouterOracle` contract through the inherited `Executor.sol` contract. The guardian is limited to actions

white-listed by the vault owner and must generally be assumed to be adversarial, although in practice, it is assumed to act in the vault owner's best interest.

Low Severity

L-01 Potential MEV Opportunities for Payers in Debt

The `LlamaPay` contracts provide an option for payers who are in debt to partially erase unpaid amounts by cancelling their payment streams. Consider this example:

- A payer has 900 USDC in their account.
- They create a stream to transfer 100 USDC per second to a payee.
- After 20 seconds, the payee is theoretically entitled to 2000 USDC.
- If the payer cancels the stream, the payee receives only 900 USDC, while the remaining debt is effectively erased by the protocol.

This behavior, in conjunction with the `cancelExpiredStream` function in the `LlamaPayRouterOracle`, creates a scenario where streams that are cancelled later can withdraw more funds than earlier ones. Imagine the following scenario:

- A payer sets up two streams to payee1 and payee2 with 900 USDC in their account, with each stream designed to transfer 100 USDC per second.
- After 10 seconds, both streams are cancelled.
- The first cancelled stream secures 400 USDC since floor(900/200) = 4, and 4 * 200 = 800 is divided between the two streams.
- The second cancelled stream then takes the remaining 100 USDC.

However, the impact of this issue is minimal as long as the `amountPerSec` is low. Currently, the MEV opportunities that arise favor block builders and are not distributed among all participants. Consider addressing the MEV issue either by altering the method of stream cancellations or by imposing a cap on `amountPerSec` to mitigate the problem.

Update: *Acknowledged, not resolved. The Gauntlet team stated:*

> *For treasuries that use a multisig and are less trusted by contributors we will propose amountPerSec caps for each token to give an extra level of trust.*

Notes & Additional Information

N-01 Vault Value Can Go Below Daily Lower Bound

Since a withdrawal action by the beneficiary of a LlamaPay instance does not affect the daily multiplier of a vault, the value locked within a vault over any 24-hour period can reach a lower value than would be expected, given the `minDailyValue`.

Assume the following scenario:

- The vault holds `X` amount of value and has a `minDailyValue` multiplier of `0.9`.
- There is `Y = 0.03 * X` amount of value sitting in a LlamaPay contract managed by the LlamaPayRouterOracle contract.
- The beneficiary of the LlamaPay stream withdraws `Y` value. Now there is `X - Y = 0.97 * X` value in the vault, but the `cumulativeDailyMultiplier` stays at `1`.
- The guardian submits a batch of actions which reduces the amount of value in the vault by `0.09 * X`. This would put the vault at `0.88 * X` value which is below the threshold of `0.9 * X`. Since the `cumulativeDailyMultiplier` did not account for the withdrawal of funds from the LlamaPay stream, the calculation of the `newMultiplier` in the `afterSubmit` hook results in a value greater than `0.9` and therefore allows it. The exact calculation of `newMultiplier` in line 250 in this case is the following `(1 * (0.88 * X)) / (0.97 * X)`, which results in `~0.907 > 0.9`. Therefore, the check in line 253 passes.

While this does not immediately lead to an unexpected loss of funds, it breaks the assumed invariant of total value locked in the vault being above the lower bound over any given 24-hour period. Consider documenting this fact thoroughly at the appropriate place.

Update: *Partially resolved in commit 9783d00. A note has been added to the code explaining the issue, and in this case, the invariant is only considered for the vault value changes during submit actions and the behaviour is as expected.*

Conclusion

The provided code is well-written and thoroughly documented. The overall structure of the code is sound and correctly manages the created streams. Our team has pinpointed one low and one note issue, where the low issue could enable minor MEV opportunities that benefit entities controlling transaction order. Nonetheless, these issues are unlikely to have significant repercussions, and the overall integrity and safety of the code remain intact.