SOLIDIFIED

Audit Report for Reflexer - June 24, 2021

Summary

Audit Report prepared by Solidified covering the Reflexer Uniswap V3 Liquidity Manager smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on June 21, 2021, and the results are presented here.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/reflexer-labs/geb-uni-v3-manager

Commit number: `6c59ae05b2a5e99d1c2eb11896b957418b3cc58c`

UPDATE: Fixes received on 23 June in PR:
https://github.com/reflexer-labs/geb-uni-v3-manager/pull/15

Final Commit number: `5d15f33bed17e7b6606de940e87c00f7b61ec0b1`

Intended Behavior

Uniswap V3 Liquidity Manager is a contract that manages Uniswap V3 positions for a pool containing a GEB system coin and wraps these positions into an ERC20 on behalf of the LPs.

SOLIDIFIED

Audit Report for Reflexer - June 24, 2021

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	Medium	-
Test Coverage	Medium	-

SOLIDIFIED

Audit Report for Reflexer - June 24, 2021

Issues Found

Solidified found that the Uniswap V3 Liquidity Manager contracts contain 1 critical issue, no major issues, 3 minor issues, and 8 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	PeripheryPayments.sol: Anyone can call function unwrapWETH9(), potentially draining the entire contract WETH balance	Critical	Resolved
2	GebUniswapV3ManagerBase.sol: Function removeAuthorization() does not check for self deauthorization	Minor	Acknowledged
3	GebUniswapV3ManagerBase.sol: Function modifyParameters() does not revert on invalid parameter	Minor	Resolved
4	Multiple castings between different integers sizes without result check	Minor	Resolved
5	GebUniswapV3LiquidityManager.sol: Redundant value check for delay_ in constructor	Note	Resolved
6	GebUniswapV3ManagerBase.sol: authorizedAccounts mapping uses uint256 instead of bool	Note	Acknowledged
7	GebUniswapV3ManagerBase.sol: Documentation mismatch for function _getTokenAmountsFromLiquidity()	Note	Resolved
8	GebUniswapV3ManagerBase.sol: Contract does not explicitly import ERC20	Note	Acknowledged
9	Multiple spelling mistakes exist in both code and comments	Note	Resolved

10	GebUniswapV3ManagerBase.sol: Consider using named constants instead of magic numbers in getTargetTick()	Note	Resolved
11	PoolViewer.sol: Simulation functionality is fragile	Note	Resolved
12	State modifying functions naming can currently be misleading	Note	Resolved

SOLIDIFIED

Audit Report for Reflexer - June 24, 2021

Critical Issues

1. PeripheryPayments.sol: Anyone can call function unwrapWETH9(), potentially draining the entire contract WETH balance

Function `unwrapWETH9()` is declared as `public`, allowing anyone to call it and potentially drain any `WETH` balance that exists in the contract.

Recommendation
Restrict caller access for function `unwrapWETH9()`.

Status
Resolved

Major Issues

No critical issues have been found.

Minor Issues

2. GebUniswapV3ManagerBase.sol: Function removeAuthorization() does not check for self deauthorization

Function `removeAuthorization()` does not check that the passed `account` is not the same as the calling account. This can potentially lead to the last authorized account accidentally deauthorizing themselves, thus leaving the contract stuck with no authorized accounts and without a way of reauthorizing any new ones.

Recommendation
Require that `account != msg.sender`.

Status
Acknowledged. Development team response:
"We are aware of this possibility. All our contracts behave this way, allowing for the sender himself to revoke his authorization".

3. GebUniswapV3ManagerBase.sol: Function modifyParameters() does not revert on invalid parameter

---

Function `modifyParameters(bytes32,address)` does not revert when passed an invalid `parameter` name, and proceeds to emit the `ModifyParameters` event regardless of whether a parameter was actually modified.

Recommendation
Revert on invalid parameter names.

Status
Resolved

4. Multiple castings between different integers sizes without result check

---

Multiple castings between different signed and unsigned integer sizes without check can lead to over or under flows. Several instances include:

GebUniswapV3ManagerBase.sol
`getTargetTick()`: casts `sqrt()` (`uint256`) into `uint160`
`getTicksWithThreshold()`: casts `threshold` (`uint256`) into `int24`
`_getTokenAmountsFromLiquidity()`: casts `uint256` into `uint128`

GebUniswapV3LiquidityManager.sol
`deposit()`: casts `newLiquidity` (`uint256`) into `uint128`

GebUniswapV3TwoTrancheManager.sol
`getAmountFromRatio()`: Safemath is used for `uint128`, despite never setting a library up for it
`getTokenAmountsFromLiquidity()`: `_liquidity` is scaled up to `uint256` and back to `uint128`

Recommendation
Check for over/under flows after casting.

Status
Resolved

Informational Notes

5. `GebUniswapV3LiquidityManager.sol`: Redundant value check for `delay_` in constructor

The value for `delay_` is already checked in `GebUniswapV3ManagerBase`'s constructor and does not need to be rechecked in `GebUniswapV3LiquidityManager`.

Status
Resolved

6. `GebUniswapV3ManagerBase.sol`: `authorizedAccounts` mapping uses `uint256` instead of `bool`

Consider using `bool` instead of `uint256` for `authorizedAccounts` mapping.

Status
Acknowledged. Development team response:
"All our contracts use uint256 on the authorizedAccounts mapping, we will maintain this to ensure consistency with the remaining interfaces".

7. GebUniswapV3ManagerBase.sol: Documentation mismatch for function _getTokenAmountsFromLiquidity()

The documentation states that function `_getTokenAmountsFromLiquidity()` is a `view` function, while in actuality it's state modifying.

Status
Resolved

8. GebUniswapV3ManagerBase.sol: Contract does not explicitly import ERC20

The `GebUniswapV3ManagerBase` contract inherits from `ERC20` without explicitly importing it.

Recommendation
Explicitly import `ERC20` to clarify which exact implementation is being used.

Status
Acknowledged. Development team response:
"ERC20 is imported in PeripheryPayments.sol, that is imported in GebUniswapV3Base.sol. The implementation used is in src/erc20/ERC20.sol".

9. Multiple spelling mistakes exist in both code and comments

`GebUniswapV3ManagerBase`: fuctions, worthiwhile, fview, aproved
`GebUniswapV3LiquidityManager`: liquidty
`GebUniswapV3TwoTrancheManager`: thresold, liquidty
`PherypheryPayments`: Pheryphery

Status
Resolved

10. GebUniswapV3ManagerBase.sol: Consider using named constants instead of magic numbers in getTargetTick()

Consider using named constants for the `scale` variable and the bit shift amounts.

Status
Resolved

11. PoolViewer.sol: Simulation functionality is fragile

Since the return of `delegatecall` cannot be checked due to an unconditional revert, a call that reverts inside the pool function cannot be distinguished from a genuine (0, 0) return.

Status
Resolved

12. State modifying functions naming can currently be misleading

Several of the state modifying function names can currently be misleading. For instance, the function named `getMaxLiquidity()` sounds like a view function due to the passive verb "get", while in actuality it's a state modifying function.

Status
Resolved

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Reflexer or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.