Zellic

Crocswap

Smart Contract Patch Review

October 16, 2023

Prepared for:

Doug Colkitt

Crocodile Labs

Prepared by:

Katerina Belotskaia

Zellic Inc.

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than to simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please email us at hello@zellic.io or contact us on Telegram at https://t.me/zellic_io.

1 Introduction

We were asked to review a patch to Crocswap which fixed an issue that the claimable fee rewards was zero for some affected LP positions, where the upper tick of the side of the range was initialized at a later time than the lower tick.

1.1 Scope

The engagement involved a review of the following targets:

Crocswap

Repository	https://github.com/CrocSwap/CrocSwap-protocol
Versions	`e140123ff5351b9326903acacf6aee1940da501c`
Type	Solidity
Platform	EVM-compatible

Contact Information

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia, Engineer
kate@zellic.io

1.2 Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any additional code added to the assessed project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial

advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.

2 Smart Contract Patch Review

The patch involved updating the CrocQuery, LiquidityMath, LevelBook, PoolRegistry, PositionRegistrar, StorageLayout, TradeMatcher and CrocLpErc20 contracts. The patch fixed an issue related to the claimable fee reward calculation.

The accumulated rewards for an LP position are calculated as the delta between the snapshot of the tick range rewards accumulator at the LP position's start time and the tick range's accumulator at the current time.

However, in a situation where the snapshot of the tick range reward as a result of underflow is greater than the tick range's accumulator at the current time, the `deltaRewardsRate` function returns 0 and as a result, no reward will be assigned.

2.1 Summary of changes

Updated the CrocQuery contract

- The `queryConcRewards` view function has been updated to point at 72-bit LP positions by adding the `uint72(type(uint64).max)` offset to `odometer`.

Updated the LiquidityMath contract

- The `blendMileage72`, `calcBlend72` and `deltaRewardsRate72` function has been added. They serve as equivalents to the `blendMileage`, `calcBlend` and `deltaRewardsRate` functions but handle `uint72` mileage instead of `uint64`.

Updated the LevelBook contract

- The `addBookLiq72`, `removeBookLiq72` and `clockFeeOdometer72` functions have been added. The `addBookLiq72` and `removeBookLiq72` are equivalent to the `addBookLiq` and `removeBookLiq` but use the 72-bit `clockFeeOdometer72` function for `feeOdometer` calculation. The difference of this function is that it adds a fixed offset of `type(uint64).max` to avoid the possibility of underflow.

Updated the PoolRegistry contract

- The `MAX_TAKE_RATE` constant has been moved to the StorageLayout.

Updated the PositionRegistrar contract

- The view function `lookupPosition` has been updated to return 72-bit version of mapping `positions72_` for `RangePosition72`.
- The `burnPosLiq`, `decrementLiq`, `harvestPosLiq`, `markPosAtomic`, `mintPosLiq` and

`incrementPosLiq` functions have been updated to handle 72-bit positions and are using the 72-bit versions of LiquitidyMath functions.

Updated the StorageLayout contract

- Added `RangePosition72` structure for 72-bit range positions.
- Added new mapping `positions72_` slot for `RangePosition72`.
- Added `POS_MAP_SLOT_72` constant variable equal to `65554`.
- The proxy indexes `LP_PROXY_IDX`, `LONG_PROXY_IDX` and `MICRO_PROXY_IDX` has been changed to new unused slots.
- The previous proxy indexes have been saved in `LP_PROXY_LEGACY_IDX`, `LONG_PROXY_LEGACY_IDX` and `MICRO_PROXY_LEGACY_IDX` to avoid collision.

Updated the TradeMatcher contract

- The `mintRange`, `burnRange` and `harvestRange` function have been updated, with the `feeMileage` variable type being updated to `uint72` and using the 72-bit versions function from the LevelBook contract.
- In the `depositConduit` and `withdrawConduit` function the `deflator` and `mileage` parameter's data types have been changed to `uint72`.

Updated the CrocLpErc20 and ICrocLPConduit

- In the `depositCrocLiq` and `withdrawCrocLiq` functions, the `mileage` parameter's data type has been changed to `uint72`.

3 Discussion

3.1 Insufficient Test Coverage

Upon reviewing the smart contract patch, our opinion is that the current testing suite is insufficient to ensure the robustness and security of the contract. Comprehensive testing is crucial to identify potential vulnerabilities and ensure the contract behaves as expected under various conditions.

The following functions do not have sufficient test coverage:

- The `queryConcRewards` function has only one test, but the updated `odometer` calculation depends on the `feeUpper` and `feeLower` values which can be calculated in different ways, depending on whether the value of `lowerTick` and `upperTick` is greater than `curveTick`. Tests should be added for all possible cases.
- The new `calcBlend72` function does not have unit tests.