CONIC FINANCE V2 SECURITY AUDIT REPORT

January 31, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Conic is a protocol on Ethereum that introduces "Omnipools", which are liquidity pools where users can deposit a single asset. Each Omnipool allocates liquidity to a set of whitelisted Curve pools.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Conic Finance
Project name	Conic Finance v2
Timeline	September 4 2023 - January 30 2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
04.09.2023	7a66d26ef84f93059a811a189655e17c11d95f5c	Commit for the audit
20.11.2023	7b0169ca9d8f1a3301f4b25207c66327f0ff8246	Commit for the re-audit
24.12.2023	5f6549833f0fa41014459755117f16a075bfe3cf	Commit for the re-audit 2
18.01.2024	681a4d5f88591a6446362812e86c3858377211a1	Commit for the re-audit 3

Project Scope

The audit covered the following files:

File name	Link
contracts/BaseConicPool.sol	BaseConicPool.sol
contracts/ConicEthPool.sol	ConicEthPool.sol

File name	Link
contracts/ConicPool.sol	ConicPool.sol
contracts/Controller.sol	Controller.sol
contracts/ConvexHandler.sol	ConvexHandler.sol
contracts/CurveHandler.sol	CurveHandler.sol
contracts/CurveRegistryCache.sol	CurveRegistryCache.sol
contracts/LpToken.sol	LpToken.sol
contracts/Pausable.sol	Pausable.sol
contracts/RewardManager.sol	RewardManager.sol
contracts/ConicDebtToken.sol	ConicDebtToken.sol
contracts/adapters/CurveAdapter.sol	CurveAdapter.sol
contracts/access/GovernanceProxy.sol	GovernanceProxy.sol
contracts/access/SimpleAccessControl.sol	SimpleAccessControl.sol
contracts/oracles/ChainlinkOracle.sol	ChainlinkOracle.sol
contracts/oracles/CrvUsdOracle.sol	CrvUsdOracle.sol
contracts/oracles/CurveLPOracle.sol	CurveLPOracle.sol
contracts/oracles/GenericOracle.sol	GenericOracle.sol
contracts/zaps/EthZap.sol	EthZap.sol
libraries/ArrayExtensions.sol	ArrayExtensions.sol
libraries/CurvePoolUtils.sol	CurvePoolUtils.sol
libraries/MerkleProof.sol	MerkleProof.sol

File name	Link
libraries/ScaledMath.sol	ScaledMath.sol
libraries/Types.sol	Types.sol
contracts/tokenomics/CNCDistributor.sol	CNCDistributor.sol
contracts/tokenomics/CNCLockerV2.sol	CNCLockerV2.sol
contracts/tokenomics/CNCLockerV2Wrapper.sol	CNCLockerV2Wrapper.sol
contracts/tokenomics/CNCMintingRebalancingRewardsHandler.sol	CNCMintingRebalancingRewardsHandler.sol
contracts/tokenomics/InflationManager.sol	InflationManager.sol
contracts/tokenomics/InflationRedirectionPool.sol	InflationRedirectionPool.sol
contracts/tokenomics/LpTokenStaker.sol	LpTokenStaker.sol

Deployments

File name	Contract deployed on mainnet	Comment
CNCLockerV3	0x8b318d...025B2c93	
CNCMintingRebalancingRewardsHandler	0x53a2f0...54077f9D	
ChainlinkOracle	0xd91868...AaFe04B7	
ConicDebtToken	0xFB5888...855E928A	
Controller	0x2790EC...3DF57EaE	
ConvexHandler	0xca432A...6f92D798	
CurveAdapter	0x396b4C...c10E898C	
CurveHandler	0xBcFacE...C0529b13	
CurveLPOracle	0x143f4f...7E951be0	
CurveRegistryCache	0x29E06b...636273C0	
EthZap	0x78036D...5d0d42B3	
GenericOracle	0x865934...C3114E07	
GovernanceProxy	0x38A409...576CdaAf	
InflationManager	0x05F494...30105B16	
LpTokenStaker	0xA52415...FFff0CF9	
RewardManager	0x71e182...2FD1C558	RewardManager for ETH Pool
RewardManager	0x15C606...04119532	RewardManager for USDC Pool

File name	Contract deployed on mainnet	Comment
RewardManager	0xBf65Fa...ee60F9Ad	RewardManager for crvUSD Pool
ConicPool	0x80a360...3FA64316	Pool for USDC
ConicPool	0x89dc3E...dE591988	Pool for crvUSD
ConicEthPool	0x336707...496f3543	Pool for ETH
LpToken	0x58649E...29A394d3	cncETH
LpToken	0xd02bCd...8F5e035e	cncUSDC
LpToken	0x9961Bd...3f564A8c	cncCRVUSD

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	20
Low	16

ID	Name	Severity	Status
M-1	Transferring tokens without tainting	Medium	Fixed
M-2	`CurveLPOracle` is not working	Medium	Fixed
M-3	An incorrect interface version definition	Medium	Acknowledged
M-4	Unsafe `safeApprove` and `safeIncreaseAllowance`.	Medium	Fixed
M-5	GenericOracle public frontrun for `initialize()`	Medium	Fixed
M-6	ChainlinkOracle fails to return WBTC price, pools with WBTC are not supported	Medium	Acknowledged
M-7	Reentrancy in `GovernanceProxy._executeChange()`	Medium	Fixed
M-8	GovernanceProxy DOS via `updateDelay()`	Medium	Fixed
M-9	GovernanceProxy pending change cannot expire	Medium	Acknowledged
M-10	LpToken taint griefing	Medium	Acknowledged

M-11	ChainlinkOracle integration problems	Medium	Fixed
M-12	`Controller.updateWeights()` can set a total weight differing from one	Medium	Fixed
M-13	`BaseConicPool.handleInvalidConvexPid()` doesn't set `rebalancingRewardActive` when invoking `_setWeightToZero()`	Medium	Acknowledged
M-14	Incorrect depeg check in `_isDepegged()` in ConicEthPool and ConicPool	Medium	Acknowledged
M-15	BaseConicPool's `cachedTotalUnderlying()` and `usdExchangeRate()` might work incorrectly	Medium	Acknowledged
M-16	BaseConicPool's `usdExchangeRate()` might use outdated `_cachedTotalUnderlying`	Medium	Fixed
M-17	Incorrect rebalancing for curve pools weighted above `100%-maxDeviation`	Medium	Fixed
M-18	Potential rewards loss due to LpTokenStaker switch in RewardManager	Medium	Acknowledged
M-19	Potential delays in `updatePoolWeights()` leading to unfair reward distribution in conic pools	Medium	Acknowledged
M-20	`exchangeRate` can be manipulated	Medium	Acknowledged
L-1	Inaccuracy in rounding	Low	Fixed
L-2	Reentrancy can be in base pool	Low	Fixed
L-3	RewardManager's extra reward token might have excessive slippage	Low	Acknowledged
L-4	No max length for pools indicated	Low	Fixed
L-5	Chainlink `min&max` price is not checked	Low	Acknowledged
L-6	CNCLockerV2 griefing	Low	Fixed
L-7	Multiple CNCMintingRebalancingRewardsHandler can break the targeted CNC TotalSupply distribution	Low	Acknowledged

L-8	No kick motivation in case of many little locks	Low	Fixed
L-9	Inconsistent logic in rebalancing rewards for withdrawals	Low	Acknowledged
L-10	`InflationManager.lastUpdate` is not used	Low	Fixed
L-11	Additional checks for `switchMintingRebalancingRewardsHandler()` are required	Low	Fixed
L-12	`airdropBoost` is not checked to be above ONE	Low	Fixed
L-13	Redundant shutdown check in donation	Low	Acknowledged
L-14	`previousRewardsHandler` may be null	Low	Fixed
L-15	`LpTokenStaker.getTimeToFullBoost()` - a full boost can be reached faster in some cases	Low	Fixed
L-16	`BaseConicPool. setWeightToZero()` does not update `Controller.lastWeightUpdate` leading to excessive minting of rewards	Low	Fixed

1.6 Conclusion

The audited scope includes well-written smart contracts. Test coverage is sufficient. After the audit 20 Medium and 16 Low severity findings have been discovered. All the findings have been confirmed and acknowledged or fixed by the client.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Transferring tokens without tainting
Severity	Medium
Status	Fixed in 7b0169ca

Description

- LpToken.sol#L74
- LpToken.sol#L81

If `_minimumTaintedTransferAmount` is large enough, then an attacker can do `deposit`/`withdraw` in a single transaction in small amounts.

For example:

- A hacker makes a deposit into `ConicPool`.
- Transfer several times with small amounts (less than `_minimumTaintedTransferAmount`) to another account.
- The hacker invokes `withdraw` from the new account.

This attack is unlikely, but if `_minimumTaintedTransferAmount` is large, it is a possibility.

Recommendation

We recommend adding a max limit to the `_minimumTaintedTransferAmount`.

Client's commentary

> We agree that this in an issue and have addresses this here: 46b6a34d
> Given that the likleihood is low and would require malicious governance, we think this should be a low severity issue (impact: medium, likelihood:low).

M-2	`CurveLPOracle` is not working
Severity	Medium
Status	Fixed in 7b0169ca

Description

- GenericOracle.sol#L34

`_chainlinkOracle` is the first in sequence in `GenericOracle.getUSDPrice`.

Since there is already a `crvUSD` oracle on the mainnet, `CurveLPOracle` will never be used. Current `Aggregator` on mainnet for `crvUSD`: `0x145f040dbCDFf4cBe8dEBBd58861296012fCB269` (https://data.chain.link/ethereum/mainnet/stablecoins/crvusd-usd).

Recommendation

We recommended reprioritising the selection of Oracles (`customOracles` should be first). If necessary

Client's commentary

> We have addressed this issue, following your recommendation here:
> 3c75e414

M-3	An incorrect interface version definition
Severity	Medium
Status	Acknowledged

Description

- CurveHandler.sol#L183

If `interfaceVersion_ == 0`, `_version_0_remove_liquidity_one_coin` is called in `CurveHandler`.

The following pools were found in the project with `0`:

- REN_BTC (0x930541...eDf0895B)
- SUSD_DAI_USDT_USDC (0xA5407e...C53efBfD)

All pools are considered as version `0`, but in `REN_BTC` there is a `remove_liquidity_one_coin` method. So, there is no need to use `_version_0_remove_liquidity_one_coin` for `REN_BTC`.

Recommendation

We recommend revising the pool versioning process (CurveRegistryCache.sol#L235).

Client's commentary

> We have double-checked this and it seems that the only advantage of using `remove_liquidity_one_coin` are gas savings.
> For simplicity and because we currently do not use any BTC pools in the protocol (and don't plan to do so for now), we continue to use the interfaces as it was.
> Given that this is not a security vulnerability, but a gas efficiency consideration, we would consider this issue low severity or "informational".

M-4	Unsafe `safeApprove` and `safeIncreaseAllowance`.
Severity	Medium
Status	Fixed in 7b0169ca

Description

- CurveHandler.sol#L246
- CurveHandler.sol#L266

Version 4.8.0 of `safeApprove` and `safeIncreaseAllowance` (SafeERC20.sol#L46-L68) is currently in use.

USDT approve method (it checks that allowance is zero):

```
function approve(
    address _spender,
    uint _value
    ) public onlyPayloadSize(2 * 32) {
...
    require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));
    allowed[msg.sender][_spender] = _value;
...
}
```

In the current implementation, if the ConicPool is left with an extra allowance for `CurvePool`, then users will not be able to withdraw their tokens due to revert.

Recommendation

We recommend doing `approve(0)` before calling the main approve (currently OpenZeppelin contracts have this logic SafeERC20.sol#L52).

Client's commentary

> The allowance is always fully consumed, so we do not see a case where that could actually be an issue.
> Therefore, we think that the severity should be "informational" or "best practices".
> We updated the code to use `forceApprove` in order to follow best practices: 32a4099d

M-5	GenericOracle public frontrun for `initialize()`
Severity	Medium
Status	Fixed in 7b0169ca

Description

The first call of `GenericOracle.initialize()` can be made by anyone.

- GenericOracle.sol#L20-L24

As a result, anyone can frontrun the official initialize() call and set malicious oracles, which is extremely dangerous for the whole project.

Recommendation

We recommend implementing an onlyOwner modifier, the same way as for `initialize()` functions in `Conroller` and`RewardManager`.

Client's commentary

> We updated the code to protect `initialize()`: 83b83e92.

M-6	ChainlinkOracle fails to return WBTC price, pools with WBTC are not supported
Severity	Medium
Status	Acknowledged

Description

`oracle.getUSDPrice(wbtc)` returns the error `"token not supported"`
As the result, it is impossible to add Curve pools with WBTC.

But, Conic mentions pools with WBTC in tests:
ConicTest.sol#L39

```
...
address internal constant REN_BTC = 0x9305...895B;
address internal constant BBTC = 0x071c...8F4b;
...
```

Also, there are many Curve pools with WBTC, some of them supported by Convex as well.
BTC is a reserved denomination with the address: `0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB`

- Denominations.sol#L7

So WBTC price feed can be received only by:

```
WBTC = 0x514910771AF9Ca656af840dff83E8264EcF986CA
BTC = 0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB
wbtcbtc = _feedRegistry.getFeed(WBTC, BTC)
```

It will be WBTC/BTC price; then it must be adjusted to BTC/USD price:

```
BTC = 0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB
USD = address(840)
btcusd = _feedRegistry.getFeed(WBTC, BTC)
```

Recommendation

We recommend implementing additional calculations in `ChainlinkOracle._getPrice()` and `ChainlinkOracle.isTokenSupported()` in order to support Curve Pools with WBTC.

Client's commentary

> We are not currently supporting any BTC pools in the protocol (nor do we plan to do so in the near future).
> Should we want to support BTC pools in the future, we can add an additional custom oracle to support these calculations.
> Given this, we would consider this issue low severity (impact: low, liklihood: low).

M-7	Reentrancy in `GovernanceProxy._executeChange()`
Severity	Medium
Status	Fixed in 7b0169ca

Description

The Check-Effect-Interaction pattern is violated in the `GovernanceProxy.executeChange()` function:

```
function _executeChange...
    ...
    for(uint256 i; i < change.calls.length; i++) {
        change.calls[i].target.functionCall(change.calls[i].data);
    }
    ...
    _endChange(change, index, Status.Executed);
```

GovernanceProxy.sol#L168-L172

`Change` is marked as completed only after all the `change.calls`.

If there is a call in the `change.calls` list to a contract with a hacker's hook (for example, an ERC-777 token), this could allow a malicious actor to repeatedly invoke `executeChange()` for this list of calls, leading to unforeseen consequences.

Recommendation

Move `_endChange()` before the calls to adhere to the Check-Effect-Interaction pattern.

Client's commentary

> We have addressed this issue, implementing your recommendation, here:
> a91d1e59

M-8	GovernanceProxy DOS via `updateDelay()`
Severity	Medium
Status	Fixed in 7b0169ca

Description

If an admin mistakenly calls `updateDelay(someImportantFunction, SUPER_BIG_NUMBER)` in GovernanceProxy, restoring a lower delay for `someImportantFunction` becomes impossible. To execute `updateDelay(someImportantFunction, LOW_NUMBER)`, one would need to wait out the initially set `SUPER_BIG_NUMBER` delay:

```
function _computeDelay(
    bytes calldata data
    ) internal view returns (uint64) {
    bytes4 selector = bytes4(data[:4]);

    // special case for `updateDelay`, we want to set the delay
    // as the delay for the current function for which the delay
    // will be changed, rather than a generic delay for `updateDelay` itself
    // for all the other functions, we use their actual delay
    if (selector == GovernanceProxy.updateDelay.selector) {
        bytes memory callData = data[4:];
        (selector, ) = abi.decode(callData, (bytes4, uint256));
    }

    return delays[selector];
}
```

GovernanceProxy.sol#L196-L210

In this situation, function calls might effectively get blocked for an excessively long period.

Recommendation

We recommended capping the maximum delay at a reasonable number.

Client's commentary

> We have addressed this issue, implementing your recommendation, here:
> b980b71a

M-9	GovernanceProxy pending change cannot expire
Severity	Medium
Status	Acknowledged

Description

One can imagine a situation where a pending change is created, locked by a delay, but over time becomes irrelevant and no one executes it. Or, for instance, over time it becomes evident that the requested change reverts upon attempted execution, and it's forgotten.

This could result in lingering transactions accumulating in `pendingChanges`, which might not only become irrelevant in the future but even detrimental (e.g., setting outdated thresholds for certain contracts). Any such change could be executed by any user in any order, leading to adverse effects.

Recommendation

Implement an expiration mechanism for pending changes.

Client's commentary

> We agree that this is an issue and are currently working on a fix.
>
> **Update:**
> After internal discussion, we reached the conclusion that a proposal should always be executed unless it is explicitly cancelled.
> If any actions are order dependent, they should be part of the same proposal, so we accept the fact that proposals could be executed in a different order from the one in which they have been proposed or they became ready.
> Therefore, we are happy with the current design.

M-10	LpToken taint griefing
Severity	Medium
Status	Acknowledged

Description

The LpToken.sol cannot be minted or burned by a user if someone has transferred an amount of LpToken to them greater than `controller.getMinimumTaintedTransferAmount(token)`:

```
function _ensureSingleEvent(address ubo, uint256 amount) internal {
    if(
        !controller.isAllowedMultipleDepositsWithdraws(ubo) &&
        amount > controller.getMinimumTaintedTransferAmount(address(this))
    ) {
        require(
            _lastEvent[ubo] != block.number,
            "cannot mint/burn twice in a block");
        ...
```

LpToken.sol#L81

This means a malicious actor could send a minimum number of tokens to any "whale" (a user with a large balance) trying to make a large deposit or withdraw, thereby blocking their operation.

For example, this could be used to attack users who want to urgently burn their LP tokens to repay an overcollateralized debt and avoid liquidation. This can also be used to block MEV bots that utilize mint or burn lp tokens in their path strategies. It also removes the ability to buy LP tokens on an exchange and burn them in a single transaction.

It's worth noting that by default, `controller.getMinimumTaintedTransferAmount(token)` equals zero, so the attacker would only need to pay for the gas to block a specific user's operations.

Recommendation

We recommended allowing users to burn LP tokens received before the current taint.

For instance, if in the first block a user mints 1000 tokens for themselves, and then in the second block they receive another token, it would be desirable in the second block to still allow them to burn the initial 1000 tokens. However, it should revert if they try to burn more than that amount.

Client's commentary

> In practice, we will make the minimum tainted value around ~1k USD, making griefing prohibitive enough and channelling a large flash loan through many accounts almost impossible.

M-11	ChainlinkOracle integration problems
Severity	Medium
Status	Fixed in 7b0169ca

Description

There are several shortcomings in the current implementation of the interaction with Chainlink.

1. `ChainlinkOracle.isTokenSupported()` returns true if a Chainlink feed exists, but it does not consider the case when it has been abandoned (not updated for a long time):

```
function isTokenSupported(...) external view override returns (bool) {
   ...
   try _feedRegistry.getFeed(...) returns (IAggregatorV2V3) {
      return true;
   } catch Error(string memory) {
      try _feedRegistry.getFeed(...) returns (IAggregatorV2V3) {
         return true;
```

ChainlinkOracle.sol#L52-L56

2. `ChainlinkOracle._getPrice()` uses the deprecated `answeredInRound`, see https://docs.chain.link/data-feeds/api-reference#latestrounddata

```
function _getPrice(
    ...
    require(answeredInRound_ >= roundID_, "stale price");
```

ChainlinkOracle.sol#L83

3. `ChainlinkOracle._getPrice()` doesn't check for stale prices.

Each feed has a `heartbeat`, and for each call to `latestRoundData()` the equation `updatedAt < block.timestamp - heartbeat` must be checked, see https://ethereum.stackexchange.com/questions/133890/chainlink-latestrounddata-security-fresh-data-check-usage.

4. `ChainlinkOracle.getUSDPrice()` has a check for `price_ != 0` which should actually be `price_ > 0` since it is `int256` and could hypothetically be negative:

```
function _getPrice
    ...
    require(price_ != 0, "negative price");
```

ChainlinkOracle.sol#L82

Recommendation

Recommendations are as follows:

1. Add a check for abandoned pools in `isTokenSupported()`.
2. Remove the deprecated `answeredInRound` check.
3. Implement checks for stale prices.
4. Ensure the `price_ > 0`.

Client's commentary

> We have addressed this issue, following your recommendation, here:
> e0485c53

M-12	`Controller.updateWeights()` can set a total weight differing from one
Severity	Medium
Status	Fixed in 7b0169ca

Description

When invoking `Controller.updateWeights()` or `BaseConicPool updateWeights()`, it's possible to pass in duplicate pools so that the sum of the weights in the provided list would equal one. However, the resulting sum of `BaseConicPool.weights` can be either greater or less than one.

The problem lies in the fact that when setting the weights, only the number of pools and the total weight of the provided list are checked, but the final weights in `BaseConicPool.weights` are not verified:

```
function updateWeights(
    PoolWeight[] memory poolWeights
    ) external onlyController {
    ...
    require(poolWeights.length == _pools.length(), "invalid pool weights");
    ...
    for (uint256 i; i < poolWeights.length; i++) {
        ...
        uint256 newWeight = poolWeights[i].weight;
        total += newWeight;
        ...
    require(total == ScaledMath.ONE, "weights do not sum to 1");
```

BaseConicPool.sol#L528

As a result, for example, if there are pools `A 25%, B 25%, C 25%, D 25%`, and the admin mistakenly calls function `updateWeights(A 70%, B 10%, B 10%, B 10%)`, both checks would pass:

```
require(poolWeights.length == _pools.length(), "invalid pool weights");
...
require(total == ScaledMath.ONE, "weights do not sum to 1");
```

Subsequently, the pool weights would become `(A 70%, B 10%, C 25%, D 25%)`, which totals 130%.

Recommendation

It is recommended to check the invariant that the sum of `BaseConicPool.weights` is equal to one at the end of the function.

Client's commentary

> We have addressed this issue, by enforcing pool uniqueness in the function, here:
> f21f0175
> Since this would require malicious governance, we would consider this a low severity issue (impact: medium, likelihood: low).

M-13	`BaseConicPool.handleInvalidConvexPid()` doesn't set `rebalancingRewardActive` when invoking `_setWeightToZero()`
Severity	Medium
Status	Acknowledged

Description

Generally, when weights are changed in BaseConicPool, `rebalancingRewardActive` gets updated. For instance, `BaseConicPool.handleDepeggedCurvePool()` invokes `_setWeightToZero()` and sets `rebalancingRewardActive = true`. The `updateWeights()` function also sets `rebalancingRewardActive = !_isBalanced(allocatedPerPool, totalAllocated)`.

However, the `handleInvalidConvexPid()` function invokes `_setWeightToZero()` but, unlike `handleDepeggedCurvePool()`, it doesn't alter the `rebalancingRewardActive` variable. This seems to be a deviation from the intended logic.

Recommendation

We recommended revising the logic behind `_setWeightToZero()` and possibly updating `rebalancingRewardActive` within it, similarly to how it's done for both `handleDepeggedCurvePool()` and `handleInvalidConvexPid()`.

Client's commentary

> This is the intended behaviour.
> If this happens, it means that LPs will be missing out on some rewards for a brief period of time, which we thought does not justify paying rebalancing rewards.
> In this scenario, accelerated rebalancing is not necessary, since there is no risk to LPs (as it is the case in a depeg event).

M-14	Incorrect depeg check in `_isDepegged()` in ConicEthPool and ConicPool
Severity	Medium
Status	Acknowledged

Description

`_isDepegged()` checks the percentage change between the current token price and its cached value, which is updated every time `updateWeights()` is invoked. This implies that `_isDepegged()` checks the token price deviation not from a benchmark value but from the last call to `updateWeights()`.

It's worth noting that at the moment of invoking `updateWeights()`, the price can be abnormal. For instance, if the depeg threshold is 10%, and the token price deviates by 9%, then `_isDepegged()` will assume a 9% deviation. If `updateWeights()` is subsequently invoked, the cache will update to the new price, and `_isDepegged()` will consider the token price deviation as 0%. If the token price then rises by another 9% (accumulatively 18% from the starting point), `_isDepegged()` will consider it a mere 9% deviation, assuming it's within the norm when in fact, it's not.

Recommendation

We recommend that you rework the architecture so that the `_isDepegged()` function accounts for the full deviation of the token from the benchmark value, not just from the last `updateWeights()` call.

Client's commentary

> The Curve pools are picked during the LAVs and we would expect that the chosen pools are not depegged at that point. This means that they only risk would be some depeg just when we want to execute this LAV, which is unlikely.
> We consider the token value at the time of the last LAV to be a better benchmark value than the intended peg of stable coins included in the pools.
> Consequently, we do not consider this an issue.

M-15	BaseConicPool's `cachedTotalUnderlying()` and `usdExchangeRate()` might work incorrectly
Severity	Medium
Status	Acknowledged

Description

Both `cachedTotalUnderlying()` and `usdExchangeRate()` utilize `_cachedTotalUnderlying` in their computations. The `_cachedTotalUnderlying` variable gets updated in the `withdraw()` function. Note that `underlyingWithdrawn_` could be less than the complete extracted value from curve pools due to slippage, meaning the actual total underlying value might be less than `totalUnderlying_`.

In such scenarios, `_cachedTotalUnderlying` is inaccurately computed, making its value appear larger than it should be. This affects the computations in `cachedTotalUnderlying()` and `usdExchangeRate()`.

Recommendation

Client's commentary

> We are aware that the `_cachedTotalUnderlying` can be off slightly and have accepted this risk. We are fine with this because of the way in which this value is used. We do not use the cached value for anything where the actual amount is critical to the protocol. It is only used for distributing CNC, where light variances are considered acceptable and do not impact the protocol negatively. We do not want to obtain a fresh underlying balance like we do in the `depositFor` function because this would make the `withdraw` function significantly more gas expensive.
> We would therefore not agree that this is an issue.

M-16	BaseConicPool's `usdExchangeRate()` might use outdated `_cachedTotalUnderlying`
Severity	Medium
Status	Fixed in 7b0169ca

Description

`usdExchangeRate()` uses `_cachedTotalUnderlying` for its calculation:

```
function usdExchangeRate()
    external view virtual override returns (uint256) {
    uint256 underlyingPrice =
        controller.priceOracle().getUSDPrice(address(underlying));
    return _exchangeRate(_cachedTotalUnderlying).mulDown(underlyingPrice);
```

BaseConicPool.sol#L304-L308

However, `_cachedTotalUnderlying` can be arbitrarily outdated, which might render `usdExchangeRate()` inaccurate.

Recommendation

We recommend using `cachedTotalUnderlying()`, which accounts for `_TOTAL_UNDERLYING_CACHE_EXPIRY`, instead of `_cachedTotalUnderlying`,

Client's commentary

> We have implemented the recommendation: 40d16566

M-17	Incorrect rebalancing for curve pools weighted above `100%-maxDeviation`
Severity	Medium
Status	Fixed in 7b0169ca

Description

In situations where the weight of the curve pool + `maxDeviation` exceeds 100%, all deposits go to that curve pool, while the balance of other curve pools remains unchanged.

Let's look in detail at why this happens:

```
function _getDepositPool
    ...
    for (uint256 i; i < poolsCount_; i++) {
        ...
        uint256 allocatedUnderlying_ = allocatedPerPool[i];
        uint256 targetAllocation_ =
            totalUnderlying_.mulDown(weights.get(pool_));

        if (allocatedUnderlying_ >= targetAllocation_) continue;
        uint256 maxBalance_ =
            targetAllocation_ +
            targetAllocation_.mulDown(_getMaxDeviation());

        uint256 maxDepositAmount_ = maxBalance_ - allocatedUnderlying_;
        if (maxDepositAmount_ <= maxDepositAmount) continue;
```

BaseConicPool.sol#L271-L272

In this function:

- `totalUnderlying_`: total underlying tokens after deposit;
- `targetAllocation_`: how many underlying tokens, in an ideal scenario, `ConicPool` should deposit in the curve pool;
- `maxBalance_`: the maximum allowable amount to be stored in the curve pool.

The expression is as follows:

```
max balance = total underlying * weight * (1 + max deviation)
```

If `weight * (1 + max deviation)` is above 100%, then the pool's maximum balance will always accommodate any new deposit.

Examples of such situations:

- pool weight 99%, max deviation 2%
- pool weight 84%, max deviation 20%

Note, that the problem arises only when `rebalancingRewardActive` is `false`:

```
function _getMaxDeviation() internal view returns (uint256) {
    return rebalancingRewardActive ? 0 : maxDeviation;
}
```

By default, the maxDeviation in the new Conic Pool is set to 2%:

```
uint256 public maxDeviation = 0.02e18; // 2%
```

However, an admin can increase this value up to 20%:

```
uint256 internal constant _MAX_DEVIATION_UPPER_BOUND = 0.2e18;
```

Recommendation

When changing weights or deviation, we recommended ensuring that any weight is strictly less than `100% - maxDeviation`.

Client's commentary

> We partially agree with this issue.
> In a case where the deviation plus the allocated weight of a pool can accommodate all deposits, allocating all deposits to that pool is intended behaviour.
> However, we agree that the weight of a pool plus the deviation should never exceed 100% and have addressed this issue here:
> 82058c9b
> Given that this issue would require a governance mistake and no funds would be at risk, we would consider this issue low severity (impact: low, likelihood: low).

M-18	Potential rewards loss due to LpTokenStaker switch in RewardManager
Severity	Medium
Status	Acknowledged

Description

Changing the LpTokenStaker via `Controller.switchLpTokenStaker()` results in all users losing a portion of their rewards in RewardManager. It happens because after changing the LpTokenStaker, `RewardManager._accountCheckpoint()` will use a new address for which `LpTokenStaker.getUserBalanceForPool(conicPool, account)` will return zero. As a result, the accumulated integral for the staking period will be reset to zero as the `balance` variable is zero:

```
function _updateAccountRewardsMeta(
    bytes32 key,
    address account,
    uint256 balance
    )
    ...
    uint256 share =
        balance.mulDown(
            meta.earnedIntegral - meta.accountIntegral[account]
            );
```

- RewardManager.sol#L229-L234

Recommendation

We recommend ensuring user rewards in `RewardManager` remain unaffected during transitions.

Client's commentary

> We accept this issue and choose not to make any changes.
> It should be very unlikley that we need to switch the `LPTokenStaker`.
> (impact:medium, likelihood:low).

M-19	Potential delays in `updatePoolWeights()` leading to unfair reward distribution in conic pools
Severity	Medium
Status	Acknowledged

Description

`InflationManager.updatePoolWeights()` updates `currentPoolWeights` proportionally to the dollar-equivalent of funds locked in conic pools. This value is used in `getCurrentPoolInflationRate()` to determine how much CNC should be minted for a specific conic pool.

However, the system lacks guarantees for the timely invocation of `updatePoolWeights()`. Currently, this function is called from two places: `Controller.shutdownPool()` and `InflationManager._executeInflationRateUpdate()`.

`Controller.shutdownPool()` is not designed for frequent execution.

`InflationManager._executeInflationRateUpdate()` will trigger an update no more often than `_INFLATION_RATE_PERIOD = 365 days`:

```
function _executeInflationRateUpdate() internal {
    if (
        block.timestamp >= lastInflationRateDecay + _INFLATION_RATE_PERIOD
        ) {
        updatePoolWeights();
        currentInflationRate =
            currentInflationRate.mulDown(_INFLATION_RATE_DECAY);
        lastInflationRateDecay = block.timestamp;
    }
}
```

- InflationManager.sol#L193-L200

This means that the `updatePoolWeights()` function must be called manually which can occur with a significant delay relative to the real distribution of funds in different conic pools. This will lead to an unfair distribution of rewards.

Recommendation

We recommend implementing an automated and more frequent mechanism for invoking the `updatePoolWeights()` function to ensure timely and fair distribution of rewards across conic pools.

Client's commentary

> In the current implementation, we prioritize gas efficiency over accuracy of the inflation weights. This function is called regularly by the protocol maintainers and can be called altruistically by any user.
> We would therefore consider this issue low severity as the inaccuracies do not end up being significant in practice.

M-20	`exchangeRate` can be manipulated
Severity	Medium
Status	Acknowledged

Description

- BaseConicPool.sol#L290

When creating a pool, the user has an option to change the `exchangeRate` directly by sending tokens to ConicPool.

Test example:

```
vm.startPrank(bb8); # hacker is setting `exchangeRate`
conicPool.deposit(2, 0, false);
underlying.transfer(address(conicPool), 2 * 10 ** 18);
vm.stopPrank();

# now totalSupply = 1,
# exchangeRate = 200000000000000001000000000000000000 (~10**36)

# victims try to deposit to ConicPool and will get zero lp tokens
vm.startPrank(r2);
underlying.approve(address(conicPool), 10 ** 18);
conicPool.deposit(10 ** 18, 0, false);
vm.stopPrank();

vm.startPrank(bb8); # withdraw all tokens from pool by hacker
conicPool.withdraw(1, 0);
```

This way an attacker can make the use of omnipool unprofitable at the very beginning.

Recommendation

There are different approaches on how to solve the Inflation Attack problem. Some of the approaches along with their pros and cons, can be found in the OpenZeppelin github issue: https://github.com/OpenZeppelin/openzeppelin-contracts/issues/3706.

One way to resolve the problem is to use virtual dead shares, as implemented in the latest OpenZeppelin ERC-4626 vault:

- ERC4626.sol#L200

- ERC4626.sol#L207

In case this particular fix is chosen, it is recommended to use a virtual offset of 1000 (UniswapV2Pair.sol#L120), as this will make the residual possibility of griefing practically unattainable.

However, one of the simplest solutions is making a deposit immediately after the deployment so that the `exchangeRate` is adjusted.

Client's commentary

> This issue can be easily prevented by seeding the pool immediately after it is deployed.
> For all pools we deploy, we follow that approach.
> We would only add the pool to the controller after it is seeded.
> Therefore, we chose not to implement any additional changes to address this potential issue.
> Given that this issue can only occur in a fairly unlikely scenario and cannot affect the protocol as we check the pool before adding it, we disagree with the severity rating of this issue and believe it should be low severity (impact: low, likelihood: low).

2.4 Low

L-1	Inaccuracy in rounding
Severity	Low
Status	Fixed in 7b0169ca

Description

- CurveAdapter.sol#L103

```
underlyingAmount =
    usdAmount.convertScale(18, decimals).divDown(underlyingPrice);
```

In the `computePoolValueInUnderlying` method, if `decimals` is small, accuracy in rounding is lost.

Recommendation

We recommend calling `convertScale` after `divDown`.

Client's commentary

> We followed the recommendation in the following commit: 119ad8d5

L-2	Reentrancy can be in base pool
Severity	Low
Status	Fixed in 7b0169ca

Description

- CurveHandler.sol#L74

It is supposed to use `basePool` in `CurveHandler` to work with LP tokens.

However, there is no read-only reentrancy check for base pools (ConicEthPool.sol#L48).

In some situations, where the underlying pool contains ETH, reentrancy is possible. Currently, only 3CRV is used.

Recommendation

We don't recommend using the base pool which contains the read-only reentrant.

Client's commentary

> Although we do not have an immediate use case for it, we added a protection against read-only reentrancy in the `ConicEthPool`: ef134f5a

L-3	RewardManager's extra reward token might have excessive slippage
Severity	Low
Status	Acknowledged

Description

If the extra reward token is not supported by conic oracles, its slippage is set to zero upon sale:

```
function _minAmountOut
    ...
    if (!oracle_.isTokenSupported(tokenIn_) ||
        !oracle_.isTokenSupported(tokenOut_)
        ) {
        return 0;
    }
```

- RewardManager.sol#L504-L506

Such high slippage might be excessive for quality, highly liquid tokens.

Recommendation

This is a known issue and developers stated that they accept the risk.

By default, when adding new tokens via `addExtraReward()`, it's recommended to ensure they are supported by the oracle to prevent administrators from accidentally forgetting to add the necessary oracle. It might make sense to have a separate function, `addExtraRewardWithBadSlippage()`, for adding illiquid unsupported tokens.

Client's commentary

> We are aware of this and accept this risk. It is rare for us to receive additional reward tokens, and when we do, it is rare that they do not have a Chainlink oracle. So in this very rare case that we have an additional reward token with no oracle, we accept the risk of slippage on these swaps.

L-4	No max length for pools indicated
Severity	Low
Status	Fixed in 7b0169ca

Description

Ethereum gaslimit will not allow Conic Pool operation if too many pools are added.
Marginal gas consumption for additional pools:

- ~ 820 000 for deposits
- ~ 1 300 000 for withdrawals

So, the approximate max length is 22 pools.

Recommendation

We recommend indicating some desired limit for the length of pools.

Client's commentary

> We limited the maximum number of pools to 10: 4b4fd4f3

L-5	Chainlink `min&max` price is not checked
Severity	Low
Status	Acknowledged

Description

- ChainlinkOracle.sol#L68

Some chainlink aggregators have min and max price. The price of an asset cannot go outside the range of min and max price.

- https://docs.chain.link/data-feeds#check-the-latest-answer-against-reasonable-limits

`When the reported answer is close to reaching reasonable minimum and maximum limits ... it can alert you to potential market events.`

Recommendation

According to Chainlink's recommendation:
`Configure your application to detect and respond to extreme price volatility or prices that are outside of your acceptable limits.`

Client's commentary

> Thanks for recommending this. We are not currently using this feature.

L-6	CNCLockerV2 griefing
Severity	Low
Status	Fixed in 7b0169ca

Description

CNCLockerV2 allows for locking **0 wei** in favor of any account. If a hacker creates enough **0 wei** locks in favor of a victim over the maximum period `_MAX_LOCK_TIME = 240 days`, the `voteLocks[user]` array will become too large to iterate over in a loop.

This will break all functions that iterate over this loop:

- lockFor(relock_=true)
- executeAvailableUnlocksFor()
- unlockableBalance()
- unlockableBalanceBoosted()
- totalRewardsBoost()
- totalVoteBoost()

It can also potentially cause a revert in _getLockIndexById() which leads to the breaking of:

- executeUnlocks()
- _relock()
- relockMultiple()
- relock()
- kick()

Recommendation

We recommend limiting the maximum number of locks for a user and ensuring that the lock amount exceeds a certain minimum.

Client's commentary

> This has been fixed by following your recommendations here 4b7b7b12.

L-7	Multiple CNCMintingRebalancingRewardsHandler can break the targeted CNC TotalSupply distribution
Severity	Low
Status	Acknowledged

Description

The TotalSupply is strictly capped with 10 mln tokens with zero mints if trying to mint above this value.

- CNCToken.sol#L30

But `InflationManager` does not control the whole inflation.
In fact, there can be many `CNCMintingRebalancingRewardsHandler` contracts, each with custom `_MAX_REBALANCING_REWARDS`.

There are a few evidences that there can be many `CNCMintingRebalancingRewardsHandler` contracts:

- InflationManager.sol#L126
- InflationManager.sol#L162-L167

As a result, it is a risk that some new `CNCMintingRebalancingRewardsHandler` can break the targeted total supply structure.

In addition, all current minters sum up to 100%:

1. 25% InflationManager (LPTokenStaker)
   InflationManager.sol#L24-L26
2. 56% PreMint, Treasury, Airdrop
   CNCToken.sol#L18-L22
3. 19% one RebalancingHandler
   CNCMintingRebalancingRewardsHandler.sol#L28

So, there is no place for additional RebalancingHandlers.

Recommendation

We recommend removing the logic allowing multiple RebalancingHandlers.
If it is necessary, we recommend using `CNCToken.inflationMintedRatio()` as a limitation and an estimation for a correct minting amount allowed to all RebalancingHandlers.

Client's commentary

> There should only ever be a single instance of `CNCMintingRebalancingRewardsHandler` at any time.
> We added support for multiple reward handlers to be able to reward users in different ways for rebalancing the pools (e.g. reduce the CNC distribution and add payments using treasury funds) but we do not have any such plans just yet and we have not implemented any other type of reward handlers yet.
> Therefore, this should not be an issue.

L-8	No kick motivation in case of many little locks
Severity	Low
Status	Fixed in 7b0169ca

Description

A `kick()` caller receives a kickPenalty which is `voteLocks[].amount * 10%`.

- CNCLockerV2.sol#L342

If this reward does not exceed the gas cost, no one would have the motivation to call `kick()`. It is possible when `voteLocks[].amount` is small enough.

So, users are motivated to make many smaller locks to protect their locks from kicking.

Recommendation

We recommend introducing a function of kicking many expired voteLocks in one call.

Client's commentary

> This has been fixed by following your recommendations here 90c4c14e.

L-9	Inconsistent logic in rebalancing rewards for withdrawals
Severity	Low
Status	Acknowledged

Description

Both deposits and withdrawals can decrease deviation from target weights.

But there are two ways to treat withdrawals.

1. in `BaseConicPool`

`_handleRebalancingRewards()` is only called in `depositFor()` and not called for withdrawals.

- BaseConicPool.sol#L183-L189

2. in `CNCMintingRebalancingRewardsHandler`

`rebalance()` here makes both a deposit and a withdrawal and sums up the effect of deviation decrease from both operations.

Thus, withdrawals are rewarded here.

- CNCMintingRebalancingRewardsHandler.sol#L150-L152

Recommendation

We recommend keeping logic consistent and consider the introduction of `_handleRebalancingRewards()` for withdrawals in `BaseConicPool`. It will also allow switching `rebalancingRewardActive` when withdrawals push deviation below 2%.

Client's commentary

> This is a design decision. We are happy to incentivize deposits, as well as rebalancing, since the former increases TVL, while the latter does not have any effect on it. However, we do not want to incentivize withdrawals, since they decrease TVL.
> We therefore so not consider this an issue.

L-10	`InflationManager.lastUpdate` is not used
Severity	Low
Status	Fixed in 7b0169ca

Description

It only is set once on deployment, works as a timestamp of deployment, and is never used by other contracts.

- InflationManager.sol#L43

Recommendation

We recommend removing `lastUpdate`.

Client's commentary

> This has been fixed by following your recommendations here a822fa51.

L-11	Additional checks for `switchMintingRebalancingRewardsHandler()` are required
Severity	Low
Status	Fixed in 7b0169ca

Description

Switching a RebalancingManager requires a transaction flow described in the comments.

- CNCMintingRebalancingRewardsHandler.sol#L167-L173

`switchMintingRebalancingRewardsHandler()` perfectly checks that the old handler is removed.

- CNCMintingRebalancingRewardsHandler.sol#L179-L185
  However, it does not check that the new handler is correctly added to the InflationManager.
  As a result, there is a risk of switching to a handler that does not exist in the InflationManager.

In addition, the new handler must be deployed and initialized correctly - it can be checked onchain as well.

Recommendation

We recommend using `switchMintingRebalancingRewardsHandler()` as a high-level function which performs all necessary checks for correct switching leaving no room for mistakes:

1. checking that the `newRebalancingRewardsHandler` is added for at least one pool in the InflationManager.
   As a result, it will be the check that the transaction flow described in the comments is made correctly.
2. checking that the new handler:

- has the correct `previousRewardsHandler` set
- is initialized correctly and has imported `totalCncMinted` from the previous handler

Client's commentary

> This has been fixed by following your recommendations here 915f3ff2.

L-12	`airdropBoost` is not checked to be above ONE
Severity	Low
Status	Fixed in 7b0169ca

Description

`claimAirdropBoost()` in `CNCLocker` only checks that `amount` is not above 3.5e18

- CNCLockerV2.sol#L372-L380

But it is also important that `amount` is above 1e18.
Otherwise, `_airdroppedBoost[]` would store an amount below 1e18.

In this case:

1. it will never be deleted because `lockFor()` only deletes if `airdropBoost_` is above 1e18 CNCLockerV2.sol#L110-L114
2. `airdropBoost()` would return the value below 1e18 when even no airdrop boost means at least 1e18

- CNCLockerV2.sol#L416-L418

Recommendation

We recommend writing `_airdroppedBoost[]` as 1e18 in `claimAirdropBoost()` if `amount` is below 1e18.

Client's commentary

> This has been fixed by following your recommendations here 1db2878e.

L-13	Redundant shutdown check in donation
Severity	Low
Status	Acknowledged

Description

`CNCDistributor` has the `donate()` function with the check protecting from donations if the contract is shutdown.

- CNCDistributor.sol#L57-L60

But in fact, it is still possible to donate via direct CNC transfers to `CNCDistributor`.
Thus, the shutdown check does not make much sense.

Recommendation

If blocking CNC transfers to the `CNCDistributor` is necessary, we recommend checking `CNCDistributor` shutdown within the `CNCToken` contract and reverting transfer attempts when shutdown.

Client's commentary

> `CNCDistributor.donate` is typically only called by maintenance, so this is to make sure that we do not call this after having shut down the distributor.
> Hence, we do not consider this an issue.

L-14	`previousRewardsHandler` may be null
Severity	Low
Status	Fixed in 7b0169ca

Description

- CNCMintingRebalancingRewardsHandler.sol#L64

`previousRewardsHandler` is used to count how much totalCncMinted was minted by the previous `CNCMintingRebalancingRewardsHandler`.

If this is a new contract, then zero is transmitted. However, the initialize method does not have such a check.

Recommendation

We recommend adding `previousRewardsHandler != address(0x0)`.

Client's commentary

> This has been fixed by following your recommendations here b380ce0d.

L-15	`LpTokenStaker.getTimeToFullBoost()` - a full boost can be reached faster in some cases
Severity	Low
Status	Fixed in 7b0169ca

Description

A full boost in this function only accounts for the time boost:

- LpTokenStaker.sol#L136-L140

The function only calculates +30 days since `userBoost.lastUpdated`

But in fact, given `stakeBoost`, totalBoost can be reached much faster than 30 days if capped with MAX_BOOST amount.
For example, given max available `stakeBoost` of 5, `timeBoost` will reach the cap in ~3 days, not 30 days.

Recommendation

If this behavior is not desired, we recommend taking into account `stakeBoost` when calculating the function.

Client's commentary

> We have removed this function, as it was not used. d0746bc9.

L-16	`BaseConicPool._setWeightToZero()` does not update `Controller.lastWeightUpdate` leading to excessive minting of rewards
Severity	Low
Status	Fixed in 7b0169ca

Description

Rewards for pool rebalancing when calling `BaseConicPool.depositFor()` grow proportionally to `Controller.lastWeightUpdate`:

```
function computeRebalancingRewards
    ...
    uint256 lastWeightUpdate =
        controller.lastWeightUpdate(conicPool);
    uint256 elapsedSinceUpdate =
        uint256(block.timestamp) - lastWeightUpdate;
    return
        (elapsedSinceUpdate *
            cncRebalancingRewardPerDollarPerSecond).mulDown(
            deviationDelta.convertScale(decimals, 18)
        );
```

- CNCMintingRebalancingRewardsHandler.sol#L128

The value of `lastWeightUpdate` is updated when calling `Controller.updateWeights()`. However, in case of a depeg of one of the curve pools, the weight of the respective curve pool is directly zeroed through the `BaseConicPool._setWeightToZero()` call inside `handleDepeggedCurvePool()`, which also triggers the rebalancing reward distribution process:

```
function handleDepeggedCurvePool(address curvePool_) external override
    ...
    _setWeightToZero(curvePool_);
    rebalancingRewardActive = true;
```

- BaseConicPool.sol#L575-L577

The `_setWeightToZero()` function doesn't update `Controller.lastWeightUpdate`, so the `elapsedSinceUpdate` multiplier when calculating rewards isn't reset.

The `_setWeightToZero()` function zeros out the weight of one curve pool and proportionally increases the weights of all other pools, creating an immediate imbalance between the target and actual volume of

funds in each of the pools. However, there's no actual relative imbalance between the pools since all weights have increased proportionally.

```
function _setWeightToZero(address zeroedPool) internal
    ...
    for (uint256 i; i < curvePoolLength_; i++)
        ...
        uint256 newWeight_ = pool_ == zeroedPool ? 0 :
            weights.get(pool_).mulDown(scaleUp_);
        weights.set(pool_, newWeight_);
```

- BaseConicPool.sol#L589-L590

Nevertheless, right after this event, large rewards will be minted for any new depositors because the `elapsedSinceUpdate` multiplier remains large, and the `deviationDelta` value is calculated in absolute, not relative numbers.

Recommendation

We recommended that you update the `Controller.lastWeightUpdate` when calling `BaseConicPool._setWeightToZero()`.

Client's commentary

> It is intentional in the design of Conic that CNC reblancing rewards are high when the `handleDepeggedCurvePool` function is called.
> We want these to be high, so the Omnipool is balanced again quickly, removing liquidity from the depegged pool before there is additional risk to LPs. The way that we achieve this is by intentionally not resetting the `lastWeightUpdate` such that the CNC rewards are high.
> For the very unlikely case where `updateWeights` would not be called according to schedule (which is currently 2 weeks), we added a check to make sure that the elapsed time when distributing rewards is bounded: 8537dd4a
> Given that the design is originally intended and that the time is bounded to two weeks under normal protocol operation, we think this issue should be low severity (impact: medium, likelihood: low).

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes