Canto Liquidity Mining Protocol Findings & Analysis Report

2023-11-20

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the Canto Liquidity Mining Protocol smart contract system written in Solidity. The audit took place between October 3 — October 6 2023.

Wardens

63 Wardens contributed reports to the Canto Liquidity Mining Protocol:

1. maanas
2. ni8mare
3. Satyam_Sharma
4. BanditxOx
5. adriro
6. HChang26
7. kutugu
8. sces60107
9. 0xweb3boy
10. 0xDING99YA
11. 3docSec

12. 0xWaitress
13. emerald7017
14. twicek
15. 0xpiken
16. hunter_w3b
17. niser93
18. radev_sw
19. albahaca
20. 0xAnah
21. JCK
22. hihen
23. MatricksDeCoder
24. JP_Courses
25. 0xdice91
26. cookedcookee
27. sandy
28. ZanyBonzy
29. invitedtea
30. naman1778
31. SAQ
32. SY_S
33. tabriz
34. shamsulhaq123
35. pipidu83
36. Raihan
37. lsaudit
38. Polaris_tow
39. debo
40. Mike_Bello90

41. 0xTheC0der
42. Eurovickk
43. marqymarq10
44. orion
45. wahedtalash77
46. xAriextz
47. BoRonGod
48. gzeon
49. SovaSlava
50. matrix_0wl
51. taner2344
52. Topmark
53. 0x3b
54. 100su
55. zpan
56. GKBG (KKat7531 and Stoicov)
57. BRONZEDISC
58. lukejohn
59. IceBear
60. 0xAadi
61. pep7siup
62. tpiliposian

This audit was judged by LSDan.

Final report assembled by thebrittfactor.

Summary

The C4 analysis yielded an aggregated total of 6 unique vulnerabilities. Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity and 5 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 37 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 16 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Canto Liquidity Mining Protocol repository, and is composed of 2 smart contracts written in the Solidity programming language and includes 145 lines of Solidity code.

In addition to the known issues identified by the project team, a Code4rena bot race was conducted at the start of the audit. The winning bot, **lllllll-bot** from warden lllllll, generated the Automated Findings report and all findings therein were classified as out of scope.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

High Risk Findings (1)

[H-01] Array Length of `tickTracking_` can be purposely increased to Brick Minting and Burning of most users' liquidity positions

Submitted by BanditxOx, *also found by* BanditxOx, maanas, emerald7017, adriro, twicek, 0xDING99YA, 0xpiken, 3docSec, *and* 0xWaitress

Lines of code

https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/mixins/LiquidityMining.sol#L24-L35
https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/mixins/LiquidityMining.sol#L122

Impact

A malicious user can brick minting, burning and harvesting of liquidity for almost all liquidity providers.

Important NOTE: This is a different vector from another gas issue, which is iterating over too many ticks in `(int24 i = lowerTick + 10; i <= upperTick - 10; ++i)`. That issue affects wide liquidity positions, while this attack vector affects even liquidity positions with a relatively small number of ticks.

Proof of Concept

When `accrueConcentratedPositionTimeWeightedLiquidity` is called, under most conditions for every potentially eligible tick, it will iterate over every `tickTrackingData` in `tickTracking`:

```
    while (time < block.timestamp && tickTrackingIndex < numTickTrac
```

`tickTracking` is iterated by `tickTrackingIndex++;`

The array mapped by `tickTracking_` is increased by 1 for a tick every time a trade through the liquidity pool changes the price from a different tick to this tick. This is

implemented in the `crossTicks` function:

```
function crossTicks(
    bytes32 poolIdx,
    int24 exitTick,
    int24 entryTick
) internal {
    uint256 numElementsExit = tickTracking_[poolIdx][exitTic
    tickTracking_[poolIdx][exitTick][numElementsExit - 1]
        .exitTimestamp = uint32(block.timestamp);
    StorageLayout.TickTracking memory tickTrackingData = Stc
        .TickTracking(uint32(block.timestamp), 0);
    tickTracking_[poolIdx][entryTick].push(tickTrackingData)
}
```

A user could purposely increase the length of the `tickTracking_` array and cause the gas limit to be reached whenever the array is looped over.

The price impact required to cross a tick is from 0 to 1 bps, with 1 bps as the tick width. This is already extremely small, but the attacker could have the swap amount be a very small fraction of a bps if they first swap to make the price end very close to a tick boundary, then execute multiple extremely small swaps which bounce the price back and forth over the tick boundary.

Note that the CANTO liquidity rewards are targeted to stable pools. An attacker can be quite confident, for example, that a USDC/USDT pool will trade at around `$1`, and the ticks closest to `$1` will always be eligible for rewards and therefore be looped over by all rewardable positions when `accrueConcentratedPositionTimeWeightedLiquidity` is called. Therefore, the attack can be targeted to just one or two ticks to affect almost every user.

`accrueConcentratedPositionTimeWeightedLiquidity` is called during minting, burning and harvesting liquidity positions. Therefore, this gas griefing attack will make all these functions revert for almost every user. This would basically break the functionality of concentrated liquidity pools on Ambient.

Contrast the effect to the cost to the attacker: using the aforementioned attack vector the main cost to the attacker will be the gas costs of performing the swaps.

This is far lower than the damage that is done to the protocol/users.

One additional factor which makes this attack easy to execute, is that crossing ticks, even if the entry and exit is within the same `block.timestamp`, adds to the array length. Tracking this is unnecessary, because the tick was active for 0 blocks, and therefore, the time delta and allocated rewards is zero.

Recommended Mitigation Steps

One immediate step would to `pop() tickTrackingData` as soon as the `exitTimestamp == entryTimestamp`. This happens to the last element of the array when `crossTicks` is called. Tracking this is unnecessary, because the tick was active for 0 blocks, and therefore, the time delta and allocated rewards is zero.

The documentation stated that CANTO rewards are meant to be distributed for stable pools for this codebase. The term “stable” could have different interpretations, but this recommendation assumes that this refers to stablecoin-like or pegged asset pairs such as stETH/WETH, USDT/USDC etc.

Instead of iterating through every tick, one could assume a range where the stable assets could lie and then reward all positions that lie within the specified range; in this case, +/- 10 ticks of the price tick.

This makes an assumption that these “stable assets” will actually stay pegged to each other. However, the current accounting architecture has multiple problems:

- Given the high number of loops required by the current accounting mechanism, there are multiple reasons that gas could run out. This includes iterating through too many ticks or having too many tick entries/exits.
- The current mechanism increases the gas costs of all minting, burning and harvesting.
- DOS attacks like the one described in this issue are possible.

Assuming a stable price has the downside of misallocating rewards if the stable assets depeg from each other. However, this may be a reasonable tradeoff to prevent this DOS attack.

Assessed type

DoS

OpenCoreCH (Canto) confirmed

Note: for full discussion, see here.

Medium Risk Findings (5)

[M-01] The Liquidity mining callpath sidecar owner can pull native tokens from the `Dex`

Submitted by maanas

The owner of the liquidity mining sidecar can pull the native coins that are stored in the `CrocSwapDex` to reward the users.

Proof of Concept

The `setConcRewards` and `setAmbRewards` functions don't check if the quoted amount of rewards are actually sent by the caller. This allows the owner to specify any total amount of native coin which are available in the `CrocSwapDex` from which the funds will be used when distributing the rewards.

```
function setConcRewards(bytes32 poolIdx, uint32 weekFrom, ui
    // require(msg.sender == governance_, "Only callable by
    require(weekFrom % WEEK == 0 && weekTo % WEEK == 0, "Inv
    while (weekFrom <= weekTo) {
        concRewardPerWeek_[poolIdx][weekFrom] = weeklyRewarc
        weekFrom += uint32(WEEK);
    }
}
```

https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol#L65C7-L72

```
    function setAmbRewards(bytes32 poolIdx, uint32 weekFrom, uir
        // require(msg.sender == governance_, "Only callable by
        require(weekFrom % WEEK == 0 && weekTo % WEEK == 0, "Inv
        while (weekFrom <= weekTo) {
            ambRewardPerWeek_[poolIdx][weekFrom] = weeklyReward;
            weekFrom += uint32(WEEK);
        }
    }
```

https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol#L74-L81

According to Ambient Docs they allow for deposits in native tokens.

Demo

Update TestLiquidityMining.js:

The funds added using `hardhat.setBalance()` is being used by the owner to distribute rewards

```
diff --git a/canto_ambient/test_canto/TestLiquidityMining.js b/c
index bd21a32..b917308 100644
--- a/canto_ambient/test_canto/TestLiquidityMining.js
+++ b/canto_ambient/test_canto/TestLiquidityMining.js
@@ -7,6 +7,7 @@ const { time } = require("@nomicfoundation/hardh
 var keccak256 = require("@ethersproject/keccak256").keccak256;

 const chai = require("chai");
+const { network, ethers } = require("hardhat");
 const abi = new AbiCoder();

 const BOOT_PROXY_IDX = 0;
@@ -218,7 +219,6 @@ describe("Liquidity Mining Tests", function
                 );
                 tx = await dex.userCmd(2, mintConcentratedLiqCmc
                         gasLimit: 6000000,
-                        value: ethers.utils.parseUnits("10", "et
                 });
                 await tx.wait();
```

```
@@ -243,6 +243,17 @@ describe("Liquidity Mining Tests", functior
                         BigNumber.from("999898351768")
                 );

+                let dexBal = await ethers.provider.getBalance(de
+                expect(dexBal.eq(0)).to.be.eq(true);
+
+                // dex gains native token from other methods
+                await network.provider.send("hardhat_setBalance'
+                        dex.address,
+                        ethers.utils.parseEther("2").toHexStrinc
+                  ]);
+                dexBal = await ethers.provider.getBalance(dex.ac
+                expect(dexBal.eq(ethers.utils.parseEther("2"))).
+
                 ////////////////////////////////////////////////
                 // SET LIQUIDITY MINING REWARDS FOR CONCENTRATEI
                 ////////////////////////////////////////////////
```

Tools Used

Hardhat

Recommended Mitigation Steps

Add a `msg.value` check in the rewards function to see that the total value is passed when call the functions.

```
diff --git a/canto_ambient/contracts/callpaths/LiquidityMiningPa
index e6c63f7..44dd338 100644
--- a/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol
+++ b/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol
@@ -65,6 +65,7 @@ contract LiquidityMiningPath is LiquidityMinir
     function setConcRewards(bytes32 poolIdx, uint32 weekFrom, u
         // require(msg.sender == governance_, "Only callable by
         require(weekFrom % WEEK == 0 && weekTo % WEEK == 0, "Ir
+        require((1 +(weekTo - weekFrom) / WEEK) * weeklyReward
         while (weekFrom <= weekTo) {
             concRewardPerWeek_[poolIdx][weekFrom] = weeklyRewar
             weekFrom += uint32(WEEK);
```

Assessed type

Rug-Pull

OpenCoreCH (Canto) acknowledged and commented:

> Rewards will be set and sent in the same Canto governance proposal.

[M-02] If `dt` is not updated accurately, `timeWeightedWeeklyPositionInRangeConcLiquidity_` might be updated incorrectly

Submitted by ni8mare

In the function `accrueConcentratedPositionTimeWeightedLiquidity`, inside the while block, `dt` is initialised as:

```
        uint32 dt = uint32(
            nextWeek < block.timestamp
            ? nextWeek - time
            : block.timestamp - time
        );
```

https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/mixins/LiquidityMining.sol#L98-L102

If `tickTracking.exitTimestamp != 0` then the following else block is executed on line 117:

```
    else {
        // Tick is no longer active
        if (tickTracking.exitTimestamp < nextWeek) {
            // Exit was in this week, continue with next tick
            tickActiveEnd = tickTracking.exitTimestamp;
            tickTrackingIndex++;
            dt = tickActiveEnd - tickActiveStart;
        } else {
```

```
            // Exit was in next week, we need to consider the current t
            tickActiveEnd = nextWeek;
        }
    }
```

https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/mixins/LiquidityMining.sol#L117-L128

`dt` is now updated to `tickActiveEnd - tickActiveStart;` when `tickTracking.exitTimestamp < nextWeek` as seen in the if block of the outer else block above.

But, when `tickTracking.exitTimestamp > nextWeek`, in the inner else block the value of `dt` is not updated:

```
    else {
        // Exit was in next week, we need to consider the current t
        tickActiveEnd = nextWeek;
        //@audit - No update on dt value
        }
```

Inside this else block as well, `dt` must equal `tickActiveEnd - tickActiveStart;`, where `tickActiveEnd = nextWeek;`. Without this update, if `tickTracking.exitTimestamp > nextWeek`, then `dt = nextWeek - time` or `dt = block.timestamp - time` as it was declared initially. For example, let's say that it was the former, that is, `dt = nextWeek - time` (according to the initial declaration). Assume that `tickActiveStart = tickTracking.enterTimestamp` (this is possible when tickTracking.enterTimestamp > time). Had the else block above (check @audit tag), updated `dt`, then `dt = tickActiveEnd - tickActiveStart;` => `dt = nextWeek - tickTracking.enterTimestamp`

So, on comparing again, initially -> `dt = nextWeek - time` and had there been an update on `dt` at the audit tag, `dt` would now be -> `dt = nextWeek - tickTracking.enterTimestamp`. Note that here, `tickTracking.enterTimestamp > time` (check the above para). So, `nextWeek - tickTracking.enterTimestamp`

`< nextWeek - time`. That means `dt` would be a smaller value had it been equal to `nextWeek - tickTracking.enterTimestamp`. But, since it's not updated, `dt` equals `nextWeek - time`, which is a bigger value.

`dt` is used to increase the value of time (used as an iterator in while loop). Since `dt` is a bigger value than required, the number of iterations of the while loop would be less than what it should be. This means that fewer iterations would be used to update `timeWeightedWeeklyPositionInRangeConcLiquidity_` on line 129

```
    timeWeightedWeeklyPositionInRangeConcLiquidity_[poolIdx][posKey]
                                (tickActiveEnd - tickActiveStart) *
```

`timeWeightedWeeklyPositionInRangeConcLiquidity_` is used to calculate the `rewardsToSend` value in `claimConcentratedRewards` function. If `timeWeightedWeeklyPositionInRangeConcLiquidity_` is incorrect, then the rewards to be sent to the user will be calculated incorrectly.

https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/mixins/LiquidityMining.sol#L181-L188

```
    uint256 overallInRangeLiquidity = timeWeightedWeeklyGlobalConcLi
       if (overallInRangeLiquidity > 0) {
          uint256 inRangeLiquidityOfPosition;
          for (int24 j = lowerTick + 10; j <= upperTick - 10; ++j) {
             inRangeLiquidityOfPosition += timeWeightedWeeklyPositic
           }
           // Percentage of this weeks overall in range liquidity th
           rewardsToSend += inRangeLiquidityOfPosition * concRewardR
```

Also, due to fewer number of iterations, `tickTrackingIndexAccruedUpTo_` might not be updated correctly.

```
    if (tickTrackingIndex != origIndex) {
      tickTrackingIndexAccruedUpTo_[poolIdx][posKey][i] = tickTracki
    }
```

```
    /// @notice Accrues the in-range time-weighted concentrated
    /// @dev Needs to be called whenever a position is modified
    function accrueConcentratedPositionTimeWeightedLiquidity(
        address payable owner,
        bytes32 poolIdx,
        int24 lowerTick,
        int24 upperTick
    ) internal {
        RangePosition72 storage pos = lookupPosition(
            owner,
            poolIdx,
            lowerTick,
            upperTick
        );
        bytes32 posKey = encodePosKey(owner, poolIdx, lowerTick,
        uint32 lastAccrued = timeWeightedWeeklyPositionConcLiqui
            poolIdx
        ][posKey];
        // Only set time on first call
        if (lastAccrued != 0) {
            uint256 liquidity = pos.liquidity_;
            for (int24 i = lowerTick + 10; i <= upperTick - 10;
                uint32 tickTrackingIndex = tickTrackingIndexAccr
                uint32 origIndex = tickTrackingIndex;
                uint32 numTickTracking = uint32(tickTracking_[pc
                uint32 time = lastAccrued;
                // Loop through all in-range time spans for the
                while (time < block.timestamp && tickTrackingInc
                    TickTracking memory tickTracking = tickTrack
                    uint32 currWeek = uint32((time / WEEK) * WEE
                    uint32 nextWeek = uint32(((time + WEEK) / WE
                    uint32 dt = uint32(
                        nextWeek < block.timestamp
                            ? nextWeek - time
                            : block.timestamp - time
                    );
                    uint32 tickActiveStart; // Timestamp to use
                    uint32 tickActiveEnd;
                    if (tickTracking.enterTimestamp < nextWeek)
                        // Tick was active before next week, nee
                        if (tickTracking.enterTimestamp < time)
                            // Tick was already active when last
```

```
                        tickActiveStart = time;
                    } else {
                        // Tick has become active this week
                        tickActiveStart = tickTracking.enter
                    }
                    if (tickTracking.exitTimestamp == 0) {
                        // Tick still active, do not increas
                        tickActiveEnd = uint32(nextWeek < bl
                    } else {
                        // Tick is no longer active
                        if (tickTracking.exitTimestamp < nex
                            // Exit was in this week, contir
                            tickActiveEnd = tickTracking.exi
                            tickTrackingIndex++;
                            dt = tickActiveEnd - tickActiveS
                        } else {
                            // Exit was in next week, we nee
                            tickActiveEnd = nextWeek;
                        }
                    }
                    timeWeightedWeeklyPositionInRangeConcLic
                        (tickActiveEnd - tickActiveStart) *
                }
                time += dt;
            }
            if (tickTrackingIndex != origIndex) {
                tickTrackingIndexAccruedUpTo_[poolIdx][posKe
            }
        }
    } else {
        for (int24 i = lowerTick + 10; i <= upperTick - 10;
            uint32 numTickTracking = uint32(tickTracking_[pc
            if (numTickTracking > 0) {
                if (tickTracking_[poolIdx][i][numTickTrackir
                    // Tick currently active
                    tickTrackingIndexAccruedUpTo_[poolIdx][p
                } else {
                    tickTrackingIndexAccruedUpTo_[poolIdx][p
                }
            }
        }
    }
    timeWeightedWeeklyPositionConcLiquidityLastSet_[poolIdx]
        posKey
    ] = uint32(block.timestamp);
}
```

Recommended Mitigation Steps

Add the line `dt = tickActiveEnd - tickActiveStart` in the place of the @audit tag, as shown above.

OpenCoreCH (Canto) confirmed and commented:

> Note that if the mentioned `else` branch is reached, `dt` is necessarily set to `nextWeek - time`, it cannot be `block.timestamp - time` (if the tick was exited in the next week, the next week cannot be in the future and greater than the block timestamp). So the only possible difference is regarding `tickActiveStart`, namely when `tickActiveStart = tickTracking.enterTimestamp` (in the other case, when `tickActiveStart = time`, we have `dt = nextWeek - time = tickActiveEnd - tickActiveStart`).

> I agree with the warden that in this particular case, the logic for updating `dt` is different in the `if` and `else` branch, which should not be the case. However, I think the logic in the `if` branch is wrong, not in the `else`: We want to set `dt` such that the next `time` value is equal to `tickActiveEnd` (because we accrued up to `tickActiveEnd`). To achieve this in all cases, we need to set `dt = tickActiveEnd - time` in the `if` block. Otherwise, when `tickTracking.enterTimestamp > time` we only add the time where the tick was in range to `time` (but not the time before that where the tick was out of range). Because we are also increasing the tick tracking index, this should not cause any problems in practice (the next enter timestamp will be greater than `time` by definition and we will only start accruing from there on again), but I think it should still be changed.

Note: for full discussion, see here.

[M-03] Rewards cannot be transferred when calling protocol command

Submitted by adriro, also found by HChang26

Rewards are set up using protocol commands, but its entry point is not payable.

Rewards can be set up by protocol authorities using the functions `setConcRewards()` and `setAmbRewards()` present in the LiquidityMiningPath contracts. These two are part of the "command" pattern used in the protocol: the main entry point receives *commands* which are then routed to the proper place using *codes*.

The protocol command flow starts at the `CrocSwapDex` contract:

https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/CrocSwapDex.sol#L103-L106

```
103:     function protocolCmd (uint16 callpath, bytes calldata c
104:         protocolOnly(sudo) public payable override {
105:         callProtocolCmd(callpath, cmd);
106:     }
```

https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/mixins/ProxyCaller.sol#L22-L28

```
22:     function callProtocolCmd (uint16 proxyIdx, bytes calldat
23:         returns (bytes memory) {
24:         assertProxy(proxyIdx);
25:         (bool success, bytes memory output) = proxyPaths_[pr
26:             abi.encodeWithSignature("protocolCmd(bytes)", ir
27:         return verifyCallResult(success, output);
28:     }
```

The `protocolCmd()` function checks the call is properly authorized and calls `callProtocolCmd()`, which then executes a `delegatecall` to the corresponding implementation, in this case the `LiquidityMiningPath` contract:

https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol#L26-L37

```
26:     function protocolCmd(bytes calldata cmd) public virtual
27:         (uint8 code, bytes32 poolHash, uint32 weekFrom, uint
28:             abi.decode(cmd, (uint8, bytes32, uint32, uint32,
29:
30:         if (code == ProtocolCmd.SET_CONC_REWARDS_CODE) {
31:             setConcRewards(poolHash, weekFrom, weekTo, weekl
32:         } else if (code == ProtocolCmd.SET_AMB_REWARDS_CODE)
33:             setAmbRewards(poolHash, weekFrom, weekTo, weekly
34:         } else {
35:             revert("Invalid protocol command");
36:         }
37:     }
```

While `CrocSwapDex::protocolCmd()` is payable, its counterpart in `LiquidityMiningPath` is not. This means that rewards cannot be transferred while setting them up, rejecting any command that has positive `callvalue`.

Recommendation

Make `LiquidityMiningPath::protocolCmd()` payable.

```
-   function protocolCmd(bytes calldata cmd) public virtual {
+   function protocolCmd(bytes calldata cmd) public virtual paya
        (uint8 code, bytes32 poolHash, uint32 weekFrom, uint32 w
            abi.decode(cmd, (uint8, bytes32, uint32, uint32, uir

        if (code == ProtocolCmd.SET_CONC_REWARDS_CODE) {
            setConcRewards(poolHash, weekFrom, weekTo, weeklyRew
        } else if (code == ProtocolCmd.SET_AMB_REWARDS_CODE) {
            setAmbRewards(poolHash, weekFrom, weekTo, weeklyRewa
        } else {
            revert("Invalid protocol command");
        }
    }
```

Assessed type

Payable

141345 (lookout) commented:

> `protocolCmd()` should be payable.

> One walk around is to call `setAmbRewards()` / `setConcRewards()` directly to send funds. But seems the expected way is to use this contract through `delegatecall`, if the contract is deployed without this option, the described issue could be some problem.

[OpenCoreCH (Canto) confirmed]

[M-04] Accrued liquidity can be lost and never be recovered for a given tick period.

Submitted by [Satyam_Sharma]

Accrued liquidity can be lost and never be recovered for a given tick period due to not incrementing `tickTrackingIndex`, which sets the tick end timestamp to `nextweek` and will start accruing liquidity for next tick period.

Proof of Concept

`LiquidityMining.accrueConcentratedPositionTimeWeightedLiquidity` sets `tickActiveEnd = nextWeek` and doesn't increment `tickTrackingIndex`, which makes current tick period to end by setting `tickActiveEnd = nextWeek` and moves to next tick period without incrementing `tickTrackingIndex`. When tick is no longer active that is `tickTracking.exitTimestamp < nextWeek`, it means `tickTracking.exitTimestamp` should be strictly less than the `nextWeek` to set `tickActiveEnd = tickTracking.exitTimestamp` and then calculate `dt` based on the `tickActiveStart` and `tickActiveEnd` values.

```
    else {
                                // Tick is no longer active
                                if (tickTracking.exitTimestamp < nex
                                    // Exit was in this week, contir
                                    tickActiveEnd = tickTracking.exi
                                    tickTrackingIndex++;
                                    dt = tickActiveEnd - tickActiveS
                                } else {
```

```
                // Exit was in next week, we nee
                tickActiveEnd = nextWeek;
            }
```

Now the issue here, is with the `if` statement; `if (tickTracking.exitTimestamp < nextWeek)` calculates `dt` if `exitTimestamp` is strictly lesser than the `nextWeek`. Suppose there is a condition in which user `exitTimestamp = nextWeek` i.e.; user exits the concentrated liquidity position with some `X` poolIdx with the `nextWeek` timestamp. Meaning when the `nextWeek` is about to end, the user triggers an exit and as soon as the exit triggers, this `if` statement will revert and `tickTrackingIndex` will not be incremented for that user or poolId. `tickTrackingIndexAccruedUpTo_[poolIdx][posKey][i]` and therefore, the accrued concentrated liquidity might be lost for a given tick period and will never be recovered due to setting `tickActiveEnd = nextWeek` in the `else` statement; which will declare the tick end and move to the next tick period to accrued liquidity .

Recommended Mitigation Steps

Edit the `if` statement for `exitTimestamp`, which increments `tickTrackingIndex` when `tickTracking.exitTimestamp == nextWeek`:

```
- if (tickTracking.exitTimestamp < nextWeek) {
            }
+  if (tickTracking.exitTimestamp <= nextWeek) {
            }
```

Assessed type

Error

[141345 (lookout) commented](#):

> An edge case when `exitTimestamp == nextWeek`, rewards could be lost

```
File: canto_ambient/contracts/mixins/LiquidityMining.sol
24:     function crossTicks() internal {
```

```
29:         uint256 numElementsExit = tickTracking_[poolIdx][exi
30:         tickTracking_[poolIdx][exitTick][numElementsExit - 1
31:             .exitTimestamp = uint32(block.timestamp);
```

> Due to the low likelihood, high severity might not be appropriate.

OpenCoreCH (Canto) confirmed and commented:

> Good point, requires quite a few conditions to actually cause damage (exactly aligned `exitTimestamp` and even then, the accrual has to happen a few weeks later with additional ticks that were in range since then for it to cause skipping ticks), but will definitely be fixed.

LSDan (judge) decreased severity to Medium

Note: for full discussion, see here.

[M-05] Positions that are not eligible for rewards will affect the reward income of eligible positions

Submitted by kutugu, also found by sces60107 and BanditxOx (1, 2)

Lines of code

https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/mixins/LiquidityMining.sol#L181
https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/mixins/LiquidityMining.sol#L188
https://github.com/code-423n4/2023-10-canto/blob/40edbe0c9558b478c84336aaad9b9626e5d99f34/canto_ambient/contracts/mixins/LiquidityMining.sol#L48

Impact

When calculating reward distribution, the contract uses the current position's liquidity time weight divided by the total liquidity time weight of the pool, instead of the total liquidity time weight of the pool that meets the reward conditions. This

means that if there is a large amount of liquidity in the pool that is not eligible for rewards, the total weekly reward distribution will be much less than the `rewardPerWeek`, which should not be expected.

Proof of Concept

```
diff --git a/canto_ambient/test_canto/TestLiquidityMining.js b/c
index bd21a32..617b070 100644
--- a/canto_ambient/test_canto/TestLiquidityMining.js
+++ b/canto_ambient/test_canto/TestLiquidityMining.js
@@ -32,7 +32,7 @@ chai.use(solidity);

 describe("Liquidity Mining Tests", function () {
        it("deploy contracts and init pool", async function () {
-               const [owner] = await ethers.getSigners();
+               const [owner, others] = await ethers.getSigners(

                ////////////////////////////////////////////
                // DEPLOY AND MINT cNOTE and USDC
@@ -49,6 +49,14 @@ describe("Liquidity Mining Tests", function (
                        owner.address,
                        ethers.utils.parseUnits("1000000", 6)
                );
+               await cNOTE.deposit(
+                       others.address,
+                       ethers.utils.parseEther("1000000")
+               );
+               await USDC.deposit(
+                       others.address,
+                       ethers.utils.parseUnits("1000000", 6)
+               );

                ////////////////////////////////////////////
                // DEPLOY DEX CONTRACT AND ALL PROXIES
@@ -145,6 +153,17 @@ describe("Liquidity Mining Tests", functior
                );
                await approveCNOTE.wait();

+               approveUSDC = await USDC.connect(others).approve(
+                       dex.address,
+                       BigNumber.from(10).pow(36)
+               );
+               await approveUSDC.wait();
```

```
+                approveCNOTE = await cNOTE.connect(others).appro
+                        dex.address,
+                        BigNumber.from(10).pow(36)
+                );
+                await approveCNOTE.wait();
+
                 /*
         /       2. set new pool liquidity (amount to lock up for
         /          params = [code, liq]
@@ -222,6 +241,40 @@ describe("Liquidity Mining Tests", functior
                 });
                 await tx.wait();

+                mintConcentratedLiqCmd = abi.encode(
+                        [
+                                "uint8",
+                                "address",
+                                "address",
+                                "uint256",
+                                "int24",
+                                "int24",
+                                "uint128",
+                                "uint128",
+                                "uint128",
+                                "uint8",
+                                "address",
+                        ],
+                        [
+                                11, // code (mint concentrated l
+                                cNOTE.address, // base token
+                                USDC.address, // quote token
+                                36000, // poolIDX
+                                currentTick - 5, // tickLower
+                                currentTick + 5, // tickUpper
+                                BigNumber.from("100000000000000(
+                                BigNumber.from("166020696663385(
+                                BigNumber.from("202914184810805(
+                                0, // reserve flag
+                                ZERO_ADDR, // lp conduit address
+                        ]
+                );
+                tx = await dex.connect(others).userCmd(2, mintCo
+                        gasLimit: 6000000,
+                        value: ethers.utils.parseUnits("10", "et
+                });
+                await tx.wait();
```

```
+
                ////////////////////////////////////////////
                // SAMPLE SWAP TEST (swaps 2 USDC for cNOTE)
                ////////////////////////////////////////////
@@ -300,9 +353,9 @@ describe("Liquidity Mining Tests", function
                const dexBalAfter = await ethers.provider.getBal
                const ownerBalAfter = await ethers.provider.getB

-               // expect dex to have 2 less CANTO since we clai
+               // @audit Expect to have less CANTO rewards due
                expect(dexBalBefore.sub(dexBalAfter)).to.equal(
-                       BigNumber.from("20000000000000000")
+                       BigNumber.from("5014124016834945 42")
                );
        });
 });
```

In the pool, there is only one position that meets the reward conditions, and there is also one position that does not meet the conditions.

According to common sense, weekly rewards should be distributed to the only position that meets the reward conditions based on `rewardPerWeek`. However, in reality, you can see that the rewards that should be distributed every week have been reduced to `1/4` due to the ineligible position.

Tools Used

Hardhat

Recommended Mitigation Steps

Since the reward distribution of concentrated type rewards is conditionally restricted, the conditional restrictions should also be taken into account when calculating the total weight, rather than directly counting the total amount of liquidity.

Assessed type

Context

[OpenCoreCH (Canto) disputed and commented via duplicate Issue #94](#):

> Responding to this and #98 here:
>
> The warden seems to assume that `curve.concLiq_` is influenced by the width of a position, which is not the case. If someone creates a position with range [0, 1000] and liq. 100 or [480, 520] with liq. 100, `curve.concLiq_` will be 100 in both cases when the active tick is 500 (and not 0.1 & 2.5, which seems to be the assumption of the warden). This can be seen in `TradeMatcher`, `LevelBook`, and `LiquidityMath`. The logic is not that simple, but the high level summary is that we store for every tick the liquidity (lots) of positions that have the lower / upper tick here (`lvl.bidLots_`, `lvl.askLots_`). When a tick is then crossed, the whole liquidity is added or removed from `curve.concLiq_`.
>
> However, it can also be easily verified by looking at our test. If the assumption of the warden were true, the user in the test (that has a position with width 30) would not receive 100% of the rewards, but only `1/30`. Because `concLiq_` works this way, the recommendation of the warden would break the system.
>
> Something that is true is point 1 from #98. If a user has a very narrow position (less than 21 ticks), it still contributes to `curve.concLiq_`.

LSDan (judge) commented:

> TL;DR - in some cases reward amounts may be miscalculated, resulting in lower than expected reward amounts.

Low Risk and Non-Critical Issues

For this audit, 37 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **adriro** received the top score from the judge.

The following wardens also submitted reports: radev_sw, albahaca, MatricksDeCoder, JP_Courses, 0xDING99YA, 0xdice91, Mike_Bello90, 0xTheC0der, Eurovickk, marqymarq10, orion, sces60107, wahedtalash77, xAriextz, BoRonGod, gzeon, SovaSlava, matrix_0wl, 3docSec, taner2344, Topmark, 0x3b, cookedcookee, HChang26, 100su, zpan, GKBG, BRONZEDISC,

lukejohn, IceBear, hunter_w3b, 0xAadi, pep7siup, kutugu, tpiliposian, *and* 0xWaitress.

Low Issue Summary

ID	Issue
[L-01]	`claimConcentratedRewards()` and `claimAmbientRewards()` should be an internal function
[L-02]	Governance check is commented out
[L-03]	Rewards can be unintentionally overridden
[L-04]	Unused rewards cannot be claimed back
[L-05]	Missing week validation while claiming rewards

Non Critical Issue Summary

ID	Issue
[N-01]	Use Solidity time units

Low Issues

[L-01] `claimConcentratedRewards()` and `claimAmbientRewards()` should be an internal function

- https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol#L54
- https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol#L61

Both of these functions present in the `LiquidityMiningPath` contract are intended to be called via user commands, whose entry point is the `userCmd()` function.

Consider changing the visibility of `claimConcentratedRewards()` and `claimAmbientRewards()` to internal or private.

[L-02] Governance check is commented out

- https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol#L66
- https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol#L75

Both `setConcRewards()` and `setAmbRewards()` have a privilege check that is commented out and currently ignored:

```
// require(msg.sender == governance_, "Only callable by governar
```

These functions are called as protocol commands, which undergo a privilege check in the entry point (see here), hence the low severity of this issue.

However, it is not clear if there's a missing additional check to ensure these are called by the governance. Consider either removing the lines or un-commenting the check.

[L-03] Rewards can be unintentionally overridden

- https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol#L65
- https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/callpaths/LiquidityMiningPath.sol#L74

Lidiquity mining rewards are set up using `setConcRewards()` or `setAmbRewards()`. In both cases, the rewards are overridden instead of accumulated. Taking `setConcRewards()` as the example:

```
65:     function setConcRewards(bytes32 poolIdx, uint32 weekFrom
66:         // require(msg.sender == governance_, "Only callable
67:         require(weekFrom % WEEK == 0 && weekTo % WEEK == 0,
68:         while (weekFrom <= weekTo) {
69:             concRewardPerWeek_[poolIdx][weekFrom] = weeklyRe
70:             weekFrom += uint32(WEEK);
71:         }
72:     }
```

Line 69 assigns the new value `weeklyReward` to the storage mapping. If rewards were already set for this pool, i.e. `concRewardPerWeek_[poolIdx][weekFrom] != 0`, then the new assignment will override the existing value.

[L-04] Unused rewards cannot be claimed back

Although highly unlikely, there is no provided mechanism to claim back unused rewards assigned to periods of time of inactivity in the pool.

[L-05] Missing week validation while claiming rewards

- https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/mixins/LiquidityMining.sol#L156
- https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/mixins/LiquidityMining.sol#L256

In `claimConcentratedRewards()` and `claimAmbientRewards()`, the implementation takes an array of the weeks which are expected to be claimed. This is user input, and lacks any validation over the provided argument values.

Using `claimConcentratedRewards()` as the example, we can see each element in `weeksToClaim` is not checked to be an actual *week*, i.e. that `week % WEEK == 0`.

```
174:         for (uint256 i; i < weeksToClaim.length; ++i) {
175:             uint32 week = weeksToClaim[i];
176:             require(week + WEEK < block.timestamp, "Week nc
177:             require(
178:                 !concLiquidityRewardsClaimed_[poolIdx][posK
179:                 "Already claimed"
```

```
180:             );
181:             uint256 overallInRangeLiquidity = timeWeightedW
```

Consider adding an explicit check to ensure the elements in `weeksToClaim` are valid weeks.

```
        uint32 week = weeksToClaim[i];
        require(week + WEEK < block.timestamp, "Week not over yet");
+       require(week % WEEK == 0, "Invalid week");
```

Non Critical Issues

[N-01] Use Solidity time units

- https://github.com/code-423n4/2023-10-canto/blob/main/canto_ambient/contracts/mixins/LiquidityMining.sol#L13

Instead of defining spans of time manually using seconds, consider using Solidity built-in time units.

OpenCoreCH (Canto) confirmed

LSDan (judge) commented:

> I agree with all reported issues and their severity.

Gas Optimizations

For this audit, 16 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **niser93** received the top score from the judge.

The following wardens also submitted reports: 0xAnah, JCK, hihen, naman1778, SAQ, SY_S, tabriz, shamsulhaq123, pipidu83, Raihan, lsaudit, hunter_w3b, MatricksDeCoder, Polaris_tow, *and* debo.

Gas Optimizations

ID	Issue	Instances	Total Gas Saved
[G-01]	Remove ternary operator in order to use unchecked and `>=` instead of `<`	1	4526
[G-02]	State variable read in a loop	5	1908
[G-03]	Using ternary operator instead of `if/else`	1	1896
[G-04]	Using a positive conditional flow to save a NOT opcode	1	2750

Total: 8 instances over 4 issues with **11080 gas** saved.

Gas totals are estimates using `npx hardhat test` and `hardhat-gas-reporter`.

[G-01] Remove ternary operator in order to use unchecked and `>=` instead of `<`

Estimated saving: | Method call or Contract deployment | Before | After | After - Before | (After - Before) / Before | | :- | :-: | :-: | :-: | :-: | | `LiquidityMiningPath` | 1540432 | 1535906 | -4526 | -0.29% |

There is 1 instance of this issue:

[53-57]

```
File: LiquidityMining.sol

-   uint32 dt = uint32(
-       nextWeek < block.timestamp
-           ? nextWeek - time
-           : block.timestamp - time
-   );
+   uint32 dt;
+   if(nextWeek >= block.timestamp) {
+       unchecked{dt = uint32(block.timestamp - time);}
+   }
+   else{
```

```
+           unchecked{dt = uint32(nextWeek - time);}
+       }
```

[G-02] State variable read in a loop

The state variable should be cached in a local variable rather than reading it on every iteration of the `for-loop`, which will replace each Gwarmaccess (**100 gas**) with a much cheaper stack read.

Estimated saving: | Method call or Contract deployment | Before | After | After - Before | (After - Before) / Before | | :- | :-: | :-: | :-: | :-: | | `LiquidityMiningPath` | 1540432 | 1538524 | -1908 | -0.12% |

There are 5 instances of this issue:

[87-97]

```
File: LiquidityMining.sol

        uint256 liquidity = pos.liquidity_;
+       uint256 wweek = WEEK;
        for (int24 i = lowerTick + 10; i <= upperTick - 10; ++i) {
            uint32 tickTrackingIndex = tickTrackingIndexAccruedUpTo_
            uint32 origIndex = tickTrackingIndex;
            uint32 numTickTracking = uint32(tickTracking_[poolIdx][i
            uint32 time = lastAccrued;
            // Loop through all in-range time spans for the tick or
            while (time < block.timestamp && tickTrackingIndex < num
                TickTracking memory tickTracking = tickTracking_[poo
-                   uint32 currWeek = uint32((time / WEEK) * WEEK);
-                   uint32 nextWeek = uint32(((time + WEEK) / WEEK) *
+                    uint32 currWeek = uint32((time / WEEK) * WEEK);
+                    uint32 nextWeek = uint32(((time + WEEK) / WEEK)
```

[G-03] Using ternary operator instead of `if/else`

Estimated saving: | Method call or Contract deployment | Before | After | After - Before | (After - Before) / Before | | :- | :-: | :-: | :-: | :-: | | `LiquidityMiningPath` | 1540432 | 1538536 | -1896 | -0.12% |

There is 1 instance of this issue:

[53-57]

```
File: LiquidityMining.sol

-   if (tickTracking.enterTimestamp < time) {
-       // Tick was already active when last claim happened, onl
-       tickActiveStart = time;
-   } else {
-       // Tick has become active this week
-       tickActiveStart = tickTracking.enterTimestamp;
-   }
+   tickActiveStart = tickTracking.enterTimestamp >= time ? time
```

[G-04] Using a positive conditional flow to save a NOT opcode

In order to save some gas (NOT opcode costing 3 gas), switch to a positive statement:

```
- if(!condition){
-     action1();
- }else{
-     action2();
- }
+ if(condition){
+     action2();
+ }else{
+     action1();
+ }
```

Estimated saving: | Method call or Contract deployment | Before | After | After - Before | (After - Before) / Before | | :- | :-: | :-: | :-: | :-: | | `LiquidityMiningPath` | 1540432 | 1537682 | -2750 | -0.18% |

There is 1 instance of this issue:

[86-150]

▶ Details

Audit Analysis

For this audit, 11 analysis reports were submitted by wardens. An analysis report examines the codebase as a whole, providing observations and advice on such topics as architecture, mechanism, or approach. The **report highlighted below** by **0xweb3boy** received the top score from the judge.

The following wardens also submitted reports: hunter_w3b, sandy, 0xdice91, radev_sw, BanditxOx, JP_Courses, ZanyBonzy, cookedcookee, albahaca, *and* invitedtea.

Table of Contents

1. Executive Summary

In focusing on the ongoing audit 2023-10-canto, my analysis starts by delineating the code audit methodology applied to the contracts within the defined scope. Subsequently, I provided insights into the architectural aspects, offering my perspective. Finally, I offered observations pertaining to the code implementation.

I want to emphasize that unless expressly specified, any potential architectural risks or implementation concerns discussed in this document should not be construed as vulnerabilities or suggestions to modify the architecture or code based solely on this analysis. As an auditor, I recognize the necessity of a comprehensive evaluation of design choices in intricate projects, considering risks as only one component of a larger evaluative process. It's essential to acknowledge that the project team may have already evaluated these risks and established the most appropriate approach to mitigate or coexist with them.

2. Code Audit Approach

2.1 Audit Documentation and Scope

Commencing the analysis, the first phase entailed a thorough review of the repo to fully grasp the fundamental concepts and limitations of the audit. This initial step was crucial in guiding the prioritization of my audit efforts. Notably, the README associated with this audit stands out for its excellent quality, offering valuable insights and actionable guidance that significantly streamline the onboarding process for auditors.

2.2 Code review

Initiating the code review, the starting point involved gaining a comprehensive understanding of "LiquidityMining.sol," a pivotal component responsible for implementing a liquidity mining protocol for Ambient. Canto's strategic intention is to leverage this sidecar mechanism to incentivize liquidity provision for Ambient pools deployed on their platform. This understanding of the core pattern significantly facilitated the comprehension of the protocol contracts and their interconnections. During this phase, meticulous documentation of observations and the formulation of pertinent questions regarding potential exploits were undertaken, striking a balance between depth and breadth of analysis.

2.3 Threat Modelling

The initial step involved crafting precise assumptions that, if breached, could present notable security risks to the system. This approach serves to guide the identification of optimal exploitation strategies. Although not an exhaustive threat modeling exercise, it closely aligns with the essence of such an analysis.

2.4 Exploitation and Proofs of Concept

Progressing from this juncture, the primary methodology took the form of a cyclic process, conditionally encompassing steps 2.2, 2.3, and 2.4. This involved iterative attempts at exploitation and the subsequent creation of proofs of concept, occasionally aided by available documentation or the helpful community on Discord. The key focus during this phase was to challenge fundamental assumptions, generate novel ones in the process, and refine the approach by utilizing coded proofs of concept to hasten the development of successful exploits.

2.5 Report Issues

While this particular stage might initially appear straightforward, it harbors subtleties worth considering. Hastily reporting vulnerabilities and subsequently overlooking them is not a prudent course of action. The optimal approach to augment the value delivered to sponsors (and ideally, to auditors as well) entails thoroughly documenting the potential gains from exploiting each vulnerability. This comprehensive assessment aids in the determination of whether these exploits could be strategically amalgamated to generate a more substantial impact on the system's security. It's important to recognize that seemingly minor and moderate issues, when skillfully leveraged, can compound into a critical vulnerability. This assessment must be weighed against the risks that users might encounter. Within the realm of Code4rena audits, a heightened level of caution and an expedited reporting channel are accorded to zero-day vulnerabilities or highly sensitive bugs impacting deployed contracts.

3. Architecture overview

3.1 Overview of Liquidity Mining Feature by Canto for Ambient Finance

Introduction to the Feature: Canto, in its pursuit of enhancing liquidity dynamics, has introduced a new liquidity mining feature tailored specifically for Ambient Finance, a targeted approach to bolster the platform's liquidity infrastructure.

Integration through Sidecar Contract: The implementation strategy involves the creation of a sidecar contract meticulously designed to integrate into the Ambient ecosystem. This integration is achieved through the utilization of Ambient's proxy contract pattern, a structured approach to seamlessly amalgamate the new liquidity mining feature.

3.2 Key Contracts Introduced

LiquidityMiningPath.sol: This contract is pivotal in providing the essential interfaces, enabling seamless interactions for users interacting with the liquidity mining protocol.

LiquidityMining.sol: This contract forms the operational backbone, encapsulating the logic and functionalities necessary for the effective operation of the liquidity mining feature.

Understanding the LiquidityMining Sidecar
Objective of the Sidecar: The LiquidityMining sidecar is meticulously engineered to realize a robust liquidity mining protocol within the Ambient ecosystem. Canto envisions utilizing this sidecar to stimulate and incentivize liquidity contributions to the Ambient pools hosted on Canto's platform.

3.3 Incentive Mechanism

The sidecar employs an incentivization mechanism targeting a specific width of liquidity, primarily based on the current tick. Focused on stabilizing liquidity pools, the incentivization range spans from the current tick minus 10 to the current tick plus 10.

To qualify for incentives, a user's liquidity range must encompass at least 21 ticks, inclusive of the current tick and 10 ticks on each side. Additionally, users are advised to maintain a slight buffer on either side of the stipulated range to ensure uninterrupted rewards, particularly in response to minor price fluctuations.

3.4 Liquidity Mining Rewards

The core idea behind the liquidity mining sidecar centers on tracking the time-weighted liquidity across global and per-user levels. This tracking is conducted for both ambient and concentrated positions on a per-tick basis.

The protocol calculates the user's proportion of in-range liquidity over a specific time span, subsequently disbursing this percentage of the global rewards to the user.

3.5 Reward Distribution Example

For illustrative purposes, if the weekly rewards amount to 10 CANTO and only LP A is contributing liquidity, "LP A" will receive the entire 10 CANTO as their reward. However, if there are multiple liquidity providers, such as "LP A" and "LP B", who contribute liquidity throughout the week, the rewards will be shared evenly. In this scenario, each provider will receive 5 CANTO. Implementation Insights

3.6 Setting Weekly Reward Rate

The liquidity mining sidecar incorporates functions dedicated to setting the weekly reward rate. The reward rates are determined by establishing a total disbursement amount per week, providing the governing body the flexibility to choose the number of weeks for which the reward rate will be set.

Focus on LiquidityMining.sol: The critical aspect of the implementation lies within the LiquidityMining.sol contract, housing the core logic for reward accumulation and claim processes. Auditors and wardens are strongly advised to direct their primary focus towards this segment of the codebase, recognizing its pivotal role in the successful operation of the liquidity mining feature.

4. Implementation Notes

During the course of the audit, several noteworthy implementation details were identified, and among these, a significant subset holds potential value for the ongoing analysis.

4.1 General Impressions

Overview of Ambient

Ambient Finance: Ambient is a single-contract decentralized exchange (dex) designed to facilitate liquidity provision in a flexible manner. It allows liquidity providers to deposit liquidity in either the "ambient" style (similar to uniV2) or the "concentrated" style (similar to uniV3) into any token pair.

Main Contract: The primary contract in Ambient Finance is called `CrocSwapDex`. Users interact with this contract, and it is the main interface for all actions related to the protocol.

Sidecar Contracts

Ambient utilizes modular proxy contracts called “sidecars,” each responsible for a unique function within the protocol.

Here are the sidecar contracts:

- BootPath: Special sidecar used to install other sidecar contracts.
- ColdPath: Handles the creation of new pools.
- WarmPath: Manages liquidity operations like minting and burning.
- LiquidityMiningPath: New sidecar developed by Canto to handle liquidity mining for both ambient and concentrated liquidity.
- KnockoutPath: Manages logic for knockout liquidity.
- LongPath: Contains logic for parsing and executing arbitrarily long compound orders.
- MicroPaths: Contains functions related to single atomic actions within a longer compound action on a pre-loaded pool’s liquidity curve.
- SafeModePath: Reserved for emergency mode.

Interacting with Ambient Contracts

User and Protocol Commands: Interaction with Ambient contracts is facilitated through two main functions:

- `userCmd(uint16 callpath, bytes calldata cmd)` : Used by users for various actions.
- `protocolCmd(uint16 callpath, bytes calldata cmd, bool sudo)` : Utilized for governance-related actions.

`Callpath Parameter` : The callpath parameter determines which sidecar contract receives the command. Different values correspond to different sidecar contracts, each serving a specific purpose.

`Command Encoding (cmd)` : The cmd parameter is ABI encoded calldata fed to the specified sidecar contract. The code parameter within cmd specifies the function to be called within the sidecar contract based on the desired action.

4.2 Composition over Inheritance

Canto has used inheritance over composition because inheritance is a prevalent choice due to the need for standardized behaviors and code reusability. Smart contracts often adhere to established standards such as ERC-20 or ERC-721, and inheritance facilitates the straightforward integration of these standards, promoting interoperability and adherence to well-defined interfaces. By centralizing common functionalities in a base contract and allowing other contracts to inherit these functionalities, developers can streamline deployment, reduce redundancy, and simplify interactions. In addition, this approach aids in efficient gas usage, aligning with the limitations imposed by the Ethereum Virtual Machine (EVM) contract size. Moreover, inheritance supports logical code organization and maintenance, enabling easier updates and enhancing readability by segregating related functionalities into distinct contracts.

4.3 Comments

Importance of Comments for Clarity

Comments serve a pivotal role in enhancing the understandability of the codebase. While the code is generally clean and logically structured, judiciously placed comments can provide valuable insights into the functionalities and intentions behind the code. They contribute to a better comprehension of the code's purpose, especially for auditors and developers involved in the analysis.

Strategic Comment Placement

The codebase would greatly benefit from an increased presence of comments, particularly within the `LiquidityMining.sol` contract. Strategic placement of comments within this essential contract can significantly aid auditors in comprehending the implementation details. These comments should elucidate the logic, processes, and methodologies employed, promoting a seamless audit experience.

Facilitating Audits and Code Readability

Comprehensive comments in the `LiquidityMining.sol` contract can expedite the auditing process by allowing auditors to swiftly grasp the intended functionality of the code. This, in turn, enhances the readability of the functions and methods, making it easier for auditors to identify any inconsistencies or deviations between the documented intentions and the actual code implementation.

Detecting Discrepancies in Code Intentions

An important aspect of code review is discerning any mismatches between the documented intentions in the comments and the actual code implementation. Comments that accurately reflect the code's purpose are crucial for auditors, as discrepancies between the two can be indicators of potential vulnerabilities or errors. The act of aligning comments with the true code behavior is fundamental to ensuring the reliability and security of the smart contract.

4.4 Solidity Versions

Though there are valid arguments both in support of and against adopting the latest Solidity version, I find that this discussion bears little significance for the current state of the project. Without a doubt, choosing the most up-to-date version is a far superior decision when compared to the potential risks associated with outdated versions.

5. Conclusion

Positive Audit Experience

The process of auditing this codebase and evaluating its architectural choices has been thoroughly enjoyable and enriching. Navigating through the intricacies and nuances of the project has been enlightening, presenting an opportunity to delve into the complexities of the system.

Strategic Simplifications for Complexity Management

Inherent complexity is a characteristic of many systems, especially those in the realm of blockchain and smart contracts. The strategic introduction of simplifications within this project has proven to be a valuable approach. These simplifications are well-thought-out and strategically implemented, demonstrating an understanding of how to manage complexity effectively.

Achieving a Harmonious Balance

A notable achievement of this project is striking a harmonious balance between the imperative for simplicity and the challenge of managing inherent complexity. This equilibrium is crucial in ensuring that the codebase remains comprehensible, maintainable, and scalable, even as the system becomes more intricate.

Importance of Methodology Overview

The overview provided regarding the methodology employed during the audit of the contracts within the defined scope is invaluable. It offers a clear and structured insight into the analytical approach undertaken, shedding light on the depth and rigor of the evaluation process.

Relevance for Project Team and Stakeholders

The insights presented are not only beneficial for the project team but also extend to any party with an interest in analyzing this codebase. The detailed observations, considerations, and recommendations have the potential to guide and inform decision-making, contributing to the project's overall improvement and security.

Time spent

16 hours

OpenCoreCH (Canto) acknowledged

Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top