Canto Dex Oracle contest Findings & Analysis Report

2023-03-31

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 contest 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 contest outlined in this document, C4 conducted an analysis of the Canto Dex Oracle smart contract system written in Solidity. The audit contest took place between September 7—September 8 2022.

Wardens

70 Wardens contributed reports to the Canto Dex Oracle contest:

1. Critical
2. csanuragjain
3. Respx
4. linmiaomiao
5. hickuphh3
6. __141345__
7. sorrynotsorry
8. Jeiwan
9. SinceJuly
10. Chom
11. 0xSmartContract
12. cccz
13. V_B (Barichek and vlad_bochok)
14. 0xNazgul
15. 0xSky
16. CertoraInc (egjlmn1, OriDabush, ItayG, shakedwinder, and RoiEvenHaim)
17. Deivitto
18. fatherOfBlocks
19. hansfriese
20. oyc_109
21. rbserver
22. rokinot
23. Tomo
24. 0xhunter
25. BipinSah
26. m_Rassska

27. prasantgupta52
28. Rohan16
29. Sm4rty
30. 0x040
31. 0x1f8b
32. 0x52
33. 0xA5DF
34. a12jmx
35. ajtra
36. ak1
37. Bnke0x0
38. Bronicle
39. codexploder
40. CodingNameKiki
41. cryptphi
42. Diraco
43. Dravee
44. erictee
45. EthLedger
46. gogo
47. hake
48. ignacio
49. IgnacioB
50. JansenC
51. JC
52. lukris02
53. ontofractal
54. p_crypt0
55. pashov

56. peritoflores
57. R2
58. rajatbeladiya
59. RaymondFam
60. ReyAdmirado
61. Rolezn
62. rvierdiiev
63. tnevler
64. TomJ
65. Yiko

This contest was judged by 0xean.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 7 unique vulnerabilities. Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity and 6 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 51 reports detailing issues with a risk rating of LOW severity or non-critical.

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 Dex Oracle contest repository, and is composed of 2 smart contracts written in the Solidity programming language and includes 2,001 lines of Solidity code.

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] Hardcoded USD pegs can be broken

Submitted by hickuphh3, also found by __141345__, Critical, linmiaomiao, and sorrynotsorry

The prices of USDC and USDT, which (I assume) are the underlying tokens of `cUSDC` and `cUSDT`, have been hardcoded to parity. Such practices are highly discouraged because while the likelihood of either stablecoin de-pegging is low, it is not zero.

Because of the UST debacle, the price of USDT dropped to `$0.95` before making a recovery.

Impact

Here is an example of how a lending protocol on Fantom was affected by such a depeg event because they hardcoded the value.

To quote philosopher George Santayana, *"Those who cannot remember the past are condemned to repeat it."*

Recommended Mitigation Steps

Consider using a price feed by trusted and established oracle providers like Chainlink, Band Protocol or Flux. The USDC/NOTE or USDT/NOTE price feed may

be used as well, but NOTE has its own volatility concerns.

Medium Risk Findings (6)

[M-01] unbounded loop length dos

Submitted by 0xhunter, also found by BipinSah, fatherOfBlocks, m_Rassska, oyc_109, prasantgupta52, Rohan16, rokinot, Sm4rty, and Tomo

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully: Due to the block gas limit, transactions can only consume a certain amount of gas. Either explicitly or just due to normal operation, the number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point.

By calling createPair function a pair will be pushed to `allPairs` array , an admin can call setPeriodSize function and set newPeriod for every pairs in `allpair` array , however by spamming createPair function the loop in setPeriodSize function may revert in case of hitting gas limit of the network . since there is no way to remove `allPairs` or decrease their length in setPeriodSize function , its possible to totally make it impossible to call the setPeriodSize function.

In order to fix the issue setPeriodSize function has to be able to become executed in multiple times in case of facing gas limit.

nivasan1 (Canto) disagreed wtih severity and commented:

> Given that it is not expected for admin to change the period size in the router often, we do not consider this a high-risk vulnerability. This would also cost an infinite amount of Canto if orchestrated by a single user.

0xean (judge) decreased severity to Medium and commented:

> I am not sure the frequency that this is set to be used matter, it does lead to the functionality being lost. However, I do not see it leading to a loss of user funds, so will downgrade to a medium severity.

> `This would also cost an infinite amt of Canto if orchestrated by a single user` <- I am also not sure I understand this point. Can you explain further? What are the transaction gas limits set on this network?

nivasan1 (Canto) commented:

> @0xean, the method mentioned requires admin privileges to call it. As such, in order for an address that isn't timelock to access this method a malicious governance proposal must be passed to call the method from timelock, or to pass governance privileges to the malicious address. Secondly, the malicious user would have to spend a significant amount of Canto in deploying and adding sufficient liquidity to the contracts desired. As such, the risk-rewards for this action makes it unclear why any user would attempt to do this.

0xean (judge) commented:

> I don't think this is necessarily an attack vector, simply a way that the intended logic of the contracts could fail. If Canto is wildly successful there could be sufficient pairs that this would fail, I think Medium is a reasonable severity.

[M-02] Calculated `token0TVL` may be zero under certain scenarios

Submitted by hickuphh3, also found by 0xNazgul, 0xSky, CertoraInc, Deivitto, hansfriese, Jeiwan, linmiaomiao, rbserver, and SinceJuly

```
uint token0TVL = assetReserves[i] * (prices[i] / decimals);
```

Because of the brackets, the division of `prices[i] / decimals` is executed before multiplication, causing `token0TVL` to potentially be zero.

Proof of Concept

Add the following test in `oracle.test.ts`. Note: `getPriceLP()` should have its visibility changed from internal to public as the test relies on it.

To summarise what the test is doing, a stablecoin of 24 decimals is deployed, whose address will be greater than the `note` address so that `token0 = note`. It will enter the following case:

```
if (pair.stable()) { // stable pairs will be priced in terms of
  if (token0 == note) { //token0 is the unit, token1 will be pri
      decimals = 10 ** (erc20(token1).decimals()); // we must nc
      prices = pair.sample(token1, decimals, 8, 1);
      (unitReserves, assetReserves) = pair.sampleReserves(8, 1);
```

such that the `prices`'s denomination is smaller than the stablecoin's decimals of 24.

To see the difference in test results, apply the recommended fix after running the test once. In essence, the LP's price will double from `999500000001499` to `1999999998838589`, which is expected since the LP token should be worth the combined value of both stablecoins.

```
it.only("will have 0 token0TVL", async () => {
  // NOTE: change getPriceLP() from internal to public so that f
  let tokenFactory = await ethers.getContractFactory("ERC20", de
  let stablecoin = await tokenFactory.deploy("STABLE","STABLE",e
  await stablecoin.deployed()
  // we want note to be token0
  // redeploy till it is
  while (stablecoin.address < note.address) {
      stablecoin = await tokenFactory.deploy("STABLE","STABLE",e
      await stablecoin.deployed()
  }
  // give token approvals to router
  let noteIn = ethers.utils.parseUnits("10000", "18")
  let stableIn = ethers.utils.parseUnits("10000", "24")
  await (await note.approve(router.address, ethers.constants.Max
  await (await stablecoin.approve(router.address, ethers.constar

  // borrow note
  await (await comptroller._supportMarket(cUsdc.address)).wait()
  // set collateral factors for cCanto
  await (await comptroller._setCollateralFactor(cUsdc.address, e
  // borrow note against usdc
  await (await comptroller.enterMarkets([cUsdc.address, cNote.ac
```

```
await (await usdc.approve(cUsdc.address, ethers.utils.parseUni
// supply usdc
await (await cUsdc.mint(ethers.utils.parseUnits("100000000", '
// borrow note
await (await cNote.borrow(ethers.utils.parseUnits("9000000", '

// add liquidity
await (await router.addLiquidity(
    note.address,
    stablecoin.address,
    true,
    noteIn,
    stableIn,
    0,
    0,
    dep.address,
    9999999999,
    )).wait()
// get pair address
let pairAddr = await factory.getPair(note.address, stablecoin.
pair = await ethers.getContractAt("BaseV1Pair", pairAddr)

//set period size to zero for instant observations
await (await factory.setPeriodSize(0)).wait()

// swap 10 times for price observations
for(var i = 0; i < 10; i++) {
    if (i % 2) {
        //swap 0.01 note for stable
        await (await router.swapExactTokensForTokensSimple(
            ethers.utils.parseUnits("10", "18"),
            0,
            note.address,
            stablecoin.address,
            true,
            dep.address,
            9999999999999
        )).wait()
    } else {
        //swap stable for note
        await (await router.swapExactTokensForTokensSimple(
            ethers.utils.parseUnits("10", "24"),
            0,
            stablecoin.address,
            note.address,
            true,
```

```
                dep.address,
                9999999999999
            )).wait()
        }
    }
    // check lpToken price
    // Actual price calculated is 999500000001499
    // But expected price (after removing brackets) is 1999999988
    console.log((await router.getPriceLP(pairAddr)).toString());
});
```

Recommended Mitigation Steps

```
- uint token0TVL = assetReserves[i] * (prices[i] / decimals);
+ uint token0TVL = assetReserves[i] * prices[i] / decimals;
```

nivasan1 (Canto) confirmed

0xean (judge) decreased severity to Medium

[M-03] Hackers can deploy token with respective name as the stable one to impersonate the stable token

Submitted by Chom, also found by 0xSmartContract, cccz, Jeiwan, linmiaomiao, SinceJuly, and V_B

Hackers can deploy tokens with respective names as the stable ones to impersonate the stable token. Then hackers can get profit from the malicious price oracle.

Proof of Concept

```
        string memory symbol = ctoken.symbol();
        if (compareStrings(symbol, "cCANTO")) {
            underlying = address(wcanto);
            return getPriceNote(address(wcanto), false);
        } else {
            underlying = address(ICErc20(address(ctoken)).underl
```

```
        }
        //set price statically to 1 when the Comptroller is retr
        if (compareStrings(symbol, "cNOTE")) { // note in terms
            return 1e18; // Stable coins supported by the lendir
        }
        else if (compareStrings(symbol, "cUSDT") && (msg.sender
            uint decimals = erc20(underlying).decimals();
            return 1e18 * 1e18 / (10 ** decimals); //Scale Price
        }
        else if (compareStrings(symbol, "cUSDC") && (msg.sender
            uint decimals = erc20(underlying).decimals();
            return 1e18 * 1e18 / (10 ** decimals); //Scale Price
        }
```

If hackers or malicious admin deploy a non-stable token but has “cNOTE”, “cUSDT”, or “cUSDC” as symbols, these tokens will act as a stable token while in fact, it isn’t.

Recommended Mitigation Steps

Use fixed address whitelisting instead for example `if (address(ctoken) == cUSDC_address) ...` where `cUSDC_address` is an immutable variable set on the constructor.

tkkwon1998 (Canto) disputed and commented:

> The warden is saying that malicious parties could deploy a token with the same name to impersonate other tokens, but that these tokens will use the same pricing methodology as the token it is impersonating.

> However, as this is an open EVM, anyone could deploy any token name with any underlying price method. I fail to see how this would be an issue, since everything is open source and users can see what they are doing.

0xean (judge) commented:

> What is the benefit the sponsor is trying to achieve by comparing a non unique string compared to a unique address?

> While everything is open source, it does seem like this is a pretty bad design choice when compared to the alternative solutions.

nivasan1 (Canto) commented:

> @0xean, the design choice presented here prevents from having multiple dependencies in the initialization of the oracle. Furthermore, the tokens that are referenced here must be supported by governance, so to override the actual prices would require a >51% voting power. As such, even if the exploit exists it is essentially impossible. As such, we do not believe that this is a high / med -risk issue.

0xean (judge) decreased severity to Medium and commented:

> Will downgrade to Medium as there are several external factors for this to become an issue.

[M-04] Period Size not updated on creating new Pair

Submitted by csanuragjain

The period size is not updated to current while creating a new pair. This means even if period size has been reduced from default value, this new pair will still point to the higher default value.

Proof of Concept

1. Assume Pair P1,P2 exists in BaseV1Factory with default period size as 1800
2. Admin decides to decrease the period size to 900 using setPeriodSize function

```
function setPeriodSize(uint newPeriod) external {
        require(msg.sender == admin);
        require(newPeriod <= MaxPeriod);

        for (uint i; i < allPairs.length; ) {
            BaseV1Pair(allPairs[i]).setPeriodSize(newPeriod);
            unchecked {++i;}
        }
    }
```

3. This changes period size of P1, P2 to 900

4. Admin creates a new Pair P3 using createPair function

```
function createPair(address tokenA, address tokenB, bool stable)
        require(tokenA != tokenB, "IA"); // BaseV1: IDENTICAL_AI
        (address token0, address token1) = tokenA < tokenB ? (to
        require(token0 != address(0), "ZA"); // BaseV1: ZERO_ADI
        require(getPair[token0][token1][stable] == address(0), '
        bytes32 salt = keccak256(abi.encodePacked(token0, token1
        (_temp0, _temp1, _temp) = (token0, token1, stable);
        pair = address(new BaseV1Pair{salt:salt}());
        getPair[token0][token1][stable] = pair;
        getPair[token1][token0][stable] = pair; // populate mapp
        allPairs.push(pair);
        isPair[pair] = true;
        emit PairCreated(token0, token1, stable, pair, allPairs.
    }
```

5. A new Pair is created but the period size is not updated which means P3's period size will be 1800 instead of 900 which is incorrect

Recommended Mitigation Steps

Add a new variable which stores the updated period size. Once a pair is created, update its period size using this new variable:

```
uint periodSizeUpdated=1800;

function setPeriodSize(uint newPeriod) external {
        ...
periodSizeUpdated=newPeriod;
    }

function createPair(address tokenA, address tokenB, bool stable)
...
BaseV1Pair(pair).setPeriodSize(newPeriod);

}
```

nivasan1 (Canto) confirmed

[M-05] `getUnderlyingPrice()` should return `0` when errored

Submitted by Critical

The `Comptroller` is expecting `oracle.getUnderlyingPrice` to return `0` for errors (Compound style returns, no revert). The current implementation will throw errors, resulting in the consumer of the oracle getting unexpected errors.

Proof of Concept

```
    function getUnderlyingPrice(CToken ctoken) external override vie
             address underlying;
            { //manual scope to pop symbol off of stack
            string memory symbol = ctoken.symbol();
            if (compareStrings(symbol, "cCANTO")) {
                underlying = address(wcanto);
                return getPriceNote(address(wcanto), false);
            } else {
                underlying = address(ICErc20(address(ctoken)).underl
            }
            //set price statically to 1 when the Comptroller is retr
            if (compareStrings(symbol, "cNOTE")) { // note in terms
                return 1e18; // Stable coins supported by the lendir
            }
            else if (compareStrings(symbol, "cUSDT") && (msg.sender
                uint decimals = erc20(underlying).decimals();
                return 1e18 * 1e18 / (10 ** decimals); //Scale Price
            }
            else if (compareStrings(symbol, "cUSDC") && (msg.sender
                uint decimals = erc20(underlying).decimals();
                return 1e18 * 1e18 / (10 ** decimals); //Scale Price
            }
            }

            if (isPair(underlying)) { // this is an LP Token
                return getPriceLP(IBaseV1Pair(underlying));
            }
            // this is not an LP Token
            else {
                if (isStable[underlying]) {
                    return getPriceNote(underlying, true); // value
```

```
            }

            return getPriceCanto(underlying) * getPriceNote(addr
        }
    }
```

The `Comptroller` is expecting `oracle.getUnderlyingPrice` to return `0` for errors (Compound style returns, no revert).

However, the current implementation will revert when errored:

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-periphery.sol#L549-L593

```
    function getPriceLP(IBaseV1Pair pair) internal view returns(uint
        uint[] memory supply = pair.sampleSupply(8, 1);
        uint[] memory prices;
        uint[] memory unitReserves;
        uint[] memory assetReserves;
        address token0 = pair.token0();
        address token1 = pair.token1();
        uint decimals;
```

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-core.sol#L271-L289

```
    function sampleSupply(uint points, uint window) public view retu
        uint[] memory _totalSupply = new uint[](points);

        uint lastIndex = observations.length-1;
        require(lastIndex >= points * window, "PAIR::NOT READY E
        uint i = lastIndex - (points * window); // point from wh
        uint nextIndex = 0;
        uint index = 0;
        uint timeElapsed;

        for(; i < lastIndex; i+=window) {
            nextIndex = i + window;
```

```
            timeElapsed = observations[nextIndex].timestamp - ob
            _totalSupply[index] = (observations[nextIndex].total
            index = index + 1;
        }

        return _totalSupply;
    }
```

Recommended Mitigation Steps

Consider using `try catch` and return 0 when errored.

nivasan1 (Canto) confirmed

[M-06] System is Vulnerable to Downtime and has no Checks for it

Submitted by Respx

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-core.sol#L224-L258

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-core.sol#L271-L289

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-core.sol#L187-L194

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-core.sol#L201-L222

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-periphery.sol#L487-L593

Description

There is insufficient resilience in the design for the case where there has been no call to `_update()` for a long time. Consider these possible scenarios:

1. A denial of service attack on the blockchain has prevented transactions occurring for a significant period of time.
2. An extreme spike in gas prices has prevented transactions occurring for a significant period of time.
3. Some unforeseen technical error takes the blockchain down, perhaps during an upgrade or fork.

In any of these scenarios, it is possible that the price of any non-stablecoin token might be prone to serious volatility, `$CANTO` in particular.

The system has no provision for this issue.

Impact

Consider the following attack scenario:

1. Assume a period of network downtime, perhaps a DOS attack.
2. Assume a large drop in the price of `$CANTO` during this time.
3. Assume attacker is able to queue a transaction to be executed as soon as network service resumes (perhaps through producing a block themselves or high gas pricing). This transaction uses a lending system that relies on this oracle, and that lending system calls `reserves()` to calculate the TWAP price of `$CANTO`. The values returned are out of date and far too high. The attacker is then able to borrow stablecoins against their `$CANTO` at too high a rate.

Proof of Concept

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-core.sol#L224-L258

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-

core.sol#L271-L289

`reserves()`, `sampleReserves()` and `sampleSupply()` make no use of `block.timestamp`. They all measure only the time duration of the observations. There is no awareness of how much time has passed since the most recent observation was made.

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-core.sol#L187-L194

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-core.sol#L201-L222

https://github.com/code-423n4/2022-09-canto/blob/65fbb8b9de22cf8f8f3d742b38b4be41ee35c468/src/Swap/BaseV1-periphery.sol#L487-L593

`quote()` and `sample()` have the same logic, and these functions are called in `getPriceCanto()` and `getPriceNote()`. `getPriceLP()` relies on `sampleReserves()`. They therefore have the same issue.

Recommended Mitigation Steps

The system could track the average duration time of observations and, if any of the observations in a sample are significantly greater than this average, the system could either refuse to return a sample, or could return a warning flag to indicate that the sample data could be unreliable.

There is precedent in the system for refusing to return a sample (see line 242 of BaseV1-core.sol).

nivasan1 (Canto) disputed and commented:

> The oracle has been designed specifically for the purpose of use in the lending market, for pairs that have seen a long period of down-time (which is highly unexpected for supported pairs) the collateral factors can be adjusted to scale the

> price of the asset's collateral value downwards to reflect the volatility after the long period of downtime.

0xean (judge) decreased severity to Medium and commented:

> This has a high level of external factors that are required to be realized but knowing that the CANTO network has already suffered downtime, I don't think these are too far out of the realm of possibility. Downgrading to Medium.

Low Risk and Non-Critical Issues

For this contest, 51 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **lukris02** received the top score from the judge.

The following wardens also submitted reports: **p_crypt0**, **Dravee**, **hickuphh3**, **Deivitto**, **ajtra**, **BnkeOx0**, **Rolezn**, **tnevler**, **fatherOfBlocks**, **rvierdiiev**, **rbserver**, **ReyAdmirado**, **0xNazgul**, **erictee**, **oyc_109**, **0x52**, **Chom**, **Tomo**, **gogo**, **cryptphi**, **CodingNameKiki**, **SinceJuly**, **0x1f8b**, **peritoflores**, **TomJ**, **0xA5DF**, **codexploder**, **Bronicle**, **ignacio**, **Jeiwan**, **hansfriese**, **a12jmx**, **hake**, **R2**, **rokinot**, **RaymondFam**, **0xSky**, **ontofractal**, **pashov**, **csanuragjain**, **Diraco**, **rajatbeladiya**, **CertoraInc**, **ak1**, **JansenC**, **Yiko**, **IgnacioB**, **0x040**, **EthLedger**, **JC**.

Summary

№	Title	Risk Rating	Instance Count
L-01	Large number of observations may cause out-of-gas error	Low	2
L-02	Incorrect comment	Low	1
L-03	Misleading comment	Low	1
N-01	Order of Functions	Non-Critical	5
N-02	Maximum line length exceeded	Non-Critical	2+
N-03	Constants may be used	Non-Critical	18
N-0	Inconsistent comment spacing and location	Non-	1

№	Title	Risk Rating	Instance Count
4		Critical	
N-05	Loop parameter may be changed for clarity	Non-Critical	2
N-06	Functions without comments	Non-Critical	4
N-07	Require statement may be placed before allocating memory for arrays	Non-Critical	2
N-08	Check zero denominator	Non-Critical	2
N-09	Missing check for input variables	Non-Critical	2

[L-01] Large number of observations may cause out-of-gas error

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully: Due to the block gas limit, transactions can only consume a certain amount of gas. Either explicitly or just due to normal operation, the number of iterations in a loop can grow beyond the block gas limit, which can cause the complete contract to be stalled at a certain point.

Instances

- `for(; i < lastIndex; i+=window) {` (function sampleReserves)
- `for(; i < lastIndex; i+=window) {` (function sampleSupply)

Recommendation

Restrict the maximum number of sample observations (`points`).

[L-02] Incorrect comment

// note in terms of note will always be 1

Recommendation

Probably the comment should be like this “price in terms of note will always be 1”.

[L-03] Misleading comment

The comment is misleading, and there is an extra comma and an empty comment line.

Instances

Link:

```
            for (uint i = 0; i < _reserves0.length; ++i) {
                reserveAverageCumulative0 += _reserves0[i]; //normal
                reserveAverageCumulative1 += _reserves1[i]; //
            }
```

Recommendation

Change or delete comment.

[N-01] Order of Functions

Some internal functions are between public, some external functions are between public, and some public functions are between external.

Instances

- BaseV1-core.sol: 137
- BaseV1-core.sol: 224
- BaseV1-core.sol: 260
- BaseV1-core.sol: 237
- BaseV1-core.sol: 271

Recommendation

According to Style Guide, ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier.

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private

[N-02] Maximum line length exceeded

Some lines of code are too long.

Instances

- `observations.push(Observation(blockTimestamp, reserve0CumulativeLast, reserve1CumulativeLast, totalSupplyCumulativeLast));`
- `_totalSupply[index] = (observations[nextIndex].totalSupplyCumulative - observations[i].totalSupplyCumulative) / timeElapsed;`
- and more, if you take into account the beginning indent and / or comments.

Recommendation

According to Style Guide, maximum suggested line length is 120 characters.

Make the lines shorter.

[N-03] Constants may be used

Constants may be used instead of literal values.

Instances

- `uint[] memory supply = pair.sampleSupply(8, 1);`
- `prices = pair.sample(token1, decimals, 8, 1);`

- (unitReserves, assetReserves) = pair.sampleReserves(8, 1);
- prices = pair.sample(token0, decimals, 8, 1);
- (assetReserves, unitReserves) = pair.sampleReserves(8, 1);
- prices = pair.sample(token1, decimals, 8, 1);
- (unitReserves, assetReserves) = pair.sampleReserves(8, 1);
- prices = pair.sample(token0, decimals, 8, 1);
- (assetReserves, unitReserves) = pair.sampleReserves(8, 1);
- for(uint i; i < 8; ++i) {
- return 1e18; // Stable coins supported by the lending market are instantiated by governance and their price will always be 1 note
- return 1e18 * 1e18 / (10 ** decimals); //Scale Price as a mantissa to maintain precision in comptroller
- return 1e18 * 1e18 / (10 ** decimals); //Scale Price as a mantissa to maintain precision in comptroller
- return getPriceCanto(underlying) * getPriceNote(address(wcanto), false) / 1e18;
- LpPricesCumulative += (token0TVL + token1TVL) * 1e18 / supply[i];
- return LpPrice * getPriceNote(address(wcanto), false) / 1e18; // return the price in terms of Note
- return price * 1e18 / decimals; //return the scaled price
- return price * 1e18 / decimals; // divide by decimals now to maintain precision

Recommendation

Define constant variables for repeated values (8 and 1e18).

[N-04] Inconsistent comment spacing and location

Some comments are above the line of code and some next to it.

Some comments are indented between *// and the comment text*, some are not.

Instances

- Link:

```
{ //manual scope to pop symbol off of stack
 string memory symbol = ctoken.symbol();
```

- `underlying = address(ICErc20(address(ctoken)).underlying()); // We are getting the price for a CErc20 lending market`
- Link:

```
//set price statically to 1 when the Comptroller is retrieving I
if (compareStrings(symbol, "cNOTE")) { // note in terms of note
    return 1e18; // Stable coins supported by the lending market
```

- `return 1e18 * 1e18 / (10 ** decimals); //Scale Price as a mantissa to maintain precision in comptroller`
- all other functions...

Recommendation

Use consistent comment spacing and location.

[N-05] Loop parameter may be changed for clarity

In loop are used `_reserves0.length`. It is equal to input variable `granularity`. It can be clearer and more consistent if you use an input variable in the loop.

Instances

1. Link:

```
    function reserves(uint granularity) external view returns(ui
        (uint[] memory _reserves0, uint[] memory _reserves1)= sa
        uint reserveAverageCumulative0;
        uint reserveAverageCumulative1;
```

```
        for (uint i = 0; i < _reserves0.length; ++i) {
```

2. Link:

```
    function totalSupplyAvg(uint granularity) external view retu
        uint[] memory _totalSupplyAvg = sampleSupply(granularity
        uint totalSupplyCumulativeAvg;

        for (uint i = 0; i < _totalSupplyAvg.length; ++i) {
```

Recommendation

1. Change to

```
    function reserves(uint granularity) external view returns(ui
        (uint[] memory _reserves0, uint[] memory _reserves1)= sa
        uint reserveAverageCumulative0;
        uint reserveAverageCumulative1;

        //HERE
        for (uint i = 0; i < granularity; ++i) {
```

2. Change to

```
    function totalSupplyAvg(uint granularity) external view retu
        uint[] memory _totalSupplyAvg = sampleSupply(granularity
        uint totalSupplyCumulativeAvg;

        //HERE
        for (uint i = 0; i < granularity; ++i) {
```

[N-06] Functions without comments

Some functions do not have comments describing them.

Instances

- function reserves(uint granularity)

- `function sampleReserves(uint points, uint window)`
- `function totalSupplyAvg(uint granularity)`
- `function sampleSupply(uint points, uint window)`

Recommendation

Add comments.

[N-07] Require statement may be placed before allocating memory for arrays

Instances

1. Link:

```
uint[] memory _reserves0 = new uint[](points);
uint[] memory _reserves1 = new uint[](points);

uint lastIndex = observations.length-1;
require(lastIndex >= points * window, "PAIR::NOT READY I
```

2. Link:

```
uint[] memory _totalSupply = new uint[](points);

uint lastIndex = observations.length-1;
require(lastIndex >= points * window, "PAIR::NOT READY I
```

Recommendation

1. Change to

```
uint lastIndex = observations.length-1;
require(lastIndex >= points * window, "PAIR::NOT READY I

uint[] memory _reserves0 = new uint[](points);
uint[] memory _reserves1 = new uint[](points);
```

2. Change to

```
uint lastIndex = observations.length-1;
require(lastIndex >= points * window, "PAIR::NOT READY I

uint[] memory _totalSupply = new uint[](points);
```

[N-08] Check zero denominator

If the input parameter is equal to zero, this will cause the function call failure on division.

Instances

- `return (reserveAverageCumulative0 / granularity, reserveAverageCumulative1 / granularity);`
- `return (totalSupplyCumulativeAvg / granularity);`

Recommendation

Add the check to prevent function call failure.

[N-09] Missing check for input variables

If input variable `points` == 0, function will return empty array.

More critical, if input variable `window` == 0, function will return array with default values, which may lead to further incorrect calculations.

Instances

- `function sampleReserves(uint points, uint window)`
- `function sampleSupply(uint points, uint window)`

Recommendation

Add require statement or custom error - `points!= 0 && window!= 0`.

Disclosures

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

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest 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