Y2k Finance contest
Findings & Analysis Report

2023-01-09

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 Y2k Finance smart contract system written in Solidity. The audit contest took place between September 14—September 19 2022.

Wardens

114 Wardens contributed reports to the Y2k Finance contest:

1. PwnPatrol (obront and throttle)
2. eierina
3. Lambda
4. 0x52
5. cccz
6. csanuragjain
7. 0xDecorativePineapple
8. unforgiven
9. yixxas
10. 0xPanas (Bronicle and Deivitto)
11. Respx
12. Bahurum
13. Jeiwan
14. rbserver
15. rvierdiiev
16. pauliax
17. 0x1f8b
18. datapunk
19. async
20. hyh
21. R2
22. carrotsmuggler
23. pashov
24. thebensams
25. bin2chen
26. ak1
27. Tointer
28. Deivitto
29. robee

30. Rolezn
31. imare
32. ladboy233
33. 0xNazgul
34. rokinot
35. scaraven
36. Chom
37. nalus
38. wagmi
39. fatherOfBlocks
40. Saintcode_
41. Ch_301
42. joestakey
43. 0x4non
44. KIntern_NA (TrungOre and duc)
45. rotcivegaf
46. leosathya
47. c3phas
48. gogo
49. jonatascm
50. lukris02
51. oyc_109
52. RaymondFam
53. simon135
54. durianSausage
55. Haruxe
56. peritoflores
57. auditor0517
58. zzzitron

59. pfapostol
60. __141345__
61. _Adam
62. zishansami
63. ch0bu
64. cryptostellar5
65. dharma09
66. Diana
67. gianganhnguyen
68. jag
69. malinariy
70. MiloTruck
71. peiw
72. prasantgupta52
73. ReyAdmirado
74. Rohan16
75. seyni
76. 0xc0ffEE
77. 0xSmartContract
78. ajtra
79. Aymen0909
80. Bnke0x0
81. djxploit
82. erictee
83. Ruhum
84. 0xmuxyz
85. brgltd
86. CodingNameKiki
87. kv

88. Picodes
89. SooYa
90. V_B (Barichek and vlad_bochok)
91. teawaterwire
92. 0x040
93. 0xkatana
94. d3e4
95. delfin454000
96. ignacio
97. JAGADESH
98. peanuts
99. RoiEvenHaim
100. Samatak
101. slowmoses
102. Sm4rty
103. SnowMan
104. sryysryy
105. tnevler
106. Tomio
107. Tomo
108. WilliamAmbrozic
109. cryptphi
110. JC

This contest was judged by hickuphh3.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 25 unique vulnerabilities. Of these vulnerabilities, 9 received a risk rating in the category of HIGH severity and 16 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 59 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 72 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 Y2k Finance contest repository, and is composed of 7 smart contracts written in the Solidity programming language and includes 1,966 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into 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

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (9)

[H-01] Incorrect handling of `pricefeed.decimals()`

Submitted by carrotsmuggler, also found by 0x52, 0xDecorativePineapple, 0xPanas, auditor0517, Bahurum, durianSausage, hyh, Jeiwan, ladboy233, Lambda, pauliax, PwnPatrol, R2, Respx, scaraven, teawaterwire, and zzzitron

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/oracles/PegOracle.sol#L46-L83

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#L299-L300

Impact

Wrong math for handling pricefeed decimals. This code will only work for pricefeeds of 8 decimals, any others give wrong/incorrect data. The maths used can be shown in three lines:

```
    nowPrice = (price1 * 10000) / price2;
    nowPrice = nowPrice * int256(10**(18 - priceFeed1.decimals()));
    return nowPrice / 1000000;
```

Line1: adds 4 decimals Line2: adds (18 - d) decimals, (where d = pricefeed.decimals()) Line3: removes 6 decimals

Total: adds (16 - d) decimals

when d=8, the contract correctly returns an 8 decimal number. However, when d = 6, the function will return a 10 decimal number. This is further raised by (18-d = 12) decimals when checking for depeg event, leading to a 22 decimal number which is 4 orders of magnitude incorrect.

if d=18, (like usd-eth pricefeeds) contract fails / returns 0.

All chainlink contracts which give price in eth, operate with 18 decimals. So this can cripple the system if added later.

Proof of Concept

Running the test AssertTest.t.sol:testPegOracleMarketCreation and changing the line on

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/test/AssertTest.t.sol#L30

to

```
PegOracle pegOracle3 = new PegOracle(
            0xB1552C5e96B312d0Bf8b554186F846C40614a540,  //usd-e
            btcEthOracle
        );
```

gives this output

```
oracle3price1: 1085903802394919427
oracle3price2: 13753840915281064000
oracle3price1 / oracle3price2: 0
```

returning an oracle value of 0. Simulating with a mock price feed of 6 decimals gives results 4 orders of magnitude off.

Tools Used

Foundry, VS-Code

Recommended Mitigation Steps

Since only the price ratio is calculated, there is no point in increasing the decimal by (18-d) in the second line. Proposed solution:

```
nowPrice = (price1 * 10000) / price2;
nowPrice = nowPrice * int256(10**(priceFeed1.decimals())) * 100;
return nowPrice / 1000000;
```

This returns results in d decimals, no matter the value of d.

MiguelBits (Y2K Finance) confirmed

[H-02] End epoch cannot be triggered preventing winners to withdraw

Submitted by eierina

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#L198

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#L246

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#L261

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#L277-L286

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L203

Impact

At the end of an epoch, the triggerEndEpoch(...) is called to trigger 'epoch end without depeg event', making risk users the winners and entitling them to withdraw (risk + hedge) from the vault. In the case of the Arbitrum sequencer going down or restarting, there is a grace period of one hour before the getLatestPrice() returns to execute without reverting. This means that the triggerEndEpoch(...) cannot complete during this time, because it calls the getLatestPrice().

Making this high-priority because unless the triggerEndEpoch(...) completes:

- winners cannot withdraw althought the epoch is over;
- during this time the strike price might be reached causing a depeg event at all effects turning the table for the winners;
- the getLatestPrice() is not functional to the completion of the triggerEndEpoch(...), nor to the withdraw, but only informative used to initialize the event object emitted at the very end of the triggerEndEpoch function.

First two points each constitute independent justification, third point reinforces the first 2 points.

Proof of Concept

triggerEndEpoch reverts if arbiter down or restarted less than eq GRACE*PERIOD*TIME ago (1hr)

File: Controller.sol:L246

Revert if getLatestPrice reverts.

```
function triggerEndEpoch(uint256 marketIndex, uint256 epochEnd)

    < ... omitted ... >

    emit DepegInsurance(
        keccak256(
            abi.encodePacked(
                marketIndex,
                insrVault.idEpochBegin(epochEnd),
                epochEnd
            )
        ),
        tvl,
        false,
        epochEnd,
        block.timestamp,
        getLatestPrice(insrVault.tokenInsured()) // @audit getLa
    );
}
```

File: Controller.sol:L277-L286

Revert if sequencer down or grace period after restart not over.

```
    function getLatestPrice(address _token)
        public
        view
        returns (int256 nowPrice)
    {
        < ... omitted ... >

        bool isSequencerUp = answer == 0;
        if (!isSequencerUp) {
            revert SequencerDown();
        }

        // Make sure the grace period has passed after the sequencer
        uint256 timeSinceUp = block.timestamp - startedAt;
        if (timeSinceUp <= GRACE_PERIOD_TIME) { // @audit 1 hour
            revert GracePeriodNotOver();
        }

        < ... omitted ... >
    }
```

withdraw fails if triggerEndEpoch did not execute successfully

File: Vault.sol:L203

Can execute if block.timestamp > epochEnd, but fails if trigger did not execute. Winners cannot withdraw.

```
    function withdraw(
        uint256 id,
        uint256 assets,
        address receiver,
        address owner
    )
        external
        override
        epochHasEnded(id) // @audit same as require((block.timestamp
        marketExists(id)
        returns (uint256 shares)
    {
        < ... omitted ... >

        uint256 entitledShares = beforeWithdraw(id, shares); // @auc

        < ... omitted ... >

        emit Withdraw(msg.sender, receiver, owner, id, assets, entit
        asset.transfer(receiver, entitledShares);

        return entitledShares;
    }
```

Recommended Mitigation Steps

The latest price is retrieved at the very end of the [triggerEndEpoch(...)]() for the only purpose of initializing the DepegInsurance event. Since it is used for informational purpose (logging / offchain logging) and not for functional purpose to the [triggerEndEpoch(...)]() execution, it can be relaxed.

Depending on how the event is used, when `getLatestPrice()` is called for informative/logging purpose only, there could be few alternatives:

- log a 0 when SequencerDown or GRACE*PERIOD*TIME not passed
- log a 0 when SequencerDown and ignore GRACE*PERIOD*TIME

Once events are logged off-chain, some post processing may be used to correct/update the values with accurate data.

[3xHarry (Y2K Finance) commented]():

> Great catch!

[MiguelBits (Y2K Finance) confirmed and commented]():

> Fixed this by changing triggerEndEpoch,

```
    AggregatorV3Interface priceFeed = AggregatorV3Interface(
                vaultFactory.tokenToOracle(insrVault.tokenInsured())
```

```
        );
        (
            ,
            int256 price,
            ,
            ,

        ) = priceFeed.latestRoundData();

        emit DepegInsurance(
            keccak256(
                abi.encodePacked(
                    marketIndex,
                    insrVault.idEpochBegin(epochEnd),
                    epochEnd
                )
            ),
            tvl,
            true,
            epochEnd,
            block.timestamp,
            price
        );
```

HickupHH3 (judge) commented:

> Agree with the points raised by the warden, especially on how `getLatestPrice()` is merely for informational purposes in the event emission.

[H-03] A design flaw in the case of using 2 oracles (aka PegOracle)

Submitted by PwnPatrol

A design flaw in the case of using 2 oracles (aka PegOracle).

Proof of Concept

Chainlink provides price feeds denominated either in ETH or USD. But some assets don't have canonical value accessed on-chain. An example would be BTC and it's many on-chain forms like renBTC, hitBTC, WBTC, aBTC etc... For example in the case of a market on renBTC depegging from BTC value, probably a pair like renBTC/WBTC would be used (leveraging PegOracle). But even if renBTC perfectly maintains it's value to BTC, the depeg event can be triggered when WBTC significantly depreciates or appreciates against BTC value. This depeg event will be theoretically unsound since renBTC behaved as expected. The flaw comes from PegOracle because it treats both assets symmetrically.

This is also true for ETH pairs like stETH/aETH etc.. or stablecoin pairs like FRAX/MIM etc.. Of course, it should never be used like this because Chainlink provides price feeds with respect to true ETH and USD values but we have found that test files include PegOracle for stablecoin pairs.

Recommended Mitigation Steps

Support markets only for assets that have access to an oracle with price against canonical value x/ETH or x/USD.

MiguelBits (Y2K Finance) disputed

HickupHH3 (judge) commented:

> From what I understand, the warden is arguing that if the underlying asset is itself a pegged asset, then it wouldn't be a very good measure against the "canonical price".

> Eg. `MIM` (pegged) -> `USDC` (underlying), and USDC de-pegs, even though MIM is close to `$1` , the protocol would recognise this as a de-peg event.

> I agree with the issue, but disagree with the severity. The choice of the underlying token is quite obviously important; I think the sponsor can attest to this.

> @3xHarry thoughts? Perhaps low severity is more appropriate because it isn't a technical vulnerability per-se, more of the choice of underlying to be used.

HickupHH3 (judge) commented:

> Keeping high severity even though there are a couple of prerequisites:

- the protocol uses a poor underlying token (Eg. USDT that has de-pegged to `$0.97` before)

- underlying token de-pegs substantially to inaccurately trigger (or not trigger) a de-peg event

> I classify this as indirect loss of assets from a valid attack path that does not have hand-wavy hypotheticals

> 3 — High: Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).

[H-04] Users who deposit in one vault can lose all deposits and receive nothing when counterparty vault has no deposits

Submitted by rbserver, also found by 0x52, carrotsmuggler, Ch_301, imare, Jeiwan, ladboy233, Lambda, Tointer, unforgiven, and wagmi

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Controller.sol#L148-L192

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L350-L352

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L203-L234

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L378-L426

Impact

For a market, if users only deposit in the hedge vault or only deposit in the risk vault but not in both, then these users will lose their deposits and receive nothing when they call the following `withdraw` function after the depeg event occurs.

If the vault that has deposits is called Vault A, and the counterparty vault that has no deposit is called Vault B, then:

- As shown by the `triggerDepeg` function below, when executing `insrVault.sendTokens(epochEnd, address(riskVault))` and `riskVault.sendTokens(epochEnd, address(insrVault))`, the deposits of

Vault A are transferred to Vault B but nothing is transferred to Vault A since Vault B has no deposit;

- When `triggerDepeg` executes `insrVault.setClaimTVL(epochEnd, riskVault.idFinalTVL(epochEnd))` and `riskVault.setClaimTVL(epochEnd, insrVault.idFinalTVL(epochEnd))`, Vault B's `idClaimTVL[id]` is set to Vault A's `idFinalTVL(epochEnd))` but Vault A's `idClaimTVL[id]` is set to 0 because Vault B's `idFinalTVL(epochEnd)` is 0.

Because of these, calling the `beforeWithdraw` function below will return a 0 `entitledAmount`, and calling `withdraw` then transfers that 0 amount to the user who has deposited. As a result, these users' deposits are transferred to the counterparty vault, and they receive nothing at all.

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Controller.sol#L148-L192

```
    function triggerDepeg(uint256 marketIndex, uint256 epochEnd)
        public
        isDisaster(marketIndex, epochEnd)
    {
        address[] memory vaultsAddress = vaultFactory.getVaults(
        Vault insrVault = Vault(vaultsAddress[0]);
        Vault riskVault = Vault(vaultsAddress[1]);

        //require this function cannot be called twice in the sa
        if(insrVault.idFinalTVL(epochEnd) != 0)
            revert NotZeroTVL();
        if(riskVault.idFinalTVL(epochEnd) != 0)
            revert NotZeroTVL();

        insrVault.endEpoch(epochEnd, true);
        riskVault.endEpoch(epochEnd, true);

        insrVault.setClaimTVL(epochEnd, riskVault.idFinalTVL(epo
        riskVault.setClaimTVL(epochEnd, insrVault.idFinalTVL(epo

        insrVault.sendTokens(epochEnd, address(riskVault));
        riskVault.sendTokens(epochEnd, address(insrVault));

        VaultTVL memory tvl = VaultTVL(
            riskVault.idClaimTVL(epochEnd),
            insrVault.idClaimTVL(epochEnd),
            riskVault.idFinalTVL(epochEnd),
            insrVault.idFinalTVL(epochEnd)
        );

        emit DepegInsurance(
            keccak256(
                abi.encodePacked(
                    marketIndex,
                    insrVault.idEpochBegin(epochEnd),
                    epochEnd
                )
            ),
            tvl,
            true,
            epochEnd,
            block.timestamp,
            getLatestPrice(insrVault.tokenInsured())
        );
    }
```

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L350-L352

```
        function setClaimTVL(uint256 id, uint256 claimTVL) public or
            idClaimTVL[id] = claimTVL;
        }
```

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L203-L234

```
        function withdraw(
            uint256 id,
            uint256 assets,
            address receiver,
            address owner
        )
            external
            override
            epochHasEnded(id)
            marketExists(id)
            returns (uint256 shares)
        {
            if(
                msg.sender != owner &&
                isApprovedForAll(owner, receiver) == false)
                revert OwnerDidNotAuthorize(msg.sender, owner);

            shares = previewWithdraw(id, assets); // No need to chec

            uint256 entitledShares = beforeWithdraw(id, shares);
            _burn(owner, id, shares);

            //Taking fee from the amount
            uint256 feeValue = calculateWithdrawalFeeValue(entitledS
            entitledShares = entitledShares - feeValue;
            asset.transfer(treasury, feeValue);

            emit Withdraw(msg.sender, receiver, owner, id, assets, e
            asset.transfer(receiver, entitledShares);

            return entitledShares;
        }
```

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L378-L426

```
    function beforeWithdraw(uint256 id, uint256 amount)
        public
        view
        returns (uint256 entitledAmount)
    {
        // in case the risk wins aka no depeg event
        // risk users can withdraw the hedge (that is paid by th
        // hedge pay for each hedge seller = ( risk / tvl before
        // in case there is a depeg event, the risk users can or
        if (
            keccak256(abi.encodePacked(symbol)) ==
            keccak256(abi.encodePacked("rY2K"))
        ) {
            if (!idDepegged[id]) {
                //depeg event did not happen
                /*
                entitledAmount =
                    (amount / idFinalTVL[id]) *
                    idClaimTVL[id] +
                    amount;
                */
                entitledAmount =
                    amount.divWadDown(idFinalTVL[id]).mulDivDowr
                        idClaimTVL[id],
                        1 ether
                    ) +
                    amount;
            } else {
                //depeg event did happen
                entitledAmount = amount.divWadDown(idFinalTVL[ic
                    idClaimTVL[id],
                    1 ether
                );
            }
        }
        // in case the hedge wins aka depegging
        // hedge users pay the hedge to risk users anyway,
        // hedge guy can withdraw risk (that is transfered from
        // withdraw = % tvl that hedge buyer owns
        // otherwise hedge users cannot withdraw any Eth
        else {
            entitledAmount = amount.divWadDown(idFinalTVL[id]).n
                idClaimTVL[id],
                1 ether
            );
        }

        return entitledAmount;
    }
```

Proof of Concept

Please append the following tests in `test\AssertTest.t.sol` . These tests will pass to demonstrate the described scenarios.

```
    function testWithdrawFromRiskAfterDepegWhenThereIsNoCounterp
        vm.deal(chad, AMOUNT * CHAD_MULTIPLIER);

        vm.startPrank(admin);
        FakeOracle fakeOracle = new FakeOracle(oracleFRAX, STRIK
        vaultFactory.createNewMarket(FEE, tokenFRAX, DEPEG_AAA,
        vm.stopPrank();

        address hedge = vaultFactory.getVaults(1)[0];
        address risk = vaultFactory.getVaults(1)[1];

        Vault vHedge = Vault(hedge);
        Vault vRisk = Vault(risk);

        // chad deposits in risk vault, and no one deposits in h
        vm.startPrank(chad);
        ERC20(WETH).approve(risk, AMOUNT * CHAD_MULTIPLIER);
        vRisk.depositETH{value: AMOUNT * CHAD_MULTIPLIER}(endEpo

        assertTrue(vRisk.balanceOf(chad,endEpoch) == (AMOUNT * C
        vm.stopPrank();

        vm.warp(beginEpoch + 10 days);

        // depeg occurs
        controller.triggerDepeg(SINGLE_MARKET_INDEX, endEpoch);

        vm.startPrank(chad);

        // chad withdraws from risk vault
        uint256 assets = vRisk.balanceOf(chad,endEpoch);
        vRisk.withdraw(endEpoch, assets, chad, chad);

        assertTrue(vRisk.balanceOf(chad,endEpoch) == NULL_BALANC
        uint256 entitledShares = vRisk.beforeWithdraw(endEpoch,
        assertTrue(entitledShares - vRisk.calculateWithdrawalFee

        // chad receives nothing
        assertEq(entitledShares, 0);
        assertEq(ERC20(WETH).balanceOf(chad), 0);

        vm.stopPrank();
    }


    function testWithdrawFromHedgeAfterDepegWhenThereIsNoCounter
        vm.deal(alice, AMOUNT);

        vm.startPrank(admin);
        FakeOracle fakeOracle = new FakeOracle(oracleFRAX, STRIK
        vaultFactory.createNewMarket(FEE, tokenFRAX, DEPEG_AAA,
        vm.stopPrank();

        address hedge = vaultFactory.getVaults(1)[0];
        address risk = vaultFactory.getVaults(1)[1];

        Vault vHedge = Vault(hedge);
        Vault vRisk = Vault(risk);

        // alice deposits in hedge vault, and no one deposits ir
        vm.startPrank(alice);
        ERC20(WETH).approve(hedge, AMOUNT);
        vHedge.depositETH{value: AMOUNT}(endEpoch, alice);

        assertTrue(vHedge.balanceOf(alice,endEpoch) == (AMOUNT))
        vm.stopPrank();

        vm.warp(beginEpoch + 10 days);

        // depeg occurs
        controller.triggerDepeg(SINGLE_MARKET_INDEX, endEpoch);

        vm.startPrank(alice);

        // alice withdraws from hedge vault
        uint256 assets = vHedge.balanceOf(alice,endEpoch);
        vHedge.withdraw(endEpoch, assets, alice, alice);

        assertTrue(vHedge.balanceOf(alice,endEpoch) == NULL_BAL
        uint256 entitledShares = vHedge.beforeWithdraw(endEpoch,
        assertTrue(entitledShares - vHedge.calculateWithdrawalFe

        // alice receives nothing
        assertEq(entitledShares, 0);
        assertEq(ERC20(WETH).balanceOf(alice), 0);

        vm.stopPrank();
    }
```

Tools Used

VSCode

Recommended Mitigation Steps

When users only deposit in one vault, and no one deposits in the counterparty vault, the insurance practice of hedging and risking actually does not exist. In this situation, after the epoch is started, the users, who have deposited, should be allowed to withdraw their full deposit amounts.

3xHarry (Y2K Finance) confirmed

[H-05] LOSS OF PRECISION RESULTING IN WRONG VALUE FOR PRICE RATIO

Submitted by 0xDecorativePineapple, also found by 0xPanas and Lambda

The project implements a price oracle in order to get the relative price between the pegged asset and the price of the original asset (example: stETH to ETH). If the ratio (the pegged asset divided by the original asset) is 1 the Token is pegged, otherwise is depegged.

Below is a code snippet from the PegOracle.sol function.

```
    if (price1 > price2) {
            nowPrice = (price2 * 10000) / price1;
        } else {
            nowPrice = (price1 * 10000) / price2;
        }

        int256 decimals10 = int256(10**(18 - priceFeed1.decimals
        nowPrice = nowPrice * decimals10;

        return (
            roundID1,
            nowPrice / 1000000,
            startedAt1,
            timeStamp1,
            answeredInRound1
        );
    }
```

To fetch the ratio at any time, the `PegOracle.sol` performs some calculations; first the relative price is multiplied by 1e4 and then it returns the above calculation divided by 1e6.

The Controller.sol file makes an external call to the PegOracle.sol oracle to get the relative price. After, the value returned, it is multiplied by `10**(18-(priceFeed.decimals())` and the result represents the relative price between the two assets.

The result is converted to 18 decimal points in order to be compared with the Strike Price passed by the admin on `VaultFactory.sol`.

Due to the fact that the first multiplication is first divided by `1e6` (PegOracle.sol#L78)(and then re-multiplied by `uint256 decimals = 10**(18-(priceFeed.decimals()));` (Controller.sol#L299-L300) it leads to loss of precision. This behavior will make the relative price between the assets incorrect.

Proof of Concept

Below is a test that illustrates the above issue for various oracle pairs. The calculated ratio is compared against a modified version of an example of different price denominator, provided by Chainlink.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.15;

import "forge-std/Test.sol";
import "../lib/AggregatorV3Interface.sol";

//run with: forge test --fork-url https://arb1.arbitrum.io/rpc -

contract PegOracle {

    /***
    @dev  for example: oracle1 would be stETH / USD, while oracl
    ***/
    address public oracle1;
    address public oracle2;

    uint8 public decimals;

    AggregatorV3Interface internal priceFeed1;
    AggregatorV3Interface internal priceFeed2;
    /** @notice Contract constructor
      * @param _oracle1 First oracle address
      * @param _oracle2 Second oracle address
      */
    constructor(address _oracle1, address _oracle2) {
        require(_oracle1 != address(0), "oracle1 cannot be the z
        require(_oracle2 != address(0), "oracle2 cannot be the z
        require(_oracle1 != _oracle2, "Cannot be same Oracle");
        priceFeed1 = AggregatorV3Interface(_oracle1);
        priceFeed2 = AggregatorV3Interface(_oracle2);
        require(
            (priceFeed1.decimals() == priceFeed2.decimals()),
            "Decimals must be the same"
        );

        oracle1 = _oracle1;
        oracle2 = _oracle2;

        decimals = priceFeed1.decimals();
    }

    /** @notice Returns oracle-fed data from the latest round
      * @return roundID Current round id
      * @return nowPrice Current price
      * @return startedAt Starting timestamp
      * @return timeStamp Current timestamp
      * @return answeredInRound Round id for which answer was co
      */
    function latestRoundData()
        public
        view
        returns (
            uint80 roundID,
            int256 nowPrice,
            uint256 startedAt,
            uint256 timeStamp,
            uint80 answeredInRound
        )
    {
        (
            uint80 roundID1,
            int256 price1,
            uint256 startedAt1,
            uint256 timeStamp1,
            uint80 answeredInRound1
        ) = priceFeed1.latestRoundData();

        int256 price2 = getOracle2_Price();

        if (price1 > price2) {
            nowPrice = (price2 * 10000) / price1;
        } else {
            nowPrice = (price1 * 10000) / price2;
        }

        int256 decimals10 = int256(10**(18 - priceFeed1.decimals
        nowPrice = nowPrice * decimals10;

        return (
            roundID1,
            nowPrice / 1000000, //1000000,
            startedAt1,
            timeStamp1,
            answeredInRound1
        );
    }

    /* solhint-disbable-next-line func-name-mixedcase */
    /** @notice Lookup first oracle price
      * @return price Current first oracle price
      */
    function getOracle1_Price() public view returns (int256 pric
        (
            uint80 roundID1,
            int256 price1,
            ,
            uint256 timeStamp1,
            uint80 answeredInRound1
        ) = priceFeed1.latestRoundData();

        require(price1 > 0, "Chainlink price <= 0");
        require(
            answeredInRound1 >= roundID1,
            "RoundID from Oracle is outdated!"
        );
        require(timeStamp1 != 0, "Timestamp == 0 !");

        return price1;
    }

    /* solhint-disbable-next-line func-name-mixedcase */
```

```
/** @notice Lookup second oracle price
  * @return price Current second oracle price
  */
function getOracle2_Price() public view returns (int256 pric
    (
        uint80 roundID2,
        int256 price2,
        ,
        uint256 timeStamp2,
        uint80 answeredInRound2
    ) = priceFeed2.latestRoundData();

    require(price2 > 0, "Chainlink price <= 0");
    require(
        answeredInRound2 >= roundID2,
        "RoundID from Oracle is outdated!"
    );
    require(timeStamp2 != 0, "Timestamp == 0 !");

    return price2;
}

function latestRoundData2()
    public
    view
    returns (
        uint80 roundID,
        int256 nowPrice,
        uint256 startedAt,
        uint256 timeStamp,
        uint80 answeredInRound
    )
{
    (
        uint80 roundID1,
        int256 price1,
        uint256 startedAt1,
        uint256 timeStamp1,
        uint80 answeredInRound1
    ) = priceFeed1.latestRoundData();

    price1 = scalePriceTo18(price1, priceFeed1.decimals());

    int256 price2 = scalePriceTo18(getOracle2_Price(), price
        return (
            roundID1,
            price1  * 1e18 / price2,
            startedAt1,
            timeStamp1,
            answeredInRound1
        );
    }


function scalePriceTo18(int256 _price, uint8 _priceDecimals)
        internal
        pure
        returns (int256)
    {
        if (_priceDecimals < 18) {
            return _price * int256(10 ** uint256(18 - _priceDeci
        } else if (_priceDecimals > 18) {
            return _price * int256(10 ** uint256(_priceDecimals
        }
        return _price;
    }
}




contract TestOracles is Test {
    address WETH = 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1;

    address tokenFRAX = 0x17FC002b466eEc40DaE837Fc4bE5c67993ddBc
    address tokenMIM = 0xFEa7a6a0B346362BF88A9e4A88416B77a57D6c2
    address tokenFEI = 0x4A717522566C7A09FD2774cceDC5A8c43C5F9FI
    address tokenUSDC = 0xFF970A61A04b1cA14834A43f5dE4533eBDDB5C
    address tokenDAI = 0xDA10009cBd5D07dd0CeCc66161FC93D7c9000da
    address tokenSTETH = 0xEfa0dB536d2c8089685630fafe88CF780596

    address oracleFRAX = 0x0809E3d38d1B4214958faf06D8b1B1a2b73f2
    address oracleMIM = 0x87121F6c9A9F6E90E59591E4Cf4804873f54A9
    address oracleFEI = 0x7c4720086E6feb755dab542c46DE4f728E8830
    address oracleUSDC = 0x50834F3163758fcC1Df9973b6e91f0F0F0434
    address oracleDAI = 0xc5C8E77B397E531B8EC06BFb0048328B30E9eC
    address oracleSTETH = 0x07C5b924399cc23c24a95c8743DE4006a32k
    address oracleETH = 0x639Fe6ab55C921f74e7fac1ee960C0B6293ba6
    address btcEthOracle = 0xc5a90A6d7e4Af242dA238FFe279e9f2BA0c
    PegOracle pegOracle = new PegOracle(oracleSTETH, oracleETH);
    PegOracle pegOracle2 = new PegOracle(oracleFRAX, oracleFEI);
    PegOracle pegOracle3 = new PegOracle(oracleDAI, oracleFEI);

    function setUp() public {}

    function convertBasedOnContractsLogic(int256 price, uint8 or
        uint256 decimals = 10**(18- oracleDecimals );
        int256 newPrice = price * int256(decimals);
        return newPrice;
    }

    function testOraclePrices() public {
        (, int256 var1 ,,,) = pegOracle.latestRoundData();
        emit log_int(convertBasedOnContractsLogic(var1, pegOracl

        (, int256 var2 ,,,) = pegOracle.latestRoundData2();
        emit log_int(var2);

        (, int256 var3 ,,,) = pegOracle2.latestRoundData();
        emit log_int(convertBasedOnContractsLogic(var3, pegOracl

        (, int256 var4 ,,,) = pegOracle2.latestRoundData2();
        emit log_int(var4);


        (, int256 var5 ,,,) = pegOracle3.latestRoundData();
        emit log_int(convertBasedOnContractsLogic(var5, pegOracl

        (, int256 var6 ,,,) = pegOracle3.latestRoundData2();
        emit log_int(var6);
    }

}
```

Here is the output after running the with: `forge test --fork-url https://arb1.arbitrum.io/rpc -vv`: 990500000000000000 990544616614592905

996300000000000000 1003669952945847834

996000000000000000 1003940775578783463

Recommended Mitigation Steps

Since the 2 assets are required to having the same amount of decimals a formula that transforms the relative price to 1e18 could be: `x * 1e18 / y` .

An example that Chainlink implements, that includes a `scalePrice` function, in order to find a different price denominator could be found here.

MiguelBits (Y2K Finance) acknowledged

HickupHH3 (judge) commented:

> Agree with the issue; the precision loss may be the decisive factor between whether a depeg is ruled to have happened. Since the core functionality and user funds are at stake, the high severity is appropriate here.

[H-06] Griefing attack on the Vaults is possible, withdrawing the winning side stakes

Submitted by hyh, also found by 0x4non, 0xNazgul, Haruxe, joestakey, KIntern_NA, pauliax, peritoflores, PwnPatrol, Respx, rotcivegaf, scaraven, and Tointer

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/SemiFungibleVault.sol#L110-L119

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L203-L218

Vulnerability details

Anyone can withdraw to `receiver` once the `receiver` is `isApprovedForAll(owner, receiver)` . The funds will be sent to `receiver` , but it will happen whenever an arbitrary `msg.sender` wants. The only precondition is the presence of any approvals.

This can be easily used to sabotage the system as a whole. Say there are two depositors in the hedge Vault, Bob and David, both trust each other and approved each other. Mike the attacker observing the coming end of epoch where no depeg happened, calls the `withdraw()` for both Bob and David in the last block of the epoch. Mike gained nothing, while both Bob and David lost the payoff that was guaranteed for them at this point.

Setting the severity to be high as this can be routinely used to sabotage the Y2K users, both risk and hedge, depriving them from the payouts whenever they happen to be on the winning side. Usual attackers here can be the users from another side, risk users attacking hedge vault, and vice versa.

Proof of Concept

`isApprovedForAll()` in withdrawal functions checks the `receiver` to be approved, not the caller.

SemiFungibleVault's withdraw:

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/SemiFungibleVault.sol#L110-L119

```
    function withdraw(
        uint256 id,
        uint256 assets,
        address receiver,
        address owner
    ) external virtual returns (uint256 shares) {
        require(
            msg.sender == owner || isApprovedForAll(owner, recei
            "Only owner can withdraw, or owner has approved rece
        );
```

Vault's withdraw:

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L203-L218

```
        function withdraw(
            uint256 id,
            uint256 assets,
            address receiver,
            address owner
        )
            external
            override
            epochHasEnded(id)
            marketExists(id)
            returns (uint256 shares)
        {
            if(
                msg.sender != owner &&
                isApprovedForAll(owner, receiver) == false)
                revert OwnerDidNotAuthorize(msg.sender, owner);
```

This way anyone at any time can run withdraw from the Vaults whenever owner has some address approved.

Recommended Mitigation Steps

Consider changing the approval requirement to be for the caller, not receiver:

SemiFungibleVault's withdraw:

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/SemiFungibleVault.sol#L110-L119

```
        function withdraw(
            uint256 id,
            uint256 assets,
            address receiver,
            address owner
        ) external virtual returns (uint256 shares) {
            require(
-               msg.sender == owner || isApprovedForAll(owner, recei
+               msg.sender == owner || isApprovedForAll(owner, msg.s
                "Only owner can withdraw, or owner has approved rece
            );
```

Vault's withdraw:

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L203-L218

```
        function withdraw(
            uint256 id,
            uint256 assets,
            address receiver,
            address owner
        )
            external
            override
            epochHasEnded(id)
            marketExists(id)
            returns (uint256 shares)
        {
            if(
                msg.sender != owner &&
-               isApprovedForAll(owner, receiver) == false)
+               isApprovedForAll(owner, msg.sender) == false)
                revert OwnerDidNotAuthorize(msg.sender, owner);
```

MiguelBits (Y2K Finance) confirmed and commented:

> Implementing this.

HickupHH3 (judge) commented:

> Agree with the warden's finding, and the impact of "depriving them (y2k users) from the payouts whenever they happen to be on the winning side".

[H-07] Risk users are required to payout if the price of the pegged asset goes higher than underlying

Submitted by 0x52, also found by 0xDecorativePineapple, hyh, Jeiwan, Lambda, and PwnPatrol

Insurance is to protect the user in case the pegged asset drops significantly below the underlying but risk users are required to payout if the pegged asset is worth more than the underlying.

Proof of Concept

```
        if (price1 > price2) {
            nowPrice = (price2 * 10000) / price1;
        } else {
            nowPrice = (price1 * 10000) / price2;
        }
```

The above lines calculates the ratio using the lower of the two prices, which means that in the scenario, the pegged asset is worth more than the underlying, a depeg event will be triggered. This is problematic for two reasons. The first is that many pegged assets are designed to maintain at least the value of the underlying. They put very strong incentives to keep the asset from going below the peg but usually use much looser policies to bring the asset down to the peg, since an upward break from the peg is usually considered benign. The second is that when a pegged asset moves above the underlying, the users who are holding the asset are benefiting from the appreciation of the asset; therefore the insurance is not needed.

Because of these two reasons, it is my opinion that sellers would demand a higher premium from buyers as a result of the extra risk introduced by the possibility of having to pay out during an upward depeg. It is also my opinion that these higher premiums would push users seeking insurance to other cheaper products that don't include this risk.

Recommended Mitigation Steps

The ratio returned should always the ratio of the pegged asset to the underlying (i.e. pegged/underlying).

MiguelBits (Y2K Finance) marked as duplicate and commented:

> Duplicate of 26.

HickupHH3 (judge) commented:

> Not a duplicate.

> Pegged tokens go both ways: either valued more or less than the asset it's pegging to (underlying token).

> The warden is arguing that when the pegged token is worth more than the underlying (eg. worth > `$1` for a stablecoin), the users are still eligible for a payout, which he argues shouldnt be the case.

> I agree with the warden; from experience, most projects see it as a positive if their algo stablecoin is worth more than the underlying, and so, wouldn't do nothing about it. In fact, they'd probably use it as a shilling point to attract more users to mint more of these tokens to help bring the price down. This scenario should not be covered by the insurers.

[H-08] Vault.sol is not EIP-4626 compliant

Submitted by 0x52, also found by Bahurum, Jeiwan, Lambda, PwnPatrol, and thebensams

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/Vault.sol#L244-L252

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/SemiFungibleVault.sol#L205-L213

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/SemiFungibleVault.sol#L237-L239

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/SemiFungibleVault.sol#L244-L246

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/SemiFungibleVa ult.sol#L251-L258

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/SemiFungibleVault.sol#L263-L270

Impact

Other protocols that integrate with Y2K may wrongly assume that the functions are EIP-4626 compliant. Thus, it might cause integration problems in the future that can lead to wide range of issues for both parties.

Proof of Concept

All official EIP-4626 requirements can be found on it's official page. Non-compliant functions are listed below along with the reason they are not compliant:

The following functions are missing but should be present:

1. mint(uint256, address) returns (uint256)
2. redeem(uint256, address, address) returns (uint256)

The following functions are non-compliant because they don't account for withdraw and deposit locking:

1. maxDeposit
2. maxMint
3. maxWithdraw
4. maxRedeem

All of the above functions should return 0 when their respective functions are disabled (i.e. maxDeposit should return 0 when deposits are disabled)

previewDeposit is not compliant because it must account for fees which it does not

totalAssets is not compliant because it does not always return the underlying managed by the vault because it fails to include the assets paid out during a depeg or the end of the epoch.

Recommended Mitigation Steps

All functions listed above should be modified to meet the specifications of EIP-4626.

MiguelBits (Y2K Finance) confirmed

HickupHH3 (judge) commented:

> The premise is valid because it's stated in the README:

> Y2K leverages ERC4626 Vault standard for this protocol, this contract is a fork of that standard, although we replaced all uses of ERC20 to ERC1155.

> As per the ruling in a previous contest regarding EIP4626.

> Judging this and all duplicate regarding EIP4626 implementation as High Risk. EIP4626 is aimed to create a consistent and robust implementation patterns for Tokenized Vaults. A slight deviation from 4626 would broke composability and potentially lead to loss of funds. It is counterproductive to implement EIP4626 but does not conform to it fully.

> The missing functions are the most problematic; one expects the `mint()` and `redeem()` to be present, but they're absent instead.

> Disagree on the `max*()` functions issues; `SemiFungibleVault` is not pausable, functions can't be disabled / paused. Perhaps the inheriting contracts should override these functions, but the way I see it, they can be arbitrarily set in the template.

> Agree on subsequent points mentioned.

[H-09] Depeg event can happen at incorrect price

Submitted by csanuragjain, also found by bin2chen, datapunk, Lambda, R2, rbserver, and unforgiven

Depeg event can still happen when the price of a pegged asset is equal to the strike price of a Vault which is incorrect.

This docs clearly mentions:

“When the price of a pegged asset is below the strike price of a Vault, a Keeper(could be anyone) will trigger the depeg event and both Vaults(hedge and risk) will swap their total assets with the other party.” - https://code4rena.com/contests/2022-09-y2k-finance-contest

Proof of Concept

1. Assume strike price of vault is 1 and current price of pegged asset is also 1
2. User calls triggerDepeg function which calls isDisaster modifier to check the depeg eligibility
3. Now lets see isDisaster modifier

```
modifier isDisaster(uint256 marketIndex, uint256 epochEnd) {
        address[] memory vaultsAddress = vaultFactory.getVaults(
        if(
            vaultsAddress.length != VAULTS_LENGTH
            )
            revert MarketDoesNotExist(marketIndex);

        address vaultAddress = vaultsAddress[0];
        Vault vault = Vault(vaultAddress);

        if(vault.idExists(epochEnd) == false)
            revert EpochNotExist();

        if(
            vault.strikePrice() < getLatestPrice(vault.tokenInsu
            )
            revert PriceNotAtStrikePrice(getLatestPrice(vault.to

        if(
            vault.idEpochBegin(epochEnd) > block.timestamp)
            revert EpochNotStarted();

        if(
            block.timestamp > epochEnd
            )
            revert EpochExpired();
        _;
```

```
    }
```

4. Assume block.timestamp is at correct timestamp (between idEpochBegin and epochEnd), so none of revert execute. Lets look into the interesting one at

```
        if(
            vault.strikePrice() < getLatestPrice(vault.tokenInsu
            )
            revert PriceNotAtStrikePrice(getLatestPrice(vault.to
```

5. Since in our case price of vault=price of pegged asset so if condition does not execute and finally isDisaster completes without any revert meaning go ahead of depeg
6. But this is incorrect since price is still not below strike price and is just equal

Recommended Mitigation Steps

Change the isDisaster modifier to revert when price of a pegged asset is equal to the strike price of a Vault

```
if(
            vault.strikePrice() <= getLatestPrice(vault.tokenIns
            )
            revert PriceNotAtStrikePrice(getLatestPrice(vault.to
```

MiguelBits (Y2K Finance) disputed and commented:

> After discussion, the docs clearly state only below the strike Price

```
This docs clearly mentions:

"When the price of a pegged asset is below the strike price of a
```

csanuragjain (warden) commented:

> @MiguelBits Exactly when it is below the strike price but in this case depeg is happening when price is equal and not below. Can you please suggest?

MiguelBits (Y2K Finance) confirmed and commented:

> Oh I see what you mean, need to correct it!

HickupHH3 (judge) commented:

> Ah, a matter of when the equality sign matters a lot. Critically, in this case. Agree with warden that it should be `<=` and not `<` only.

Medium Risk Findings (16)

[M-01] Oracle is tracked per token instead of per pair, leading to surprise results

Submitted by PwnPatrol, also found by datapunk and Lambda

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/VaultFactory.sol#L121

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/VaultFactory.sol#L221-L223

Impact

Oracles are tracked by individual token, instead of by pair. Because for some tokens (ie BTC) it is unclear which implementation is the canonical one to compare others to, this can result in a situation where different pairs (ie ibBTC-wBTC and ibBC-renBTC) using the same oracle, which will be incorrect for one of them.

Proof of Concept

The protocol assumes that there is a canonical asset to compare pegged assets to, so oracles are tracked only by the pegged asset. However, for some assets (like BTC), there is no clear canonical asset, and the result is that tracking `tokenToOracle` is not sufficient.

When there is a conflict in `tokenToOracle`, the protocol responds by skipping the assignment and keeping the old value:

```
    if (tokenToOracle[_token] == address(0)) {
        tokenToOracle[_token] = _oracle;
    }
```

The result of this is that the protocol may define a new pair with a new oracle, and have it silently skip it and use a non-matching oracle. As an example:

- The admins start with an implementation of ibBTC-wBTC, deploying the oracle
- This is set as `tokenToOracle[ibBTC]`
- Later, the admins create a new pair for ibBTC-renBTC, deploying a new oracle
- The protocol silently skips this assignment and uses the ibBTC-wBTC oracle

This can produce incorrect results, for example if wBTC appreciates relative to the other two, or if both ibBTC and renBTC depeg.

Tools Used

Foundry

Recommended Mitigation Steps

Change `tokenToOracle` to represent the pair of tokens, either by creating a `Pair` struct as the key, or by nesting a mapping inside of another mapping.

MiguelBit (Y2K Finance) disputed

[M-02] Fee-on-Transfer tokens cause problems in multiple places

Submitted by Lambda, also found by csanuragjain, Deivitto, R2, and Rolezn

https://github.com/code-423n4/2022-09-y2k-finance/blob/bca5080635370424a9fe21fe1aded98345d1f723/src/SemiFungibleVault.sol#L94

https://github.com/code-423n4/2022-09-y2k-finance/blob/bca5080635370424a9fe21fe1aded98345d1f723/src/Controller.sol#L168

https://github.com/code-423n4/2022-09-y2k-finance/blob/bca5080635370424a9fe21fe1aded98345d1f723/src/Controller.sol#L225

Impact & Proof Of Concept

Certain tokens (e.g., STA or PAXG) charge a fee for transfers and others (e.g., USDT or USDC) may start doing so in the future. This is not correctly handled in multiple places and would lead to a loss of funds:

1. `SemiFungibleVault.deposit`: Here, less tokens are transferred to the vault than the amount of shares that is minted to the user. This is an accounting mistake that will ultimately lead to the situation where the last user(s) cannot withdraw anymore, because there are no more assets left.
2. `Controller.triggerDepeg` & `Controller.triggerEndEpoch`: `sendTokens` tries to send the whole asset balance to the other contract, which will fail when less tokens are available at this point (because the previous accounting was done without incorporating fees). This will mean that the end can never be triggered and all assets are lost.

Recommended Mitigation Steps

When fee-on-transfer tokens should be supported, you need to check the actual balance differences. If they are not supported, this should be clearly documented.

MiguelBits (Y2K Finance) acknowledged

HickupHH3 (judge) commented:

> Valid because SemiFungibleVault may not be applied to WETH only (unlike Vault), but generic tokens. Furthermore, if there is an intention to make semi fungible vaults an EIP standard, then one may have to consider catering to FoT tokens as well, unless explicitly stated otherwise.

[M-03] StakingRewards: Significant loss of precision possible

Submitted by Lambda

In `notifyRewardAmount`, the reward rate per second is calculated. This calculation rounds down, which can lead to situations where significantly less rewards are paid out to stakers, because the effect of the rounding is multiplied by the duration.

Proof Of Concept

Let's say we have a `rewardsDuration` of 4 years, i.e. 126144000 seconds. We assume the `rewardRate` is currently ß and `notifyRewardAmount` is called with the reward amount 252287999. Because the calculation rounds down, `rewardRate` will be 1. After the 4 years, the user have received 126144000 reward tokens. However, 126143999 (i.e., almost 50%) of the reward tokens that were intended to be distributed to the stakers were not distributed, resulting in monetary loss for all stakers.

Recommended Mitigation Steps

You could accumulate the differences that occur due to rounding and let the users claim them in the end according to their shares.

MiguelBits (Y2K Finance) acknowledged

HickupHH3 (judge) commented:

> While it's an edge case, the numbers used in the POC are reasonable if we consider small token decimals (eg. EURS with 2 decimals).

[M-04] It is possible that receiver and treasury can receive

nothing when calling `withdraw` function due to division being performed before multiplication

Submitted by rbserver, also found by ak1, Chom, nalus, and robee

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L378-L426

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L203-L234

Impact

In the following `beforeWithdraw` function, `entitledAmount = amount.divWadDown(idFinalTVL[id]).mulDivDown(idClaimTVL[id], 1 ether)` can be executed in several places. Because it uses division before multiplication, it is possible that `entitledAmount` is calculated to be 0. As the `withdraw` function shows below, when `entitledAmount` is 0, the receiver and treasury both receive 0. As a result, calling `withdraw` with a positive `assets` input can still result in transferring nothing to the receiver and treasury.

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L378-L426

```
    function beforeWithdraw(uint256 id, uint256 amount)
        public
        view
        returns (uint256 entitledAmount)
    {
        // in case the risk wins aka no depeg event
        // risk users can withdraw the hedge (that is paid by th
        // hedge pay for each hedge seller = ( risk / tvl before
        // in case there is a depeg event, the risk users can on
        if (
            keccak256(abi.encodePacked(symbol)) ==
            keccak256(abi.encodePacked("rY2K"))
        ) {
            if (!idDepegged[id]) {
                //depeg event did not happen
                /*
                entitledAmount =
                    (amount / idFinalTVL[id]) *
                    idClaimTVL[id] +
                    amount;
                */
                entitledAmount =
                    amount.divWadDown(idFinalTVL[id]).mulDivDowr
                        idClaimTVL[id],
                        1 ether
                    ) +
                    amount;
            } else {
                //depeg event did happen
                entitledAmount = amount.divWadDown(idFinalTVL[ic
                    idClaimTVL[id],
                    1 ether
                );
            }
        }
        // in case the hedge wins aka depegging
        // hedge users pay the hedge to risk users anyway,
        // hedge guy can withdraw risk (that is transfered from
        // withdraw = % tvl that hedge buyer owns
        // otherwise hedge users cannot withdraw any Eth
        else {
            entitledAmount = amount.divWadDown(idFinalTVL[id]).n
                idClaimTVL[id],
                1 ether
            );
        }

        return entitledAmount;
    }
```

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L203-L234

```
    function withdraw(
        uint256 id,
        uint256 assets,
        address receiver,
        address owner
    )
        external
        override
        epochHasEnded(id)
        marketExists(id)
        returns (uint256 shares)
    {
        if(
            msg.sender != owner &&
            isApprovedForAll(owner, receiver) == false)
            revert OwnerDidNotAuthorize(msg.sender, owner);

        shares = previewWithdraw(id, assets); // No need to chec

        uint256 entitledShares = beforeWithdraw(id, shares);
        _burn(owner, id, shares);

        //Taking fee from the amount
        uint256 feeValue = calculateWithdrawalFeeValue(entitledS
        entitledShares = entitledShares - feeValue;
        asset.transfer(treasury, feeValue);

        emit Withdraw(msg.sender, receiver, owner, id, assets, e
        asset.transfer(receiver, entitledShares);

        return entitledShares;
    }
```

Proof of Concept

Please append the following test in `test\AssertTest.t.sol`. This test will pass to demonstrate the described scenario.

```
    function testReceiveZeroDueToDivBeingPerformedBeforeMul() pu
        vm.deal(alice, 1e24);
        vm.deal(chad, 1e24);

        vm.startPrank(admin);
        FakeOracle fakeOracle = new FakeOracle(oracleFRAX, STRIK
        vaultFactory.createNewMarket(FEE, tokenFRAX, DEPEG_AAA,
        vm.stopPrank();

        address hedge = vaultFactory.getVaults(1)[0];
        address risk = vaultFactory.getVaults(1)[1];

        Vault vHedge = Vault(hedge);
        Vault vRisk = Vault(risk);
        // alice deposits 1e24 in hedge vault
        vm.startPrank(alice);
        ERC20(WETH).approve(hedge, 1e24);
        vHedge.depositETH{value: 1e24}(endEpoch, alice);
        vm.stopPrank();

        // chad deposits 1e24 in risk vault
        vm.startPrank(chad);
        ERC20(WETH).approve(risk, 1e24);
        vRisk.depositETH{value: 1e24}(endEpoch, chad);
        vm.stopPrank();

        vm.warp(beginEpoch + 10 days);

        // depeg occurs
        controller.triggerDepeg(SINGLE_MARKET_INDEX, endEpoch);

        vm.startPrank(chad);

        // chad withdraws 1e5 from risk vault
        vRisk.withdraw(endEpoch, 1e5, chad, chad);

        // the amount to chad is 0 because division is performec
        uint256 entitledShares = vRisk.beforeWithdraw(endEpoch,

        // chad receives nothing
        assertEq(entitledShares, 0);
        assertEq(ERC20(WETH).balanceOf(chad), 0);

        // the amount to chad would be positive when multiplicat
        uint256 entitledShares2 = (1e5 * vRisk.idClaimTVL(endEpc
        assertTrue(entitledShares2 > entitledShares);

        vm.stopPrank();
    }
```

Tools Used

VSCode

Recommended Mitigation Steps

`entitledAmount = amount.divWadDown(idFinalTVL[id]).mulDivDown(idClaimTVL[id], 1 ether)` in the `beforeWithdraw` function can be updated to the following code.

```
        entitledAmount = (amount * idClaimTVL[id]) / idFinalTVL[id]
```

MiguelBits (Y2K Finance) confirmed

HickupHH3 (judge) commented:

> In addition, I recommend adding a check to ensure that `entitledShares` is greater than 0, since it would be possible to have 0 shares if `(amount * idClaimTVL[id]) < idFinalTVL[id]`.

[M-05] StakingRewards.sol#stake is intended to be pausable but isn't

Submitted by 0x52

Staking is unable to be paused as intended.

Proof of Concept

StakingRewards.sol inherits pausable and implements the whenNotPaused modifier on stake, but doesn't implement any method to actually pause or unpause the contract. Pausable.sol only implements internal functions, which requires external or public functions to be implemented to wrap them. Since nothing like this has been implemented, the entire pausing system is rendered useless and staking cannot be paused as is intended.

Recommended Mitigation Steps

Create simple external pause and unpause functions that can be called by owner.

MiguelBits (Y2K Finance) disputed

HickupHH3 (judge) commented:

> Great catch!

> While the contract is taken from Synthetix's StakingRewards; note that they use a different version of Pausable that comes with a `setPaused()` function. This is notably absent from OZ's implementation; one has to have the pause and unpause function explicitly created.

[M-06] Fees are taken on risk collateral

Submitted by 0x52

Fees are taken on funds deposited as collateral.

Proof of Concept

```
    uint256 feeValue = calculateWithdrawalFeeValue(entitledShares, i
```

In L226 of Vault.sol#withdraw the fee is taken on the entire collateral deposited by the risk users. This is problematic for two reasons. The first is that the collateral provided by the risk users will likely be many many times higher than the premium being paid by the hedge users. This will create a strong disincentive to use the protocol because it is likely a large portion of the profits will be taken by fees and the risk user may unexpectedly lose funds overall if premiums are too low.

The second issue is that this method of fees directly contradicts project documents which clearly indicate that fees are only taken on the premium and insurance payouts, not when risk users are receiving their collateral back.

Recommended Mitigation Steps

Fee calculations should be restructured to only take fees on premiums and insurance payouts.

MiguelBits (Y2K Finance) disputed

HickupHH3 (judge) commented:

> Agree with the warden. If the withdrawal fee exceeds the premium paid, risk users are disincentivised to provide collateral.

> The second issue is that this method of fees directly contradicts project documents which clearly indicate that fees are only taken on the premium and insurance payouts, not when risk users are receiving their collateral back.

> Not sure if the warden is referring to the fee as the trading fee `c` in the article, but I would agree if it's the case. Implementation isn't according to spec. The actual fees charged might be more than expected.

[M-07] User funds lost because they can't `withdraw()` their funds before epoch startTime and they are stuck in positions that become unprofitable even when epoch is not started

Submitted by unforgiven, also found by carrotsmuggler and cccz

Users deposit their funds in `Vault` when epoch is not started but as other users deposit funds too or price of pegged token changes, users get different risk to reward. And they may want to withdraw their funds before epoch start time to get out of bad position, but there is no logic in code to give them ability to withdraw their funds before epoch start time.

Proof of Concept

`Withdraw()` function in `Vault` only allows users to withdraw after epoch ends and there is no logic in the contract to allow users to withdraw their funds before epoch start time.

After users deposit their funds, the risk to reward ratio of their investment changes as other users deposit funds in one of the Vaults and user may wants to withdraw their funds if they saw that position is bad for them or maybe the price of that token has been changed dramatically before epoch startTime and users wants to withdraw. But, there is no functionality that gives users the ability to withdraw their funds before epoch start time and users lose control of their funds after depositing and before epoch start time. As epoch is not started yet, users should be able to withdraw their funds but there is no such functionality in the code.

Tools Used

VIM

Recommended Mitigation Steps

Add some logic to give users the ability to withdraw funds before epoch start time.

[MiguelBits (Y2K Finance) disputed and commented]:

> Working as intended.

[M-08] `timewindow` can be changed unexpectedly that blocks users from calling `deposit` function

Submitted by rbserver, also found by 0x1f8b, cccz, eierina, rokinot, and unforgiven

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L87-L91

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L152-L174

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/VaultFactory.sol#L327-L338

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L287-L289

Impact

As shown by the following `epochHasNotStarted` modifier, which is used by the `deposit` function below, users can only deposit when `block.timestamp <= idEpochBegin[id] - timewindow` holds true. Before depositing, a user can check if this relationship is true at that moment; if so, she or he can call the `deposit` function. However, just before the user's `deposit` function call is executed, the admin unexpectedly calls the `VaultFactory.changeTimewindow` function below, which further calls the `Vault.changeTimewindow` function below, to increase the `timewindow`. Since the admin's `VaultFactory.changeTimewindow` transaction is executed before the user's `deposit` transaction and the `timewindow` change takes effect immediately, it is possible that the user's `deposit` function call will revert. Besides wasting gas, the user can feel confused and unfair because her or his `deposit` transaction should be executed successfully if `VaultFactory.changeTimewindow` is not called unexpectedly.

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L87-L91

```
    modifier epochHasNotStarted(uint256 id) {
        if(block.timestamp > idEpochBegin[id] - timewindow)
            revert EpochAlreadyStarted();
        _;
    }
```

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L152-L174

```
    function deposit(
        uint256 id,
        uint256 assets,
        address receiver
    )
        public
        override
        marketExists(id)
        epochHasNotStarted(id)
        nonReentrant
        returns (uint256 shares)
    {
        // Check for rounding error since we round down in previ
        require((shares = previewDeposit(id, assets)) != 0, "Zer

        asset.transferFrom(msg.sender, address(this), shares);
        _mint(receiver, id, shares, EMPTY);

        emit Deposit(msg.sender, receiver, id, shares, shares);

        return shares;
    }
```

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/VaultFactory.sol#L327-L338

```
    function changeTimewindow(uint256 _marketIndex, uint256 _tim
        public
        onlyAdmin
    {
        address[] memory vaults = indexVaults[_marketIndex];
        Vault insr = Vault(vaults[0]);
        Vault risk = Vault(vaults[1]);
        insr.changeTimewindow(_timewindow);
        risk.changeTimewindow(_timewindow);

        emit changedTimeWindow(_marketIndex, _timewindow);
    }
```

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L287-L289

```
    function changeTimewindow(uint256 _timewindow) public onlyFa
        timewindow = _timewindow;
    }
```

Proof of Concept

Please add the following `error` and append the following test in `test\AssertTest.t.sol`. This test will pass to demonstrate the described scenario.

```
    error EpochAlreadyStarted();

    function testChangeTimeWindowUnexpectedly() public {
        vm.deal(alice, AMOUNT);
        vm.deal(chad, AMOUNT * CHAD_MULTIPLIER);

        vm.startPrank(admin);
        FakeOracle fakeOracle = new FakeOracle(oracleFRAX, STRIK
        vaultFactory.createNewMarket(FEE, tokenFRAX, DEPEG_AAA,
        vm.stopPrank();

        address hedge = vaultFactory.getVaults(1)[0];
        address risk = vaultFactory.getVaults(1)[1];

        Vault vHedge = Vault(hedge);
        Vault vRisk = Vault(risk);

        // alice is able to deposit in hedge vault before the ti
        vm.startPrank(alice);
        ERC20(WETH).approve(hedge, AMOUNT);
        vHedge.depositETH{value: AMOUNT}(endEpoch, alice);
        vm.stopPrank();

        // admin changes time window unexpectedly, which takes e
        vm.startPrank(admin);
        vaultFactory.changeTimewindow(1, 5 days);
        vm.stopPrank();

        // chad is unable to deposit in risk vault after the tim
        vm.startPrank(chad);
        ERC20(WETH).approve(risk, AMOUNT * CHAD_MULTIPLIER);

        vm.expectRevert(EpochAlreadyStarted.selector);
        vRisk.depositETH{value: AMOUNT * CHAD_MULTIPLIER}(endEpo
        vm.stopPrank();
    }
```

Tools Used

VSCode

Recommended Mitigation Steps

When calling the `VaultFactory.createNewMarket` or `VaultFactory.deployMoreAssets` function, the `timewindow`, which is configured for that moment, can be taken into account in the created asset's `epochBegin`.

Then, https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Vault.sol#L87-L91 can be updated to the following code.

```
    modifier epochHasNotStarted(uint256 id) {
        if(block.timestamp > idEpochBegin[id])
            revert EpochAlreadyStarted();
        _;
    }
```

MiguelBits (Y2K Finance) disputed and commented:

> Working as intended. Added timelock to this function as pointed out in another issue.

HickupHH3 (judge) commented:

> Not really an admin privilege issue; time window changes shouldnt be retroactively applied as it changes the T&Cs of users that they were fine with (feeling a sense of rug).

[M-09] StakingRewards: recoverERC20() can be used as a backdoor by the owner to retrieve rewardsToken

Submitted by cccz, also found by csanuragjain, fatherOfBlocks, pashov, Respx, Saintcode_, and unforgiven

Similar to https://github.com/code-423n4/2022-02-concur-findings/issues/210
StakingRewards.recoverERC20 rightfully checks against the stakingToken being sweeped away. However, there's no check against the rewardsToken. This is the case of an admin privilege, which allows the owner to sweep the rewards tokens, perhaps as a way to rug depositors.

```
    function recoverERC20(address tokenAddress, uint256 tokenAmo
        external
        onlyOwner
    {
        require(
            tokenAddress != address(stakingToken),
            "Cannot withdraw the staking token"
        );
        ERC20(tokenAddress).safeTransfer(owner, tokenAmount);
        emit Recovered(tokenAddress, tokenAmount);
    }
```

Proof of Concept

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/rewards/StakingRewards.sol#L213-L223

Recommended Mitigation Steps

Add an additional check

```
        require(
            tokenAddress != address(rewardsToken),
            "Cannot withdraw the rewards token"
        );
```

scaraven (warden) commented:

> I'm curious what others think about this issue, if users do not receive any reward tokens is that really a problem? Users are still able to withdraw their ERC1155 tokens at any time, and vaults still work as expected. If the admin is malicious, users will miss out on tokens which will be worthless after a rugpull anyway.

MiguelBits (Y2K Finance) disputed and commented:

> This is how synthetix rewards work, I forked their smart contracts.

HickupHH3 commented:

> Including this issue, issues #50, #51 and #52 can be considered to be admin privilege findings. There's active discussion revolving how findings of this category should be handled / standardized.

> For now, I'm keeping the medium severity due to historical context (past contest references). I also reproduce the classification rationale that I gave for a previous contest below:

Classification and thought process

> The issues raised about rugpull vulnerabilities via centralisation risks can be broadly classified into 2 categories:

1. Those that can be mitigated with contract modifications. Examples include:
2. ensuring upper / lower proper bounds on key variables (fees not exceeding max threshold, for instance)
3. adding safeguards and conditional checks (require statements)
4. Those that can't be strictly enforced
5. Use multisig, put admin under timelock

> Category 1 can be separated into the various attack vectors and actors (admin / strategist), as the mitigation is more tangible in nature. This way, the recommended fixes can also be easily identified and adopted. A warden that grouped multiple attack vectors together will have their issue made the primary issue; the rest will be marked as duplicates of it.

> Regarding category 2, for issues that are generic "put admin under timelock" without explaining how and why a compromised owner / strategist can rug, as per the rulebook and judges' general consensus, I will downgrade their severity to QA. Those that explained the impact and vulnerability in detail will be grouped together with medium severity because there isn't much that can be done about it.

[M-10] StakingRewards.sol#notifyRewardAmount() Improper reward balance checks can make some users unable to withdraw their rewards

Submitted by cccz, also found by csanuragjain and pashov

Similar to https://github.com/code-423n4/2022-02-concur-findings/issues/209

```
        uint256 balance = rewardsToken.balanceOf(address(this));
```

```
            require(
                rewardRate <= balance.div(rewardsDuration),
                "Provided reward too high"
            );
```

In the current implementation, the contract only checks if balanceOf rewardsToken is greater than or equal to the future rewards.

However, under normal circumstances, since users can not withdraw all their rewards in time, the balance in the contract contains rewards that belong to the users but have not been withdrawn yet. This means the current checks can not be sufficient enough to make sure the contract has enough amount of rewardsToken.

As a result, if the rewardsDistribution mistakenly notifyRewardAmount with a larger amount, the contract may end up in a wrong state that makes some users unable to claim their rewards.

Given:

- rewardsDuration = 7 days;
- Alice stakes 1,000 stakingToken;
- rewardsDistribution sends 100 rewardsToken to the contract;
- rewardsDistribution calls `notifyRewardAmount()` with amount = 100;
- 7 days later, Alice calls `earned()` and it returns 100 rewardsToken, but Alice choose not to `getReward()` for now;
- rewardsDistribution calls `notifyRewardAmount()` with amount = 100 without send any fund to contract, the tx will succeed;
- 7 days later, Alice calls `earned()` 200 rewardsToken, when Alice tries to call `getReward()`, the transaction will fail due to insufficient balance of rewardsToken.

Expected Results:

The tx in step 5 should revert.

Proof of Concept

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/rewards/StakingRewards.sol#L201-L205

Recommended Mitigation Steps

Consider changing the function notifyRewardAmount to addRward and use transferFrom to transfer rewardsToken into the contract:

```
    function addRward(uint256 reward)
        external
        updateReward(address(0))
    {
        require(
            msg.sender == rewardsDistribution,
            "Caller is not RewardsDistribution contract"
        );

        if (block.timestamp >= periodFinish) {
            rewardRate = reward / rewardsDuration;
        } else {
            uint256 remaining = periodFinish - block.timestamp;
            uint256 leftover = remaining * rewardRate;
            rewardRate = (reward + leftover) / rewardsDuration;
        }

        rewardsToken.safeTransferFrom(msg.sender, address(this), rew

        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp + rewardsDuration;
        emit RewardAdded(reward);
    }
```

MiguelBits (Y2K Finance) disputed

[M-11] StakingRewards reward rate can be dragged out and diluted

Submitted by cccz

Similar to https://github.com/code-423n4/2022-02-concur-findings/issues/183.

```
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(rewardsDuration);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(rewardsDuratic
        }
```

The StakingRewards.notifyRewardAmount function receives a reward amount and extends the current reward end time to now + rewardsDuration. It rebases the currently remaining rewards + the new rewards (reward + leftover) over this new rewardsDuration period. This can lead to a dilution of the reward rate and rewards being dragged out forever by malicious new reward deposits.

Proof of Concept

Imagine the current rewardRate is 1000 rewards / rewardsDuration.

20% of the rewardsDuration passed, i.e., now = lastUpdateTime + 20% * rewardsDuration.

A malicious actor notifies the contract with a reward of 0: notifyRewardAmount(0).

Then the new rewardRate = (reward + leftover) / rewardsDuration = (0 + 800) / rewardsDuration = 800 / rewardsDuration.

The rewardRate just dropped by 20%. This can be repeated infinitely. After another 20% of reward time passed, they trigger notifyRewardAmount(0) to reduce it by another 20% again: rewardRate = (0 + 640) / rewardsDuration = 640 / rewardsDuration.

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/rewards/StakingRewards.sol#L183-L195

Recommended Mitigation Steps

The rewardRate should never decrease by a notifyRewardAmount call. Consider not extending the reward payouts by rewardsDuration on every call. periodFinish probably shouldn't change at all, the rewardRate should just increase by rewardRate += reward / (periodFinish - block.timestamp).

Alternatively, consider keeping the rewardRate constant but extend periodFinish time by += reward / rewardRate.

MiguelBits (Y2K Finance) disputed

HickupHH3 (judge) commented:

> Admin privilege issue that would allow the admin to dilute current rewards. Medium severity due to loss of yield for all depositors from dilution of `rewardRate`.

[M-12] After the vault expires, users may still receive rewards through the StakingRewards contract

Submitted by cccz

When the triggerEndEpoch function of the Controller contract is called, the assets in the insrVault will be sent to the riskVault, which also means that the tokens in the insrVault will be worthless.

```
    insrVault.setClaimTVL(epochEnd, 0);
    ...
            else {
                entitledAmount = amount.divWadDown(idFinalTVL[id]).m
                    idClaimTVL[id],
                    1 ether
                );
            }
```

However, if the periodFinish > _epochEnd in the StakingRewards contract corresponding to the insrVault, the user can continue to stake his insrToken and receive rewards.

Proof of Concept

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/Controller.sol#L223

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/Vault.sol#L418-L423

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/rewards/StakingRewards.sol#L183-L210

Recommended Mitigation Steps

Add the following code at the end of the notifyRewardAmount function of the StakingRewards contract to limit the periodFinish

```
            lastUpdateTime = block.timestamp;
            periodFinish = block.timestamp.add(rewardsDuration);
+          require(periodFinish <= id);
```

MiguelBits (Y2K Finance) disputed

HickupHH3 (judge) commented:

> Agree with issue; I view this as protocol leaked value (rewards) because it enables expired vault tokens that have no / little worth to "steal" rewards from future valid epochs. 1 time mint, lifetime rewards doesn't seem right.

[M-13] Different Oracle issues can return outdated prices

Submitted by 0x1f8b, also found by 0x4non, ak1, async, Chom, cryptphi, csanuragjain, datapunk, hyh, JC, Jeiwan, ladboy233, Lambda, leosathya, nalus, pashov, PwnPatrol, Rolezn, scaraven, and unforgiven https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/oracles/PegOracle.sol#L63

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/Controller.sol#L308

https://github.com/code-423n4/2022-09-y2k-finance/blob/ac3e86f07bc2f1f51148d2265cc897e8b494adf7/src/oracles/PegOracle.sol#L126

Impact

Different problems have been found with the use of the oracle that can incur economic losses when the oracle is not consumed in a completely safe way.

Proof of Concept

The problems found are:

- The `timeStamp` check is not correct since in both cases it is done against 0, which would mean that a date of 2 years ago would be valid, so old prices can be taken.

```
    function getLatestPrice(address _token)
        public
        view
        returns (int256 nowPrice)
    {
        ...
        if(timeStamp == 0)
            revert TimestampZero();
        return price;
    }
```

- Oracle price 1 can be outdated:

The `latestRoundData` method of the `PegOracle` contract calls `priceFeed1.latestRoundData();` directly, but does not perform the necessary round or timestamp checks, and delegates them to the caller, but these checks are performed on price2 because it calls `getOracle2_Price` in this case, this inconsistency between how it take the price1 and price2 behaves favors human errors when consuming the oracle.

Recommended Mitigation Steps

For the timestamp issue, it should be checked like this:

```
+   uint constant observationFrequency = 1 hours;

    function getLatestPrice(address _token)
        public
        view
        returns (int256 nowPrice)
    {
        ...
        (
            uint80 roundID,
            int256 price,
            ,
            uint256 timeStamp,
            uint80 answeredInRound
        ) = priceFeed.latestRoundData();

        uint256 decimals = 10**(18-(priceFeed.decimals()));
        price = price * int256(decimals);

        if(price <= 0)
            revert OraclePriceZero();

        if(answeredInRound < roundID)
            revert RoundIDOutdated();

-       if(timeStamp == 0)
+       if(timeStamp < block.timestamp - uint256(observationFrec
            revert TimestampZero();

        return price;
    }
```

MiguelBits (Y2K Finance) confirmed

HickupHH3 (judge) decreased severity to Medium and commented:

> Agree with the issue, but disagree with severity given. Checking for stale prices have historically been given a Medium severity rating; there isn't a compelling argument made IMO to increase it to High.

[M-14] It's possible to change for Vault and lost control on it

Submitted by rvierdiiev, also found by async

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/VaultFactory.sol#L345-L359

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Controller.sol#L136

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Controller.sol#L152

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/VaultFactory.sol#L187-L190

Impact

`VaultFactory` allows admin to change `controller` for marketId(hedge and risk vaults) using `VaultFactory.changeController` . This method then set controller to both vaults. This address is important for `Vault` contract as it allows to call different functions.

`VaultFactory` take care about different pair vaults through `indexVaults` mapping. `Controller` can get info about pairs vaults only through the correct `VaultFactory` that is provided to `Controller` in constructor.

It's possible that `VaultFactory.changeController` will set controller whose `vaultFactory` field is not equal to current `VaultFactory` . That means that when `Controller.triggerDepeg` or `Controller.triggerEndEpoch` will be called they will not be able to find the market.

So current controller will not be able to call hedge and risk vaults.

Proof of Concept

This is how the `controller` is set to vaults. https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/VaultFactory.sol#L345-L359

Controller depends on `VaultFactory` to find vault for market.
https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Controller.sol#L136

https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/Controller.sol#L152

Recommended Mitigation Steps

Use same check as you used in `VaultFactory.createNewMarket`
https://github.com/code-423n4/2022-09-y2k-finance/blob/main/src/VaultFactory.sol#L187-L190

MiguelBits (Y2K Finance) disputed

HickupHH3 (judge) commented:

> Agree with the issue that the incoming `Controller`'s `VaultFactory` should be verified to be the VaultFactory's address itself. Otherwise, there's a loss of functionality.

[M-15] Rewards are not rolled over

Submitted by csanuragjain

If there is no deposit for sometime in start then rewards for those period are never used.

Proof of Concept

1. Admin has added reward which made reward rate as 10 reward per second using notifyRewardAmount function

    ```
    rewardRate = reward.div(rewardsDuration);
    ```

2. For initial 10 seconds there were no deposits which means total supply was 0
3. So no reward were distributed for initial 10 seconds and reward for this duration which is `10*10=100` will remain in contract
4. Since on notifying contract of new rewards, these stuck rewards are not considered, these 100 rewards will remain in contract with no usage

Recommended Mitigation Steps

On very first deposit, better to have (block.timestamp-startTime) * rewardRate amount of reward being marked unused which can be used in next notifyrewardamount.

MiguelBits (Y2K Finance) disputed

HickupHH3 (judge) commented:

> I see this as protocol leaked value since the rewards would be "lost" and isn't attributed to anyone.

> Currently, the sweeper function allows the reward token to be withdrawn, thus providing a form of recovery. However, #49 and its dups points out that this is a vuln, and if fixed, will remove this recovery.

[M-16] function changeController() has rug potential as admin can unilaterally withdraw all user funds from both risk and insure vaults

Submitted by yixxas

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L29

5

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L360-L366

Impact

Admin can rug all user funds in every vault deployed by changing the controller address.

Proof of Concept

Controller can be changed by admin anytime without any warning to users.

Vault.sol#L295

```
    function changeController(address _controller) public onlyFa
        if(_controller == address(0))
            revert AddressZero();
        controller = _controller;
    }
```

Tokens in the vaults can then be called by the malicious contract to be transferred to their own address with `sendTokens()` .

Vault.sol#L360-L366

```
    function sendTokens(uint256 id, address _counterparty)
        public
        onlyController
        marketExists(id)
    {
        asset.transfer(_counterparty, idFinalTVL[id]);
    }
```

Recommended Mitigation Steps

Allow the change of controller address only in the `VaultFactory()` . This way, markets that have been created cannot have a different controller address, so users can be made aware of the change before choosing to make deposit of assets.

MiguelBits (Y2K Finance) marked as duplicate and commented:

> Implemented timelock as issued in another finding.

HickupHH3 (judge) decreased severity to QA and commented:

> Admin privilege finding, rationale for QA explained here

yixxas (warden) commented:

> Hi @HickupHH3. Would like to seek clarifications on why this was downgraded to QA. In the rationale given, "Those that explained the impact and vulnerability in detail will be grouped together with medium severity ".

> I believe the way a compromised admin can rug is clear in this report. `changeController()` can be used to change controller to any address. This address can then be used to call `sendTokens()` to steal all assets in every vault.

HickupHH3 (judge) increased severity to Medium and commented:

> Took another look at this.

> I disagree with the recommended fix. The purpose for having the controller changeable in the deployed instances is well intentioned for upgradeability purposes: perhaps new features are to be added to the controller, and migration to a new controller for all existing vaults is required to incur less technical debt. Needing to maintain legacy controllers isn't great from a devops POV.

> That said, based on my rationale, the issue should stand as a medium severity issue until we have the introduction of centralisation reports. Kenzo said it well:

> IMO most of these trusted-actor issues basically just describe general properties of the crypto/governance ecosystems, and do not reflect a novel problem in the design/implementation (which wardens are paid to discover). Because of this, and

> because of the circular logic, I believe we should change the rules and add a dedicated centralization report.

Low Risk and Non-Critical Issues

For this contest, 59 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **Respx** received the top score from the judge.

The following wardens also submitted reports: 0x1f8b, 0xc0ffEE, 0xDecorativePineapple, 0xmuxyz, 0xNazgul, 0xPanas, 0xSmartContract, ajtra, ak1, async, auditor0517, Aymen0909, Bahurum, Bnke0x0, brgltd, c3phas, carrotsmuggler, cccz, CodingNameKiki, csanuragjain, datapunk, Deivitto, djxploit, durianSausage, eierina, erictee, gogo, imare, Jeiwan, joestakey, jonatascm, kv, ladboy233, Lambda, leosathya, lukris02, oyc_109, pashov, pauliax, Picodes, PwnPatrol, R2, RaymondFam, rbserver, robee, rokinot, Rolezn, Ruhum, rvierdiiev, Saintcode_, scaraven, simon135, SooYa, Tointer, unforgiven, V_B, wagmi, *and* zzzitron.

Overview

The code base would benefit greatly from a clear, simple overview of the protocol. After reading the contest description many times, it took me many hours to realise two key points:

- Y2K doesn't actually deal with the tokens that are being hedged/insured. Instead, users deposit WETH and they are making bets with WETH that the assets will or won't depeg.
- The amounts in the hedge/risk vaults do not have to match, so one could be 10x the other, meaning the rewards for each type of depeg will vary according to market confidence.

Perhaps other wardens found these points self evident, but they were the insights I did not notice in the description which caused the ideas to "click" for me.

Contract files should include file header comments that give an overview of the contract, similar to the descriptions currently on the competition page.

Most functions do have comment headers, many of them helpful.

The code would also be clearer if significantly more comments were added within functions. Almost every line of code can usually be clarified, and this makes reading the code a great deal easier.

The first point in the *Non Critical* section is also particularly worth noting as it runs throughout the code.

Non-Critical Issues

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#L162

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L336

The use of `epochEnd` as an ID number which exists alongside the marketplace ID number is needlessly confusing. Frequently, `epochEnd` is submitted to a function as a parameter and the parameter is called `id` (I have linked one example above, but there are more). This is very difficult to read. I strongly recommend renaming `epochEnd` as something like `epochEndId` or `epochId` and consistently using that for this value. It would also help reduce ambiguity to consistently name the marketplace ID variables something like `marketId`.

VaultFactory.sol

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/VaultFactory.sol#L242

Further to the point above, I think this comment implies that the parameter `index` is actually the value of `epochEnd`, whereas is should be the market index.

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/VaultFactory.sol#L268

`VaultFactory._createEpoch()` is missing its comment header

Controller.sol

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#L317

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#L12

Consider renaming this function `getVaultFactory()` to `getVaultFactoryAddress()` simply because its purpose seems to be to return an address, not a `VaultFactory` like the public getter on line 12.

PegOracle.sol

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/oracles/PegOracle.sol#L68-L70

Consider writing `10000` as `10_000` for clarity. https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/oracles/PegOracle.sol#L78

Similarly, consider writing `1000000` as `1_000_000` for clarity.

Vault.sol

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L150

The comment on line 150 should be punctuated, “from its shares”, similar to line 180 (ie. remove the apostrophe on 150).

Low Risk Issues

VaultFactory.sol

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/VaultFactory.sol#L312

`VaultFactory.changeTreasury()` modifies both the state variable `VaultFactory.treasury` but also the treasury value of the specified market index. This has gas implications, submitted separately, but could also cause an administration error. A change to a specific market ID might not be expected to change `VaultFactory`'s `treasury` state variable. Or a change that updates the state variable might be assumed to update all vaults.

Consider separating the function into two: one to change the state variable, the other to change the treasury within specified market vaults. This is the structure used in `setController()/changeController()`.

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/VaultFactory.sol#L184

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/VaultFactory.sol#L221

In `createNewMarket`, because of the test on line 221, if `tokenToOracle[_token]` is already set, the value of `_oracle` will be silently discarded. This could have security implications if the previous oracle is no longer considered reliable. If a new oracle address is provided, it should be updated. Consider changing line 221 to:

```
if (tokenToOracle[_token] != _oracle) {
```

There should also be a `require` that `_oracle` is not the zero address.

Controller.sol

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#

L102

This test on line 102 should probably be `vault.idEpochBegin(epochEnd) >= block.timestamp)` as users will probably expect a begin time to be inclusive.

Vault.sol

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L110

The comment on line 110 is incorrect: `_token` is the token address, not the oracle address.

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Vault.sol#L265

The comment on line 265 is incorrect. It reads:

// 0.5% = multiply by 1000 then divide by 5

It should be:

// 0.5% = multiply by 5 then divide by 1000

PegOracle.sol and Controller.sol

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/oracles/PegOracle.sol#L73

https://github.com/code-423n4/2022-09-y2k-finance/blob/2175c044af98509261e4147edeb48e1036773771/src/Controller.sol#L299

Both these contracts have expressions of the form `10**(18-(priceFeed.decimals()));` . Consider the case where the value of `priceFeed.decimals()` is greater than 18. This will cause both of these lines to revert, rendering the protocol incompatible with any such price feeds.

HickupHH3 (judge) commented:

> While this QA doesn't contain as many findings as other reports, I'm selecting it for the report because:

- even though a couple of findings were "stock" issues, the descriptions are specific to the codebase
- contains a couple of noteworthy suggestions, like the first NC issue
- bonus points for an overview

Gas Optimizations

For this contest, 72 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **pfapostol** received the top score from the judge.

The following wardens also submitted reports: __141345__, _Adam, 0x040, 0x1f8b, 0x4non, 0xc0ffEE, 0xkatana, 0xNazgul, 0xSmartContract, ajtra, ak1, async, Aymen0909, BnkeOxO, zishansami, c3phas, ch0bu, cryptostellar5, d3e4, Deivitto, delfin454000, dharma09, Diana, djxploit, durianSausage, eierina, erictee, fatherOfBlocks, gianganhnguyen, gogo, ignacio, imare, jag, JAGADESH, jonatascm, Klntern_NA, Lambda, leosathya, lukris02, malinariy, MiloTruck, oyc_109, pashov, pauliax, peanuts, peiw, prasantgupta52, R2, RaymondFam, Respx, ReyAdmirado, robee, Rohan16, RoiEvenHaim, rokinot, Rolezn, rotcivegaf, Ruhum, rvierdiiev, Saintcode_, Samatak, seyni, simon135, slowmoses, Sm4rty, SnowMan, sryysryy, tnevler, Tomio, Tomo, *and* WilliamAmbrozic.

Gas Optimizations Summary

Gas savings are estimated using the gas report of existing `forge test --gas-report --fork-url https://arb1.arbitrum.io/rpc` tests (the sum of all deployment costs and the sum of the costs of calling methods) and may vary depending on the implementation of the fix.

	Issue	Instances	Estimated gas(deployments)	Estimated gas(avg method call)
[G-01]	Use custom errors rather than revert()/require() strings to save gas	13	105 127	134 279
[G-02]	Cache the results of an external function instead of calling it again	5	102 794	7 512
[G-03]	Modifiers are redundant if used only once or not used at all.	5	33 649	2 700
[G-04]	Using bools for storage incurs overhead	1	31 843	11 635
[G-05]	State variables should be cached in stack variables rather than re-reading them from storage	9	28 932	27 525

Total: 33 instances over 5 issues

[G-01] Use custom errors rather than revert()/require() strings to save gas (13 instances)

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hitby avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas

- Deployment. Gas Saved: **105 127**
- Minumal Method Call. Gas Saved: **72**
- Average Method Call. Gas Saved: **134 279**
- Maximum Method Call. Gas Saved: **139 061**

- src/SemiFungibleVault.sol:91, 116-119

```
diff --git a/src/SemiFungibleVault.sol b/src/SemiFungibleVault.s
index caf8eb7..da2bc98 100644
--- a/src/SemiFungibleVault.sol
+++ b/src/SemiFungibleVault.sol
@@ -9,6 +9,9 @@ import {
    9,    9: } from "@openzeppelin/contracts/token/ERC1155/extens
```

```
   10,  10: import {ERC1155} from "@openzeppelin/contracts/toker
   11,  11:
+       12:+error OnlyOwnerOrApproved();
+       13:+error ZeroShares();
+       14:+
   12,  15: abstract contract SemiFungibleVault is ERC1155Supply
   13,  16:     using SafeTransferLib for ERC20;
   14,  17:     using FixedPointMathLib for uint256;
@@ -88,7 +91,7 @@ abstract contract SemiFungibleVault is ERC1155
   88,  91:         address receiver
   89,  92:     ) public virtual returns (uint256 shares) {
   90,  93:         // Check for rounding error since we round c
-  91     :-         require((shares = previewDeposit(id, assets)
+       94:+         if((shares = previewDeposit(id, assets)) ==
   92,  95:
   93,  96:         // Need to transfer before minting or ERC777
   94,  97:         asset.safeTransferFrom(msg.sender, address(t
@@ -113,10 +116,8 @@ abstract contract SemiFungibleVault is ERC1
  113, 116:         address receiver,
  114, 117:         address owner
  115, 118:     ) external virtual returns (uint256 shares) {
- 116     :-         require(
- 117     :-             msg.sender == owner || isApprovedForAll(
- 118     :-             "Only owner can withdraw, or owner has a
- 119     :-         );
+      119:+         if(msg.sender != owner && !isApprovedForAll(
+      120:+
  120, 121:
  121, 122:         shares = previewWithdraw(id, assets); // No
  122, 123:
```

- src/Vault.sol:165, 187

```
diff --git a/src/Vault.sol b/src/Vault.sol
index 1d2e6df..a7e58b9 100644
--- a/src/Vault.sol
+++ b/src/Vault.sol
@@ -162,7 +162,7 @@ contract Vault is SemiFungibleVault, Reentra
  162, 162:         returns (uint256 shares)
  163, 163:     {
  164, 164:         // Check for rounding error since we round c
- 165     :-         require((shares = previewDeposit(id, assets)
+      165:+         if((shares = previewDeposit(id, assets)) ==
  166, 166:
  167, 167:         asset.transferFrom(msg.sender, address(this)
  168, 168:
@@ -184,7 +184,7 @@ contract Vault is SemiFungibleVault, Reentra
  184, 184:         payable
  185, 185:         returns (uint256 shares)
  186, 186:     {
- 187     :-         require(msg.value > 0, "ZeroValue");
+      187:+         if(msg.value == 0) revert ZeroValue();
  188, 188:
  189, 189:         IWETH(address(asset)).deposit{value: msg.val
  190, 190:         assert(IWETH(address(asset)).transfer(msg.se
```

- src/oracles/PegOracle.sol:23-25, 28-31, 98-103, 121-126

```
diff --git a/src/oracles/PegOracle.sol b/src/oracles/PegOracle.s
index 1c65268..31f1362 100644
--- a/src/oracles/PegOracle.sol
+++ b/src/oracles/PegOracle.sol
@@ -3,6 +3,13 @@ pragma solidity 0.8.15;
    3,   3:
    4,   4: import "@chainlink/interfaces/AggregatorV3Interface.
    5,   5:
+        6:+error ZeroTimestamp();
+        7:+error OutdatedOracle();
+        8:+error InvalidPrice();
+        9:+error InvalidDecimals();
+       10:+error SameOracle();
+       11:+error ZeroAddress();
+       12:+
    6,  13: contract PegOracle {
    7,  14:     /***
    8,  15:     @dev  for example: oracle1 would be stETH / USD,
@@ -20,15 +27,13 @@ contract PegOracle {
   20,  27:       * @param _oracle2 Second oracle address
   21,  28:       */
   22,  29:     constructor(address _oracle1, address _oracle2)
-  23     :-         require(_oracle1 != address(0), "oracle1 car
-  24     :-         require(_oracle2 != address(0), "oracle2 car
-  25     :-         require(_oracle1 != _oracle2, "Cannot be sam
+       30:+         if(_oracle1 == address(0)) revert ZeroAddres
+       31:+         if(_oracle2 == address(0)) revert ZeroAddres
+       32:+         if(_oracle1 == _oracle2) revert SameOracle()
```

```
   26,  33:        priceFeed1 = AggregatorV3Interface(_oracle1)
   27,  34:        priceFeed2 = AggregatorV3Interface(_oracle2)
-  28     :-       require(
-  29     :-           (priceFeed1.decimals() == priceFeed2.dec
-  30     :-           "Decimals must be the same"
-  31     :-       );
+       35:+       if((priceFeed1.decimals() != priceFeed2.deci
+       36:+
   32,  37:
   33,  38:        oracle1 = _oracle1;
   34,  39:        oracle2 = _oracle2;
@@ -95,12 +100,9 @@ contract PegOracle {
   95, 100:            uint80 answeredInRound1
   96, 101:        ) = priceFeed1.latestRoundData();
   97, 102:
-  98     :-       require(price1 > 0, "Chainlink price <= 0");
-  99     :-       require(
- 100     :-           answeredInRound1 >= roundID1,
- 101     :-           "RoundID from Oracle is outdated!"
- 102     :-       );
- 103     :-       require(timeStamp1 != 0, "Timestamp == 0 !")
+      103:+       if(price1 <= 0) revert InvalidPrice();
+      104:+       if(answeredInRound1 < roundID1) revert Outda
+      105:+       if(timeStamp1 == 0) revert ZeroTimestamp();
  104, 106:
  105, 107:        return price1;
  106, 108:    }
@@ -118,12 +120,9 @@ contract PegOracle {
  118, 120:            uint80 answeredInRound2
  119, 121:        ) = priceFeed2.latestRoundData();
  120, 122:
- 121     :-       require(price2 > 0, "Chainlink price <= 0");
- 122     :-       require(
- 123     :-           answeredInRound2 >= roundID2,
- 124     :-           "RoundID from Oracle is outdated!"
- 125     :-       );
- 126     :-       require(timeStamp2 != 0, "Timestamp == 0 !")
+      123:+       if(price2 <= 0) revert InvalidPrice();
+      124:+       if(answeredInRound2 < roundID2) revert Outda
+      125:+       if(timeStamp2 == 0) revert ZeroTimestamp();
  127, 126:
  128, 127:        return price2;
  129, 128:    }
```

- src/rewards/StakingRewards.sol:96, 119, 202-205, 217-220, 226-229

```
diff --git a/src/rewards/StakingRewards.sol b/src/rewards/Stakir
index 5edb4e8..3d59541 100644
--- a/src/rewards/StakingRewards.sol
+++ b/src/rewards/StakingRewards.sol
@@ -18,6 +18,11 @@ import {IERC1155} from "@openzeppelin/contrac
   18,  18: import {ERC20} from "@solmate/tokens/ERC20.sol";
   19,  19: import "./Owned.sol";
   20,  20:
+       21:+error RewardPeriodMustComplite();
+       22:+error InvalidToken();
+       23:+error RewardTooHigh();
+       24:+error ZeroAmount();
+       25:+
   21,  26: // https://docs.synthetix.io/contracts/source/contra
   22,  27: contract StakingRewards is
   23,  28:     IStakingRewards,
@@ -93,7 +98,7 @@ contract StakingRewards is
   93,  98:         whenNotPaused
   94,  99:         updateReward(msg.sender)
   95, 100:     {
-  96     :-         require(amount != 0, "Cannot stake 0");
+      101:+         if(amount == 0) revert ZeroAmount();
   97, 102:         _totalSupply = _totalSupply.add(amount);
   98, 103:         _balances[msg.sender] = _balances[msg.sender
   99, 104:         stakingToken.safeTransferFrom(
@@ -116,7 +121,7 @@ contract StakingRewards is
  116, 121:         nonReentrant
  117, 122:         updateReward(msg.sender)
  118, 123:     {
- 119     :-         require(amount > 0, "Cannot withdraw 0");
+      124:+         if(amount == 0) revert ZeroAmount();
  120, 125:         _totalSupply = _totalSupply.sub(amount);
  121, 126:         _balances[msg.sender] = _balances[msg.sender
  122, 127:         stakingToken.safeTransferFrom(
@@ -199,10 +204,7 @@ contract StakingRewards is
  199, 204:         // very high values of rewardRate in the ear
  200, 205:         // Reward + leftover must be less than 2^256
  201, 206:         uint256 balance = rewardsToken.balanceOf(adc
- 202     :-         require(
- 203     :-             rewardRate <= balance.div(rewardsDuratic
- 204     :-             "Provided reward too high"
- 205     :-         );
+      207:+         if(rewardRate > balance.div(rewardsDuration)
  206, 208:
  207, 209:           lastUpdateTime = block.timestamp;
  208, 210:           periodFinish = block.timestamp.add(rewardsDu
@@ -214,19 +216,13 @@ contract StakingRewards is
  214, 216:           external
  215, 217:           onlyOwner
  216, 218:       {
- 217     :-          require(
- 218     :-              tokenAddress != address(stakingToken),
- 219     :-              "Cannot withdraw the staking token"
- 220     :-          );
+      219:+          if(tokenAddress == address(stakingToken)) re
  221, 220:           ERC20(tokenAddress).safeTransfer(owner, toke
  222, 221:           emit Recovered(tokenAddress, tokenAmount);
  223, 222:       }
  224, 223:
  225, 224:       function setRewardsDuration(uint256 _rewardsDura
- 226     :-          require(
- 227     :-              block.timestamp > periodFinish,
- 228     :-              "Previous rewards period must be complet
- 229     :-          );
+      225:+          if(block.timestamp <= periodFinish) revert P
  230, 226:           rewardsDuration = _rewardsDuration;
  231, 227:           emit RewardsDurationUpdated(rewardsDuration)
  232, 228:       }
```

[G-02] Cache the results of an external function instead of calling it again (5 instances)

- Deployment. Gas Saved: **102 794**
- Minumal Method Call. Gas Saved: **-53**
- Average Method Call. Gas Saved: **7 512**
- Maximum Method Call. Gas Saved: **7 514**

- src/Controller.sol:199-200, 202-203, 206-209

```
diff --git a/src/Controller.sol b/src/Controller.sol
index e15b0fa..54503d7 100644
--- a/src/Controller.sol
+++ b/src/Controller.sol
@@ -76,39 +76,6 @@ contract Controller {
   76,  76:          _;
   77,  77:      }
   78,  78:
-  79     :-     /** @notice Modifier to ensure market exists, cu
-  80     :-       * @param marketIndex Target market index
-  81     :-       * @param epochEnd End of epoch set for market
-  82     :-       */
-  83     :-     modifier isDisaster(uint256 marketIndex, uint256
-  84     :-         address[] memory vaultsAddress = vaultFactor
-  85     :-         if(
-  86     :-             vaultsAddress.length != VAULTS_LENGTH
-  87     :-             )
-  88     :-             revert MarketDoesNotExist(marketIndex);
-  89     :-
-  90     :-         address vaultAddress = vaultsAddress[0];
-  91     :-         Vault vault = Vault(vaultAddress);
-  92     :-
-  93     :-         if(vault.idExists(epochEnd) == false)
-  94     :-             revert EpochNotExist();
-  95     :-
-  96     :-         if(
-  97     :-             vault.strikePrice() < getLatestPrice(vau
-  98     :-             )
-  99     :-             revert PriceNotAtStrikePrice(getLatestPr
- 100     :-
- 101     :-         if(
- 102     :-             vault.idEpochBegin(epochEnd) > block.tim
- 103     :-             revert EpochNotStarted();
- 104     :-
- 105     :-         if(
- 106     :-             block.timestamp > epochEnd
- 107     :-             )
- 108     :-             revert EpochExpired();
- 109     :-         _;
- 110     :-     }
- 111     :-
  112,  79:      /*//////////////////////////////////////////////
  113,  80:                                     CONSTRUCTOR
  114,  81:      //////////////////////////////////////////////
@@ -147,12 +114,27 @@ contract Controller {
  147, 114:        */
  148, 115:      function triggerDepeg(uint256 marketIndex, uint2
  149, 116:          public
- 150     :-         isDisaster(marketIndex, epochEnd)
  151, 117:      {
  152, 118:          address[] memory vaultsAddress = vaultFactor
```

```
+      119:+        if(vaultsAddress.length != VAULTS_LENGTH)
+      120:+            revert MarketDoesNotExist(marketIndex);
+      121:+
  153, 122:         Vault insrVault = Vault(vaultsAddress[0]);
  154, 123:         Vault riskVault = Vault(vaultsAddress[1]);
  155, 124:
+      125:+        if(insrVault.idExists(epochEnd) == false)
+      126:+            revert EpochNotExist();
+      127:+
+      128:+        int256 nowPrice;
+      129:+        if(insrVault.strikePrice() < (nowPrice = get
+      130:+            revert PriceNotAtStrikePrice(nowPrice);
+      131:+
+      132:+        if(insrVault.idEpochBegin(epochEnd) > block.
+      133:+            revert EpochNotStarted();
+      134:+
+      135:+        if(block.timestamp > epochEnd)
+      136:+            revert EpochExpired();
+      137:+
  156, 138:         //require this function cannot be called twi
  157, 139:         if(insrVault.idFinalTVL(epochEnd) != 0)
  158, 140:             revert NotZeroTVL();
@@ -196,17 +178,14 @@ contract Controller {
  196, 178:       * @param epochEnd End of epoch set for market
  197, 179:       */
  198, 180:     function triggerEndEpoch(uint256 marketIndex, ui
- 199     :-        if(
- 200     :-            vaultFactory.getVaults(marketIndex).leng
+      181:+        address[] memory vaults = vaultFactory.getVa
+      182:+        if(vaults.length != VAULTS_LENGTH)
  201, 183:                 revert MarketDoesNotExist(marketInde
- 202     :-        if(
- 203     :-            block.timestamp < epochEnd)
+      184:+        if(block.timestamp < epochEnd)
  204, 185:             revert EpochNotExpired();
  205, 186:
- 206     :-        address[] memory vaultsAddress = vaultFactor
- 207     :-
- 208     :-        Vault insrVault = Vault(vaultsAddress[0]);
- 209     :-        Vault riskVault = Vault(vaultsAddress[1]);
+      187:+        Vault insrVault = Vault(vaults[0]);
+      188:+        Vault riskVault = Vault(vaults[1]);
  210, 189:
  211, 190:         if(insrVault.idExists(epochEnd) == false ||
  212, 191:             revert EpochNotExist();
```

- src/oracles/PegOracle.sol:29

```
diff --git a/src/oracles/PegOracle.sol b/src/oracles/PegOracle.s
index 1c65268..8387ed9 100644
--- a/src/oracles/PegOracle.sol
+++ b/src/oracles/PegOracle.sol
@@ -25,15 +25,16 @@ contract PegOracle {
   25,  25:          require(_oracle1 != _oracle2, "Cannot be sam
   26,  26:          priceFeed1 = AggregatorV3Interface(_oracle1)
   27,  27:          priceFeed2 = AggregatorV3Interface(_oracle2)
+       28:+         uint8 _decimals;
   28,  29:          require(
-  29     :-              (priceFeed1.decimals() == priceFeed2.dec
+       30:+              ((_decimals = priceFeed1.decimals()) ==
   30,  31:              "Decimals must be the same"
   31,  32:          );
   32,  33:
   33,  34:          oracle1 = _oracle1;
   34,  35:          oracle2 = _oracle2;
   35,  36:
-  36     :-          decimals = priceFeed1.decimals();
+       37:+         decimals = _decimals;
   37,  38:      }
   38,  39:
   39,  40:      /** @notice Returns oracle-fed data from the lat
```

- src/rewards/RewardsFactory.sol:90-91

```
diff --git a/src/rewards/RewardsFactory.sol b/src/rewards/Rewarc
index 8bee8bd..a679348 100644
--- a/src/rewards/RewardsFactory.sol
+++ b/src/rewards/RewardsFactory.sol
@@ -87,8 +87,9 @@ contract RewardsFactory {
   87,  87:      {
   88,  88:          VaultFactory vaultFactory = VaultFactory(fac
   89,  89:
-  90     :-          address _insrToken = vaultFactory.getVaults
-  91     :-          address _riskToken = vaultFactory.getVaults
+       90:+         address[] memory vaults = vaultFactory.getVa
+       91:+         address _insrToken = vaults[0];
+       92:+         address _riskToken = vaults[1];
   92,  93:
```

```
   93,  94:         if(_insrToken == address(0) || _riskToken ==
   94,  95:             revert MarketDoesNotExist(_marketIndex);
```

[G-03] Modifiers are redundant if used only once or not used at all. (5 instances)

- Deployment. Gas Saved: **33 649**
- Minumal Method Call. Gas Saved: **-202**
- Average Method Call. Gas Saved: **2 700**
- Maximum Method Call. Gas Saved: **4 931**

- src/Controller.sol:67-111

```
diff --git a/src/Controller.sol b/src/Controller.sol
index e15b0fa..b8cafb5 100644
--- a/src/Controller.sol
+++ b/src/Controller.sol
@@ -64,51 +64,6 @@ contract Controller {
   64,  64:     }
   65,  65:     /* solhint-enable  var-name-mixedcase */
   66,  66:
-  67     :-    /*////////////////////////////////////////////////
-  68     :-                                 MODIFIERS
-  69     :-    ////////////////////////////////////////////////
-  70     :-
-  71     :-    /** @notice Only admin addresses can call functi
-  72     :-      */
-  73     :-    modifier onlyAdmin() {
-  74     :-        if(msg.sender != admin)
-  75     :-            revert AddressNotAdmin();
-  76     :-        _;
-  77     :-    }
-  78     :-
-  79     :-    /** @notice Modifier to ensure market exists, cu
-  80     :-      * @param marketIndex Target market index
-  81     :-      * @param epochEnd End of epoch set for market
-  82     :-      */
-  83     :-    modifier isDisaster(uint256 marketIndex, uint256
-  84     :-        address[] memory vaultsAddress = vaultFactor
-  85     :-        if(
-  86     :-            vaultsAddress.length != VAULTS_LENGTH
- 87      :-              )
- 88      :-              revert MarketDoesNotExist(marketIndex);
- 89      :-
- 90      :-          address vaultAddress = vaultsAddress[0];
- 91      :-          Vault vault = Vault(vaultAddress);
- 92      :-
- 93      :-          if(vault.idExists(epochEnd) == false)
- 94      :-              revert EpochNotExist();
- 95      :-
- 96      :-          if(
- 97      :-              vault.strikePrice() < getLatestPrice(vau
- 98      :-              )
- 99      :-              revert PriceNotAtStrikePrice(getLatestPr
- 100     :-
- 101     :-          if(
- 102     :-              vault.idEpochBegin(epochEnd) > block.tim
- 103     :-              revert EpochNotStarted();
- 104     :-
- 105     :-          if(
- 106     :-              block.timestamp > epochEnd
- 107     :-              )
- 108     :-              revert EpochExpired();
- 109     :-          _;
- 110     :-      }
- 111     :-
  112,  67:      /*//////////////////////////////////////////////
  113,  68:                                              CONSTRUCTOR
  114,  69:      //////////////////////////////////////////////
@@ -147,9 +102,32 @@ contract Controller {
  147, 102:        */
  148, 103:      function triggerDepeg(uint256 marketIndex, uint2
  149, 104:          public
- 150     :-          isDisaster(marketIndex, epochEnd)
  151, 105:      {
  152, 106:          address[] memory vaultsAddress = vaultFactor
+       107:+         if(
+       108:+             vaultsAddress.length != VAULTS_LENGTH
+       109:+             )
+       110:+             revert MarketDoesNotExist(marketIndex);
+       111:+
+       112:+         address vaultAddress = vaultsAddress[0];
+       113:+         Vault vault = Vault(vaultAddress);
+       114:+
+       115:+         if(vault.idExists(epochEnd) == false)
+       116:+             revert EpochNotExist();
+       117:+
+       118:+           if(
+       119:+               vault.strikePrice() < getLatestPrice(vau
+       120:+               )
+       121:+               revert PriceNotAtStrikePrice(getLatestPr
+       122:+
+       123:+           if(
+       124:+               vault.idEpochBegin(epochEnd) > block.tim
+       125:+               revert EpochNotStarted();
+       126:+
+       127:+           if(
+       128:+               block.timestamp > epochEnd
+       129:+               )
+       130:+               revert EpochExpired();
  153, 131:             Vault insrVault = Vault(vaultsAddress[0]);
  154, 132:             Vault riskVault = Vault(vaultsAddress[1]);
  155, 133:
```

- src/Vault.sol:85-100

```
diff --git a/src/Vault.sol b/src/Vault.sol
index 1d2e6df..9e480ed 100644
--- a/src/Vault.sol
+++ b/src/Vault.sol
@@ -82,22 +82,6 @@ contract Vault is SemiFungibleVault, Reentrar
   82,  82:           _;
   83,  83:       }
   84,  84:
-  85      :-      /** @notice You can only call functions that use
-  86      :-        */
-  87      :-      modifier epochHasNotStarted(uint256 id) {
-  88      :-          if(block.timestamp > idEpochBegin[id] - time
-  89      :-              revert EpochAlreadyStarted();
-  90      :-          _;
-  91      :-      }
-  92      :-
-  93      :-      /** @notice You can only call functions that use
-  94      :-        */
-  95      :-      modifier epochHasEnded(uint256 id) {
-  96      :-          if((block.timestamp < id) && idDepegged[id]
-  97      :-              revert EpochNotFinished();
-  98      :-          _;
-  99      :-      }
- 100      :-
```

```
  101,  85:     /*//////////////////////////////////////////////////
  102,  86:                                               CONSTRUCTOR
  103,  87:     //////////////////////////////////////////////////
@@ -157,10 +141,11 @@ contract Vault is SemiFungibleVault, Reent
  157, 141:         public
  158, 142:         override
  159, 143:         marketExists(id)
- 160     :-        epochHasNotStarted(id)
  161, 144:         nonReentrant
  162, 145:         returns (uint256 shares)
  163, 146:     {
+      147:+        if(block.timestamp > idEpochBegin[id] - time
+      148:+            revert EpochAlreadyStarted();
  164, 149:         // Check for rounding error since we round c
  165, 150:         require((shares = previewDeposit(id, assets)
  166, 151:
@@ -208,10 +193,11 @@ contract Vault is SemiFungibleVault, Reent
  208, 193:     )
  209, 194:         external
  210, 195:         override
- 211     :-        epochHasEnded(id)
  212, 196:         marketExists(id)
  213, 197:         returns (uint256 shares)
  214, 198:     {
+      199:+        if((block.timestamp < id) && idDepegged[id]
+      200:+            revert EpochNotFinished();
  215, 201:         if(
  216, 202:             msg.sender != owner &&
  217, 203:             isApprovedForAll(owner, receiver) == fal
```

- src/rewards/RewardsFactory.sol:50-56

```
diff --git a/src/rewards/RewardsFactory.sol b/src/rewards/Rewarc
index 8bee8bd..35c45b4 100644
--- a/src/rewards/RewardsFactory.sol
+++ b/src/rewards/RewardsFactory.sol
@@ -47,13 +47,6 @@ contract RewardsFactory {
   47,  47:                                              MODIFIERS
   48,  48:     //////////////////////////////////////////////////
   49,  49:
-  50     :-    /** @notice Only admin addresses can call functi
-  51     :-      */
-  52     :-    modifier onlyAdmin() {
```

```
-  53     :-         if(msg.sender != admin)
-  54     :-             revert AddressNotAdmin();
-  55     :-         _;
-  56     :-     }
   57,  50:
   58,  51:     /** @notice Contract constructor
   59,  52:       * @param _govToken Governance token address
@@ -82,9 +75,10 @@ contract RewardsFactory {
   82,  75:       */
   83,  76:     function createStakingRewards(uint256 _marketInc
   84,  77:         external
-  85     :-         onlyAdmin
   86,  78:         returns (address insr, address risk)
   87,  79:     {
+       80:+        if(msg.sender != admin)
+       81:+            revert AddressNotAdmin();
   88,  82:         VaultFactory vaultFactory = VaultFactory(fac
   89,  83:
   90,  84:         address _insrToken = vaultFactory.getVaults(
```

[G-04] Using bools for storage incurs overhead (1 instance)

- Deployment. Gas Saved: **31 843**
- Minumal Method Call. Gas Saved: **144**
- Average Method Call. Gas Saved: **11 635**
- Maximum Method Call. Gas Saved: **17 611**

```
// Booleans are more expensive than uint256 or any type that tak
// word because each write operation emits an extra SLOAD to fir
// slot's contents, replace the bits taken up by the boolean, ar
// back. This is the compiler's defense against contract upgrade
// pointer aliasing, and it cannot be disabled.
```

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas) for the extra SLOAD, and to avoid Gsset (20000 gas) when changing from ‘false’ to ‘true’, after having been ‘true’ in the past

- src/Vault.sol:54

```
diff --git a/src/Controller.sol b/src/Controller.sol
index e15b0fa..64c68f3 100644
--- a/src/Controller.sol
+++ b/src/Controller.sol
@@ -90,7 +90,7 @@ contract Controller {
   90,  90:          address vaultAddress = vaultsAddress[0];
   91,  91:          Vault vault = Vault(vaultAddress);
   92,  92:
-  93      :-        if(vault.idExists(epochEnd) == false)
+         93:+        if(vault.idExists(epochEnd) == 0)
   94,  94:              revert EpochNotExist();
   95,  95:
   96,  96:          if(
@@ -208,7 +208,7 @@ contract Controller {
  208, 208:          Vault insrVault = Vault(vaultsAddress[0]);
  209, 209:          Vault riskVault = Vault(vaultsAddress[1]);
  210, 210:
- 211      :-        if(insrVault.idExists(epochEnd) == false ||
+       211:+        if(insrVault.idExists(epochEnd) == 0 || risk
  212, 212:              revert EpochNotExist();
  213, 213:
  214, 214:          //require this function cannot be called twi
diff --git a/src/Vault.sol b/src/Vault.sol
index 1d2e6df..12af155 100644
--- a/src/Vault.sol
+++ b/src/Vault.sol
@@ -51,7 +51,7 @@ contract Vault is SemiFungibleVault, Reentranc
   51,  51:      // @audit id can be uint32
   52,  52:      mapping(uint256 => bool) public idDepegged;
   53,  53:      // @audit id can be uint32
-  54      :-    mapping(uint256 => bool) public idExists;
+         54:+    mapping(uint256 => uint256) public idExists;
   55,  55:      mapping(uint256 => uint256) public epochFee;
   56,  56:
   57,  57:      /*//////////////////////////////////////////////
@@ -77,7 +77,7 @@ contract Vault is SemiFungibleVault, Reentranc
   77,  77:      /** @notice Only market addresses can call funct
   78,  78:        */
   79,  79:      modifier marketExists(uint256 id) {
-  80      :-        if(idExists[id] != true)
+         80:+        if(idExists[id] != 1)
   81,  81:              revert MarketEpochDoesNotExist();
   82,  82:          _;
   83,  83:      }
@@ -311,13 +311,13 @@ contract Vault is SemiFungibleVault, Reent
```

```
  311, 311:         if(_withdrawalFee > 150)
  312, 312:             revert FeeMoreThan150(_withdrawalFee);
  313, 313:
- 314     :-        if(idExists[epochEnd] == true)
+      314:+        if(idExists[epochEnd] == 1)
  315, 315:             revert MarketEpochExists();
  316, 316:
  317, 317:         if(epochBegin >= epochEnd)
  318, 318:             revert EpochEndMustBeAfterBegin();
  319, 319:
- 320     :-        idExists[epochEnd] = true;
+      320:+        idExists[epochEnd] = 1;
  321, 321:         idEpochBegin[epochEnd] = epochBegin;
  322, 322:         epochs.push(epochEnd);
  323, 323:
diff --git a/src/rewards/RewardsFactory.sol b/src/rewards/Rewarc
index 8bee8bd..d463005 100644
--- a/src/rewards/RewardsFactory.sol
+++ b/src/rewards/RewardsFactory.sol
@@ -93,7 +93,7 @@ contract RewardsFactory {
   93,  93:         if(_insrToken == address(0) || _riskToken ==
   94,  94:             revert MarketDoesNotExist(_marketIndex);
   95,  95:
-  96     :-        if(Vault(_insrToken).idExists(_epochEnd) ==
+       96:+        if(Vault(_insrToken).idExists(_epochEnd) ==
   97,  97:             revert EpochDoesNotExist();
   98,  98:
   99,  99:         StakingRewards insrStake = new StakingRewarc
```

[G-05] State variables should be cached in stack variables rather than re-reading them from storage (9 instances)

- Deployment. Gas Saved: **28 932**
- Minumal Method Call. Gas Saved: **-59**
- Average Method Call. Gas Saved: **27 525**
- Maximum Method Call. Gas Saved: **27 999**

The code can be optimized by minimising the number of SLOADs.

SLOADs are expensive (100 gas after the 1st one) compared to MLOADs/MSTOREs (3 gas each). Storage values read multiple times should instead be cached in memory the first time (costing 1 SLOAD) and then read from this cache to avoid multiple SLOADs.

- src/Vault.sol:188-190, 228

```
diff --git a/src/Vault.sol b/src/Vault.sol
index 1d2e6df..bb4d1f2 100644
--- a/src/Vault.sol
+++ b/src/Vault.sol
@@ -185,9 +185,9 @@ contract Vault is SemiFungibleVault, Reentra
  185, 185:         returns (uint256 shares)
  186, 186:     {
  187, 187:         require(msg.value > 0, "ZeroValue");
- 188     :-
- 189     :-        IWETH(address(asset)).deposit{value: msg.val
- 190     :-        assert(IWETH(address(asset)).transfer(msg.se
+      188:+        IWETH iweth = IWETH(address(asset));
+      189:+        iweth.deposit{value: msg.value}();
+      190:+        assert(iweth.transfer(msg.sender, msg.value)
  191, 191:
  192, 192:         return deposit(id, msg.value, receiver);
  193, 193:     }
@@ -225,10 +225,12 @@ contract Vault is SemiFungibleVault, Reent
  225, 225:         //Taking fee from the amount
  226, 226:         uint256 feeValue = calculateWithdrawalFeeVal
  227, 227:         entitledShares = entitledShares - feeValue;
- 228     :-        asset.transfer(treasury, feeValue);
+      228:+
+      229:+        ERC20 _asset = asset;
+      230:+        _asset.transfer(treasury, feeValue);
  229, 231:
  230, 232:         emit Withdraw(msg.sender, receiver, owner, i
- 231     :-        asset.transfer(receiver, entitledShares);
+      233:+        _asset.transfer(receiver, entitledShares);
  232, 234:
  233, 235:         return entitledShares;
  234, 236:     }
```

- src/VaultFactory.sol:188, 195, 203

```
diff --git a/src/VaultFactory.sol b/src/VaultFactory.sol
```

```
index bfd70f1..f5a7616 100644
--- a/src/VaultFactory.sol
+++ b/src/VaultFactory.sol
@@ -184,46 +184,54 @@ contract VaultFactory {
  184, 184:          address _oracle,
  185, 185:          string memory _name
  186, 186:      ) public onlyAdmin returns (address insr, addres
+      187:+         Vault hedge;
+      188:+         Vault risk;
+      189:+         uint256 _marketIndex;
+      190:+         {
+      191:+         address _controller = controller;
  187, 192:          if(
- 188     :-              IController(controller).getVaultFactory
+      193:+              IController(_controller).getVaultFactory
  189, 194:              )
  190, 195:              revert AddressFactoryNotInController();
  191, 196:
- 192     :-          if(controller == address(0))
+      197:+          if(_controller == address(0))
  193, 198:              revert ControllerNotSet();
  194, 199:
- 195     :-          marketIndex += 1;
+      200:+          _marketIndex = (marketIndex += 1);
  196, 201:
  197, 202:          //y2kUSDC_99*RISK or y2kUSDC_99*HEDGE
  198, 203:
- 199     :-          Vault hedge = new Vault(
+      204:+          address _treasury = treasury;
+      205:+
+      206:+          hedge = new Vault(
  200, 207:              WETH,
  201, 208:              string(abi.encodePacked(_name,"HEDGE")),
  202, 209:              "hY2K",
- 203     :-              treasury,
+      210:+              _treasury,
  204, 211:              _token,
  205, 212:              _strikePrice,
- 206     :-              controller
+      213:+              _controller
  207, 214:          );
  208, 215:
- 209     :-          Vault risk = new Vault(
+      216:+          risk = new Vault(
  210, 217:              WETH,
  211, 218:              string(abi.encodePacked(_name,"RISK")),
  212, 219:                 "rY2K",
- 213     :-                treasury,
+      220:+                _treasury,
  214, 221:                 _token,
  215, 222:                 _strikePrice,
- 216     :-                controller
+      223:+                _controller
  217, 224:             );
+      225:+    }
  218, 226:
- 219     :-            indexVaults[marketIndex] = [address(hedge),
+      227:+            indexVaults[_marketIndex] = [address(hedge),
  220, 228:
  221, 229:             if (tokenToOracle[_token] == address(0)) {
  222, 230:                 tokenToOracle[_token] = _oracle;
  223, 231:             }
  224, 232:
  225, 233:             emit MarketCreated(
- 226     :-                marketIndex,
+      234:+                _marketIndex,
  227, 235:                 address(hedge),
  228, 236:                 address(risk),
  229, 237:                 _token,
@@ -231,7 +239,7 @@ contract VaultFactory {
  231, 239:                 _strikePrice
  232, 240:             );
  233, 241:
- 234     :-            MarketVault memory marketVault = MarketVault
+      242:+            MarketVault memory marketVault = MarketVault
  235, 243:
  236, 244:             _createEpoch(marketVault);
  237, 245:
```

- src/oracles/PegOracle.sol:29, 36

```
diff --git a/src/oracles/PegOracle.sol b/src/oracles/PegOracle.s
index 1c65268..108b041 100644
--- a/src/oracles/PegOracle.sol
+++ b/src/oracles/PegOracle.sol
@@ -26,14 +26,14 @@ contract PegOracle {
   26,  26:         priceFeed1 = AggregatorV3Interface(_oracle1)
   27,  27:         priceFeed2 = AggregatorV3Interface(_oracle2)
   28,  28:         require(
-  29      :-             (priceFeed1.decimals() == priceFeed2.dec
+      29:+             (AggregatorV3Interface(_oracle1).decimal
   30,  30:             "Decimals must be the same"
   31,  31:         );
   32,  32:
   33,  33:         oracle1 = _oracle1;
   34,  34:         oracle2 = _oracle2;
   35,  35:
-  36      :-       decimals = priceFeed1.decimals();
+      36:+       decimals = AggregatorV3Interface(_oracle1).c
   37,  37:     }
   38,  38:
   39,  39:     /** @notice Returns oracle-fed data from the lat
```

- src/rewards/StakingRewards.sol:160, 189-190

```
diff --git a/src/rewards/StakingRewards.sol b/src/rewards/Stakir
index 5edb4e8..345821c 100644
--- a/src/rewards/StakingRewards.sol
+++ b/src/rewards/StakingRewards.sol
@@ -157,7 +157,8 @@ contract StakingRewards is
  157, 157:     }
  158, 158:
  159, 159:     function rewardPerToken() public view returns (u
- 160      :-       if (_totalSupply == 0) {
+      160:+       uint256 totakSupply_;
+      161:+       if ((totakSupply_ = _totalSupply) == 0) {
  161, 162:           return rewardPerTokenStored;
  162, 163:       }
  163, 164:       return
@@ -166,7 +167,7 @@ contract StakingRewards is
  166, 167:                     .sub(lastUpdateTime)
  167, 168:                     .mul(rewardRate)
  168, 169:                     .mul(1e18)
- 169      :-                   .div(_totalSupply)
+      170:+                   .div(totakSupply_)
  170, 171:           );
  171, 172:     }
  172, 173:
@@ -186,12 +187,14 @@ contract StakingRewards is
  186, 187:       onlyRewardsDistribution
  187, 188:       updateReward(address(0))
  188, 189:     {
```

```
- 189      :-          if (block.timestamp >= periodFinish) {
- 190      :-              rewardRate = reward.div(rewardsDuration)
+       190:+          uint256 periodFinish_;
+       191:+          uint256 rewardsDuration_ = rewardsDuration;
+       192:+          if (block.timestamp >= (periodFinish_ = peri
+       193:+              rewardRate = reward.div(rewardsDuration_
  191, 194:          } else {
- 192      :-              uint256 remaining = periodFinish.sub(blo
+       195:+              uint256 remaining = periodFinish_.sub(bl
  193, 196:              uint256 leftover = remaining.mul(rewardR
- 194      :-              rewardRate = reward.add(leftover).div(re
+       197:+              rewardRate = reward.add(leftover).div(re
  195, 198:          }
  196, 199:
  197, 200:          // Ensure the provided reward amount is not
@@ -200,12 +203,12 @@ contract StakingRewards is
  200, 203:          // Reward + leftover must be less than 2^256
  201, 204:          uint256 balance = rewardsToken.balanceOf(add
  202, 205:          require(
- 203      :-              rewardRate <= balance.div(rewardsDuratio
+       206:+              rewardRate <= balance.div(rewardsDuratio
  204, 207:              "Provided reward too high"
  205, 208:          );
  206, 209:
  207, 210:          lastUpdateTime = block.timestamp;
- 208      :-          periodFinish = block.timestamp.add(rewardsDu
+       211:+          periodFinish = block.timestamp.add(rewardsDu
  209, 212:          emit RewardAdded(reward);
  210, 213:      }
  211, 214:
```

HickupHH3 (judge) commented:

> I selected this report as the best for a few reasons:

1. Total gas saved is the highest (another report that came close was eierina's)
2. Properly benchmarked its gas optimizations against the original implementation
3. Great formatting on changes to be made to the contracts

> I downplay the custom error gas optimisation suggestion even though it might save ~8.5k gas on average as per this twitter thread, the gas savings are only materialised upon an actual revert. There are minimally deployment costs savings

> of couse. However, the translation of `require` to custom errors is non-trivial because it requires inversion of the conditional check.

> I'd like to mention other notable (and unique) gas optimisations for the report from other wardens:

- eierina: Prefer fixed size arrays in place of dynamic sized arrays where fits (~22k gas)
- rokinot: Solmate's reentrancy guard is cheaper than OZ's. (Minimally 3k gas)
- Multiple wardens: Drop the use of SafeMath for StakingRewards. Requires updating of the `StakingRewards` contract which I understand the sponsor does not wish to modify.

Disclosures