NewOrderDAO - Y2K Finance

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 31st, 2022 - Jun 13th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/13/2022	Pawel Bartunek
0.2	Document Amended	06/14/2022	Pawel Bartunek
0.3	Document Edits	06/17/2022	Gokberk Gulgun
0.4	Draft Review	06/17/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Pawel Bartunek	Halborn	Pawel.Bartunek@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

NewOrderDAO engaged Halborn to conduct a security audit on their smart contracts beginning on May 31st, 2022 and ending on Jun 13th, 2022 . The security assessment was scoped to the smart contracts provided to the Halborn team.

The project included four main contracts:

- Core Y2K contracts - a Vault where risk and insurance users may deposit tokens
- Staking Rewards - Synthetix StakingRewards contract modified for ERC1155.
- Rewards Vault - distributes rewards to governance token holders who lock their assets
- Merkle Distributor - airdrop contract

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

Overall, the code was of good quality and thoroughly documented. Test cases were implemented and covered the majority of normal user flows of the platform.

In summary, Halborn identified some security risks that should be reviewed by the NewOrderDAO team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation, automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walk through
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions (Slither)
- Local or Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The assessment was scoped to the repositories listed below:

- Y2K core, Rewards
  - commit: **96fdcf02fa80e71c3a2e4d2cc78cbddcb3e120d3**
  - contracts:
    - Controller.sol
    - SemiFungibleVault.sol
    - Vault.sol
    - VaultFactory.sol
    - IStakingRewards.sol
    - Owned.sol
    - RewardsDistributionRecipient.sol
    - StakingRewards.sol
    -
- Rewards Vault
  - commit: **4e2df3275f94b9a6e01cf0f642f3606a74afdaf4**
  - contracts:
    - LockRewards.sol
    - ILockRewards.sol
- Merkle Distributor
  - commit: **9a2b109a2141b7e1d7795599c0b2382832cdc492**
  - contracts:
    - MerkleDistributor.sol
    - MerkleProof.sol
    - IMerkleDistributor.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	8	6	10

LIKELIHOOD

IMPACT

	(HAL-02)			
(HAL-13) (HAL-14)	(HAL-04) (HAL-07) (HAL-08)	(HAL-01)		
(HAL-10)	(HAL-11)	(HAL-05) (HAL-06)	(HAL-03)	
	(HAL-12)			
(HAL-15) (HAL-16) (HAL-17) (HAL-18) (HAL-19) (HAL-20) (HAL-21) (HAL-22) (HAL-23) (HAL-24)			(HAL-09)	

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 MISSING CONTROLLER ADDRESS VALIDATION	Medium	-
HAL-02 INTEGER UNDERFLOW - MISSING FEE VALIDATION	Medium	-
HAL-03 RELOCKING OF ASSETS DOES NOT WORK PROPERLY	Medium	-
HAL-04 CLAIMED AMOUNTS ARE MAPPED TO INDEXES IN MERKLE TREE	Medium	-
HAL-05 CONTRACT IS NOT PAUSED DURING MERKLE ROOT UPDATE	Medium	-
HAL-06 ERC4626 DOES NOT WORK WITH FEE-ON-TRANSFER TOKENS	Medium	-
HAL-07 ORACLE CAN BE OVERWRITTEN ON THE MARKET CREATION	Medium	SOLVED - 05/26/2022
HAL-08 SHOULD CHECK RETURN DATA FROM CHAINLINK AGGREGATORS	Medium	SOLVED - 05/26/2022
HAL-09 FLOATING PRAGMA	Low	-
HAL-10 MISSING PAUSE/UNPAUSE FUNCTIONALITY ON THE LOCKREWARDS CONTRACT	Low	-
HAL-11 MISSING VALIDATION	Low	-
HAL-12 IMPROPER ROLE BASED ACCESS CONTROL	Low	-
HAL-13 OWNER CAN WITHDRAW ALL TOKENS	Low	-
HAL-14 MISSING RE-ENTRANCY PROTECTION	Low	-
HAL-15 LONG ERROR MESSAGES	Informational	-
HAL-16 OPTIMIZE UNSIGNED INTEGER COMPARISON	Informational	-

HAL-17 PREFIX INCREMENTS ARE CHEAPER THAN POSTFIX INCREMENTS	Informational	-
HAL-18 CHECK AMOUNT IS GREATER THAN 0 TO AVOID UNNECESSARILY CALLING SAFETRANSFER()	Informational	-
HAL-19 ROUNDING PROBLEMS IN THE EIP 4626()	Informational	-
HAL-20 EVENTS ARE NOT INDEXED	Informational	-
HAL-21 MISSING EVENTS ON CHANGES	Informational	-
HAL-22 UINT CAN'T BE LOWER THAN ZERO	Informational	-
HAL-23 NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES	Informational	-
HAL-24 IMMUTABLE VARIABLES	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING CONTROLLER ADDRESS VALIDATION - MEDIUM

Description:

The createNewMarket function in VaultFactory.sol contract does not verify that factory has controller address set. When setController function is not called after creating VaultFactory, it is possible to create new vaults without controller address configured. Users will be able to deposit assets to such vaults, but there will be no controller functionality (ending epochs, triggering depeg events).

Code location:

Listing 1: Y2K-smartcontracts/src/VaultFactory.sol (Lines 86,97)

```
68 function createNewMarket(
69     uint256 _fee,
70     address _token,
71     int256 _strikePrice,
72     uint256 epochBegin,
73     uint256 epochEnd,
74     address _oracle
75 ) public onlyAdmin returns (address insr, address rsk) {
76     marketIndex += 1;
77
78     Vault insurance = new Vault(
79         WETH,
80         "InsrY2K",
81         "Y2k",
82         treasury,
83         _fee,
84         _token,
85         _strikePrice,
86         controller
87     );
88
89     Vault risk = new Vault(
90         WETH,
91         "RiskY2K",
        "Y2K",
        treasury,
        _fee,
        _token,
        _strikePrice,
        controller
    );
```

Proof of Concept:

Scenario:
1. Create VaultFactory and Controller instances.
2. Do not call setController function on VaultFactory
3. Create new Vaults
4. Now it is not possible to trigger depeg or end epoch from controller.

Parts of Foundry script to simulate the scenario, first create VaultFactory and Controller:

Listing 2: (Line 3)

```
govToken = new GovToken();
vaultFactory = new VaultFactory(treasury);
controller = new Controller(address(govToken), address(
↳ vaultFactory));
```

Then create Vaults, deposit USDC:

Listing 3

```
function CreationNewVaults(
    uint256 fee,
    address token,
    int256 strikePrice,
    uint256 epochBegin,
    uint256 epochEnd,
    address token_oracle
) public returns (address insr, address risk) {
    return
        vaultFactory.createNewMarket(
            fee,
            token,
            strikePrice,
            epochBegin,
            epochEnd,
            token_oracle
        );
}

function testAuditUserDepositsUSDC()
    public
    returns (
        uint256 ID,
        address _insr,
        address _risk
    )
{
    uint256 fee = 10;
    int256 strikePrice = 120000000; //1$ = 100000000
    uint256 epochBegin = block.timestamp + 3 days; // 3 days from
↳ now
    uint256 epochEnd = block.timestamp + 30 days; // 30 days from
↳ now

    (address insr, address risk) = CreationNewVaults(
        fee,
        USDC,
        strikePrice,
        epochBegin,
        epochEnd,
        USDC_oracle
    );
    DepositInsurance(address(2), 1 ether, epochEnd, insr);
    DepositRisk(address(3), 2 ether, epochEnd, risk);

    return (epochEnd, insr, risk);
}
```

Then try to trigger a depeg event from controller:

Listing 4: (Line 14)

```
// create vaults without setting controller, then try to trigger
↳ depeg
function testAuditTriggerDepeg() public {
    (
        uint256 ID,
        address insr,
        address risk
    ) = testAuditUserDepositsUSDC();

    vm.warp(ID - 20 days);

    uint256 index = 1; //vaultFactory.marketIndex();

    vm.expectRevert("You are not calling from the Controller!");
    controller.triggerDepeg(index, ID);

    emit log("Transaction reverted");
}
```

The transaction is reverted, as there is no controller defined for the vaults (controller address is 0x0).

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended to validate if controller address was set in the VaultFactory during creation of the vaults - for example in createNewMarket function of VaultFactory contract.

3.2 (HAL-02) INTEGER UNDERFLOW - MISSING FEE VALIDATION - MEDIUM

Description:

Value of vault fee is not validated anywhere in Vault contract. When administrator sets vault fee over 100%, it will trigger an underflow error in beforeDeposit function, line 372 of Vault.sol. Calculated fee value (which in this case is greater than deposit) is subtracted from the number of shares deposited into the vault.

Code location:

Constructor:

Listing 5: Y2K-smartcontracts/src/Vault.sol (Line 101)

```
90  constructor(
91      address assetAddress,
92      string memory _name,
93      string memory _symbol,
94      address _treasury,
95      uint256 _fee,
96      address _token,
97      int256 _strikePrice,
98      address _controller
99  ) SemiFungibleVault(ERC20(assetAddress), _name, _symbol) {
100     tokenInsured = _token;
101     feeTaken = _fee;
102     treasury = _treasury;
103     strikePrice = _strikePrice;
104     Admin = msg.sender;
105     idCounter = 0;
106     controller = _controller;
107     timewindow = 1 days;
108 }
```

Setter:

Listing 6: Y2K-smartcontracts/src/Vault.sol (Line 293)

```
function changeFee(uint256 _fee) public onlyAdmin {
    feeTaken = _fee;
}
```

Fee calculation:

Listing 7: Y2K-smartcontracts/src/Vault.sol (Line 282)

```
function calculateFeeValue(uint256 amount)
    public
    view
    returns (uint256 feeValue)
{
    // 0.5% = multiply by 1000 then divide by 5
    return (amount * feeTaken) / 1000;
}
```

beforeDeposit function is using calculated fee, causing underflow error:

Listing 8: Y2K-smartcontracts/src/Vault.sol (Line 372)

```
function beforeDeposit(uint256 shares)
    internal
    returns (uint256 sharesMinusFee)
{
    //calculation of fee
    uint256 feeValue = calculateFeeValue(shares);
    uint256 valueMinusFee = shares - feeValue;
    //Payment of fee
    asset.safeTransferFrom(msg.sender, treasury, feeValue);

    return valueMinusFee;
}
```

Proof of Concept:

The vault is created with vault fee set to 120%, then when user deposits transaction is reverted with arithmetic error (underflow).

Listing 9: (Lines 2,32-33)

```
 1 function testAuditDepositsFeeOverflow() public {
 2     uint256 fee = 1200; // fee = 120%
 3     int256 strikePrice = 120000000; //1$ = 100000000
 4     uint256 epochBegin = block.timestamp + 3 days; // 3 days from
 ↳ now
 5     uint256 epochEnd = block.timestamp + 30 days; // 30 days from
 ↳ now
 6     (address insr, address risk) = CreationNewVaults(
 7         fee,
 8         USDC,
 9         strikePrice,
10         epochBegin,
11         epochEnd,
12         USDC_oracle
13     );
14
15     address user = address(3);
16     uint256 amount = 0.1 ether;
17     uint256 ID = epochEnd;
18
19     vm.startPrank(user);
20     emit log_named_address("Insurance Deposit User : ", user);
21     vm.deal(user, amount);
22
23     // swap to weth
24     IWETH(assetWETH).deposit{value: amount}();
25
26     uint256 balance = ERC20(assetWETH).balanceOf(user);
27     emit log_named_uint("User Balance before Deposit ", balance);
28
29     ERC20(assetWETH).approve(risk, amount);
30
31     // transaction should revert due to overflow
32     vm.expectRevert(stdError.arithmeticError);
33     Vault(risk).deposit(ID, amount, user);
34
35     uint256 vaultBalance = ERC20(assetWETH).balanceOf(risk);
```

```
    emit log_named_uint("Vault Balance ", vaultBalance);

    uint256 newbalance = ERC20(assetWETH).balanceOf(user);
    emit log_named_uint("User Balance after Deposit", newbalance);

    uint256 user_vaultbalance = Vault(risk).balanceOf(user, ID);
    emit log_named_uint("User Vault Balance ", user_vaultbalance);

    uint256 trzbalance = ERC20(assetWETH).balanceOf(treasury);
    emit log_named_uint("Treasury Balance ", trzbalance);

    // assert balances
    assertEq(user_vaultbalance, 0);
    assertEq(trzbalance, 0);
    assertEq(balance, newbalance);

    vm.stopPrank();
}
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

It is recommended to add boundary check for fee value. It should not be possible to set fees over 100%.

3.3 (HAL-03) RELOCKING OF ASSETS DOES NOT WORK PROPERLY - MEDIUM

Description:

The LockRewards contract allows the user to re-lock assets for another epoch. However, during the audit it was found that, when the user re-locks tokens for another epoch, he is able to withdraw re-locked tokens before the end of the new epoch. Also, it was found that when a user re-locks assets for another epoch, after the end of the new epoch he/she will not receive a reward.

Proof of Concept:

Below is a Brownie test script, for the following scenario:

1. User locks tokens for one epoch (10 days).
2. In the middle of the epoch (5 days), user re-locks tokens.
3. User waits for first epoch to end, claims a reward
4. User tries to withdraw assets re-locked assets

Listing 10: (Lines 6,16-17,35,38,54-55,60)

```
 1 def test_relock_one_epoch(newo, weth, deployer, owner, user1,
 ↳ user2):
 2
 3     lockRewards = prepare_lockRewards(newo, weth, deployer, owner,
 ↳     user1, user2)
 4
 5     # deposit assets for 1 epoch
 6     lockRewards.deposit(10, 1, {'from': user1})
 7
 8     # set two, ten days long epochs
 9     lockRewards.setNextEpoch(10, 10, 10, {'from': owner})
10     lockRewards.setNextEpoch(10, 10, 10, {'from': owner})
11
12     # asserts balances after deposits
13     assert(newo.balanceOf(user1) == 9990)
14     assert(weth.balanceOf(user1) == 0)
15     assert(lockRewards.balanceOf(user1) == 10)
```

```
    assert(lockRewards.balanceOfInEpoch(user1, 1) == 10)
    assert(lockRewards.balanceOfInEpoch(user1, 2) == 0)

    # assert current epoch == 1
    assert(lockRewards.currentEpoch() == 1)

    # owner tries to withdraw locked tokens - should revert
    with brownie.reverts():
        lockRewards.withdraw(10, {'from': user1})

    # wait half of epoch (5 days)
    chain.sleep(86400 * 5)
    chain.mine()

    # owner tries to withdraw locked tokens - should revert
    with brownie.reverts():
        lockRewards.withdraw(10, {'from': user1})

    # re-lock assets for next epoch before first epoch ends
    lockRewards.deposit(0, 1, {'from': user1})

    # assets should be locked for 2nd epoch
    assert(lockRewards.balanceOfInEpoch(user1, 2) == 10)

    # wait until end of epoch (5 days + 1 second)
    chain.sleep(5 * 86400 + 1)
    chain.mine()

    # owner creates another epoch, current epoch should be updated
    lockRewards.setNextEpoch(10, 10, 10, {'from': owner})

    # claim rewards for 1st epoch
    lockRewards.claimReward({'from': user1})

    # assert balances after epoch ends and user claimed reward
    assert(newo.balanceOf(user1) == 10000)
    assert(weth.balanceOf(user1) == 10)
    assert(lockRewards.balanceOf(user1) == 10)
    assert(lockRewards.balanceOfInEpoch(user1, 1) == 10)
    assert(lockRewards.balanceOfInEpoch(user1, 2) == 10)

    # verify that assets were locked for another epoch and
    # user can't withdraw re-locked tokens - should revert
    with brownie.reverts():
        lockRewards.withdraw(10, {'from': user1})

    # wait until end of epoch (10 days + 1 second)
    chain.sleep(86400 * 10 + 1)
    chain.mine()
    lockRewards.setNextEpoch(10, 10, 10, {'from': owner})

    lockRewards.claimReward({'from': user1})
    lockRewards.withdraw(10, {'from': user1})

    # assert balances after users withdrawn their assets
    assert(newo.balanceOf(user1) == 10020)
    assert(weth.balanceOf(user1) == 20)
    assert(lockRewards.balanceOf(user1) == 0)
    assert(lockRewards.balanceOfInEpoch(user1, 1) == 10)
    assert(lockRewards.balanceOfInEpoch(user1, 2) == 10)
```

The above test fails on the following step:

Listing 11: (Line 5)

```
1        # verify that assets were locked for another epoch and
2         # user can't withdraw re-locked tokens - should revert
3         with brownie.reverts():
4 >           lockRewards.withdraw(10, {'from': user1})
5 E           AssertionError: Transaction did not revert
```

User is able to withdraw re-locked tokens before end of epoch (balanceOfInEpoch returns that withdrawn tokens are locked).

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

After re-locking assets for another epoch, user should not be able to withdraw them before the end of the new epoch.

3.4 (HAL-04) CLAIMED AMOUNTS ARE MAPPED TO INDEXES IN MERKLE TREE - MEDIUM

Description:

MerkleDistributor contract is used to airdrop tokens to the users. To keep track of which users already claimed tokens, contract maps the index from the tree with already claimed amount.
NewOrder DAO implemented a possibility to update a Merkle Tree root. In such case, it is crucial to keep indexes the same way in the new tree as they were in the old one.
When Merkle Tree after update will have different index for the same account, user may not be able to withdraw greater amount assigned in a new tree or may be able to double claim the tokens.

Code location:

Mapping that stores already claimed amount:

Listing 12: merkle-distributor/contracts/MerkleDistributor.sol (Line 16)

```
contract MerkleDistributor is IMerkleDistributor, Ownable {
    address public immutable override token;
    bytes32 public override merkleRoot;
    using SafeERC20 for IERC20;

    /// inheritdoc IMerkleDistributor
    mapping(uint256 => uint256) public override claimed;
```

Validation, if user already claimed tokens:

Listing 13: merkle-distributor/contracts/MerkleDistributor.sol (Lines 30,35,37)

```
29     function claim(
30         uint256 index,
31         address account,
32         uint256 amount,
33         bytes32[] calldata merkleProof
34     ) external override {
35         uint256 alreadyClaimed = claimed[index];
36         require(
37             amount > alreadyClaimed,
38             "MerkleDistributor: airdrop limit reached"
39         );
```

Proof of Concept:

Example Hardhat test case for double-claim scenario:

Initial tree contains two elements: wallet0 and wallet1:

Listing 14

```
tree = new BalanceTree([
{ account: wallet0.address, amount: BigNumber.from(100) },
{ account: wallet1.address, amount: BigNumber.from(101) },
]);
```

First user (wallet0) claims 100 tokens from the tree. Then the tree is updated to contain three elements (in following order): wallet1, wallet2, wallet0.
User (wallet0) as his index has changed from 0 to 2 is now able to claim additional 100 tokens:

Listing 15: (Lines 6-8,16,21,27,31)

```
1 describe("Update merkle root - different order in new tree", () =>
  ↳  {
2   let newTree: BalanceTree;
3
```

```
  beforeEach(async () => {
    newTree = new BalanceTree([
      { account: wallet1.address, amount: BigNumber.from(100) },
      { account: wallet2.address, amount: BigNumber.from(150) },
      { account: wallet0.address, amount: BigNumber.from(100) },
    ]);
    await token.setBalance(distributor.address, 101 + 100);
  });

  it("claims again after updating merkle root: 0 - double claim", async ()=> {
    const proof = tree.getProof(0, wallet0.address, BigNumber.from(100));
    await expect(
      distributor.claim(0, wallet0.address, 100, proof, overrides)
    )
      .emit(distributor, "Claimed")
      .withArgs(0, wallet0.address, BigNumber.from(100));

    expect(await token.balanceOf(wallet0.address)).to.equal(BigNumber.from(100));

    await distributor.connect(owner).updateMerkleRoot(newTree.getHexRoot());

    const newProof = newTree.getProof(2, wallet0.address, BigNumber.from(100));
    await expect(
      distributor.claim(2, wallet0.address, 100, newProof, overrides)
    )
      .emit(distributor, "Claimed")
      .withArgs(2, wallet0.address, BigNumber.from(100));
    expect(await token.balanceOf(wallet0.address)).to.equal(BigNumber.from(200));
  });

});
```

Risk Level:

Likelihood - 2

Impact - 4

Recommendation:

Consider tracking which user claimed tokens with different value, like user address.

3.5 (HAL-05) CONTRACT IS NOT PAUSED DURING MERKLE ROOT UPDATE - MEDIUM

Description:

Contract is not paused during Merkle Root update, users can claim tokens during an update.

Code location:

Listing 16: merkle-distributor/contracts/MerkleDistributor.sol (Line 57)

```
function updateMerkleRoot(bytes32 newMerkleRoot) public override
↳ onlyOwner {
    emit UpdateMerkleRoot(msg.sender, merkleRoot, newMerkleRoot);
    merkleRoot = newMerkleRoot;
}
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Consider pausing contract for update of Merkle Tree root.

3.6 (HAL-06) ERC4626 DOES NOT WORK WITH FEE-ON-TRANSFER TOKENS - MEDIUM

Description:

The ERC4626.deposit/mint functions do not work well with fee-on-transfer tokens as the amount variable is the pre-fee amount, including the fee, whereas the totalAssets do not include the fee anymore.

Code location:

Listing 17: SemiFungibleVault.sol

```
    function deposit(
        uint256 id,
        uint256 assets,
        address receiver
    ) public virtual returns (uint256 shares) {
        // Check for rounding error since we round down in
↳ previewDeposit.
        require((shares = previewDeposit(id, assets)) != 0, "
↳ ZERO_SHARES");

        // Need to transfer before minting or ERC777s could
↳ reenter.
        asset.safeTransferFrom(msg.sender, address(this), assets);

        _mint(receiver, id, shares, EMPTY);

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

        afterDeposit(id, assets, shares);
    }

```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

amount should be the amount excluding the fee, i.e., the amount the contract actually received. This can be done by subtracting the pre-contract balance from the post-contract balance.

3.7 (HAL-07) ORACLE CAN BE OVERWRITTEN ON THE MARKET CREATION - MEDIUM

Description:

During the code, It has been noticed that the oracle can be overwritten If the oracle is already existing on the market. Even if, the admin is privileged on that function, the mistakenly defined oracle address can break all previous markets.

Scenario:

- Create insurance for USDC call the CreateNewMarket function.
- Than second time, Call CreateNewMarket With the USDC and wrong oracle address.
- USDC oracle address will be set wrong mistakenly. (You will overwrite USDC oracle address)

Code Location:

Listing 18: VaultFactory.sol

```
...
        indexVaults[marketIndex] = [address(insurance), address(
↳ risk)];

        insurance.createAssets(epochBegin, epochEnd);
        risk.createAssets(epochBegin, epochEnd);

        indexEpochs[marketIndex].push(epochEnd);
        tokenToOracle[_token] = _oracle;

        emit InsuranceMarketCreated(
            marketIndex,
            address(insurance),
            address(risk),
            _token,
            _strikePrice
        );
...
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Consider implementing a statement If oracle is already set in the market creation.

Listing 19

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

Remediation Plan:

SOLVED: The NewOrderDAO Team solved this issue by implementing the suggested check.

Commit ID: Commit

3.8 (HAL-08) SHOULD CHECK RETURN DATA FROM CHAINLINK AGGREGATORS - MEDIUM

Description:

The getLatestPrice function in the contract Controller.sol fetches the asset price from a Chainlink aggregator using the latestRoundData function. However, there are no checks on roundID nor timeStamp, resulting in stale prices. The oracle wrapper calls out to a Chainlink oracle receiving the latestRoundData(). It then checks freshness by verifying that the answer is indeed for the last known round. The returned updatedAt timestamp is not checked.

If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g. Chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the Chainlink system) consumers of this contract may continue using outdated stale data (if oracles are unable to submit no new round is started).

Code Location:

Listing 20: Controller.sol

```
(, int256 price , , , ) = priceFeed.latestRoundData();
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Consider adding checks on the return data with proper revert messages if the price is stale or the round is incomplete, for example:

Listing 21

```
(uint80 roundID, int256 price, , uint256 timeStamp, uint80
↳ answeredInRound) = ETH_CHAINLINK.latestRoundData();
require(price > 0, "Chainlink price <= 0");
require(answeredInRound >= roundID, "...");
require(timeStamp != 0, "...");
```

Remediation Plan:

SOLVED: The NewOrderDAO Team solved this issue by implementing the suggested check.

Commit ID: Commit

3.9 (HAL-09) FLOATING PRAGMA - LOW

Description:

Smart contract uses the floating pragma. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too new which has not been extensively tested.

Code location:

VaultFactory.sol, line: 2
Vault.sol, line: 2
IStakingRewards.sol, line: 1
Owned.sol, line: 1
RewardsDistributionRecipient.sol, line: 1
StakingRewards.sol, line: 1
LockRewards.sol, line: 2
ILockRewards.sol, line: 2

Risk Level:

Likelihood - 4
Impact - 1

Recommendation:

Consider lock the pragma version, known bugs for the compiler version. When possible, do not use floating pragma in the final live deployment. Apart from just locking the pragma version in the code, removing the caret (^).

3.10 (HAL-10) MISSING PAUSE/UNPAUSE FUNCTIONALITY ON THE LOCKREWARDS CONTRACT - LOW

Description:

In case a hack is occurring, or an exploit is discovered, the team should be able to pause functionality until the necessary changes are made to the system. To use a thorchain example again, the team behind thorchain noticed an attack was going to occur well before the system transferred funds to the hacker. However, they were not able to shut the system down fast enough. (According to the incidence report here: https://github.com/HalbornSecurity/PublicReports/blob/master/Incident%20Reports/Tho
Incident_Analysis_July_23_2021.pdf)

Code location:

LockRewards.sol, line: 2
ILockRewards.sol, line: 2

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Pause functionality on the contract would have helped secure the funds quickly.

3.11 (HAL-11) MISSING VALIDATION - LOW

Description:

During the code review, it was observed that the setter functions did not properly validate that new values for parameters are valid. For instance, in the LockRewards contract admin might be able to set epoch duration in days to a large value, which may cause user assets to get locked for an extensive period of time, or may lead to overflows. Rewards in LockRewards token are paid in two different tokens, but there is no validation the tokens are different. Also, smart contracts do not validate that new addresses for controller, treasury etc are not zero address (0x0).

Code Location:

Missing Zero address checks:

1. Y2K-smartcontracts/src/VaultFactory.sol, #152-154
2. Y2K-smartcontracts/src/VaultFactory.sol, #50
3. Y2K-smartcontracts/src/VaultFactory.sol, #176
4. Y2K-smartcontracts/src/Vault.sol, #297
5. RewardsVault/contracts/LockRewards.sol, #64,65

Both Reward tokens can be set to the same address:

Listing 22: RewardsVault/contracts/LockRewards.sol (Lines 64,65)

```
57 constructor(
58     address _lockToken,
59     address _rewardAddr1,
60     address _rewardAddr2,
61     uint256 _maxEpochs
62 ) {
63     lockToken = _lockToken;
64     rewardToken[0].addr  = _rewardAddr1;
65     rewardToken[1].addr  = _rewardAddr2;
66     maxEpochs = _maxEpochs;
67 }
```

Epoch duration validation:

The epochDurationInDays parameter is not validated, epoch may be set to very long time. On line 415, number of days is multiplied by number of seconds in a day, which may cause an overflow.

Listing 23: RewardsVault/contracts/LockRewards.sol (Lines 388,415)

```
385     function _setEpoch(
386         uint256 reward1,
387         uint256 reward2,
388         uint256 epochDurationInDays,
389         uint256 epochStart
390     ) internal {
391         if (nextUnsetEpoch - currentEpoch > 1)
392             revert EpochMaxReached(2);
393         if (epochStart < block.timestamp)
394             revert EpochStartInvalid(epochStart, block.timestamp);
395
396         uint256[2] memory rewards = [reward1, reward2];
397
398         for (uint256 i = 0; i < 2; i++) {
399             uint256 unclaimed = rewardToken[i].rewards -
  ↳ rewardToken[i].rewardsPaid;
400             uint256 balance = IERC20(rewardToken[i].addr).
  ↳ balanceOf(address(this));
401
402             if (balance - unclaimed < rewards[i])
403                 revert InsufficientFundsForRewards(rewardToken[i].
  ↳ addr, balance - unclaimed, rewards[i]);
404
405             rewardToken[i].rewards += rewards[i];
406         }
407
408         uint256 next = nextUnsetEpoch;
409
410         if (currentEpoch == next || epochStart > epochs[next - 1].
  ↳ finish + 1) {
411             epochs[next].start = epochStart;
412         } else {
413             epochs[next].start = epochs[next - 1].finish + 1;
414         }
415         epochs[next].finish = epochs[next].start + (
  ↳ epochDurationInDays * 86400); // Seconds in a day
```

```
        epochs[next].rewards1 = reward1;
        epochs[next].rewards2 = reward2;
        epochs[next].isSet = true;

        nextUnsetEpoch += 1;
        emit SetNextReward(next, reward1, reward2, epochs[next].
 ↳ start, epochs[next].finish);
    }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Halborn recommends that validation is added to the setter functions, throughout all the smart contracts. At a minimum, NewOrderDAO should ensure that these values cannot be set to zero.

3.12 (HAL-12) IMPROPER ROLE BASED ACCESS CONTROL - LOW

Description:

The smart contracts, in scope, do not implement granular access control. All the privileged functionality was assigned to one account, the owner of the smart contract. This could lead to serious consequences should the ownership of this account be lost, or a malicious admin decided to take over the platform.

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Halborn recommends that a more granular access control policy is enforced. For instance, the following user roles could be set:

- Pauser - user who can pause the contracts
- Owner - admin of the contract which can access the most sensitive functionality
- ChangeWhitelist role - user who can add or remove tokens from the whitelist

3.13 (HAL-13) OWNER CAN WITHDRAW ALL TOKENS - LOW

Description:

In LockRewards contract, the owner can withdraw all funds via changeRecoverWhitelist and recoverERC20 functions.

Code location:

Listing 24: RewardsVault/contracts/LockRewards.sol (Line 318)

```
312 function recoverERC20(address tokenAddress, uint256 tokenAmount)
    ↳ external onlyOwner {
313     if (whitelistRecoverERC20[tokenAddress] == false) revert
    ↳ NotWhitelisted();
314
315     uint balance = IERC20(tokenAddress).balanceOf(address(this));
316     if (balance < tokenAmount) revert InsufficientBalance();
317
318     IERC20(tokenAddress).safeTransfer(owner(), tokenAmount);
319     emit RecoveredERC20(tokenAddress, tokenAmount);
320 }
```

Listing 25: RewardsVault/contracts/LockRewards.sol (Line 334)

```
333 function changeRecoverWhitelist(address tokenAddress, bool flag)
    ↳ external onlyOwner {
334     whitelistRecoverERC20[tokenAddress] = flag;
335     emit ChangeERC20Whiltelist(tokenAddress, flag);
336 }
```

Recommendation:

Consider disabling possibility for the owner to withdraw all tokens deposited by the users.

3.14 (HAL-14) MISSING RE-ENTRANCY PROTECTION - LOW

Description:

To protect against cross-function re-entrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called “nonReentrant” that guards the function with a mutex against the Reentrancy attacks.

Code location:

Listing 26: Y2K-smartcontracts/src/SemiFungibleVault.sol (Line 59)

```
function deposit(
    uint256 id,
    uint256 assets,
    address receiver
) public virtual returns (uint256 shares) {
    // Check for rounding error since we round down in
↳ previewDeposit.
    require((shares = previewDeposit(id, assets)) != 0, "
↳ ZERO_SHARES");

    // Need to transfer before minting or ERC777s could reenter.
    asset.safeTransferFrom(msg.sender, address(this), assets);

    _mint(receiver, id, shares, EMPTY);

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

    afterDeposit(id, assets, shares);
}
```

Recommendation:

Functions of SemiFungibleVault contract are missing nonReentrant guard. Though, these methods are implemented following checks-effects-interactions pattern only. But in longer term it is better to use “nonReentrant” guard to avoid unfortunate event in future due to code changes.

3.15 (HAL-15) LONG ERROR MESSAGES - INFORMATIONAL

Description:

Some of the error messages in require statements of MerkleDistributor contract exceed 32 bytes.
Error messages longer than 32-bytes consume additional gas.

Code location:

Listing 27: merkle-distributor/contracts/MerkleDistributor.sol (Line 38)

```
    function claim(
        uint256 index,
        address account,
        uint256 amount,
        bytes32[] calldata merkleProof
    ) external override {
        uint256 alreadyClaimed = claimed[index];
        require(
            amount > alreadyClaimed,
            "MerkleDistributor: airdrop limit reached"
        );
```

Recommendation:

Consider shorter error messages to save gas.

3.16 (HAL-16) OPTIMIZE UNSIGNED INTEGER COMPARISON - INFORMATIONAL

Description:

The check != 0 costs less gas compared to > 0 for unsigned integers in require statements with the optimizer enabled.
While it may seem that > 0 is cheaper than !=0, this is only true without the optimizer enabled and outside a require statement. If the optimizer is enabled at 10k, and It is in a require statement, that would be more gas efficient.

Code location:

Listing 28: StakingRewards.sol

```
    function stake(uint256 amount)
        external
        nonReentrant
        whenNotPaused
        updateReward(msg.sender)
    {
        require(amount > 0, "Cannot stake 0");
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        stakingToken.safeTransferFrom(
            msg.sender,
            address(this),
            id,
            amount,
            ""
        );
        emit Staked(msg.sender, id, amount);
    }
```

Recommendation:

Change > 0 comparison with != 0.

3.17 (HAL-17) PREFIX INCREMENTS ARE CHEAPER THAN POSTFIX INCREMENTS - INFORMATIONAL

Description:

Prefix increments are cheaper than postfix increments. Furthermore, using unchecked {++x} is even more gas efficient, and the gas saving accumulates every iteration and can make a real change. There is no risk of overflow caused by incrementing the iteration index in for loops (the ++i in for (uint256 i = 0; i < numIterations; ++i)). But increments perform overflow checks that are not necessary in this case.

Code location:

Listing 29: LockRewards.sol

```
    for (uint256 i = 0; i < lockEpochs; i++) {
            epochs[i + next].totalLocked += newBalance - epochs[i
 ↳ + next].balanceLocked[msg.sender];
            epochs[i + next].balanceLocked[msg.sender] =
 ↳ newBalance;
    }
```

Recommendation:

Consider using prefix increments.

3.18 (HAL-18) CHECK AMOUNT IS GREATER THAN 0 TO AVOID UNNECESSARILY CALLING SAFETRANSFER() - INFORMATIONAL

Description:

A check should be added to make sure amount exceeds 0 to avoid unnecessarily calling safeTransfer().

Code location:

Listing 30: merkle-distributor/contracts/MerkleDistributor.sol (Line 38)

```
29      function claim(
30          uint256 index,
31          address account,
32          uint256 amount,
33          bytes32[] calldata merkleProof
34      ) external override {
35          uint256 alreadyClaimed = claimed[index];
36          require(
37              amount > alreadyClaimed,
38              "MerkleDistributor: airdrop limit reached"
39          );
```

Recommendation:

Implement **amount > 0** check in the related sections.

3.19 (HAL-19) ROUNDING PROBLEMS IN THE EIP 4626 - INFORMATIONAL

Description:

Per EIP 4626's Security Considerations (https://eips.ethereum.org/EIPS/eip-4626), several important points are marked out. During the implementation, **NewOrderDAO** should consider the following items :

Listing 31

```
Finally, ERC-4626 Vault implementers should be aware of the need
↳ for specific, opposing rounding directions across the different
↳ mutable and view methods, as it is considered most secure to favor
↳  the Vault itself during calculations over its users:

- If (1) its calculating how many shares to issue to a user for a
↳ certain amount of the underlying tokens they provide or (2) its
↳ determining the amount of the underlying tokens to transfer to
↳ them for returning a certain amount of shares, it should round
↳ down.
- If (1) its calculating the amount of shares a user has to supply
↳  to receive a given amount of the underlying tokens or (2) its
↳ calculating the amount of underlying tokens a user has to provide
↳ to receive a certain amount of shares, it should round up.
```

Recommendation:

NewOrderDAO Team should be aware of the rounding issues in the **SemiFungibleVault**.

3.20 (HAL-20) EVENTS ARE NOT INDEXED - INFORMATIONAL

Description:

The emitted events are not indexed, making off-chain scripts such as front-ends of DApps difficult to filter the events efficiently.

Recommendation:

Add the indexed keyword in each event.

3.21 (HAL-21) MISSING EVENTS ON CHANGES - INFORMATIONAL

Description:

Function performing important changes to contract state should emit events to facilitate monitoring of the protocol operation.

Code Location:

Reference

Recommendation:

Consider omitting events on the related functions.

3.22 (HAL-22) UINT CAN'T BE LOWER THAN ZERO - INFORMATIONAL

Description:

Uint value can't be lower than 0, therefore the condition: `amount <= 0` can be changed to `amount == 0`.

Code Location:

Reference

Recommendation:

Consider changing the condition to `amount == 0 || accounts[msg.sender].balance < amount`.

3.23 (HAL-23) NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES - INFORMATIONAL

Description:

uint256 variable is initialized to a default value of zero per Solidity docs. Setting a variable to the default value is unnecessary.

Code Location:

Listing 32: StakingRewards.sol

```
uint256 public periodFinish = 0;
uint256 public rewardRate = 0;
```

Recommendation:

Remove explicit initialization for default values.

3.24 (HAL-24) IMMUTABLE VARIABLES - INFORMATIONAL

Description:

There are variables that do not change, so they can be marked as immutable to greatly improve the gas costs.

Code Location:

Listing 33: StakingRewards.sol

```
ERC20 public rewardsToken;
IERC1155 public stakingToken;
```

Recommendation:

Consider marking state variables as an immutable that never changes on the contract.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

To reduce the report size, Informational and Optimization findings reported by Slither were omitted.

MerkleDistributor:

Listing 34

```
MerkleProof.verify(bytes32[],bytes32,bytes32) (contracts/MerkleProof.sol#13-30) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

MerkleDistributor.constructor(address,bytes32,address).token_ (contracts/MerkleDistributor.sol#19) lacks a zero-check on :
        - token = token_ (contracts/MerkleDistributor.sol#23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in MerkleDistributor.claim(uint256,address,uint256,bytes32[]) (contracts/MerkleDistributor.sol#29-54):
    External calls:
    - IERC20(token).safeTransfer(account,airdropAmount) (contracts/MerkleDistributor.sol#51)
    Event emitted after the call(s):
    - Claimed(index,account,airdropAmount) (contracts/
↳ MerkleDistributor.sol#53)
Reentrancy in MerkleDistributor.withdrawToken(address,uint256) (
↳ contracts/MerkleDistributor.sol#63-76):
    External calls:
    - IERC20(token).safeTransfer(to,amount) (contracts/
↳ MerkleDistributor.sol#74)
    Event emitted after the call(s):
    - WithdrawToken(msg.sender,to,amount) (contracts/
↳ MerkleDistributor.sol#75)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-3

Address.verifyCallResult(bool,bytes,string) (../../../.brownie/
↳ packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils
↳ /Address.sol#201-221) uses assembly
    - INLINE ASM (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#213-216)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#assembly-usage

Different versions of Solidity are used:
    - Version used: ['=0.8.4', '>=0.5.0', '^0.8.0', '^0.8.1']
    - ^0.8.0 (../../../.brownie/packages/OpenZeppelin/openzeppelin
↳ -contracts@4.6.0/contracts/access/Ownable.sol#4)
    - ^0.8.0 (../../../.brownie/packages/OpenZeppelin/openzeppelin
↳ -contracts@4.6.0/contracts/token/ERC20/IERC20.sol#4)
    - ^0.8.0 (../../../.brownie/packages/OpenZeppelin/openzeppelin
↳ -contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20.sol#4)
    - ^0.8.1 (../../../.brownie/packages/OpenZeppelin/openzeppelin
↳ -contracts@4.6.0/contracts/utils/Address.sol#4)
    - ^0.8.0 (../../../.brownie/packages/OpenZeppelin/openzeppelin
↳ -contracts@4.6.0/contracts/utils/Context.sol#4)
    - =0.8.4 (contracts/MerkleDistributor.sol#2)
    - =0.8.4 (contracts/MerkleProof.sol#1)
    - >=0.5.0 (contracts/interfaces/IMerkleDistributor.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (../../../.brownie/packages/
↳ OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.
↳ sol#85-87) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (../../../.
↳ brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/
↳ contracts/utils/Address.sol#114-120) is never used and should be
```

```
removed
Address.functionDelegateCall(address,bytes) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#184-193) is never used and should be removed
Address.functionStaticCall(address,bytes) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#157-166) is never used and should be removed
Address.sendValue(address,uint256) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#60-65) is never used and should be removed
Context._msgData() (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Context.sol#21-23) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20.sol#45-58) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20.sol#69-80) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20.sol#60-67) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20.sol#29-36) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/token/ERC20/IERC20.sol#4)
↳ allows old versions
Pragma version^0.8.0 (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20
↳ .sol#4) allows old versions
Pragma version^0.8.1 (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#4) allows
↳  old versions
Pragma version^0.8.0 (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/utils/Context.sol#4) allows
↳  old versions
Pragma version>=0.5.0 (contracts/interfaces/IMerkleDistributor.sol
↳ #2) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (../../../.
↳ brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/
↳ contracts/utils/Address.sol#60-65):
    - (success) = recipient.call{value: amount}() (../../../.
↳ brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/
↳ contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,
↳ uint256,string) (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#128-139):
    - (success,returndata) = target.call{value: value}(data)
↳ (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4
↳ .6.0/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string)
↳  (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4
↳ .6.0/contracts/utils/Address.sol#157-166):
    - (success,returndata) = target.staticcall(data) (../../../.
↳ brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/
↳ contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,
↳ string) (../../../.brownie/packages/OpenZeppelin/openzeppelin-
↳ contracts@4.6.0/contracts/utils/Address.sol#184-193):
    - (success,returndata) = target.delegatecall(data) (../../../.
↳ brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/
↳ contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#low-level-calls

renounceOwnership() should be declared external:
    - Ownable.renounceOwnership() (../../../.brownie/packages/
↳ OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/access/Ownable
↳ .sol#54-56)
updateMerkleRoot(bytes32) should be declared external:
    - MerkleDistributor.updateMerkleRoot(bytes32) (contracts/
↳ MerkleDistributor.sol#57-60)
withdrawAllTokens(address) should be declared external:
    - MerkleDistributor.withdrawAllTokens(address) (contracts/
↳ MerkleDistributor.sol#79-81)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#public-function-that-could-be-declared-external
contracts/ analyzed (9 contracts with 78 detectors), 31 result(s)
↳ found
```

RewardsVault:

Listing 35

```
Reentrancy in LockRewards.exit() (contracts/LockRewards.sol#251-254):
    External calls:
    - _withdraw(accounts[msg.sender].balance) (contracts/LockRewards.sol#252)
        - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20.sol#93)
        - (success,returndata) = target.call{value: value}(data) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#137)
        - IERC20(lockToken).safeTransfer(msg.sender,amount) (contracts/LockRewards.sol#435)
    - _claim() (contracts/LockRewards.sol#253)
        - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20.sol#93)
        - IERC20(rewardToken[0].addr).safeTransfer(msg.sender,reward1) (contracts/LockRewards.sol#454)
        - (success,returndata) = target.call{value: value}(data) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#137)
        - IERC20(rewardToken[1].addr).safeTransfer(msg.sender,reward2) (contracts/LockRewards.sol#459)
    External calls sending eth:
    - _withdraw(accounts[msg.sender].balance) (contracts/LockRewards.sol#252)
        - (success,returndata) = target.call{value: value}(data) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#137)
    - _claim() (contracts/LockRewards.sol#253)
        - (success,returndata) = target.call{value: value}(data) (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/utils/Address.sol#137)
    State variables written after the call(s):
    - _claim() (contracts/LockRewards.sol#253)
        - accounts[msg.sender].rewards1 = 0 (contracts/LockRewards.sol#453)
        - accounts[msg.sender].rewards2 = 0 (contracts/LockRewards
↳ .sol#458)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities

Reentrancy in LockRewards._claim() (contracts/LockRewards.sol
↳ #448-463):
    External calls:
    - IERC20(rewardToken[0].addr).safeTransfer(msg.sender,reward1)
↳  (contracts/LockRewards.sol#454)
    State variables written after the call(s):
    - accounts[msg.sender].rewards2 = 0 (contracts/LockRewards.sol
↳ #458)
Reentrancy in LockRewards._withdraw(uint256) (contracts/
↳ LockRewards.sol#431-439):
    External calls:
    - IERC20(lockToken).safeTransfer(msg.sender,amount) (contracts
↳ /LockRewards.sol#435)
    State variables written after the call(s):
    - accounts[msg.sender].balance -= amount (contracts/
↳ LockRewards.sol#437)
Reentrancy in LockRewards.deposit(uint256,uint256) (contracts/
↳ LockRewards.sol#187-228):
    External calls:
    - lToken.safeTransferFrom(msg.sender,address(this),amount) (
↳ contracts/LockRewards.sol#212)
    State variables written after the call(s):
    - accounts[msg.sender].balance += amount (contracts/
↳ LockRewards.sol#214)
    - epochs[i + next].totalLocked += newBalance - epochs[i + next
↳ ].balanceLocked[msg.sender] (contracts/LockRewards.sol#225)
    - epochs[i + next].balanceLocked[msg.sender] = newBalance (
↳ contracts/LockRewards.sol#226)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-1

LockRewards.balanceOf(address).owner (contracts/LockRewards.sol
↳ #77) shadows:
    - Ownable.owner() (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/access/Ownable.sol#35-37) (
↳ function)
LockRewards.balanceOfInEpoch(address,uint256).owner (contracts/
↳ LockRewards.sol#87) shadows:
    - Ownable.owner() (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/access/Ownable.sol#35-37) (
↳ function)
LockRewards.getAccount(address).owner (contracts/LockRewards.sol
↳ #154) shadows:
    - Ownable.owner() (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/access/Ownable.sol#35-37) (
↳ function)
LockRewards._getAccount(address).owner (contracts/LockRewards.sol
↳ #470) shadows:
    - Ownable.owner() (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/access/Ownable.sol#35-37) (
↳ function)
LockRewards.updateReward(address).owner (contracts/LockRewards.sol
↳ #520) shadows:
    - Ownable.owner() (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/access/Ownable.sol#35-37) (
↳ function)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#local-variable-shadowing

LockRewards.constructor(address,address,address,uint256).
↳ _lockToken (contracts/LockRewards.sol#58) lacks a zero-check on :
        - lockToken = _lockToken (contracts/LockRewards.sol#63)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#missing-zero-address-validation

Reentrancy in LockRewards._withdraw(uint256) (contracts/
↳ LockRewards.sol#431-439):
    External calls:
    - IERC20(lockToken).safeTransfer(msg.sender,amount) (contracts
↳ /LockRewards.sol#435)
    State variables written after the call(s):
    - totalAssets -= amount (contracts/LockRewards.sol#436)
Reentrancy in LockRewards.deposit(uint256,uint256) (contracts/
↳ LockRewards.sol#187-228):
    External calls:
    - lToken.safeTransferFrom(msg.sender,address(this),amount) (
↳ contracts/LockRewards.sol#212)
    State variables written after the call(s):
    - totalAssets += amount (contracts/LockRewards.sol#213)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-2

Reentrancy in LockRewards._claim() (contracts/LockRewards.sol
↳ #448-463):
    External calls:
    - IERC20(rewardToken[0].addr).safeTransfer(msg.sender,reward1)
↳  (contracts/LockRewards.sol#454)
    Event emitted after the call(s):
    - RewardPaid(msg.sender,rewardToken[0].addr,reward1) (
↳ contracts/LockRewards.sol#455)
Reentrancy in LockRewards._claim() (contracts/LockRewards.sol
↳ #448-463):
    External calls:
    - IERC20(rewardToken[0].addr).safeTransfer(msg.sender,reward1)
↳  (contracts/LockRewards.sol#454)
    - IERC20(rewardToken[1].addr).safeTransfer(msg.sender,reward2)
↳  (contracts/LockRewards.sol#459)
    Event emitted after the call(s):
    - RewardPaid(msg.sender,rewardToken[1].addr,reward2) (
↳ contracts/LockRewards.sol#460)
Reentrancy in LockRewards._withdraw(uint256) (contracts/
↳ LockRewards.sol#431-439):
    External calls:
    - IERC20(lockToken).safeTransfer(msg.sender,amount) (contracts
↳ /LockRewards.sol#435)
    Event emitted after the call(s):
    - Withdrawn(msg.sender,amount) (contracts/LockRewards.sol#438)
Reentrancy in LockRewards.deposit(uint256,uint256) (contracts/
↳ LockRewards.sol#187-228):
    External calls:
    - lToken.safeTransferFrom(msg.sender,address(this),amount) (
↳ contracts/LockRewards.sol#212)
    Event emitted after the call(s):
    - Deposit(msg.sender,amount,accounts[msg.sender].lockEpochs) (
↳ contracts/LockRewards.sol#216)
Reentrancy in LockRewards.exit() (contracts/LockRewards.sol
↳ #251-254):
    External calls:
    - _withdraw(accounts[msg.sender].balance) (contracts/
↳ LockRewards.sol#252)
        - returndata = address(token).functionCall(data,SafeERC20:
↳  low-level call failed) (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20
↳ .sol#93)
        - (success,returndata) = target.call{value: value}(data)
↳ (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4
↳ .6.0/contracts/utils/Address.sol#137)
        - IERC20(lockToken).safeTransfer(msg.sender,amount) (
↳ contracts/LockRewards.sol#435)
    - _claim() (contracts/LockRewards.sol#253)
        - returndata = address(token).functionCall(data,SafeERC20:
↳ low-level call failed) (../../../.brownie/packages/OpenZeppelin/
↳ openzeppelin-contracts@4.6.0/contracts/token/ERC20/utils/SafeERC20
↳ .sol#93)
        - IERC20(rewardToken[0].addr).safeTransfer(msg.sender,
↳ reward1) (contracts/LockRewards.sol#454)
        - (success,returndata) = target.call{value: value}(data)
↳ (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4
↳ .6.0/contracts/utils/Address.sol#137)
        - IERC20(rewardToken[1].addr).safeTransfer(msg.sender,
↳ reward2) (contracts/LockRewards.sol#459)
    External calls sending eth:
    - _withdraw(accounts[msg.sender].balance) (contracts/
↳ LockRewards.sol#252)
        - (success,returndata) = target.call{value: value}(data)
↳ (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4
↳ .6.0/contracts/utils/Address.sol#137)
    - _claim() (contracts/LockRewards.sol#253)
        - (success,returndata) = target.call{value: value}(data)
↳ (../../../.brownie/packages/OpenZeppelin/openzeppelin-contracts@4
↳ .6.0/contracts/utils/Address.sol#137)
    Event emitted after the call(s):
    - RewardPaid(msg.sender,rewardToken[0].addr,reward1) (
↳ contracts/LockRewards.sol#455)
        - _claim() (contracts/LockRewards.sol#253)
    - RewardPaid(msg.sender,rewardToken[1].addr,reward2) (
↳ contracts/LockRewards.sol#460)
        - _claim() (contracts/LockRewards.sol#253)
Reentrancy in LockRewards.recoverERC20(address,uint256) (contracts
↳ /LockRewards.sol#312-320):
    External calls:
    - IERC20(tokenAddress).safeTransfer(owner(),tokenAmount) (
↳ contracts/LockRewards.sol#318)
    Event emitted after the call(s):
    - RecoveredERC20(tokenAddress,tokenAmount) (contracts/
↳ LockRewards.sol#319)
Reentrancy in LockRewards.recoverERC721(address,uint256) (
↳ contracts/LockRewards.sol#341-344):
    External calls:
    - IERC721(tokenAddress).transferFrom(address(this),owner(),
↳ tokenId) (contracts/LockRewards.sol#342)
    Event emitted after the call(s):
```

```
    - RecoveredERC721(tokenAddress,tokenId) (contracts/LockRewards
 ↳ .sol#343)
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#reentrancy-vulnerabilities-3

LockRewards._setEpoch(uint256,uint256,uint256,uint256) (contracts/
 ↳ LockRewards.sol#385-423) uses timestamp for comparisons
    Dangerous comparisons:
    - epochStart < block.timestamp (contracts/LockRewards.sol#393)
    - currentEpoch == next || epochStart > epochs[next - 1].finish
 ↳  + 1 (contracts/LockRewards.sol#410)
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#block-timestamp
contracts/ analyzed (10 contracts with 57 detectors), 20 result(s)
 ↳  found
```

Staking Rewards:

Listing 36

```
StakingRewards.notifyRewardAmount(uint256) (src/rewards/
↳ StakingRewards.sol#154-181) performs a multiplication on the
↳ result of a division:
    -rewardRate = reward.div(rewardsDuration) (src/rewards/
↳ StakingRewards.sol#161)
    -leftover = remaining.mul(rewardRate) (src/rewards/
↳ StakingRewards.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#divide-before-multiply

Reentrancy in StakingRewards.exit() (src/rewards/StakingRewards.
↳ sol#147-150):
    External calls:
    - withdraw(_balances[msg.sender]) (src/rewards/StakingRewards.
↳ sol#148)
        - stakingToken.safeTransferFrom(address(this),msg.sender,
↳ id,amount,) (src/rewards/StakingRewards.sol#128-134)
    State variables written after the call(s):
    - getReward() (src/rewards/StakingRewards.sol#149)
        - _status = _ENTERED (lib/openzeppelin-contracts/contracts
↳ /security/ReentrancyGuard.sol#55)
        - _status = _NOT_ENTERED (lib/openzeppelin-contracts/
↳ contracts/security/ReentrancyGuard.sol#61)
    - getReward() (src/rewards/StakingRewards.sol#149)
        - lastUpdateTime = lastTimeRewardApplicable() (src/rewards
↳ /StakingRewards.sol#209)
    - getReward() (src/rewards/StakingRewards.sol#149)
        - rewardPerTokenStored = rewardPerToken() (src/rewards/
↳ StakingRewards.sol#208)
    - getReward() (src/rewards/StakingRewards.sol#149)
        - rewards[msg.sender] = 0 (src/rewards/StakingRewards.sol
↳ #141)
        - rewards[account] = earned(account) (src/rewards/
↳ StakingRewards.sol#211)
    - getReward() (src/rewards/StakingRewards.sol#149)
        - userRewardPerTokenPaid[account] = rewardPerTokenStored (
↳ src/rewards/StakingRewards.sol#212)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-1

RewardsDistributionRecipient.setRewardsDistribution(address) (src/
rewards/RewardsDistributionRecipient.sol#20-25) should emit an event for:
    - rewardsDistribution = _rewardsDistribution (src/rewards/RewardsDistributionRecipient.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

RewardsDistributionRecipient.setRewardsDistribution(address)._rewardsDistribution (src/rewards/RewardsDistributionRecipient.sol#20) lacks a zero-check on :
        - rewardsDistribution = _rewardsDistribution (src/rewards/RewardsDistributionRecipient.sol#24)
Owned.nominateNewOwner(address)._owner (src/rewards/Owned.sol#14) lacks a zero-check on :
        - nominatedOwner = _owner (src/rewards/Owned.sol#15)
StakingRewards.constructor(address,address,address,address,uint256)._rewardsDistribution (src/rewards/StakingRewards.sol#48) lacks a zero-check on :
        - rewardsDistribution = _rewardsDistribution (src/rewards/StakingRewards.sol#55)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in StakingRewards.exit() (src/rewards/StakingRewards.sol#147-150):
    External calls:
    - withdraw(_balances[msg.sender]) (src/rewards/StakingRewards.sol#148)
        - stakingToken.safeTransferFrom(address(this),msg.sender,id,amount,) (src/rewards/StakingRewards.sol#128-134)
    Event emitted after the call(s):
    - RewardPaid(msg.sender,reward) (src/rewards/StakingRewards.sol#143)
        - getReward() (src/rewards/StakingRewards.sol#149)
Reentrancy in StakingRewards.stake(uint256) (src/rewards/StakingRewards.sol#101-118):
    External calls:
    - stakingToken.safeTransferFrom(msg.sender,address(this),id,amount,) (src/rewards/StakingRewards.sol#110-116)
    Event emitted after the call(s):
    - Staked(msg.sender,id,amount) (src/rewards/StakingRewards.sol#117)
Reentrancy in StakingRewards.withdraw(uint256) (src/rewards/StakingRewards.sol#120-136):
    External calls:
    - stakingToken.safeTransferFrom(address(this),msg.sender,id,
↳ amount,) (src/rewards/StakingRewards.sol#128-134)
    Event emitted after the call(s):
    - Withdrawn(msg.sender,id,amount) (src/rewards/StakingRewards.
↳ sol#135)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-3

ERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32
↳ ) (lib/solmate/src/tokens/ERC20.sol#116-160) uses timestamp for
↳ comparisons
    Dangerous comparisons:
    - require(bool,string)(deadline >= block.timestamp,
↳ PERMIT_DEADLINE_EXPIRED) (lib/solmate/src/tokens/ERC20.sol#125)
StakingRewards.lastTimeRewardApplicable() (src/rewards/
↳ StakingRewards.sol#69-71) uses timestamp for comparisons
    Dangerous comparisons:
    - block.timestamp < periodFinish (src/rewards/StakingRewards.
↳ sol#70)
StakingRewards.getReward() (src/rewards/StakingRewards.sol
↳ #138-145) uses timestamp for comparisons
    Dangerous comparisons:
    - reward > 0 (src/rewards/StakingRewards.sol#140)
StakingRewards.notifyRewardAmount(uint256) (src/rewards/
↳ StakingRewards.sol#154-181) uses timestamp for comparisons
    Dangerous comparisons:
    - block.timestamp >= periodFinish (src/rewards/StakingRewards.
↳ sol#160)
    - require(bool,string)(rewardRate <= balance.div(
↳ rewardsDuration),Provided reward too high) (src/rewards/
↳ StakingRewards.sol#173-176)
StakingRewards.setRewardsDuration(uint256) (src/rewards/
↳ StakingRewards.sol#196-203) uses timestamp for comparisons
    Dangerous comparisons:
    - require(bool,string)(block.timestamp > periodFinish,Previous
↳  rewards period must be complete before changing the duration for
↳ the new period) (src/rewards/StakingRewards.sol#197-200)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#block-timestamp

Owned.nominateNewOwner(address)._owner (src/rewards/Owned.sol#14)
↳ lacks a zero-check on :
        - nominatedOwner = _owner (src/rewards/Owned.sol#15)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

RewardsDistributionRecipient.setRewardsDistribution(address) (src/rewards/RewardsDistributionRecipient.sol#20-25) should emit an event for:
    - rewardsDistribution = _rewardsDistribution (src/rewards/RewardsDistributionRecipient.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

Owned.nominateNewOwner(address)._owner (src/rewards/Owned.sol#14) lacks a zero-check on :
        - nominatedOwner = _owner (src/rewards/Owned.sol#15)
RewardsDistributionRecipient.setRewardsDistribution(address)._rewardsDistribution (src/rewards/RewardsDistributionRecipient.sol#20) lacks a zero-check on :
        - rewardsDistribution = _rewardsDistribution (src/rewards/RewardsDistributionRecipient.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
src/rewards analyzed (20 contracts with 57 detectors), 18 result(s) found
```

Y2K Core:

Listing 37

```
FixedPointMathLib.rpow(uint256,uint256,uint256) (lib/solmate/src/utils/FixedPointMathLib.sol#74-160) performs a multiplication on the result of a division:
    -x = xxRound_rpow_asm_0 / scalar (lib/solmate/src/utils/FixedPointMathLib.sol#131)
    -zx_rpow_asm_0 = z * x (lib/solmate/src/utils/FixedPointMathLib.sol#136)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).response (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476) is a local variable never initialized
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).reason (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#503) is a local variable never initialized
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).response (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#498) is a local variable never initialized
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).reason (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#480) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#467-486) ignores return value by IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476-484)
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#488-509) ignores return value by IERC1155Receiver(to).onERC1155BatchReceived(operator,from,ids,amounts,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#497-507)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#unused-return

Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,
↳ address,uint256,uint256,bytes).response (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#476)' in ERC1155.
↳ _doSafeTransferAcceptanceCheck(address,address,address,uint256,
↳ uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155
↳ /ERC1155.sol#467-486) potentially used before declaration:
↳ response != IERC1155Receiver.onERC1155Received.selector (lib/
↳ openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#477)
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,
↳ address,uint256,uint256,bytes).reason (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#480)' in ERC1155.
↳ _doSafeTransferAcceptanceCheck(address,address,address,uint256,
↳ uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155
↳ /ERC1155.sol#467-486) potentially used before declaration: revert(
↳ string)(reason) (lib/openzeppelin-contracts/contracts/token/
↳ ERC1155/ERC1155.sol#481)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,
↳ address,address,uint256[],uint256[],bytes).response (lib/
↳ openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#498)'
↳ in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#488-509) potentially used
↳ before declaration: response != IERC1155Receiver.
↳ onERC1155BatchReceived.selector (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#500)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,
↳ address,address,uint256[],uint256[],bytes).reason (lib/
↳ openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#503)'
↳ in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#488-509) potentially used
↳ before declaration: revert(string)(reason) (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#504)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#pre-declaration-usage-of-local-variables

Reentrancy in SemiFungibleVault.deposit(uint256,uint256,address) (
↳ src/SemiFungibleVault.sol#55-71):
    External calls:
    - _mint(receiver,id,shares,EMPTY) (src/SemiFungibleVault.sol
↳ #66)
```

```
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,
↳ amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
↳ ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assets,shares) (src/
↳ SemiFungibleVault.sol#68)
Reentrancy in SemiFungibleVault.mint(uint256,uint256,address) (src
↳ /SemiFungibleVault.sol#73-88):
    External calls:
    - _mint(receiver,id,assets,EMPTY) (src/SemiFungibleVault.sol
↳ #83)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,
↳ amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
↳ ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assets,shares) (src/
↳ SemiFungibleVault.sol#85)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-3

ERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32
↳ ) (lib/solmate/src/tokens/ERC20.sol#116-160) uses timestamp for
↳ comparisons
    Dangerous comparisons:
    - require(bool,string)(deadline >= block.timestamp,
↳ PERMIT_DEADLINE_EXPIRED) (lib/solmate/src/tokens/ERC20.sol#125)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#block-timestamp

FixedPointMathLib.rpow(uint256,uint256,uint256) (lib/solmate/src/
↳ utils/FixedPointMathLib.sol#74-160) performs a multiplication on
↳ the result of a division:
    -x = xxRound_rpow_asm_0 / scalar (lib/solmate/src/utils/
↳ FixedPointMathLib.sol#131)
    -zx_rpow_asm_0 = z * x (lib/solmate/src/utils/
↳ FixedPointMathLib.sol#136)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#divide-before-multiply

ERC1155._doSafeTransferAcceptanceCheck(address,address,address,
↳ uint256,uint256,bytes).response (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#476) is a local variable never
↳  initialized
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes).reason (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#503) is a local
↳ variable never initialized
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes).response (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#498) is a local
↳ variable never initialized
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,
↳ uint256,uint256,bytes).reason (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#480) is a local variable never
↳  initialized
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#uninitialized-local-variables

ERC1155._doSafeTransferAcceptanceCheck(address,address,address,
↳ uint256,uint256,bytes) (lib/openzeppelin-contracts/contracts/token
↳ /ERC1155/ERC1155.sol#467-486) ignores return value by
↳ IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,
↳ data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.
↳ sol#476-484)
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#488-509) ignores return value
↳ by IERC1155Receiver(to).onERC1155BatchReceived(operator,from,ids,
↳ amounts,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
↳ ERC1155.sol#497-507)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#unused-return

Vault.changeFee(uint256) (src/Vault.sol#292-294) should emit an
↳ event for:
    - feeTaken = _fee (src/Vault.sol#293)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#missing-events-arithmetic

Controller.constructor(address,address)._govToken (src/Controller.
↳ sol#62) lacks a zero-check on :
        - govToken = _govToken (src/Controller.sol#65)
Vault.constructor(address,string,string,address,uint256,address,
↳ int256,address)._token (src/Vault.sol#96) lacks a zero-check on :
        - tokenInsured = _token (src/Vault.sol#100)
Vault.constructor(address,string,string,address,uint256,address,
↳ int256,address)._treasury (src/Vault.sol#94) lacks a zero-check on
↳  :
        - treasury = _treasury (src/Vault.sol#102)
Vault.constructor(address,string,string,address,uint256,address,
↳ int256,address)._controller (src/Vault.sol#98) lacks a zero-check
↳ on :
        - controller = _controller (src/Vault.sol#106)
Vault.changeTreasury(address)._treasury (src/Vault.sol#296) lacks
↳ a zero-check on :
        - treasury = _treasury (src/Vault.sol#297)
VaultFactory.constructor(address)._treasury (src/VaultFactory.sol
↳ #47) lacks a zero-check on :
        - treasury = _treasury (src/VaultFactory.sol#50)
VaultFactory.setController(address)._controller (src/VaultFactory.
↳ sol#152) lacks a zero-check on :
        - controller = _controller (src/VaultFactory.sol#153)
VaultFactory.changeTreasury(address,uint256)._treasury (src/
↳ VaultFactory.sol#172) lacks a zero-check on :
        - treasury = _treasury (src/VaultFactory.sol#176)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#missing-zero-address-validation

Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,
↳ address,uint256,uint256,bytes).response (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#476)' in ERC1155.
↳ _doSafeTransferAcceptanceCheck(address,address,address,uint256,
↳ uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155
↳ /ERC1155.sol#467-486) potentially used before declaration:
↳ response != IERC1155Receiver.onERC1155Received.selector (lib/
↳ openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#477)
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,
↳ address,uint256,uint256,bytes).reason (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#480)' in ERC1155.
↳ _doSafeTransferAcceptanceCheck(address,address,address,uint256,
↳ uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155
↳ /ERC1155.sol#467-486) potentially used before declaration: revert(
↳ string)(reason) (lib/openzeppelin-contracts/contracts/token/
↳ ERC1155/ERC1155.sol#481)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,
↳ address,address,uint256[],uint256[],bytes).response (lib/
↳ openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#498)'
↳ in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#488-509) potentially used
↳ before declaration: response != IERC1155Receiver.
↳ onERC1155BatchReceived.selector (lib/openzeppelin-contracts/
contracts/token/ERC1155/ERC1155.sol#500)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).reason (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#503)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#488-509) potentially used before declaration: revert(string)(reason) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#504)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in VaultFactory.createNewMarket(uint256,address,int256,uint256,uint256,address) (src/VaultFactory.sol#68-120):
    External calls:
    - insurance.createAssets(epochBegin,epochEnd) (src/VaultFactory.sol#102)
    - risk.createAssets(epochBegin,epochEnd) (src/VaultFactory.sol#103)
    State variables written after the call(s):
    - indexEpochs[marketIndex].push(epochEnd) (src/VaultFactory.sol#105)
    - tokenToOracle[_token] = _oracle (src/VaultFactory.sol#108)
Reentrancy in VaultFactory.deployMoreAssets(uint256,uint256,uint256) (src/VaultFactory.sol#128-146):
    External calls:
    - Vault(insurance).createAssets(beginEpoch,endEpoch) (src/VaultFactory.sol#136)
    - Vault(risk).createAssets(beginEpoch,endEpoch) (src/VaultFactory.sol#137)
    State variables written after the call(s):
    - indexEpochs[marketIndex].push(endEpoch) (src/VaultFactory.sol#139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in VaultFactory.createNewMarket(uint256,address,int256,uint256,uint256,address) (src/VaultFactory.sol#68-120):
    External calls:
    - insurance.createAssets(epochBegin,epochEnd) (src/VaultFactory.sol#102)
    - risk.createAssets(epochBegin,epochEnd) (src/VaultFactory.sol#103)
    Event emitted after the call(s):
```

```
    - InsuranceMarketCreated(marketIndex,address(insurance),
 ↳ address(risk),_token,_strikePrice) (src/VaultFactory.sol#111-117)
Reentrancy in VaultFactory.deployMoreAssets(uint256,uint256,
 ↳ uint256) (src/VaultFactory.sol#128-146):
    External calls:
    - Vault(insurance).createAssets(beginEpoch,endEpoch) (src/
 ↳ VaultFactory.sol#136)
    - Vault(risk).createAssets(beginEpoch,endEpoch) (src/
 ↳ VaultFactory.sol#137)
    Event emitted after the call(s):
    - InsuranceEpochDeployed(marketIndex,Vault(insurance).
 ↳ tokenInsured(),Vault(insurance).strikePrice()) (src/VaultFactory.
 ↳ sol#141-145)
Reentrancy in SemiFungibleVault.deposit(uint256,uint256,address) (
 ↳ src/SemiFungibleVault.sol#55-71):
    External calls:
    - _mint(receiver,id,shares,EMPTY) (src/SemiFungibleVault.sol
 ↳ #66)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,
 ↳ amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
 ↳ ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assets,shares) (src/
 ↳ SemiFungibleVault.sol#68)
Reentrancy in Vault.deposit(uint256,uint256,address) (src/Vault.
 ↳ sol#121-146):
    External calls:
    - _mint(receiver,id,sharesMinusFee,EMPTY) (src/Vault.sol#139)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,
 ↳ amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
 ↳ ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,sharesMinusFee,shares) (src/
 ↳ Vault.sol#141)
Reentrancy in SemiFungibleVault.mint(uint256,uint256,address) (src
 ↳ /SemiFungibleVault.sol#73-88):
    External calls:
    - _mint(receiver,id,assets,EMPTY) (src/SemiFungibleVault.sol
 ↳ #83)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,
 ↳ amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
 ↳ ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assets,shares) (src/
↳ SemiFungibleVault.sol#85)
Reentrancy in Vault.mint(uint256,uint256,address) (src/Vault.sol
↳ #155-180):
    External calls:
    - _mint(receiver,id,assetsMinusFee,EMPTY) (src/Vault.sol#173)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,
↳ amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
↳ ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assetsMinusFee,shares) (src/
↳ Vault.sol#175)
Reentrancy in Controller.triggerDepeg(uint256,uint256) (src/
↳ Controller.sol#75-98):
    External calls:
    - insrVault.endEpoch(mintId,true) (src/Controller.sol#83)
    - riskVault.endEpoch(mintId,true) (src/Controller.sol#84)
    - insrVault.setClaimTVL(mintId,riskVault.idFinalTVL(mintId)) (
↳ src/Controller.sol#86)
    - riskVault.setClaimTVL(mintId,insrVault.idFinalTVL(mintId)) (
↳ src/Controller.sol#87)
    - insrVault.sendTokens(mintId,vaultsAddress[1]) (src/
↳ Controller.sol#89)
    - riskVault.sendTokens(mintId,vaultsAddress[0]) (src/
↳ Controller.sol#90)
    Event emitted after the call(s):
    - DepegInsurance(marketIndex,mintId,block.timestamp,
↳ getLatestPrice(insrVault.tokenInsured())) (src/Controller.sol
↳ #92-97)
Reentrancy in Controller.triggerEndEpoch(uint256,uint256) (src/
↳ Controller.sol#103-126):
    External calls:
    - insrVault.endEpoch(mintId,false) (src/Controller.sol#117)
    - riskVault.endEpoch(mintId,false) (src/Controller.sol#118)
    - insrVault.setClaimTVL(mintId,0) (src/Controller.sol#120)
    - riskVault.setClaimTVL(mintId,insrVault.idFinalTVL(mintId)) (
↳ src/Controller.sol#121)
    - insrVault.sendTokens(mintId,vaultsAddress[1]) (src/
↳ Controller.sol#123)
    Event emitted after the call(s):
    - InsuranceExpired(marketIndex,mintId) (src/Controller.sol
↳ #125)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-3
```

```
ERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32
↳ ) (lib/solmate/src/tokens/ERC20.sol#116-160) uses timestamp for
↳ comparisons
    Dangerous comparisons:
    - require(bool,string)(deadline >= block.timestamp,
↳ PERMIT_DEADLINE_EXPIRED) (lib/solmate/src/tokens/ERC20.sol#125)
Controller.triggerEndEpoch(uint256,uint256) (src/Controller.sol
↳ #103-126) uses timestamp for comparisons
    Dangerous comparisons:
    - require(bool,string)(block.timestamp >= mintId,Epoch for
↳ this insurance has not expired!) (src/Controller.sol#108-111)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#block-timestamp

FixedPointMathLib.rpow(uint256,uint256,uint256) (lib/solmate/src/
↳ utils/FixedPointMathLib.sol#74-160) performs a multiplication on
↳ the result of a division:
    -x = xxRound_rpow_asm_0 / scalar (lib/solmate/src/utils/
↳ FixedPointMathLib.sol#131)
    -zx_rpow_asm_0 = z * x (lib/solmate/src/utils/
↳ FixedPointMathLib.sol#136)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#divide-before-multiply

ERC1155._doSafeTransferAcceptanceCheck(address,address,address,
↳ uint256,uint256,bytes).reason (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#480) is a local variable never
↳  initialized
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,
↳ uint256,uint256,bytes).response (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#476) is a local variable never
↳  initialized
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes).reason (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#503) is a local
↳ variable never initialized
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes).response (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#498) is a local
↳ variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#uninitialized-local-variables

ERC1155._doSafeTransferAcceptanceCheck(address,address,address,
↳ uint256,uint256,bytes) (lib/openzeppelin-contracts/contracts/token
↳ /ERC1155/ERC1155.sol#467-486) ignores return value by
↳ IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,
↳ data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.
↳ sol#476-484)
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#488-509) ignores return value
↳ by IERC1155Receiver(to).onERC1155BatchReceived(operator,from,ids,
↳ amounts,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
↳ ERC1155.sol#497-507)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#unused-return

Vault.changeFee(uint256) (src/Vault.sol#292-294) should emit an
↳ event for:
    - feeTaken = _fee (src/Vault.sol#293)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#missing-events-arithmetic

Vault.constructor(address,string,string,address,uint256,address,
↳ int256,address)._token (src/Vault.sol#96) lacks a zero-check on :
        - tokenInsured = _token (src/Vault.sol#100)
Vault.constructor(address,string,string,address,uint256,address,
↳ int256,address)._treasury (src/Vault.sol#94) lacks a zero-check on
↳  :
        - treasury = _treasury (src/Vault.sol#102)
Vault.constructor(address,string,string,address,uint256,address,
↳ int256,address)._controller (src/Vault.sol#98) lacks a zero-check
↳ on :
        - controller = _controller (src/Vault.sol#106)
Vault.changeTreasury(address)._treasury (src/Vault.sol#296) lacks
↳ a zero-check on :
        - treasury = _treasury (src/Vault.sol#297)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#missing-zero-address-validation

Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,
↳ address,uint256,uint256,bytes).response (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#476)' in ERC1155.
↳ _doSafeTransferAcceptanceCheck(address,address,address,uint256,
↳ uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155
↳ /ERC1155.sol#467-486) potentially used before declaration:
↳ response != IERC1155Receiver.onERC1155Received.selector (lib/
openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#477)
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).reason (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#480)' in ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#467-486) potentially used before declaration: revert(string)(reason) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#481)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).response (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#498)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#488-509) potentially used before declaration: response != IERC1155Receiver.onERC1155BatchReceived.selector (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#500)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).reason (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#503)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#488-509) potentially used before declaration: revert(string)(reason) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#504)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in SemiFungibleVault.deposit(uint256,uint256,address) (src/SemiFungibleVault.sol#55-71):
	External calls:
	- _mint(receiver,id,shares,EMPTY) (src/SemiFungibleVault.sol#66)
		- IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476-484)
	Event emitted after the call(s):
	- Deposit(msg.sender,receiver,id,assets,shares) (src/SemiFungibleVault.sol#68)
Reentrancy in Vault.deposit(uint256,uint256,address) (src/Vault.sol#121-146):
	External calls:
	- _mint(receiver,id,sharesMinusFee,EMPTY) (src/Vault.sol#139)
```

```
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,sharesMinusFee,shares) (src/Vault.sol#141)
Reentrancy in SemiFungibleVault.mint(uint256,uint256,address) (src/SemiFungibleVault.sol#73-88):
    External calls:
    - _mint(receiver,id,assets,EMPTY) (src/SemiFungibleVault.sol#83)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assets,shares) (src/SemiFungibleVault.sol#85)
Reentrancy in Vault.mint(uint256,uint256,address) (src/Vault.sol#155-180):
    External calls:
    - _mint(receiver,id,assetsMinusFee,EMPTY) (src/Vault.sol#173)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assetsMinusFee,shares) (src/Vault.sol#175)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

ERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (lib/solmate/src/tokens/ERC20.sol#116-160) uses timestamp for comparisons
    Dangerous comparisons:
    - require(bool,string)(deadline >= block.timestamp,PERMIT_DEADLINE_EXPIRED) (lib/solmate/src/tokens/ERC20.sol#125)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

FixedPointMathLib.rpow(uint256,uint256,uint256) (lib/solmate/src/utils/FixedPointMathLib.sol#74-160) performs a multiplication on the result of a division:
    -x = xxRound_rpow_asm_0 / scalar (lib/solmate/src/utils/FixedPointMathLib.sol#131)
    -zx_rpow_asm_0 = z * x (lib/solmate/src/utils/FixedPointMathLib.sol#136)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).reason (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#480) is a local variable never initialized
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).response (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#498) is a local variable never initialized
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).response (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476) is a local variable never initialized
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).reason (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#503) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#467-486) ignores return value by IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476-484)
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#488-509) ignores return value by IERC1155Receiver(to).onERC1155BatchReceived(operator,from,ids,amounts,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#497-507)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Vault.changeFee(uint256) (src/Vault.sol#292-294) should emit an event for:
    - feeTaken = _fee (src/Vault.sol#293)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
```

```

Vault.constructor(address,string,string,address,uint256,address,
↳ int256,address)._token (src/Vault.sol#96) lacks a zero-check on :
        - tokenInsured = _token (src/Vault.sol#100)
Vault.constructor(address,string,string,address,uint256,address,
↳ int256,address)._treasury (src/Vault.sol#94) lacks a zero-check on
↳  :
        - treasury = _treasury (src/Vault.sol#102)
Vault.constructor(address,string,string,address,uint256,address,
↳ int256,address)._controller (src/Vault.sol#98) lacks a zero-check
↳ on :
        - controller = _controller (src/Vault.sol#106)
Vault.changeTreasury(address)._treasury (src/Vault.sol#296) lacks
↳ a zero-check on :
        - treasury = _treasury (src/Vault.sol#297)
VaultFactory.constructor(address)._treasury (src/VaultFactory.sol
↳ #47) lacks a zero-check on :
        - treasury = _treasury (src/VaultFactory.sol#50)
VaultFactory.setController(address)._controller (src/VaultFactory.
↳ sol#152) lacks a zero-check on :
        - controller = _controller (src/VaultFactory.sol#153)
VaultFactory.changeTreasury(address,uint256)._treasury (src/
↳ VaultFactory.sol#172) lacks a zero-check on :
        - treasury = _treasury (src/VaultFactory.sol#176)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#missing-zero-address-validation

Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,
↳ address,uint256,uint256,bytes).response (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#476)' in ERC1155.
↳ _doSafeTransferAcceptanceCheck(address,address,address,uint256,
↳ uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155
↳ /ERC1155.sol#467-486) potentially used before declaration:
↳ response != IERC1155Receiver.onERC1155Received.selector (lib/
↳ openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#477)
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,
↳ address,uint256,uint256,bytes).reason (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#480)' in ERC1155.
↳ _doSafeTransferAcceptanceCheck(address,address,address,uint256,
↳ uint256,bytes) (lib/openzeppelin-contracts/contracts/token/ERC1155
↳ /ERC1155.sol#467-486) potentially used before declaration: revert(
↳ string)(reason) (lib/openzeppelin-contracts/contracts/token/
↳ ERC1155/ERC1155.sol#481)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,
↳ address,address,uint256[],uint256[],bytes).response (lib/
↳ openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#498)'
↳ in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#488-509) potentially used
↳ before declaration: response != IERC1155Receiver.
↳ onERC1155BatchReceived.selector (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#500)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,
↳ address,address,uint256[],uint256[],bytes).reason (lib/
↳ openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#503)'
↳ in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,
↳ address,uint256[],uint256[],bytes) (lib/openzeppelin-contracts/
↳ contracts/token/ERC1155/ERC1155.sol#488-509) potentially used
↳ before declaration: revert(string)(reason) (lib/openzeppelin-
↳ contracts/contracts/token/ERC1155/ERC1155.sol#504)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#pre-declaration-usage-of-local-variables

Reentrancy in VaultFactory.createNewMarket(uint256,address,int256,
↳ uint256,uint256,address) (src/VaultFactory.sol#68-120):
    External calls:
    - insurance.createAssets(epochBegin,epochEnd) (src/
↳ VaultFactory.sol#102)
    - risk.createAssets(epochBegin,epochEnd) (src/VaultFactory.sol
↳ #103)
    State variables written after the call(s):
    - indexEpochs[marketIndex].push(epochEnd) (src/VaultFactory.
↳ sol#105)
    - tokenToOracle[_token] = _oracle (src/VaultFactory.sol#108)
Reentrancy in VaultFactory.deployMoreAssets(uint256,uint256,
↳ uint256) (src/VaultFactory.sol#128-146):
    External calls:
    - Vault(insurance).createAssets(beginEpoch,endEpoch) (src/
↳ VaultFactory.sol#136)
    - Vault(risk).createAssets(beginEpoch,endEpoch) (src/
↳ VaultFactory.sol#137)
    State variables written after the call(s):
    - indexEpochs[marketIndex].push(endEpoch) (src/VaultFactory.
↳ sol#139)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-2

Reentrancy in VaultFactory.createNewMarket(uint256,address,int256,
uint256,uint256,address) (src/VaultFactory.sol#68-120):
    External calls:
    - insurance.createAssets(epochBegin,epochEnd) (src/VaultFactory.sol#102)
    - risk.createAssets(epochBegin,epochEnd) (src/VaultFactory.sol#103)
    Event emitted after the call(s):
    - InsuranceMarketCreated(marketIndex,address(insurance),address(risk),_token,_strikePrice) (src/VaultFactory.sol#111-117)
Reentrancy in VaultFactory.deployMoreAssets(uint256,uint256,uint256) (src/VaultFactory.sol#128-146):
    External calls:
    - Vault(insurance).createAssets(beginEpoch,endEpoch) (src/VaultFactory.sol#136)
    - Vault(risk).createAssets(beginEpoch,endEpoch) (src/VaultFactory.sol#137)
    Event emitted after the call(s):
    - InsuranceEpochDeployed(marketIndex,Vault(insurance).tokenInsured(),Vault(insurance).strikePrice()) (src/VaultFactory.sol#141-145)
Reentrancy in SemiFungibleVault.deposit(uint256,uint256,address) (src/SemiFungibleVault.sol#55-71):
    External calls:
    - _mint(receiver,id,shares,EMPTY) (src/SemiFungibleVault.sol#66)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assets,shares) (src/SemiFungibleVault.sol#68)
Reentrancy in Vault.deposit(uint256,uint256,address) (src/Vault.sol#121-146):
    External calls:
    - _mint(receiver,id,sharesMinusFee,EMPTY) (src/Vault.sol#139)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,sharesMinusFee,shares) (src/Vault.sol#141)
Reentrancy in SemiFungibleVault.mint(uint256,uint256,address) (src/SemiFungibleVault.sol#73-88):
    External calls:
```

```
    - _mint(receiver,id,assets,EMPTY) (src/SemiFungibleVault.sol
 ↳ #83)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,
 ↳ amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
 ↳ ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assets,shares) (src/
 ↳ SemiFungibleVault.sol#85)
Reentrancy in Vault.mint(uint256,uint256,address) (src/Vault.sol
 ↳ #155-180):
    External calls:
    - _mint(receiver,id,assetsMinusFee,EMPTY) (src/Vault.sol#173)
        - IERC1155Receiver(to).onERC1155Received(operator,from,id,
 ↳ amount,data) (lib/openzeppelin-contracts/contracts/token/ERC1155/
 ↳ ERC1155.sol#476-484)
    Event emitted after the call(s):
    - Deposit(msg.sender,receiver,id,assetsMinusFee,shares) (src/
 ↳ Vault.sol#175)
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#reentrancy-vulnerabilities-3

ERC20.permit(address,address,uint256,uint256,uint8,bytes32,bytes32
 ↳ ) (lib/solmate/src/tokens/ERC20.sol#116-160) uses timestamp for
 ↳ comparisons
    Dangerous comparisons:
    - require(bool,string)(deadline >= block.timestamp,
 ↳ PERMIT_DEADLINE_EXPIRED) (lib/solmate/src/tokens/ERC20.sol#125)
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#block-timestamp
src/ analyzed (59 contracts with 57 detectors), 95 result(s) found
```

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

No major issues found by MythX.

THANK YOU FOR CHOOSING

// HALBORN