Phi Token Contracts
Security Review

JULY, 2025

www.chaindefenders.xyz
https://x.com/DefendersAudits

Lead Auditors

PeterSR

0x539.eth

Table of Contents

Protocol Summary

Phi Protocol is an innovative DeFi platform that combines fair token launches, dynamic fee mechanisms, and yield-generating strategies. Built on top of Uniswap V4's hook system, it provides a comprehensive ecosystem for creating, trading, and managing Board tokens while generating yield through integrated DeFi strategies.

It implements a so called "Artist-First Design":

- Artists create their own PhiEth tokens with automatic yield generation
- Community participation and ownership sharing through Board tokens
- Trust verification through attestation system

Disclaimer

The Chain Defenders team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Likelihood/Impact	High	Medium	Low
High	H	H/M	M
Medium	H/M	M	M/L
Low	M	M/L	L

Audit Details

Commit Hash

c530931944e0f3a52797b288cbeb84a4cdd16369

Scope

Id	Files in scope
1	BoardManager.sol
2	Board.sol
3	PhiEthImpl.sol
4	BoardHook.sol
5	PhiEthHooks.sol
6	TokenFeePool.sol
7	FairLaunch.sol
8	PoolSwap.sol
9	PremineZap.sol
10	AerodromeSwapper.sol
11	DynamicFeeCalculator.sol
12	FeeDistributor.sol
13	ReferralEscrow.sol
14	PhiEthFactory.sol
15	PhiEthDeployer.sol
16	BasePhiEthStrategy.sol
17	MorphoVaultStrategyImpl.sol
18	AaveV3StrategyImpl.sol
19	StrategyDeployer.sol
20	MuseManager.sol
21	FeeValidationLib.sol

Roles

Id	Roles
1	Owner
2	User

Executive Summary

Issues found

Severity	Count	Description
High	3	Critical vulnerabilities
Medium	8	Significant risks
Low	16	Minor issues with low impact
Informational	20	Best practices or suggestions
Gas	0	Optimization opportunities

Findings

High

High 01 Insufficient Initiator Validation In Premine Logic

Location

BoardHook.sol:333

Description

The premine logic in the `BoardHook` contract is vulnerable due to insufficient validation of the swap initiator. Currently, a premine swap is considered valid if it occurs in the same block and with the same `amountSpecified` as recorded in the `premineInfo` struct. However, the contract does not verify that the caller of the swap is the original initiator of the premine. This means that any user can trigger the premine swap, as long as they match the `amountSpecified` and block number, potentially allowing unauthorized parties to claim the premine allocation.

This issue arises because the `_sender` parameter is not reliably the original user (EOA) who requested the premine; it may be a contract (such as a router or zap).

Without tracking and validating the true initiator, the premine can be stolen by another party in the same block.

Proof of Concept

Add the following test to the `BoardHook.t.sol` file:

```
function test_PremineValidationWeakness() public {
    // Define creator and attacker addresses
    address creator = address(this);
    address attacker = address(0x123);

    // Attest a board
    string memory boardId = "test-board-123";
    createSignAndAttestV2(boardId);

    // Set premine amount and future publiphiAt
    uint256 premineAmount = 1000e18; // 1000 tokens with 18 decimals
    uint256 publiphiAt = block.timestamp;

    // Creator calls publiphi with premineAmount and future publiphiAt
    vm.startPrank(creator);
    address boardToken = boardHook.publiphi(
        IBoardHook.PubliphiParams({
            attestBoardId: boardId,
            initialTokenFairLaunch: supplyShare(50), // Adds liquidity for swaps
            premineAmount: premineAmount,
            creator: creator,
            phiEthToken: address(phiEth),
            publiphiAt: publiphiAt,
            initialPriceParams: abi.encode(""),
            feeCalculatorParams: abi.encode(1000),
            merkleRoot: bytes32(0),
            inspirer: address(0)
        })
    );
    vm.stopPrank();

    // Get the pool key for the board token
    PoolKey memory poolKey = boardHook.poolKey(boardToken);

    // Keep the same block number for the attacker's swap
    uint256 currentBlock = block.number;
    vm.roll(currentBlock);

```

```
39      // Provide PoolManager with sufficient phiEth to handle the take
        call
40      deal(address(phiEth), address(poolManager), 1000 ether);
41
42      // Attacker performs a swap to trigger the premine
43      vm.startPrank(attacker);
44      // Provide attacker with phiEth for the swap
45      deal(address(phiEth), attacker, 1000 ether);
46      IERC20(address(phiEth)).approve(address(poolSwap), type(uint256).
        max);
47
48      // Define swap parameters: exact output swap to receive
        premineAmount of boardToken
49      IPoolManager.SwapParams memory swapParams = IPoolManager.
        SwapParams({
50          zeroForOne: true, // phiEth (currency0) -> boardToken (
        currency1)
51          amountSpecified: int256(premineAmount),
52          sqrtPriceLimitX96: TickMath.MIN_SQRT_PRICE + 1
53      });
54
55      // Expect the PoolPremine event to be emitted
56      vm.expectEmit(true, true, true, true);
57      emit IBoardHook.PoolPremine(poolKey.toId(), int256(premineAmount))
        ;
58
59      // Execute the attacker's swap
60      poolSwap.swap(poolKey, swapParams);
61      vm.stopPrank();
62  }
```

Recommendation

Store the original initiator's address in the `premineInfo` struct when the premine is created. During premine validation, extract the true initiator from `_sender` (using the same logic as in `_validateWhitelist`: if `_sender` is a trusted router, use `IMsgSender(_sender).msgSender()`, otherwise use `_sender` directly or skip validation). Then, check that the caller matches the stored initiator before allowing the premine swap to proceed. This ensures only the authorized user can execute the premine swap and prevents unauthorized access.

```
1  if (boardManager.isTrustedRouter(_sender)) {
2      // If the sender is a trusted router, get the original EOA
3      initiator = IMsgSender(_sender).msgSender();
4  }
```

Status

Fixed

High 02 `withdrawAndSwap` Will Not Work Correctly

Location

AerodromeSwapper.sol:53

BoardHook.sol:796

Description

The `withdrawAndSwap()` function assumes the contract itself holds the `_amount` of `PhiEth` tokens necessary to perform the withdrawal. This creates two problems:

1. In `BoardHook`, `withdrawAndSwap` is used, but no funds are transferred to the `AerodromeSwapper` and the swap will fail.
2. If `withdrawAndSwap` is called without the boardHook, this introduces a problematic user flow:

- A user must send `PhiEth` tokens to the contract manually before calling `withdrawAndSwap()`.
- If another user (or bot) observes the transaction of the token transfer and calls `withdrawAndSwap()` before the original user, they can front-run and withdraw and swap the funds to their own `_recipient` address.

While front-running is currently less of a concern on the BASE chain due to its architecture, deploying this contract on other L2s or L1s where MEV is prevalent can make it vulnerable to front-running and fund theft.

Proof of Concept

Add this test to `AerodromeSwapperTest`:

```
address public recipient;
address public maliciousRecipient;
```

```
address public bob;
....


function setUp() public forkBaseBlock(5_000_000) {
   owner = makeAddr("owner");
   user = makeAddr("user");
   recipient = makeAddr("recipient");
   bob = makeAddr("bob");
   maliciousRecipient = makeAddr("maliciousRecipient");

....

function testWithdrawAndSwapSuccessFrontrun() public {
    uint256 swapAmount = 1 ether;

    deal(address(phiEthToken), address(bob), 5 ether);
    vm.prank(bob);
    phiEthToken.approve(address(this), swapAmount);
    vm.stopPrank();

    phiEthToken.transferFrom(bob, address(swapper), swapAmount);

    // Get initial balances
    uint256 initialRecipientBalance = IERC20(USDC).balanceOf(recipient
    );

    // Perform swap
    vm.expectEmit(true, false, false, false);
    emit TokensSwapped(USDC, swapAmount, 0); // amountOut will be > 0
    but we don't know exact value

    // not bob, but another user calls withdrawAndSwap and the funds
    are transferred to maliciousRecipient
    uint256 amountOut = swapper.withdrawAndSwap(address(phiEthToken),
    swapAmount, maliciousRecipient);

    // Verify results
    assertGt(amountOut, 0, "Should receive some USDC");
    assertEq(
        IERC20(USDC).balanceOf(recipient), initialRecipientBalance +
    amountOut, "Recipient should receive USDC"
    );
}
```

Recommendation

To mitigate both problems:

1. Require users to call `approve()` on the `PhiEth` token and let the contract `transferFrom()` the tokens directly within `withdrawAndSwap()`. This ensures atomicity and prevents front-running attacks.

```
IPhiEth(_phiEthToken).transferFrom(msg.sender, address(this), _amount)
    ;
IPhiEth(_phiEthToken).withdraw(_amount);
```

If you are not planning to use it alone, you can just call `transferFrom` before the `withdrawAndSwap` in the BoardHook.

Status

Fixed

High 03 Premine Logic Fails Due To Insufficient Funds

Location

BoardHook.sol

Description

During the initial deployment of a new Uniswap V4 pool in the `BoardHook` contract via the `publiphi()` function, a premine mechanism is optionally triggered to allocate tokens to the creator in the same transaction.

However, during mitigation phase we saw that the premine logic was implemented to run before the first swap of the pool. This is problematic because the `PoolManager` has not yet been supplied with the required `PhiEth` tokens to fulfill the settlement in `_settleDelta()`.

Specifically, the call to:

```
poolManager.take(_poolKey.currency1, address(this), uint256(int256(
    _delta.amount1())));
```

in `_settleDelta()` fails because the pool manager has no `PhiEth` to "give" to the hook contract. This is effectively a premature token transfer request on an empty balance, and results in a revert.

This manifests as a critical failure in premine execution that prevents the proper functioning of early token distribution and disrupts the protocol's launch flow.

Recommendation

Refactor the `premine` mechanism to ensure that the `PoolManager` possesses the required tokens before attempting to `take()` from it.

Status

Fixed

Response: Universal Router was used to ensure proper premine flow.

Medium

Mid 01 Excessive Swap Fee Exploit

Location

DynamicFeeCalculator.sol:205

Description

The `DynamicFeeCalculator` contract has a vulnerability in its fee calculation mechanism, specifically within the `determineSwapFee` function. This function is responsible for calculating the swap fee users pay when performing swaps in the pool. According to its documentation, the swap fee should never exceed a `MAXIMUM_FEE` of 2,000 basis points (bps), or 20%, due to supposed constraints enforced elsewhere in the contract (`_trackSwap`). However, this claim is inaccurate because of a flaw

in how the gas-based fee component is calculated, allowing the total swap fee to exceed this limit under certain conditions.

The root of the problem lies in the `calculateGasBasedFee` function, which determines the gas-based portion of the fee. This function uses the following formula:

```
uint256 gasBasedFeeScaled = (maxGasFee * txPriorityFee) /
    topPriorityFee;
```

- maxGasFee: A configurable upper limit for the gas-based fee (e.g., 10,000, representing 10%).
- txPriorityFee: The priority fee of the current transaction, capped at a maximum of 20 gwei (20,000,000,000 wei).
- topPriorityFee: The highest priority fee observed in the last block processed for the pool.

The vulnerability occurs when `topPriorityFee` is extremely small (e.g., 1 wei) while `txPriorityFee` is large (e.g., 20 gwei). In such cases, the ratio `txPriorityFee / topPriorityFee` becomes enormous, causing `gasBasedFeeScaled` to balloon far beyond `maxGasFee` or even the overall `MAXIMUM_FEE_SCALED` (200,000, or 2,000 bps). For example:

If `topPriorityFee = 1 wei`, `txPriorityFee = 20 gwei = 20,000,000,000 wei`, and `maxGasFee = 10,000`:

- `gasBasedFeeScaled = (10,000 * 20,000,000,000) / 1 = 200,000,000,000,000` (200 trillion bps).

This unbounded gas-based fee is then combined with a volume-based fee in `determineSwapFee`. If the gas-based fee has a high weight (e.g., 100%), the final swap fee can vastly exceed the documented 2,000 bps limit. This contradicts the contract's stated guarantees and exposes users to unexpectedly high fees.

The impact of this issue is significant:

- Economic Disruption: Users could face fees that are orders of magnitude higher than anticipated, rendering the pool impractical or causing financial losses.
- Manipulation Risk: Attackers could manipulate the fee by setting a low `topPriorityFee` in block N rendering swaps useless due to extreme fees.
- Trust Issues: This flaw undermines confidence in the ecosystem by violating its documented fee constraints.

Proof of Concept

Add the following test to the `DynamicFeeCalculator.t.sol` file:

```
function test_SwapFeeExceedsMaximumDueToGasFee() public {
    PoolId poolId = _poolKey.toId();

    // Step 1: Configure gas fee parameters to emphasize gas-based fee
    vm.startPrank(address(this));
    feeCalculator.setGasFeeParameters(
        poolId,
        1000,    // 1% min gas fee (1,000 bps)
        10_000,  // 10% max gas fee (10,000 bps, scaled)
        0,       // 0% volume weight
        100      // 100% gas weight
    );
    vm.stopPrank();

    // Step 2: Simulate a swap to set a very low topPriorityFee
    uint256 baseFee = 20 gwei;
    uint256 lowPriorityFee = 1 wei; // Extremely low priority fee
    vm.fee(baseFee);
    vm.txGasPrice(baseFee + lowPriorityFee);

    vm.startPrank(BOARD_HOOK);
    _trackSwap(1 ether); // Perform a swap to set topPriorityFee to 1 wei
    vm.stopPrank();

    // Step 3: Simulate a swap with a high txPriorityFee
    uint256 highPriorityFee = 20 gwei; // High priority fee
    vm.txGasPrice(baseFee + highPriorityFee);

    // Step 4: Calculate the swap fee
    uint24 swapFee = feeCalculator.determineSwapFee(_poolKey, _getSwapParams(5 ether), 100);

    // Step 5: Verify that the swap fee exceeds the maximum (2,000 bps)
    assertGt(swapFee, 2000, "Swap fee should exceed the maximum fee due to gas-based fee component");
}
```

Recommendation

To fix this vulnerability, the gas-based fee calculation must be constrained to prevent it from exceeding reasonable limits, such as `maxGasFee` or the over-

all `MAXIMUM_FEE_SCALED`. One effective solution is to clamp the gas-based fee within a defined range in the `calculateGasBasedFee` function. Here's a proposed modification:

```
function calculateGasBasedFee(PoolId _poolId) internal view returns (uint256) {
    PoolInfo storage poolInfo = poolInfos[_poolId];
    uint64 minGasFee = poolInfo.minGasFee == 0 ? DEFAULT_MIN_GAS_FEE : poolInfo.minGasFee;
    uint64 maxGasFee = poolInfo.maxGasFee == 0 ? DEFAULT_MAX_GAS_FEE : poolInfo.maxGasFee;
    uint128 lastBlockSeen = poolInfo.lastBlockSeen;
    uint256 topPriorityFee = topPriorityFees[_poolId][lastBlockSeen];
    uint256 rawTxPriorityFee = tx.gasprice - block.basefee;
    uint256 txPriorityFee = rawTxPriorityFee > MAX_PRIORITY_FEE ? MAX_PRIORITY_FEE : rawTxPriorityFee;
    if (topPriorityFee != 0) {
        uint256 gasBasedFeeScaled = (maxGasFee * txPriorityFee) / topPriorityFee;
        if (gasBasedFeeScaled < minGasFee) {
            return minGasFee;
        } else if (gasBasedFeeScaled > maxGasFee) {
            return maxGasFee;
        } else {
            return gasBasedFeeScaled;
        }
    } else {
        return minGasFee;
    }
}
```

Status

Fixed

Mid 02 Underflow In Decay Factor Calculation

Location

DynamicFeeCalculator.sol:244

Description

There is a vulnerability in the `DynamicFeeCalculator` contract, in the calculation of the `decayFactor` within the `trackSwap` function. The code computes `decayFactor` as `1e18 - (DECAY_RATE_PER_SECOND * timeElapsed)`. `DECAY_RATE_PER_SECOND` is `277_777_777_777_777` (approximately `1e18 / 3600`). If `timeElapsed` exceeds 3600 seconds (1 hour), the product `(DECAY_RATE_PER_SECOND * timeElapsed)` becomes greater than `1e18`, causing an underflow in the subtraction.

Due to the fact that the calculation executes in an `unchecked` block this results in `decayFactor` wrapping around to a very large value due to unsigned integer underflow. When this large `decayFactor` is used to update `accumulatorWeightedVolume`, it can cause the accumulator to be set to an abnormally high value, which in turn leads to excessively high swap fees. This breaks the intended fee decay logic and can make the protocol unusable or unfairly expensive for users.

Proof of Concept

Add the following test to the `DynamicFeeCalculator.t.sol` file:

```
function test_DecayFactorUnderflow(uint timeWindow) public {
    vm.assume(timeWindow ⩾ 0 && timeWindow ⩽ 3600); // Ensure timeWindow is between 1 second and 59 minutes

    PoolId poolId = _poolKey.toId();

    // Step 1: Perform an initial swap to set up the accumulator
    vm.startPrank(BOARD_HOOK);
    _trackSwap(1 ether);
    vm.stopPrank();

    // Step 2: Warp time to be 3600 seconds
    vm.warp(block.timestamp + timeWindow);

    // Step 3: Perform another swap to trigger the decay factor calculation
    vm.startPrank(BOARD_HOOK);
    _trackSwap(1 ether);
    vm.stopPrank();

    // Step 4: Calculate the swap fee before the underflow occurs
    uint24 swapFeeBeforeUnderflow = feeCalculator.determineSwapFee(_poolKey, _getSwapParams(1 ether), 100);

    // Step 5: Warp time to exceed 3600 seconds (e.g., 3601 seconds)
```

```
23      vm.warp(block.timestamp + 3601);
24
25      // Step 6: Perform another swap to trigger the decay factor
        calculation with underflow
26      vm.startPrank(BOARD_HOOK);
27      _trackSwap(1 ether);
28      vm.stopPrank();
29
30      // Step 7: Calculate the swap fee to observe the effect of the
        underflow
31      uint24 swapFeeAfterUnderflow = feeCalculator.determineSwapFee(
        _poolKey, _getSwapParams(1 ether), 100);
32
33      // Step 8: Assert that the fee is abnormally high due to the
        underflow
34      assertGt(swapFeeAfterUnderflow, swapFeeBeforeUnderflow, "Fee
        should be incorrectly high due to decay factor underflow");
35  }
```

Recommendation

Clamp the value of `(DECAY_RATE_PER_SECOND * timeElapsed)` to a maximum of `1e18` before performing the subtraction. This ensures that `decayFactor` never underflows and always remains within the valid range of `0` to `1e18`. Alternatively, add a check to set `decayFactor` to zero if `timeElapsed` is greater than or equal to 3600 seconds. This will prevent the accumulator from being artificially inflated and maintain the intended fee decay behavior.

```
uint256 decay = DECAY_RATE_PER_SECOND * timeElapsed;

if(decay > 1e18) {
    decay = 1e18; // Clamp decay to a maximum of 1e18
}

uint256 decayFactor = 1e18 - decay;

if(decayFactor == 0) {
    decayFactor = 1; // Avoid division by zero
}
```

Status

Fixed

Mid 03 AAVE v3 Missing Claim Incentives Function

Location

AaveV3StrategyImpl.sol:17

Description

The Aave protocol provides “Incentives” (e.g., staking rewards or liquidity mining rewards, see here: https://aave.com/docs/primitives/incentives) to users who supply assets to the protocol. These incentives are typically distributed in the form of additional tokens (e.g., AAVE or other governance tokens) and can be claimed by users who interact with Aave’s incentive mechanisms.

In the current implementation of the `AaveV3StrategyImpl.sol` contract, there is no functionality to claim these incentives. This is a missing feature that could prevent users from accessing the full benefits of supplying assets to Aave.

In `Base`, the Aave rewards contract is:

https://basescan.org/address/0xf9cc4F0D883F1a1eb2c253bdb46c254Ca51E1F44

Recommendation

To address this issue, we need to add a function that allows claiming rewards from Aave (similar to the one in Morpho). This involves interacting with Aave’s Incentives Controller or Rewards Distributor contracts.

Status

Fixed

Mid 04 Inconsistent Fee Unit Scaling

Location

DynamicFeeCalculator.sol:91

DynamicFeeCalculator.sol:92

DynamicFeeCalculator.sol:93

Description

The `DynamicFeeCalculator` contract defines default gas fee constants (`DEFAULT_MIN_GAS_FEE`, `DEFAULT_MAX_GAS_FEE`, and `ABSOLUTE_MAX_GAS_FEE`) that are incorrectly scaled. These values are intended to represent percentages in basis points multiplied by 100 (due to their current usage in `determineSwapFee`) (i.e., 1% should be represented as 10,000), but the current values are set as if they are in basis points only. For example, `DEFAULT_MIN_GAS_FEE` is set to 800 (0.08%), `DEFAULT_MAX_GAS_FEE` is set to 20,000 (2.0%), and `ABSOLUTE_MAX_GAS_FEE` is set to 10,000 (1.0%). This mis-scaling leads to two main issues:

- The actual minimum and maximum gas fee defaults are much lower than intended, resulting in lower fees than expected.
- The absolute maximum gas fee (used as an upper bound in `setGasFeeParameters`) is set too low, preventing the contract owner from setting a gas fee higher than 1%, which may be insufficient for certain market conditions.

This can cause incorrect fee calculations, limit protocol flexibility, and potentially reduce protocol revenue.

Proof of Concept

Add the following tests to the `DynamicFeeCalculator.t.sol` file:

```
function test_InconsistentFeeUnitScaling() public {
    PoolId poolId = _poolKey.toId();

    // Set weights to 0% volume, 100% gas to isolate gas-based fee
    vm.startPrank(address(this));
    feeCalculator.setGasFeeParameters(
        poolId,
        9_999,
        10_000,    // Close min/max to force fixed value
        0,         // 0% volume weight
        100        // 100% gas weight
    );
    vm.stopPrank();

    // Simulate network conditions
    uint256 baseFee = 20 gwei;
    uint256 priorityFee = 1 gwei;
    vm.fee(baseFee);
    vm.txGasPrice(baseFee + priorityFee);

```

```
21      // First swap to initialize state
22      vm.startPrank(BOARD_HOOK);
23      _trackSwap(1 ether);
24      vm.stopPrank();
25
26      // Get fee with baseFee = 0 to see raw gas component
27      uint24 swapFee = feeCalculator.determineSwapFee(_poolKey,
        _getSwapParams(1 ether), 0);
28
29      // With correct scaling we'd expect 1000 (10%) but get 100 (1%)
30      assertEq(swapFee, 100, "Gas fee is 10% (1000 basis points)");
31  }
```

And

```
1   function test_InconsistentFeeUnitScalingBlocksNormalValue() public {
2       PoolId poolId = _poolKey.toId();
3
4       // Try to set a fee that is more than 1% -> reverts with
        MaxFeeTooHigh
5       vm.expectRevert(DynamicFeeCalculator.MaxFeeTooHigh.selector);
6       vm.startPrank(address(this));
7       feeCalculator.setGasFeeParameters(
8           poolId,
9           10_000,
10          10_001,
11          0,
12          100
13      );
14      vm.stopPrank();
15  }
```

Recommendation

Update the default gas fee constants to use the correct scaling. Multiply the intended basis point values by 100 to match the scaling used elsewhere in the contract. For example:

- Set `DEFAULT_MIN_GAS_FEE` to 8,000 for 0.8%
- Set `DEFAULT_MAX_GAS_FEE` to 200,000 for 20%
- Set `ABSOLUTE_MAX_GAS_FEE` to the appropriate upper limit (e.g., 1,000,000 for 100% if desired or 100,000 for 10% to keep current intention)

Review all usages of these constants to ensure consistency and correct fee calculations throughout the contract.

Status

Fixed

Mid 05 Aave V3 Is Pausable

Location

AaveV3StrategyImpl.sol

Description

The Aave V3 protocol includes a “Pausable” feature, which allows the protocol administrators to pause all or specific operations (such as deposits, withdrawals, or borrows) in response to emergencies or governance decisions. When Aave V3 is paused, any attempt to interact with its core functions will revert. In the current implementation of `AaveV3StrategyImpl`, there is no explicit handling for this scenario. If Aave is paused, user or protocol interactions (such as deposits or withdrawals) will fail with generic or unclear errors, making it difficult for users and integrators to understand the cause of the failure. This lack of clear error propagation can lead to confusion, poor user experience, and challenges in automated monitoring or recovery.

Recommendation

Add explicit checks or error handling in the strategy functions that interact with Aave V3. When a paused state is detected (e.g., by catching a revert or querying Aave’s pause status before making a call), revert with a custom, descriptive error such as `AavePaused()`. This will make it clear to users and integrators that the failure is due to Aave V3 being paused, not a bug or unrelated issue. Additionally, consider surfacing this status in view functions or through events so that frontends and monitoring systems can react appropriately. Document the pausability of the Aave V3 protocol in the documentation so users have expectation of such scenario.

Status

Fixed

Mid 06 Top Contributor Sybil Attack

Location

MuseManager.sol

Description

The `MuseManager::tryInspire` function is called every time a `BoardToken` is transferred. This function is used to track contributions and determine the top contributor, who becomes eligible for fee rewards via the `inspiredFeeRecipient`. However, this system is vulnerable to a Sybil attack, where a user can manipulate the contribution tracking to appear as the top contributor.

Attack Scenario

1. A malicious contributor creates multiple Sybil addresses (e.g., 100 addresses).
2. The contributor transfers `BoardToken`s to `sybilRecipient1`, who then sends them back to the original contributor.
3. This process is repeated for all Sybil addresses, artificially inflating the contributor's interaction count.
4. As a result, the contributor is falsely marked as the top contributor for the epoch.

This behavior undermines the integrity of the top contributor selection mechanism and could lead to unfair distribution of fees.

Proof of Concept

Add the following test case to `MuseManager.t.sol`:

```
address[] public sybilRecipients;

function test_TryInspire_SybilAttack() public {
    uint256 amount = museManager.inspirationThreshold();
    uint256 currentEpoch = museManager.currentEpoch();

    // Bob initiates a valid inspire
```

```
        vm.prank(boardToken);
        bool result = museManager.tryInspire(bob, alice, amount);
        assertTrue(result);
        vm.stopPrank();

        // Confirm Bob is top contributor
        assertEq(museManager.topContributor(boardToken, currentEpoch), bob
    );

        // Create Sybil recipients
        for (uint256 i = 0; i < 10; i++) {
            address sybilRecipient = makeAddr(string(abi.encodePacked("
    sybilRecipient", i)));
            sybilRecipients.push(sybilRecipient);
            vm.deal(sybilRecipient, 1 ether);
        }

        // Alice performs transfers to Sybil addresses
        for (uint256 i = 0; i < 10; i++) {
            vm.prank(boardToken);
            result = museManager.tryInspire(alice, sybilRecipients[i],
    amount);
            assertTrue(result);
        }

        // Verify Alice becomes top contributor
        assertEq(museManager.topContributor(boardToken, currentEpoch),
    alice);
        assertEq(museManager.topContributorCount(boardToken, currentEpoch)
    , 10);
    }
```

Recommendation

Introduce a fee mechanism to slash every transfer.

Status

Acknowledged

Response: Accepted as intended behavior.

Mid 07 Aerodrome Is Pausable

Location

BoardHook.sol:796

Description

The `BoardHook::_distributeFees` function relies on `AerodromeSwapper`, which uses an `aeroRouter` for swap operations. However, the `aeroRouter` can be paused at any time by its maintainers. If this occurs, any call to `ISwapper(phiEthFactory.poolSwapZap()).withdrawAndSwap( ... )` will revert.

This creates a vulnerability: if the router is paused, fee distribution fails, and the entire swap operation reverts.

Recommendation

Instead of immediately performing the swap in `BoardHook::_distributeFees` (a push model), consider adopting a pull model where:

The fee is allocated and recorded, but not immediately swapped.

A protocolFeeRecipient can later claim these fees and swaps.

Status

Fixed

Mid 08 Premine Allocation Loss Due to Fragile Same-Block Requirement

Location

BoardHook.sol:333

Description

The premine functionality in `BoardHook` contains a vulnerability where premine logic can be permanently missed due to network congestion. Users have a choice to either use the `PremineZap` to atomically execute `publiphi` and take the premine amount or to do it manually by executing the two transactions sequentially. Choosing to execute the transactions sequentially is fundamentally flawed because:

1. Atomicity Requirement: Premine validation requires:

   ```
   _premineInfo.blockNumber == block.number
   ```

   This demands perfect same-block execution of two separate transactions.
2. Network Reality: On Base chain (2-second blocks):
   - First transaction (creation) can be executed in block N
   - Second transaction (swap) can arrive in block N+1 or later
3. Permanent Failure: Missed premines:
   - Cannot be reclaimed or reattempted
   - Waste user gas costs
4. Economic Impact: Failed premines:
   - Deny creators their guaranteed allocation
   - Distort token distribution
   - Damage protocol reputation

Proof of Concept

Add the following test to the `BoardHook.t.sol` file:

```
function test_PremineMissDueToNetworkCongestion() public {
    address creator = address(this);
    address premineUser = address(0x123);

    // Attest a board
    string memory boardId = "test-board-123";
    createSignAndAttestV2(boardId);

    // Set premine amount
    uint256 premineAmount = 1000e18;
```

```
11      uint256 publiphiAt = block.timestamp; // Schedule for immediate
    launch
12
13      // Creator deploys token with premine
14      vm.startPrank(creator);
15      address boardToken = boardHook.publiphi(
16          IBoardHook.PubliphiParams({
17              attestBoardId: boardId,
18              initialTokenFairLaunch: supplyShare(50),
19              premineAmount: premineAmount,
20              creator: creator,
21              phiEthToken: address(phiEth),
22              publiphiAt: publiphiAt,
23              initialPriceParams: abi.encode(""),
24              feeCalculatorParams: abi.encode(1000),
25              merkleRoot: bytes32(0),
26              inspirer: address(0)
27          })
28      );
29      vm.stopPrank();
30
31      PoolKey memory poolKey = boardHook.poolKey(boardToken);
32      PoolId poolId = poolKey.toId();
33
34      vm.roll(block.number + 1);
35
36      uint256 nextBlock = block.number + 1; // Force next block
37      vm.roll(nextBlock);
38
39      // Provide PoolManager with sufficient phiEth to handle the take
    call
40      deal(address(phiEth), address(poolManager), 1000 ether);
41
42      // Prepare swap
43      deal(address(phiEth), premineUser, 1000 ether);
44      vm.startPrank(premineUser);
45      IERC20(address(phiEth)).approve(address(poolSwap), type(uint256).
    max);
46
47      IPoolManager.SwapParams memory swapParams = IPoolManager.
    SwapParams({
48          zeroForOne: true,
49          amountSpecified: int256(premineAmount),
50          sqrtPriceLimitX96: TickMath.MIN_SQRT_PRICE + 1
51      });
52
53      // Expect the PoolPremine event to be emitted
```

```
54      vm.expectEmit(true, true, true, true);
55      emit IBoardHook.PoolPremine(poolKey.toId(), int256(premineAmount))
       ;
56
57      // Premine should fail - no event emitted thus test fails
58      poolSwap.swap(poolKey, swapParams);
59      vm.stopPrank();
60  }
```

It fails due to no `PoolPremine` event being emitted.

Recommendation

Implement a premine time window:

```
1 uint256 constant PREMINE_WINDOW_BLOCKS = 5;
```

Instead of allowing premines only on the same block create a deadline in which the creator can successfully premine the intended premine amount.

Another possible mitigation is disallowing manual `publiphi` execution and leaving this logic to the `PremineZap` contract which does it atomically.

Status

Acknowledged

Response: In practice, `publiphi` transactions will always be executed using `PremineZap`.

Low

Low 01 Wrong Old Board Manager In Event Emission

Location

MuseManager.sol:402

Description

In the `setBoardManager` function of the `MuseManager` contract, the `BoardManagerUpdated` event is emitted to signal a change in the `boardManager` address. However, the `oldBoardManager` variable is incorrectly set to the new value (`_boardManager`) instead of the previous value of `boardManager` before the update. As a result, the event logs both the old and new values as the new address, making it impossible to track what the previous `boardManager` was. This reduces the transparency and auditability of contract state changes, which can be important for off-chain monitoring and debugging.

Recommendation

Before updating the `boardManager` state variable, store its current value in `oldBoardManager`. Then, emit the `BoardManagerUpdated` event using the correct previous value and the new value. For example:

```
function setBoardManager(address _boardManager) external onlyOwner {
    if (_boardManager == address(0)) revert InvalidBoardManager();
    address oldBoardManager = boardManager; // Store the current value
    boardManager = _boardManager;
    emit BoardManagerUpdated(oldBoardManager, _boardManager);
}
```

This ensures the event accurately reflects the change from the old to the new `boardManager` address.

Status

Fixed

Low 02 No Tie-Breaking Logic For Top Contributor

Location

MuseManager.sol:160

Description

The current logic for determining the top contributor in the `MuseManager` contract only considers the number of unique active recipients each contributor has inspired within an epoch. If two or more contributors reach the same count, the contract simply assigns top contributor status to the first one who reaches that count, with no further tie-breaking mechanism. This approach can be unfair, as contributors who inspire the same number of unique users but contribute more overall (e.g., by inspiring with larger token amounts) are not recognized. It also makes the outcome dependent on transaction ordering, which can be influenced by network conditions or miner/validator behavior.

Recommendation

Implement an additional tie-breaking criterion for selecting the top contributor. One effective approach is to track the total amount of tokens inspired by each contributor in addition to the unique recipient count. In the event of a tie in unique recipient count, the contributor with the higher total inspired token amount should be selected as the top contributor. This ensures a fairer and more meaningful ranking, rewarding both breadth (unique recipients) and depth (total contribution). Update the relevant data structures and logic to maintain and compare this additional metric when determining the top contributor.

Status

Acknowledged

Response: The current implementation is working as intended. The `inspirationThreshold` mechanism ensures that any contribution above 1,000 tokens is valued equally - this is a deliberate design choice to promote democratic participation rather than plutocracy.

Low 03 `isActiveWallet` Is Susceptible To A Flash Loan Attack

Location

MuseManager.sol:247

Description

The `isActiveWallet` function in the `MuseManager` contract checks if a wallet is "active" by verifying that its ETH balance is greater than or equal to the hard-coded `ACTIVE_WALLET_MIN_BALANCE` (0.005 ether). However, this check is vulnerable to flash loan or temporary funding attacks: an attacker can momentarily transfer or flash loan enough ETH to their wallet to pass the balance check, trigger the desired protocol action (such as being counted as an active contributor or recipient), and then immediately withdraw the ETH. This means wallets that are not genuinely active or funded can still appear as active, undermining the integrity of the contributor and inspiration tracking system.

Recommendation

To mitigate this risk, consider implementing additional or alternative criteria for wallet activity that cannot be easily manipulated in a single transaction. Options include:

- Tracking historical balances over a period of time (e.g., using an oracle or snapshot mechanism).
- Requiring a minimum balance to be maintained for a certain number of blocks or epochs before a wallet is considered active.
- Making `ACTIVE_WALLET_MIN_BALANCE` configurable or dynamic, adjusting it based on current ETH price or network conditions to make manipulation more costly.
- Combining ETH balance checks with other activity metrics, such as transaction count or token holdings, to better assess genuine activity.

These measures will make it significantly harder for attackers to game the system using flash loans or temporary transfers.

Status

Acknowledged

Response: This is not considered as a problem at all in the current implementation. Since the `BoardHook` can update the Muse contract, no fixes will be applied for this issue now.

Low 04 Wrong `PoolFeesDistributed` Event Emission

Location

BoardHook.sol

Description

The `PoolFeesDistributed` event in the `BoardHook` contract is intended to log the breakdown of fee distribution for a pool, including the total distributed amount and the specific amounts allocated to the creator, swapper, muse (inspired), and protocol. However, the current implementation emits the event with incorrect values: it passes `creatorFee` twice (for both the creator and swapper fields) and omits the actual values for the swapper, muse, and protocol fees. This misalignment between the event parameters and the actual fee distribution makes it difficult for off-chain services, analytics, and users to accurately track how fees are being distributed. It can lead to confusion, incorrect reporting, and a lack of transparency regarding protocol operations.

Recommendation

Update the event emission to correctly reflect the intended fee breakdown. Specifically, emit the `PoolFeesDistributed` event with the following arguments in order: `poolId`, `distributeAmount`, `creatorFee`, `swapperFee`, `inspiredFee`, and `protocolFee`. This ensures that the event data matches the actual distribution logic and provides accurate, transparent information for monitoring and analytics. The corrected line should be:

```
1 emit PoolFeesDistributed(poolId, distributeAmount, creatorFee,
      swapperFee, inspiredFee, protocolFee);
```

Status

Fixed

Low 05 Default Value Is Set Only When Both Values Are Not Initialized

Location

DynamicFeeCalculator.sol:220

Description

In the `DynamicFeeCalculator` contract, the logic for setting default weights for the volume-based and gas-based fee components only applies if both `weightVolume` and `weightGas` are zero. If, for any reason, one of these weights is initialized (non-zero) while the other remains zero, the default values are not applied. This results in the non-initialized weight staying at zero, which can cause the fee calculation to be skewed or incorrect. For example, if `weightVolume` is set to 80 and `weightGas` is left at zero, the gas-based component will be ignored entirely, even though the intention may have been to use the default split (80/20). This can lead to unexpected or unfair fee outcomes.

Recommendation

Update the logic to assign default values to any unset (zero) weight individually, rather than only when both are zero. Alternatively, require that both weights are explicitly set and validate that their sum matches the expected denominator (e.g., 100). This ensures that the fee calculation always uses valid and intended weights, preventing accidental misconfiguration and ensuring the correct blend of volume-based and gas-based fee components.

Status

Acknowledged

Response: The `setGasFeeParameters` function already validates that weights sum to 100.

Low 06 Non Working Deadline For Swapping

Location

AerodromeSwapper.sol:79

Description

In the `AerodromeSwapper` contract, the deadline parameter for the `swapExactTokensForTokens` function is currently set as `block.timestamp + 300`, meaning the swap will be valid for the next 5 minutes from the time the transaction is mined. However, the Aerodrome router's internal check is `deadline ⩾ block.timestamp`, which means that the actual check will be `block.timestamp + 300 ⩾ block.timestamp` which is always true. This approach makes the deadline parameter redundant. It does not allow users to specify their own deadline, which is a common DeFi best practice for protecting against front-running and unexpected delays. Without user control, users cannot enforce stricter deadlines for their swaps, potentially exposing them to unfavorable execution if network congestion or miner delays occur.

Recommendation

Modify the `withdrawAndSwap` function to accept a `deadline` parameter from the user, and pass this value directly to the router's `swapExactTokensForTokens` call. This gives users full control over the validity window of their swap, allowing them to set a deadline that matches their risk tolerance and transaction expectations. For example:

```
function withdrawAndSwap(
    address _phiEthToken,
    uint256 _amount,
    address _recipient,
    uint256 deadline // <-- new parameter
) external returns (uint256 amountOut) {
     ...
    uint256[] memory amounts =
        aeroRouter.swapExactTokensForTokens(_amount, minAmountOut,
    routes, _recipient, deadline);
     ...
}
```

This change improves user safety and aligns with standard DeFi practices.

Status

Acknowledged

Response: The design is specifically tailored for small-value transactions, with a clear intent to prioritize user experience by reducing revert frequency. In case any issues arise, updates are considered through `PhiEthFactory`'s `setPoolSwapZap` function.

Low 07 EIP7572 Non Compliance

Location

Board.sol

Description

The `contractURI()` function in the `Board` contract is intended to comply with EIP-7572, which defines a standard JSON schema for contract-level metadata. According to the EIP, the returned JSON object must include specific fields such as `name`, `symbol`, `description`, `image`, `banner_image`, `featured_image`, `external_link`, and `collaborators`, with `"name"` being required. However, the current implementation returns a JSON object that includes non-standard fields like `uri` and `creator`, which are not part of the EIP-7572 schema. This non-compliance can cause issues for platforms, tools, or marketplaces that expect the standard schema, potentially leading to metadata parsing errors or incomplete display of contract information.

Recommendation

Update the `contractURI()` implementation to strictly conform to the EIP-7572 schema. Only include the fields specified in the standard (`name`, `symbol`, `description`, `image`, `banner_image`, `featured_image`, `external_link`, and `collaborators`). Remove any non-standard fields such as `uri` and `creator` from the returned JSON. If additional metadata is needed, consider using the optional fields provided by the EIP or proposing an extension to the standard. This will ensure compatibility with third-party services and maintain adherence to the evolving Ethereum metadata standards.

Status

Acknowledged

Response: Parsers typically just ignore non-standard fields like `uri` and `creator`.

Low 08 PremineZap Can Overrefund Ether

Location

PremineZap.sol:43

Description

In the `PremineZap` contract's `publiphi` function, after performing the premine and swap, the contract refunds any remaining ETH balance to the user by transferring the entire contract balance (`address(this).balance`). However, if the contract already held ETH from a previous transaction or accidental transfer before the current call, this logic will refund not only the user's leftover ETH but also any pre-existing ETH in the contract. This results in the user receiving more ETH than they are entitled to, effectively allowing users to drain unrelated funds from the contract. This overrefund can lead to loss of funds for the protocol or other users.

Recommendation

Track the contract's ETH balance at the start of the function and only refund the excess ETH that was sent in the current transaction. One way to do this is to record the initial balance at the beginning of the function, and after all operations, refund only the difference between the current balance and the initial balance (minus the ETH actually spent). For example:

```
uint256 initialBalance = address(this).balance - msg.value;
// ... perform operations ...
uint256 refundAmount = address(this).balance - initialBalance;
if (refundAmount > 0) {
    SafeTransferLib.safeTransferETH(msg.sender, refundAmount);
}
```

This ensures that only the ETH associated with the current transaction is refunded, preventing accidental or malicious overrefunds.

Status

Fixed

Low 09 Missing Disable Initializers Call

Location

BoardManager.sol:57-68 — in the constructor

Description

The `BoardManager` contract inherits from `Initializable` and uses the `initializer` modifier in the `initialize()` function to prevent multiple initializations. However, the constructor does not call `_disableInitializers()`, which leaves the implementation contract vulnerable to initialization attacks.

Without `_disableInitializers()` in the constructor, an attacker could potentially initialize the implementation contract directly before it's used as a proxy implementation. This could lead to the implementation contract being in an initialized state, which may cause unexpected behavior or security issues when the proxy attempts to delegate calls to it.

Recommendation

Add `_disableInitializers()` call in the constructor to prevent the implementation contract from being initialized:

```
constructor(
    address _boardImplementation,
    address _poolSwapRouter,
    address _protocolOwner
)
    Ownable(_protocolOwner)
{
    if (_boardImplementation == address(0)) revert InvalidAddress();
```

```
9       if (_poolSwapRouter == address(0)) revert InvalidAddress();
10      if (_protocolOwner == address(0)) revert InvalidAddress();
11
12      boardImplementation = _boardImplementation;
13      verifiedRouters[_poolSwapRouter] = true;
14
15      // Disable initializers for the implementation contract
16      _disableInitializers();
17  }
```

Status

Acknowledged

Response: `BoardManager` is not going to be used as an upgradeable contract.

Low 10 Missing Storage Gap

Location

BoardManager.sol

Description

The `BoardManager` contract inherits from `Initializable` and uses the `initializer` pattern, which indicates it's designed to be used with proxy contracts and potentially upgraded. However, the contract does not include a storage gap (`__gap`) variable at the end of its storage layout.

Storage gaps are crucial for upgradeable contracts as they reserve storage slots for future variables in contract upgrades. Without a storage gap, adding new state variables in future versions could cause storage slot collisions with child contracts that inherit from `BoardManager`, leading to data corruption or unexpected behavior.

The contract defines multiple state variables:

- `nextBoardId`
- `museManager`, `poolManager`, `phiAttester`, `phiEthFactory`
- `boardHook`, `boardImplementation`

- Various mappings (`boardAddress`, `creator`, `boardToPhiEth`, `verifiedRouters`, `boardCreated`)

Recommendation

Add a storage gap at the end of the contract to reserve storage slots for future upgrades:

```
contract BoardManager is IBoardManager, Initializable, Ownable2Step {
    // ... existing state variables ...

    /**
     * @dev This empty reserved space is put in place to allow future versions to add new
     * variables without shifting down storage in the inheritance chain.
     * See https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
     */
    uint256[50] private __gap;
}
```

The gap size of 50 slots is a common practice that provides sufficient room for future variables while being conservative with storage usage.

Status

Acknowledged

Response: `BoardManager` is not going to be used as an upgradeable contract.

Low 11 Non Compliance With EIP165 For `ISemver` Interface

Location

Board.sol

Description

Although the `Board` contract includes the `version()` function defined by `ISemver`, it does not properly declare support for the interface using the EIP-165 `supportsInterface` mechanism.

As a result, external contracts or tools that rely on EIP-165 to detect support for `ISemver` will incorrectly determine that the `Board` contract does not implement the interface.

Recommendation

Ensure EIP-165 compliance by updating the `supportsInterface` function as follows:

```
function supportsInterface(bytes4 _interfaceId) public view virtual override returns (bool) {
    return
    // Base token interfaces
    (
        _interfaceId == type(IERC20).interfaceId
        // Permit interface
        || _interfaceId == type(IERC20Permit).interfaceId
        // Superchain interfaces
        || _interfaceId == type(IERC7802).interfaceId || _interfaceId == type(IERC165).interfaceId
            || _interfaceId == type(IERC7572).interfaceId
        // Board interface
        || _interfaceId == type(IBoard).interfaceId
        // ISemver interface
        || _interfaceId == type(ISemver).interfaceId
    );
}
```

This explicitly declares support for the `ISemver` interface.

Status

Fixed

Low 12 Unrestricted Emission Of BoardTransfer Events

Location

BoardHook.sol – in the `emitBoardTransfer()` function

Description

The `emitBoardTransfer` function allows any contract or user to emit a `BoardTransfer` event by passing itself as `_board`, as long as it passes the check `msg.sender == _board`.

This design opens the door for any contract that mimics the `IBoard` interface (i.e., has a `balanceOf` function) to emit seemingly valid `BoardTransfer` events with arbitrary data. These events could be used maliciously to spoof board-related activity in off-chain systems (e.g., indexers, analytics, bots), even though no actual token transfer occurred.

Since event logs are often relied upon for state indexing and off-chain decisions, this introduces a data integrity risk.

Recommendation

Restrict access to `emitBoardTransfer` to only known and trusted board contracts. Options include:

Registry-based approach:

Maintain a list of authorized board contracts and check against it:

```
if (!authorizedBoards[msg.sender]) {
    revert UnauthorizedBoard();
}
```

Status

Fixed

Low 13 User Can Buy All Fair Launch Tokens

Location

FairLaunch.sol

Description

During the Fair Launch, users can buy tokens at a fixed price. However, there is no limitation on how many tokens one user can purchase. This allows a single user to buy all fair launch tokens, gaining access to a significant portion of the total token supply and undermining the fairness principle of the launch mechanism.

Impact: This vulnerability enables token monopolization, defeats the purpose of a “fair” launch, and could lead to price manipulation post-launch.

Recommendation

Implement per-user purchase limits to ensure equitable token distribution:

1. Add maximum purchase cap per address:

```
mapping(address => uint256) public userPurchases;
uint256 public maxPurchasePerUser;
```

2. Enforce limits in purchase function:

```
require(userPurchases[msg.sender] + amount ⩽ maxPurchasePerUser, "
    Exceeds purchase limit");
```

3. Consider additional measures:

- Whitelist mechanism for early participants
- Anti-sybil protections to prevent multiple wallet exploitation

Status

Fixed

Low 14 Missing Check If All Fess Are Less Than 100 Percent

Location

FeeValidationLib.sol – in the `validateFeeDistribution()` function

Description

The `validateFeeDistribution` function checks individual fee components (`baseSwapFee`, `referrer`, `protocol`) for validity against maximum thresholds. However, it does not verify that the sum of all fee components is less than or equal to 100% (`ONE_HUNDRED_PERCENT`).

Without this check, the combined fees could exceed 100%, which would lead to incorrect fee distribution and malfunctioning of the protocol (e.g., overcharging users or arithmetic overflows).

Recommendation

Add a check to ensure the sum of all fee components does not exceed `ONE_HUNDRED_PERCENT`:

```
if (
    uint256(_feeDistribution.baseSwapFee) +
    uint256(_feeDistribution.referrer) +
    uint256(_feeDistribution.protocol) > ONE_HUNDRED_PERCENT
) {
    revert FeeDistributionInvalid(); // Add a new error for clarity
}
```

Also declare the missing custom error:

```
error FeeDistributionInvalid();
```

Status

Fixed

Low 15 EIP7572 Is In Draft Version But Is Used

Location

Board.sol

Description

The `Board` contract implements the `contractURI()` function as defined in EIP-7572, which is currently in draft status and not finalized.

Using an unfinalized standard introduces the risk of future incompatibility if the standard changes before or during finalization. This may impact external integrations, tooling, or contract upgrades that assume conformance with finalized EIPs.

Recommendation

Do the following steps:

1. Document the dependency on EIP-7572 and track changes to the draft.
2. Consider deferring implementation of draft EIPs in production contracts or encapsulating them behind a versioning abstraction layer to isolate potential changes.

Status

Acknowledged

Response: `BoardManager` can change future board implementation using `setBoardImplementation`

Low 16 Insufficient Validation In Callback Accounting Logic

Location

PoolSwap.sol:112

Description

The `unlockCallback` function in `PoolSwap.sol` performs insufficient validation of token deltas after a swap, leading to several possibilities:

- Free Token Extraction: In an exact output swap, the function allows the input delta (`deltaAfter0`) to be zero or negative. If the pool mistakenly takes zero input tokens but provides output tokens, the check (`deltaAfter0 ⩽ 0`) passes, enabling users to extract tokens from the pool without providing any input.
- Zero-Output Swap: In an exact input swap, the function only checks that the output delta (`deltaAfter1`) is non-negative. If the swap returns zero output tokens (e.g., due to minimal input or a pool malfunction), the check (`deltaAfter1 ⩾ 0`) passes, allowing swaps that consume input tokens but return nothing to the user.

These flawed checks can result in extracting free tokens or losing tokens with no compensation, depending on the swap direction and pool behavior.

Proof of Concept

1. Add the following mock to `tests/mocks` folder:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.29;

import { BalanceDelta } from "@uniswap/v4-core/src/types/BalanceDelta.sol";
import { Currency } from "@uniswap/v4-core/src/types/Currency.sol";
import { Hooks, IHooks } from "@uniswap/v4-core/src/libraries/Hooks.sol";
import { IPoolManager } from "@uniswap/v4-core/src/interfaces/IPoolManager.sol";

import { IUnlockCallback } from "@uniswap/v4-core/src/interfaces/callback/IUnlockCallback.sol";
import { PoolKey } from "@uniswap/v4-core/src/types/PoolKey.sol";
import { TransientStateLibrary } from "@uniswap/v4-core/src/libraries/TransientStateLibrary.sol";
import { IBoardHook } from "../../src/interfaces/IBoardHook.sol";
import { CurrencySettler } from "../../src/libraries/CurrencySettler.sol";
import { Locker } from "../../src/libraries/Locker.sol";

```

```
16 contract PoolSwapMock is IUnlockCallback {
17     mapping(Currency => int256) public deltas;
18 
19     using CurrencySettler for Currency;
20     using Hooks for IHooks;
21     using TransientStateLibrary for IPoolManager;
22 
23     /**
24      * Stores information to be passed back when unlocking the
       callback.
25      *
26      * @member sender The sender of the swap
27      * @member key The poolKey being swapped against
28      * @member params Swap parameters
29      * @member referrer An optional referrer
30      */
31     struct CallbackData {
32         address sender;
33         PoolKey key;
34         IPoolManager.SwapParams params;
35         address referrer;
36         bytes32[] proof;
37         bytes leafPart;
38     }
39 
40     function setDelta(Currency currency, int256 delta) external {
41         deltas[currency] = delta;
42     }
43 
44     /**
45      * Performs the swap call using information from the CallbackData.
46      */
47     function unlockCallback(bytes calldata rawData) external returns (
       bytes memory) {
48         // Decode our CallbackData
49         CallbackData memory data = abi.decode(rawData, (CallbackData))
       ;
50 
51         // Skipped concrete `swap` logic for the sake of the mock
52 
53         int256 deltaAfter0 = deltas[data.key.currency0];
54         int256 deltaAfter1 = deltas[data.key.currency1];
55 
56         if (data.params.zeroForOne) {
57             if (data.params.amountSpecified < 0) {
58                 // exact input, 0 for 1
59                 require(
```

```
                    deltaAfter0 ⩾ data.params.amountSpecified,
                    "deltaAfter0 is not greater than or equal to data.
    params.amountSpecified"
                );
                require(deltaAfter1 ⩾ 0, "deltaAfter1 is not greater
    than or equal to 0");
            } else {
                // exact output, 0 for 1
                require(deltaAfter0 ⩽ 0, "deltaAfter0 is not less
    than or equal to zero");
                require(
                    deltaAfter1 ⩽ data.params.amountSpecified,
                    "deltaAfter1 is not less than or equal to data.
    params.amountSpecified"
                );
            }
        } else {
            if (data.params.amountSpecified < 0) {
                // exact input, 1 for 0
                require(
                    deltaAfter1 ⩾ data.params.amountSpecified,
                    "deltaAfter1 is not greater than or equal to data.
    params.amountSpecified"
                );
                require(deltaAfter0 ⩾ 0, "deltaAfter0 is not greater
    than or equal to 0");
            } else {
                // exact output, 1 for 0
                require(deltaAfter1 ⩽ 0, "deltaAfter1 is not less
    than or equal to 0");
                require(
                    deltaAfter0 ⩽ data.params.amountSpecified,
                    "deltaAfter0 is not less than or equal to data.
    params.amountSpecified"
                );
            }
        }

        // Skipped `settle` and `take` logic for the sake of the mock

        return abi.encode("");
    }
}
```

2. Add the following file in the `tests` folder:

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

import "forge-std/Test.sol";
import { BalanceDelta } from "@uniswap/v4-core/src/types/BalanceDelta.sol";
import { Currency } from "@uniswap/v4-core/src/types/Currency.sol";
import { IHooks } from "@uniswap/v4-core/src/libraries/Hooks.sol";
import { PoolKey } from "@uniswap/v4-core/src/types/PoolKey.sol";
import { PoolSwapMock } from "./mocks/PoolSwapMock.sol";
import { IBoardHook } from "../src/interfaces/IBoardHook.sol";
import { IPoolManager } from "@uniswap/v4-core/src/interfaces/IPoolManager.sol";

contract PoolSwapCallback is Test {
    PoolSwapMock public poolSwapMock;
    Currency public currency0;
    Currency public currency1;
    PoolKey public poolKey;

    function setUp() public {
        poolSwapMock = new PoolSwapMock();

        // Setup dummy currencies and pool key
        currency0 = Currency.wrap(address(0x1000));
        currency1 = Currency.wrap(address(0x2000));
        poolKey = PoolKey({
            currency0: currency0,
            currency1: currency1,
            fee: 3000,
            hooks: IHooks(address(0)),
            tickSpacing: 60
        });
    }

    // Test 1: Exact Input Swap with Zero Output Delta
    /*
        In an exact input swap, the contract checks deltaAfter1 ⩾ 0. If the swap returns 0 output tokens
        (e.g., due to minimal input), the condition passes, allowing swaps that consume input tokens but return nothing.
    */
    function test_ExactInputZeroOutputDelta() public {
        // Setup exact input swap (negative amountSpecified)
        IPoolManager.SwapParams memory params = IPoolManager.SwapParams({
            zeroForOne: true, // token0 -> token1
```

```
43              amountSpecified: -100, // Spend exactly 100 token0
44              sqrtPriceLimitX96: 0
45          });
46
47          // Setup callback data
48          PoolSwapMock.CallbackData memory callbackData = PoolSwapMock.
    CallbackData({
49              sender: address(this),
50              key: poolKey,
51              params: params,
52              referrer: address(0),
53              proof: new bytes32[](0),
54              leafPart: ""
55          });
56          bytes memory rawData = abi.encode(callbackData);
57
58          // Mock currencyDelta after swap to return 0 for output token
59          poolSwapMock.setDelta(currency0, -100);
60          poolSwapMock.setDelta(currency1, 0);
61
62          // Should NOT revert despite 0 output delta
63          poolSwapMock.unlockCallback(rawData);
64      }
65
66      // Test 2: Free Token Extraction Swap
67      /*
68          In an exact output swap, the contract accepts deltaAfter0 ⩽
    0. If the pool mistakenly takes 0 input tokens
69          while providing output tokens, the condition passes, allowing
    users to steal tokens from the pool without cost.
70      */
71      function test_FreeTokenExtractionSwap() public {
72          // Setup exact output swap (positive amountSpecified)
73          IPoolManager.SwapParams memory params = IPoolManager.
    SwapParams({
74              zeroForOne: true, // token0 -> token1
75              amountSpecified: 100, // Receive exactly 100 token0
76              sqrtPriceLimitX96: 0
77          });
78
79          // Setup callback data
80          PoolSwapMock.CallbackData memory callbackData = PoolSwapMock.
    CallbackData({
81              sender: address(this),
82              key: poolKey,
83              params: params,
84              referrer: address(0),
```

```
            proof: new bytes32[](0),
            leafPart: ""
        });
        bytes memory rawData = abi.encode(callbackData);

        // Mock currencyDelta after swap to return 0 for input token
        poolSwapMock.setDelta(currency0, 0);
        poolSwapMock.setDelta(currency1, 100);

        // Should NOT revert despite 0 input delta
        poolSwapMock.unlockCallback(rawData);
    }
}
```

Recommendation

Implement stricter validation to ensure that:

- For exact output swaps, the contract must verify that the input amount is strictly negative (i.e., tokens were actually provided).
- For exact input swaps, the contract must verify that the output amount is strictly positive (i.e., the user receives tokens in return for their input).

Status

Acknowledged

Informational

Info 01 Incrementing Epoch Could Be Automated

Location

MuseManager.sol:93

Description

The `incrementEpoch` function in the `MuseManager` contract is responsible for advancing the global epoch, which is used for inspiration tracking and contributor statis-

tics. Currently, this function can only be called manually by the contract owner. This manual process introduces operational risk: if the owner forgets or delays calling `incrementEpoch`, the epoch will not advance as intended. This can result in outdated contributor rankings, inaccurate inspiration tracking, and a poor user experience, as epoch-based rewards or logic may not update on schedule. Additionally, relying on a single privileged actor for regular maintenance reduces decentralization and increases the risk of human error or malicious inaction.

Recommendation

Integrate an automated mechanism for incrementing the epoch. This can be achieved by using a decentralized off-chain automation service (a "keeper"), such as Chainlink Keepers or Gelato, to periodically call `incrementEpoch` according to a predefined schedule (e.g., daily, weekly, or based on block intervals). This approach ensures epochs advance reliably and on time, reduces reliance on manual intervention, and improves the protocol's decentralization and robustness.

Status

Acknowledged

Response: Admin controls epoch term. No need to implement auto increment.

Info 02 `ACTIVE_WALLET_MIN_BALANCE` May Become Obsolete

Location

MuseManager.sol

Description

The `ACTIVE_WALLET_MIN_BALANCE` constant in the `MuseManager` contract is hard-coded to `0.005 ether`. This value is used to determine whether a wallet is considered "active" for inspiration tracking and contributor eligibility. However, the fixed threshold may become inadequate over time due to fluctuations in the price of ETH or changes in network conditions. If the value of ETH increases

significantly, the threshold could become too high, excluding legitimate users. Conversely, if ETH's value drops, the threshold could become too low, allowing inactive or spam wallets to qualify as active. This lack of adaptability can reduce the effectiveness and fairness of the protocol's contributor tracking.

Recommendation

Make the `ACTIVE_WALLET_MIN_BALANCE` configurable by the contract owner or governed by a DAO, allowing it to be updated as market conditions change. Alternatively, consider integrating an oracle to dynamically adjust the threshold based on the current USD value of ETH or other relevant metrics. This will ensure the active wallet criteria remain relevant and effective over time, regardless of ETH price volatility or network upgrades.

Status

Acknowledged

Info 03 `changeStrategy` Griefing Attack

Location

PHIETHImpl.sol:316

Description

In the `changeStrategy` function of the `PhiEthImpl` contract, a check is performed to ensure that the current strategy's ETH balance is exactly zero before allowing a strategy change. However, this check is vulnerable to griefing by malicious actors. For example, with Aave, anyone can transfer a minimal amount of aWETH (such as 1 wei) to the strategy contract, or with Morpho, anyone can deposit and mint a tiny amount of shares to the strategy. This causes the `balanceInETH()` check to fail, preventing the contract owner from ever changing the strategy. As a result, the protocol can be locked into a particular strategy indefinitely, even if the balance is negligible and operationally irrelevant.

Recommendation

Introduce a small threshold value (e.g., a few wei or a configurable minimal amount) when checking the strategy's balance before allowing a strategy change. Instead of requiring the balance to be exactly zero, allow the strategy change if the balance is less than or equal to this threshold. This approach prevents griefing attacks with dust amounts while still protecting against significant unclaimed funds being left in the old strategy. Make sure to document and, if possible, allow governance or the owner to adjust this threshold as needed.

Status

Fixed

Info 04 FairLaunch Can Move Definitions To Interface

Location

FairLaunch.sol

Description

The `FairLaunch` contract currently defines its structs, custom errors, events, and function signatures directly within the contract itself. This approach can lead to a less modular and harder-to-maintain codebase, especially as the protocol grows or if multiple contracts need to interact with or inherit from `FairLaunch`. Keeping these definitions inside the contract makes it more difficult for other contracts or external tools to reference or implement the same interfaces, and can result in code duplication or inconsistencies if similar logic is needed elsewhere.

Recommendation

Refactor the `FairLaunch` contract by moving its structs, custom errors, events, and function signatures into a dedicated interface file (e.g., `IFairLaunch.sol`). This interface should define all relevant types and function signatures that external contracts or inheritors might need. The main `FairLaunch` contract should then implement this interface. This change will improve code clarity, promote reuse,

and make it easier for other contracts to interact with or extend the fair launch logic in a consistent and type-safe manner.

Status

Fixed

Info 05 Wrong Revert Reason On _buyTokens

Location

PremineZap.sol:111

Description

In the `PremineZap` contract's `_buyTokens` function, after performing a swap, the code calculates `remainingETH` as `_ethAmount - phiEthSwapped_`. However, if `phiEthSwapped_` is greater than `_ethAmount` (which can occur due to slippage or unexpected pool behavior), this subtraction will underflow and revert with a generic arithmetic error. This does not provide a clear or user-friendly explanation for the failure, making it difficult for users and integrators to understand that the transaction failed due to excessive slippage or an unfavorable swap rate. The lack of a specific revert reason for this scenario reduces transparency and can hinder debugging or automated monitoring.

Recommendation

Add an explicit check after the swap to ensure that `phiEthSwapped_` does not exceed `_ethAmount`. If this condition is violated, revert with a custom error that clearly indicates the cause (e.g., `SlippageExceeded()` or `SwapAmountExceedsInput()`). This will provide a clear and descriptive revert reason, improving the user experience and making it easier to diagnose and handle slippage-related failures in integrations and frontends. For example:

```
error SlippageExceeded();

...

```

```
5 if (phiEthSwapped_ > _ethAmount) {
6     revert SlippageExceeded();
7 }
```

Status

Fixed

Info 06 Slippage Is Not Configurable And May Be Not Enough

Location

AerodromeSwapper.sol:73

Description

The `AerodromeSwapper` contract currently uses a fixed 1% slippage tolerance when performing swaps (`minAmountOut = expectedAmounts[1] * 99 / 100`). While this may be sufficient during periods of low volatility and high liquidity, it can be too restrictive during periods of high market volatility or low liquidity. If the price moves more than 1% between the time the transaction is submitted and when it is mined, the swap will fail, causing user transactions to revert unnecessarily. This can lead to a poor user experience, especially during volatile market conditions, and may prevent users from executing swaps when they need them most.

Recommendation

Allow users to specify their own slippage tolerance as a parameter when calling the `withdrawAndSwap` function, or provide a way for the contract owner to adjust the default slippage tolerance. This flexibility will enable users to set a higher tolerance during volatile periods or for illiquid pairs, reducing the likelihood of failed transactions. If user input is not feasible, consider increasing the default slippage tolerance to a more conservative value (e.g., 2-3%) to better accommodate market fluctuations.

Status

Fixed

Info 07 Missing Event Emission On Failed `tryInspire` Call

Location

Board.sol:283

Description

In the `Board` contract, the `_update` function calls `museManager.tryInspire(from, to, amount)` to attempt to record an inspiration event when tokens are transferred. However, if `tryInspire` returns `false` (indicating that the inspiration conditions were not met or the inspiration was not recorded), there is currently no event emitted to signal this outcome. As a result, off-chain services, analytics platforms, and users have no visibility into failed or skipped inspiration attempts. This lack of transparency can make it difficult to debug issues, audit protocol behavior, or provide accurate user feedback regarding why an inspiration did not occur.

Recommendation

Emit a dedicated event whenever `tryInspire` returns `false`, indicating that an inspiration attempt was made but did not succeed. The event should include relevant details such as the sender, recipient, amount, and possibly the reason for failure if available. For example:

```
event InspirationAttemptFailed(address indexed from, address indexed to, uint256 amount);

function _update(address from, address to, uint256 amount) internal override {
    super._update(from, to, amount);

    if (museManager.shouldInspire(from, amount)) {
        bool inspired = museManager.tryInspire(from, to, amount);
        if (!inspired) {
```

```
                emit InspirationAttemptFailed(from, to, amount);
            }
        }
    }
```

This will improve transparency, facilitate monitoring, and help users and developers understand when and why inspiration events do not take place.

Status

Fixed

Info 08 `withdrawFees` Has A Missing Zero Address Check For `_recipient`

Location

FeeDistributor.sol — in the `withdrawFees()` function

Description

The `withdrawFees()` function accepts a `_recipient` parameter but does not validate that it is not the zero address. This could result in fees being sent to the zero address, effectively burning the tokens.

Recommendation

Add a zero address check:

```
if (_recipient == address(0)) revert InvalidAddress();
```

Status

Fixed

Info 09 `claimTokens` Has A Missing Zero Address Check For `_recipient`

Location

ReferralEscrow.sol — in the `claimTokens()` function

Description

The `claimTokens()` function accepts a `_recipient` parameter but does not validate that it is not the zero address. This could result in tokens being sent to the zero address, effectively burning them.

Recommendation

Add a zero address check:

```
if (_recipient == address(0)) revert InvalidAddress();
```

Status

Fixed

Info 10 Board Hook Missing Zero Address Check

Location

BoardHook.sol — `setPremineZap()`, `setMuseManager()`, `setInitialPrice()` functions

Description

The `setPremineZap()`, `setInitialPrice()` and `setMuseManager()` functions accept address parameters but do not validate that they are not the zero address. This could result in critical contract functionality being disabled if zero addresses are accidentally set.

Recommendation

Add zero address checks for address parameters:

```
function setPremineZap(address _premineZap) external onlyOwner {
    if (_premineZap == address(0)) revert InvalidAddress();
    premineZap = _premineZap;
}

function setInitialPrice(address _initialPrice) public onlyOwner {
    if (_initialPrice == address(0)) revert InvalidAddress();
    initialPrice = IInitialPrice(_initialPrice);
}

function setMuseManager(address _museManager) external onlyOwner {
    if (_museManager == address(0)) revert InvalidAddress();
    museManager = IMuseManager(_museManager);
}
```

Status

Acknowledged

Info 11 Board Hook Missing Events

Location

BoardHook.sol — `setPremineZap()`, `setInitialPrice()`, `setMuseManager()` functions

Description

The following functions modify critical contract state variables but do not emit events:

```
function setPremineZap(address _premineZap) external onlyOwner {
    premineZap = _premineZap;
}

function setInitialPrice(address _initialPrice) public onlyOwner {
    initialPrice = IInitialPrice(_initialPrice);
}
```

```
8
9  function setMuseManager(address _museManager) external onlyOwner {
10     museManager = IMuseManager(_museManager);
11 }
```

This reduces transparency and makes it difficult to track important configuration changes off-chain.

Recommendation

Add event emissions for better transparency:

```
1  event PremineZapUpdated(address indexed previousZap, address indexed
       newZap);
2  event InitialPriceUpdated(address indexed previousPrice, address
       indexed newPrice);
3  event MuseManagerUpdated(address indexed previousManager, address
       indexed newManager);
4
5  function setPremineZap(address _premineZap) external onlyOwner {
6      address oldZap = premineZap;
7      premineZap = _premineZap;
8      emit PremineZapUpdated(oldZap, _premineZap);
9  }
10
11 function setInitialPrice(address _initialPrice) public onlyOwner {
12     address oldPrice = address(initialPrice);
13     initialPrice = IInitialPrice(_initialPrice);
14     emit InitialPriceUpdated(oldPrice, _initialPrice);
15 }
16
17 function setMuseManager(address _museManager) external onlyOwner {
18     address oldManager = address(museManager);
19     museManager = IMuseManager(_museManager);
20     emit MuseManagerUpdated(oldManager, _museManager);
21 }
```

Status

Acknowledged

Info 12 Muse Manager Missing Excluded Address Events

Location

MuseManager.sol — `addExcludedAddress()`, `removeExcludedAddress()` functions

Description

The following functions modify the excluded addresses mapping but do not emit events:

```
function addExcludedAddress(address account) external onlyOwner {
    excludedAddresses[account] = true;
}

function removeExcludedAddress(address account) external onlyOwner {
    excludedAddresses[account] = false;
}
```

This reduces transparency and makes it difficult to track changes to excluded addresses off-chain.

Recommendation

Add event emissions for better transparency:

```
event AddressExcluded(address indexed account);
event AddressIncluded(address indexed account);

function addExcludedAddress(address account) external onlyOwner {
    excludedAddresses[account] = true;
    emit AddressExcluded(account);
}

function removeExcludedAddress(address account) external onlyOwner {
    excludedAddresses[account] = false;
    emit AddressIncluded(account);
}
```

Status

Fixed

Info 13 Muse Manager Wrong `tryInspire` Triggers

Location

Board.sol — in the `_update()` function override

Description

The `_update()` function override calls `museManager.tryInspire()` for all token operations, including mints and burns:

```
function _update(address from, address to, uint256 amount) internal
    override {
     super._update(from, to, amount);

     if (museManager.shouldInspire(from, amount)) {
         museManager.tryInspire(from, to, amount);
     }
}
```

This means `tryInspire` will be triggered for:

- **Mints:** `from = address(0), to = recipient`
- **Burns:** `from = account, to = address(0)`
- **Transfers:** `from = sender, to = recipient`

This may result in unintended inspiration calculations for mint/burn operations that were not originally designed to trigger inspiration mechanics.

Recommendation

Consider adding operation type checks to ensure `tryInspire` only executes for intended operations:

```
function _update(address from, address to, uint256 amount) internal
    override {
     super._update(from, to, amount);

     // Only inspire on actual transfers, not mints/burns
     if (from ≠ address(0) && to ≠ address(0) && museManager.
    shouldInspire(from, amount)) {
         museManager.tryInspire(from, to, amount);
     }
}
```

Status

Fixed

Info 14 No Check If Fair Launch Is Enabled

Location

BoardHook.sol – in the `publiphi()` function

Description

The `BoardHook` contains logic that attempts to interact with the `FairLaunch` contract, including calls to `FairLaunch::createPosition()` and potentially `FairLaunch::closePosition()` during swap handling. However, there is no explicit check to determine whether the fair launch is initialized for a pool.

If the fair launch is not enabled, the function still proceeds with checking fair launch info, attempting to close positions, and processing fair launch swaps. This can lead to unintended behavior.

Recommendation

Introduce a check if the fair launch is initialized for a pool and if it's not just skip.

Status

Fixed

Info 15 `phiFactory` Is Set Two Times

Location

FeeDistributor.sol

BoardHook.sol

Description

The `phiFactory` variable is initialized in both `FeeDistributor` and its child contract `BoardHook`. Since `BoardHook` inherits from `FeeDistributor`, redeclaring and setting `phiFactory` again in `BoardHook` is redundant and may lead to confusion or unintentional bugs. However, `phiFactory` cannot be removed from `FeeDistributor` as it is required there, so it should not be re-set or shadowed in `BoardHook`.

Recommendation

Remove the reassignment or redundant declaration of `phiFactory` in `BoardHook`.

Status

Fixed

Info 16 Change Function Visibility From Public To External

Location

BoardManager.sol – in the `addRouter()` function

BoardManager.sol – in the `removeRouter()` function

FairLaunch.sol – in the `inFairLaunchWindow()` function

FairLaunch.sol – in the `fairLaunchInfo()` function

FairLaunch.sol – in the `closePosition()` function

FairLaunch.sol – in the `fillFromPosition()` function

FairLaunch.sol – in the `modifyRevenue()` function

FairLaunch.sol – in the `isWhitelisted()` function

BoardHook.sol – in the `getPubliphiFee()` function

BoardHook.sol – in the `getPubliphiMarketCap()` function

BoardHook.sol – in the `setInitialPrice()` function

PhiEthFactory.sol – in the `updateDeploymentMappings()` function

PhiEthFactory.sol – in the `setProtocolFeeRecipient()` function

PhiEthFactory.sol – in the `setPoolSwapZap()` function

Board.sol – in the `initialize()` function

Board.sol – in the `tokenURI()` function

Board.sol – in the `uri()` function

AerodromeSwapper.sol – in the `setSwapConfig()` function

PoolSwap.sol – in the first instance of the `swap()` function

PremineZap.sol – in the `calculateFee()` function

Description

Multiple functions across the codebase are marked as `public` but are not used internally within their respective contracts. Functions marked as `public` consume more gas than `external` functions because they need to copy arguments from calldata to memory to handle both internal and external calls. When functions are only called externally, marking them as `external` provides gas savings.

Gas Impact: Each `public` function that should be `external` wastes approximately 20-40 gas per call due to unnecessary memory allocation and copying operations.

Recommendation

Change the visibility modifier from `public` to `external` for all functions that are not used internally:

```
// Before
function addRouter(address router) public onlyOwner {
    // function body
}

```

```
// After
function addRouter(address router) external onlyOwner {
    // function body
}
```

Status

Fixed

Info 17 Consider Adding A Check To Ensure Only The `PhiEth` Contract Can Send Ether

Location

ReferralEscrow.sol – in the `receive()` function

PremineZap.sol – in the `receive()` function

AerodromeSwapper.sol – in the `receive()` function

FeeDistributor.sol – in the `receive()` function

Description

These contracts include a `receive()` function that accepts ETH, but they currently lack validation on the sender.

If this is overlooked it can lead to locked ETH in the given contracts.

Recommendation

Add a `require()` check to each `receive()` function to ensure that only the `PhiEth` contract can send ETH:

```
receive() external payable {
    require(msg.sender == address(phiEth), "Unauthorized ETH sender");
}
```

Status

Acknowledged

Info 18 Missing Event Emission In PhiETHImpl Critical State Updates

Location

PhiETHImpl.sol — in the `setRebalanceThreshold()` function

PhiETHImpl.sol — in the `setYieldReceiver()` function

Description

The `setRebalanceThreshold()` and `setYieldReceiver()` functions in `PhiETHImpl` are missing event emissions after updating critical contract state variables.

Recommendation

Add appropriate event emissions to both functions:

1. Define events at the contract level:

```
event RebalanceThresholdUpdated(uint256 oldThreshold, uint256
    newThreshold);
event YieldReceiverUpdated(address oldReceiver, address newReceiver);
```

2. Emit events in the respective functions:

```
function setRebalanceThreshold(uint256 rebalanceThreshold_) external
    override onlyCreator {
    if (rebalanceThreshold_ > MAX_REBALANCE_THRESHOLD) revert
    RebalanceThresholdExceedsMax();
    uint256 oldThreshold = rebalanceThreshold;
    rebalanceThreshold = rebalanceThreshold_;
    emit RebalanceThresholdUpdated(oldThreshold, rebalanceThreshold_);
}

function setYieldReceiver(address yieldReceiver_) external override
    onlyCreator {
    if (yieldReceiver_ == address(0)) revert YieldReceiverIsZero();
    address oldReceiver = yieldReceiver;
    yieldReceiver = yieldReceiver_;
    emit YieldReceiverUpdated(oldReceiver, yieldReceiver_);
}
```

Status

Fixed

Info 19 `validateFeeDistribution` Has Wrong Comment Above The `_feeDistribution.protocol` Validation

Location

FeeValidationLib.sol – in the `validateFeeDistribution()` function

Description

The inline comment above the `_feeDistribution.protocol` validation is inaccurate. It currently reads:

```
// Ensure our protocol fee is below 10%
```

However, the actual constant `MAX_PROTOCOL_ALLOCATION` is defined as `5_000`, which corresponds to 50%, not 10%.

Recommendation

Update the comment to correctly reflect the maximum protocol allocation threshold:

```
// Ensure our protocol fee is below 50%
```

Status

Fixed

Info 20 State Variables Can Be Marked Immutable

Location

BoardHook.sol

Description

The state variables `notifier`, `fairLaunch`, `feeExemptions`, and `boardManager` are only assigned once during the contract's constructor and are never modified afterward. These variables are ideal candidates for the `immutable` keyword. Using `immutable` can optimize gas usage and improve contract safety by ensuring these values cannot be changed after deployment.

Recommendation

Declare the variables as `immutable`:

```
address public immutable notifier;
address public immutable fairLaunch;
address public immutable feeExemptions;
address public immutable boardManager;
```

Ensure the assignments remain within the constructor.

Status

Acknowledged

Response: Using immutable variables in Solidity increases contract size because their values are directly embedded into the bytecode, no fix can be implemented because `BoardHook` contract size will become more than 24KB.