Hypersurface Audit Report

Version: v1.0

November 7th, 2025

Hypersurface Protocol - Security Audit Report

Audit Date: 13/10/25 - 17/10/25

Auditor: Simply Staking

Repositories:

- https://github.com/hypersurface-io/contracts/tree/audit
- https://github.com/hypersurface-io/GammaProtocol-original/pull/8

Documentation: https://docs.hypersurface.io/

Table of Contents

Executive Summary

This report presents the findings from a comprehensive security audit of the Hypersurface Protocol smart contracts and forked Gamma Protocol. The following commits were audited:

- **Forked Gamma Protocol:** 694996b4ca15dfb1507df00c5cc781f004410a6d
- **Hypersurface Smart Contracts:** bdb44c2c85ce2f55788f085fe85a12db5d284240

Code Quality

- **Readability**: Medium
  - Code was readable but could use further decoupling of functionality for easier understanding
- **Level of Documentation**: Low
  - No Architecture and System documentation
  - More inline code documentation was expected.
- **Test Coverage**: Medium - High
  - *Statements*: 77.84%
  - *Branches*: 60.34%
  - *Functions*: 64.98%
  - *Lines*: 78.51%

Key Statistics

- **Critical Issues:** 0
- **Major Issues:** 5
- **Minor Issues:** 20
- **Informational Issues:** 11

Severity Classification

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary.

Major Findings

M-01: Unbounded Loop in setHedger() Can Cause Out of Gas

Severity: Major
Contract: `HedgedPool.sol`
Lines: 746-765
Status: Acknowledged. The client states that the `setHedger()` function will only be called during initial deployment and the early stages when the number of underlying tokens is small. Furthermore, the client states that since Hedgers are upgradeable it is very unlikely that the Hedger address will be changed to a different contract address.

Description:

The `setHedger()` function loops through all `underlyingTokens` to approve/unapprove the hedger. If the number of underlying tokens grows too large, this function will exceed the block gas limit and become unexecutable.

Affected Code:

`HedgedPool.sol` - Lines 746-765:

```
/// @notice Enable/disable a hedger
function setHedger(address newHedgerAddress) external onlyOwner {
    if (hedger != address(0) && hedger != newHedgerAddress) {
        // remove old approvals
        for (uint i = 0; i < underlyingTokens.length(); i++) {  // ←
Unbounded loop
            IERC20(underlyingTokens.at(i)).approve(hedger, 0);
        }
        collateralToken.approve(hedger, 0);
    }

    hedger = newHedgerAddress;

    collateralToken.approve(newHedgerAddress, type(uint256).max);
    for (uint i = 0; i < underlyingTokens.length(); i++) {
        IERC20(underlyingTokens.at(i)).approve(
            newHedgerAddress,
            type(uint256).max
        );
    }
    emit HedgerSet(newHedgerAddress);
}
```

If `underlyingTokens.length()` > becomes too large, function becomes unexecutable causing hedger to not be updated.

Recommendation:

Break down into smaller functions:

```
function setHedgerAddress(address newHedgerAddress) external onlyOwner {
    hedger = newHedgerAddress;
    emit HedgerSet(newHedgerAddress);
}

function approveHedgerForToken(address token, bool approve) external
onlyOwner {
    uint256 amount = approve ? type(uint256).max : 0;
    IERC20(token).approve(hedger, amount);
}
```

M-02: Missing slippage deadline in UniswapOrdersLib market functions

Severity: Major
Contract: `UniswapOrdersLib.sol`
Lines: 501-528, 530-565
Status: Resolved

Description:

The `marketBuy()` and `marketSell()` functions in UniswapOrdersLib don't include a `deadline` parameter in the Uniswap swap params. This allows transactions to be stuck in the mempool and executed at unfavorable prices.

Affected Code:

`UniswapOrdersLib.sol` - Lines 501-528, 530-565:

```
ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
    .ExactInputSingleParams({
        tokenIn: tokenIn,
        tokenOut: tokenOut,
        fee: fee,
        recipient: recipient,
        deadline: block.timestamp,
        amountIn: amountIn,
        amountOutMinimum: amountOutMin,
        sqrtPriceLimitX96: 0
    });
```

Transactions can be executed at significantly worse prices if stuck in mempool.

Recommendation:

Add deadline parameter to all market functions:

```
function marketBuy(
    address swapRouter,
    address tokenIn,
    address tokenOut,
    uint24 fee,
    uint256 amountIn,
    uint256 amountOutMin,
    address recipient,
    uint256 deadline
) internal returns (uint256 amountOut) {
    ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
        .ExactInputSingleParams({
            tokenIn: tokenIn,
            tokenOut: tokenOut,
            fee: fee,
            recipient: recipient,
            deadline: deadline,
            amountIn: amountIn,
            amountOutMinimum: amountOutMin,
            sqrtPriceLimitX96: 0
        });
```

M-03: ecrecover Signature Malleability in OrderUtil

Severity: Major
Contract: `OrderUtil.sol`
Lines: 210-215
Status: Resolved

Description:

The `_getSignatory()` function uses `ecrecover` without validating the `v` parameter. This allows signature malleability attacks where an attacker can create a second valid signature for the same message.

Affected Code:

`OrderUtil.sol` - Lines 210-215:

```
function _getSignatory(
    bytes32 hash,
    Signature calldata signature
) internal pure returns (address) {
    return ecrecover(hash, signature.v, signature.r, signature.s);
}
```

The `v` parameter should be either 27 or 28. Without validation attackers can flip `v` and negate `s` to create alternate valid signatures: https://detectors.auditbase.com/signature-malleability-of-evms-ecrecover

Recommendation:

Add validation for the `v` and `s` parameters:

```
function _getSignatory(
    bytes32 hash,
    Signature calldata signature
) internal pure returns (address) {
    require(uint256(signature.s) <=
0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0,
"Invalid s parameter");
    require(signature.v == 27 || signature.v == 28, "Invalid v
parameter");

    address signatory = ecrecover(hash, signature.v, signature.r,
signature.s);
    require(signatory != address(0), "Invalid signature");

    return signatory;
}
```

M-04: Missing Order Legs Validation - Array Out of Bounds Risk

Severity: Major
Contract: `HedgedPool.sol`
Lines: 525-572, 414-424
Status: Resolved.

Description:

The `processOrder()` function validates that an order doesn't have more than one leg, but fails to check if the order has zero legs. This allows orders with empty `legs` arrays to pass validation, which will cause a Solidity panic (array out-of-bounds) when the code attempts to access `order.legs[0]`.

Affected Code:

`HedgedPool.sol` - Lines 531, 534:

```
// Allow only one leg
if (order.legs.length > 1) revert CustomErrors.TooManyLegs();

// Validate each leg
for (uint i = 0; i < order.legs.length; i++) {
```

`HedgedPool.sol` - Lines 414-424:

```
function trade(
    IOrderUtil.Order calldata order,
    uint256 traderVaultId,
    bool autoCreateVault
) public nonReentrant {
    onlyAccessKey();

    // validate that the order signer has QUOTE_PROVIDER role. The signing
contract has to return the recovered signer.
    processOrder(order);
    address[] memory oTokens;
    uint256 fee = order.legs[0].fee;
```

Recommendation:

Change the validation to explicitly check for exactly one leg:

```
// Require exactly one leg
if (order.legs.length != 1) revert CustomErrors.InvalidLegsCount();
```

M-05: Unbounded Loop in settleAll() - Denial of Service Risk

Severity: Major
Contract: `HedgedPool.sol` + `HedgedPoolLib.sol`
Lines: HedgedPool.sol:155-177
Status: Resolved.

Description:

The `settleAll()` function contains nested unbounded loops through `underlyingTokens` and `oTokensByExpiry`.

Affected Code:

`HedgedPool.sol` - Lines 155-177:

```
/**
 * @notice Settle all expired oTokens
 */
function settleAll() public {
    uint256 expiry = lastSettledExpiry + 1 days;
    uint8 numProcessed;
    while (expiry <= block.timestamp && numProcessed < 5) {
        numProcessed += 1;
        HedgedPoolLib.settleExpiry(
            expiry,
            underlyingTokens,
            oTokensByExpiry,
            controller,
            notionalExposure,
            activeOTokens
        );

        lastSettledExpiry = expiry;
```

`HedgedPoolLib.sol` - Lines 315-352 - settleExpiry (nested loops):

```
function settleExpiry(
    uint256 expiry,
    EnumerableSet.AddressSet storage underlyingTokens,
    mapping(address => mapping(uint256 => address[])) storage
oTokensByExpiry,
    address controller,
    mapping(address => mapping(bool => uint256)) storage notionalExposure,
    EnumerableSet.AddressSet storage activeOTokens
) internal {
    for (uint256 i = 0; i < underlyingTokens.length(); i++) {

        address underlying = underlyingTokens.at(i);
        address[] memory oTokens = oTokensByExpiry[underlying][expiry];

        for (uint256 j = 0; j < oTokens.length; j++) {
```

If an excessive number of oTokens accumulate, settlement may become impossible, resulting in locked funds. While this issue was not classified as critical because only the admin can create tokens, it remains a major concern since misconfiguration errors are still possible.

Recommendation:

Add a separate `onlyOwner` function with params on individual expiries to settle, this will be used in emergencies if `settleAll()` becomes unusable.

Minor Findings

MIN-01: Redundant For Loop in processOrder()

Severity: Minor
Contract: `HedgedPool.sol`
Lines: 531-533
Status: Resolved.

Description:

The code enforces that only one leg is allowed (`order.legs.length > 1` check), yet immediately uses a `for` loop to iterate through legs. This is inefficient and confusing.

Affected Code:

`HedgedPool.sol` - Lines 531-533:

```
// Allow only one leg
if (order.legs.length > 1) revert CustomErrors.TooManyLegs();

// Validate each leg
for (uint i = 0; i < order.legs.length; i++) {
    IOrderUtil.OptionLeg memory leg = order.legs[i];
    if (leg.expiration <= block.timestamp) {
        revert CustomErrors.SeriesExpired();
    }
```

Recommendation:

Since only one leg is allowed, remove the loop:

```
// Require exactly one leg
if (order.legs.length != 1) revert CustomErrors.InvalidLegsCount();

// Validate the single leg
IOrderUtil.OptionLeg memory leg = order.legs[0];
if (leg.expiration <= block.timestamp) {
```

```
        revert CustomErrors.SeriesExpired();
    }
    // ... rest of validations
```

MIN-02: Missing Event Emission in setKeeper()

Severity: Minor
Contract: `HedgedPool.sol`
Lines: 662-667
Status: Resolved.

Description:

The `setKeeper()` function modifies critical access control but doesn't emit an event, making it difficult to track keeper changes off-chain.

Affected Code:

`HedgedPool.sol` - Lines 662-667:

```
/// @notice Allow/disallow an address to perform keeper tasks
function setKeeper(
    address keeperAddress,
    bool isPermitted
) external onlyOwner {
    keepers[keeperAddress] = isPermitted;
}
```

Recommendation:

Add event emission:

```
event KeeperSet(address indexed keeper, bool isPermitted);

function setKeeper(
    address keeperAddress,
    bool isPermitted
) external onlyOwner {
    keepers[keeperAddress] = isPermitted;
    emit KeeperSet(keeperAddress, isPermitted);
}
```

MIN-03: Missing Event Emission in setQuoteProvider()

Severity: Minor
Contract: `HedgedPool.sol`

Lines: 669-675
Status: Resolved.

Description:

The `setQuoteProvider()` function modifies critical access control but doesn't emit an event.

Affected Code:

`HedgedPool.sol` - Lines 669-675:

```
/// @notice Add/remove an address from allowed quote providers
function setQuoteProvider(
    address quoteProviderAddress,
    bool isPermitted
) external onlyOwner {
    quoteProviders[quoteProviderAddress] = isPermitted;
}
```

Recommendation:

Add event emission:

```
event QuoteProviderSet(address indexed quoteProvider, bool isPermitted);

function setQuoteProvider(
    address quoteProviderAddress,
    bool isPermitted
) external onlyOwner {
    quoteProviders[quoteProviderAddress] = isPermitted;
    emit QuoteProviderSet(quoteProviderAddress, isPermitted);
}
```

MIN-04: Missing Event Emission in setLock()

Severity: Minor
Contract: `HedgedPool.sol`
Lines: 709-712
Status: Resolved.

Description:

The `setLock()` function changes access key requirements but doesn't emit an event.

Affected Code:

`HedgedPool.sol` - Lines 709-712:

```
/// @notice Set access key token requirement
function setLock(bool _isLocked) external onlyOwner {
    isLocked = _isLocked;
}
```

Recommendation:

Add event emission:

```
event LockStatusChanged(bool isLocked);

function setLock(bool _isLocked) external onlyOwner {
    isLocked = _isLocked;
    emit LockStatusChanged(_isLocked);
}
```

MIN-05: Missing Event Emission in setAddressBook() - TradeExecutor

Severity: Minor
Contract: `TradeExecutor.sol`
Lines: 43-45
Status: Resolved.

Description:

The `setAddressBook()` function updates critical protocol addresses but doesn't emit an event.

Affected Code:

`setAddressBook.sol` - Lines 42-45:

```
    /// Initialize the contract, and create an lpToken to track ownership
    function setAddressBook(address _addresBookAddress) external onlyOwner
{
        addressBook = IAddressBook(_addresBookAddress);
    }
```

Recommendation:

Add event emission:

```
event AddressBookUpdated(address indexed newAddressBook);

function setAddressBook(address _addressBook) external onlyOwner {
    addressBook = IAddressBook(_addressBook);
```

```
    emit AddressBookUpdated(_addressBook);
}
```

MIN-06: Missing Address(0) Validation in HedgedPool._*refreshConfigInternal*

Severity: Minor
Contract: `HedgedPool.sol`
Lines: 683-707
Status: Acknowledged. The client states that they are running automated scripts to ensure the address book is configured correctly, reducing the risk of misconfiguration.

Description:

The `_refreshConfigInternal()` function fetches addresses from AddressBook but doesn't validate they're not address(0), which could lead to broken functionality if AddressBook is misconfigured.

Affected Code:

`HedgedPool.sol` - Lines 683-707:

```
function _refreshConfigInternal() internal {
    // remove old approvals
    if (marginPool != address(0)) {
        collateralToken.approve(marginPool, 0);
    }
    if (feeCollector != address(0)) {
        collateralToken.approve(feeCollector, 0);
    }
    if (tradeExecutor != address(0)) {
        collateralToken.approve(tradeExecutor, 0);
    }

    controller = addressBook.getController();

    // ... existing code
```

Recommendation:

Add validation:

```
function _refreshConfigInternal() internal {
    // ... existing code

    address newController = addressBook.getController();
    require(newController != address(0), "Invalid controller");
    controller = newController;
```

```
    // ... existing code
}
```

Also add event emissions:

```
event ConfigUpdated(
    address controller,
    address oracle,
    address marginPool,
    // ... other addresses
);
```

MIN-07: Missing Emergency Event for closeRoundAdmin()

Severity: Minor
Contract: `HedgedPool.sol`
Lines: 229-231
Status: Resolved.

Description:

The `closeRoundAdmin()` function bypasses price-per-share guardrails, which is likely used in emergency situations. This critical action should emit a specific emergency event for monitoring.

Affected Code:

`HedgedPool.sol` - Lines 229-231:

```
/// @notice Manually close round using price per share outside of the
guardrails
/// @param pricePerShare price per share * 1e8
function closeRoundAdmin(uint256 pricePerShare) external onlyOwner {
    _closeRound(pricePerShare);
}
```

Recommendation:

Add emergency event:

```
event EmergencyRoundClosed(uint256 pricePerShare, uint256 timestamp);

function closeRoundAdmin(uint256 pricePerShare) external onlyOwner {
    emit EmergencyRoundClosed(pricePerShare, block.timestamp);
    _closeRound(pricePerShare);
}
```

MIN-08: Incomplete Event Data in DepositRequested

Severity: Minor
Contract: `LpManager.sol`
Lines: 18-23
Status: Acknowledged. The client states that, for now, the LP Manager will only be used by permissioned users, and therefore considers this issue to be low priority.

Description:

The `DepositRequested` event should include `unredeemedShares` updates for complete state tracking.

Affected Code:

`LpManager.sol` - Lines 18-23:

```
/// @notice Emitted when LP deposits
event DepositRequested(
    address pool,
    address lp,
    uint256 amount,
    uint256 roundId
);
```

Recommendation:

```
event DepositRequested(
    address indexed pool,
    address indexed lp,
    uint256 amount,
    uint256 roundId,
    uint256 unredeemedShares
);
```

MIN-09: Incomplete Event Data in WithdrawalRequested

Severity: Minor
Contract: `LpManager.sol`
Lines: 26-31
Status: Acknowledged. The client states that, for now, the LP Manager will only be used by permissioned users, and therefore considers this issue to be low priority.

Description:

The `WithdrawalRequested` event should include the new `sharesTotal` amount for complete state tracking.

Affected Code:

`LpManager.sol` - Lines 26-31:

```
/// @notice Emitted when LP requests a withdrawal
event WithdrawalRequested(
    address pool,
    address lp,
    uint256 shares,
    uint256 roundId
);
```

Recommendation:

```
event WithdrawalRequested(
    address indexed pool,
    address indexed lp,
    uint256 shares,
    uint256 roundId,
    uint256 newSharesTotal
);
```

MIN-10: Incomplete Event Data in DepositRoundClosed

Severity: Minor
Contract: `LpManager.sol`
Lines: 54-58
Status: Acknowledged. The client states that, for now, the LP Manager will only be used by permissioned users, and therefore considers this issue to be low priority.

Description:

The `DepositRoundClosed` event should include `sharesAdded` for complete state tracking.

Affected Code:

`LpManager.sol` - Lines 54-58:

```
event DepositRoundClosed(
    address pool,
    uint256 roundId,
    uint256 pricePerShare
);
```

Recommendation:

```
event DepositRoundClosed(
    address indexed pool,
    uint256 roundId,
    uint256 pricePerShare,
    uint256 sharesAdded
);
```

MIN-11: Inconsistent Use of Custom Errors vs Require Statements

Severity: Minor
Contract: Multiple contracts
Lines: LpManager.sol: 78, 126; HedgedPool.sol: 629
Status: Acknowledged. The client considers this issue to be low priority.

Description:

The codebase mixes `require()` statements and custom errors. Custom errors are more gas-efficient and should be used consistently throughout the protocol.

Affected Code Examples:

`LpManager.sol` - Lines 78, 126:

```
require(pricePerShare > 0, "!price");
```

`HedgedPool.sol` - Line 629:

```
require(_underlying != address(0));
```

Recommendation:

Convert all `require` statements to custom errors:

```
// Add to CustomErrors.sol
error InvalidPricePerShare();
error InvalidAddress();

// Usage in LpManager.sol
if (pricePerShare == 0) revert CustomErrors.InvalidPricePerShare();

// Usage in HedgedPool.sol
if (_underlying == address(0)) revert CustomErrors.InvalidAddress();
```

MIN-12: Missing Reentrancy Guard in depositToPool() - HyperliquidHedger

Severity: Minor
Contract: `HyperliquidHedger.sol`
Lines: 96-113
Status: Resolved.

Description:

The `depositToPool()` function lacks reentrancy protection despite making external calls. While `WHYPE_ADDRESS` is a trusted wrapper contract, and `onlyAuthorized` reduces the risk of exploitation, a reentrancy guard should be added for defense in depth and future changes.

Affected Code:

`HyperliquidHedger.sol` - Lines 96-113:

```
function depositToPool(
    address token,
    uint256 amount,
    bool needWrap,
    bool fromCore
) external onlyAuthorized {
    if (needWrap) {
        _wrapHype(amount);
    }
    if (fromCore) {
        if (externalPendingDeposits[token] < amount) {
            revert CustomErrors.InvalidPendingAmount();
        }
        externalPendingDeposits[token] -= amount;
    }

    UniswapOrdersLib._depositToPoolAndTransfer(hedgedPool, token, amount);
}
```

Recommendation:

Add `nonReentrant` modifier:

```
function depositToPool(
    address token,
    uint256 amount,
    bool needWrap,
    bool fromCore
) external onlyAuthorized nonReentrant {
    // ... existing code
}
```

MIN-13: Commented-Out Liquidation and Call Actions in Gamma Protocol Controller

Severity: Minor
Contract: `Controller.sol` (Gamma Protocol submodule)
Lines: 650-653
Status: Acknowledged. The clients states that this issue will be solved in future iterations.

Description:

The Gamma Protocol Controller has commented-out code for `Liquidate` and `Call` actions in the `_runActions()` function. While the implementation functions `_liquidate()` and `_call()` exist in the contract, they are not reachable through the main `operate()` entry point.

Affected Code:

`Controller.sol` - Lines 650-653:

```
} else if (actionType == Actions.ActionType.SettleVault) {
    _settleVault(Actions._parseSettleVaultArgs(action));
    //            } else if (actionType == Actions.ActionType.Liquidate) {
    //                _liquidate(Actions._parseLiquidateArgs(action));
    //            } else if (actionType == Actions.ActionType.Call) {
    //                _call(Actions._parseCallArgs(action));
}
```

Recommendation:

Document the decision by adding explanations to why these actions are disabled or remove the dead code and unused functions.

MIN-14: Incomplete HedgedPoolInitialized Event

Severity: Minor
Contract: `HedgedPool.sol`
Lines: 33-39, 131-137
Status: Acknowledged. The client considers this issue to be low priority.

Description:

The `HedgedPoolInitialized` event could include more useful information such as decimals, initial price per share, and round end dates.

Affected Code:

`HedgedPool.sol` - Event Declaration Lines 33-39:

```
/// @notice Pool is initialized
event HedgedPoolInitialized(
    address strikeToken,
```

```
    address collateralToken,
    string tokenName,
    string tokenSymbol
);
```

`HedgedPool.sol` - Event Emission Lines 131-137:

```
emit HedgedPoolInitialized(
    _strikeToken,
    _collateralToken,
    _tokenName,
    _tokenSymbol
);
```

Recommendation:

```
event HedgedPoolInitialized(
    address strikeToken,
    address collateralToken,
    string tokenName,
    string tokenSymbol,
    uint8 decimals,
    uint256 initialPricePerShare,
    uint256 depositRoundEnd,
    uint256 withdrawalRoundEnd
);
```

MIN-15: Dead Code - Redundant Vault Type Checks in ControllerLib - Gamma Protocol

Severity: Minor
Contract: `ControllerLib.sol` (Gamma Protocol submodule)
Lines: 157, 184
Status: Acknowledged. The client states that they might re-enable `vaultType=1` functionality in the future.

Description:

The `_openVault()` function enforces `require(_args.vaultType == 0, "C50")`, meaning only vault type 0 can be created. However, `ControllerLib.sol` contains dead code checking for `typeVault == 1`.

Dead Code Locations:

1. `depositCollateral()` - Line 157:

```
if (typeVault == 1) { // Never executes
    nakedPoolBalance[_args.asset] =
nakedPoolBalance[_args.asset].add(_args.amount);
    require(nakedPoolBalance[_args.asset] <= nakedCap[_args.asset],
"C37");
}
```

2. `withdrawCollateral()` - Line 184:

```
if (typeVault == 1) { // Never executes
    nakedPoolBalance[_args.asset] =
nakedPoolBalance[_args.asset].sub(_args.amount);
}
```

Recommendation:

Remove dead code and unused parameters, or enable vault type 1 if intended, current checks waste gas.

MIN-16: Missing Access Control in Various LPManager Functions - LpManager

Severity: Minor
Contract: `LpManager.sol`
Lines: 75-323
Status: Acknowledged. The client states that, for now, the LP Manager will only be used by permissioned users, and therefore considers this issue to be low priority.

Description:

The `closeWithdrawalRound`, `closeDepositRound`, `addPricedCash`, `addPendingCash`, `requestWithdrawal`, `requestDeposit`, `withdrawCash`, `cancelPendingDeposit` and `redeemShares` functions assume that the message sender will always be the Hedged Pool. If any random user interacts with these functions, stale data will be stored in state mappings. For example, when any user interacts with the `closeWithdrawalRound` function, `withdrawalRoundId` gets incremented by 1.

This can make operations more costly in the future if large mappings are to be iterated.

Affected Code:

`LpManager.sol` - Lines 75-323:

```
function closeWithdrawalRound(
    uint256 pricePerShare
) external returns (uint256 sharesRemoved) {
    require(pricePerShare > 0, "!price");

    address poolAddress = msg.sender;
    // ... existing code
```

```
}

function closeDepositRound(
    uint256 pricePerShare
) external returns (uint256 sharesAdded) {
    require(pricePerShare > 0, "!price");

    address poolAddress = msg.sender;
    // ... existing code
}

function addPricedCash(uint256 cashAmount, uint256 shareAmount) external {
    require(cashAmount > 0, "!cashAmount");
    require(shareAmount > 0, "!shareAmount");

    address pool = msg.sender;
    // ... existing code
}

function addPendingCash(uint256 cashAmount) external {
    address pool = msg.sender;
    // ... existing code
}

function requestWithdrawal(
    address lpAddress,
    uint256 sharesAmount
) external {
    address pool = msg.sender;
    // ... existing code
}

function requestDeposit(address lpAddress, uint256 cashAmount) external {
    require(cashAmount > 0, "!cashAmount");
    address pool = msg.sender;
    // ... existing code
}

function withdrawCash(
    address lpAddress
) external returns (uint256, uint256) {
    address pool = msg.sender;
    // ... existing code
}

function cancelPendingDeposit(address lpAddress, uint256 amount) external
{
    address pool = msg.sender;
    // ... existing code
}

function redeemShares(address lpAddress) external returns (uint256) {
    address pool = msg.sender;
    // ... existing code
```

Recommendation:

Make sure that only known hedged pools can execute the affected functions.

MIN-17: getPoolInfo Should be Marked view - Lens

Severity: Minor
Contract: `Lens.sol`
Lines: 37-98
Status: Resolved.

Description:

The `getPoolInfo` function does not modify contract state and should therefore be marked as `view` to enforce read-only access and prevent accidental state changes in future updates.

Affected Code:

`Lens.sol` - Lines 37-98:

```
function getPoolInfo(
    address poolAddress
) external returns (PoolInfo memory) {
    // ... existing code
}
```

Recommendation:

Mark the function as `view`

```
function getPoolInfo(
    address poolAddress
) external view returns (PoolInfo memory) {
    // ... existing code
}
```

MIN-18: _withdrawFee Does Not Verify Referrer - FeeCollector

Severity: Minor
Contract: `FeeCollector.sol`
Lines: 89-99
Status: Acknowledged. The client considers this issue to be low priority.

Description:

The `_withdrawFee` function depends on the caller to correctly specify whether the withdrawal is for the owner or a referrer. To prevent potential bugs or misuse, it is best practice to implement this verification within the `_withdrawFee` function itself.

Affected Code:

`FeeCollector.sol` - Lines 89-99:

```
function _withdrawFee(address feeAsset, address referrer) internal {
    uint256 feeAmount = getBalance(feeAsset, referrer);

    if (feeAmount == 0) return;

    feesWithdrawn[referrer][feeAsset] += feeAmount;

    IERC20(feeAsset).safeTransfer(msg.sender, feeAmount);

    emit FeeWithdrawn(referrer, feeAsset, feeAmount);
}
```

Recommendation:

Add `referrer` verification

```
function _withdrawFee(address feeAsset, address referrer) internal {
    // Only allow referrer or owner to withdraw
    require(
        msg.sender == referrer || (referrer == address(0) && msg.sender ==
owner()),
        "Not authorized"
    );
    // ... existing code
}
```

MIN-19: _withdrawFee Sends Fees to msg.sender - FeeCollector

Severity: Minor
Contract: `FeeCollector.sol`
Lines: 89-99
Status: Acknowledged. The client considers this issue to be low priority.

Description:

The `_withdrawFee` function currently transfers collected fees to `msg.sender` rather than the intended `referrer`. If this function were made external, malicious actors could withdraw fees from legitimate users simply by specifying their address. To eliminate this risk, the function should transfer fees directly to the referrer address.

Affected Code:

`FeeCollector.sol` - Lines 89-99:

```
function _withdrawFee(address feeAsset, address referrer) internal {
    // ... existing code

    IERC20(feeAsset).safeTransfer(msg.sender, feeAmount);

    // ... existing code
}
```

Recommendation:

Send fees to referrer

```
function _withdrawFee(address feeAsset, address referrer) internal {
    // ... existing code

    address recipient = referrer == address(0) ? owner() : referrer;
    IERC20(feeAsset).safeTransfer(recipient, feeAmount);

    // ... existing code
}
```

MIN-20: Missing Address(0) Validation in AddressBook Setters - AddressBook

Severity: Minor
Contract: `AddressBook.sol`
Lines: 133-178, 239-245
Status: Acknowledged. The client considers this issue to be low priority.

Description:

The setter functions in `AddressBook.sol` do not validate that the new address is not `address(0)`. If the owner accidentally sets a critical protocol address to `address(0)`, it can lead to loss of funds, failed external calls, or denial of service in dependent modules.

The severity of this issue is Minor only because the setters are gated.

Affected Code:

`AddressBook.sol` - Lines 133-178, 239-245:

```
function setOpynAddressBook(address opynAddressBook) external onlyOwner {
    _addresses[OPYN_ADDRESS_BOOK] = opynAddressBook;
    emit OpynAddressBookUpdated(opynAddressBook);
```

```
}

function setLpManager(address lpManagerAddress) external onlyOwner {
    _addresses[LP_MANAGER] = lpManagerAddress;
    emit LpManagerUpdated(lpManagerAddress);
}

function setOrderUtil(address orderUtilAddress) external onlyOwner {
    _addresses[ORDER_UTIL] = orderUtilAddress;
    emit OrderUtilUpdated(orderUtilAddress);
}

function setFeeCollector(address feeCollectorAddress) external onlyOwner {
    _addresses[FEE_COLLECTOR] = feeCollectorAddress;
    emit FeeCollectorUpdated(feeCollectorAddress);
}

function setLens(address lensAddress) external onlyOwner {
    _addresses[LENS] = lensAddress;
    emit LensUpdated(lensAddress);
}

function setTradeExecutor(address tradeExecutorAddress) external onlyOwner
{
    _addresses[TRADE_EXECUTOR] = tradeExecutorAddress;
    emit TradeExecutorUpdated(tradeExecutorAddress);
}

function setPerennialMultiInvoker(
    address multiInvokerAddress
) external onlyOwner {
    _addresses[PERENNIAL_MULTI_INVOKER] = multiInvokerAddress;
    emit PerennialMultiInvokerUpdated(multiInvokerAddress);
}

function setPerennialLens(address lensAddress) external onlyOwner {
    _addresses[PERENNIAL_LENS] = lensAddress;
    emit PerennialLensUpdated(lensAddress);
}

// ... existing code

function setAddress(
    bytes32 id,
    address newAddress
) external override onlyOwner {
    _addresses[id] = newAddress;
    emit AddressSet(id, newAddress, false);
}
```

Recommendation:

Make sure that the new address is not `address(0)`

```
function setOpynAddressBook(address opynAddressBook) external onlyOwner {
    require(opynAddressBook != address(0), "Invalid address");
    // ... existing code
}

function setLpManager(address lpManagerAddress) external onlyOwner {
    require(lpManagerAddress != address(0), "Invalid address");
    // ... existing code
}

function setOrderUtil(address orderUtilAddress) external onlyOwner {
    require(orderUtilAddress != address(0), "Invalid address");
    // ... existing code
}

function setFeeCollector(address feeCollectorAddress) external onlyOwner {
    require(feeCollectorAddress != address(0), "Invalid address");
    // ... existing code
}

function setLens(address lensAddress) external onlyOwner {
    require(lensAddress != address(0), "Invalid address");
    // ... existing code
}

function setTradeExecutor(address tradeExecutorAddress) external onlyOwner
{
    require(tradeExecutorAddress != address(0), "Invalid address");
    // ... existing code
}

function setPerennialMultiInvoker(
    address multiInvokerAddress
) external onlyOwner {
    require(multiInvokerAddress != address(0), "Invalid address");
    // ... existing code
}

function setPerennialLens(address lensAddress) external onlyOwner {
    require(lensAddress != address(0), "Invalid address");
    // ... existing code
}

// ... existing code

function setAddress(
    bytes32 id,
    address newAddress
) external override onlyOwner {
    require(newAddress != address(0), "Invalid address");
    // ... existing code
}
```

Informational Findings

INFO-01: Typo in Parameter Name - __HedgedPool_init

Severity: Informational
Contract: `HedgedPool.sol`
Lines: 101-108
Status: Resolved.

Description:

The parameter name `_addresBookAddress` is misspelled (should be `_addressBookAddress`).

Affected Code:

`HedgedPool.sol` - Lines 101-108:

```
function __HedgedPool_init(
    address _addresBookAddress,
    address _strikeToken,
    address _collateralToken,
    string calldata _tokenName,
    string calldata _tokenSymbol
) public initializer {
    addressBook = IAddressBook(_addresBookAddress);
```

Recommendation:

Fix the typo:

```
function __HedgedPool_init(
    address _addressBookAddress,
    address _strikeToken,
    address _collateralToken,
    string calldata _tokenName,
    string calldata _tokenSymbol
) public initializer {
    addressBook = IAddressBook(_addressBookAddress);
```

INFO-02: TODO Comments in Production Code

Severity: Informational
Contract: `HedgedPool.sol`
Lines: 540, 564
Status: Acknowledged. The client considers this issue to be low priority.

Description:

Production code contains unresolved TODO comments that should be addressed before deployment.

Affected Code:

Line 540:

```
// same as exercise window.
// TODO: should we set this as param?
if (block.timestamp > leg.expiration - 24 * 3600) {
```

Line 564:

```
// TODO: check n-of-m co-signers
// Check that the signatory has the Role Quote provider
```

Recommendation:

Either implement the TODOs or remove the comments if the functionality is not needed.

INFO-03: TODO Comment in LpManager

Severity: Informational
Contract: `LpManager.sol`
Lines: Top of contract
Status: Acknowledged. The client considers this issue to be low priority.

Description:

Comment states `// TODO: make upgradeble` but contract is not upgradeable.

Recommendation:

Either make the contract upgradeable or remove the TODO if upgradeability is not required.

INFO-04: Missing SPDX License Identifiers

Severity: Informational
Contract: Multiple contracts
Status: Resolved.

Description:

14 files are missing SPDX license identifiers. All Solidity files should include license information.

Recommendation:

Add appropriate SPDX identifier at the top of each file:

```
// SPDX-License-Identifier: MIT
```

Or if proprietary:

```
// SPDX-License-Identifier: None
```

INFO-05: Use Ownable2Step Instead of Ownable

Severity: Informational
Contract: All contracts using OwnableUpgradeable
Status: Acknowledged. The client plans to investigate this issue in future iterations.

Description:

All contracts use OpenZeppelin's `OwnableUpgradeable` which uses one-step ownership transfer. This is risky as transferring to an incorrect address permanently locks the contract.

Recommendation:

Use `Ownable2StepUpgradeable` instead:

```
contract HedgedPool is
    Initializable,
    Ownable2StepUpgradeable,
    ERC20Upgradeable,
    ReentrancyGuardUpgradeable,
    ERC1155Holder,
    HedgedPoolStorageV1
{
    // ...
}
```

This requires the new owner to accept ownership via `acceptOwnership()`, preventing accidental transfers to wrong addresses.

INFO-06: Missing Event Emission in setKeeper() - Gamma Protocol OtokenFactory

Severity: Informational
Contract: `OtokenFactory.sol` (Gamma Protocol submodule)
Lines: 224-226
Status: Acknowledged. The client considers this issue to be low priority.

Description:

The `setKeeper()` function allows the owner to grant or revoke keeper permissions, which is a critical authorization change. However, the function does not emit an event when keeper status changes, making it difficult to track authorization changes off-chain.

Affected Code:

`OtokenFactory.sol` - Lines 224-226:

```
/// @notice Allow/disallow an address to perform keeper tasks
function setKeeper(address keeperAddress, bool isPermitted) external
onlyOwner {
    keepers[keeperAddress] = isPermitted;
}
```

Recommendation:

Add an event emission to track keeper authorization changes:

```
/// @notice emitted when a keeper's authorization status changes
event KeeperUpdated(address indexed keeperAddress, bool isPermitted);

/// @notice Allow/disallow an address to perform keeper tasks
function setKeeper(address keeperAddress, bool isPermitted) external
onlyOwner {
    keepers[keeperAddress] = isPermitted;
    emit KeeperUpdated(keeperAddress, isPermitted);
}
```

INFO-07: Misleading Error Message in whitelistProduct() - Gamma Protocol Whitelist

Severity: Informational
Contract: `Whitelist.sol` (Gamma Protocol submodule)
Lines: 138
Status: Acknowledged. The client considers this issue to be low priority.

Description:

The error message "Only allow fully collateralized products" in the `whitelistProduct()` function is misleading. The current implementation only enforces full collateralization for PUT options, but allows ANY collateral type for CALL options.

Affected Code:

`Whitelist.sol` - Line 138:

```
require((_isPut && (_strike == _collateral)) || !_isPut, "Whitelist: Only
allow fully collateralized products");
```

Misleading error messages states "Only allow fully collateralized products" but call options can use any collateral.

Recommendation:

Update the error message to accurately reflect the validation:

```
require(
    (_isPut && (_strike == _collateral)) || !_isPut,
    "Whitelist: Put options must be fully collateralized with strike
asset"
);
```

INFO-08: Magic numbers in various contracts - HedgePool, LpManager, FeeCollector, HyperliquidHedger

Severity: Informational
Contract: `HedgePool.sol`, `FeeCollector.sol`, `LpManager.sol`, `HyperliquidHedger.sol`
Lines: `HedgePool.sol` 141-145; `LpManager.sol` 75-119; `FeeCollector.sol` 50-76; `HyperliquidHedger` 228-233
Status: Acknowledged. The client considers this issue to be low priority.

Description:

Throughout the codebase, several contracts use hardcoded numeric constants ("magic numbers") for calculations. This practice reduces code clarity and maintainability, as the meaning of these numbers is not immediately obvious and changes require updates in multiple locations.

Replacing magic numbers with named constants improves readability, reduces the risk of errors, and makes future modifications easier.

Affected Code:

`HedgePool.sol` - Lines 141-145:

```
function getTotalPoolValue(
    uint256 pricePerShare
) public view returns (uint256) {
    return (totalSupply() * pricePerShare) / 1e8;
}
```

`LpManager.sol` - Lines 75-119:

```
function closeWithdrawalRound(
    uint256 pricePerShare
) external returns (uint256 sharesRemoved) {
    // ... existing code

    // process pending withdrawal cash
    uint256 cashNeeded = ((previousRound.sharesTotal -
        previousRound.sharesFilled) * pricePerShare) / 1e8;

    // ... existing code

    // mark cash as ready for withdrawal
    uint256 pendingShares = (previousRound.cashPending * 1e8) /
        pricePerShare;

    // ... existing code
}
```

FeeCollector.sol - Lines 50-76:

```
function collectFee(
    address feeAsset,
    uint256 feeAmount,
    address referrer
) external {
    // ... existing code

    if (referrer != address(0)) {
        referrerAmount = (feeAmount * referrerFeeSplitBips) / 10_000;
        // record referrer fees
        feesCollected[referrer][feeAsset] += referrerAmount;
    }

    // record protocol fees
    feesCollected[address(0)][feeAsset] += feeAmount - referrerAmount;

    // ... existing code
}
```

HyperliquidHedger.sol - Lines 228-233

```
function _extraWeiDecimals(uint32 token) internal view returns (int8) {
    if (token == HYPE_TOKEN_ID) {
        return 10;
    }
    // ... existing code
}
```

Recommendation:

Review all contracts and replace magic numbers with named constants. Example:

```
// ********** At the top of the contract **********

// HedgePool.sol
uint256 private constant PRICE_SCALE = 1e8;

// FeeCollector.sol
uint256 private constant BASIS_POINTS = 10_000;

// HyperliquidHedger.sol
int8 private constant HYPE_TOKEN_DECIMALS = 10;

// ********** Usage **********

// HedgePool.sol
return (totalSupply() * pricePerShare) / PRICE_SCALE;

// FeeCollector.sol
referrerAmount = (feeAmount * referrerFeeSplitBips) / BASIS_POINTS;

// HyperliquidHedger.sol
if (token == HYPE_TOKEN_ID) {
    return HYPE_TOKEN_DECIMALS;
}
```

INFO-09: Unconfigurable WHYPE_ADDRESS, HYPE_TRANSFER_ADDRESS, TOKEN_INFO_PRECOMPILE_ADDRESS, and HYPE_TOKEN_ID - HyperliquidHedger

Severity: Informational
Contract: `HyperliquidHedger.sol`
Lines: `HyperliquidHedger` 24-33
Status: Acknowledged. The client considers this issue to be low priority.

Description:

The `HyperliquidHedger` contract currently hardcodes critical protocol addresses and identifiers such as `WHYPE_ADDRESS`, `HYPE_TOKEN_ID`, `HYPE_TRANSFER_ADDRESS`, and `TOKEN_INFO_PRECOMPILE_ADDRESS`. While this may be suitable for immutable values, it reduces flexibility and upgradeability. If any of these values change due to protocol upgrades or migrations, the contract would require redeployment, risking downtime and operational complexity.

Affected Code:

`HyperliquidHedger.sol` - Lines 24-33:

```
address internal constant WHYPE_ADDRESS =
    0x5555555555555555555555555555555555555555;

address internal constant HYPE_TRANSFER_ADDRESS =
    0x2222222222222222222222222222222222222222;

address internal constant TOKEN_INFO_PRECOMPILE_ADDRESS =
    0x000000000000000000000000000000000000080C;

uint64 internal constant HYPE_TOKEN_ID = 150;
```

Recommendation:

Make these protocol addresses and identifiers configurable by storing them as internal state variables and providing an `onlyOwner` setter function.

```
address internal WHYPE_ADDRESS;
address internal HYPE_TRANSFER_ADDRESS;
address internal TOKEN_INFO_PRECOMPILE_ADDRESS;
uint64 internal HYPE_TOKEN_ID.

function setProtocolAddressesAndIdentifiers(
    address _whypeAddress,
    address _hypeTransferAddress,
    address _tokenInfoPrecompileAddress,
    uint64 _hypeTokenId
) external onlyOwner {
    require(_whypeAddress != address(0), "Invalid WHYPE address");
    require(_hypeTransferAddress != address(0), "Invalid transfer
address");
    require(_tokenInfoPrecompileAddress != address(0), "Invalid precompile
address");
    WHYPE_ADDRESS = _whypeAddress;
    HYPE_TRANSFER_ADDRESS = _hypeTransferAddress;
    TOKEN_INFO_PRECOMPILE_ADDRESS = _tokenInfoPrecompileAddress;
    HYPE_TOKEN_ID = _hypeTokenId;
}
```

INFO-10: Missing Zero-Value Check in spotSend - HyperliquidHedger

Severity: Informational
Contract: `HyperliquidHedger.sol`
Lines: `HyperliquidHedger` 272-296
Status: Acknowledged. The client considers this issue to be low priority.

Description:

The `spotSend` function does not check whether the `weiAmount` parameter is zero. This can result in unnecessary computation which wastes gas.

Affected Code:

`HyperliquidHedger.sol` - Lines 272-296:

```
function spotSend(uint64 token, uint64 weiAmount) external onlyAuthorized
{
    address destination;
    if (token == HYPE_TOKEN_ID) {
        destination = HYPE_TRANSFER_ADDRESS;
    } else {
        destination = _generateSystemAddress(token);
    }
    if (destination == address(0)) {
        revert CustomErrors.InvalidDestinationAddress();
    }

    // ... existing code
}
```

Recommendation:

Add a check at the beginning of the function to revert if `weiAmount` is zero.

```
function spotSend(uint64 token, uint64 weiAmount) external onlyAuthorized
{
    if (weiAmount == 0) {
        revert CustomErrors.ZeroValue();
    }

    // ... existing code
}
```

INFO-11: Missing Access Control on settleAll()

Severity: Informational
Contract: `HedgedPool.sol`
Lines: 155-177
Status: Acknowledged. The client states that they may want to enable or encourage other parties to perform settlements on behalf of the protocol to avoid paying gas fees. The client states that while they will execute settlements for now, there are no exploits created by leaving this function unpermissioned.

Description:

The `settleAll()` function has no access control, allowing any address to trigger settlement operations. While not a security risk currently, this could lead to unintended behavior if future functionality is added that

relies on controlled settlement timing or authorization.

Affected Code:

`HedgedPool.sol` - Lines 155-177:

```
/**
 * @notice Settle all expired oTokens
 */
function settleAll() public {
    uint256 expiry = lastSettledExpiry + 1 days;
    uint8 numProcessed;
    while (expiry <= block.timestamp && numProcessed < 5) {
        numProcessed += 1;
        HedgedPoolLib.settleExpiry(
            expiry,
            underlyingTokens,
            oTokensByExpiry,
            controller,
            notionalExposure,
            activeOTokens
        );
```

Recommendation:

Add keeper access control:

```
function settleAll() public {
    onlyKeeper();
    uint256 expiry = lastSettledExpiry + 1 days;
    // ... rest of function
}
```