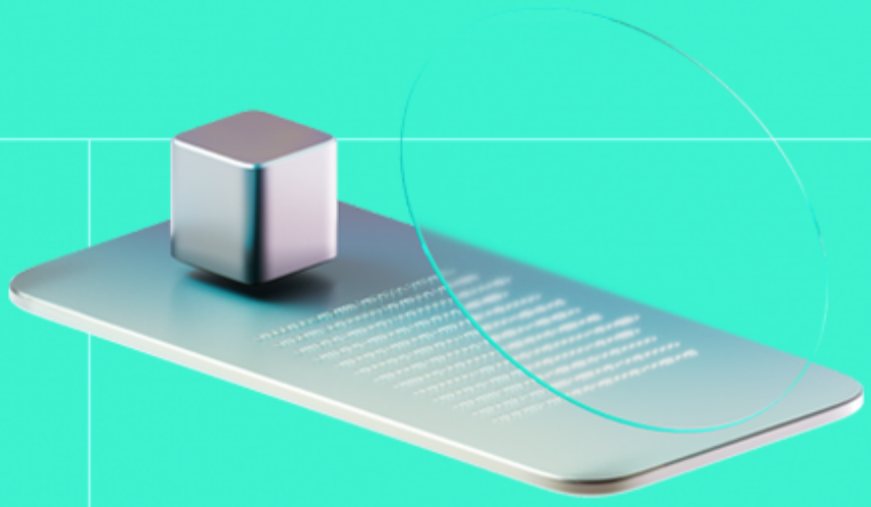
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Lys

Date: 10/12/2024

We express our gratitude to the Lys team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The LYS protocol aims to create an innovative Ethereum staking solution. By integrating liquid staking, re-staking and yield boosters, it enables users to earn staking rewards while maintaining liquidity of their assets.

Document

Name	Smart Contract Code Review and Security Analysis Report for Lys
Audited By	David Camps Novi, Andy Cho
Approved By	Ataberk Yavuzer
Website	https://lys.xyz
Changelog	26/11/2024 - Preliminary Report
	10/12/2024 - Final Report
Platform	EVM, Stader, Lido, Kelp
Language	Solidity
Tags	Staking, Incentives, Vault
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/protofire/lys-protocol
Commit	d304f92

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

6	3	2	1
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	3
Low	3

Vulnerability	Severity
F-2024-7337 - Vault Valuation Should Always be Updated before Deposits and Withdrawals	Medium
F-2024-7359 - Lack of Slippage can Result in Loss of Funds	Medium
F-2024-7362 - Leftover Funds can Become Locked in the Strategy Contract	Medium
F-2024-7126 - Outdated Price Usage due to Lack of Price Oracle Consistency	Low
F-2024-7140 - Selected Vaults in Convergent Portfolio Transfer may not Match Actual Portfolio	Low
F-2024-7347 - Aggregator Fee can Surpass 100%	Low

Documentation quality

- Functional requirements are provided.
- Technical description is provided.
- Missing system roles description.

Code quality

- The development environment is configured.
- The testing environment is not configured.

Test coverage

Code coverage of the project is **64%** (branch coverage).

- The test suite is extensive but does not cover all cases.
- Pendle interactions are not not tested thoroughly: the Pendle Adapter and Strategy contracts do not have sufficient test coverage.

Table of Contents

System Overview

The LYS protocol aims to create an innovative Ethereum staking solution. By integrating liquid staking, re-staking and yield boosters, it enables users to earn staking rewards while maintaining liquidity of their assets.

The project consists of the following contracts:

- **AggregatorToken** - yield-bearing token and entry point for users to interact with the system vaults.
- **VaultsRegistry** - used for deployment of new vaults and managing their configurations.
- **Vault** - ERC4626 vault that allows users to deposit and withdraw assets (from AggregatorToken entry point) in exchange for shares. It is also the starting point of the deployment of deposited capital towards external protocols in order to get yield.
- **UniformTransferStrategy** - configures and executes a method to transfer the Lys protocol token among users in a uniform manner across the held vaults by the user.
- **ConvergentPortfolioTransferStrategy** - configures and executes a method to transfer the Lys protocol token among users in a convergent manner across the held vaults by the user.
- **NStepsStrategy** - configures the different yield-enabling strategies that will be executed on different external protocols. It also holds the yield-bearing tokens obtained from third party protocols interactions.
- **PendleStrategy** - configures the Pendle interactions.
- **Adapters** - the protocol includes several Adapter contracts, each create specifically for each external protocol to interact. Each adapter configures and execute the corresponding interactions in order to deposit or withdraw tokens into third parties.
- **PriceFetcher** - configures external oracles and queries price data from them.
- **Swapper** - manages swaps with Uniswap and Curve protocols.
- **Wrapper** - manages wrapping of tokens that are necessary in order to perform correct interactions with external protocols.

Privileged roles

- Description and usage of system roles was not provided by the development team.

Potential Risks

- The project utilizes Solidity version `0.8.20` or higher, which includes the introduction of the `PUSH0` (`0x5f`) opcode. This opcode is currently supported on the Ethereum mainnet but may not be universally supported across other blockchain networks. Consequently, deploying the contract on chains other than the Ethereum mainnet, such as certain Layer 2 (L2) chains or alternative networks, might lead to compatibility issues or execution errors due to the lack of support for the `PUSH0` opcode. In scenarios where deployment on various chains is anticipated, selecting an appropriate Ethereum Virtual Machine (EVM) version that is widely supported across these networks is crucial to avoid potential operational disruptions or deployment failures.
- The Vault valuation can be modified by the `VAULT_MANAGER_ROLE` when adding or retrieving tokens via `increaseCapital()` and `decreaseCapital()`. This will affect the `totalAssets()` of the vault which, in turn, will affect the amount of `shares` minted during `deposits` and their valuation during `withdrawals`.
- The protocol uses the OpenZeppelin implementation of the `ERC4626` vault as a basis to build the `Vaults`. It should be noted that such contract implemented an specific way to minimize the side effects of the so-called inflation attack. Whilst is provides a reasonable solution to the problem, it has some side effects that are explained in the OpenZeppelin ERC4626 contract NatSpec.
- The deposit and withdrawal of user's underlying tokens can be paused by the protocol managers at will.
- The protocol uses slippage in order to calculate different amounts of tokens, as it can be seen in `PendleAdapter::_calculateSlippageAdjustedAmount()` or `BaseAdapter::_calculateMaxAmountIn()`. It should be noted than in some cases, the slippage is applied as an increase of percentage, whilst in other cases it is applied as a decremental percentage, which will result in different absolute values (i.e. 100 wei - 5% does not match 95 wei + 5%). The parameters need to be correctly input by the management team in order to get the desired results.
- The helper contract `PriceFetcher` used to query token prices from Chainlink checks the freshness of data with the implementation of the state variable `max_price_staleness`. However, the refresh rate may change amongst data feeds whilst the `PriceFetcher` is used for several of them. Therefore, if the system were to be shared for data feeds with different refresh rates, the current version of the project could not be very precise since it could only check the more permissive refresh rate amongst all feeds.
- The protocol's yield-bearing token generated by the `AggregatorToken`contract has the `pause` functionality affecting token transfers (i.e. `transfer()` and `transferFrom()` methods), locking out users from interacting with any other protocols such as staking or selling their tokens, as well as any other desired purposes.

Findings

Vulnerability Details

F-2024-7337 - Vault Valuation Should Always be Updated before Deposits and Withdrawals - Medium

Description:

In ERC4626 vaults, `deposits` and `withdrawals` are dependant on the amount of underlying tokens the vault contains at the time of such functions executions. This can be seen in the methods `_convertToShares()` and `_convertToAssets()` below:

```
/**
 * @dev Internal conversion function (from assets to shares) with support for rounding direction.
 */
function _convertToShares(uint256 assets, Math.Rounding rounding) internal view virtual returns (uint256) {
    return assets.mulDiv(totalSupply() + 10 ** _decimalsOffset(), totalAssets() + 1, rounding);
}

/**
 * @dev Internal conversion function (from shares to assets) with support for rounding direction.
 */
function _convertToAssets(uint256 shares, Math.Rounding rounding) internal view virtual returns (uint256) {
    return shares.mulDiv(totalAssets() + 1, totalSupply() + 10 ** _decimalsOffset(), rounding);
}
```

In order to contemplate the current amount of underlying token at the moment of each `deposit` and `withdraw` operation, the `totalAssets()` should include the `deployedAssetsValue` at that moment. This is done by overriding `totalAssets()` in the `Vault` contract.

```
function totalAssets() public view override returns (uint256) {
    // check strategy is defined
    _checkZeroAddress(vaultStrategyAddress, "vaultStrategyAddress");

    return IVaultStrategy(vaultStrategyAddress).getDeployedAssetsValue() + pe
```

```
ndingDepositAssets;
}
```

However, the aforementioned `deployedAssetsValue` is only updated during the deployment of capital, or after tokens were removed from the strategy contracts. Otherwise, it can only be updated via calls to `updateDeployedAssetValue()`.

```
/// @notice Updates the valuation of deployed assets based on current prices
/// @return deployedAssetsValue The new valuation of the deployed assets
function updateDeployedAssetVaule() public onlyRole(VAULT_MANAGER_ROLE) retur
ns (uint256) {
    // get adapter
    StepData memory adapter = _getLastAdapter();

    // get value and store it if it is different from before
    uint256 valuation = IAdapter(adapter.adapterAddress).getValuation(address
(this));

    if (valuation != deployedAssetsValue) {
        emit DeployedAssetsValueUpdated(valuation);
        deployedAssetsValue = valuation;
    }

    return deployedAssetsValue;
}
```

The lack of such calls before performing `deposits` and `withdrawals` mean that changes in the vault valuation due to external factors such as token price changes or increased yields will not be reflected. As a result, the minted `shares` during `deposits` and the valuation of such same `shares` in order to withdraw underlying assets from the vaults will not be accurate.

Assets:

- vaultStrategies/BaseStrategy.sol [https://github.com/protofire/lys-protocol/]
- Vault.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Classification

Impact: 3/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Medium

Severity: Medium

Recommendations

Remediation: It is recommended to update the vault valuation before executing `deposits` and `withdrawals`.

Resolution: The development team accepted the finding and the risks arising from it, providing the following explanation:

> Strategy Valuation updates has been decoupled from vault operations (withdraw and deposit) because of scalability reasons.
> Your are right there is some lost in accuracy but compensated between users.
>
> Total assets are composed of two factors:
>
> A. Money deposited into the vault but not invested yet. This is Represented by "pendingDepositAssets"
> B. money invested intro the underlying strategy. This value is read from underlying strategy “deployedAssetsValue”.
>
> We decided that normal operations like deposits and withdrawals cannot have unpredictable gas consumption. Running the actual reevaluation of the entire investment managed by a given strategy is a process that can potentially grow in complexity and varies from one strategy to another. It's a process that can evolve and account for an increasing number of tokens and positions. So we decided that updating that valuation should be a permission-less function that anyone can execute before his operation if they want to operate with the last updated value by increasing accuracy. Or they can relay just in the automated vault management, which means that we are going monitor with an off-chain process the strategies valuations and execute revaluations at some reasonable frequency.
>
> In this way we are just making the same assumption we already have for capital deployments, as it's decoupled from deposits. Nothing guaranties that when a user execute a deposit the money is instantly invested, and this means depositors are "loosing some APY" because they wait for the capitalDeployment execution. TO be consistent ins terms of protocol design, and in the approach, they also have the possibility to force the execution just after they run the deposit function cause capital deployment is also a permission-less vault management function.

At some point capitalDeployment and Update valuations are going to be executed. Being by the automated vault management. Or by the users themselves if they have a big enough incentive to do that. In the meanwhile the "lost" in accuracy of the operations are going to benefit some users and "damage" other users, depending on the context.

Actually we are letting the users decide if they want to operate by optimizing accuracy or by optimizing their contextual benefits. I mean, is not true that it'll be always convenient for depositors or withdrawers to have a previous valuationUpdate before their operations, so they may decide to have it or not

F-2024-7359 - Lack of Slippage can Result in Loss of Funds - Medium

Description:

Slippage tolerances establish a margin of change acceptable to the user beyond price impact. As long as the execution price is within the slippage range (e.g., %1), the transaction will be executed. If the execution price ends up outside of the accepted slippage range, the transaction will fail, and the swap will not occur.

In the Pendle adapter, no slippage was defined in withdrawals. As a consequence, an undesired amount of tokens can be obtained, potentially leading to loss of funds.

```
function _redeemSyAndTransferToStr(
    address owner_,
    IStandardizedYield SY,
    uint256 netSyOut,
    address ptUnderlying
) internal returns (uint256 netTokenOut) {
    netTokenOut = SY.redeem(
        address(this),
        netSyOut,
        ptUnderlying,
        0,
        true
    );
    IERC20(ptUnderlying).transfer(owner_, netTokenOut);
}
```

Assets:

- adapters/pendle/PendleAdapter.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Classification

Impact: 4/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Medium

Severity: Medium

Recommendations

Remediation: It is recommended to introduce a minimum amount of tokens out to be received from the Pendle Swap.

Resolution: The development team accepted the finding and the risks arising from it, providing the following comment:

> When it comes to withdrawals, the slippage is being controlled on a swapper side (we are using uniswap's exactOutput, so we are 100% sure that in the end we will receive the exact amountOut we specified in withdraw args)

F-2024-7362 - Leftover Funds can Become Locked in the Strategy Contract - Medium

Description: During asset withdrawals, some strategies will require a final step where a swap is performed in order to obtain the desired token.

```
function swap(
    address owner_,
    address receiver_,
    uint256 maxAmountIn_,
    uint256 amountOut_,
    address tokenIn_,
    address tokenOut_,
    uint256
)
    external
    nonReentrant
    onlyRole(SWAPPER_EXECUTOR)
    returns (uint256 amountSpent, uint256 amountReceived)
{
    _checkZeroAddress(owner_, "owner_");
    _checkZeroAddress(receiver_, "receiver_");
    _checkZeroAmount(maxAmountIn_, "maxAmountIn_");
    _checkZeroAmount(amountOut_, "amountOut_");
    _checkZeroAddress(tokenIn_, "tokenIn_");
    _checkZeroAddress(tokenOut_, "tokenOut_");

    Dex memory dex = tokenSwapPath[tokenIn_];
    // check swap and dex data
    if (dex.swapSource == address(0)) revert InvalidToken();
    if (dex.target == SwapTarget.None) revert InvalidTarget();

    // pull from strategy
    IERC20(tokenIn_).safeTransferFrom(owner_, address(this), maxAmountIn_);
    /// approve dex to pull funds
    IERC20(tokenIn_).approve(dex.swapSource, maxAmountIn_);

    // redirect call
    if (dex.target == SwapTarget.Uniswap) {
        // make swap on uniswap
        (amountSpent, amountReceived) = _makeUniswapSwap(
            dex,
            tokenIn_,
            maxAmountIn_,
            amountOut_,
            receiver_
```

```
            );
        } else if (dex.target == SwapTarget.Curve) {
            // make swap on cruve
            (amountSpent, amountReceived) = _makeCurveSwap(
                dex,
                tokenIn_,
                maxAmountIn_,
                amountOut_,
                receiver_
            );
        }

        // emit event
        emit SwapExecuted(owner_, receiver_, tokenIn_, maxAmountIn_, amountSpent,
amountOut_);

        // check for leftovers and transfer back to strategy
        // this is not tested
        uint256 leftovers = maxAmountIn_ > amountSpent ? maxAmountIn_ - amountSpe
nt : 0;
        if (leftovers > 0) {
            IERC20(tokenIn_).safeTransfer(owner_, leftovers);
        }

        return (amountSpent, amountReceived);
    }
```

In case there are any `leftovers` from the swap, the remaining tokens will be sent back to the strategy contract. However, the strategy contract cannot handle any tokens, but only some specific tokens. Due to the lack of any method to recover those funds or reinvest them, this may result in a loss of funds, even if it may be a small amount.

Additionally, there is no update of the vault valuation depending on whether or not there is a leftover amount being sent back to the strategy.

Assets:

- _helpers/Swapper.sol [https://github.com/protofire/lys-protocol/]

Status: Fixed

Classification

Impact: 4/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Medium

Severity: Medium

Recommendations

Remediation: There are different strategies that can be used to mitigate the reported scenario, depending on the desired outcome:

- Re-deposit the leftover amount, following the corresponding strategy (a threshold can be setup in order to avoid operating with low amounts).
- Introduce a method to withdraw leftover tokens from the strategy contract.
- Send the leftover amount to the user.

Other alternatives can also be considered valid, depending on the development team requirements.

Resolution: Fixed in commit ID `f03a66f`: the development team introduced the `annihilate()` method into the `strategy` contracts, which will swap the leftover tokens to `wETH`, send them back to the `Vault` and update the `pendingDepositAssets`.

```
function annihilate(
    address swapper,
    address vault,
    address[] memory tokensIn,
    address tokenOut,
    uint256[] memory maxAmountsIn,
    uint256[] memory amountsOut
) external nonReentrant onlyRole(DEFAULT_ADMIN_ROLE) {
    _checkZeroAddress(swapper, "swapper");
    _checkZeroAddress(vault, "vault");
    _checkZeroAddress(tokenOut, "tokenOut");

    uint256 totalReceived = _processSwaps(swapper, tokensIn, tokenOut, maxAmo
untsIn, amountsOut);

    // Transfer the received tokens to the vault and update pendingDepositAss
ets on the vault
    IERC20(tokenOut).transfer(vault, totalReceived);
    IVault(vault).addLeftovers(totalReceived);
}

function _processSwaps(
```

```
    address swapper,
    address[] memory tokensIn,
    address tokenOut,
    uint256[] memory maxAmountsIn,
    uint256[] memory amountsOut
) internal returns (uint256 totalReceived) {
    for (uint256 i = 0; i < tokensIn.length; ++i) {
        _checkZeroAddress(tokensIn[i], "token");
        _checkZeroAmount(maxAmountsIn[i], "maxAmountIn");
        _checkZeroAmount(amountsOut[i], "amountOut");

        // Approve and execute the swap
        IERC20(tokensIn[i]).approve(swapper, maxAmountsIn[i]);
        (, uint256 amountReceived) = ISwapper(swapper).swap(
            address(this),
            address(this),
            maxAmountsIn[i],
            amountsOut[i],
            tokensIn[i],
            tokenOut,
            0
        );

        totalReceived += amountReceived;
    }
}

function addLeftovers(
    uint256 leftovers_
) external onlyRole(VAULT_MANAGER_ROLE){
    _checkZeroAmount(leftovers_, "leftovers_");

    pendingDepositAssets += leftovers_;

    uint256 vaultValuation = totalAssets();
    emit VaultValuationUpdated(vaultValuation);
}
```

F-2024-7126 - Outdated Price Usage due to Lack of Price Oracle Consistency - Low

Description:

The `_getTokenPrice()` function calls out to a Chainlink oracle receiving the `latestRoundData()`. The current implementation only verifies that the retrieved price is greater than zero:

```
function _getTokenPrice(address token_) internal view returns (uint256) {
    ...
    (, int256 price, , , ) = AggregatorV3Interface(priceData.contractLocation
)
        .latestRoundData();
    require(price > 0, "Invalid price data");
    ...
}
```

The current check ensures the price is positive, it does not confirm that the data is up-to-date. If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g. Chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the chainlink system) consumers of this contract may continue using outdated stale or incorrect data (if oracles are unable to submit no new round is started).

Outdated results may directly affect the system's prices.

Assets:

- _helpers/PriceFetcher.sol [https://github.com/protofire/lys-protocol/]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Medium

Severity: Low

Recommendations

Remediation: To mitigate the risk of using stale or incorrect data from Chainlink oracles, implement additional checks to make sure the acquired result is the latest one. Specifically, validate the timestamp of the latest data against the current block timestamp, ensure that the data retrieval round has been fully finalized and is not still in progress, and verify that the data comes from the latest or a more recent round.

```
(uint80 roundID, int256 answer, uint256 timestamp , uint256 updatedAt, uint80
answeredInRound) =
,→ priceFeed.latestRoundData();
require(timestamp != 0,"Round not complete");
require(updatedAt >= block.timestamp - feedRefreshRate, "Stale Price");
require(answer > 0, "Invalid Price");
```

Note that each data feed has it's own refresh rate, which may differ from others.

Resolution: Fixed in commit ID `43b081d`: the development team included the following checks in the Chainlink Oracle calls.

```
require(price > 0, "Invalid price data");
// Ensure the data is up-to-date (not stale)
uint256 timeElapsed = block.timestamp - updatedAt;
if(timeElapsed > max_price_staleness){
    revert StaleData();
}
```

F-2024-7140 - Selected Vaults in Convergent Portfolio Transfer may not Match Actual Portfolio - Low

Description:

When performing a transfer of vault shares via a convergent portfolio transfer strategy, the system predefines the vaults that will be screened manually:

```
function setVaultsForNextTransfer(
    address[] calldata vaults_
) external onlyRole(LYSADMIN_MANAGER_ROLE) {
    // Update the vaults in storage
    selectedVaults = vaults_;

    emit VaultsUpdated(vaults_);
}
```

These selected vaults are used as the source for iteration among users' portfolios to subtract shares from them and also to perform some calculations. However, there is no guarantee that the selected vaults will match the user's actual vaults since this amount is common for anyone using this transfer strategy.

```
function _executeTransferStrategy(
    address from_,
    uint256 shares_
) internal view returns (address[] memory, uint256[] memory) {
    address[] memory usedVaults = new address[](selectedVaults.length);
    uint256[] memory usedShares = new uint256[](selectedVaults.length);

    uint256 remainingShares = shares_;
    uint256 totalVaults = selectedVaults.length;

    // Distribute shares across the selected vaults
    for (uint256 i = 0; i < totalVaults; i++) {
        uint256 vaultBalance = aggregatorToken.vaultBalanceOf(from_, selected
Vaults[i]);
        uint256 transferAmount;
        if (i == totalVaults - 1) {
            // For the last vault, transfer all remaining shares to ensure al
l shares are distributed
            transferAmount = remainingShares;
        } else {
            // Distribute remaining shares proportionally
            transferAmount = remainingShares / (totalVaults - i);
        }
```

```
        // Ensure we don't transfer more than the vault balance
        if (transferAmount > vaultBalance) {
            transferAmount = vaultBalance;
        }

        usedVaults[i] = selectedVaults[i];
        usedShares[i] = transferAmount;

        // Decrease the remaining shares by the amount that will be transferr
ed
        remainingShares -= transferAmount;
    }

    return (usedVaults, usedShares);
}
```

As a consequence, the numbers may be different than expected or result in unexpected behavior.

Assets:

- transferStrategy/ConvergentPortfolioTransferStrategy.sol [https://github.com/protofire/lys-protocol/]

Status: Mitigated

Classification

Impact: 2/5

Likelihood: 3/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: Consider using the vault that each user has in its portfolio to perform the convergent transfer strategy.

Resolution: The development team mitigated the finding and the risks arising from it, providing the following comment:

> This one was designed to handle DEX transfers and should only be considered in conjunction with an off-chain

component. The off-chain component compares global portfolio weights with the pair's portfolio after each transfer and determines which vaults will be used for subsequent transfers. The off-chain component guaranties the match. Also, this feature is still under development.

F-2024-7347 - Aggregator Fee can Surpass 100% - Low

Description:

The `aggregatorFee` set up in the `BaseAggregatorToken` contract is defined as follows:

```
function setAggregatorFee(
    uint256 aggregatorFee_
) external onlyRole(FEE_MANAGER_ROLE) {
    aggregatorFee = aggregatorFee_;
    emit AggregatorFeeSet(aggregatorFee_);
}
```

It is later used to calculate the amount of tokens that users will pay to the protocol during `deposit()`:

```
function _calculateAggregatorFee(uint256 value_) internal view returns (uint256) {
    return aggregatorFee == 0 ? 0 : (value_ * aggregatorFee) / BASIS_POINTS_DENOMINATOR;
}
```

However, due to the lack of an upper limit, that at maximum be set to `BASIS_POINTS_DENOMINATOR`, it could surpass `100%`.

Assets:

- BaseAggregatorToken.sol [https://github.com/protofire/lys-protocol/]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 3/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to define a maximum fee that is lower than `BASIS_POINTS_DENOMINATOR` (e.g. 10%, 25%).

Resolution: Fixed in commit ID `43b081d`: the check was introduced into the method `setAggregatorFee()` to prevent the fee from surpassing 100%.

```
if(aggregatorFee_ > BASIS_POINTS_DENOMINATOR){
    revert MaxFee();
}
```

Observation Details

F-2024-6803 - Floating Pragma - Info

Description:

In Solidity development, the pragma directive specifies the compiler version to be used, ensuring consistent compilation and reducing the risk of issues caused by version changes. However, using a floating pragma (e.g., `^0.8.xx`) introduces uncertainty, as it allows contracts to be compiled with any version within a specified range. This can result in discrepancies between the compiler used in testing and the one used in deployment, increasing the likelihood of vulnerabilities or unexpected behavior due to changes in compiler versions.

The project currently uses floating pragma declarations (`^0.8.24`) in its Solidity contracts. This increases the risk of deploying with a compiler version different from the one tested, potentially reintroducing known bugs from older versions or causing unexpected behavior with newer versions. These inconsistencies could result in security vulnerabilities, system instability, or financial loss. Locking the pragma version to a specific, tested version is essential to prevent these risks and ensure consistent contract behavior.

Assets:

- _helpers/Swapper.sol [https://github.com/protofire/lys-protocol/]
- interfaces/ISwapper.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation:

It is recommended to **lock the pragma version** to the specific version that was used during development and testing. This ensures that the contract will always be compiled with a known, stable compiler version, preventing unexpected changes in behavior due to compiler updates. For example, instead of using `^0.8.24`, explicitly define the version with `pragma solidity 0.8.24;`.

Before selecting a version, review known bugs and vulnerabilities associated with each Solidity compiler release. This can be done by referencing the official Solidity compiler release notes: Solidity GitHub releases or Solidity Bugs by Version. Choose a compiler version with a good track record for stability and security.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-6993 - Typo in Comment - Info

Description: There is a couple of spelling errors in the function name and the comment for the description of granting roles.

- `toles` → `roles`

```
function initialize(
    ...
) external initializer {
    ...
    // manager toles to owner and master token
    ...
}
```

- `whitdrawAssets` → `withdrawAssets`

```
function _whitdrawAssets(
    address owner_,
    uint256 assets_,
    address receiver_
) internal returns (uint256, uint256) {
    ...
}
```

Assets:

- Vault.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation: Fix the typos for the aforementioned comment and function to prevent any confusions.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-7143 - Missing Zero Address Checks - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "zero address". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address. The "
Missing zero address control" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.
For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

The lack of zero address checks may result in different scenarios:

- Waste of Gas: `contracts/transferStrategy/ConvergentPortfolioTransferStrategy.sol: setVaultsForNextTransfer()`
- Incorrectly working functionality: `contracts/adapters/pendle/PendleAdapter.sol: updateNextDepositMarket(), updateNextWithdrawMarket(), setPendleStrategy(), setMarketData().`

Assets:

- transferStrategy/ConvergentPortfolioTransferStrategy.sol [https://github.com/protofire/lys-protocol/]
- adapters/pendle/PendleAdapter.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation: Introduce checks to prevent the reported address from being introduced as `address(0)`.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-7144 - Upgradeable Contracts can be Initialized by External Actors - Info

Description:

The protocol implements several upgradable contracts. After deploying, or upgrading, such contracts on the blockchain, the development team must invoke their `initialize()` function to set up basic functionalities. However, the absence of a call to `_disableInitializers()` in the `constructor()` of the implementation contracts exposes a vulnerability where external actors can directly initialize the implementation contract, potentially disrupting the intended upgrade flow.

Assets:

- adapters/pendle/PendleAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/frax/FraxAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/renzo/RenzoAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/swell/SwellStakingAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/swell/SwellReStakingAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/lido/LidoAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/etherfi/EtherFiAdapter.sol [https://github.com/protofire/lys-protocol/]
- Vault.sol [https://github.com/protofire/lys-protocol/]
- AggregatorToken.sol [https://github.com/protofire/lys-protocol/]
- transferStrategy/ConvergentPortfolioTransferStrategy.sol [https://github.com/protofire/lys-protocol/]
- VaultsRegistry.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/NStepsStrategy.sol [https://github.com/protofire/lys-protocol/]
- _helpers/Wrapper.sol [https://github.com/protofire/lys-protocol/]
- transferStrategy/UniformTransferStrategy.sol [https://github.com/protofire/lys-protocol/]
- _helpers/PriceFetcher.sol [https://github.com/protofire/lys-protocol/]
- _helpers/Swapper.sol [https://github.com/protofire/lys-protocol/]
- adapters/stader/StaderAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/kelp/KelpAdapter.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/PendleStrategy.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation: It is recommended to include a call to `_disableInitializers()` in the `constructor()` of the implementation contract. This will prevent any direct initialization of the implementation contract and ensure the security of the upgrade process.

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
   _disableInitializers();
}
```

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-7146 - Unused Code Decreases Code Quality - Info

Description:

In Solidity, functions often have arguments that are intended for specific operations or logic within the function. However, sometimes these arguments remain unused in the function body, either due to changes during development or oversight. Unused variables can lead to confusion, making the code less readable and potentially misleading for other developers who might use or audit the contract.

Moreover, they can create a false impression of the function's purpose and behavior. It's crucial to either implement these arguments in the function's logic as originally intended or remove them to ensure clarity and efficiency of the code.

The following imports are integrated into the protocol contracts but are unused.

`contracts/transferStrategy/ConvergentPortfolioTransferStrategy.sol: IERC20.sol, IERC4626.sol`

The following state variables are unused:

`contracts/transferStrategy/ConvergentPortfolioTransferStrategy.sol: SCALING_FACTOR.`

`contracts/BaseAggregatorToken.sol: maxVaultsPerSecondaryMartketPair.`

`contracts/VaultsRegistry.sol: defaultFeeRate.`

The following Roles are set up but never used:

`contracts/AggregatorToken.sol: VAULTS_MANAGER_ROLE, VALUATIONS_MANAGER_ROLE.`

`contracts/BaseAggregatorToken.sol:VAULTS_MANAGER_ROLE, VALUATIONS_MANAGER_ROLE.`

The following functions are not used:

`contracts/BaseAggregatorToken.sol: receive().`

The following contracts include unused the following testing code:

`contracts/transferStrategy/ConvergentPortfolioTransferStrategy.sol`

`contracts/transferStrategy/UniformTransferStrategy.sol`

`contracts/Vault.sol`

`contracts/vaultStrategies/PendleStrategy.sol`

`contracts/vaultStrategies/NStepsStrategy.sol`

`contracts/vaultStrategies/BaseStrategy.sol`

`contracts/adapters/BaseAdapter.sol`

`contracts/adapters/swell/SwellReStakingAdapter.sol`

`contracts/adapters/swell/SwellStakingAdapter.sol`

`contracts/adapters/stader/StaderAdapter.sol`

`contracts/adapters/renzo/RenzoAdapter.sol`

`contracts/adapters/pendle/PendleAdapter.sol`

`contracts/adapters/lido/LidoAdapter.sol`

`contracts/adapters/kelp/KelpAdapter.sol`

`contracts/adapters/frax/FraxAdapter.sol`

`contracts/adapters/etherfi/EtherFiAdapter.sol`

```
// import {console} from "hardhat/console.sol";
```

Assets:

- vaultStrategies/PendleStrategy.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/BaseStrategy.sol [https://github.com/protofire/lys-protocol/]
- adapters/kelp/KelpAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/stader/StaderAdapter.sol [https://github.com/protofire/lys-protocol/]
- transferStrategy/UniformTransferStrategy.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/NStepsStrategy.sol [https://github.com/protofire/lys-protocol/]
- transferStrategy/ConvergentPortfolioTransferStrategy.sol [https://github.com/protofire/lys-protocol/]
- AggregatorToken.sol [https://github.com/protofire/lys-protocol/]
- adapters/BaseAdapter.sol [https://github.com/protofire/lys-protocol/]
- Vault.sol [https://github.com/protofire/lys-protocol/]
- BaseAggregatorToken.sol [https://github.com/protofire/lys-protocol/]
- adapters/etherfi/EtherFiAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/lido/LidoAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/swell/SwellReStakingAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/swell/SwellStakingAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/renzo/RenzoAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/frax/FraxAdapter.sol [https://github.com/protofire/lys-protocol/]
- adapters/pendle/PendleAdapter.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation: It is recommended to remove unused code from the contracts.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-7147 - Missing Storage Gaps in Base Contracts - Info

Description:

When working with upgradeable contracts, it is necessary to introduce storage gaps into the base contracts to allow for storage extension during upgrades.

Storage gaps are a convention for reserving storage slots in a base contract, allowing future versions of that contract to use up those slots without affecting the storage layout of child contracts.

Note: OpenZeppelin Upgrades checks the correct usage of storage gaps.

Assets:

- BaseAggregatorToken.sol [https://github.com/protofire/lys-protocol/]
- adapters/BaseAdapter.sol [https://github.com/protofire/lys-protocol/]
- BaseVault.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/BaseStrategy.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation:

Introduce Storage Gaps in the affected contracts.

To create a storage gap, declare a fixed-size array in the base contract with an initial number of slots. This can be an array of `uint256` so that each element reserves a 32 byte slot. Use the name `__gap` or a name starting with `__gap_` for the array so that OpenZeppelin Upgrades will recognize the gap.

To help determine the proper storage gap size in the new version of your contract, you can simply attempt an upgrade using `upgradeProxy` or just run the validations with `validateUpgrade` (see docs for Hardhat or Truffle). If a storage gap is not being reduced properly, you will see an error message indicating the expected size of the storage gap.

Resolution:

The development team accepted the finding and the risks arising from it.

F-2024-7302 - Unsafe Use of transfer()/transferFrom() - Info

Description:

Some major tokens went live before ERC20 was finalised, resulting in a discrepancy whether the transfer functions should either return a boolean or revert/fail on error. The current best practice is that they should revert, but return `true` on success. However, not every token claiming ERC20-compatibility is doing this — some only return `true`/`false`; some revert, but do not return anything on success.

While the ERC20 transfer functionality is only implemented in the `transferToken` function, a sound feedback about the transfer's success would be beneficial.

In the `AggregatorToken` contract, the `deposit()`, `withdraw()` and `withdrawFees` functions rely on the standard `transferFrom()` and `transfer()` methods of the ERC20 token interface (`IERC20`). However, these standard methods pose risks due to varying implementations across different ERC20 tokens.

In `Vault`, the methods `increaseCapital()` and `decreaseCapital()` use `transferFrom()` and `transfer()` respectively.

The same for the `_transferAndApproveERC20()`, `redeemSyAndTransferToStr()`, and `_swapToSy()` functions in the `PendleAdapter` contract.

Assets:

- AggregatorToken.sol [https://github.com/protofire/lys-protocol/]
- adapters/pendle/PendleAdapter.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation:

The best practice for handling ERC20 transfers is to make sure that the transfer functions revert on error and return `true` on success. This is to ensure that the transfer is successful and to prevent any potential security issues.

However, since some tokens do not return any value at all, it is suggested to use a wrapper like OpenZeppelin's SafeERC20 library when dealing with ERC20 transfers, in order to have a clear feedback about the transfers' success.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-7303 - TODO Comment Affect The Code Quality - Info

Description:

Several contracts in the project, including `BaseVault`, `Vault`, `PendleAdapter`, `NStepsStrategy`, and `PendleStrategy` contracts have been identified to contain 'TODO' comments.

`TODO` comments are typically used by developers as reminders for features, optimizations, or fixes that should be addressed at a later stage. These comments should ideally not be present in production-ready code as they can indicate unresolved tasks, possible incomplete features, or areas that require further attention.

Assets:

- adapters/pendle/PendleAdapter.sol [https://github.com/protofire/lys-protocol/]
- BaseVault.sol [https://github.com/protofire/lys-protocol/]
- Vault.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/PendleStrategy.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/NStepsStrategy.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation:

- Review all `TODO` comments in the identified contracts.
- Address and implement the necessary changes based on the comments, if required.
- If a `TODO` comment is outdated or no longer relevant, remove it from the code.
- Ensure that any action taken or removal of the `TODO` comment aligns with the project's current requirements and goals.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-7304 - Missing Events in Key Functions - Info

Description:

Sensitive functions that make state changes or update critical variables lack event emissions to log these changes. Event emissions are critical in blockchain applications for tracking state changes, providing transparency, and enabling off-chain systems to monitor contract activity. Without events, changes made by these functions are harder to detect, which could result in diminished auditability and potential misuse going unnoticed.

The following functions lack event emissions:

- `PendleAdapter.sol` - `setPendleStrategy()` and `setMarketData()`
- `AggregatorToken.sol` - `setName()` and `setSymbol()`

Lack of event emissions reduces transparency and limits usability for off-chain systems that rely on events to track state changes.

Assets:

- AggregatorToken.sol [https://github.com/protofire/lys-protocol/]
- adapters/pendle/PendleAdapter.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation:

Add event emissions to all affected functions to log sensitive changes. Emit relevant data for transparency and off-chain monitoring.

Resolution:

The development team accepted the finding and the risks arising from it.

F-2024-7334 - Inconsistent Definition of the Withdraw Strategy - Info

Description:

The `withdrawStrategyAddress` is set up both in the `BaseVault` and the `BaseStrategy` contracts:

- BaseVault.sol:

```
/// @notice Sets the address of the withdrawal strategy.
/// @param withdrawStrategyAddress_ The new strategy address for handling wit
hdrawals.
function setWithdrawStrategyAddress(
    address withdrawStrategyAddress_
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    _checkZeroAddress(withdrawStrategyAddress_, "withdrawStrategyAddress_");

    emit AddressUpdated("withdrawStrategyAddress", withdrawStrategyAddress_);

    withdrawStrategyAddress = withdrawStrategyAddress_;
}
```

- BaseStrategy.sol:

```
/// @notice Sets the address for the withdrawal strategy
/// @param withdrawStrategyAddress_ New address for the withdrawal strategy
function setWithdrawStrategyAddress(
    address withdrawStrategyAddress_
) external virtual onlyRole(DEFAULT_ADMIN_ROLE) {
    _checkZeroAddress(withdrawStrategyAddress_, "withdrawStrategyAddress_");

    emit AddressUpdated("withdrawStrategyAddress", withdrawStrategyAddress_);

    withdrawStrategyAddress = withdrawStrategyAddress_;
}
```

This may result in an inconsistency due to the duplication of this contract feature, since the aforementioned variable is checked from different sources during the contract flow. For example, in the following function from the `Vault` contract, the zero address check is on the variable defined in the same `BaseVault` contract whilst the actual address called is the one defined in the `BaseStrategy`. As a consequence, the same variable may have different values defined in the different contracts, leading to unexpected behavior.

```
function _whitdrawAssets(
    address owner_,
    uint256 assets_,
    address receiver_
) internal returns (uint256, uint256) {
    // check withdrawStrategyAddress is defined
    _checkZeroAddress(withdrawStrategyAddress, "withdrawStrategyAddress");
    // shares amount are checked on masterToken
    // shares amount are checked on masterToken

    uint256 exactAssetAmount;
    uint256 exactShares;
    // this checks if withdraw needs to call adapters or not
    if (pendingDepositAssets >= assets_) {
        // there is enough underlying to cover the withdraw
        exactAssetAmount = assets_;

        // get shares for exact assets amount
        exactShares = previewWithdraw(exactAssetAmount);

        // update pending deposit assets
        pendingDepositAssets -= assets_;

        // call standard vault withdraw
        super._withdraw(owner_, receiver_, owner_, exactAssetAmount, exactSha
res);
    } else {//note if the contract lacks the tokens, withdraw from adapters
        // store the total assets to get the shares right
        uint256 previousTotalAssets = totalAssets();

        // there is NOT enough underlying to conver... call the strategy to w
ithdraw
        uint256 assetsAmount = IWithdrawStrategy(withdrawStrategyAddress)
            .executeWithdrawStrategy(withdrawStrategyAddress, receiver_, asse
t(), assets_);

        // _checkZeroAmount(assetsAmount, "assetsAmount");

        // return the exact amount
        exactAssetAmount = assetsAmount;

        // get shares for actual received amount
        exactShares = _customConvertToShares(exactAssetAmount, previousTotalA
ssets);

        // emit event here because it is not withdrawing from standard
        emit Withdraw(owner_, receiver_, owner_, exactAssetAmount, exactShare
s);
```

```
        // burn master token shares
        _burn(owner_, exactShares);
    }

    // get total assets to emit event
    uint256 vaultValuation = totalAssets();
    emit VaultValuationUpdated(vaultValuation);

    return (exactAssetAmount, exactShares);
}
```

Assets:

- BaseVault.sol [https://github.com/protofire/lys-protocol/]
- Vault.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/BaseStrategy.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation: It is recommended to define and read the variable from a single contract instead of different sources. Alternatively, a check could be implemented to ensure both addresses match before performing critical operations.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-7338 - Inefficient Withdrawal of Assets from Vaults - Info

Description:

As observed from the development team strategy when a user requests to `withdraw` their assets, it is a priority to withdraw underlying asset tokens from the Vault before reaching out to the deployed capital. This decision is based on whether or not the `pendingDepositAssets` is enough to cover the requested `assets` to withdraw.

However, if `pendingDepositAssets > 0` the defined logic will directly turn over the adapters to withdrawing deployed assets the full amount to withdraw instead of withdrawing the remaining `pendingDepositAssets` and then retrieving the missing amount from the external protocols.

For example, if the requested amount to withdraw is 3000, and `pendingDepositAssets = 2500` the current logic would withdraw all 3000 tokens from the adapters instead of taking 2500 from the vault and the remaining 500 from the adapters.

```
function _whitdrawAssets(
    address owner_,
    uint256 assets_,
    address receiver_
) internal returns (uint256, uint256) {
    // check withdrawStrategyAddress is defined
    _checkZeroAddress(withdrawStrategyAddress, "withdrawStrategyAddress");
    // shares amount are checked on masterToken
    // shares amount are checked on masterToken

    uint256 exactAssetAmount;
    uint256 exactShares;
    // this checks if withdraw needs to call adapters or not
    if (pendingDepositAssets >= assets_) {
        // there is enough underlying to cover the withdraw
        exactAssetAmount = assets_;

        // get shares for exact assets amount
        exactShares = previewWithdraw(exactAssetAmount);

        // update pending deposit assets
        pendingDepositAssets -= assets_;

        // call standard vault withdraw
        super._withdraw(owner_, receiver_, owner_, exactAssetAmount, exactSha
res);
    } else {//note if the contract lacks the tokens, withdraw from adapters
        // store the total assets to get the shares right
```

```
        uint256 previousTotalAssets = totalAssets();

        // there is NOT enough underlying to conver... call the strategy to w
ithdraw
        uint256 assetsAmount = IWithdrawStrategy(withdrawStrategyAddress)
            .executeWithdrawStrategy(withdrawStrategyAddress, receiver_, asse
t(), assets_);

        // _checkZeroAmount(assetsAmount, "assetsAmount");

        // return the exact amount
        exactAssetAmount = assetsAmount;

        // get shares for actual received amount
        exactShares = _customConvertToShares(exactAssetAmount, previousTotalA
ssets);

        // emit event here because it is not withdrawing from standard
        emit Withdraw(owner_, receiver_, owner_, exactAssetAmount, exactShare
s);

        // burn master token shares
        _burn(owner_, exactShares);
    }

    // get total assets to emit event
    uint256 vaultValuation = totalAssets();
    emit VaultValuationUpdated(vaultValuation);

    return (exactAssetAmount, exactShares);
}
```

Assets:

- Vault.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation: Consider checking if `pendingDepositAssets >= 0` to decide where to withdraw the assets from. It may be reasonable to use a certain threshold too, instead of 0, to avoid wasting Gas on complex operations that will result in small returns.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-7348 - NatSpec Roles do not Match Actual Roles - Info

Description: The methods `setMaxVaultsPerSecondaryMartketPair()`, `setMaxVaultsPerHolder()` and `setTransferStrategy()` are only callable by the `LYSADMIN_MANAGER_ROLE`. However, their NatSpec refers to `PORTFOLIO_MANAGER_ROLE` and `TRANSFERS_MANAGER_ROLE`.

```
/**
 * @dev Sets the transfer strategy contract address. Can only be called by ac
counts with the TRANSFERS_MANAGER_ROLE.
 * @param transferStrategy_ The address of the new transfer strategy contract
.
 */
function setTransferStrategy(
    address transferStrategy_,
    bool isSecondaryMarket
) external onlyRole(LYSADMIN_MANAGER_ROLE) {
    transferStrategy[isSecondaryMarket] = transferStrategy_;
    emit TransferStrategySet(transferStrategy_, isSecondaryMarket);
C}
```

Assets:

- BaseAggregatorToken.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation: Consider updating the NatSpec comments of the reported functions.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-7351 - Missing Initialization of Base Contracts - Info

Description: The following contracts lack the initialization of their base contracts:

- `contracts/vaultStrategies/NStepsStrategy.sol: __pausable_init(), __ReentrancyGuard_init()`
- `contracts/vaultStrategies/PendleStrategy.sol: __pausable_init(), __ReentrancyGuard_init()`
- `contracts/_helpers/Wrapper.sol: __ReentrancyGuard_init()`
- `contracts/OK_AggregatorToken.sol: __ReentrancyGuard_init()`

Assets:

- AggregatorToken.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/PendleStrategy.sol [https://github.com/protofire/lys-protocol/]
- vaultStrategies/NStepsStrategy.sol [https://github.com/protofire/lys-protocol/]
- _helpers/Wrapper.sol [https://github.com/protofire/lys-protocol/]

Status: Accepted

Recommendations

Remediation: Base contracts should be initialized by derived contracts.

Resolution: The development team accepted the finding and the risks arising from it.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project’s performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/protofire/lys-protocol
Commit	d304f9291e52664c0910940f312893c3b88ab90d
Whitepaper	link
Requirements	link
Technical Requirements	link

Asset	Type
_helpers/Helpers.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
_helpers/PriceFetcher.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
_helpers/Swapper.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
_helpers/Wrapper.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/BaseAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/etherfi/EtherFiAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/etherfi/interfaces/IEtherFiLiqudityPool.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/etherfi/interfaces/IEtherFiLiquifier.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/etherfi/interfaces/IEtherFiWithdrawRequestNFT.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/etherfi/interfaces/IweETH.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/frax/FraxAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/frax/interfaces/IFraxMinter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/frax/interfaces/ISfrxETH.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/kelp/interfaces/IKelpProtocol.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/kelp/KelpAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract

Asset	Type
adapters/lido/interfaces/ILidoProtocol.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/lido/LidoAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/pendle/interfaces/IPendle.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/pendle/interfaces/IPPrincipalToken.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/pendle/interfaces/IPYieldToken.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/pendle/interfaces/IStandardizedYield.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/pendle/PendleAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/renzo/interfaces/IRenzoLiquifier.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/renzo/RenzoAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/stader/interfaces/IStaderStakePoolsManager.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/stader/StaderAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/swell/interfaces/ISwellProtocol.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/swell/SwellReStakingAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
adapters/swell/SwellStakingAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
AggregatorToken.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
BaseAggregatorToken.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
BaseVault.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IAggregatorToken.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IBaseAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IBaseStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract

Asset	Type
interfaces/IBaseVault.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/ICurvePool.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IGenericWrapper.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IPendleAdapter.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IPendleStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IPriceFetcher.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/ISwapper.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/ITransferStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IUniswapV3Protocol.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IVault.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IVaultsRegistry.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IVaultStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IWETH.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IWithdrawStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
interfaces/IWrapper.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
transferStrategy/ConvergentPortfolioTransferStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
transferStrategy/UniformTransferStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
Vault.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
VaultsRegistry.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
vaultStrategies/BaseStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract
vaultStrategies/NStepsStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract

Asset	Type
vaultStrategies/PendleStrategy.sol [https://github.com/protofire/lys-protocol/]	Smart Contract

Appendix 3. Additional Valuables

Verification of System Invariants

During the audit of Lys Protocol, Hacken followed its methodology by performing fuzz-testing on the project's main functions. Foundry, a tool used for smart contracts development and testing, was employed to check how the protocol behaves under various inputs. Due to the complex and dynamic interactions within the protocol, unexpected edge cases might arise. Therefore, it was important to use fuzz-testing to ensure that several system invariants hold true in all situations.

Fuzz-testing allows the input of many random data points into the system, helping to identify issues that regular testing might miss. A specific Echidna fuzzing suite was prepared for this task, and throughout the assessment, 3 invariants were tested over 773 runs. This thorough testing ensured that the system works correctly even with unexpected or unusual inputs.

Invariant	Test Result	Run Count
Uniform portfolio transfer strategy is correctly performed	Passed	10017
Convergent portfolio transfer strategy is correctly performed	Passed	10017
User balance is correctly updated while depositing to the aggregator token	Passed	10017
User balance is correctly updated while withdrawing from the aggregator token	Passed	10017
User balances and shares are correctly updated while transferring aggregator tokens	Passed	10017
Depositing should be reverted if the vault passed is not active	Passed	10017
Removing a vault with zero balance from senders portfolio is correctly performed	Passed	10017
Withdrawals by token receivers is correctly performed	Passed	10017
Depositing to vaults is correctly performed	Passed	10017
Withdrawing from vaults is correctly performed	Passed	10017
Deploying a new vault on the vault registry is correctly performed	Passed	10017

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.