code4rena

AmplifiedFi

Smart Contract Security Assessment

Version 2.0

Audit dates: Sep 18 — Sep 27, 2024

Audited by: koolexcrypto
0xadriii

Contents

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About AmplifiedFi

Amplified Protocol serves as a Liquidity & Yield Layer for Liquid Staking Derivatives (LST/LRT), enabling autonomous on-chain structured products that optimize liquidity and yield generation. By integrating seamlessly with leading LSD providers, Amplified automates complex DeFi strategies, including lending, AMMs, leveraged trading, and derivatives management, providing a simplified pathway for users to access high-yield opportunities.

2.2 Scope

Repository	Amplifiedfi
Commit Hash	c734851a0e8f119ee51273279f6b271ab15267ba
Mitigation Hash	d1190ed34a394b6a407b509169ac1ad1068a271e

2.3 Audit Timeline

DATE	EVENT
Sep 18, 2024	Audit start
Sep 27, 2024	Audit end
Nov 20, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	2
High Risk	3
Medium Risk	5
Low Risk	6
Informational	0
Total Issues	16

3. Findings Summary

ID	DESCRIPTION	STATUS
C-1	Incorrect conversion logic between shares and assets makes yield be incorrectly distributed among depositors	Resolved

C-2	Conversion between shares and assets is bricked, leading to loss of funds	Resolved
H-1	Premature Share Calculation in Deposit Function Causes Incorrect Minting for Fee-on-Transfer Tokens	Acknowledged
H-2	Incorrect Accounting for Fee-on-Transfer Tokens Leads to Share Overvaluation and Potential Vault Insolvency	Acknowledged
H-3	Incorrect parameter in `TokenAmount` struct might Dos Pendle interactions	Resolved
M-1	Risk of Unexpected Reverts Due to Self-Transfer in AaveATokenAdaptor Withdraw Function	Resolved
M-2	Lack of slippage protection when redeeming SY tokens from Pendle	Resolved
M-3	Inadequate Safeguards in Adaptor Removal Process Risking Asset Lock-up	Resolved
M-4	DoS in functionality for ETH deposits by directly transferring to the contract	Resolved
M-5	Missing pausability checks in deposit and minting flows	Resolved
L-1	Redundant Zero-Check Assertion in PendleAdaptor's mintSyFromToken	Resolved
L-2	Unbounded loops in StrategyManager	Resolved
L-3	Lack of Interface Validation in AdaptorConfigurator's addAdaptor Function	Acknowledged
L-4	Incorrect number of topics emitted in ERC4626's `Deposit()` event	Resolved
L-5	Current implementation is incompatible with fee-on-transfer tokens	Acknowledged
L-6	Potential DoS for pools with low observation cardinality	Acknowledged

4. Findings

4.1 Critical Risk

A total of 2 critical risk findings were identified.

[C-1] Incorrect conversion logic between shares and assets makes yield be incorrectly distributed among depositors

Severity: Critical	Status: Resolved

Context:

- [ERC4626.sol#L425]

Description: Currently, the conversion between shares and assets is not done following usual ERC4626 computations. Instead, a mints/shares ratio is used. When users deposit into the ERC4626 vault, the amount of shares minted to the user will be equivalent to the amount of assets deposited. However, an internal `mints` tracker will also be increased, which will track the amount of "mints" (LP token) obtained after depositing into the external yield protocol:

```
// File: ERC4626.sol
function _deposit(
        uint256 assets,
        address receiver
    ) internal nonReentrant returns (uint256 shares) {
        ...

        uint256 mints = IERC20(assetOut).balanceOf(address(this));
        // slither-disable-next-line unused-return
        Address.functionDelegateCall(
            _adaptor(),
            _makeDepositAdaptorCallData(amountOut)
        );
        mints = IERC20(assetOut).balanceOf(address(this)) - mints;
        // slither-disable-next-line incorrect-equality
        if (mints == 0) {
            revert ZeroMintAmount(proxy, assetIn, assetOut, amountOut);
        }
        _increaseTotalMints(mints);
        shares = assets;
        _mint(receiver, shares);
        emitViaProxy_Deposit(proxy, sender, receiver, assets, shares);
```

```
    }
```

Then, when performing withdrawals, an approach similar to the usual ERC4626 conversions will be used to obtain the amount of assets that actually belongs to the user, where the `mints` to deduct will be given by the `(shares * _totalMints()) / _totalSupply();` formula:

```
// File: ERC4626.sol
 function _withdraw(
        uint256 assets_,
        address receiver_,
        address owner_
    ) internal nonReentrant returns (uint256 shares) {
        StrategyAdaptorStorage storage $ = _getStrategyAdaptorStorage();
        shares = assets_;
        ...
        uint256 mints = (shares * _totalMints()) / _totalSupply();
        ...
        _burn(owner_, shares);
        ...
        _decreaseTotalMints(mints);
       ...
}
```

The problem with this approach is that it does not consider changes in total assets held in the protocol, and could lead to the following scenario:

1. User 1 deposits 1 ETH, and obtains 1e18 shares. After conversions and swaps, 1e18 SY tokens are obtained (we assume an initial 1:1 rate for SY token in Pendle).
2. The SY token rate increases, and 1e18 SY tokens are now worth 2 ETH.
3. User 2 deposits 1 ETH, and still gets 1e18 shares (due to directly assigning shares to assets). However, due to the SY exchange rate change, only 0.5e18 SY tokens are obtained. At this point, total mints is equal to 1.5e18, and total shares is equal to 1.5e18 as well.
4. User 1 withdraws. User 1 should get 2 ETH, given that the rate increased prior to user 2 depositing. However, `mints` is computed by `shares * _totalMints()) / _totalSupply();`, which is `1e18` * `1.5e18` / 2e18, so mints is finally 0,75e18 (so 0,75 SY tokens will be burnt, instead of 1 full SY token). Essentially, the yield is not distributed pro-rata considering the time where deposits where made, leading to a loss of funds for some depositors.

Recommendation: It is recommended to use the regular ERC4626 conversion functions, and obtain the amount of assets and shares that should be minted/burnt in each flow by

leveraging ERC4626's preview functions.

Amplified: Fixed with the following commit.

Zenith: Verified.

[C-2] Conversion between shares and assets is bricked, leading to loss of funds

Severity: Critical	Status: Resolved

Context:

- StrategyVault.sol#L243

Description: Conversions between shares and assets are used inconsistently. As per the current implementation, `assets` is always equal to `shares`, when depositing, and they are used interchangeably. A good example to see how this might impact the vault is by comparing the `deposit()` and withdraw() implementations:

- In `deposit()`, the amount of assets input by the user is directly set as the total shares that will be minted. This will lead to issues, given that the conversion to shares should account for the ratio of assets and shares that the vault is currently holding (this is typically done by calling previewDeposit()).
- In `withdraw()`, the approach is different, as the user's supplied `assets_` are actually converted to shares via `previewWithdraw()`, so the conversion between shares and assets is actually carried out. Another good example of an inconsistency is `maxWithdraw()`:

```
// File: StrategyVault.sol
function maxWithdraw(
        address owner_
    ) public view returns (uint256 maxAssets) {
        maxAssets = previewWithdraw(maxRedeem(owner_));
    }
```

As shown, `previewWithdraw()` is passed the amount of shares obtained from `maxRedeem()` as parameter. However, in `withdraw()`, the `previewWithdraw()` function is passed `assets_` instead of actual vault shares:

```
// File: StrategyVault.sol
function withdraw(
        uint256 assets_,
        address receiver_,
        address owner_
    ) external nonReentrant returns (uint256 shares) {
        ...

        shares = previewWithdraw(assets_);
```

```
            _redeem(msg.sender, receiver_, owner_, shares);
        }
```

This makes to the overall accounting in the vault be broken, and will lead to stuck funds and missed rewards from the yield generated by the strategies connected to the vault.

Recommendation: It is recommended to ensure that the usual ERC4626 conversion computations are used consistently in all strategy interactions, leveraging ERC4626's preview methods to address issues when converting from assets to shares, and viceversa.

Amplified: Fixed in commit e58f4e7f. **Zenith:** Verified. The most important flaws have been correctly addressed. Conversion between assets and shares is now following conventional logic with the correct rounding applied. `_totalAssets()` now returns a value obtained by querying the strategy's balance and converting it to assets.

4.2 High Risk

A total of 3 high risk findings were identified.

[H-1] Premature Share Calculation in Deposit Function Causes Incorrect Minting for Fee-on-Transfer Tokens

Severity: High	Status: Acknowledged

Context:

- strategy/StrategyVault.sol#L247

Description: The `deposit` function in the StrategyVault contract calculates shares before the actual transfer of assets occurs. This can lead to an undervaluation of shares when dealing with fee-on-transfer tokens.

The vulnerability comes from the following code sequence in the `deposit` :

```
shares = assets;
address caller = msg.sender;
asset_.safeTransferFrom(caller, address(this), assets);
_mint(caller, receiver, shares);
```

Shares is set equal to assets before the transfer occurs. For standard ERC20 tokens, this isn't problematic. However, for fee-on-transfer tokens, the actual amount received by the contract will be less than the amount sent by the user.

A scenario:

- A user wants to deposit 1000 tokens of a fee-on-transfer token with a 2% transfer fee.
- The user calls deposit(1000, userAddress).
- The function sets shares = 1000.
- The safeTransferFrom is called, transferring 1000 tokens from the user.
- Due to the 2% fee, the contract actually receives 980 tokens.
- The function mints 1000 shares to the user.

Result: The user has deposited 1000 tokens but only 980 tokens have entered the vault. However, they've received 1000 shares, which now represent 980 tokens. This discrepancy leads to two significant issues:

1. The user who deposited has more shares than they should, diluting other users' holdings.

2. The total supply of shares no longer accurately represents the total assets in the vault, potentially causing accounting issues and unfair value distribution in future operations.

Recommendation:

track balance before and after the actual transfer, the diff should be the shares minted to the user,

Note: this issue looks similar to #30 However, the root cause, fix is different.

Amplified: The protocol don't use fee-on-transfer tokens - won't fix.

Zenith: Acknowledged.

[H-2] Incorrect Accounting for Fee-on-Transfer Tokens Leads to Share Overvaluation and Potential Vault Insolvency

Severity: High	Status: Acknowledged

Context:

- strategy/StrategyVault.sol:L247
- strategy/base/StrategyManager.sol:L258

Description: The StrategyVault does not properly account for fee-on-transfer tokens when depositing assets into strategies. This creates a discrepancy between the amount of shares minted and the actual value backing those shares, potentially leading to overvaluation of shares and insolvency of the vault.

In the deposit function of StrategyVault.sol, we see:

```
uint256 balanceBefore = asset_.balanceOf(address(this));
asset_.safeTransferFrom(caller, address(this), assets);
assets = asset_.balanceOf(address(this)) - balanceBefore;
```

This correctly accounts for any fee-on-transfer by calculating the actual received amount. This is User to StrategyVault transfer. However, for StrategyVault to Strategies transfer, it does not.

In the `_mint` function, the same 'shares' value is used both for depositing into strategies and for minting shares to users:

```
function _mint(address caller, address receiver, uint256 shares) internal
{
    _deposit_strategies(shares);
    _mint(receiver, shares);
    emit Deposit(caller, receiver, shares, shares);
}
```

For fee-on-transfer tokens, the actual amount deposited into strategies will be less than 'shares' due to the fee deduction. However, the full 'shares' amount is still minted to the user.

The `_deposit_strategies` and `_deposit_strategy` functions also do not account for potential fees:

```
function _deposit_strategies(uint256 assets_) internal {
    // ... (strategy enumeration code)
    _deposit_strategy(strategy, (weight * assets_) / TOTAL_WEIGHT);
}

function _deposit_strategy(address strategy_, uint256 assets_) internal {
    // ...
    IERC20(asset_).forceApprove(strategy_, assets_);
    if (IERC4626(strategy_).deposit(assets_, address(this)) == 0) {
        revert Errors.BadMint(strategy_, 1, 0);
    }
}
```

These functions use the full `assets_` amount without adjusting for fees, leading to inconsistent strategy deposits and incorrect accounting within the vault.

This issue can result in:

- Over-minting of shares relative to the actual deposited value
- Inconsistent strategy deposits
- Cumulative discrepancies between vault accounting and actual assets under management
- Risk of vault insolvency over time

Recommendation: Update the `_mint` function to use the actual deposited amount (post-fee) for minting shares, rather than the pre-fee amount.

Amplified: The protocol don't use fee-on-transfer tokens - won't fix

Zenith: Acknowledged

[H-3] Incorrect parameter in `TokenAmount` struct might Dos Pendle interactions

Severity: High	Status: Resolved

Context:

- PendleSYTokenConvertor.sol#L94

Description: Passing amount as the minimum output amount to expect after redeeming from Pendle might lead to a complete DoS of ERC4626 withdrawals. The `TokenOutput`'s `minTokenOut` parameter allows users to specify the minimal amount that the user finally receives. `makeWithdrawAdaptorCallData()` hardcodes it to the same `amount` that is requested to be withdrawn, which means that the slippage tolerance for Pendle redemptions is always of 0%. Although this can work for some strategies that don't perform any internal swap in Pendle, other SY tokens might actually perform some swaps that might incur some slippage (see `PendleConvexLPSY`, which removes liquidity from curve in some situations, potentially leading to slippage). If some slippage is incurred in the interaction with Pendle, the transaction will revert, making it impossible for users to withdraw their assets.

Recommendation: It is recommended to apply a slippage tolerance when filling the `TokenOutput` struct. Given that slippage is already checked later in the withdrawal flow, the parameter could be set to 0 to avoid additional computations.

Amplified: Fixed with the following commit.

Zenith: Verified.

4.3 Medium Risk

A total of 5 medium risk findings were identified.

[M-1] Risk of Unexpected Reverts Due to Self-Transfer in AaveATokenAdaptor Withdraw Function

Severity: Medium	Status: Resolved

Context:

- adaptor/external/AaveATokenAdaptor.sol:L205

Description: The `withdraw` function in the `AaveATokenAdaptor` contract contains an unnecessary token transfer operation. After withdrawing assets from Aave, the function attempts to transfer these assets to the `receiver`, which is always `address(this)` due to the `_externalReceiverCheck` function. This self-transfer is redundant and potentially problematic since some ERC20 tokens may prohibit self-transfers, which could cause the `withdraw` function to revert unexpectedly.

```
IERC20(token.UNDERLYING_ASSET_ADDRESS()).safeTransfer(receiver, assets);
```

Where receiver is always address(this).

```
function _externalReceiverCheck(address receiver) internal view {
        if (receiver != address(this))
            revert BaseAdaptor__ExternalReceiverBlocked();
    }
```

Recommendation: Remove the unnecessary token transfer operation from the withdraw function.

Amplified: Resolved with the following commit

Zenith: Verified.

[M-2] Lack of slippage protection when redeeming SY tokens from Pendle

Severity: Medium	Status: Resolved

Context:

- adaptor/external/PendleAdaptor.sol:L150

Description: The PendleAdaptor lacks slippage protection when redeeming SY tokens for underlying tokens using the Pendle Router. The `redeemSyToToken` function in the PendleAdaptor contract calls the Pendle Router's `redeemSyToToken` method without specifying a minimum output amount. This oversight could potentially lead to significant losses for users due to unexpected price movements or malicious actors front-running transactions.

The Pendle Router's implementation uses 0 as the `minTokenOut` parameter:

```
netTokenRedeemed = IStandardizedYield(SY).redeem(receiver, netSyIn,
out.tokenRedeemSy, 0, true);
```

https://github.com/pendle-finance/pendle-core-v2-public/blob/main/contracts/router/base/ActionBase.sol#L120

While the StandardizedYield contract does provide the option to specify a minTokenOut value, this protection is not being utilized.

```
function redeem(
    address receiver,
    uint256 amountSharesToRedeem,
    address tokenOut,
    uint256 minTokenOut,
    bool burnFromInternalBalance
) external nonReentrant returns (uint256 amountTokenOut) {
```

https://github.com/pendle-finance/pendle-core-v2-public/blob/main/contracts/core/StandardizedYield/SYBase.sol#L63

However, you can still use the router and do the slippage protection here. For example,

```
    tokenReceived = router.redeemSyToToken(address(this), sy, netSyIn,
output);
```

```
        require(tokenReceived >= minTokenOut, "Slippage tolerance exceeded");
```

Recommendation: I is recommended to implement slippage protection in the PendleAdaptor contract. This can be achieved by modifying the redeemSyToToken function to include a minTokenOut parameter and adding a check to ensure the received tokens meet this minimum threshold.

Amplified: Fixed with the following commit

Zenith: Verified

[M-3] Inadequate Safeguards in Adaptor Removal Process Risking Asset Lock-up

Severity: Medium	Status: Resolved

Context:

- adaptor/AdaptorConfigurator.sol:L109

Description: The AdaptorConfigurator contract allows for the removal and re-addition of adaptors without adequate safeguards for active positions or assets. The `removeAdaptor` function, which internally calls `_Proxy_remove`, deletes the adaptor's entry from the `_proxies` mapping without checking for any existing assets or active positions associated with the adaptor.

```
function removeAdaptor(string memory name_) external override onlyOwner {
        address proxy_ = _getProxy(name_);
        _ERC20_Proxy_remove(proxy_);
        _Adaptor_Proxy_remove(proxy_);
        _Proxy_remove(name_);
        emit AdaptorRemoved(name_, proxy_);
}
```

- adaptor/AdaptorConfigurator.sol:L109

While it is possible to re-add an adaptor with the same name and address after removal, this process creates a new Proxy and does not restore any previous state or assets. This implementation poses a risk of funds becoming inaccessible if an adaptor is removed while still holding user assets or having open positions. The current code lacks checks for active positions or remaining assets prior to adaptor removal. Consequently, this design could lead to the potential loss of user funds or assets, as the removal of adaptors is irreversible.

Recommendation:

- Implement a check in the `removeAdaptor` function to verify that the adaptor has no active positions or remaining assets before allowing removal.
- Introduce a two-step removal process: a. First, deactivate the adaptor to prevent new deposits. b. Allow full removal only after all positions are closed and assets are withdrawn.
- Add events to notify users when an adaptor is scheduled for removal or deactivation.

Amplified: Fixed with the following commit

Zenith: Verified

[M-4] DoS in functionality for ETH deposits by directly transferring to the contract

Severity: Medium	Status: Resolved

Context:

- Proxy.sol#L17

Description: Logic for direct ETH deposits won't work due to `EIP173Proxy` reverting on `receive()`.

`StrategyVault` contracts are deployed via `StrategyVaultFactory`'s `createStrategyVault()`. As shown in the following snippet, all strategy vaults will use a proxy pattern where the proxy is an `EIP173Proxy`:

```
// File: StrategyVaultFactory.sol
function createStrategyVault(
        address owner_,
        address asset_,
        string calldata name_,
        string calldata symbol_,
        IStrategyManager.Strategy[] memory strategies_
    ) external onlyOwner returns (address newVault) {
        // Create a new EIP173Proxy instance
        newVault = address(
            new EIP173Proxy(strategyVaultImplementation, address(this),
""")
        );

        // Cast the proxy to StrategyVault and initialize it
        IVaultInitializer(newVault).initialize(
            owner_,
            asset_,
            name_,
            symbol_,
            strategies_
        );
        ...
}
```

However, `EIP173Proxy` inherits from a custom implementation of an EIP-1967 proxy (`Proxy.sol`), which rejects all transactions that receive ETH:

```
// File: Proxy.sol

receive() external payable virtual {
        revert("ETHER_REJECTED"); // explicit reject by default
}
```

Because of this, the `receive()` logic in `StrategyVault` won't work, as the proxy will directly revert transactions directly transferring ETH to the vault, making it impossible to deposit into strategies by directly transferring ETH to the contract.

Recommendation: It is recommended to remove the revert in `Proxy.sol`. This will allow the Strategy `receive()` logic to work, making it possible to deposit ETH by directly transferring it to the contract.

```
// File: Proxy.sol

receive() external payable virtual {
-        revert("ETHER_REJECTED"); // explicit reject by default
}
```

Amplified: Fixed in commit f05ba693. **Zenith:** Verified. The `receive()` function has been updated. Now, instead of reverting, it will transfer funds to the creator address.

[M-5] Missing pausability checks in deposit and minting flows

Severity: Medium	Status: Resolved

Context:

- StrategyVault.sol#L231
- StrategyVault.sol#L302

Description: `deposit()` and `mint()` are missing the `_paused()` check. This allows users to still deposit into the vault even if it has been paused, making pausability in the vault ineffective.]receive()` should also prevent direct ETH deposits in case the vault is paused and the underlying asset is WETH.

In addition, the `withdraw()` and `redeem()` functions actually check for pausability. This could lead to users being unable to withdraw their funds in case the vault has been paused due to an unexpected event, which is incorrect.

Recommendation: It is recommended to add pausability checks in the `deposit()` and `mint()` functions. This will ensure that user's can't deposit assets into the vault in an emergency pausing situation. It is also recommended to remove pausability checks in `withdraw()` and `redeem()`, so that users can withdraw funds from the protocol in case there is an emergency and the contract is paused.

Amplified: Fixed in commit e58f4e7f. **Zenith:** Verified. A check to `_paused()` has been added in the deposit and mint flows, ensuring such functionalities are paused if the pause is active.

4.4 Low Risk

A total of 6 low risk findings were identified.

[L-1] Redundant Zero-Check Assertion in PendleAdaptor's mintSyFromToken

Severity: Low	Status: Resolved

Context:

- adaptor/external/PendleAdaptor.sol:L130

Description: The `mintSyFromToken` function in the PendleAdaptor contains an unnecessary assertion that checks if the output of `router.mintSyFromToken` is greater than zero because the Pendle router already enforces the `minSyOut` parameter, which would cause a revert if the output is less than the specified minimum.

- StandardizedYield/SYBase.sol#L50

Recommendation: Remove the assertion `assert(router.mintSyFromToken(address(this), sy, minSyOut, input) > 0);`. Add a check at the beginning of the function to revert if `minSyOut` is zero.

For example:

```
function mintSyFromToken(
    IPendleMarket market,
    uint256 minSyOut,
    TokenInput memory input
) external {
    if (minSyOut == 0) {
        revert ZeroMinSyOut();
    }

    _verifyMarket(market);
    _verifyDexAggregatorInputIsNotUsed(input);
    (address sy, , ) = market.readTokens();

    IERC20 inputToken = IERC20(input.tokenIn);
    input.netTokenIn = _maxAvailable(inputToken, input.netTokenIn);
    inputToken.forceApprove(address(router), input.netTokenIn);
    router.mintSyFromToken(address(this), sy, minSyOut, input);
    _revokeExternalApproval(inputToken, address(router));
}
```

Amplified: Fixed with the following commit

Zenith: Verified

[L-2] Unbounded loops in StrategyManager

Severity: Low	Status: Resolved

Context:

- strategy/base/StrategyManager.sol:L258
- strategy/base/StrategyManager.sol:L272
- strategy/base/StrategyManager.sol:L147

Description: In the StrategyManager, there are three functions that contain loops iterating over all strategies: `_deposit_strategies`, `_redeem_strategies`, and `_rebalanceStrategy`. These functions share similar loop structures and potential issues:

- Each function iterates over all strategies in a single transaction, regardless of the number of strategies. The loop structure is as follows:

```
Strategy[] storage strategies = _getStrategies();
uint256 len = strategies.length;
for (uint256 i = 0; i < len; i++) {
    Strategy storage strategy = strategies[i];
    // Perform operations on each strategy
}
```

- There's no upper bound on the number of strategies, which means as the number of strategies grows, these functions could potentially hit the block gas limit, causing transactions to fail.

Recommendation:

Consider adding a maximum number of strategies that StrategyManager can have. This would ensure that the functions don't exceed gas limits.

Amplified: Fix applied

Zenith: Verified.

[L-3] Lack of Interface Validation in AdaptorConfigurator's addAdaptor Function

Severity: Low	Status: Acknowledged

Context:

- adaptor/AdaptorConfigurator.sol:L88

Description: The `addAdaptor` function in the AdaptorConfigurator currently only performs a basic check to ensure the `cellarAdaptor_` address is non-zero. This implementation lacks a robust verification mechanism to confirm that the provided address actually implements the expected adaptor interface. Such a minimal check could potentially allow the addition of invalid or malicious adaptors to the system, which may lead to unexpected behavior or security vulnerabilities.

Recommendation:

- Define an interface (e.g., `IBaseAdaptor`) that all valid adaptors must implement.
- Modify the `addAdaptor` function to attempt to call a function from the `IBaseAdaptor` interface (e.g., `identifier()`) on the provided adaptor address. If the call succeeds, proceed with adding the adaptor. If it fails, revert the transaction with an appropriate error message.

Example :

```
import "@openzeppelin/contracts/utils/Address.sol";

interface IBaseAdaptor {
    function identifier() external view returns (bytes32);
}

function addAdaptor(
    string memory name_,
    string memory symbol_,
    address cellarAdaptor_,
    AdaptorData memory adaptorData_
) external override onlyOwner {
    if (cellarAdaptor_ == address(0)) {
        revert Errors.ZeroAddress();
    }

    require(
        Address.isContract(cellarAdaptor_),
        "AdaptorConfigurator: Adaptor must be a contract"
```

```
        );

        try IBaseAdaptor(cellarAdaptor_).identifier() returns (bytes32) {
            // Adaptor likely implements the interface, proceed with addition

    } catch {
            revert("AdaptorConfigurator: Invalid adaptor interface");
        }
    }
```

Amplified: A governance (owner) is responsible only for adding an adaptor, so he should have checked it before. If added by mistake then the adaptor could be removed - Won't fix.

Zenith: Acknowledged

[L-4] Incorrect number of topics emitted in ERC4626's `Deposit()` event

Severity: Low	Status: Resolved

Context:

- ERC4626.sol#L414
- ProxyStorageBase.sol#L91

Description: When depositing into the vault, the `emitViaProxy_Deposit()` function is used to emit EIP-4626's Deposit event.

From ERC4626's standard, we can see that only the sender and owner parameters are indexed. This means that emitting such event using assembly should be done via the `log3` opcode, given that the `Deposit()` event has three topics (the event signature, the `sender` and the `owner`).

However, `emitViaProxy_Deposit()` uses the `log4`opcode instead:

```
// File: Proxy.sol
function emitViaProxy_Deposit(
        address proxyAddr,
        address sender,
        address owner,
        uint256 assets,
        uint256 shares
    ) internal FREEMEM {
        // slither-disable-next-line low-level-calls
        (bool success, ) = proxyAddr.call(
            abi.encodePacked(
                uint8(4),

keccak256(bytes("Deposit(address,address,uint256,uint256)")),
                bytes32(uint(uint160(sender))),
                bytes32(uint(uint160(owner))),
                assets,
                shares
            )
        );
        require(success, "log-proxy-fail");
    }
```

This will make the `Deposit()` event from EIP-4626's standard be incorrectly emitted, as there will be an additional indexed parameter, effectively breaking the expected standard behavior.

Recommendation: In order to comply with ERC4626's standard, use the `log3` opcode to emit the Deposit event, instead of `log4`:

```
// File: Proxy.sol
function emitViaProxy_Deposit(
        address proxyAddr,
        address sender,
        address owner,
        uint256 assets,
        uint256 shares
    ) internal FREEMEM {
        // slither-disable-next-line low-level-calls
        (bool success, ) = proxyAddr.call(
            abi.encodePacked(
-               uint8(4),
+               uint8(3),

keccak256(bytes("Deposit(address,address,uint256,uint256)")),
                bytes32(uint(uint160(sender))),
                bytes32(uint(uint160(owner))),
-               assets,
-               shares
+               abi.encode(assets, shares)
            )
        );
        require(success, "log-proxy-fail");
    }
```

Amplified: Fixed in [commit f05ba693](#) **Zenith:** Verified. A 3 is now passed instead of a 4 to indicate that log3 must be used, and assets and shares are encoded.

[L-5] Current implementation is incompatible with fee-on-transfer tokens

Severity: Low	Status: Acknowledged

Context:

- ERC4626.sol#L359
- Proxy.sol#L37

Description: The current logic of transferring tokens performs a transfer without checking balances:

```
// File: ERC4626.sol
function _deposit(
        uint256 assets,
        address receiver
    ) internal nonReentrant returns (uint256 shares) {
       ...
        if (
            !viaProxy_TransferFrom(
                proxy,
                $._asset,
                sender,
                address(this),
                assets
            )
        ) {
            revert Errors.TokenBadTransfer(
                $._asset,
                sender,
                address(this),
                assets
            );
        }
        ...
}
```

```
// File: Proxy.sol
function proxy_TransferFrom(
        address token,
        address from,
        address to,
        uint256 value
```

```
    ) external override returns (bool) {
        if (msg.sender != creator) {
            revert Errors.AccessDenied(msg.sender);
        }
        // slither-disable-next-line arbitrary-send-erc20
        return IERC20(token).transferFrom(from, to, value);
    }
```

This will cause a DoS with fee-on-transfer tokens, as the contract's balance will be less than the actual transferred tokens due to the applied fee.

Recommendation: It is recommended to perform balance checks after transferring assets. As an example, for deposit:

```
// File: ERC4626.sol

function _deposit(
        uint256 assets,
        address receiver
    ) internal nonReentrant returns (uint256 shares) {
        StrategyAdaptorStorage storage $ = _getStrategyAdaptorStorage();
        (address sender, address proxy) = unpackTrailingParams();
+       uint256 balanceBeforeTransfer =
IERC20(assetOut).balanceOf(address(this));
        if (
            !viaProxy_TransferFrom(
                proxy,
                $._asset,
                sender,
                address(this),
                assets
            )
        ) {
            revert Errors.TokenBadTransfer(
                $._asset,
                sender,
                address(this),
                assets
            );
        }
+      assets = IERC20(assetOut).balanceOf(address(this)) -
balanceBeforeTransfer;
}
```

Amplified: The protocol doesn't support fee-on-transfer tokens - won't fix.

[L-6] Potential DoS for pools with low observation cardinality

Severity: Low	Status: Acknowledged

Context:

- Quoter.sol#L161

Description: The quoter contract will use the corresponding Uniswap V3 pools' `observe()` function to obtain a price using the TWAP, by calling `OracleLibrary`'s `consult()`:

```
// File: Quoter.sol
function _getUniswapV3Price(
        address token
    ) private view returns (uint256 price) {
        ...
        // slither-disable-next-line unused-return
        (int24 arithmeticMeanTick, ) = OracleLibrary.consult(
            config.uniswapV3pool,
            config.twapPeriod
        );
        price = OracleLibrary.getQuoteAtTick(
            arithmeticMeanTick,
            1e18,
            token,
            _referenceAsset
        );
    }
```

However, the call could fail for certain pools if the observation cardinality is lower than the expected TWAP window to be queried. The pool should have enough recorded price observations to cover at least the entire TWAP (Time-Weighted Average Price) window. This would lead to users being unable to interact with the protocol, given that calls to `observe()` would revert, or to the Amplified team not being able to fetch prices for the desired TWAP window if the observation cardinality for the pool is not enough.

Recommendation: It is recommended to in crease the observation cardinality for the pool by calling its `increaseObservationCardinalityNext()`, method, ensuring that the new cardinality is enough to cover the TWAP window.

Amplified: Acknowledged.